Ulrich Flegel, Michael Meier (Eds.)

Detection of Intrusions and Malware & Vulnerability Assessment

GI Special Interest Group SIDAR Workshop, DIMVA 2004
Dortmund, Germany, July 6-7, 2004
Proceedings

Gesellschaft für Informatik 2004

Lecture Notes in Informatics (LNI) - Proceedings
Series of the Gesellschaft für Informatik (GI)

Volume P-46

ISBN 3-88579-375-X
ISSN 1617-5468

Volume Editors
Ulrich Flegel
University of Dortmund,
Computer Science Department, Chair VI, ISSI
D-44221 Dortmund, Germany
ulrich.flegel@udo.edu
Michael Meier
Brandenburg University of Technology Cottbus,
Computer Science Department, Chair Computer Networks
P.O. Box 10 13 44, D-03013 Cottbus, Germany
mm@informatik.tu-cottbus.de

Series Editorial Board
Heinrich C. Mayr, Universität Klagenfurt, Austria (Chairman, mayr@ifit.uni-klu.ac.at)
Jörg Becker, Universität Münster, Germany
Ulrich Furbach, Universität Koblenz, Germany
Axel Lehmann, Universität der Bundeswehr München, Germany
Peter Liggesmeyer, Universität Potsdam, Germany
Ernst W. Mayr, Technische Universität München, Germany
Heinrich Müller, Universität Dortmund, Germany
Heinrich Reinermann, Hochschule für Verwaltungswissenschaften Speyer, Germany
Karl-Heinz Rödiger, Universität Bremen, Germany
Sigrid Schubert, Universität Siegen, Germany

Dissertations
Dorothea Wagner, Universität Karlsruhe, Germany

Seminars
Reinhard Wilhelm, Universität des Saarlandes, Germany

© Gesellschaft für Informatik, Bonn 2004
printed by Köllen Druck+Verlag GmbH, Bonn

Vorwort

Mit der wachsenden Abhängigkeit unserer Gesellschaft von der Zuverlässigkeit informationstechnischer Systeme (IT) gewinnen Fragen der IT-Sicherheit an Bedeutung. Während bisher vorrangig präventive Maßnahmen und Mechanismen im Vordergrund standen, wird zunehmend deutlich, dass IT-Sicherheit nicht allein durch Prävention erreichbar ist. Vielmehr stellt Prävention einen Grundpfeiler dar, neben dem ergänzend die reaktiven Aspekte der IT-Sicherheit stehen.

Die Fachgruppe SIDAR (Security - Intrusion Detection and Response) des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. fokussiert die reaktiven Aspekte der IT-Sicherheit sowie deren Umfeld. Mit dem Workshop DIMVA 2004 veranstaltet die FG SIDAR erstmals im deutschsprachigen Raum einen Workshop, der die Themen Intrusion Detection, Malware-Bekämpfung und Verwundbarkeitsanalyse in den Mittelpunkt stellt.

Der Workshop richtet sich an Personen und Organisationen, die in diesen Themenbereichen in Industrie, Dienstleistung, Verwaltung und Wissenschaft tätig sind. Ziel des Workshops ist es, sowohl den kommunikativen Austausch zu fördern, als auch reaktive Aspekte der IT-Sicherheit stärker in das Blickfeld der Öffentlichkeit zu rücken.

Dieser Band enthält ausgewählte Beiträge, die von den Autoren auf dem Workshop am 6. und 7. Juli in Dortmund (Deutschland) präsentiert wurden. Das Programmkomitee erhielt 41 Beiträge von Autoren aus zwölf Ländern und drei Kontinenten. 78% der Beiträge stammen von Autoren aus Europa. Die meisten Beiträge wurden von Autoren aus den folgenden Ländern eingereicht: Deutschland (26), USA (4), Vereinigte Arabische Emirate (3), Iran (2) und Schweiz (2). Jeder Beitrag wurde sorgfältig von mindestens drei Mitgliedern des Programmkomitees oder zusätzlichen Experten begutachtet und nach den folgenden Kriterien bewertet: wissenschaftliche Neuheit, Bedeutung für das Gebiet und technische Qualität.

Das Programmkomitee hat insgesamt 19 Beiträge zur Präsentation auf dem Workshop ausgewählt (46%). Davon wurden 14 Beiträge zur Veröffentlichung in diesem Band akzeptiert (34%) und fünf Kurzbeiträge werden auf der Web-Seite des Workshops veröffentlicht (12%). Die akzeptierten Beiträge wurden von den Autoren vor der Veröffentlichung überarbeitet. Die überarbeiteten Beiträge wurden keiner weiteren Begutachtung unterzogen, und die Autoren tragen die Verantwortung für den Inhalt ihrer Beiträge.

Das Workshop-Programm umfasste neue theoretische und praktische Ansätze und Resultate aus der Forschung sowie Erfahrungsberichte zum Schwerpunktthema Intrusion Detection und zu den Themen Honeypots, Verwundbarkeiten und Malware-Bekämpfung. Zum Auftakt des Workshop-Programms hielt Hans-Michael Hepp (Intelligent Risk Solutions) den Keynote-Vortrag "Verfahren der Transaktionsanalyse am Beispiel der Missbrauchsfrüherkennung im Kreditkartengeschäft". Die Präsentationen und die Kurzbeiträge sind auf der Web-Seite des Workshops verfügbar:
http://www.gi-fg-sidar.de/dimva2004/

Wir danken allen, die zum Gelingen des Workshops beigetragen haben, insbesondere den Autoren, dem eingeladenen Redner, dem Programmkomitee und den Gutachtern.

Juli 2004 Ulrich Flegel, Michael Meier

Preface

Along with the growing dependency of our society on information technology systems (IT), questions regarding IT security are becoming more urgent. While up to now primarily preventive measures and mechanisms were focused, it becomes more and more apparent that IT security cannot be reached by prevention alone. Rather, preventive measures and reactive aspects need to complement one another.

The special interest group SIDAR (Security – Intrusion Detection and Response) of the Security and Safety division of the German Informatics Society (GI) focuses on reactive aspects of IT security and closely related topics. The SIG SIDAR Workshop DIMVA 2004 is the first workshop in the German-speaking area that is dedicated to the topics Intrusion Detection, Malicious Agents (Malware) and Vulnerability Assessment. While targeting the active players working in reactive IT security in industry, service, government and research, the DIMVA workshop is intended to further exchange, advances and public awareness of reactive aspects of IT security.

These proceedings contain selected papers that were presented by the authors at the workshop on July 6-7 in Dortmund, Germany. The program committee received 41 submissions from authors from twelve countries on three continents. 78% of the papers originate from authors from Europe. Most papers were submitted by authors from the following countries: Germany (26), USA (4), United Arab Emirates (3), Iran (2) and Switzerland (2). Each paper has been carefully reviewed by at least three members of the program committee or external experts and has been evaluated according to the criteria of scientific novelty, importance to the field, and technical quality.

The program committee selected a total of 19 papers for presentation at the workshop (46%). Thereof 14 papers have been accepted for publication in these proceedings (34%) and five short papers have been published at the website of the workshop. Accepted papers have been revised by the authors before publishing. The revisions have not been checked for correctness, and the authors bear full responsibility for the contents of their papers.

The workshop program comprised new theoretical and practical approaches and results from research as well as experience reports on the principal topic Intrusion Detection and on the topics Honeypots, Vulnerabilities and Malware. Hans-Michael Hepp (Intelligent Risk Solutions) gave an opening keynote on “Techniques for transaction analysis for early misuse detection in credit card business”. The presentations and short papers are available on the website of the workshop: http://www.gi-fg-sidar.de/dimva2004/

We warmly thank all those who have contributed to the success of the workshop, especially the authors and the invited speaker as well as the members of the program committee and the external reviewers.

July 2004 Ulrich Flegel, Michael Meier

DIMVA 2004

July 6-7, 2004, Dortmund, Germany

Program and General Chairs

Ulrich Flegel	University of Dortmund, Germany
Michael Meier	Brandenburg University of Technology Cottbus, Germany

Program Committee

Thomas Biege	SuSE Linux, Germany
Roland Büschkes	T-Mobile, Germany
Toralv Dirro	Network Associates, Germany
Anja Feldmann	Technical University Munich, Germany
Ulrich Flegel	University of Dortmund, Germany
Christian Freckmann	TÜV-IT, Germany
Oliver Göbel	RUS-CERT, Germany
Christian Götz	Cirosec, Germany
Dirk Häger	BSI, Germany
Marc Heuse	n.runs, Germany
Klaus Julisch	IBM Research Zurich, Switzerland
Oliver Karow	Symantec, Germany
Klaus-Peter Kossakowski	Presecure, Germany
Hartmut König	BTU Cottbus, Germany
Heiko Krumm	University of Dortmund, Germany
Christopher Krügel	UC Santa Barbara, USA
Holger Mack	Secorvo, Germany
Michael Meier	BTU Cottbus, Germany
Jens Nedon	Consecur, Germany
Christian Schmid	Linz, Austria
Morton Swimmer	IBM Research Zurich, Switzerland
Stefan Strobel	Cirosec, Germany
Marco Thorbrügge	DFN-CERT, Germany
Andreas Wespi	IBM Research Zurich, Switzerland
Stephen Wolthusen	Fraunhofer IGD Darmstadt, Germany
Ralf Zessin	Maxpert, Germany

External Reviewers

Friedemann Bauer	RUS-CERT, Germany
Holger Dreger	Technical University Munich, Germany
Peter Herrmann	University of Dortmund, Germany
Birk Richter	secunet, Germany
Mario Schölzel	BTU Cottbus, Germany
Robin Sommer	Technical University Munich, Germany

Cooperations

IEEE Task Force on
Information Assurance

German Chapter of the ACM

University of Dortmund

Support

dortmund-project
www.dortmund-project.de

Horst Görtz Institute
for Security in Information Technology
www.hgi.ruhr-uni-bochum.de

Contents

Intrusion Detection

Honeypots

Vulnerabilities

Malware

Alarm Reduction and Correlation in Intrusion Detection Systems

Tobias Chyssler[1], Stefan Burschka[2], Michael Semling[2], Tomas Lingvall[2], Kalle Burbeck[1]

[1]Dept. of Computer and Information Science
Linköping University,

S-581 83 Linköping ,Sweden
[tobch,kalbu]@ida.liu.se

[2]Swisscom Innovations
Service & Security Management,

Ostermundigenstrasse 93

3050 Bern,Switzerland
[Stefan.Burschka,Michael.Semling,
Tomas.Lingvall]@Swisscom.com

Abstract: Large Critical Complex Infrastructures are increasingly dependent on IP networks. Reliability by redundancy and tolerance are an imperative for such dependable networks. In order to achieve the desired reliability, the detection of faults, misuse, and attacks is essential. This can be achieved by applying methods of intrusion detection. However, in large systems, these methods produce an uncontrollable vast amount of data which overwhelms human operators. This paper studies the role of alarm reduction and correlation in existing networks for building more intelligent safeguards that support and complement the decisions by the operator. We present an architecture that incorporates Intrusion Detection Systems as sensors, and provides quantitatively and qualitatively improved alarms to the human operator. Alarm reduction via static and adaptive filtering, aggregation, and correlation is demonstrated using realistic data from sensors such as Snort, Samhain, and Syslog.

1 Introduction

The economy and security of Europe is increasingly dependent on a range of Large Complex Critical Infrastructures (LCCI) such as electricity and telecommunication networks. Protecting these infrastructures requires an understanding of the vulnerabilities that exist in every layer of the network; from the physical layer up to the network and service layers as well as the organisational layer that supports the complex operation of these networks. This paper focuses on the support of operators analyzing messages created by various sources, in particular Network Intrusion Detection Systems (NIDS), Host Intrusion Detection Systems (HIDS) and general log data. The amount of data being produced by such Intrusion Detection Systems (IDS) exceeds by far human

capability of information processing. One study [YBU03] estimates that the number of intrusion attempts over the entire Internet is in the order of 25 billion each day and increasing. McHugh [Hug01] claims that attacks are getting more and more sophisticated while they get more automated. This requires detailed analysis in-depth leaving no time for adequate reactions. Most INFOrmation SECurity systems (INFOSEC) based on IDS technology attempt to respond to the operator's information overload.

INFOSEC systems either apply knowledge-based techniques (typically realised as signature-based misuse detection), or behaviour-based techniques (e.g. by applying machine learning for detection of anomalies). On the other hand, there are tools performing methods of data mining (e.g. on log results), or by simply collecting and grouping alerts for further examination by a human operator [MCZ+00]. However, these INFOSEC systems generate far too many alarms. While post mortem studies are possible on large data sets, the ability of reacting in real-time to intrusions is highly dependent on improved quality of the alarms, and in particular reduction of the false alarm rates.

When studying the range of problems in dealing with security issues in the management network of telecom service providers, the following needs are identified: (1) Reduction of the message counts such that the operator can cope with them. (2) A lower rate of false alarms. (3) Information collection and correlation from various sources to identify indices of attacks. (E.g. The combining of information about the network topology with IDS alarms). (4) Indication of general network "health" with predictive elements so that total service collapse is avoided. The work in this paper addresses the first two issues and to some extent the third issue which is a prerequisite for future sophisticated analysis of alarm data. The work is carried out in the context of the European Safeguard project [Saf03] aiming to demonstrate the use of distributed and coordinated software agents for enhancing existing defence mechanisms in telecom and electricity management networks. Safeguard agents at higher levels use data that is processed as presented in this paper. Thus, this paper describes methods how to improve the quality of IDS data, thus enabling a human operator to fulfil an in-depth analysis within acceptable time.

The structure of the paper is as follows. First, we present a safeguard architecture that has emerged during the work in the above project. Next, the improvement of the information quality of knowledge-based and behaviour-based approaches is discussed. The knowledge-based approach filters and aggregates alarms. Moreover, methods of automated text classifications utilizing naïve Bayesian networks are used to adapt the IDS filtering. Behaviour-based methods correlate the aggregated alarms. The three considered techniques are: An additive correlator, a Neuronal Network (NN) classifier, and a classifier based on K-nearest-neighbours.

We evaluated our methods on data generated in a test network of appropriate size; because the datasets used for a number of recent evaluations of alarm correlation approaches [HA03] were not available to the wide research community. Our test network consists of 50 machines, and has been set up at Swisscom as part of the Safeguard research effort, especially for evaluating various approaches to recognition,

reaction and recovery mechanisms. The data produced for these experiments can be shared and tested by other researchers. Finally we comment on the conclusions so far.

1.1. Related works

Several proposals for alarm correlation (e.g. [CM02], [DW01], [MD03], [NCR02] and [PFV02]) are limited to predefined rules for attack scenarios and countermeasures. Another proposal to find new scenarios is given by Qin et al. [QL03]. A different approach is the probabilistic alert correlation taken by Valdes et al. [VS01], where a mathematical framework is used to collate alerts based on their similarity. Our mechanism is based on text based distance metrics combined with Neural Networks and K-nearest neighbours' algorithms.

The idea to perform data mining in order to reduce for false alarm has been explored by Julisch et al. [JD02]; using conceptual clustering of old alarms to derive new filters. Our method for adaptive filtering has the same objective, but we use automatic text classification instead.

2 The Safeguard context

The Safeguard architecture is outlined in Figure 1.

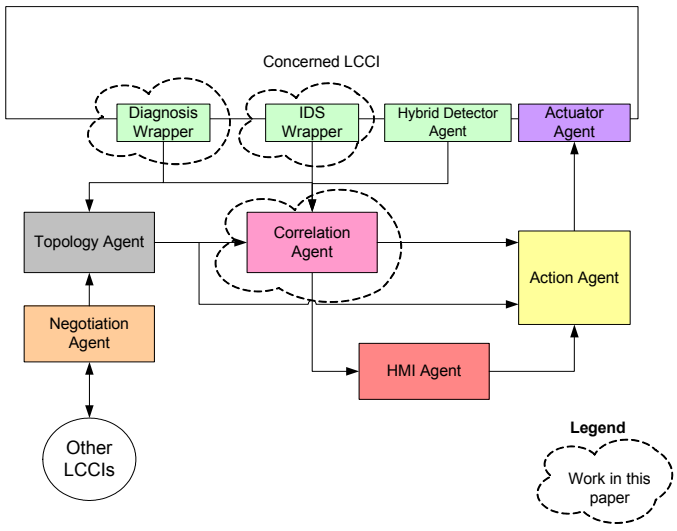


Figure 1: The Safeguard agent architecture

The roles of agents can be described as follows:

- **Wrapper agent**: Wraps standard INFOSEC devices and existing LCCI diagnosis mechanisms, and applies static filtering and normalisation on their outputs.

- **Topology agent:** Gather network topology information dynamically, e.g. host types, operating system types, services provided, known vulnerabilities.

- **Hybrid detector agent:** Uses data mining techniques such as clustering to detect anomalies.

- **Correlation agent:** Filters, aggregates, and correlates (section 3.4) normalized messages from different sources. Special filter lists are applied for network critical services [Bu03]. In practice the agent is split into a filter, an aggregation and a separate correlation and alarm generation part.

- **Action agent:** Initiates automatic and human controlled responses.

- **Negotiation agent:** Communicates with negotiation agents in neighboured LCCIs to pass on information.

- **HMI (Human Machine Interface) agent:** Provide an interface supporting the operator in his analysis. This agent also facilitates the incorporation of adjustable autonomy within the agent-based system [Sc01].

- **Actuator:** Interface for communication with lower layer software and hardware (e.g. changing firewall rules, renicing or killing processes, traffic shaping).

There may be several instances of each agent in each LCCI and only some variants are described in this paper.

2.1 Choice of sensors

As reported by Yin et al. [Yi03] the success of IDSs depends upon the data they process. In order to assess the relevance of an alarm, several sources of information have to be considered. For this reason we use three different INFOSEC devices. One the network level, the NIDS Snort, on host level the HIDS Samhain and various agents providing information about system relevant information as well as application logs via Syslog are applied. Multiple sources of additional information such as network topology information, connection statistics, policies, and vulnerabilities are further included in our selection of sensors. Anomaly detectors for network traffic as described in [BN04].

3 Alarm Reduction

In the following we describe one possible human way of analysing alarms. The expressiveness of alarms is evaluated by its:

- Severity
- Number
- Frequency
- Variety
- Uniqueness
- Payload

As outlined in Figure 2: Analysis procedure , the analysis starts by removing the alarms that are known to be of no interest. Syslog is checked for messages that look suspect. For each message found, the corresponding machine is checked in the HIDS to detect suspicious changes and the NIDS is checked for possible sources for attacks around the time that roughly coincides with the event.

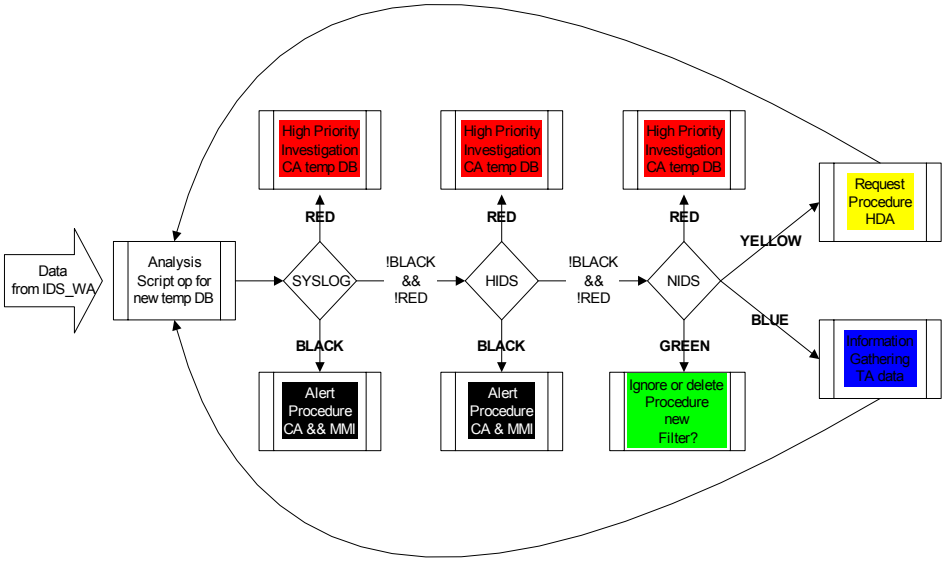


Figure 2: Analysis procedure

To enforce the relevance of alarms, knowledge about

- The attack signatures in the payload
- Perimeter defense settings (firewalls, reverse proxies, etc)
- Host properties and vulnerabilities (Operation System, patch level, etc)

- Abnormal messages around the time for the alarms. (e.g. reboots, service failures, etc)

The last step is to investigate the most frequent alarms. These may be an indication of misconfigurations or software bugs, denial of service attacks or virus respectively worms.

Overall, the alarm analysis activities can be broadly grouped into filtering, aggregation and correlation. Using the sum of the severities during a time period captures some of the characteristics for listed alarms (high severity, lots of alarms and high rate). This is one of the approaches taken in this paper.

Figure 3 presents the implementation in two of our agents based on the description above. Normalisation is the process to transform the data into a uniform format. Messages are aggregated using a text based distance metric. In the following the detail of static and adaptive filtering, aggregation and correlation is discussed.

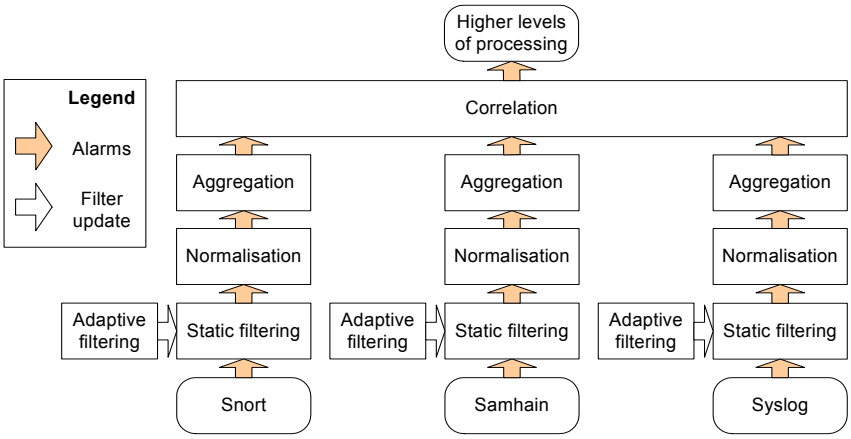


Figure 3: Overview of Methods

3.1 Static filtering

Under normal operations, IDSs produce a lot of messages, therefore it excessively time consuming to distinguish false alarms from real attacks. In particular there were a lot of uninteresting messages coming from untuned Syslog and Samhain. Practically in LCCIs, tuning requires a static network, were changes almost never happen or it requires a non-affordable effort for the administration. The resulting amounts of uninteresting messages have a low severity, though through their number of occurrence, they cannot be neglected. Therefore static filters exclude data from Samhain and Syslog that does not carry any valuable information, such as messages that Samhain checks a file or that Syslog is doing a garbage collection.

Filters are implemented either as an *ignore* filter or as a *delete* filter. The ignore filters keep the alarms in the database but they are not forwarded to the next agent. But they are saved for forensic investigation. The delete filters removes alarms permanently from the database. All static filters have a limited duty cycle defined by the operator and have to be reactivated afterwards.

3.2 Adaptive filtering

Static filter deals with known problems. However, since it is not possible to foresee all future misconfigurations, adaptive filtering algorithms were implemented to suggest new filter rules to the administrator. This is based on Naïve Bayesian (NB) learning [Hec96]. The idea is to train a NB text classifier to classify messages by looking at word statistics of messages. The NB-classifier has to be trained first with a subset of messages labelled as interesting or not. During training, a knowledge base is automatically acquired, enabling the NB-classifier to assess whether unknown messages are interesting.

The adaptive filters are used in the following workflow:

1. For performance reasons, the algorithm for adaptive filtering is launched on a periodic basis.

2. The algorithm suggests filter rules for top scoring events to a human expert via HMI.

3. The reply is also used to optimise the training corpus, achieving on-line retraining.

4. In order to avoid over learning effects, features, hardly ever occurring in time, are reduced in their significance and are finally forgotten.

3.3 Aggregation

Repeated, identical alarms do not provide any additional information. It would reduce the information overload if each all these alarms were represented in only one alarm including the number of its frequency. The relevant fields for aggregation have to be defined for each source individually. For Snort the source and destination IP, as well the server ports and the messages are relevant. Ephemeral ports can be ignored. For Syslog, the PID number is unimportant but the program and the message field is relevant. All data is aggregated within a given time window.

3.4 Correlation

Motivation to build an automatic correlation agent is the fact that human operators successfully correlate information from different INFOSEC mechanisms in search for known attacks. Moreover, humans are capable to find indications for attacks even though they are not known. However, the way a security expert analyses the information is

complicated therefore a complete model is impossible to find. The correlator uses data from the NIDS, HIDS and SYSLOG to find indications of unknown attacks.

All alarms regarding one host during a time window are considered, i.e. the selected correlation parameters are time, IP address, ports resp. application, etc. Given a time window, a time slot of a three dimensional vector is produced by taking the sum of severities from the three sensors. Based on the input pattern, the correlator decides whether this is an interesting alarm or not. For further processing just one single alarm is generated. The idea is illustrated in Figure 4.

This three-dimensional vector is then used as input for the following three correlation algorithms:

1. A simple additive correlator with the parameter 'added severity threshold'
2. A neural network [Sim98] with a variable set of neurons and hidden layers. Training is achieved by back propagation.
3. K-Nearest Neighbour [DGL96] with K as a variable parameter.

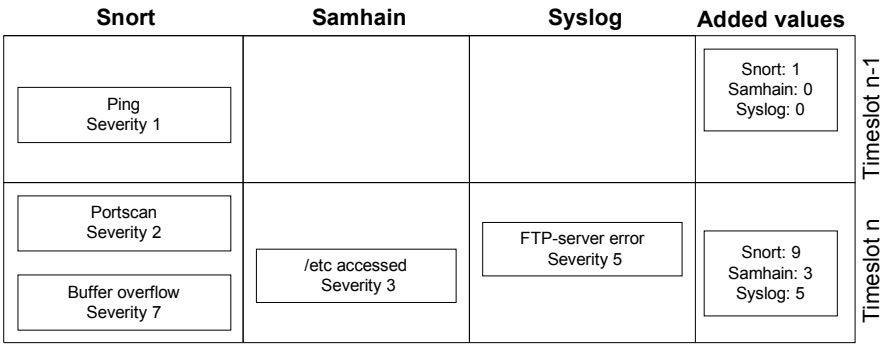


Figure 4: Alarms Appearing in Two Time Slots

The additive correlator simply builds a weighted sum of the number of occurrences of the severities within a given timeslot. The sum is then compared to a threshold generating an alarm if the chosen limit is exceeded. By varying the threshold different detection rates and false positive rates can be achieved. The other two correlation algorithms must be first trained on a subset of data before it is applied to the new real time dataflow.

4 Evaluation

4.1 Data generation

The topology of the test network used to perform the evaluation is given in Figure 5.

The test network is an image of a realistic IP environment and consists of:

- A server zone and a workstation zone. Both zones consists of Sparc 5, 10 and 20, Ultra 2,5,10 machines running Solaris 5.6 to 5.10 with various patch levels as well as several versions of Linux and PCs running on Windows 98, NT, 2000, XP installed on VMWARE.
- A jump-station running on OpenBSD can be used to login via SSH from the Internet.
- Switches, routers and a hub connecting the different nodes.
- An external zone simulating the Internet in case of trials with worms, virus, and (D)DoS.

This way, it is ensured, that no "bad" traffic contaminates the Internet.

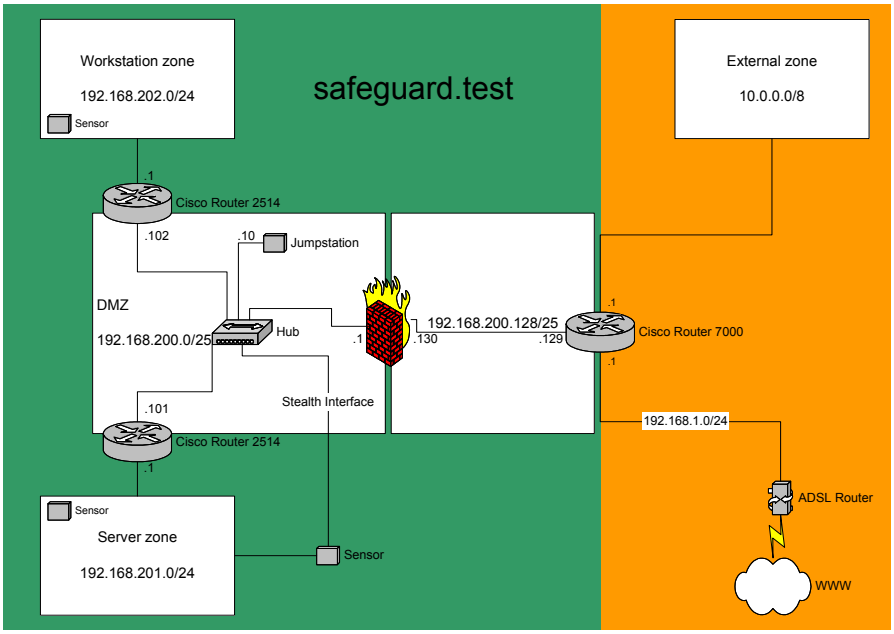


Figure 5: The Safeguard Telecom test network

From the test network data can be collected and labelled. This data may contain "normal" traffic and some attacks. In our experiments the attacks were performed using various tools and techniques, including:

- Scanning of the network using ping script once, Nessus (www.nessus.org) four times and Nmap (www.insecure.org/nmap) five times. Both Nessus and Nmap are security scanners.
- Brute-force password guessing for telnet with Brutus (www.hoobie.net/brutus), attack used twice.
- Sadmin buffer overflow attack, launched against two different hosts at separate times.
- Installing a rootkit for Solaris 2.6 once (www.honeynet.org/papers/motives).
- Various "bad" behaviour when logged in on a computer, such as allocating all space on the disk, killing as many processes as possible (once), imitating memory leaks.

The attacks were launched from the external zone simulating Internet attacks and from the server or workstation zone simulating insider attacks. During the attacks, normal usage of the computers by people working on them generated normal background noise.

Applying the resulting datasets led to the different experiments described in this paper.

4.2 Results

4.2.1 Static filtering

The data used to test the result of static filtering is gathered from all relevant hosts on the network during three weeks and includes alarms generated from attacks and from normal traffic. Figure 6 illustrates the result of static filtering on the Syslog, Snort alarms, and Samhain alarms. No alarms related to attacks were removed. Clearly Samhain alarms were reduced most drastically by this knowledge-based approach (order of magnitude). The method had the lowest impact on Snort alarms. This can be explained by the huge diversity of different network traffic patterns in contrast to the more precise knowledge of uninteresting Syslog and Samhain alarm types.

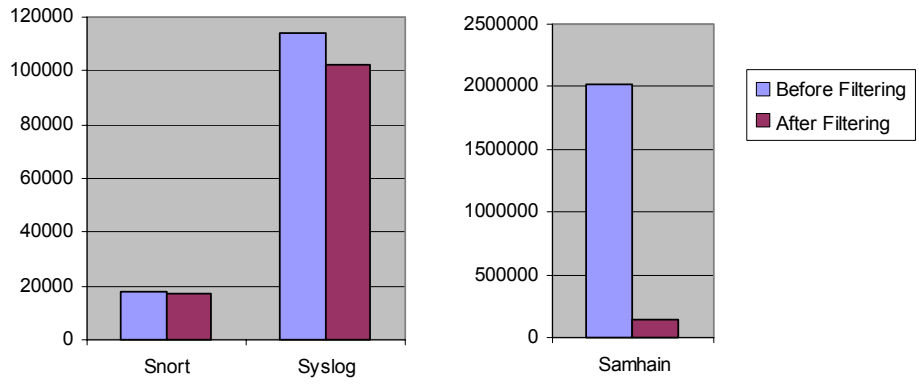


Figure 7: Static filtering for Snort Syslog and Samhain

4.2.2 Adaptive filtering

Text classification methods for adaptive filtering were applied to Syslog alarms. The data set used, when evaluating the adaptive filtering, consisted of messages collected over the period of two months. The data was labelled as "interesting" or "uninteresting" by a security expert. An interesting message contains valuable indications of an attack. For example, a message saying "telnetd[1]: Successful login as user root" or "File changed" is classified as interesting, but messages such as "Garbage collecting while idle" will be classified as uninteresting. When classifying the messages, the fields: Facility, Priority, Program and Message of each Syslog message were included. The field Level was excluded since it has the same value as the field priority.

The whole data set contained 156 212 alarms, where 18 941 of them are classified as interesting. These alarms where then divided into different sets for training and testing. The test data set contained 53 722 alarms, where 10 621 of them are classified as interesting.

Table 2 shows the results of the adaptive learning algorithm in terms of the precision of the results produced on the test data set. By precision here we mean the number of correct classification divided by the total number of alarms.

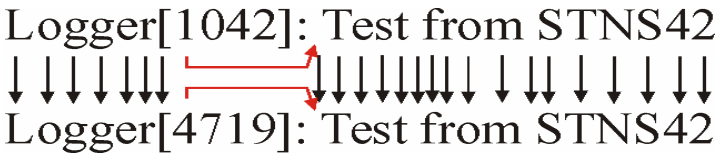
Data set	**Correct classifications**	**Incorrect classifications**	**Precision**
Test part	53682	40	0,99926

Table 1: Result of text classification for adaptive filtering

As the last column indicates, the method is likely to classify the alarms with a very high precision as long as the selected attributes for classifying the alarm messages as interesting/uninteresting and the training data remain valid in the eyes of the security expert.

4.2.3 Aggregation

Next we show the results of the aggregation algorithm, which is based on an edit distance measure with blank synchronisation. As long as the characters are the same, the next character is considered, if not, the next space is used for synchronisation. The number of equal characters in relation to the length is the percentage of similarity.



Empirical tests showed that with more than 65% similarity the same messages are recognised as being identical. A similarity of 70% means a reduction for Snort of 96.5% (from 3755 to 130) and for Syslog of 99.8% (from 13691 to 28).

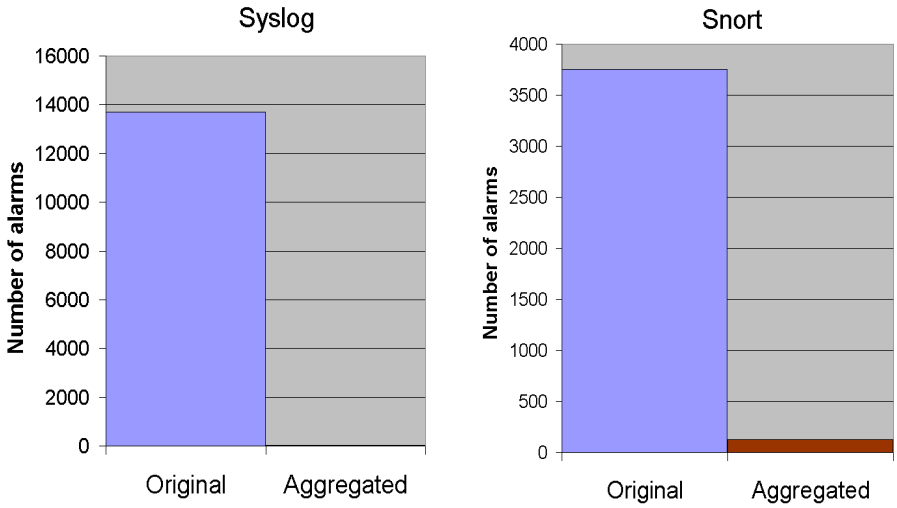


Figure 7: Original and Aggregated Messages

The influence of the time window size for the aggregation performance is shown in Figure 8. The data set being use here is smaller, but has the same characteristics as in above.

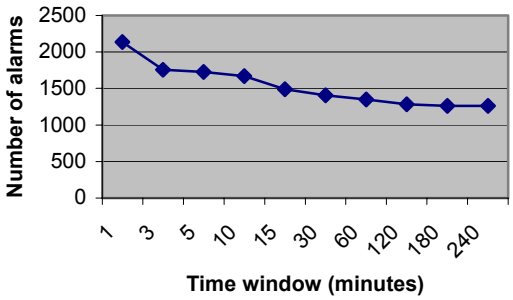


Figure 8: Number of Snort alarms after aggregation, varied size of time window

Obviously, the larger the time window the fewer alarms are left after aggregation. In order to limit the delay until alarms are sent to the higher layer processes, the first instance of a new alarm is instantly sent while at end of the time window the final aggregation result is delivered.

However, this is only the first stage in dealing with the information quality, and higher-level correlation agents are dependent on recognition of anomalous situations within an adequate short time period for vital reactions [Bu03].

4.2.4 Correlation

As mentioned in chapter 3.4 we compared the messages from the NIDS, HIDS and Syslog to produce estimation for unknown attacks. Three techniques for behaviour-based correlation were applied. Figure 9 shows the additive correlator, a neural network and a 1-nearest neighbour classifier algorithm. For the training we used a subset of eight hosts on the network during eight days.

Using a time window of 15 minutes, 6 048 three-dimensional time slots were created. 54 of the time slots were attacks. The dataset was divided into a training part, 3993 timeslots, 36 of these timeslots were attacks and the test part was the remaining part.

The additive correlator obviously gives less false positives (see ROC curve of Figure 9, left part) with a lower threshold but the detection rate decreases. For higher values of the threshold, the 1-NN algorithm and the neural network have lower false-positives rates.

The neural network correlator had an overall better performance with detection rate of 94,4% and false positives rate of 1.4%. Considering the vast amount of data being presented to a human per day this false positive rate is still rather high. Ongoing work using more correlation sources e.g. topology information, policy definition and health observations are expected to show significant improvement.

Figure 9 (right part) shows the results of the study of the K-NN algorithm for K=1...10 using the same data set, where K=1 performs best. Explanation: In real data, there are far

more time slots, that correspond to non-attack situations than to attack situations. Thus, the K-NN algorithm has more difficulties to distinguish our data sets.

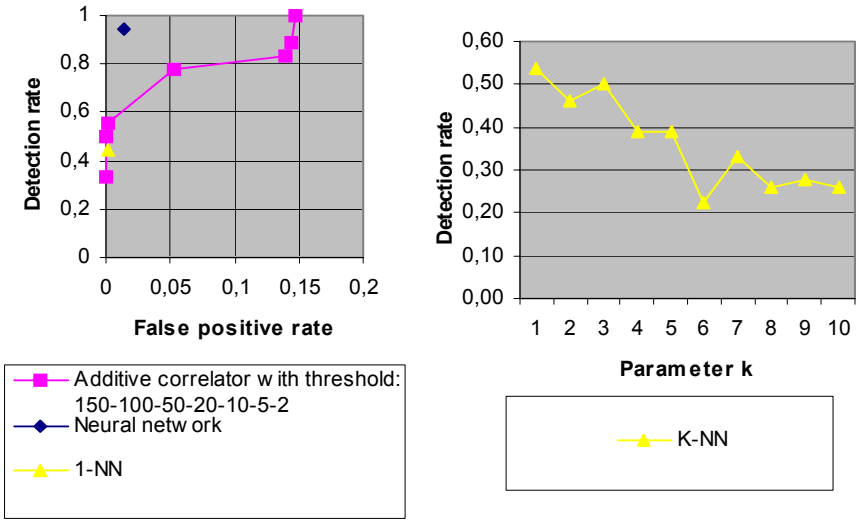


Figure 9: (Left) ROC curve for correlation methods.
(Right) Result of K-NN correlator for different values of parameter K

Table 2 summarizes the three studied methods utilizing the same data. TP, FP, FN, and TN stand for true positives, false positives, false negatives, and true negatives respectively. The overall accuracy is the sum of TP and TN divided by the sum of all four values.

Type	**TP**	**FP**	**FN**	**TN**	**Detection rate**	**FP rate**	**Overall accuracy**
Additive (Thrs.=20)	14	106	4	1931	0,778	0,052	0,946
Neural network	17	29	1	2008	0,944	0,014	0,985
1-NN	8	2	10	2035	0,444	0,001	0,994

Table 2: Correlation result summary

5 Conclusions

This paper shows that static and adaptive filtering as well as aggregation is required in order to provide a useable pre-processing of realistic IDS datasets. Practical experience revealed, that naïve Bayes classifiers reduced the amount of time to create filter rules and improved their quality.

In order to improve standard methods of correlation, e.g. rule based, three different classifiers were tested only operating on information of NIDS, HIDS, and SYSLOG. Used as an add-on, these classifiers also provide the benefit to detect unknown attacks. The neuronal network performed best. K-NN was found to be inappropriate for real world alarm message classification, where the system has to be adapted to its environment.

Nevertheless, human knowledge about the network, the vulnerabilities and the configuration is still inevitable in future. But an on-line retainable NB-classifier helps the expert to adapt the system to it dynamic environment, thus saves valuable time for appropriate countermeasures. Moreover the reduced amount and the quality of the alarms enable the expert to concentrate on the vital information.

The work in the Safeguard [Saf03] project is currently in progress, dealing with other aspects of recognition, e.g. anomaly detection [BN04], in combination with correlation of dynamic topology data, automatic actions, and information presentation to humans.

Acknowledgements

This present work was supported by the European project Safeguard IST-2001-32685. We would like to thank Thomas Dagonnier at Swisscom for valuable support. The Safeguard agent architecture has been developed with the input from all the research nodes of the project, the cooperation of whom is gratefully acknowledged.

References

[MCZ+00] S. Managanaris, M. Christensen, D. Zerkle, and K. Hermiz, A Data Mining Analysis of RTID Alarms, Computer Networks, 34(4), Elsevier pub., Oct 2000.

[BN04] K. Burbeck, S. Nadjm-Tehrani, "ADWICE - Anomaly Detection with Fast Incremental Clustering.", Technical Report, Dept. of Computer and Information Science, Linköping University. February 2004.

[Bu03] K. Burbeck, S. G. Andres, S. Nadjm-Tehrani, M. Semling, and T. Dagonnier. “Time as a Metric for Defence in Survivable Networks”. Proceedings of the Work in Progress session of 24th IEEE Real-Time Systems Symposium (RTSS 2003), Dec. 2003.

[Chy03] T. Chyssler. “Reducing False Alarm Rates in Intrusion Detection Systems”, Master thesis No. LITH-IDA-EX-03/067-SE, Linköping University (2003).

[CM02] F. Cuppens and A. Miége. “Alert Correlation in a Cooperative Intrusion Detection Framework”. Proceedings of the 2002 IEEE Symposium on Security and Privacy. 2002. Pages 187 – 200.

[DGL96] L.Devroy, L. Györfi and G. Lugosi. “A Probabilistic Theory of pattern Recognition”. Springer Verlag, New York Inc, 1996.

[DW01] H. Debar and A. Wespi. “Aggregation and Correlation of Intrusion-Detection Alerts”. Proceedings of the fourth International Symposium on Recent Advances in Intrusion Detection (RAID). Springer Verlag, October 2001. Pages 85-103.

[Hec96] D. Heckerman. “A Tutorial on Learning With Bayesian Networks”. Technical Report MSR-TR-95-06, Microsoft Research. March 1995 (Revised November 1996).

[HA03] J. Haines, D. K. Ryder, L. Tinnel and S. Taylor. “Validation of Sensor Alert Correlators”., Security & Privacy Magazine, IEEE, Vol. 1, No. 1. January/February 2003. Pages. 46 - 56.

[Hug01] J. McHugh. “Intrusion and Intrusion Detection”. International Journal of Information Security. Vol 1, No 1. Springer Verlag, August 2001. Pages 14 – 35.

[JD02] K. Julisch and M. Dacier. “Mining Intrusion Detection Alarms for Actionable Knowledge”. Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM Press, July 2002. Pages 366-375.

[MD03] B. Morin and H. Debar. “Correlation of Intrusion Symptoms: an Application of Chronicles”. Proceedings of the sixth International Symposium on Recent Advances in Intrusion Detection (RAID). Springer Verlag, November 2003. Pages 97 – 112.

[NCR02] P. Ning. Y. Cui and D. S. Reeves. “Constructing Attack Scenarios through Correlation of Íntrusion Alerts”. Proceedings of the 9th ACM conference on Computer and communications security. ACM Press, 2002. Pages 245 – 254.

[PFV02] P. A. Porras, M. W. Fong and A. Valdes. ”A Mission-Impact-Based Approach to INFOSEC Alarm Correlation”. Proceedings of the fifth International Symposium on Recent Advances in Intrusion Detection (RAID). Springer Verlag, October 2002. Pages 95–114.

[QL03] X. Qin and W. Lee. “Statistical Causality Analysis of INFOSEC Alert Data”. Proceedings of the sixth International Symposium on Recent Advances in Intrusion Detection (RAID). Springer Verlag, November 2003. Pages 73 – 93.

[Saf03] Safeguard website: http://www.ist-safeguard.org.

[Sc01] P. Scerri. ”Designing Agents for Systems with Adjustable Autonomy”. PhD thesis No. 724, Linköping University (2001).

[Sim98] J. Sima. “Introductions to Neural Networks”. Technical report No V-755, ICS CAS, Prague, 1998.

[VS01] A. Valdes and K. Skinner. ”Probabilistic Alert Correlation”. Proceedings of the fourth International Symposium on Recent Advances in Intrusion Detection (RAID). Springer Verlag, October 2001. Pages 54-68.

[YBU03] V. Yegneswaran, P. Barford and J. Ullrich. “Internet Intrusions: Global Characteristics and Prevalence”. Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and Modelling of Computer Systems, June 2003. Pages 138-147.

[Yi03] X. Yin, K. Lakkaraju, Y. Li and W. Yurcik. ”Selecting Log Data Sources to Correlate Attack Traces for Computer Network Security: Preliminary Results”. Proceedings of the 11th International Conference on Telecommunication Systems, Modelling and Analysis, October 2003.

Alert Verification
Determining the Success of Intrusion Attempts

Christopher Kruegel and William Robertson
Reliable Software Group
University of California, Santa Barbara

{chris,wkr}@cs.ucsb.edu

1 Introduction

Recently, intrusion detection systems (IDSs) have been increasingly brought to task for failing to meet the expectations that researchers and vendors were raising. Promises that IDSs would be capable of reliably identifying malicious activity never turned into reality. While virus scanners and firewalls have visible benefits and remain virtually unnoticed during normal operation, intrusion detection systems are known for producing a large number of alerts that are either not related to malicious activity (false positives) or not representative of a successful attack (non-relevant positives). Although tuning and proper configuration may eliminate the most obvious spurious alerts, the problem of the vast imbalance between actual and false or non-relevant alerts remains.

One problem is the fact that intrusion detection systems are often run without any (or very limited) information of the network resources that they protect. Marty Roesch, the developer of Snort [Sno, Ro99], routinely brings up this point in his whitepaper [RNA] and posts to security mailing lists [Sec] and calls for an IDS that possesses knowledge of the network components it defends. The classic example that Marty uses is the scenario of a Code Red attack that targets a Linux web server. It is a valid attack that is seen on the network, however, the alert that an IDS raises is of no use because the service is not vulnerable (as Code Red can only exploit vulnerabilities in Microsoft's IIS web server). To mitigate this problem, Roesch introduces a concept called RNA, real-time network awareness [RNA]. RNA is based on passive network monitoring to establish an overview of the hosts and services that are being protected. This overview contains enough contextual information to distinguish between Linux and Windows servers, thus enabling a "network-aware" IDS to discard a Code Red attack against a Linux machine.

The problem is that the concept of network-awareness is not broad enough to completely capture the complexity that is at the core of excessive amounts of false alarms. When a sensor outputs an alert, there are three possibilities.

1. The sensor has correctly identified a successful attack. This alert is most likely relevant (i.e., a true positive).
2. The sensor has correctly identified an attack, but the attack failed to meet its objectives (i.e., non-relevant positive).
3. The sensor incorrectly identified an event as an attack. The alert represents incorrect information (i.e., a false positive).

Most people/sites are only interested in type-1 alerts. Although some sites might be interested in failed attack attempts (type-2), the corresponding alert should be differentiated from a successful instance. The key idea of alert verification is to distinguish between successful and failed intrusion attempts (both false and non-relevant positives). While contextual information can be helpful to perform this distinction (as we have seen in the example with the Code Red worm above), it is not always sufficient. Consider a Code Red worm attacking a patched Microsoft IIS server. In this case, it is not enough to know which operating system the host is using, but it is also required to know which application is running and which patches have been applied.

Alert verification is a term that we use for all mechanisms that can help to determine whether an attack was successful or not. This information is passed to the intrusion detection system to help differentiate between type-1 alerts on one hand and type-2 and type-3 alerts on the other hand. When the success of an attack is *a priori* impossible (e.g., no vulnerable service is running) or cannot be verified (e.g., the attack failed because incorrect offsets were used), the IDS can react accordingly and suppress the alert or reduce its priority.

The next section classifies different mechanisms to implement alert verification. In Section 3, we present our implementation, which is based on Nessus [Nes] and Snort [Sno]. With this configuration, we demonstrate how Snort, an open-source network intrusion detection system, was modified to utilize information provided by Nessus, a popular vulnerability scanner, to significantly improve Snort's detection accuracy. Section 4 gives more details on our experience with the deployed tool. Section 5 discusses related work and potential areas where the presented system could be applied to. Section 6 concludes and outlines future work.

2 Alert Verification

Alert verification is defined as the process of verifying the success of attacks. That is, given an attack (and a corresponding alert raised by an intrusion detection system), it is the task of the alert verification process to determine whether this attack has succeeded or not.

There are different techniques that can be used to perform this verification. One possibility is to compare the configuration of the victim machine (e.g., operating system, running services, service version) to the requirements of a successful attack. When the victim is

not vulnerable to a particular attack (because the configuration does not satisfy the attack requirements), then the alert can be tagged as failed. For example, a certain exploit might require that the victim is running a vulnerable version of a Microsoft IIS server. When the victim's configuration shows that it is running an Apache server on Linux, the exploit cannot succeed.

Another possibility is to model the expected "outcome" of attacks. The "outcome" describes the visible and checkable traces that a certain attack leaves at a host or on the network (e.g., a temporary file or an outgoing network connection). When an alert has to be verified, the system can check for these traces.

An important distinction between different alert verification mechanisms is whether they are *active* or *passive*. Active verification mechanisms are defined as mechanisms that gather configuration data or forensic traces after an alert occurs. Passive mechanisms, on the other hand, gather configuration data once (or at regular, scheduled intervals) and have data available before the attack occurs. Both active and passive techniques can be used to check attack requirements against victim configurations. To check for traces that might be left after an attack, only active mechanisms can be employed. Note that the distinction between active and passive mechanisms is solely based on the *point in time* when the configuration or forensic data is collected. Passive mechanisms perform this task before an alert is received, active mechanisms perform it as a reaction to a received alert.

The most important requirement for the alert verification process is *accuracy*. An accurate verification process will keep the number of false negatives (i.e., an alert is marked as non-relevant, when in fact it is) and false positives (i.e., an alert is marked as relevant, although it is not) low. There are different factors that influence accuracy. One factor is the quality of the data that is gathered, another factor is its timeliness. Both factors are critical; it is not sufficient to have high quality data that is out-of-date, but it is also unsatisfactory when incorrect data is collected.

Another requirement is a low cost of the verification process, where cost is measured along two axes. One axis reflects the cost of deploying and maintaining the alert verification system. The other axis reflects the costs of impact of the verification process on the normal operation of the network. This cost includes whether it necessary to shut down regular network operations to perform alert verification, or whether the alert verification process has adverse effects on the running services.

In the following, we describe the different ways to verify the success of attacks in more detail and highlight the individual advantages and disadvantages. Note that the following description present individual approaches. However, it is possible and common to combine techniques to compensate for drawbacks of individual techniques and to combine their advantages.

2.1 Passive

As mentioned above, passive verification mechanisms depend on *a priori* gathered information about the hosts, the network topology, and the installed services. A description of

the network installation is required and can be, for example, specified in a formal model such as M2D2 [MMDD02].

Given an alert, it is possible to verify whether the target of the attack exists and whether a (potentially vulnerable) service is running. For remote attacks, it is also possible to check whether malicious packets can possibly reach the target, given the network topology and the firewall rule configuration, or whether the target host reassembles the packets as expected by the intruder (e.g., using the tool by Shankar and Paxson [SP03]). The real-time network awareness approach [RNA] described above would also fall into this class.

The advantage of passive mechanisms is the fact that they do not interfere with the normal operation of the network. In addition, it is not necessary to perform additional tests that delay the notification of administrators or the start of active countermeasures. A disadvantage of passive mechanisms are potential differences between the state stored in the knowledge base and the actual security status of the network. New services might have been installed or the firewall rules might have been changed without updating the knowledge base. This can lead to attacks that are tagged as non-relevant, even though a vulnerable target exists. Another disadvantage is the limitation of the type of information that can be gathered in advance. When the signature of an attack is matched against a packet sent to a vulnerable target, the attack could still fail for a number of other reasons (e.g., incorrect offset for a buffer overflow exploit). To increase the confidence in verification results, it is often required to actively check audit data recorded at the victim machine.

2.2 Active

Active alert verification mechanisms do not rely on *a priori* gathered information. Instead, the verification process actively initiates the information gathering process when an alert is received. This information gathering process can check the current configuration of the victim host (see Section 2.2.1), or scan for attack traces (see Section 2.2.2 and Section 2.2.3).

2.2.1 Active with remote access

Mechanisms in this group require that a network connection can be established to the victim machine. One active verification mechanism with remote access is based on the use of vulnerability scanners. A vulnerability scanner is a program specifically designed to search a given target (piece of software, computer, network, etc.) for weaknesses. The scanner systematically engages the target in an attempt to assess where the target is vulnerable to certain known attacks. When an attack has been detected, a scanner can be used to check for the vulnerability that this attack attempts to exploit. Note that a vulnerability scanner could also be used in a passive setup. In this case, the full range of scans would be run in advance (or in regular intervals).

A network connection permits scanning of the attack target and allows one to assess whether a target service is still responding or whether it has become unresponsive. It

also enables the alert verification system to check whether unknown ports accept connections, which could be evidence that a back-door is installed. In this case, however, care must be taken to prevent false positives that stem from dynamically allocated ports. To this end, one could use black-lists of well-known back-door ports, white-lists that specify port ranges for well-known applications (e.g., X servers), or service fingerprinting (such as the one recently added to nmap [Fy]) to detect legitimate applications. Also, the active verification system can keep a list of applications that were found running during the last scan and raise an alert when this list changes.

Active alert verification has the advantage, compared to passive mechanisms, that the information is current. This allows one to assess the status of the target host and the attacked service and to recognize changes at the victim host that serve as an indication of an attack.

Although the information is current, however, it might not be completely accurate. One has to consider that a vulnerability scanner can also have false positives and false negatives. When an alert is verified, if the vulnerability scanner determines that the service is vulnerable when in fact it is not, the alert is simply reported by the IDS. In this case, the alert is a false positive (because the service is not vulnerable) and the verification mechanism has failed. However, the security of the system is not affected, and without verification, the alert would have been reported as well. A more significant problem are false negatives. In this case, a valid alert is suppressed because the vulnerability scanner determines that the target is not vulnerable when in fact, it is. Although such a scenario is very undesirable, it is not very likely to occur frequently. The reason is that a vulnerability scanner actually launches a basic instance of the attack. When this attack fails, it is very improbable that a more sophisticated instance succeeds.

Another drawback is the fact that active actions are visible on the network and it is possible that scanning has an adverse effect on one's own machines. Port scanning consumes network bandwidth and resources at the scanned host. To minimize the impact on a operational network, results can be cached for some time. This is especially important when an intruder runs scripts that repeat the same attack with different parameters. Note, however, that caching involves a trade-off between resource usage and accuracy. When results are cached too long, the advantage of active verification is reduced. As scans are only initiated on a per-alert base, it is not necessary to run all tests that a vulnerability scanner includes, but at most a single one for each alert (minus those for which cached results are available).

In addition, tests run by a vulnerability scanner might crash a service. A vulnerability scanner can perform tests in a *non-intrusive* or in an *intrusive* manner. When running non-intrusive tests, a vulnerability is not actually exploited, but inferred from the type and version of a running service (e.g., by analyzing banner information). When running an intrusive test, the vulnerability is actually exploited. While this delivers more accurate results, it often results in the crash or disruption of the victim service. Sometimes, the crash of a service process can be tolerated, for example, when the service is implemented using multiple threads (such as Apache's thread pool). In this case, the failure of a single thread does not have a negative impact, because the other threads still serve requests. In addition, the failed thread is automatically restarted after a short period of time. On the other hand, when the crash of a service process interrupts the whole service, then the corresponding test should be excluded altogether from the active verification process. This also helps to

prevent a possible attack where an attacker triggers an alert to have the alert verification system check and subsequently crash the service. The problem of selecting the appropriate tests is a result of the conflict between the goal of getting accurate results and the goal of having minimal impact on the operational network. While intrusive tests are more reliable in obtaining proper results, the risk of affecting services is greater.

Note that the alert verification mechanism should only be used to check alerts raised by packets that can possibly reach their destination. That is, the intrusion detection system (together with the alert verification system) should be located behind a firewall. This makes sure that only relevant packets are scanned for attacks by the IDS and later verified. Otherwise, an attacker could potentially bypass the firewall and launch attacks by means of the alert verification system.

The scope of remote scans is also limited, in that the identification of some evidence associated with an attack might require local access to the victim machine. In addition, one has to make sure that the alerts generated in response to the activity of the vulnerability scanner are excluded from the correlation process.

2.2.2 Active with authenticated access

Mechanisms in this group gather evidence about the result of an attack using authenticated access to the victim host. The difference with respect to the previous group of techniques is the fact that the alert verification system presents authentication credentials to the target host.

Active verification with authenticated access can be implemented by creating dedicated user accounts with appropriate privilege settings at the target machines. The alert verification system can then remotely log in and execute scripts or system commands. This allows one to monitor the integrity of system files (e.g., the password file or system specific binaries) or check for well-known files that are created by attacks (e.g., worms usually leave an executable copy of the worm on the file system). In addition, programs that retrieve interesting forensic data such as open network connections (such as netstat), open files (such as lsof) or running processes (such as ps) can be invoked.

The advantage of mechanisms in this group is the access to high-quality data gathered directly from a target machine. One downside is the need to configure each machine for authenticated remote access. This might be cumbersome in large network installations or when hosts with many different operating systems are used. On the other hand, in large networks, such accounts may already exist for maintenance purposes and can be also used for gathering of forensic evidence. Another problem is the fact that the information provided by general user-space tools might not be as complete and accurate as it is possible with specialized, often kernel-space tools.

2.2.3 Active with dedicated sensor support

Mechanisms in this group require, in addition to authenticated access, special auditing support installed at the target machines. This auditing support can be operating system

extensions or special purpose tools, such as host-based intrusion detection systems. The differences between using standard tools and relying on dedicated sensors is that standard tools are common in most distributions. In addition, dedicated sensors often need complex configuration.

Dedicated sensors can be used to monitor system calls issued by user applications. This allows one to check for the spawning of suspicious processes (e.g., shell invocations) or for accesses to critical files. As opposed to standard tools that present a current snapshot of the system, sensors can keep a record of malicious activity. This enables the verification system to gather events that are only visible for a short period of time, which could be missed by a snapshot.

The advantage of dedicated sensor support is the ability to provide the most detailed and accurate audit records. The drawback is the effort required to install and configure these sensors, and the fact that certain sensors are not available for all platforms.

2.2.4 General issues of active verification

One issue that affects all active verification mechanisms is the problem that information is gathered directly from the victim machine. It can be argued that an attacker can tamper with the compromised system to eliminate suspicious traces or, at least, hide her activity from the auditing system. This is particularly true when the information is gathered remotely (e.g., using a vulnerability scanner).

There are different approaches to addressing this problem. One possibility is to operate in a best-effort mode and attempt to scan the potential victim host as fast as possible after the alert is received. This, of course, offers a small window of vulnerability that can be exploited by the attacker. A more secure option is to delay packets that have raised an alert until the verification mechanism has finished. This makes sure that the victim host has not been compromised by this attack, but it requires an in-line intrusion detection system.

Another option can be used when data is directly gathered on the victim machines via scripts or dedicated sensors. Here, audit tools should be run at least with privileges that require administrative (i.e., root) access to be turned off. This maintains the integrity of the sensor when the intruder obtains user access only or manages to crash a service with a denial-of-service attack. The sensors operate in a best-effort mode and deliver accurate results as long as possible. Also, simply disabling auditing is a suspicious action by itself. A more secure option is the use of a more restrictive access control system such as LIDS [LID] or Security-Enhanced Linux [LS01]. These systems can prevent the administrator from interfering with the audit facility such that physical access to the machine is required to change or disable security settings.

3 Implementation

After the general discussion of various alert verification mechanisms in the previous section, the remainder of the paper presents the implementation and an evaluation of our system, an active verification tool with remote access. The system is realized as an extension to Snort [Sno, Ro99] that uses NASL [Ar02] scripts written for the Nessus [Nes] vulnerability scanner. More precisely, the system is implemented as a patch to Snort, which integrates the Nessus vulnerability scanner into Snort's core to perform verification of alerts. Nessus was chosen as a verification mechanism because of the generally high quality of its vulnerability checks, its minimal impact on production networks, and the ease with which it could be integrated into Snort. Deployment of the combined system is also no more costly than deploying a stand-alone Snort sensor. The modifications mainly consist of an addition to Snort's alert processing pipeline which intercepts alerts to be passed to enabled alert plug-ins. These alerts are queued for verification by a pool of verification threads. This allows Snort to continue processing events while alert verification takes place in the background. Because our verification system is implemented as a part of the Snort sensor, both are located together. This is not a requirement of active verification, however, and it would also be possible to have a single verification system that receives alerts via the network from multiple sensors. In this case, the alert verification tool could be integrated into the alert collection framework. Because we wanted to allow the stand-alone use of Snort with the verification enhancements, the use of multiple Snort sensors would imply that multiple verification modules are running. This should be no problem, because the performance impact of the verification tool is low. The only drawback is the fact that cached results cannot be shared between different instances.

For each alert selected for processing by a verification thread, the CVE ID, a unique identifier for vulnerabilities which is assigned by the Common Vulnerabilities and Exposures project [CVE], is extracted and used as an index into Nessus' collection of NASL scripts. NASL is the scripting language designed for the Nessus security scanner. Its aim is to allow anyone to easily and quickly write a plug-in to test for a certain security hole, which can then be used by the Nessus scanner. If an appropriate NASL script is found, it is executed by an embedded NASL interpreter against the victim host or network identified by the current alert. The vulnerable status of the target is extracted from the NASL interpreter's output and is used to flag the detected attack as either successful or unsuccessful. The alert is then queued for output by any enabled alert plug-ins. The result of each verification is also cached for a configurable period in order to reduce load on the network. When no appropriate NASL script is found, the alert has to be flagged as undetermined.

Post-processing systems (e.g., alert correlation engines or system administrator scripts) can then utilize this additional information when performing their analysis upon the alert stream generated by verification-enabled Snort sensors.

An overview of the architecture of the current implementation is depicted in Figure 1. The tool is available as a patch to Snort for download at `http://www.cs.ucsb.edu/~wkr/projects/ids_alert_verification/` [SAV].

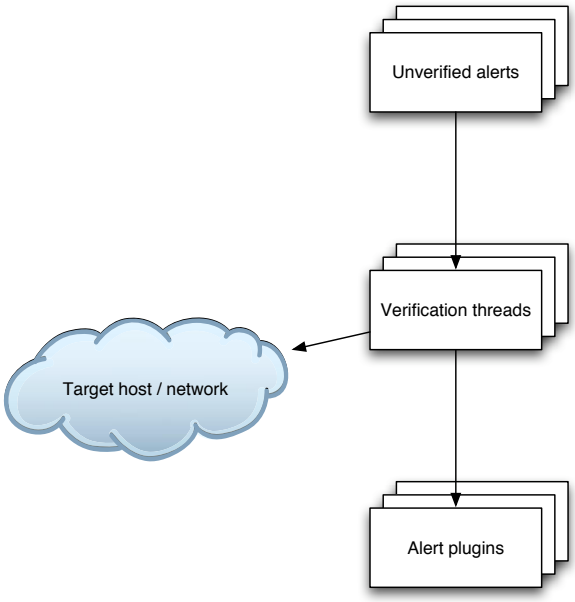


Figure 1: Snort Alert Verification Architecture

4 Evaluation

The current revision of our alert verification patch to Snort has been evaluated on an experimental test bed with regards to its effectiveness in reducing Snort's false alarm rate. Three machines were present on this test bed:

1. an attacker machine
2. a target machine
3. a machine with an alert verification-enabled Snort sensor deployed

A variety of known vulnerabilities were introduced on the target machine, and corresponding signatures to detect attacks using these vulnerabilities were enabled on the sensor machine. A wide range of attacks were run against the target by the attacker with alert verification enabled and disabled to compare the number of false positives produced by Snort. Attack traffic was generated from a mix of Nessus runs and publicly-available exploits. The results are shown in Table 1.

As one can see, with Snort running in stand-alone mode, 6659 attacks against the target machine were reported. However, because either no vulnerable service was actually present on the target or the targeted service was not vulnerable, most of these attacks could not have been successful and can thus be considered non-relevant. Only 24 of the alerts

	Alerts	True Positives	False Positives
Stand-alone	6,659	24	99.64%
Verification enabled	24	24	00.00%

Table 1: Alert Verification – Evaluation Results.

produced by Snort were true positives, and we arrive at a false positive rate (or, to be more precise, a non-relevant positive rate) of 99.64%. With alert verification enabled, however, alerts which attempted to exploit missing or invulnerable services were tagged as such; thus, the false alarm rate for Snort with alert verification enabled dropped to 0.00% and only the 24 actual attacks were reported. Manual inspection of the alert stream was used to verify that no false positives or non-relevant alerts were produced.

It is important to note in interpreting these results that Snort and Nessus are open to generating both true and false positives and negatives. Thus, the following scenarios are possible given their combination in our alert verification implementation:

1. true positive / true positive

   In this scenario, the attack is correctly detected, and the service is correctly reported as vulnerable.

2. true positive / false positive

   Here, the attack is correctly detected, and the service is incorrectly reported as vulnerable.

3. true positive / true negative

   Under this scenario, the attack is correctly detected, and the service is correctly detected as invulnerable.

4. true positive / false negative

   In this scenario, the attack is correctly detected, but the service is incorrectly determined to be invulnerable.

5. false positive / true positive

   With this scenario, benign traffic is misreported as an attack, and the service is reported as vulnerable to the reported attack.

6. false positive / false positive

   Here, benign traffic is misreported as an attack, and the service is incorrectly reported as vulnerable to the reported attack.

7. false positive / true negative

   Under this scenario, benign traffic is misreported as an attack, and the service is correctly determined to be invulnerable.

8. false positive / false negative

   With this scenario, benign traffic is misreported as an attack, and the service is incorrectly determined to be invulnerable.

9. false negative / true positive

   Here, an attack is undetected by the IDS, but the service would have been reported as vulnerable by the vulnerability scanner.

10. false negative / false negative

    Here, an attack is undetected by the IDS, and the service would have been misreported as invulnerable by the vulnerability scanner.

Clearly, from the above list one can see that the ideal scenarios are 1 and 3 in that alert verification correctly reinforces confidence in IDS alerts in the first case as well as suppresses incorrectly alerts in the second case. Scenarios 2, 5, and 6 correspond to a false positive from an IDS without alert verification, thus the addition of alert verification does not degrade the effectiveness of the IDS on its own. Scenarios 9 and 10 correspond to a successfully evaded IDS without alert verification; since alert verification triggers on IDS alerts, the technique cannot help in these scenarios. In scenario 7, alert verification is successful in suppressing a false positive that would be reported in a stand-alone IDS. In scenario 8, although it is unfortunate that both components fail, the end result is that no successful attack occurs, and furthermore a false positive from the IDS is suppressed; therefore, it is not a cause for concern outside of the inaccuracy of the IDS and vulnerability scanner. Thus, the only scenario in which alert verification may degrade the effectiveness of a stand-alone IDS is 4. However, because of the relative ease of writing correct vulnerability assessment checks as compared to IDS signatures, the probability of this scenario occurring in the real world is not great. Additionally, in this evaluation manual inspection was used to verify that no scenarios resulting in false negatives were present.

To gather real-world attack traffic and assess the amount of alerts that the system is capable of identifying as non-relevant in a more realistic scenario, we deployed two honeypots. One of the honeypot machines was running a standard RedHat 7.2 Linux installation, the other one was running an unpatched version of Microsoft Windows 2000 Server. Both hosts had a considerable amount of services with known vulnerabilities. The network link to both honeypots was monitored by Snort-2.0.2, using its complete set of 2625 rules.

During a period of 14 days, Snort reported 164,415 raw alerts referring to attacks against the RedHat Linux machine and 79,198 raw alerts referring to attacks against the Windows machine. Among these raw alerts, we noticed a large amount of attacks related to the Slammer and Nachia worms. Also, a large amount of scan activity against ports commonly used by web proxy and socks proxy servers was registered. We believe that these scans are performed by spammers that use these proxies as mail relays. Given the raw alerts, the alert verification process was capable of tagging 161,166 attacks against the Linux host (98.3%) and 78,785 attacks against the Windows host (99.4%) as unsuccessful. This tagging was manually verified, and we concluded that all attacks that have been tagged as unsuccessful actually failed (the manual checks were doable because most attacks targeted non-existent

services). Although a default installation of Snort was used, the numbers clearly indicate that real-world attack traffic produces many false or non-relevant positives that can be suppressed using alert verification.

The results shown above demonstrate that alert verification improves the false positive rate of NIDS implementations. However, the current alert verification implementation for Snort suffers from several limitations. One is that the granularity of CVE IDs, which is somewhat necessitated by the choice of Nessus as the verification component, reduces the effectiveness of the tool as a whole. This stems from the lack of other additional information, such as host architecture, revision of the vulnerable program, etc. which could result in the vulnerability testing script reporting the service as not vulnerable when in fact it is. It is also worth noting that this limitation generalizes to the fact that, barring implementation flaws, active alert verification is only as good as the available verification scripts, just as the quality of a signature-based IDS depends on the quality of its signatures.

Another issue is that the classification scheme of vulnerable, not vulnerable, or undetermined may, as members of the focus-ids mailing list [Sec] have pointed out, not be expressive enough to capture information that is relevant to network security officers.

5 Related Work

Several vendors and researchers [Gu02, De03, RNA] have proposed to include vulnerability analysis data when processing IDS alerts. The idea is to utilize previously gathered information to reduce the noise of the alert stream produced by intrusion detection sensors and disambiguate their results. These methods are all different realizations of passive alert verification techniques as described in Section 2. In this paper, on the other hand, an active alert verification mechanism is proposed. We query the potential victim in response to the sign of an attack to get the current configuration of the victim that either supports or refutes the hypothesis that a successful intrusion has occurred.

An important, related analysis process that also takes as input the alerts produced by intrusion detection systems is *alert correlation*. Its main task is the aggregation of alerts to provide a high-level view (i.e., the "big picture") of malicious activity on the network. A major problem for correlation systems are false positives, which can degrade the quality of their results significantly. It is evident that correlating alerts that refer to failed attacks can result in the detection of whole attack scenarios that are actually non-existent.

Previous work [CM02, NCR02] states that alert correlation can be used both to reduce the total number of alerts and to reduce the number of false alerts. The latter, namely the reduction of false alerts, is directly related to our goal. However, the correlation systems mentioned above assume that real attacks trigger more than a single alert. As a result, the systems can focus on alert clusters and discard all alerts that have not been correlated. Unfortunately, this assumption has not been substantiated by experimental data or supported by a rigorous discussion. We claim, therefore, that the reduction of false alerts is an important *prerequisite* to achieve good correlation results instead of an outcome of the correlation process itself. Also, a recent paper [NX03] on alert correlation mentions that

"false alerts generated by IDSs have a negative impact". This supports our assumption that alert verification can act as a pre-processing step for correlation systems, cleaning the input stream from spurious alerts and thus improving their results.

6 Conclusions and Future Work

We propose alert verification as a process that is launched in response to an alert raised by an intrusion detection system to check whether the corresponding attack has succeeded or not. When the attack has not succeeded, the alert can be suppressed or its priority reduced. This provides an effective mean to lower the number of false alarms that an administrator has to deal with. It also improves the results of alert correlation systems by cleaning their input data from spurious attacks.

We have developed an active verification system based on Snort and Nessus. As the current implementation stands, it is a useful tool for reducing the false alarm rate of Snort. There is, however, always room for improvement, and in this spirit we have planned some future directions for further development of our alert verification system. One issue is the coarse granularity of CVE IDs, which we plan to address by extending Nessus. Another planned area of development is the possible integration of an *a priori* knowledge base along with passive information gathering techniques to supplement the active verification techniques.

7 Acknowledgment

We would like to thank our shepherd Roland Büschkes for his numerous comments and the thorough reviews that helped to improve the quality of this paper.

References

[Ar02] Arboi, M.: *The Nessus Attack Scripting Language Reference Guide*. 2002. `http://www.nessus.org/doc/nasl2_reference.pdf`.

[CM02] Cuppens, F. und Miege, A.: Alert Correlation in a Cooperative Intrusion Detection Framework. In: *Proceedings of the IEEE Symposium on Security and Privacy*. Oakland, CA. May 2002.

[CVE] Common Vulnerabilities and Exposures. `http://www.cve.mitre.org/`.

[De03] Desai, N. IDS Correlation of VA Data and IDS Alerts. `http://www.securityfocus.com/infocus/1708`. June 2003.

[Fy] Fyodor. Nmap: The Network Mapper. `http://www.insecure.org/nmap/`.

[Gu02] Gula, R.: Correlating IDS Alerts with Vulnerability Information. Technical report. Tenable Network Security. December 2002.

[LID] Linux Intrusion Detection System. http://www.lids.org/.

[LS01] Loscocco, P. und Smalley, S.: Integrating Flexible Support for Security Policies into the Linux Operating System. In: *Freenix Track of Usenix Annual Technical Conference*. 2001.

[MMDD02] Morin, B., Me, L., Debar, H., und Ducasse, M.: M2D2: A Formal Data Model for IDS Alert Correlation. In: *Proceedings of the International Symposium on the Recent Advances in Intrusion Detection*. S. 115–137. Zurich, Switzerland. October 2002.

[NCR02] Ning, P., Cui, Y., und Reeves, D.: Constructing Attack Scenarios through Correlation of Intrusion Alerts. In: *Proceedings of the ACM Conference on Computer and Communications Security*. S. 245–254. Washington, D.C. November 2002.

[Nes] Nessus Vulnerabilty Scanner. http://www.nessus.org/.

[NX03] Ning, P. und Xu, D.: Learning Attack Strategies from Intrusion Alert. In: *Proceedings of the ACM Conference on Computer and Communications Security (CCS '03)*. Washington, DC. October 2003.

[RNA] RNA - Real-time Network Awareness. http://www.sourcefire.com/technology/whitepapers.html.

[Ro99] Roesch, M.: Snort - Lightweight Intrusion Detection for Networks. In: *Proceedings of the USENIX LISA '99 Conference*. November 1999.

[SAV] Snort Alert Verification. http://www.cs.ucsb.edu/~wkr/projects/ids_alert_verification/.

[Sec] SecurityFocus Mailing Lists Archive. http://www.securityfocus.com/archive.

[Sno] Snort - The Open Source Network Intrusion Detection System. http://www.snort.org.

[SP03] Shankar, U. und Paxson, V.: Active Mapping: Resisting NIDS Evasion Without Altering Traffic. In: *Proceedings of the IEEE Symposium on Security and Privacy*. 2003.

Komponenten für kooperative Intrusion-Detection in dynamischen Koalitionsumgebungen

Marko Jahnke, Martin Lies
Sven Henkel, Michael Bussmann
Forschungsgesellschaft für Angewandte
Naturwissenschaften e.V. (FGAN)
Neuenahrer Str. 20, 53347 Wachtberg
{jahnke|lies|henkel|bus}@fgan.de

Jens Tölle
Universität Bonn
Institut für Informatik, Abt. IV
Römerstr. 164, 53117 D-Bonn
toelle@cs.uni-bonn.de

Abstract:
Koalitionsumgebungen sollen für alle miteinander kooperierenden Mitglieder einen Vorteil bei der Verfolgung eines gemeinsamen Ziels erbringen. Dies gilt für die verschiedensten Anwendungsbereiche, etwa bei kooperierenden Strafverfolgungsbehörden, Wirtschaftsunternehmen oder Streitkräfte. Auch bei der Erkennung von sicherheitsrelevanten Vorgängen in vernetzten Computersystemen erhofft man sich von der Zusammenarbeit eine verbesserte Erkennungsfähigkeit sowie eine schnelle und koordinierte Reaktion auf Einbruchsversuche.

Dieser Beitrag stellt verschiedene praxisorientierte Werkzeuge für die koalitionsweite Vernetzung von Ereignismeldungs-produzierenden Sicherheitswerkzeugen vor, die wesentliche Probleme des Anwendungsszenarios lösen helfen:
Frühzeitige Anomaliewarnung – ein graphbasierter Anomaliedetektor wird als adaptives Frühwarnmodul für großflächige und koordinierte Angriffe, z.B. Internet-Würmer, eingesetzt.
Informationsfilterung – Meldungen werden beim Verlassen der lokalen Domäne entsprechend der domänenspezifischen Richtlinien zur Informationsweitergabe modifiziert (d.h. insbesondere anonymisiert bzw. pseudonymisiert).
Datenreduktion – zusätzliche Filter zur Datenreduzierung auf der Basis von vordefinierten Abhängigkeitsregeln steigern die Handhabbarkeit des Datenflusses.
Die Funktionsfähigkeit der genannten Komponenten wird derzeit in Form einer prototypischen Implementierung eines *Meta-IDS* für dynamische Koalitionsumgebungen nachgewiesen.

1 Einführung

Koalitionsumgebungen (*Coalition Environments, CEs*) finden sich in den verschiedensten Bereichen der Gesellschaft, so z. B. bei internationalen Kooperationen von Strafverfolgungsbehörden, bei Allianzen von Wirtschaftsunternehmen oder zusammenarbeitenden Streitkräften (z.B. NATO). Kennzeichnend sind oft gemeinsame Interessen oder ein gemeinsam durchzuführender Auftrag der Koalitionsteilnehmer. Gleichzeitig existiert gegenüber den Partnern in einer Koalition oft nur ein beschränktes Vertrauen, etwa wenn ein Konkurrenzverhältnis besteht.

Aus offensichtlichen Gründen können großflächige sicherheitsrelevante Aktivitäten in Computernetzen schneller aufgedeckt werden, wenn Zugriff auf eine größere Menge von angriffsabhängigen Ereignismeldungen möglich ist. Andererseits muss die Informationsweitergabe über kritische Aktivitäten und Tendenzen an andere Zusändigkeitsbereiche (Domänen) einer Koalitionsumgebung schnell und in einem interoperablen Format erfolgen, damit ein entsprechender Vorteil aus der Kooperation entstehen kann. Das Ziel besteht also darin, alle verfügbaren Angriffsinformationen zusammenzuführen, sie effizient auszuwerten und die Analyseergebnisse weiterzuleiten.

Die technische Realisierung einer derartigen Kooperation stellt sich in Form der Vernetzung verschiedener Sicherheitswerkzeuge (z.B. IDS, Paketfilter oder Integritätsprüfprogramme) dar, die Informationen über an anderer Stelle im Koalitionsnetzwerk registrierte relevante Aktivitäten erhalten, auswerten und weitergeben. Die besondere Herausforderung in dynamischen Koalitionsnetzwerken ensteht zum Einen durch Fluktuation bei den Koalitionsmitgliedern und durch sich ändernde Vertrauensverhältnisse. Diese schlagen sich in der Bereinigung von Informationen, die eine Domäne verlassen, nieder (z. B. durch Anonymisierung bzw. Pseudonymisierung von Ereignismeldunge). Zum Anderen stellt die Heterogenität der zugrundeliegenden Sicherheitswerkzeuge sowie deren durchzusetzenden Sicherheitsrichtlinien ein Hindernis für die effiziente Erkennung von Angriffen dar.

Dieser Beitrag stellt Komponenten für die Verarbeitung und Auswertung koalitionsweiter Ereignismeldungsvolumina vor, die zur Lösung dieser Fragestellungen beitragen. Der Nachweis ihrer Funktionsfähigkeit wird durch die prototypische Realisierung eines *Meta-IDS* geführt. Der Rest dieses Beitrages ist wie folgt strukturiert: In den Abschnitten 2, 3 und 4 werden die neuen Komponenten beschrieben. In Abschnitt 5 werden das Konzept und die bisher erzielten Ergebnisse vorgestellt, eingeschlossen der Implementierung. Zum Abschluss werden in Abschnitt 6 unsere Ansätze zusammengefasst und mögliche Erweiterungen skizziert.

2 Anomalieerkennung im Ereignismeldungs-Datenmodell

Die Zusammenführung von Ereignismeldungen aus unterschiedlichen Quellen dient als Ausgangsbasis für einen Ansatz zur Beurteilung der “Normalität” des aktuellen Meldungsaufkommens. Somit wird an dieser Stelle ein Ansatz der Anomalieerkennung eingesetzt. Die Details dazu werden im folgenden erläutert.

Eine der wesentlichen Herausforderungen für ein in kooperierenden IDS eingesetztes Anomalieerkennungsverfahren ist sein Umfeld. Über entsprechende Transportwege trifft eine große Zahl von Ereignismeldungen aus den angeschlossenen Domänen ein. Es ist jedoch auf Grund der Heterogenität der Datenbasis nur sehr begrenzt möglich, Annahmen über die Art und die Häufigkeit der Meldungen zu machen. Aus bereits angesprochenen Gründen kann sowohl die Frequenz der eintreffenden Ereignismeldungen, deren Inhalte und deren Aufbau stark von der konkreten Installation und Konfiguration abhängen. Das hier eingesetzte Verfahren ist ursprünglich für die Überwachung und Anomalieerkennung des

Verkehrs in Netzwerken entwickelt worden und funktioniert folgendermaßen:

Es hat sich gezeigt, dass die typischen Strukturen über die Zeit recht stabil sind, also nur selten grundlegende Veränderungen auftreten (vgl. [TdW02]). Aus diesem Grund kann man regelmäßig in kurzen Abständen den aktuellen Verkehr erfassen (möglich durch Netzwerk-Monitoring-Geräte oder auch durch Packetsniffer, in geswitchten Netzwerken an einem Spiegelport eines Switches) und in Form einer Verkehrsmatrix abspeichern. Diese Verkehrsmatrix kann als Graph $G = (V, K)$ interpretiert werden. Knoten $v_i \in V$ des Graphen G sind damit an der Kommunikation im aktuellen Zeitabschnitt beteiligte Geräte, während Kanten $k_{i,j} \in K$ der Kommunikation zwischen zwei Geräten v_i und v_j entsprechen. Die Kanten sind dabei mit der "Intensität" der Kommunikation gewichtet.

Diese Graphen können mittels *Graph-Clustering*-Algorithmen in Teilgraphen zerlegt werden. Das Clustering dieses Graphen repräsentiert somit die typische Kommunikationsstruktur innerhalb des überwachten Netzes. Clustering bedeutet das Finden einer Zuordnung jedes Knotens zu einem von mehreren Clustern. Diese exklusive Klassifizierung nennt man auch Partitionierung der Objektmenge. Jedem Objekt der Objektmenge wird genau ein Cluster zugeordnet. Formal ist ein Clustering $\Re$ die Aufteilung eines Graphen $G = (V, K)$ in Cluster

- $C_i \subseteq V, C_i \neq \emptyset, 0 \leq i \leq n-1$ mit
- $C_0 \cup C_1 \cup \cdots \cup C_{n-1} = V$ und
- $\forall 0 \leq i, j \leq n-1, i \neq j : C_i \cap C_j = \emptyset$.

Plötzliche Änderungen dieser Strukturen werden als Anomalie aufgefasst und gemeldet. Dazu ist eine Metrik erforderlich, die Ähnlichkeiten von zeitlich aufeinanderfolgenden Clusterings $\Re_t$ und $\Re_{t+1}$ bewertet.

Das im Bereich der Verkehrstrukturen genutzte Verfahren benötigt nun einige Anpassungen an das hier betrachtete Ereignismeldungsmodell. Viele der an der Anomalieerkennungskomponente eintreffenden Meldungen können analog den oben betrachteten Verkehrsgraphen unmittelbar für den Aufbau eines Graphen verwendet werden. In dieser Kategorie fallen alle Ereignismeldungen, die detailliert Ziel (das betroffene System) sowie Quelle (vermuteter Auslöser) eines Ereignisses angeben. Damit ist eine weitere Kante $k \in K$ des aufzuteilenden Verkehrsgraphen G gefunden: Die Kante k führt von der Quelle $v_{Quelle} \in V$ zum Knoten $v_{Ziel} \in V$.

In Abhängigkeit des Detaillierungsgrades kann es vorkommen, dass durch die Anwendung der lokalen Richtlinie zur Informationsweitergabe (vgl. Abschnitt 3) in manchen Ereignismeldungen nur Informationen über das betroffene Zielsystem vorhanden sind, jedoch keine Angabe über einen (vermuteten) Auslöser gemacht werden. Um solche Ereignismeldungen trotzdem in geeigneter Form im Ereignismeldungsgraph zu berücksichtigen, kann den Domänen für bestimmte Ereignismeldungstypen ein Pseudoknoten zugeordnet werden, der über Ereignismeldungen repräsentierende Kanten mit betroffenen Systemen repräsentierende Knoten verbunden wird.

Der mittels Graph-Clustering-Verfahren aufgeteilte Graph beschreibt somit die typische Struktur der eintreffenden Ereignismeldungen. Abweichungen der typischen Meldungs-

struktur sind auch hier wieder als Anomalie anzusehen und zu melden. Dabei können die im Kontext der Anomalieerkennung in Netzwerkverkehrsstrukturen genutzten Vergleichsmaße eingesetzt werden.

Auch bei dieser Anwendung von Anomalieerkennungsverfahren besteht wieder die bekannte Herausforderung, dass solche Verfahren grundsätzlich anfällig sind für die Erzeugung von Fehlalarmen (*False Positives*) oder reale Bedrohungen nicht erkennen (*False Negatives*). Auch die hier angewandten Methoden können diese Problematik nicht grundsätzlich umgehen, sondern es ist durch sorgfältige Abstimmung aller Parameter im konkreten Einsatzszenario darauf zu achten, dass die Anzahl der False Positves und False Negatives erträglich bleibt. Grundsätzlich ist anzumerken, dass Verfahren dieser Kategorie nicht ausschließlich verwendet werden sollten und immer nur zusätzliche Informationen zum aktuellen Gesundheitszustand des überwachten Netzes geben können.

3 Informationsbereinigung bei Ereignismeldungen

Bevor Ereignismeldungen eine Domäne verlassen, müssen Sie entsprechend der lokalen Richtlinie zur Informationsweitergabe (Information Sharing Policy, *IShP*) bereinigt (d.h. anonymisiert bzw. pseudonymisiert, vgl. [Fle02a], [Fle02b]) werden. Dies kann beispielsweise durch entsprechende Gateways an den Domänengrenzen bewerkstelligt werden. Dieser Prozess kann als ein Problem der bedingten Musterübereinstimmungsprüfung und Mustertransformation (*Conditional Pattern Matching and Transformation, CPMT*) für Ereignismeldungen aufgefasst werden. Es erweist sich als sinnvoll, die Problematik wie folgt zu formalisieren:

Sei Σ die Menge der Ereignismeldungen oder *Ereignisse*. Ein *Ereignis-Matching-Template* E ist ein Ausdruck, der sich zu einer Menge von Ereignissen expandieren lässt, d.h. $Expand(E) \in \mathcal{P}(\Sigma)$ mit der Potenzmenge $\mathcal{P}(\Sigma)$ der Ereignismeldungen. Ein Ereignis e stimmt genau dann mit einem Template E überein (notiert als $e \dashv E$), wenn $e \in Expand(E)$ gilt. Die *Matching-Funktion* ergibt sich intuitiver Weise als

$$E : \Sigma \rightarrow I\!B, \quad E(e) = \begin{cases} true, & \text{wenn } e \in Expand(E) \\ false, & \text{wenn } e \notin Expand(E) \end{cases}$$

Sei $P = \{(p_1, \ldots p_m)\}$ eine beliebige Menge zeitvarianter Parameterbelegungen. Eine *bedingte Transformationsvorschrift* ist ein Tripel

$$R = (E^M, P, E^T)$$

mit einer Menge von *Matching-Templates*

$$E^M = \{E_i^M \mid E_i^M : \Sigma \rightarrow I\!B,\ i = 0, \ldots, n\}$$

und einem *bedingten Transformations-Template*

$$E^T : \Sigma \times P \ \rightarrow \ \mathcal{P}(\Sigma)$$

Sei T das betrachtete Zeitintervall, $e(t) \in \Sigma,\ \forall t \in T$ eine Sequenz von Ereignismeldungen und $p(t) \in P$ eine Sequenz von Parameterbelegungen. Die *bedingte Transformationsfunktion* der Vorschrift R ist

$$f_R : \Sigma \times T \rightarrow \mathcal{P}(\Sigma) \text{ mit}$$

$$f_R(e(t), t) = \begin{cases} E^T(e(t), p(t)), & \text{wenn } e(t) \dashv E_i^M,\ \forall i = 0, \ldots, n \\ \{e(t)\}, & \text{sonst} \end{cases}$$

d. h. abhängig von der Übereinstimmung der Templates E_i^M mit dem Eingabe-Ereignis $e(t)$ werden Null oder mehr Ausgabe-Ereignisse erzeugt. Man beachte, dass diese Transformation zustandslos ist, d.h. keine Ereignisse der Eingabe im Filter gespeichert werden.

Betrachtet man den Anwendungskontext der Informationsbereinigungs-Gateways an den Domänengrenzen, so muss zunächst aus der lokalen IShP eine Menge von bedingten Transformationsregeln der Form $R = (\{E_0^M\}, P, E^T)$ generiert werden, wobei wir im Folgenden o. B. d. A. von $P = \emptyset$ ausgehen. Unglücklicherweise kann ein sinnvoller Informationsbereinigungsprozess nicht ausschliesslich auf einer statischen Textsubstitution basieren. Wenn beispielsweise IP-Adressen als Bestandteil von Meldungen durch feste Werte ersetzt werden sollen, gehen alle Informationen über die Topologie des Netzwerks verloren. Offensichtlich behindert dies den Erkennungsprozess, insbesondere, wenn verkehrsbezogene Anomalien entdeckt werden sollen. Daher benötigt man eine flexiblere Methode, um Transformationsregeln zu spezifizieren: *Submatching-Referenzen*.

Sei $s(E_0^M)$ eine Menge von qualifizierten Teilausdrücken im Matching-Template E_0^M, und sei $s(E_0^M, e(t))$ die Menge von in $e(t)$, die den Teilausdrücken in $s(E_0^M)$ entsprechen. Wenn man nun das Transformations-Template zu

$$E^T : \Sigma \times s(E_0^M) \times P \ \rightarrow \ \mathcal{P}(\Sigma)$$

erweitert, erhält man die Möglichkeit, neue Meldungen zu konstruieren, die auf den übereinstimmenden Teilausdrücken des alten Ereignisses basieren. Dies ist offensichtlich eine Grundvoraussetzung für die Anwendung als Verfahren zur Informationsbereinigung.

Man beachte, dass es – nur durch Festlegung entsprechender Regeln – mit diesem Verfahren ebenfalls möglich ist, eine *Normalisierung* von Ereignismeldungen zu realisieren (z. B. wenn dasselbe Ereignis mit unterschiedlichen Signaturbezeichnungen oder Referenzen auf Angriffsdatenbanken gemeldet wird). Um die Möglichkeiten der Transformationsregeln noch erweitern zu können, ist es offensichtlich sinnvoll, Systemparameter – wie den Zeitpunkt des letzten Matchings, die aktuelle Systemzeit oder die IP-Adresse des GWs – als Parametermenge P einzubeziehen.

4 Module für die Verarbeitung von Meldungskombinationen

In unserem Ansatz gibt es zwei Verarbeitungsmodule, die den Umgang mit hohen Meldungsraten durchführbar machen sollen: *Redundanzfilter* und das Anwenden von vordefinierten Erkennungsregeln für miteinander korrelierte Elementarereignisse (*Kombinati-*

onsdetektor). Beide Aufgabenstellungen sind Spezialfälle der sog. *Event Corellation* und bereits mehrfach untersucht worden.

Im Laufe unserer Forschungsarbeiten hat sich jedoch herausgestellt, dass man zur Realisierung ein Verfahren einsetzen kann, das die in Abschnitt 3 beschriebenen Methode zur Informationsbereinigung erweitert und ein sehr universelles und mächtiges Werkzeug zur Verfügung stellt.

4.1 Filterung von Redundanzen

Um die Anzahl von Ereignismeldungen reduzieren zu können, ist es offensichtlich notwendig, Redundanzen zu vermeiden. Deshalb ist es oft sinnvoll, mehrere Meldungen mit einem "ähnlichen" Inhalt zu einer einzelnen, neuen Meldung zusammenzufassen. In [DW01] beispielsweise wurde diese einfache Form der Zusammengehörigkeit durch die Definition von sog. Duplikaten modelliert.

Es ist möglich, das im Abschnitt 3 beschriebene CPMT-Verfahren derart zu erweitern, dass es zur diesbezüglichen Filterung des Meldungsflusses herangezogen werden kann. Die Matching-Templates E^M müssen durch entsprechende Erweiterungen auch die Ähnlichkeitseigenschaft von Meldungen spezifizieren können. Da man nicht nur absolute (initiale) Matchings benötigt, sondern auch *relative Matchings* (d.h. abhängig von vorhergehenden, unter Festlegung der Differenz), muss man den Matchingvorgang erweitern. Man führt ein zweites Matching-Template E_1^M ein, das auf Submatchings des initialen Matchings E_0^M verweist. Für die Menge der Matching-Templates $E^M = \{E_0^M, E_1^M\}$, erhält man

$$E_0^M : \Sigma \rightarrow I\!B, \qquad E_1^M : \Sigma \times s(E_0^M) \rightarrow I\!B$$

Beispiel: Es sollen alle Meldungen mit identischer Ereignisklassifikation und Quell-/Ziel–IP-Adresse gefiltert werden. Gleichzeitig soll der Entstehungszeitpunkt der Meldungen innerhalb einer Sekunde liegen. So muss

- E_0^M einen geeigneten Teilausdruck für den Entstehungszeitpunkt der Meldung und für die Quell-/Ziel–Adresse besitzen,
- E_1^M einen bedingten Ausdruck für die Quell-/Ziel–Adresse haben, der identisch zu den entsprechenden Werten des initialen Matchings $e(t) \dashv E_0^M$ ist und
- E_1^M einen bedingten Ausdruck für den Wert des Entstehungszeitpunktes beinhalten, der nicht um mehr als eine Sekunde vom Entstehungszeitpunkt des initialen Matchings entfernt ist.

Im Gegensatz zu (zustandslosen) bedingten Transformationen, bei denen ein absolutes Matching für alle Templates E_i^M erforderlich ist, benötigt man zwei Speicherplätze (*Buckets*)

$$B_i = \{e | e \vdash E_i^M\},\ i = 0, 1$$

mit $0 \leq |B_0(t)| \leq 1, |B_1(t)| \geq 0\ \forall t \in T$ für bereits mit den Templates übereinstimmende Ereignisse. Zusätzlich benötigt man den Speicherzeitpunkt t_{B_0} des ersten Ereignisses,

welches im Bucket B_0 abgelegt wurde, um das Ende der Zusammenfassungphase nach der maximalen Speicherzeit Δt_{B_0} feststellen zu können. Die Transformationsfunktion hängt nicht vom aktuellen Ereignis $e(t)$, sondern von den vorher gespeicherten Ereignissen ab, d.h. wir definieren ein Transformationstemplate

$$E^T : \mathcal{P}(\Sigma) \times \mathcal{P}(\Sigma) \times s(E_0^M) \times P \rightarrow \mathcal{P}(\Sigma)$$

sowie für die Transformationsfunktion f_R einen Algorithmus für den Redundanzfilterprozess, der nicht auf externe Parameter zurückgreift und sich daher wie folgt gestaltet:

Eingabe: $R = (\{E_0^M, E_1^M\}, \emptyset, E^T)$, Δt_{B_0} und eine Sequenz von Ereignissen $e(t)$.
Ausgabe: Eine Sequenz von Ereignismengen $\{e\}$, in der ähnliche Ereignisse zusammengefasst sind.
Algorithmus:

```
(1)   B_0(0) := ∅ ; B_1(0) := ∅ ; t_{B_0} := 0
(2)   t := 1
(3)   while true do
(4)       read e(t)
(5)       if t_{B_0} ≠ 0 and t − t_{B_0} ≥ Δt_{B_0} then
(6)           output E^T(B_0(t), B_1(t), s(E_0^M))
(7)           B_0(t) := ∅ ; B_1(t) := ∅
(8)           t_{B_0} := 0
(9)       fi
(10)      if e(t) ⊀ E_0^M and e(t) ⊀ E_1^M then
(11)          output {e(t)}
(12)      fi
(13)      else if |B_0| = 0 and e(t) ⊣ E_0^M then
(14)          B_0(t) := {e(t)}
(15)          t_{B_0} := t
(16)      fi
(17)      else if |B_0| > 0 and e(t) ⊣ E_1^M then
(18)          B_1(t) := B_1(t − 1) ∪ {e(t)}
(19)      fi
(20)      t := t + 1
(21)  done
```

Man beachte, dass dieser Algorithmus nicht voraussetzt, dass “ähnliche” Ereignisse sequentiell und ohne andere dazwischenliegende Meldungen eintreffen. Zusätzlich ist es möglich, mehrere Buckets $B_{ij}(t)$ parallel zu verwenden. Die Anzahl dieser Buckets muss aber begrenzt sein, um Überlastsituationen und DoS-Angriffe zu vermeiden. Weiterhin ist es möglich, anstatt $e' = E^T(B_0, B_1, s(E_0^M))$ erst nach Beendigung der Zusammenfassungsphase zu berechnen (Algorithmus Zeile `(6)`), nach jedem Teil-Matching das spätere Ausgabeereignis e' entsprechend anzupassen. Da in diesem Falle nur ein Bucket für ein Ereignis benötigt wird, reduziert sich der Speicheraufwand erheblich.

4.2 Kombinationsdetektoren

Im Gegensatz zur Redundanzeigenschaft definieren wir *Ereigniskombinationen* als Mengen von miteinander korrelierten Ereignismeldungen, die – separat betrachtet – keine besondere Aufmerksamkeit erfordern, in ihrer Gesamtheit jedoch eine möglicherweise wesentliche höhere Kritikalität des Systemzustandes implizieren. Die Anwendung besteht also nicht primär in der Substitution, sondern in der zusätzlichen Meldung derartiger kritischer Kombinationen. In der Literatur finden sich verschiedene Beispiele für diesen komplexeren Fall der "Zusammengehörigkeit". Das in [Jul02] beschriebene Clustering von Meldungen beschreibt eine Zuordnung mehrerer Meldungen zu einer einzigen, die als Ursache der gemeldeten Ereignisse betrachtet werden kann (*Root Cause*). In [CM02] werden Ereigniskombinationen zusätzlich mit externen Bedingungen bezüglich der Konfiguration der betrachteten Netze verknüpft. Auch die dezentralisierte Verarbeitung der Korrelationsregeln durch das Versenden von Nachrichten an verteilte Verarbeitungsknoten wurde bereits untersucht [KTK02].

Auch die Kombinationserkennung läßt sich in Form eines CPMT-Problems formulieren. Offensichtlich handelt es sich um einen verallgemeinerten Fall der Redundanzfilter, die im vorhergehenden Abschnitt beschrieben wurden. Statt genau eines absoluten und eines relativen Matchings spezifiziert man eine Menge von relativen Matching-Templates $E^M = \{E_1^M, \ldots, E_n^M\}$, um eine Ereigniskombination beschreiben zu können. Jedes Matching kann sich dabei nicht nur auf Submatchings im initialen Template E_0^M beziehen, sondern auch auf Submatchings in den übrigen Templates

$$E_i^M : \Sigma \times \prod_{j=0, j\neq i}^{n} s(E_j^M) \rightarrow I\!B, \; \forall i = 1, \ldots, n$$

Unter Zuhilfenahme mehrerer Buckets $B_i, 0 \leq i < n$ ist es möglich, auf alle in den Buckets gespeicherten Ereignisse zu verweisen, indem E^T erweitert wird zu

$$E^T : \prod_{i=0}^{n} \mathcal{P}(\Sigma) \times \prod_{j=0}^{n} s(E_j^M) \times P \; \rightarrow \; \mathcal{P}(\Sigma)$$

Mit diesem Ansatz können wir verschiedene Korrelationsbeziehungen modellieren, die für die Erkennung sicherheitsrelevanter Vorkommnisse von Bedeutung sind. So lassen sich beispielsweise folgende Korrelationseigenschaften zwischen zwei Matching-Templates E_0^M und E_1^M ausdrücken (vgl. [Cup01]):

- *Zusammenhänge der Meldungs-Klassifikation*:
  Definiere Ausdrücke in E_0^M und E_1^M, die eine Teilmenge von möglichen Ereignisklassifikationen (z.B. Aufzählungen oder Bereiche von IDs aus einer Angriffsdatenbank) beschreiben.
- *Zeitliche Zusammenhänge*:
  Definiere einen Teilausdruck in E_0^M, der den Zeitwert beinhaltet, und einen arithmetischen Ausdruck in E_1^M, der den Zusammenhang spezifiziert (z.B. eine maximale Zeitdifferenz der Detektion).

- *Räumliche Zusammenhänge*:
  Definiere einen Teilausdruck in E_0^M, der den Adresswert beinhaltet, und einen arithmetischen Ausdruck in E_1^M, der den Zusammenhang spezifiziert (z.B. Zugehörigkeit zum selben Subnetz).

Allerdings ist der bei einer Implementierung der zu erwartende Speicheraufwand im Falle einer *Online*-Verarbeitung (d. h. ohne Datenbankzugriff, einzig auf die in der Verarbeitungspipeline befindlichen Meldungen zurückgreifend) gewaltig. Der Grund besteht darin, dass alle betreffenden Ereignisse im Speicher vorgehalten werden müssen, da während der Aggregation noch nicht gesichert ist, dass alle Bedingungen für die Kombination erfüllt werden.

5 Bisherige Ergebnisse

Die hier beschriebenen Kompontenten wurden zum überwiegenden Teil bereits prototypisch implementiert und ihre Funktionsfähigkeit verifiziert.

5.1 Architektur

Zum Nachweis der Funktionsfähigkeit der verschiedenen Komponenten wurde eine *Meta-IDS-Architektur* implementiert, die die Anforderungen dynamischer Koalitionsumgebungen eher erfüllt als der Ansatz kooperierender Einzelsysteme (vgl. [LJT+04], [JTBH04]). Die Architektur besteht im wesentlichen aus zentralen Einheiten für die Datenanalyse und Steuerung des Gesamtsystems (*Management-Konsolen*) und Aufschaltpunkten an den Domänengrenzen (*IDS-Gateways, GWs*), die einerseits die anfallenden Ereignismeldungen an die Konsolen vorverarbeiten und weiterleiten sowie andererseits die etwaigen Warnmeldungen über potentiell sicherheitskritische Vorgänge an die Domänen übermitteln. Diese Komponenten und ihre Funktionseinheiten sind in Abb. 1 dargestellt. Die Implementation erfolgte auf Basis generischer IDS-Infrastrukturkomponenten, die bei früheren Arbeiten entwickelt worden sind [Jah02].

Das gemeinsame Datenmodell der Ereignismeldungen entspricht dem Intrusion Detection Message Exchange Format (*IDMEF*, [CD03]). Meldungen werden mittels des Intrusion Detection Exchange Protocol (*IDXP*, [FMW02]) transparent übertragen. Um einen gewissen Schutz gegen Denial-of-Service-Angriffe zu gewährleisten, werden alle in [Jah03] geschilderten Maßnahmen angewendet, so z.B. die Überwachung der Gateway-Prozessumgebungen sowie die redundante Auslegung der Management-Konsolen für den Fall einer Netzverbindungs- oder Prozessterminierung.

Während die zentralen Komponenten von einer geeigneten, für alle Beiteiligten akzeptierbaren Autorität betrieben werden, stehen die Gateways unter der Kontrolle des lokalen Sicherheitspersonals einer Domäne. Es hat u. a. die jeweiligen Informationsweitergaberichtlinien in Form einer geeigneten Konfiguration des Gateways durchzusetzen und

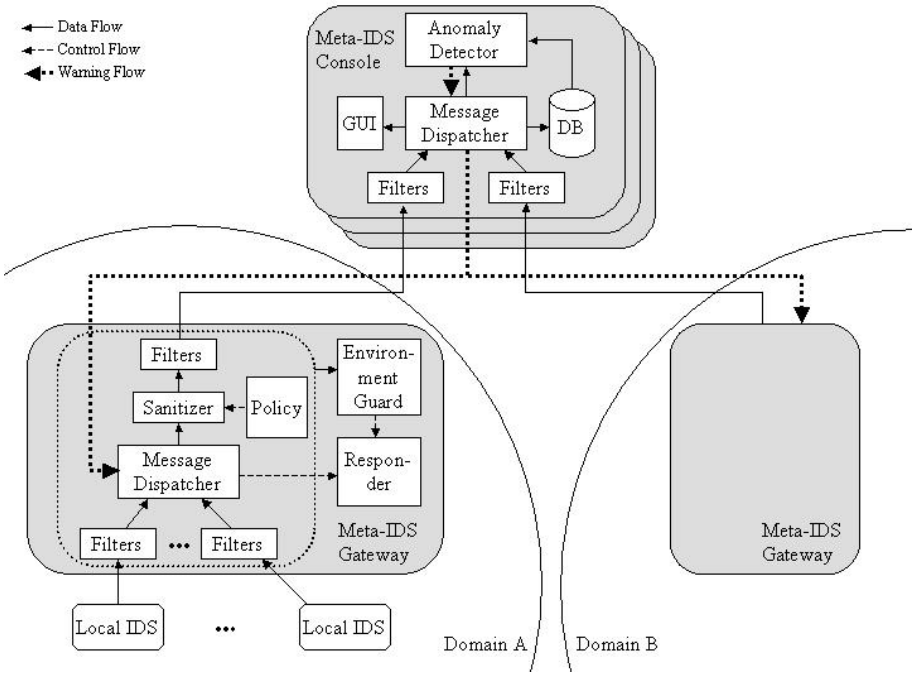


Abbildung 1: Architekturkomponenten des prototypischen Meta-IDS.

gleichzeitig darauf zu achten, dass Randbedingungen für die Art und den Grad der Anonymisierung der Meldungen eingehalten werden. Diese Randbedingungen ergeben sich aus der Art der Detektionsstrategie und müssen in weitergegenden Arbeiten noch ermittelt werden.

Damit eine Domäne nicht etwa durch geschicktes Abfragen des anonymisierten Meldungsbestandes der zentralen Komponenten verborgene Information wiederherstellen kann, ist dieser Meldungsbestand entsprechend unter Verschluss zu halten. Sind in Warnmeldungen, die an alle Domänen weitergegeben werden sollen, domänenspezifische Angaben enthalten, so sollte die Weitergabe (u. U. manuell) autorisiert werden. Anderenfalls könnte eine Domain durch Provozieren einer Anomalie dem System gezielt Informationen über eine andere Domäne entlockt werden.

5.2 Anomalieerkennung

Zur Zeit wird das Anomalieerkennungssystem in die bestehende IDS-Infrastruktur integriert. In einer ersten Erprobungsstufe findet eine Beschränkung auf den ursprünglich für die Verkehrsanomalie-Erkennung entwickelten Ansatz statt, d.h. es werden zur Erkennung

typischer Ereignismeldungsstrukturen ausschließlich Source-Target-Beziehungen für den Aufbau der Graphen verwendet. Die Portierung der bisher verwendeten Algorithmen geschieht aber schon mit Rücksicht auf die eingeplanten Erweiterungen des Verfahrens.

Zur Bewertung des Systems wird auf operationelle Datenquellen aus einer bestehenden DMZ zurückgegriffen, die gezielt und reproduzierbar mit künstlich generierten Ereignismeldungen gemischt werden können. Diese künstlichen Meldungen werden beispielsweise aus Paketfilter-Logdaten gewonnen, die zuvor von einem "Wurm-Simulator" (vgl. [Sch02]), dem Ausbreitungsverhalten realer oder fiktiver internet-Würmer entsprechend, für die jeweilige Netzkonfiguration ermittelt wurden. Erste Ergebnisse haben gezeigt, dass nicht nur die Ausbreitung von Würmern, sondern auch diverse gezielte (reale) Angriffe gegen einzelne Knoten zuverlässig erkannt werden können.

5.3 Informationsbereinigung bei Ereignismeldungen

Die in den Abschnitten 3 und 4 beschriebenen Ansätze der Informationsbereinigung (einschließlich einer Normalisierung), der Redundanzfilterung und der Erkennung von Ereigniskombinationen können auf die bedingte Ereignismeldungs-Transformation reduziert werden. Ein häufig gewählter Ansatz, um komplexe Transformationen von XML-formatierten Daten durchzuführen, ist die Anwendung von XSLT-Stylesheets [W3C99]. Die bekanntermaßen nicht besonders hohe Performanz von XSL steht einer großen Flexibilität und universellen Einsetzbarkeit entgegen. XSL erlaubt es, ohne Modifikation des Programmcodes – selbst bei Änderungen der IDMEF-Spezifikation – bei der Regelbearbeitung auf beliebige Bestandteile der zu verarbeitenden Meldungen zuzugreifen, die über die üblichen Zeitangaben, Klassifikationen, Adressen und Portnummern hinausgehen. Die Verwendung marktverfügbarer Werkzeuge bei der Definition, Umsetzung und Wartung von Sicherheits- und Informationsweitergabe-Richtlinien ermöglicht zudem einen kostensparenden Einsatz.

Unglücklicherweise beinhalten XSLT-Stylesheets keine Regulären Ausdrücke (REs), und zum Implementierungszeitpunkt waren auch keine geeigneten Erweiterungen verfügbar. Deshalb haben wir für einen ersten Meta-IDS-Prototypen einen XML-Prozessor implementiert, der XSLT-Sheets um POSIX.1-REs sowie um zusätzliche boolesche und arithmetische Ausdrücke bereichert. Er wendet das beschriebene Matching- und Transformationsverfahren auf IDMEF-Meldungen an.

In dem folgenden Beispiel wird ein kombiniertes Matching- und Transformations-Template (sog. *Transformations-Sheet*) als Teil einer modifizierten IDMEF-Nachrichtensyntax spezifiziert:

```
(1)  <IDMEF-Message xmlns:xsl=...>
       <Alert>
       ...
(2)      <address>
(3)        192\.22\.([0-9]{1,3})$X$\.([0-9]{1,3})$Y$
(4)        <condition>
(5)          ($Y<255)
```

```
(6)         </condition>
(7)         <transform>
(8)           <xsl:copy>
(9)            191.72
(10)               .<xsl:value-of select="$X"/>
(11)               .<xsl:value-of select="$Y"/>
(12)          </xsl:copy>
(13)        </transform>
(14)      </address>
        ...
(15)    </Alert>
(16) </IDMEF-Message>
```

Die Matchings der letzten beiden Bytes einer IP-Adresse `192.22.x.y` werden als Variablen `$X` und `$Y` referenziert (Zeile `(3)`). Und bei der Transformation entsprechend eingesetzt (Zeilen `(9)`-`(11)`). Die zusätzliche Bedingung dafür, dass die Transformation durchgeführt wird, ist, dass `$Y` nicht dem traditionellen Broadcastsuffix 255 entsprechen darf (Zeilen `(4)`-`(6)`).

Mit diesen Transformationstechniken wurden die Informationsbereinigungs-Funktionen der Gateways auf Basis der GNOME libxml2 und libxslt in Form von C++-Filterklassen im Rahmen des SNAF/SIDI-Frameworks (vgl. [Jah02]) implementiert. Um die Effizienz der Implementierung zu messen, wurde zunächst eine Anzahl von Transformations-Sheets erstellt, die beispielsweise IP-Adressen und Adressbestandteile, Hostnamen, Zeitangaben sowie Produktangaben über Sicherheitswerkzeuge anonymisieren. Unter Verwendung eines PIII/1GHz-PCs mit 256MB RAM und Debian GNU/Linux bewältigte ein Gateway für die Durchführung von 10 sequentiellen Transformationsschritten zwischen 19 und 23 Meldungen pro Sekunde (ohne Codeoptimierung), was zumindest für die prototypische Anwendung ein ausreichendes Ergebnis darstellt.

5.4 Redundanzfilter

Auch die in Abschnitt 4 beschriebene Redundanzfilterung wurde auf Basis von XSLT-Stylesheets implementiert. Ein Beispiel für die zu diesem Zweck abermals erweiterte IDMEF-Syntax zur Spezifikation ist folgendes:

```
        ...
(1)       <Source>
(2)        <Node>
(3)          <Address>
(4)            <address>
(5)              ([0-9]{1,3}\.[0-9]{1,3}\.5\.1)$SA$
(6)            </address>
(7)          </Address>
(8)        </Node>
(9)         <Service>
(10)         <protocol>(.*)$SP$</protocol>
(11)       </Service>
(12)     </Source>
```

```
(13)   <relMatch deltaT="20000" maxMatch="40">
(14)     <xsl:copy>
(15)       <Source>
(16)         <Service>
(17)          <protocol><xsl:value-of select="$SP"/></protocol>
(18)         </Service>
(19)       </Source>
(20)       <Target>
(21)         <Node>
(22)           <Address>
(23)             <address>(.*)$TA$</address>
(24)          </Address>
(25)         </Node>
(26)         <Service>
(27)            <port>(.*)$TP$</port>
(28)         </Service>
(29)       </Target>

(30)       <summary do="create">
(31)         <AdditionalData>
(32)           matched: sourceaddress=$SA sourceprotocol=$SP
(33)         </AdditionalData>
(34)       </summary>

(35)       <summary do="update">
(36)         <AdditionalData>
(37)           target: address=$TA port=$TP
(38)         </AdditionalData>
(39)       </summary>
      ...
```

In diesem Beispiel wird E_0^M durch die Zeilen `(1)-(12)` und E_1^M durch die Zeilen `(13)-(29)` spezifiziert. Die resultierende Nachricht wird erzeugt, wenn das absolute Matching E_0^M zutrifft und wird durch E^T (Zeilen `(30)-(34)`) beschrieben. Bei jedem darauf folgenden Eintreffen einer Nachricht, die mit E_1^M übereinstimmt, wird dieses Resultat modifiziert, wie in den Zeilen `(35)-(39)` beschrieben.

Auch die Implementierung der Redundanzfilterung wurde bezüglich ihrer Leistungsfähigkeit auf obengenannter Hardware getestet; sie wurde in Kombination mit dem Informationsbereinigungsmodul betrachtet. Erwarteter Weise stellten sich bei den in der Abbildung 2 skizzierten Messungen deutlich höhere Durchsatzraten für den Fall heraus, in dem die Informationsbereinigung erst nach der Redundanzfilterung durchgeführt wurde, anstatt sie vorzuschalten (gestrichelte im Ggs. zur durchgezogenen Linie).

Während dieser Tests bestätigte sich ausserdem, dass das Zusammenfassen von Eigenschaften mehrerer Templates – sofern anwendbar – zu einer signifikanten Verbesserung der Leistungsfähigkeit führt (in der Abbildung Einzelpunkte im Ggs. zu Linien). Dies ist durch die Tatsache begründet, dass Meldungsbestandteile nicht mehrfach auf Übereinstimmungen geprüft werden müssen. Allerdings gestaltet sich eine derartige Zusammenfassung nichttrivial und wird schnell unübersichtlich.

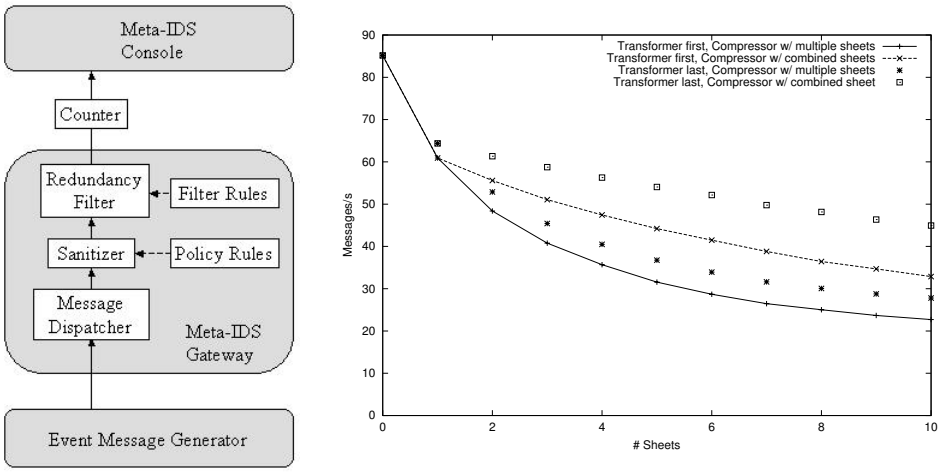


Abbildung 2: Szenario und Ergebnisse der Durchsatzmessungen für die kombinierte Redundanzfilterung mit Informationsbereinigung.

6 Zusammenfassung und Ausblick

In diesem Beitrag wurden verschiedene Verfahren vorgestellt, die sich zur Realisierung von Verarbeitungskomponenten kooperativer Intrusion-Detection-Syteme in dynamischen Koalitionsumgebungen eignen. Hier stellten sich insbesondere die Heterogenität der lokalen Sicherheitsrichtlinien, der zur Durchsetzung verwendeten Sicherheitswerkzeuge (IDS, Paketfilter, Integritätsprüfprogramme etc.), sowie der Richtlinien zur Informationsweitergabe als wesentliche Herausforderungen für die Realisierung eines koalitionsweiten Systems zur frühzeitigen Warnung vor großflächigen sicherheitsrelevanten Aktivitäten in Computernetzen heraus. Die vorgestellten Komponenten bestehen zum Einen aus einem Anomalieerkennungsmodul, das ein abnormales Aufkommen von Ereignismeldungen aus Abweichungen von einem zuvor erlernten Normalprofil erkennen kann, und zum Anderen aus Modulen zur Datenreduktion und zur Erkennung zuvor spezifizierter Kombinationen von Ereignismeldungen.

Die Erkennung von Anomalien erfolgt mittels etablierter Techniken, die auf der Bildung von Clustern in Quelle-Ziel-Graphen beruhen. Diese Graphen werden anhand der eintreffenden Ereignismeldungen gebildet und eine Abweichung eines speziellen Distanzmaßes zwischen dem aktuellen und dem Normalprofil wird als Anomalie gewertet. Alle Aufgaben, die die Verarbeitung von Ereignismeldungen betreffen, wurden auf bedingtes Pattern-Matching und Transformation zurückgeführt, das in Form eines erweiterten XSLT-Stylesheet-Prozessors realisiert wurde. Die beschriebenen Komponenten wurden zum Teil in einem prototypischen Meta-IDS realisiert und auf ihre Leistungsfähigkeit hin untersucht.

Das Verfahren der Anomalieerkennung wird derzeit ausgewertet, verfeinert und in das Gesamtsystem integriert. Seine Grenzen bei wachsender Modifikation der Ereignismeldungen durch die Informationsbereinigung sowie bei stark dynamischem Verhalten der Koalitionsumgebung werden untersucht. Für die Zukunft sind verschiedene Erweiterungen des Systems – insbesondere hinsichtlich der Extraktion relevanter Informationen, die als ergänzende Angaben in Anomaliewarnungen dienen – angedacht. Aspekte der Skalierbarkeit der Meldungsverarbeitung durch Parallelisierung werden be-

trachtet. Zudem wird an einem kontinuierlich laufenden Testszenario mit realen Ereignismeldungen aus verschiedenen entmilitarisierten Zonen (DMZ) von Forschungsnetzwerken gearbeitet.

Literatur

[CD03] D. Curry and H. Debar. Intrusion Detection Message Exchange Format – Data Model and Extensible Markup Language (XML) Document Type Definition. IETF Internet Draft draft-ietf-idwg-idmef-xml-10.txt, January 2003. IETF IDWG.

[CM02] F. Cuppens and A. Miège. Alert Correlation in a Cooperative Intrusion Detection Framework. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 202–215, Oakland, CA, May 2002. IEEE Computer Society, Technical Committee on Security and Privacy, IEEE Computer Society Press.

[Cup01] F. Cuppens. Managing Alerts in a Multi-Intrusion Detection Environment. In *Proceedings of the 17th Annual Conference on Computer Security Applications (ACSAC'01)*, New Orleans, LA, December 2001. ACM.

[DW01] H. Debar and Andreas Wespi. Aggregation and Correlation of Intrusion-Detection Alerts. *Lecture Notes in Computer Science*, 2212:85ff., 2001.

[Fle02a] U. Flegel. Anonyme Audit-Daten im Überblick. *DuD – Datenschutz und Datensicherheit*, 1(26), 2002.

[Fle02b] U. Flegel. Pseudononyzing Unix Log Files. In *Proceedings of the Infrastructure Security Conference (InfraSec2002), Bristol, UK*, October 2002.

[FMW02] B. Feinstein, G. Matthews, and J. White. The Intrusion Detection Exchange Protocol. IETF Internet Draft draft-ietf-idwg-beep-idxp-07.txt, October 2002. IETF IDWG.

[Jah02] M. Jahnke. An Open and Secure Infrastructure for Distributed Intrusion Detection Sensors. In *Proceedings of the NATO Regional Conference on Communication and Information Systems (RCMCIS'02), Zegrze, Poland*, October 2002.

[Jah03] M. Jahnke. Schutz verteilter Intrusion-Detection-Systeme gegen Denial-of-Service-Angriffe. In *10. DFN-CERT/PCA-Workshop "Sicherheit in vernetzten Systemen"*, pages J–1–J–12, Hamburg, February 2003.

[JTBH04] M. Jahnke, J. Tölle, M. Bussmann, and S. Henkel. Components for Cooperative Intrusion Detection in Dynamic Coalition Environments. Presented at NATO/RTO IST Symposium "Adaptive Defence in Unclassified Networks", Toulouse, April 2004.

[Jul02] K. Julisch. Mining Alarm Clusters to Improve Alarm Handling Efficiency. In *Proceedings of the NATO/RTO IST Workshop on Inforensics and Incident Response, The Hague, The Netherlands*, October 2002.

[KTK02] C. Krügel, T. Toth, and C. Kerer. Decentralized Event Correlation for Intrusion Detection. *Lecture Notes in Computer Science*, 2288:114, 2002.

[LJT+04] M. Lies, M. Jahnke, J. Tölle, M. Bussmann, and S. Henkel. Ein Intrusion-Warning-System für dynamische Koalitionsumgebungen. In *Proceedings of the 11th DFN-CERT/PCA Workshop "Sicherheit in vernetzten Systemen", Hamburg*, February 2004.

[Sch02] H. Schmidt. Simulation und Erkennung der Ausbreitungsstrukturen von Würmern. Master's thesis, Universität Bonn, Institut für Informatik, 2002.

[TdW02] J. Tölle and C. de Waal. A Simple Traffic Model Using Graph Clustering For Anomaly Detection. Proc. of Applied Simulation and Modelling (ASM) Crete, Greece, June 2002.

[W3C99] W3C. W3C Recommendation 16: XSL Transformations (XSLT) Version 1.0. http://www.w3.org, 1999.

Vertrauensbasierte Laufzeitüberwachung verteilter komponentenstrukturierter E-Commerce-Software

Peter Herrmann, Lars Wiebusch und Heiko Krumm

Peter.Herrmann@udo.edu, lars-wiebusch@web.de, Heiko.Krumm@udo.edu

Abstract: Die Entwicklung komponentenstrukturierter E-Commerce-Software ist kostengünstig und schnell, da man die Systeme recht einfach aus wiederverwendbaren Softwarekomponenten zusammensetzt. Allerdings führt diese Entwurfsmethode zu einer neuen Art an Problemen für die Datensicherheit dieser Systeme. Insbesondere besteht die Gefahr, dass eine bösartige Komponente die gesamte Anwendung, in die sie eingebunden ist, bedroht. Zur Abwehr dieser Gefahr verwenden wir Security Wrapper, die das Verhalten von Komponenten zur Laufzeit überwachen und die Sicherheitsanforderungen der Anwendung durchsetzen. Ein Security Wrapper beobachtet das Verhalten an der Schnittstelle einer Komponenten und vergleicht es mit den vom Komponentenentwickler garantierten Sicherheitspolicies, die in der Komponentenspezifikation formal beschrieben werden. Wir stellen vor, wie man die Sicherheitspolicies zustandsbasiert beschreibt und führen eine Sammlung an Spezifikationsmustern ein, aus denen man die Modelle der Sicherheitspolicies für eine Komponente ableitet. Schließlich zeigen wir den Einsatz der Security Wrapper anhand eines E-Procurement-Beispiels. Darüberhinaus erläutern wir, wie man unter Berücksichtigung der Erfahrungen anderer Nutzer mit einer Komponente den Aufwand der Laufzeittests reduzieren kann. Dazu verwenden wir einen speziellen Vertrauensmanagement-Service, der gute und schlechte Erfahrungen unterschiedlicher Benutzer mit Komponenten verwaltet. Abhängig von diesen Erfahrungsberichten können die Security Wrapper das Ausmaß der Überwachung absenken, indem sie anstatt einer vollständigen Überwachung nur Stichproben durchführen oder die Überwachung sogar abbrechen.

1 Einleitung

Komponentenbasierte Systeme erfreuen sich einer immer größeren Beliebtheit, da sie mit geringerem Kosten- und Entwicklungsaufwand als monolithische Anwendungen entwickelt werden können. Die Softwarekomponenten, die jeweils nur eine Teilfunktion eines E-Commerce-Systems realisieren, werden separat vertrieben. Der Betreiber einer Anwendung wählt geeignete Komponenten aus und koppelt sie miteinander zu einem auf seine speziellen Bedürfnisse zugeschnittenen System (vgl. [Sz97]). Eine Komponente wird dabei entweder lokal in die Applikation eingebunden oder sie läuft auf einem fremden Wirtsrechner, von dem aus sie mittels eines Telekommunikationsdienstes mit der Anwendung verbunden wird.

Allerdings ist die Architektur komponentenstrukturierter Systeme, die unter Umständen

aus heterogenen Bausteinen bestehen, komplexer als diejenige klassischer objektorientierter Applikationen, wodurch neuartige Probleme auftreten. Insbesondere in Hinsicht auf die Datensicherheit entstehen neue Herausforderungen, da man neben Systembetreibern und -nutzern auch die Komponentenentwickler, Wirtsrechnerbetreiber und gegebenenfalls die Anwendungsersteller als beteiligte Prinzipale berücksichtigen muss. Zum einen hat jeder Prinzipal eigene Sicherheitsziele, die eingehalten werden müssen. Zum anderen ist jeder Prinzipal aber auch eine potentielle Bedrohung für das System und damit für die anderen Beteiligten. Die größte Gefahr dabei ist, dass eine bösartige Komponente das gesamte System, in das sie eingebunden ist, unsicher macht. Um den Einsatz sicherer komponentenbasierter Systeme zu ermöglichen, benötigt man somit ein Verfahren, das die Sicherheit einer Anwendung auch in dem Fall garantiert, dass ihre Komponenten von nicht-vertrauenswürdigen Quellen erworben wurden.

Unser Ansatz zur Lösung der Sicherheitsprobleme setzt auf die Komponentenkontrakte auf, die die Schnittstelleneigenschaften einer Komponente explizit und im Idealfall rechtsverbindlich beschreiben. Nach Beugnard et al. [BJPW99] besteht ein Kontrakt aus vier Teilen, in denen die Schnittstellenstruktur (d.h., die Methoden bzw. Ereignisse mit ihren Eingabe- und Ausgabeparametern sowie Ausnahmebehandlung), das erwartete Verhalten der Komponenten und ihrer Umgebung, Synchronisationsaspekte sowie quantitative Dienstgüteeigenschaften festgelegt sind. Wir ergänzen die Verhaltensbeschreibung eines Kontrakt um Sicherheitspolicies, die die Sicherheitsziele des Anwenders während der Komponentennutzung gewährleisten sollen. Die Policies werden durch Verhaltensconstraints formal dargestellt. Dabei gehen wir davon aus, dass man bösartige Komponenten oder von Dritten verfälschter Code (z.B. durch einen Computer-Virus) erkennen kann, wenn das reale Schnittstellenverhalten von den Kontraktbeschreibungen abweicht. Unser Ansatz besteht aus zwei Phasen. Zur Entwurfszeit wird untersucht, ob die Kombination der in den Modellen beschriebenen Verhaltensweisen die von der gesamten Anwendung geforderten Sicherheitsziele erfüllt. Zur Laufzeit wird für alle relevanten Komponenten überprüft, ob ihr tatsächliches Verhalten mit den Verhaltensconstraints übereinstimmt. Ein Verhaltensconstraint wird im Kontrakt durch ein Zustandstransitionssystem modelliert und in der temporalen Logik cTLA [HK00] spezifiziert, die eine modulare Beschreibung der einzelnen Eigenschaften in separaten Spezifikationen erlaubt. Wenn man auch die Sicherheitsziele der Anwendung in cTLA beschreibt, kann man durch logische Deduktion formal verifizieren, dass die Ziele von den Komponenten erfüllt werden (vgl. [He03a]).

Dieses Papier konzentriert sich auf die Laufzeitüberprüfungen von Komponenten, die durch so genannte Security Wrapper [HK01] erfolgt. Ein Security Wrapper wird an die Schnittstelle zwischen der Komponente und ihrer Umgebung eingefügt. Er blockiert die Ereignisse an der Schnittstelle und überprüft sie auf Übereinstimmung mit den cTLA-Verhaltensbeschreibungen. Wenn ein Ereignis übereinstimmt, wird es weitergeleitet. Falls der Wrapper jedoch ein Fehlverhalten entdeckt, blockiert er alle weiteren Schnittstellenereignisse und benachrichtigt den Anwendungsadministrator.

Security Wrapper können zu signifikantem Mehraufwand an Rechnerleistung führen. Deshalb verfolgen wir zusätzlich zu der Lauftzeitüberwachung durch die Wrapper einen komplementären Ansatz [He01, He03b], der explizites Vertrauensmanagement ermöglicht (vgl. [Jø96]) und sich an das Prinzip der Reputation Systems anlehnt (vgl. [RZFK00]).

In den nächsten beiden Abschnitten beschreiben wir kurz die Ansätze des Vertrauensmanagements und der Security Wrapper. Danach wird der Einsatz dieser Methoden anhand einer typischen E-Commerce-Anwendung veranschaulicht. Wir untersuchen dazu ein komponentenorientiertes Warenbeschaffungssystem für Schnellrestaurants, das um Komponenten zur elektronischen Beschaffung (E-Procurement) ergänzt wurde.

2 Vertrauensmanagement

Das zentrale Element dieses Ansatz ist der so genannte Vertrauensinformationsdienst, der gute und schlechte Bewertungen über Sicherheitsaspekte einer Komponente sammelt und die Ergebnisse interessierten Nutzern zur Verfügung stellt. Ein Nutzer kann die Bewertungen in seine Beschaffungsentscheidung einfließen lassen. Darüberhinaus kann er sie auch für die Laufzeitüberwachung heranziehen und die Intensität der Überwachung von ihnen abhängig machen.

Um die vertrauensbasierte Steuerung der Überwachung automatisieren zu können, beschreibt der Vertrauensinformationsdienst die Ergebnisse der Bewertungen in Form von Vertrauenswerten (vgl. [Jø96]). Ein Vertrauenswert ist ein mathematischer Ausdruck, mit dem man zum einen das Vertrauen bzw. Misstrauen in eine Komponente beschreiben kann. Zum anderen kann man mit ihm auch ausdrücken, dass die Anzahl der Bewertungen noch zu gering ist, um Vertrauen zu erlangen. Jøsang hat in [Jø99] ein so genanntes Opinion Triangle zur formalen Beschreibung von Vertrauenswerten definiert (siehe Abbildung 1). Reellwertige Variablen b (belief), d (disbelief) und u (uncertainty) aus dem Intervall zwischen 0 und 1 beschreiben das Vertrauen, das Misstrauen bzw. die Unsicherheit in eine Komponente. Da die Summe der drei Werte immer 1 betragen muss, kann man den aktuellen Vertrauenswert durch einen Punkt im Dreieck darstellen. Dabei zeigt ein hochliegender Punkt an, dass bisher nur wenige Bewertungen vorliegen. Bei einer steigenden Zahl an Ergebnissen liegt der Punkt immer niedriger. Wächst aufgrund vieler positiver Ereignisse das Vertrauen in die Komponente, liegt er eher auf der rechten Seite und bei wachsendem Misstrauen weiter links.

Vertrauenswerte werden aus der Anzahl positiver (Wert p) und negativer (Wert n) Bewertungen mit Hilfe von Metriken errechnet. Jøsang und Knapskog [JK98] empfehlen folgende Umrechnungsformel:

$$b = \frac{p}{p+n+1} \qquad d = \frac{n}{p+n+1} \qquad u = \frac{1}{p+n+1}$$

Hiernach kann eine negative Bewertung durch eine entsprechende Zahl positiver Wertungen ausgeglichen werden, so dass eine relativ tolerante Vertrauensmanagement-Philosophie vertreten wird. Dagegen schlagen Beth et al. [BBK94] eine unversöhnliche Methode vor, bei der eine einzige negative Bewertung das Vertrauen in die Komponente für immer zerstört. Wenn keine negativen Wertungen aufgetreten sind, wächst bei dieser Metrik das Vertrauen mit der Anzahl der gutartigen Bewertungen.

Um unterschiedliche Überwachungspolicies zu unterstützen, berechnet der Vertrauensinformationsdienst für jede Komponente zwei Vertrauenswerte nach den beiden oben vorge-

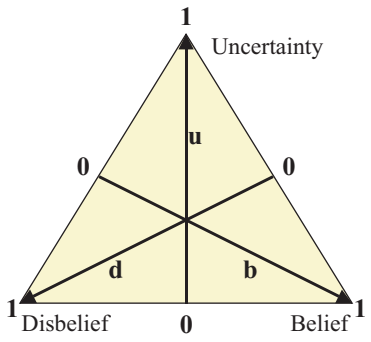


Abbildung 1: Opinion Triangle (entnommen aus [Jø99])

stellten Metriken. Die Struktur dieses Dienstes ist in Abbildung 2 aufgeführt. Er kommuniziert mit Komponentenentwicklern bzw. -vertreibern, mit interessierten Anwendungsbetreibern, die eine Komponente erwerben möchten oder sie zur Laufzeit überwachen, sowie mit Zertifizierungsinstanzen, die eine Komponente im Auftrag des Entwicklers oder eines Betreibers zertifizieren.

Komponentenentwickler können freie oder kommerzielle Komponenten bei dem Vertrauensinformationsdienst registrieren. Sie akzeptieren damit, dass Bewertungen von Nutzern in Form von Vertrauenswerten an Interessenten weitergegeben werden. Kaufinteressenten und Betreiber können jederzeit Vertrauenswerte einer Komponenten abrufen. Sie können darüberhinaus einen Alarm Service buchen, der sie sofort informiert, wenn eine ernste Warnung über die Komponente eingetroffen ist. Um eine hohe Zahl an Ergebnissen zu erhalten, werden alle Nutzer in zeitlichen Abständen um eine Bewertung gebeten. Darüberhinaus sollte ein Nutzer, wenn er eine schwerwiegende Verletzung der Sicherheit durch eine Komponente entdeckt, sofort eine Warnung an den Vertrauensinformationsdienst abgeben, der dann die anderen Nutzer benachrichtigt. Neben den Erfahrungen der Nutzer sind auch die Ergebnisse von Zertifizierungen von großer Bedeutung und fließen in die Vertrauenswertberechnung ein.

Um einen ausreichenden Datenschutz für die Komponentenentwickler zu garantieren, werden die Bewertungen getrennt von den Komponentenbeschreibungen gehalten. Der Verschlüsselungsdienst (Cipher Service) speichert die Daten einer registrierten Komponente und erzeugt einen eindeutigen Schüssel, den der Komponentenentwickler an Interessenten weitergeben kann. Der Vertrauenswertmanager (Trust Value Manager) speichert die Bewertungen und berechnet die Vertrauenswerte referenziert durch den Schlüssel, ohne die Komponenten im Klartext zu kennen. Somit hat weder der Verschlüsselungsdienst noch der Vertrauenswertmanager vollständige Kenntnis über die Bewertung der Komponenten im Klartext. Eine genauere Einführung in die Funktionen des Vertrauensinformationsdienstes wird in [He01, He03b] gegeben.

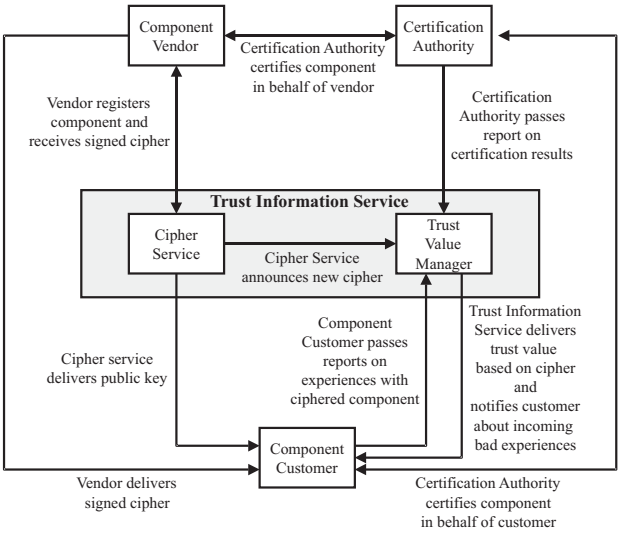


Abbildung 2: Vertrauensinformationsdienst

3 Laufzeitüberwachung

Zur Überprüfung, ob das tatsächliche Systemverhalten einer Komponente ihrer Kontraktspezifikation entspricht, werden so genannte Security Wrapper eingesetzt [HK01]. Dazu koppelt man die Komponente nicht direkt mit ihren Partnerkomponenten, sondern leitet alle ein- und ausgehenden Schnittstellenereignisse über spezielle Komponenten. Dort werden die Ereignisse auf Konformität mit den Sicherheitsspezifikationen im Schnittstellenkontrakt überprüft und nur bei Verhaltenskorrektheit weitergeleitet. Entspricht ein Schnittstellenereignis nicht der Spezifikation, wird der zuständige Applikationsadministrator benachrichtigt. Zudem kann ein Security Wrapper die fehlerhafte Komponente versiegeln, indem bis zu einer anderweitigen Entscheidung des Administrators keine Schnittstellenereignisse mehr weitergeleitet werden.

Ein Security Wrapper für Java Beans-basierte komponentenstrukturierte Systeme wurde in [Ma00] realisiert (siehe Abbildung 3). Für jedes zu überwachende Bean wird ein spezieller Adapter erzeugt, über den alle Schnittstellenereignisse des Beans mit seiner Umgebung laufen. Die Konformitätsüberprüfungen werden durch die so genannten Observer-Objekte durchgeführt, wobei für jede cTLA-Verhaltensspezifikation ein Observer vorgesehen ist. Ein über den Adapter laufendes Ereignis wird zunächst blockiert und die beteiligten Observer informiert. Jeder Observer überprüft durch Simulation seiner Verhaltensspezifikation, ob das Ereignis zulässig ist. Wenn alle Observer das Ereignis akzeptieren, wird es vom Adapter freigegeben und an seinen Empfänger weitergeleitet. Falls jedoch ein Observer eine Verletzung der Spezifikation erkennt, wird das fehlerhafte Ereignis ebenso wie die nachfolgenden Ereignisse vom Adapter blockiert. Ein weiterer Bestandteil des Wrapper ist der Monitor, der die Nutzerschnittstelle des Administrators bildet. Wenn ein Obser-

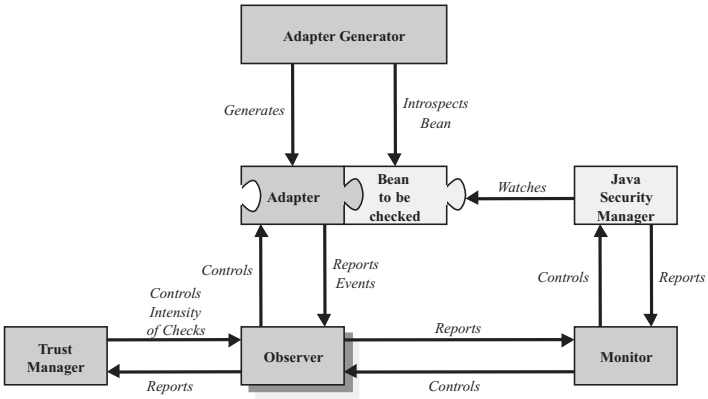


Abbildung 3: Security Wrapper-Architektur

ver eine Kontraktverletzung erkennt, zeigt er sie dem Administrator auf dem Monitor an. Außerdem wird ein so genannter Adapter Generator verwendet, der ein zu überwachendes Bean per Introspektion analysiert und einen geeigneten Adapter erzeugt. Schließlich verwendet das Werkzeug den im Java-Standard enthaltenen Java Security Manager, mit dem der Administrator den Zugriff des Beans auf Systemressourcen einschränken kann, um verdeckte Kanäle auszuschließen, über die das Bean den Adapter umgehen könnte.

Die Verbindung zwischen einem Security Wrapper und dem Vertrauensinformationsdienst wird durch ein Trust Manager-Objekt gebildet (vgl. [He01]). Der Trust Manager erhält periodisch vom Vertrauensinformationsdienst Anfragen über die Erfahrung mit den überwachten Softwarekomponenten. Falls die Observer seit der letzten Anfrage keine Kontraktverletzungen feststellen konnten, wird eine positive Beurteilung zurückgesendet, während Fehlverhalten der Komponente zu einem negativen Urteil führt. Falls der Trust Manager eine Kontraktverletzung als schwerwiegend ansieht, kann er darüberhinaus eine Alarmmitteilung an den Vertrauensinformationsdienst senden, der dann die anderen eingetragenen Nutzer des Beans warnt. Erhält umgekehrt der Trust Manager eine Warnung über ein schwerwiegendes Fehlverhalten einer vom Security Wrapper überwachten Komponente, initiiert er sofort deren Versiegelung, um eine Gefährdung der Applikation zu verhindern.

Schließlich wird der Trust Manager auch zur Überwachungssteuerung eingesetzt. Obwohl Laufzeittests gezeigt haben, dass der Überwachungsaufwand bei der existierenden Implementierung nur zu einem Overhead von 5 % führt (vgl. [HK01]), erscheint eine vertrauensabhängige Steuerung der Überwachungsintensität sinnvoll. Der Trust Manager kann deshalb abhängig vom aktuellen Vertrauenswert, der in zeitlichen Abständen vom Vertrauensinformationsdienst geladen wird, unterschiedliche Überwachungspolicies durchführen. Bei geringer Anzahl an Bewertungen einer Komponenten oder bei einem schlechten Vertrauenswert wird die Komponente vollständig überwacht. Übersteigt der Vertrauenswert eine bestimmte Grenze, überprüft man die Komponente nur noch stichpro-

benhaft. Hierbei erfolgt die Konformitätsüberprüfung nur noch zeitweise [1] und die Länge der Überwachungspausen hängt ebenfalls vom Vertrauenswert ab. Schließlich kann man die Überwachung vollständig einstellen und den entsprechenden Observer löschen, wenn die Komponente als sicher angesehen wird.

4 Muster für Sicherheitspolicies

Eine bösartige Komponente kann die sie einbindende Anwendung auf vielfältige Art angreifen und ihre Vertraulichkeit, Integrität, Verfügbarkeit sowie Nichtabstreitbarkeit verletzen. Eine Gefahr für die Vertraulichkeit liegt in der Änderung von Datenflüssen im System, so dass fehlgeleitete Dateneinheiten von nicht dazu berechtigten Personen gelesen werden. Diese Angriffsart ist insbesondere für verteilte komponentenstrukturierte Systeme relevant, in denen die Komponenten auf verschiedene Rechner mit unterschiedlichen Zugriffspolicies verteilt sind. Eine weitere Angriffsart auf die Vertraulichkeit eines komponentenstrukturierten Systems sind verborgene Kanäle. Dabei wird Information entweder in zwischen Komponenten transportierten Daten versteckt (Steganographie) oder man nutzt die Reihenfolge, Anzahl oder den Ausführungszeitpunkt von Schnittstellenereignissen dazu, geheime Zusatzinformation darzustellen. Bei Angriffen auf die Systemintegrität werden die Funktionalität der Anwendung oder die gespeicherten Daten abgeändert. Hierbei kann man fehlerhafte Schnittstellenereignisse auslösen, Attribute mit falschen Werten belegen oder die Systemkonfigurationsparameter abändern. Die Verfügbarkeit von komponentenbasierter Software kann man durch zwei Angriffsarten gefährden. Zum einen kann ein von einer Komponente angebotene Dienst so häufig aufgerufen werden, dass die Komponente dritte Komponenten nicht mehr bedienen kann (Denial-of-Service Angriffe). Zum anderen kann eine Komponente ihren Partner dadurch blockieren, dass bestimmte geforderte Schnittstellenereignisse nicht ausgelöst werden. Der Partner kann während der Wartezeit nicht mehr dritte Komponenten bedienen. Ein typischer Angriff auf die Nichtabsteitbarkeit von Komponenten ist dann erfolgt, wenn der Komponentenentwickler oder -betreiber in der Lage ist, später erfolgreich abzustreiten, dass die Komponente bestimmte Schnittstellenereignisse ausgelöst oder empfangen hat.

Um Anwendungen vor derartigen Angriffen zu schützen, muss der Betreiber geeignete Schutzziele definieren und Sicherheitspolicies anwenden, die diese Ziele durchsetzen. Mit den Sicherheitspolicies wird das Verhalten einer Komponente in einer Weise eingeschränkt, dass Angriffe entweder unmöglich oder nur unter erheblichen Aufwand durchführbar sind. Um Angriffe auf die Vertraulichkeit zu unterbinden, muss der Datenfluss zwischen Komponenten so eingeschränkt werden, dass Daten nur zu Komponenten gelangen, bei denen ein Zugriff von nicht-authorisierten Lesern ausgeschlossen ist. Die Nutzung versteckter Kanäle kann man dadurch erschweren, dass nur deterministisches Schnittstellenverhalten zugelassen wird (vgl. [ZFK+98]). Zur Verhinderung von Steganographie müssen zum Beispiel Ausgabedaten in einer eindeutigen funktionalen Abhängigkeit von vorhergehenden Eingaben stehen. Um Angriffe auf die Integrität von

[1] Allerdings wird der Zustand der simulierten cTLA-Spezifikation immer aktualisiert, um später mit der Überwachung wieder aufsetzen zu können.

Komponenten zu erschweren, werden Schnittstellenereignisse und deren Argumente eingeschränkt. Außerdem kann man zur Laufzeit überprüfen, ob die Ereignisse plausibel sind. Durch Erzwingung minimaler Wartezeiten zwischen bestimmten Schnittstellenereignissen kann man Denial-of-Service Angriffe verhindern, da durch die Wartezeiten genügend Zeit zur Abarbeitung von Aufträgen dritter besteht. Umgekehrt kann man Blockadeangriffe verhindern, indem man verlangt, dass ein erwartetes Schnittstellenereignis innerhalb einer maximalen Reaktionszeit ausgeführt wird. Relativ schwer zu realisieren sind Policies zur Verhinderung, dass Ereignisse später abgestritten werden. Ein Schritt hierzu ist die Verwendung einer neutralen Instanz, die Information über Schnittstellenereignisse sammelt und sie sich von den beteiligten Komponenten digital signieren lässt.

Ein Komponentenentwickler kann für seine Komponente die Einhaltung einer Sicherheitspolicy garantieren, indem er eine Beschreibung der Schnittstellenereignisse in den Komponentenkontrakt einfügt, aus der hervorgeht, dass kein von der Komponente initiiertes Ereignis die Sicherheitspolicy verletzt. Um diese Verhaltensbeschreibungen erstellen zu können, haben wir spezielle cTLA-basierte Spezifikationsmuster entwickelt, aus denen man die Spezifikationen der Schnittstellenereignisse durch einfaches Instantiieren von Parametern erzeugen kann. Die cTLA-Spezifikationen können sehr einfach in Zustandstransitionssysteme umgesetzt werden, die man durch die Security Wrapper simulieren kann (vgl. [HK01]). In einer derartigen Simulation führt ein Schnittstellenereignis einer Komponente zu jeweils einem Zustandsübergang. Dadurch berücksichtigen die Wrapper die Ereignishistorie und können überprüfen, ob Schnittstellenereignisse zu der Historie konform sind. Die Muster der Sicherheitspolicies sind wie auch die Musterinstanzen, die in unserem E-Commerce-Beispiel verwendet werden, im WWW verfügbar (siehe URL: ls4-www.informatik.uni-dortmund.de/RVS/P-SACS/eReq).

Im folgenden listen wir für die Schutzziele der Vertraulichkeit, Integrität, Verfügbarkeit und Nichtabstreitbarkeit einsetzbaren Spezifikationsmuster auf. Sicherheitspolicies für die Vertraulichkeit von komponentenbasierter Systeme kann man mit Hilfe der folgenden fünf Muster beschreiben:

Data Flow Access: Eine Dateneinheit darf nur an eine Komponente übergeben werden, wenn diese die Leserechte der Dateneinheit erfüllt.

Data Flow History: Eine Dateneinheit darf nur an eine Komponente übergeben werden, wenn zuvor eine bestimmte Folge an Schnittstellenereignissen aufgetreten ist.

Hidden Channel Functional Dependency: Eine weitergeleitete Dateneinheit steht in einer funktionalen Beziehung zu zuvor gesendeten oder empfangenen Daten.

Hidden Channel Enabling History: Die Schaltbedingung eines Schnittstellenereignisses mit einer bestimmten Argumentbelegung steht in einer funktionalen Abhängigkeit zu vorhergehenden Schnittstellenereignissen.

Hidden Channel Execution Time: Ein Schnittstellenereignis muss nach einem vorangegangenen Ereignis innerhalb eines festgelegten Zeitintervalls geschaltet werden.

Muster von Sicherheitspolicies zur Garantie von Integritäts-Sicherheitszielen werden im folgenden aufgeführt:

Integrity Enabling Condition: Ein Schnittstellenereignis mit einer bestimmten Argumentbelegung darf nur unter festgelegten Bedingungen geschaltet werden, so dass die Integrität der Komponente, die das Ereignis empfängt, nicht verletzt wird.

Integrity Enabling History: Ein Schnittstellenereignis mit einer bestimmten Argumentbelegung darf nur geschaltet werden, wenn zuvor eine festgelegte Folge an Schnittstellenereignissen aufgetreten ist. Dadurch garantiert man, dass die Integrität der Komponente, die das Ereignis empfängt, nicht verletzt wird.

Sicherheitspolicies, mit denen man die beiden Angriffsarten auf die Verfügbarkeit von Komponenten ausschließt, werden durch die folgenden Muster beschrieben:

Denial-of-Service Minimum Waiting Time: Ein Schnittstellenereignis darf nur ausgeführt werden, wenn nach einem vorhergehenden Ereignis eine minimale Wartezeit verstrichen ist.

Denial-of-Service Enabling History: Ein Schnittstellenereignis mit einer bestimmten Argumentbelegung darf nur geschaltet werden, wenn zuvor eine festgelegte Folge an Schnittstellenereignissen aufgetreten ist. Zusätzlich muss eine minimale Wartezeit seit dem letzten Ereignis der Folge eingehalten werden.

Blocking Maximum Waiting Time: Ein Schnittstellenereignis muss geschaltet werden, bevor nach einem vorangegangenen Ereignis eine maximale Reaktionszeit verstrichen ist.

Blocking Enabling History: Abhängig von der Folge vorangegangener Ereignisse muss ein Schnittstellenereignis innerhalb einer maximalen Reaktionszeit geschaltet werden.

Die Beschreibung der Nichtabstreitbarkeit von Schnittstellenereignissen unterstützen wir mit dem folgenden Muster:

Event Logging: Ein aus- oder eingehendes Schnittstellenereignis muss zusammen mit einer digitalen Signatur bei einer neutralen Instanz hinterlegt werden.

5 Komponentenbasierte E-Procurement-Anwendung

Das OBI-Konsortium hat zur Standardisierung der Prozeduren bei elektronischen Beschaffungsmaßnahmen (E-Procurement) die Norm OBI (Open Buying on the Internet) [OBI99] herausgegeben. In diesem Standard wird eine Architektur für Beschaffungsaktivitäten definiert, die aus einem Käufer, einer Anzahl an Lieferanten, einem Bank- oder Kreditinstitut und einem Beschaffer (Requisitioner) besteht. Der Beschaffer führt im Auftrag des Käufers Beschaffungen von Gütern bei einem Lieferanten durch. Die Bank rechnet die Bestellungen ab. Das von OBI definierte Geschäftsprozessmodell sieht dabei die folgenden Schritte vor:

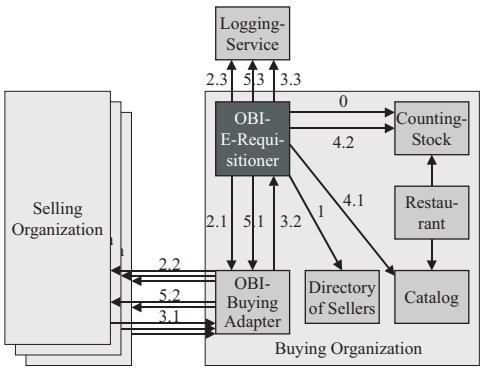


Abbildung 4: Beispiel eines E-Procurement-Systems für Schnellrestaurants

1. Der Beschaffer fordert beim Käufer Verbindungsadressen für Zugangsrechner von Lieferanten an.

2. Der Beschaffer holt für die zu erwerbenden Güter Angebote der Lieferanten ein.

3. Die Lieferanten erzeugen die Angebote und codieren sie nach dem EDI-Standard (vgl. [Dat01]) in dem von OBI festgelegten Order Request-Format. Die Angebote werden dann entweder direkt an den Käufer gesendet zum Beschaffer übertragen, der sie dann an den Käufer weiter leitet.

4. Auf der Basis der Angebote wählt der Beschaffer — gegebenenfalls in Zusammenarbeit mit anderen Einheiten der Käuferorganisation — einen Lieferanten aus und erzeugt eine Bestellung.

5. Die Bestellung wird ebenfalls nach dem EDI-Standard im so genannten Order Format kodiert und dem ausgewählten Lieferanten übermittelt.

6. Der Lieferant führt die Bestellung aus.

7. Die Bank stellt im Auftrag des Lieferanten eine Rechnung an den Käufer aus und empfängt die Bezahlung.

Unser Anwendungsbeispiel simuliert den Einkauf und die Lagerhaltung von Schnellrestaurants. Es wurde als komponentenstrukturiertes System auf der Basis des SalesPoint-Framework entwickelt (vgl. [Sc99]). SalesPoint ist Freeware und ermöglicht durch fertige Module für Geschäftsfunktionen wie Einkauf, Verkauf oder Leasing von Gütern sowie Module für Verwaltungsfunktionen wie Abrechnung, Lagerverwaltung oder Erzeugung des Produktkatalogs die recht einfache Erzeugung unterschiedlicher Warenwirtschaftssysteme. Auf der Basis von SalesPoint haben wir drei Java Beans entwickelt, die die Verkaufsfunktionen eines Restaurants, die Lagerverwaltung sowie die Verwaltung des Produktkatalogs realisieren.

Abbildung 4 beschreibt die Komponenten unseres Beispiels. Um die elektronische Beschaffung der Speisen und Getränke zu automatisieren, haben wir neben den von SalesPoint abgeleiteten Komponenten *Restaurant*, *Counting Stock* und *Catalog* drei weitere Komponenten implementiert. Die Beschaffungen werden durch die Komponente *OBI-E-Requisitioner* vorgenommen, die den im OBI-Standard vorgesehenen Beschaffer ersetzt. Außerdem wurde eine Komponente *Directory of Sellers* eingefügt, die die Adressen der Lieferanten und ihre Produktpalette speichert. Schließlich verwenden wir eine Komponente *OBI-Buying Adapter*, die die Kodierung und Dekodierung von Angebotsanfragen, Angeboten und Bestellungen übernimmt und als Schnittstelle zu den Systemen der Lieferanten dient. Die sechs Komponenten wurden miteinander zum Warenwirtschaftsystem des Schnellrestaurants gekoppelt. Die Lieferantensysteme wurden ebenfalls mit Hilfe der Module des SalesPoint-Framework entwickelt. Schließlich haben wir die Komponente *Logging Service* als neutrale Instanz implementiert, um das spätere Abstreiten des Austausch von Angebotsanfragen, Angeboten oder Bestellungen zu erschweren. Da wir uns im Weiteren nur für die Phasen bis zur Abgabe der Bestellung interessieren, haben wir, um das System einfach zu halten, auf die Komponenten zur Realisierung der Bank verzichtet.

Für unser Ziel, die Warenbeschaffung zu automatisieren, mussten wir allerdings den OBI-Standard, der keine formale Definition für Anfragen nach Angeboten vorsieht, ergänzen. Dazu haben wir EDI durch cXML (Commercial eXtensible Markup Language) [cXM01] als Transfersyntax ersetzt, die im Gegensatz zu EDI ein Format für Angebotsanfragen vorsieht. cXML basiert auf der populären Datencodiersprache XML.

Die einzelnen Schritte einer Beschaffung werden durch die Kanten in Abbildung 4 dargestellt. Da der E-Requisitioner nicht nur für den Beschaffungsablauf zuständig ist, sondern auch entscheidet, wann die Beschaffung gestartet werden soll, haben wir einen neuen ersten Schritt 0 eingeführt, in dem der E-Requisitioner das Lager in zeitlichen Abständen inspiziert und abhängig von der Menge an vorhandenen Waren festlegt, ob und für welche Güter eine Beschaffung vorgenommen wird. Wenn neue Güter bestellt werden müssen, liest der E-Requisitioner vom Verkäuferverzeichnis die Adressen geeigneter Lieferanten (Schritt 1). Danach erzeugt er eine Angebotsanfrage und übermittelt sie über den OBI-Buying Adapter an die Lieferanten (Schritt 2). Die Lieferanten übermitteln die Angebote an den Adapter, der sie an den E-Requisitioner weiter leitet (Schritt 3). Anschließend konsultiert der E-Requisitioner den Produktkatalog und die Lagerhaltung und wählt abhängig von den Angeboten, den Verkaufspreisen und dem Warenbestand einen Lieferanten aus (Schritt 4). Schließlich wird die Bestellung an den Adapter gesendet, der sie an den entsprechenden Lieferanten weiter leitet (Schritt 5). Die Schritte 6 und 7 wurden nicht realisiert. Um die Angebotsanfragen, Angebote und Bestellungen später nicht abstreiten zu können, sendet der E-Requisitioner in den Schritten 2, 3 und 5 die gesendeten bzw. empfangenen Daten zusammen mit seiner digitalen Signatur an die neutrale Instanz.

6 Verhaltensüberwachung im Beispielsystem

Um einen korrekten und sicheren Ablauf der Beschaffungen zu garantieren, muss sich der E-Requisitioner immer gutmütig verhalten. Er darf zum Beispiel nicht Angebote an kon-

kurrierende Lieferanten weitergeben, um sie bei der Auftragsvergabe zu bevorzugen. Auch muss er immer ein günstigstes Angebot für seine Bestellung auswählen und darf die Lagerhaltung des Käufers nicht durch zu große, zu kleine oder zu späte Bestellungen schädigen. Im folgenden nehmen wir an, dass der E-Requisitioner von einer nicht vollständig vertrauenswürdigen Quelle erworben wurde. Zur Sicherung des gesamten Systems müssen wir ihn deshalb durch einen Security Wrapper überwachen lassen. Wir haben dazu 13 Sicherheitspolicies formuliert, die für den korrekten Systemablauf wichtig sind. Anschließend haben wir aus den in Abschnitt 4 vorgestellten Mustern cTLA-Spezifikationen dieser Policies erzeugt. Im nächsten Schritt wurden die Spezifikationen in Java-Code überführt, der dann in die Observer des Security Wrappers eingebettet wurde.

Im folgenden listen wir die Sicherheitspolicies kurz auf[2]. Zum Schutz der Vertraulichkeit der Anwendung werden die folgenden Policies gefordert:

1. Eine Angebotsanforderung an einen Lieferanten darf nur Güter enthalten, die dieser Lieferant in seinem Warenbestand hat (Data Flow Access). Damit erhalten Lieferanten keine Information über Güter, die nicht für sie relevant sind.

2. Eine Angebotsanforderung darf nur an Lieferanten gesendet werden, die im Lieferantenverzeichnis aufgeführt sind (Data Flow Access). Damit werden Informationen über die vom Restaurant vertriebenen Speisen und Getränke nur an Lieferanten weitergegeben, die von der Käuferorganisation akzeptiert sind.

3. Die Menge eines angefragten bzw. bestellten Artikels hängt eindeutig mit dem Lagerbestand des Artikels zusammen (Hidden Channel Functional Dependency). Damit soll ausgeschlossen werden, dass der E-Requisitioner durch Angabe bestimmter Bestellmengen geheime Zusatzinformation an einen Lieferanten weiter gibt.

4. Eine neue Beschaffungsrunde darf nur gestartet werden, wenn die vorhergehende mit der Warenlieferung abgeschlossen ist (Hidden Channel Enabling History). Damit soll ausgeschlossen werden, dass eine Bestellung in mehrere Unteraufträge aufgeteilt wird. Die Art der Aufteilung könnte eine geheime Zusatzinformation enthalten.

Zur Integritätssicherung verwenden wir die folgenden Sicherheitspolicies:

1. Eine Bestellung darf nur vorgenommen werden, wenn zuvor eine bestimmte Anzahl an Angeboten eingegangen ist (Integrity Enabling History). Dadurch wird verhindert, dass der E-Requisitioner durch eine schnelle Bestellung bei einem bevorzugten Lieferanten später eintreffende Angebote nicht berücksichtigt.

2. Der E-Requisitioner wählt immer eines der günstigsten Angebote aus (Integrity Enabling History).

3. Die Daten der Lagerverwaltung, des Produktkatalogs, des OBI-Buying Adapter und der neutralen Instanz werden nicht geändert (Integrity Enabling Condition „false“

[2]Dabei gibt der in Klammern gesetzte Bezeichner den Namen des Musters an, aus dem die Spezifikation abgeleitet wurde (siehe Kapitel 4).

für alle Methoden, die die Daten ändern). Damit wird die Datenintegrität der mit dem E-Requisitioner gekoppelten Komponenten geschützt.

4. Bei einer Bestellung liegt die Bestellmenge immer in einem festen Bestellintervall (Integrity Enabling Condition). Dadurch werden zu große oder zu kleine Bestellmengen vermieden.

Zur Verhinderung von Angriffen auf die Verfügbarkeit des Systems soll die E-Requisitioner-Komponente die folgenden Sicherheitspolicies erbringen:

1. Zwischen zwei an die Lagerverwaltung, den Produktkatalog, das Lieferantenverzeichnis oder den Buying Adapter gesendete Schnittstellenereignisse liegt immer eine minimale Wartezeit (Denial-of-Service Minimum Waiting Time). Dadurch wird verhindert, dass die Komponenten ständig belastet werden und damit nicht mehr andere Komponenten (z.B. die Verkaufskomponente) bedienen können.

2. Der Warenbestand in der Lagerverwaltung wird innerhalb eines maximalen Zeitintervalls kontrolliert (Blocking Maximum Waiting Time). Dadurch ist sichergestellt, dass der E-Requisitioner einen niedrigen Lagerbestand einer Ware rechtzeitig erkennt.

3. Wenn der E-Requisitioner feststellt, dass eine Ware knapp ist, startet er den Prozess zur Ersatzbeschaffung innerhalb einer maximalen Zeit (Blocking Enabling History). Dadurch verhindert man, dass der E-Requisitioner eine Bestellung so lange aufschiebt, bis die Ware nicht mehr verfügbar ist.

4. Nach Empfang der in der ersten Policy zur Integritätssicherung angegeben Mindestanzahl an Angeboten, die für eine Bestellung notwendig ist, führt der E-Requisitioner die Bestellung innerhalb einer maximalen Reaktionszeit aus (Blocking Enabling History). Auch hiermit wird sichergestellt, dass Bestellungen nicht zu lange verzögert werden.

Schließlich muss man garantieren, dass der Käufer später gegenüber den Lieferanten aber auch dem Entwickler der E-Requisitioner-Komponente nachweisen kann, dass Angebotsanfragen und Bestellungen tatsächlich getätigt und dass Angebote tatsächlich empfangen wurden. Dazu verwenden wir die folgende Sicherheitspolicy:

1. Angebotsanfragen, Angebote und Bestellungen müssen der neutralen Instanz innerhalb einer maximalen Zeit unter Hinzufügung einer eindeutigen digitalen Signatur gemeldet werden (Event Logging).

Der Security Wrapper überprüft, ob die angegebenen 13 Sicherheitspolicies von der Komponente eingehalten werden. Messungen haben ergeben, dass in unserer Beispielimplementierung die Mehrbelastung für diese Laufzeittests etwa bei 5 % liegt. Um bei wachsendem Vertrauen in die Komponente den Überprüfungsaufwand durch Stichproben oder die Aufgabe der Überprüfung einzelner Policies senken zu können, setzen wir neben dem eigentlichen Security Wrapper auch den Vertrauensinformationsdienst sowie einen Trust

Manager ein. Zur Konfigurierung des Trust Managers müssen wir dazu für jede einzelne Policy festlegen, welche Überprüfungsmaßnahmen bei welchen Vertrauenswerten durchgeführt werden. Dazu betrachten wir zunächst, wie sicherheitskritisch die Einhaltung der jeweiligen Policy für das Gesamtsystem ist.

Die Sicherheitspolicies garantieren zum einen, dass Information über die vom Restaurant vertriebenen Waren nur an bestimmte Lieferanten weitergegeben wird. Im Bereich der Schnellrestaurants scheint der Geheimhaltungsbedarf aufgrund der allgemeinen Bekanntheit der Produktpalette jedoch nicht groß zu sein. Allerdings muss man berücksichtigen, dass ein Restaurant geschädigt werden kann, wenn die Einführung eines neuen Artikels der Konkurrenz frühzeitig bekannt wird. Aufgrund dieser gegenteiligen Effekte verwenden wir einen mittelstarken Schutz. Die ersten beiden Vetraulichkeits-Sicherheitspolicies werden nur vollständig überprüft, wenn nach der Metrik von Jøsang und Knapskog [JK98] die Variable b des aktuellen Vertrauenswertes unter dem Wert $0,99$ liegt. Stichpunkthafte Überprüfungen führen wir durch, wenn b einen Wert zwischen $0,99$ und $0,999$ hat, während die Überwachung bei noch höheren Werten von b eingestellt wird. Dass heißt, das wir vollständige Überprüfungen durchführen, wenn die Anzahl positiver Ereignisse die Anzahl negativer Ereignisse um weniger als den Faktor Hundert übersteigt. Bei einem Faktor von mehr als Tausend wird die Überwachung unwiderruflich abgebrochen.

Die Forderung, dass bei den Bestellvorgängen Angebote nicht an konkurrierende Lieferanten gelangen können, ist dagegen von sehr großer Bedeutung. Da die OBI-Buying Adapter-Komponente an Lieferanten nur die Übertragung von cXML-Angebotsanforderungen und Bestellungen erlaubt, stehen für die illegale Übermittlung der Angebote nur versteckte Kanäle zur Verfügung, die man durch die zwei zuletzt genannten Vertraulichkeitspolicies verhindern will. Die Einhaltung dieser Policies ist deshalb so wichtig, dass wir sie mit dem Wrapper permanent vollständig überprüfen.

Von großer Bedeutung sind auch alle vier Sicherheitspolicies zur Garantie der Integrität. Bei Verletzung der ersten beiden Policies kann der Käufer durch zu teure Bestellungen geschädigt werden. Sehr wesentlich ist auch der durch die dritte Policy erzwungene Schutz der Datenintegrität, da deren Verletzung andere Angriffe nach sich ziehen kann [3]. Schließlich ist auch die vierte Sicherheitspolicy für die Systemintegrität von Bedeutung, da zu hohe Bestellmengen der meist leicht verderblichen Güter zu hohen finanziellen Schäden führen können. Wir überprüfen aus diesem Grund alle vier Policies ständig.

Die Anforderungen an die Systemverfügbarkeit sind gegenüber denjenigen an die Integrität von etwas geringerer Bedeutung. Zwar sind Denial-of-Service Angriffe unangenehm, da sie den Verkauf schädigen können. Allerdings wird das Verkaufspersonal das Problem schnell erkennen, so dass die Fehlerquelle frühzeitig gefunden und behoben werden kann. Da uns der Gefährdungsgrad mit dem des Bekanntwerdens der Produktpalette vergleichbar erscheint, verwenden wir hier ebenfalls die Werte $b \geq 0,99$ für Stichproben und $b \geq 0,999$ für den Abbruch der Überprüfung.

Leere Lager aufgrund zu später Bestellungen sind im Vergleich zu Denial-of-Service-Angriffen kritischer, da sie zu finanziellen Einbußen führen. Allerdings sind wir hier der Meinung, dass die Restaurantleitung auch unabhängig vom E-Requisitioner auf den La-

[3] Zum Beispiel könnte ein preislich günstiger Lieferant aus dem Lieferantenverzeichnis gelöscht werden.

gerbestand achten sollte, so dass ein Warenengpass meist noch rechtzeitig erkannt und behoben werden kann. Wir verwenden hier etwas rigidere Überwachungsmethoden als für die Sicherheitspolicy gegen Denial-of-Service-Angriffe. Zum einen setzen wir die Metrik von Beth et al. ein [BBK94], um die Policies schon nach einer beim Vertrauensinformationsdienst eingegangenen negativen Bewertung immer vollständig zu überwachen. Außerdem fangen wir erst bei einem Wert von $b \geq 0,999$ mit Stichproben an. Die Überprüfung wird erst bei einem Wert von $b \geq 0,99999$ vollständig aufgegeben.

Die Meldung von Interaktionen mit den Lieferanten an die neutrale Instanz ist von hoher Bedeutung, da sie rechtliche Relevanz erlangen kann. Wir überprüfen die Einhaltung der entsprechenden Sicherheitspolicy deshalb ständig.

7 Schlußbemerkungen

In diesem Papier stellten wir die Security Wrapper vor, mit denen man durchsetzen kann, dass die Policies von den Komponenten zur Laufzeit eingehalten werden. Schließlich wurde eine kurze Einführung in das Vertrauensmanagement von Komponenten gegeben. In unserem E-Procurement-Beispiel führte der Einsatz der Wrapper zu einem Leistungsmehraufwand von etwa 5%. Der Einsatz des Trust Managers, mit dem man bei guten Vertrauenswerten einige Observer abschalten kann, reduziert den Mehraufwand auf etwa 3%.

Neben dieser Arbeit konzentriert sich unser Ansatz auch auf die Entwicklung eines cTLA-Frameworks zur Modellierung der Sicherheitseigenschaften von komponentenbasierten Systemen (vgl. [He03a]). Das Framework enthält eine Sammlung an Mustern zur Beschreibung von Sicherheitspolicies einzelner Komponenten, wie wir sie in Abschnitt 4 vorgestellt haben. Daneben wird ihm eine weitere Sammlung an Mustern hinzugefügt, mit der man an das Gesamtsystem gestellte Sicherheitsanforderungen formal beschreiben kann. Darüberhinaus werden von uns bereits verifizierte Theoreme hinzugefügt, die jeweils aussagen, dass eine Kombination von Komponenten-Sicherheitspolicies eine Systemsicherheitsanforderung erbringt. Mit den Mustersammlungen kann man die Sicherheitsanforderungen recht einfach spezifizieren, indem man entsprechende Muster parametrisiert und zu einer umfassenderen Sicherheitsspezifikation kombiniert. Das Framework erleichtert zudem die logischen Deduktionsbeweise. Ein Beweis kann nämlich in Lemmata zerlegt werden, die den Frameworktheoremen entsprechen.

Literatur

[BBK94] Beth, T., Borcherding, M., und Klein, B.: Valuation of Trust in Open Networks. In: *Proc. European Symposium on Research in Security (ESORICS)*. LNCS 875. S. 3–18. Brighton. 1994. Springer-Verlag.

[BJPW99] Beugnard, A., Jézéquel, J.-M., Plouzeau, N., und Watkins, D.: Making Components Contract Aware. *IEEE Computer*. 32(7):38–45. 1999.

[cXM01] cXML.org: *cXML User's Guide*. 1.2.006. August 2001.

[Dat01] Data Interchange Standards Association: *X12 Standard*. Release 4050. December 2001.

[He01] Herrmann, P.: Trust-Based Procurement Support for Software Components. In: *Proc. 4th International Conference on Electronic Commerce Research (ICECR-4)*. S. 505–514. Dallas. November 2001. ATSMA, IFIP.

[He03a] Herrmann, P.: Formal Security Policy Verification of Distributed Component-Structured Software. In: König, H., Heiner, M., und Wolisz, A. (Hrsg.), *Proc. 23rd IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'2003)*. LNCS 2767. S. 257–272. Berlin. September 2003. Springer-Verlag.

[He03b] Herrmann, P.: Trust-Based Protection of Software Component Users and Designers. In: Nixon, P. und Terzis, S. (Hrsg.), *Proc. 1st International Conference on Trust Management*. LNCS 2692. S. 75–90. Heraklion. May 2003. Springer-Verlag.

[HK00] Herrmann, P. und Krumm, H.: A Framework for Modeling Transfer Protocols. *Computer Networks*. 34(2):317–337. 2000.

[HK01] Herrmann, P. und Krumm, H.: Trust-adapted enforcement of security policies in distributed component-structured applications. In: *Proc. 6th IEEE Symposium on Computers and Communications*. S. 2–8. Hammamet. July 2001. IEEE Computer Society Press.

[JK98] Jøsang, A. und Knapskog, S. J.: A metric for trusted systems. In: *Proc. 21st National Security Conference*. NSA. 1998.

[Jø96] Jøsang, A.: The right type of trust for distributed systems. In: *Proc. UCLA conference on New security paradigms workshops*. S. 119–131. Lake Arrowhead. September 1996. ACM.

[Jø99] Jøsang, A.: An Algebra for Assessing Trust in Certification Chains. In: Kochmar, J. (Hrsg.), *Proc. Network and Distributed Systems Security Symposium (NDSS'99)*. The Internet Society. 1999.

[Ma00] Mallek, A.: Sicherheit komponentenstrukturierter verteilter Systeme: Vertrauensabhängige Komponentenüberwachung. Diplomarbeit. Universität Dortmund, Informatik IV. D-44221 Dortmund. 2000.

[OBI99] OBI Consortium: *OBI Technical Specifications — Open Buying on the Internet*. Draft release v2.1. 1999.

[RZFK00] Resnick, P., Zeckhauser, R., Friedman, E., und Kuwabara, K.: Reputation Systems: Facilitating Trust in Internet Interactions. *Communications of the ACM*. 43(12):45–48. December 2000.

[Sc99] Schmitz, L.: The SalesPoint Framework — Technical Overview. Available via WWW: ist.unibw-muenchen.de/Lectures/SalesPoint/overview/english/TechDoc.htm. November 1999.

[Sz97] Szyperski, C.: *Component Software — Beyond Object Oriented Programming*. Addison-Wesley Longman. 1997.

[ZFK+98] Zöllner, J., Federrath, H., Klimant, H., Pfitzmann, A., Piotraschke, R., Westfeld, A., Wicke, G., und Wolf, G.: Modeling the security of steganographic systems. In: *Proc. 2nd Workshop of Information Hiding*. LNCS 1525. S. 345–355. Portland. 1998. Springer-Verlag.

Intrusion detection in unlabeled data with quarter-sphere Support Vector Machines

Pavel Laskov and Christin Schäfer
Fraunhofer-FIRST
Kekuléstr. 7
12489 Berlin, Germany
{laskov,christin}@first.fhg.de

Igor Kotenko
SPIIRAS
14th Liniya 39
199178 St. Petersburg, Russia
ivkote@spiiras.nw.ru

Abstract: Practical application of data mining and machine learning techniques to intrusion detection is often hindered by the difficulty to produce clean data for the training. To address this problem a geometric framework for unsupervised anomaly detection has been recently proposed. In this framework, the data is mapped into a feature space, and anomalies are detected as the entries in sparsely populated regions. In this contribution we propose a novel formulation of a one-class Support Vector Machine (SVM) specially designed for typical IDS data features. The key idea of our "quarter-sphere" algorithm is to encompass the data with a hypersphere anchored at the center of mass of the data in feature space. The proposed method and its behavior on varying percentages of attacks in the data is evaluated on the KDDCup 1999 dataset.

1 Introduction

The majority of current intrusion detection methods can be classified as either misuse detection or anomaly detection [NWY02]. The former identify patterns of known illegitimate activity; the latter focus on unusual activity patterns. Both groups of methods have their advantages and disadvantages. Misuse detection methods are generally more accurate but are fundamentally limited to known attacks. Anomaly detection methods are usually less accurate than misuse detecion methods — in particular, their false alarm rates are hardly acceptable in practice — however, they are at least in principle capable of detecting novel attacks. This feature makes anomaly detection methods the topic of active research.

In some early approaches, e.g. [DR90, LV92], it was attempted to describe the normal behavior by means of some high-level rules. This turned out to be quite a difficult task. More successful was the idea of collecting data from normal operation of a system and computing, based on this data, features describing normality; deviation of such features would be considered an anomaly. This approach is known as "supervised anomaly detection". Different techniques have been proposed for characterizing the concept of normality, most notably statistical techniques, e.g. [De87, JLA+93, PN97, WFP99], and data mining techniques, e.g. [BCJ+01, VS00]. In practice, however, it is difficult to obtain clean data to implement these approaches. Verifying that no attacks are present in the training data may

be an extremely tedious task, and for large samples this is infeasible. On the other hand, if the "contaminated" data is treated as clean, intrusions similar to the ones present in the training data will be accepted as normal patterns.

To overcome the difficulty in obtaining clean data, the idea of *unsupervised* anomaly detection has been recently proposed and investigated on several intrusion detection problems [PES01, EAP+02, LEK+03]. These methods compute some relevant features and use techniques of unsupervised learning to identify sparsely populated areas in feature space. The points — whether in the training or in the test data — that fall into such areas are treated as anomalies.

More precisely, two kinds of unsupervised learning methods have been investigated: clustering methods and one-class SVM. In this contribution we focus on one-class SVM methods and investigate the application of the underlying geometric ideas in the context of intrusion detection.

We present three formulations of one-class SVM that can be derived following different geometric intuitions. The formulation used in previous work was that of the hyperplane separating the normal data from the origin [SPST+01]. Another formulation, motivated by fitting a sphere over the normal data, is also well-known in the literature on kernel methods [TD99]. The novel formulation we propose in this paper is based on fitting a sphere centered at the origin to normal data. This formulation, to be refered to as a *quarter-sphere*, is particularly suitable to the features common in intrusion detection, whose distributions are usually one-sided and concentrated at the origin.

Finally, we present an experimental evaluation of the one-class SVM methods under a number of different scenarios.

2 One-class SVM formulations

Support Vector Machines have received great interest in the machine learning community since their introduction in the mid-1990s. We refer the reader interested in the underlying statistical learning theory and the practice of designing efficient SVM learning algorithms to the well-known literature on kernel methods, e.g. [Va95, Va98, SS02]. The one-class SVM constitutes the extension of the main SVM ideas from supervised to unsupervised learning paradigms.

We begin our investigation into the application of the one-class SVM for intrusion detection with a brief re-capitulation and critical analysis of the two known approaches to one-class SVM. It will follow from this analysis that the quarter-sphere formulation, described in section 2.4, could be better suited for the data common in intrusion detection problems.

2.1 The plane formulation

The original idea of the one-class SVM [SPST+01] was formulated as an "estimation of the support of a high-dimensional distribution". The essense of this approach is to map the data points x_i into the feature space by some non-linear mapping $\Phi(x_i)$, and to separate the resulting image points from the origin with the largest possible margin by means of a hyperplane. The geometry of this idea is illustrated in Fig. 1. Due to nonlinearity of

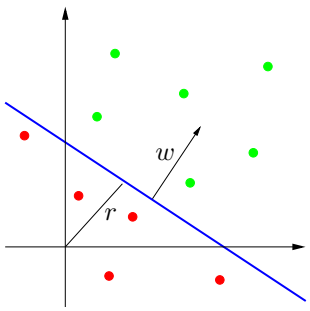


Figure 1: The geometry of the plane formulation of one-class SVM.

feature space, maximization of the separation margin limits the volume occupied by the normal points to a relatively compact area in feature space. Mathematically, the problem of separating the data from the origin with the largest possible margin is formulated as follows:

$$\begin{aligned} \min_{w\in\mathcal{F},\xi\in\mathbb{R}^l,r\in\mathbb{R}} \quad & \tfrac{1}{2}||w||^2 + \tfrac{1}{\nu l}\textstyle\sum_{i=1}^{l}\xi_i - r, \\ \text{subject to:} \quad & (w\cdot\Phi(x_i)) \geq r - \xi_i, \\ & \xi_i \geq 0. \end{aligned} \tag{1}$$

The weight vector w, characterizing the hyperplane, "lives" in the feature space $\mathcal{F}$, and therefore is not directly accessible (as the feature space may be extremely high-dimensional). The non-negative slack variables ξ_i allow for some points, the anomalies, to lie on the "wrong" side of the hyperplane. Instead of the primal problem (1), the following dual problem, in which all the variables have low dimensions, is solved in practice:

$$\begin{aligned} \min_{\alpha\in\mathbb{R}^l} \quad & \textstyle\sum_{ij=1}^{l}\alpha_i\alpha_j k(x_i,x_j), \\ \text{subject to:} \quad & \textstyle\sum_{i=1}^{l}\alpha_i = 1, \\ & 0 \leq \alpha_i \leq \tfrac{1}{\nu l}. \end{aligned} \tag{2}$$

Once the solution α is found, one can compute the threshold parameter $r = \sum_j \alpha_j k(x_i,x_j)$ for some example i such that α_i lies strictly between the bounds (such points are called *support vectors*). The decision, whether or not point x is normal, is computed as:

$$f(x) = \text{sgn}\left(\textstyle\sum_i \alpha_i k(x_i,x) - r\right). \tag{3}$$

The points with $f(x) = -1$ are considered to be anomalies.

2.2 The sphere formulation

Another, somewhat more intuitive geometric idea for the one-class SVM is realized in the sphere formulation [TD99]. The normal data can be concisely described by a sphere (in a feature space) encompassing the data, as shown in Fig. 2. The presence of anomalies in the

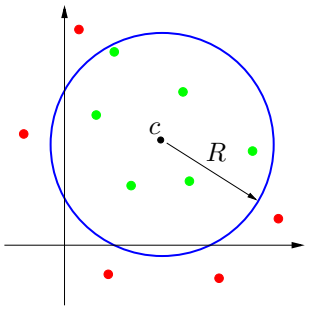


Figure 2: The geometry of the sphere formulation of one-class SVM.

training data can be treated by introducing slack variables ξ_i, similarly to the plane formulation. Mathematically the problem of "soft-fitting" the sphere over the data is described as:

$$\begin{aligned} \min_{R\in\mathbb{R},\xi\in\mathbb{R}^l,c\in\mathcal{F}} \quad & R^2 + \tfrac{1}{\nu l}\textstyle\sum_{i=1}^{l}\xi_i, \\ \text{subject to:} \quad & ||\Phi(x_i) - c|| \leq R^2 + \xi_i, \\ & \xi_i \geq 0. \end{aligned} \tag{4}$$

Similarly to the primal formulation (1) of the plane one-class SVM, one cannot directly solve the primal problem (4) of the sphere formulation, since the center c belongs to the possibly high-dimensional feature space. The same trick can be employed — the solution is sought to the dual problem:

$$\begin{aligned} \min_{\alpha\in\mathbb{R}^l} \quad & \textstyle\sum_{ij=1}^{l}\alpha_i\alpha_j k(x_i,x_j) - \sum_{i=1}^{l}\alpha_i k(x_i,x_i), \\ \text{subject to:} \quad & \textstyle\sum_{i=1}^{l}\alpha_i = 1, \\ & 0 \leq \alpha_i \leq \tfrac{1}{\nu l}. \end{aligned} \tag{5}$$

The decision function can be computed as:

$$f(x) = \text{sgn}\left(R^2 - \textstyle\sum_{ij=1}^{l}\alpha_i\alpha_j k(x_i,x_j) + 2\sum_{i=1}^{l}\alpha_i k(x_i,x) - k(x,x)\right). \tag{6}$$

The radius R^2 plays the role of a threshold, and, similarly to the plane formulation, it can be computed by equating the expression under the "sgn" to zero for any support vector.

The similarity between the plane and the sphere formulations goes beyond merely an analogy. As it was noted in [SPST+01], for kernels $k(x,y)$ which depend only on the difference $x-y$, the linear term in the objective function of the dual problem (5) is constant, and the solutions are equivalent.

2.3 Analysis

When applying one-class SVM techniques to intrusion detection problems, the following observation turns out to be of crucial importance: *A typical distribution of the features used in IDS is one-sided on* $\mathbb{R}_0^+$. Several reasons contribute to this property. First, many IDS features are of temporal nature, and their distribution can be modeled using distributions common in survival data analysis, for example by an exponential or a Weibull distribution. Second, a popular approach to attain coherent normalization of numerical attributes is the so-called "data-dependent normalization" [EAP+02]. Under this approach, the features are defined as the deviations from the mean, measured in the fraction of the standard deviation. This quantity can be seen as F-distributed. Summing up, the overwhelming mass of data lies in the vicinity of the origin.

The consequences of the one-sidedness of the data distribution for the one-class SVM can be seen in Fig. 3. The one-sided distribution in the example is generated by taking the

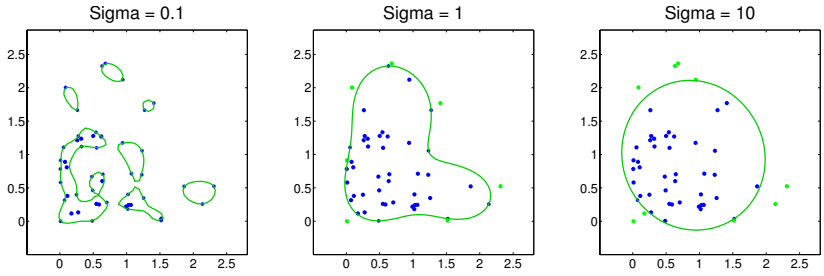


Figure 3: Behavior of the one-class SVM on the data with a one-sided distribution.

absolute values of the normally distributed points. The anomaly detection is shown for a fixed value of the parameter ν and varying smoothness σ of the RBF kernel. The contours show the separation between the normal points and anomalies. One can see that even for the heavily regularized separation boundaries, as in the right picture, some points close to the origin are detected as anomalies. As the regularization is diminished, the one-class SVM produces a very ragged boundary and does not detect any anomalies.

The message that can be carried from this example is that, in order to account for the one-sidedness of the data distribution, one needs to use a geometric construction that is in some sense asymmetric. The new construction we propose here is the quarter-sphere one-class SVM described in the next section.

2.4 The quarter-sphere formulation

A natural way to extend the ideas of one-class SVM to one-sided non-negative data is to require the center of the fitted sphere be fixed at the origin. The geometry of this approach is shown in Fig. 4. Repeating the derivation of the sphere formulation for $c = 0$, the

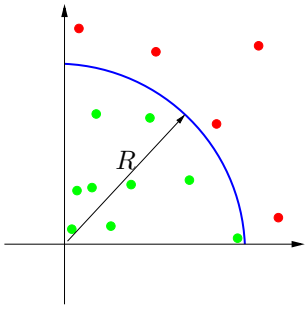


Figure 4: The geometry of the quarter-sphere formulation of one-class SVM.

following dual problem is obtained:

$$\begin{aligned} \min_{\alpha \in \mathbb{R}^l} \quad & -\sum_{i=1}^{l} \alpha_i k(x_i, x_i), \\ \text{subject to:} \quad & \sum_{i=1}^{l} \alpha_i = 1, \\ & 0 \leq \alpha_i \leq \tfrac{1}{\nu l}. \end{aligned} \tag{7}$$

Note that, unlike the other two formulations, the dual problem of the quarter-sphere SVM amounts to a linear rather than a quadratic program. Herein lies the key to the significantly lower computational cost of our formulation.

It may seem somewhat strange that the non-linear mapping affects the solution only through the norms $k(x_i, x_i)$ of the examples, i.e. that the geometric relations *between* the objects are ignored. This feature indeed poses a problem for the application of the quarter-sphere SVM with the distance-based kernels. In such case, the norms of the points are equal, and no meaningful solution to the dual problem can be found. This predicament, however, can be easily fixed. A well-known technique, originating from kernel PCA [SSM98], is to center the images of the training points $\Phi(x_i)$ in feature space. In other words, the values of image points are re-computed in the local coordinate system anchored at the center of mass of the image points. This can be done by subtracting the mean from all image values:

$$\tilde{\Phi}(x_i) = \Phi(x_i) - \tfrac{1}{l} \sum_{i=1}^{l} \Phi(x_i).$$

Although this operation may not be directly computable in feature space, the impact of centering on the kernel values can be easily computed (e.g. [SSM98, SMB$^+$99]):

$$\tilde{K} = K - 1_l K - K 1_l + 1_l K 1_l, \tag{8}$$

where K is the $l \times l$ kernel matrix with the values $K_{ij} = k(x_i, x_j)$, and 1_l is an $l \times l$ matrix with all values equal to $\frac{1}{l}$. After centering in feature space, the norms of points in the local coordinate system are no longer all equal, and the dual problem of the quarter-sphere formulation can be easily solved.

3 Experiments

To compare the quarter-sphere formulation with the other one-class SVM approaches, and to investigate some properties of our algorithm, experiments are carried out on the KDDCup 1999 dataset. This dataset comprises connection record data collected in 1998 DARPA IDS evaluation. The features characterizing these connection records are pre-computed in the KDDCup dataset.

One of the problems with the connection record data from the KDDCup/DARPA data is that a large proportion (about 75%) of the connections represent the anomalies. In previous work [PES01, EAP+02] it was assumed that anomalies constitute only a small fraction of the data, and the results are reported on subsampled datasets, in which the ratio of anomalies is artificially reduced to 1-1.5%. To render our results comparable with previous work we also subsample the data. The results reported below are averaged over 10 runs of the algorithms in any particular setup.

3.1 Comparison of one-class SVM formulations

We first compare the quarter-sphere one-class SVM with the other two algorithms. Since the sphere and the plane formulations are equivalent for the RBF kernels, identical results are produced for these two formulations.

The experiments are carried out for two different values of the parameter σ of the RBF kernel: 1 and 12 (the latter value used in [EAP+02]). These values correspond to low and moderate regularization. As the evaluation criterion, we use the portion of the ROC curve between the false alarm rates of 0 and 0.1, since higher false alarm rates are unacceptable for intrusion detection. The comparison of ROCs of the three formulations for the two values of σ are shown in Fig. 5. It can be easily seen that the quarter-sphere formulation

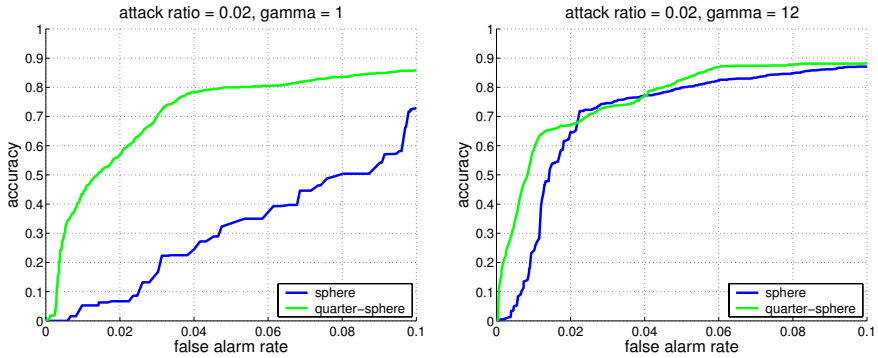


Figure 5: Comparison of the three one-class SVM formulations.

consistently outperforms the other two formulations; especially at the low value of regularization parameter. The best overall results are achieved with the medium regularization

with $\sigma = 12$, which has most likely been selected in [EAP+02] after careful experimentation. The advantage of the quarter-sphere in this case is not so dramatic as with low regularization, but is nevertheless very significant for low false alarm rates.

3.2 Dependency on the ratio of anomalies

The assumption that intrusions constitute a small fraction of the data may not be satisfied in a realistic situation. Some attacks, most notably the denial-of-service attacks, manifest themselves precisely in a large number of connections. Therefore, the problem of a large ratio of anomalies needs to be addressed.

In the experiments in this section we investigate the performance of the sphere and the quarter-sphere one-class SVM as a function of the attack ratio. It is known from the literature [TD99, SPST+01] that the parameter ν of the one-class SVM can be interpreted as an upper bound on the ratio of the anomalies in the data. The effect of this parameter on the quarter-sphere formulation is different: it specifies that *exactly ν fraction of points is expected to be the anomalies*. This is agreeably a more stringent assumption, and methods for the automatic determination of the anomaly ratio must be further investigated. Herein we perform a simple comparison of the algorithms under the following three scenarios:

- the parameter ν matches exactly the anomaly ratio,
- the parameter ν is fixed whereas the anomaly ratio varies,
- the ratio of anomalies is fixed and the parameter ν varies.

Under the scenario that ν matches the anomaly ratio it is assumed that perfect information about the anomaly ratio is available. One would expect that the parameter ν can tune both kinds of one-class SVM to the specific anomaly ratio. This, however, does not happen, as can be seen from Fig. 6. One can observe that the performance of both formulations noticeably degrades with the increasing anomaly ratio. We believe that the reason for this lies in the data-dependent normalization of the features: since the features are normalized with respect to the mean, having a larger anomaly ratio shifts the mean towards the anomalies, which leads to worse separability of the normal data and the anomalies.

Under the scenario with fixed ν it is assumed that no information about the anomaly ratio is available, and that this parameter is simply set by the user to some arbitrary value. As one can see from Fig. 7, the performance of both formulations of one-class SVM degrades with increasing anomaly ratio similarly to the scenario with ν matching the true anomaly ratio. Notice that the spread in the accuracy, as the anomaly ratio increases, is similar for both scenarios. This implies that, at least for the data-dependent normalization as used in the current experiments, setting the parameter ν to a fixed value is a reasonable strategy.

Under the scenario with fixed anomaly ratio and the varying ν we investigate what impact the adjustment of the parameter has on the same dataset. As it can be seen from Fig. 8, varying the parameter only has an impact on the sphere one-class SVM, the best accuracy

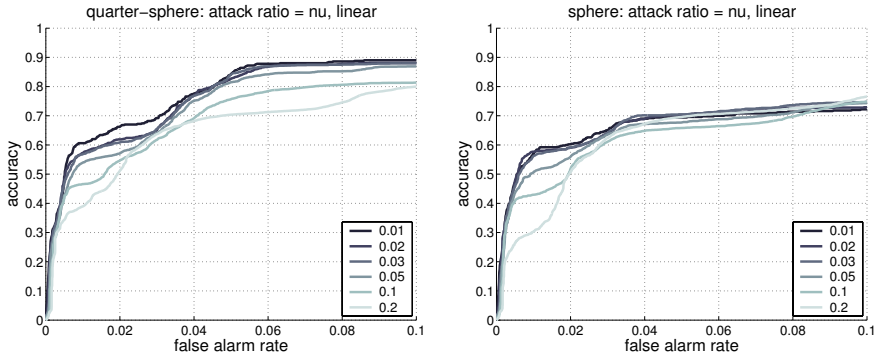


Figure 6: Impact of the anomaly ratio on the accuracy of the sphere and quarter-sphere SVM: anomaly ratio is equal to ν.

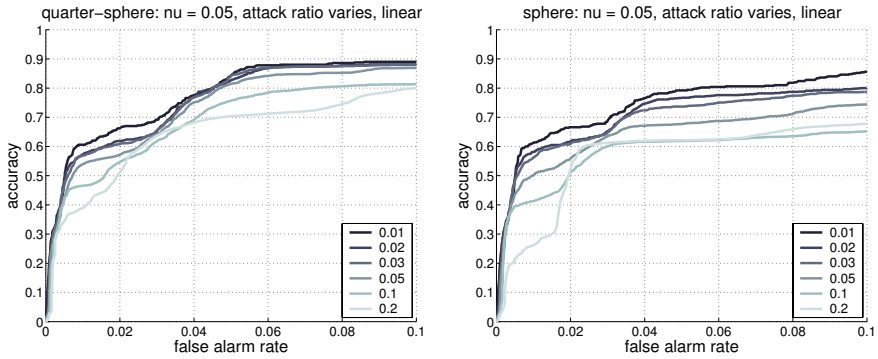


Figure 7: Impact of the anomaly ratio on the accuracy of the sphere and quarter-sphere SVM: ν is fixed at 0.05, anomaly ratio varies.

achieved on the higher values. *The parameter ν does not have any impact on the accuracy of the quarter-sphere one-class SVM*.

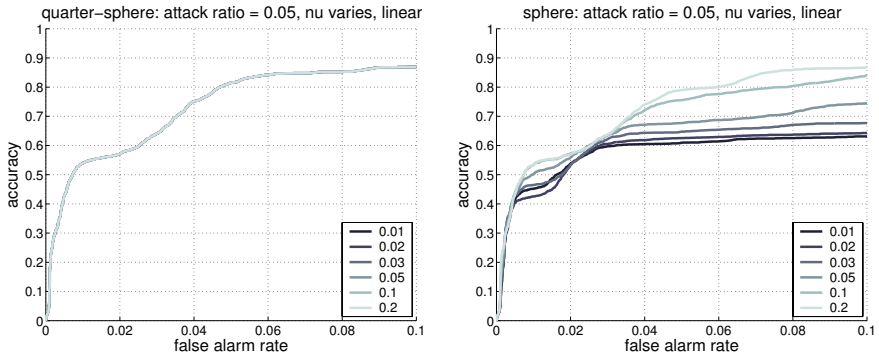


Figure 8: Impact of the anomaly ratio on the accuracy of the sphere and quarter-sphere SVM: anomaly ratio is fixed at 5%, ν varies.

4 Conclusions and future work

We have presented a novel one-class SVM formulation, the quarter-sphere SVM, that is optimized for non-negative attributes with one-sided distribution. Such data is frequently used in intrusion detection systems. The one-class SVM formulations previously applied in the context of unsupervised anomaly detection do not account for non-negativity and one-sidedness; as a result, they can potentially detect very common patterns, their attributes close to the origin, as anomalies. The quarter-sphere SVM avoids this problem by aligning the center of the sphere fitted to the data with the "center of mass" of the data in feature space.

Our experiments conducted on the KDDCup 1999 dataset demonstrate significantly better accuracy of the quarter-sphere SVM in comparison with the previous, sphere or plane, formulations. Especially noteworthy is the advantage of the new algorithm at low false alarm rates.

We have also investigated the behavior of one-class SVM as a function of attack rate. It is shown that the accuracy of all three formulations of one-class SVM considered here degrades with the growing percentage of attacks, contrary to the expectation that the parameter ν of one-class SVM, if properly set, should tune it to the required anomaly rate. We have found that the performance degradation with the perfectly set tuning parameters is essentially the same as when the parameter is set to some arbitrary value. We believe that performance of anomaly detection algorithms on higher anomaly rates should be given special attention in the future work, especially with respect to the data normalization techniques.

Acknowledgements

The authors gratefully acknowledge the funding from the *Bundesministerium für Bildung und Forschung* under the project MIND (FKZ 01-SC40A). We also thank Klaus-Robert Müller and Stefan Harmeling for valuable suggestions and discussions.

References

[BCJ+01] Barbará, D., Couto, J., Jajodia, S., Popyack, L., und Wu, N.: ADAM: Detecting intrusions by data mining. In: *Proc. IEEE Workshop on Information Assurance and Security*. S. 11–16. 2001.

[De87] Denning, D.: An intrusion-detection model. *IEEE Transactions on Software Engineering*. 13:222–232. 1987.

[DR90] Dowell, C. und Ramstedt, P.: The ComputerWatch data reduction tool. In: *Proc. 13th National Computer Security Conference,*. S. 99–108. 1990.

[EAP+02] Eskin, E., Arnold, A., Prerau, M., Portnoy, L., und Stolfo, S.: *Applications of Data Mining in Computer Security*. chapter A geometric framework for unsupervised anomaly detection: detecting intrusions in unlabeled data. Kluwer. 2002.

[JLA+93] Jagannathan, R., Lunt, T. F., Anderson, D., Dodd, C., Gilham, F., Jalali, C., Javitz, H. S., Neumann, P. G., Tamaru, A., und Valdes, A.: Next-generation intrusion detection expert system (NIDES). Technical report. Computer Science Laboratory, SRI International. 1993.

[LEK+03] Lazarevic, A., Ertoz, L., Kumar, V., Ozgur, A., und Srivastava, J.: A comparative study of anomaly detection schemes in network intrusion detection,. In: *Proc. SIAM Conf. Data Mining*. 2003.

[LV92] Liepins, G. und Vaccaro, H.: Intrusion detection: its role and validation. *Computers and Security,*. 11(4):347–355. 1992.

[NWY02] Noel, S., Wijesekera, D., und Youman, C.: *Applications of Data Mining in Computer Security*. chapter Modern intrusion detection, data mining, and degrees of attack guilt. Kluwer. 2002.

[PES01] Portnoy, L., Eskin, E., und Stolfo, S.: Intrusion detection with unlabeled data using clustering. In: *Proc. ACM CSS Workshop on Data Mining Applied to Security*. 2001.

[PN97] Porras, P. A. und Neumann, P. G.: Emerald: event monitoring enabling responses to anomalous live disturbances. In: *Proc. National Information Systems Security Conference*. S. 353–365. 1997.

[SMB+99] Schölkopf, B., Mika, S., Burges, C., Knirsch, P., Müller, K.-R., Rätsch, G., und Smola, A.: Input space vs. feature space in kernel-based methods. *IEEE Transactions on Neural Networks*. 10(5):1000–1017. September 1999.

[SPST+01] Schölkopf, B., Platt, J., Shawe-Taylor, J., Smola, A., und Williamson, R.: Estimating the support of a high-dimensional distribution. *Neural Computation*. 13(7):1443–1471. 2001.

[SS02] Schölkopf, B. und Smola, A.: *Learning with Kernels*. MIT Press. Cambridge, MA. 2002.

[SSM98] Schölkopf, B., Smola, A., und Müller, K.-R.: Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*. 10:1299–1319. 1998.

[TD99] Tax, D. und Duin, R.: Data domain description by support vectors. In: Verleysen, M. (Hrsg.), *Proc. ESANN*. S. 251–256. Brussels. 1999. D. Facto Press.

[Va95] Vapnik, V.: *The nature of statistical learning theory*. Springer Verlag. New York. 1995.

[Va98] Vapnik, V.: *Statistical Learning Theory*. Wiley. New York. 1998.

[VS00] Valdes, A. und Skinner, K.: Adaptive, model-based monitoring for cyber attack detection. In: *Proc. RAID 2000*. S. 80–92. 2000.

[WFP99] Warrender, C., Forrest, S., und Perlmutter, B.: Detecting intrusions using system calls: alternative data methods. In: *Proc. IEEE Symposium on Security and Privacy*. S. 133–145. 1999.

Sensors for Detection of Misbehaving Nodes in MANETs

Frank Kargl Andreas Klenk Michael Weber
Stefan Schlott
Department of Multimedia Computing
University of Ulm

Abstract: The fact that security is a critical problem when implementing mobile ad hoc networks (MANETs) is widely acknowledged. One of the different kinds of misbehavior a node may exhibit is selfishness. A selfish node wants to preserve its resources while using the services of others and consuming their resources. One way of preventing selfishness in a MANET is a detection and exclusion mechanism. In this paper, we focus on the detection and present different kinds of sensors that will find selfish nodes. First we present simulations that show the negative effects which selfish nodes cause in MANET. In the related work section we will analyze the detection mechanisms proposed by others. Our new detection mechanisms that we describe in this paper are called *activity-based overhearing*, *iterative probing*, and *unambiguous probing*. Simulation-based analysis of these mechanisms show that they are highly effective and can reliably detect a multitude of selfish behaviors.

1 Selfish nodes in MANETs

Mobile ad hoc networks (MANETs) rely on the cooperation of all participating nodes. The more nodes cooperate to transfer traffic, the more powerful a MANET gets. But supporting a MANET is a cost-intensive activity for a mobile node. Detecting routes and forwarding packets consumes local CPU time, memory, network-bandwidth, and last but not least energy. Therefore there is a strong motivation for a node to deny packet forwarding to others while at the same time using their services to deliver own data.

In table 1, we analyze different possibilities for a selfish node to save its own resources in a MANET based on the DSR routing protocol [JMHJ03, Pe01]. It uses the attack-tree notation proposed by Bruce Schneier [Sc99] that allows the analysis of different ways how an attacker can achieve his goal. Alternatives to reach a certain goal are denoted by OR, multiple steps that are necessary to reach a goal are denoted by AND. Using the numbers in the table, we can easily describe different attacks. For example, attack 3.1 stands for ”Drop data packets”.

Whereas most of the attacks based on manipulations of routing data can be detected by the use of a secure routing protocol like Ariadne [HPJ02], SRP [PH02a, PH02b, PH02c, PHS02, PH03], ARAN [SDL$^+$02], or SAODV [Gu02, GA02], there remain two attacks in the attack tree that cannot be detected this easily. When nodes simply drop packets (case 1.1 and 3.1 in the attack tree), all of the secure routing protocols fail as they focus only on

Attack tree: Save own resources										
OR	1.	Do not participate in routing								
	OR	1.	Do not relay routing data							
		OR	1.	Do not relay route requests						
			2.	Do not relay route replies						
			3.	Set hop limit or TTL value in route request/reply to smallest possible value						
		2.	Modify routing data/topology							
		OR	1.	Modify route request						
			OR	1.	Insert additional hops					
				2.	Modify route reply					
				OR	1.	Replace own ID in returned route with detour leading through neighboring nodes				
					2.	Return completely wrong route, provoking RERR and salvaging				
					3.	Insert additional hops				
					4.	Declare own ID in source route as external				
	2.	Stop participation in current route								
	AND	1.	Provoke route error							
		OR	1.	Create arbitrary RERR messages						
			2.	Do not send ACK messages (causing RERRs in other nodes)						
		2.	Do not participate in following route request (A.1)							
	3.	Do not relay data packets								
	OR	1.	Drop data packets							
		2.	Set hop limit/TTL to 0/1 (causing a RERR)							

Table 1: Attack Tree: Save own resources

Parameter	Value
Number of Nodes	50
Area X (m)	1500
Area Y (m)	300
Transmission Range Radius (m)	250
Traffic Model	cbr
Sending rate (packets/s)	4.0
Max. number of connections	20
Packet size (byte)	512
Simulation time (s)	900

Table 2: Simulation parameters

the detection of modifications to routing data but not on the concealment of existing links.

We have done a number of simulations that show how this behavior affects a MANET. The simulations were done using ns-2.1b8a and the DSR routing protocol. The scenario included 50 nodes moving in an area of 1500x300m according to the random waypoint model at speeds of $1\frac{m}{s}$ and $20\frac{m}{s}$ with no pause time. Twenty of the nodes were CBR sources sending 4 packets per second. Details of the simulation parameters are given in table 2. These parameters are used for all following simulations.

Figure 1 shows the results of these simulations. We have varied the number of selfish nodes from 0 to 50 (the total number of nodes in the network). It is obvious that the number of selfish nodes has a significant effect on the rate of packets that are successfully delivered in the network. Further the movement rate has a clear effect. The faster nodes move, the lower the delivery ratio becomes. Finally we see that at lower speeds nodes of case 3.1 are more detrimental to the network than those of type case 1.1, whereas at higher speeds there are no big differences.

What explanations can be found for this? When the number of case-1.1 nodes rises in a

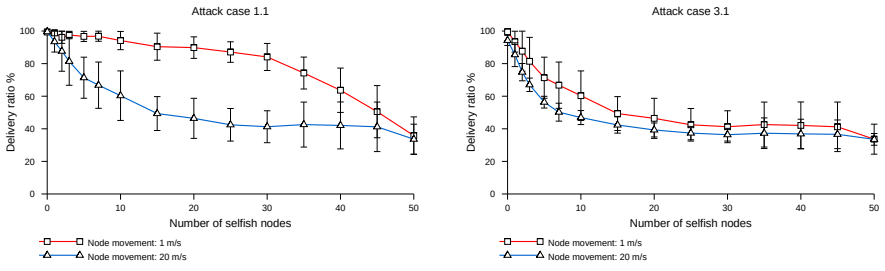


Figure 1: Selfish attack simulation

network, there are less nodes available for building up routes. So if no alternative route can be established, there is no route to the destination which means that packets have to be discarded. That reduces the delivery rate. When movement speed rises, the delivery ratio also diminishes as the network in general gets more fragile. But the network still has a reasonable chance of routing around the selfish nodes. This changes with type case-3.1. Here the nodes behave correctly during the route discovery phase. Thus they can be included in regular routes, but then they start to drop all packets. This isn't detected by DSR and no countermeasures are taken. So at a movement speed of $20\frac{m}{s}$ only 10% of the selfish nodes push the probability of a successful packet delivery below 50%.

Our simulations with AODV have revealed a similar behavior. This demonstrates clearly that an effective protection against selfish and malicious nodes is absolutely mandatory for ad hoc networks.

2 Motivation vs. Detection & Exclusion

There are two approaches of dealing with selfish nodes. The first approach tries to give a motivation for participating in the network function. A typical system representing this approach is Nuglets by Hubeaux et al. [BH01, BH03]. The authors suggest to introduce a virtual currency called Nuglets that is earned by relaying foreign traffic and spent by sending own traffic. The major drawback of this approach is the demand for trusted hardware to secure the currency. There are arguments that tamper-resistant devices in general might be next to impossible to be realized [AK96, AK97]. A similar approach without the need of tamper-proof hardware has been suggested by Zhong et al. in [ZCY03].

Most of the existing work in this field concentrates on the second approach: detecting and excluding misbehaving nodes. The first to propose a solution to the problem of selfish (or as they call it "misbehaving") nodes in an ad hoc network were Marti, Giuli, Lai, and Baker in [MGLB00]. Their system uses a watchdog that monitors the neighboring nodes to check if they actually relay the data the way they should do. Then a component called pathrater will try to prevent paths which contain such misbehaving nodes. As they indicate in their paper, their detection mechanism has a number of severe drawbacks. Relying only on overhearing transmissions in promiscuous mode may fail due to a number of reasons. In case of sensor failure, nodes may be falsely accused of misbehavior. The second drawback is that selfish nodes profit from being recognized as misbehaving. The paths in the network are then routed around them, but there is no exclusion from service. We will later present more advanced sensors that will allow a better detection of selfish nodes.

In [ZL00, ZLH03], the authors describe a distributed intrusion detection system (IDS) for MANETs that consists of the local components "data collection", "detection"and "response", and of the global components "cooperative detection"and "global response". Their architecture is very promising and similar to the one we use in our project, but they neglect the aspect how their local data collection should find out on incidents like dropped packets, concealed links, etc.

Another system is the "Collaborative Reputation Mechanism"or CORE [MM, MM02]. It

is similar to the distributed IDS by Zhang et al. and consists of local observations that are combined and distributed to calculate a reputation value for each node. Based on this reputation, nodes are allowed to participate in the network or are excluded. In their work, the authors specify in detail how the different nodes should cooperate to combine the local reputation values to a global reputation and how they should react to negative reputations of nodes. For the actual detection of selfish nodes, they only refer to the work of Marti.

A similar approach is conducted by Buchegger et al. with their system called CONFIDANT [BB02a, BB02b]. Again, they only marginally describe their detection mechanism and rely mostly on promiscuous overhearing.

3 MobIDS

We have developed a *Mob*ile *I*ntrusion *D*etection *S*ystem (*MobIDS*) that has a similar structure like some of the systems mentioned above. As you can see in figure 2, different sensors collect data from the network.

MobIDS is embedded in a secure system framework called *Security Architecture for Mobile Ad hoc Networks* or *SAM*. SAM also includes mechanisms for

- *uniquely identifying nodes* within the network. Nodes cannot change their identity or create additional identities to fool sensors.
- *secure routing* with a special routing protocol called *Secure Dynamic Source Routing* (*SDSR*). Using SDSR nodes cannot alter routing data, so MobIDS does not need to detect attacks that are based on forging of valid routing data. Therefore MobIDS sensors can concentrate on the detection of selfish behavior which in turn cannot be detected by a routing protocol.
- *exchange of symmetric keys* used by some of the MobIDS sensors. This key exchange is integrated into the SDSR routing protocol, so whenever a valid route is established between two nodes, the necessary keys are exchanged in an efficient way.

Due to limited space, we cannot describe the full SAM system here. See [Ka03] for a detailed and complete description. For the rest of this text, we focus on MobIDS and its detection sensors. As you can see, data from the secure routing protocol SDSR is can also taken as input to some sensors.

The sensors generate *observations*. $\sigma_n^s \in [-1;1]$ represents the n^{th} observation of sensor s. Positive values represent a positive behavior whereas a negative value expresses non-cooperative behavior. All local observations of a node k_i and a sensor s regarding another node k_j at time t lead to a *sensor rating* $r_{k_i}^t(k_j|s) \in [-1;1]$:

$$r_{k_i}^t(k_j|s) = \left(\sum_{\forall n} \rho(t,t_n) \cdot \sigma_n^s \right) / n$$

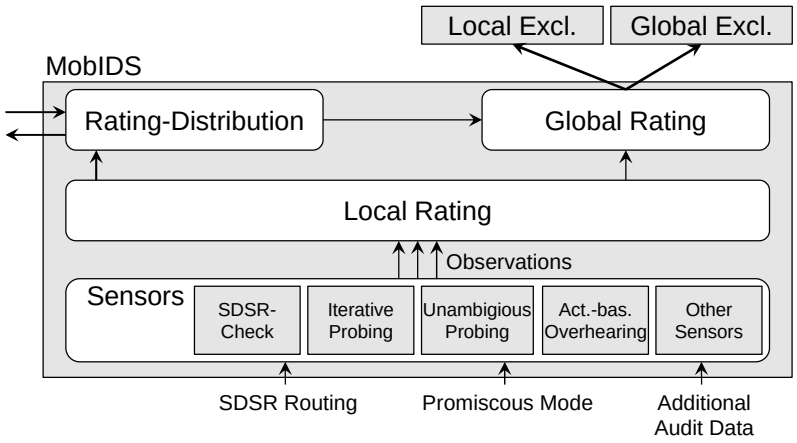


Figure 2: Overview of MobIDS

where

$$\rho(t, t_n) = 1 - \left(\frac{t - t_n}{T}\right)^x$$

t_n is the time when a specific observation σ_n^s was made. The function ρ makes older observations less important than newer once, observations older than $t - T$ are ignored and can be discarded. x controls the degradation of older observations.

Finally all sensor ratings $r_{k_i}^t(k_j|s)$ are combined into a *local rating* $r_{k_i}^t(k_j) \in [-1; 1]$ that expresses the judgment of node k_i regarding node k_j at time t:

$$r_{k_i}^t(k_j) = \sum_{\forall s} w_s \cdot r_{k_i}^t(k_j|s)$$

w_s is a weighting factor which represents the credibility of different sensors. Very reliable sensors receive a higher weight than less reliable ones.

The local ratings are then *distributed* to neighboring nodes by flooding them periodically in a certain diameter surrounding a node. A node averages all received local ratings (including his own) which results in the *global rating* $gr_{k_i}^t(k_j)$.

As the initial observations are often based on statistical sensors, no node can prove that his rating is actually accurate. So when distributing ratings, these are signed by private keys of each node, but no further attempt is made to prove the credibility of a rating. Instead, global ratings are only accepted when at least N nodes have contributed to the rating. This prevents alliances of less than N nodes from excluding other nodes from the network.

Based on the global rating, nodes may be excluded from the current network. MobIDS defines different thresholds t_t, t_e and t_r, where t_e is the *exclusion threshold*. If the rating

of a node k_i regarding a node k_j sinks below t_e, k_i will invalidate all routes containing k_j and will ignore all packets related to k_j. After some time, old negative observations will expire, so the rating of k_j will eventually increase again. As soon as the global rating exceeds the *rehabilitation threshold* t_r, k_j will be serviced again.

There is one problem: As the distribution process takes some time to deliver the local ratings to all nodes, the global ratings of different nodes regarding k_j may differ by a certain amount ϵ. If $r^t_{k_i}(k_j) < t_e < r^t_{k_l}(k_j)$ then node k_i will stop servicing k_j whereas k_l will still regard k_j as a cooperating node. So when k_i stops forwarding packets to k_j, sensors of k_l may detect this and punish k_i.

Therefore the system contains a third threshold t_t, the so-called *tolerance threshold*, where $t_e < t_r < t_t - \epsilon$. When $r^t_{k_l}(k_j)$ is below t_t, k_l will tolerate any node to deny service to k_j without deducing negative ratings from this.

In addition, the security architecture contains a mechanism that allows global exclusion of nodes from MANETs by invalidating their cryptographic identity. But this is outside the scope of this paper.

Another question is how the different thresholds should be chosen. Up to now we have adjusted them manually for each type of simulation by running different simulations and testing the results. In the final section we will outline future research on how to adjust them automatically.

It is obvious that without good sensors all the following steps (local and global rating, exclusion) will fail to deliver good results. So the rest of the paper focuses on this aspect of MobIDS.

4 Advanced Sensors

4.1 Activity-Based Overhearing

We already mentioned that there are a number of problems when a node wants to determine whether another node actually relays its packet by listening in promiscuous mode for the transmission. In promiscuous mode, a wireless network interface consumes more power than in standard mode. Furthermore, there are a lot of cases where a relay node actually forwards a packet but the node overhearing the relay node's activity will fail to realize that. If e.g. the overhearing node is currently transmitting or receiving data in a IEEE 802.11 network at a lower wirespeed (e.g. 5.5 or 2 Mbps) then it will not be able to capture transmissions that happen at other speeds. Other problems include collisions, cooperating selfish nodes, and many more.

In our new activity-based overhearing mechanism a node also tries to overhear forwarding of data packets by its next hop. But this time it only triggers an alarm when it recently saw normal traffic from this node but then detects no forwarding activity. Using this mechanism we can improve the detection accuracy significantly. Furthermore, our architecture introduces a threshold. The monitoring node will only trigger an alarm when it detects a

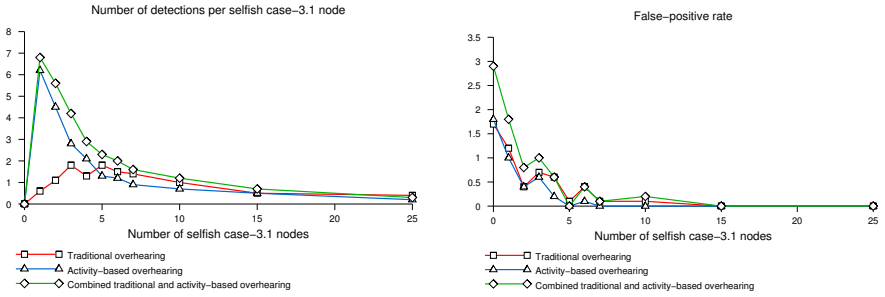


Figure 3: Traditional vs. Activity-based Overhearing at $1m/s$

certain number of packets being dropped within a certain timeframe.

Figure 3 shows simulation results at a movement speed of $1m/s$. It verifies the better performance of the activity-based overhearing mechanism. The left graph shows the detection rate of MobIDS in the presence of a specific number of selfish nodes that operate according to case 3.1 in the attack tree (forward routing traffic, but drop subsequent data traffic). All values are taken as the average of 10 different simulation runs. Lets assume a network with 2 selfish nodes. Then each of the two nodes is (on average) detected by 1.1 monitoring nodes using traditional overhearing. When we use activity-based overhearing there are 4.5 nodes detecting each selfish node. When the number of selfish nodes gets higher (from 5 to 10), the traditional overhearing performs better than activity-based overhearing. We can use both approaches when the results of both sensors are combined. This delivers highly acceptable results.

When the number of selfish nodes become large[1], detection rates get really bad. Only one or two nodes will detect a selfish node during the average simulation run. This is partially because we assume that selfish nodes do not act as sensors anymore. So in case of 10 selfish nodes you also have to take into consideration that 20% of the sensors are gone. In order to get good results here, we need to combine the overhearing sensor with other sensors like the probing sensor described later.

The graph on the right in figure 3 shows the false-positives that the overhearing sensors produce. Here is significant that these values are always low compared to the correct positive identifications of selfish nodes. In MobIDS, a node is excluded from the network only if a number of different nodes agree on it being selfish or malicious. So when only one node has a false-positive this has no negative effects on the detected node.

Simulations at $20m/s$ (figure 4) show that the detection rate of the activity-based and combined overhearing even increases at higher speeds. This is due to the larger number of routing protocol packets that circulate in the network. This enables the activity detector to predict more precisely whether another host is still in communication range.

[1] more than 10 or 20% of all nodes!

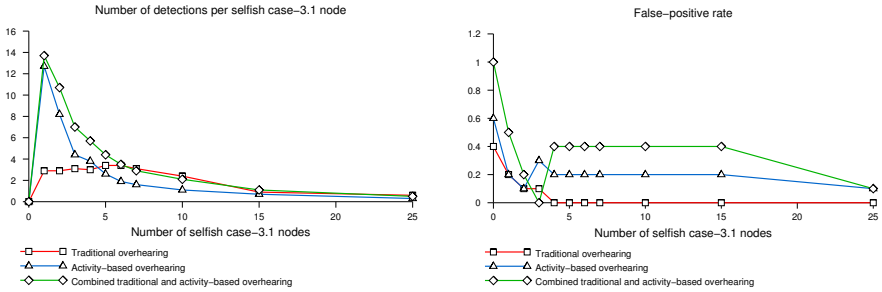


Figure 4: Traditional vs. Activity-based Overhearing at $20m/s$

4.2 Iterative Probing

In [AHNRR02] the authors describe a mechanism called *probing* to detect selfish or malicious nodes in a MANET route from source S to destination D. They use onion encryption to embed a probe command for a specific node X into normal data packets. When X decrypts its onion layer, it will find this command and send back an acknowledge packet to the source. As soon as an acknowledge is missing, S starts a binary search in the path to find out, where packets are being dropped. S simply sends probes to the selected nodes and waits for their replies. Figure 5 shows the binary search after which we call it *binary probing*.

This approach has a number of drawbacks. The onion encryption is very expensive, as each packet has to be encrypted multiple times depending on the path length. Furthermore each node has to decrypt the packet once and each packet has to be acknowledged explicitly by the recipient D.

But there is an even more severe problem. There is no reliable detection of the node dropping packets. When a selfish node gets a probe packet it can choose to forward packets for a limited time (until the probe is over) and then continue to drop packets. Even worse, depending on how the probing is realized, it may even be able to selectively drop probe packets destined for another host. This host will not acknowledge the probe and will be marked as hostile.

In our mechanism, that we call iterative probing, we use a different approach. Like in [AHNRR02] we assume that a source S has established a secret key k_{SX_i} with each node X_i $(i = 1...n-1)$ in its path to a destination X_n. There is a command field C included in the packet header that *may* contain a node id X_i which is encrypted by k_{SX_i}, so $C = enc_{k_{SX_i}}(X_i)$ $(i = 1...n)$. Otherwise the field contains a random number. Each intermediate node X_j will now try to decrypt C. If the result is its node ID, it will send an (encrypted) probe reply packet back to S, otherwise it will process the packet as usual. So S has to encrypt only a small portion of the packet and it has to do so only once (compared to the onion-encryption approach) Intermediary nodes will only have to decrypt the small

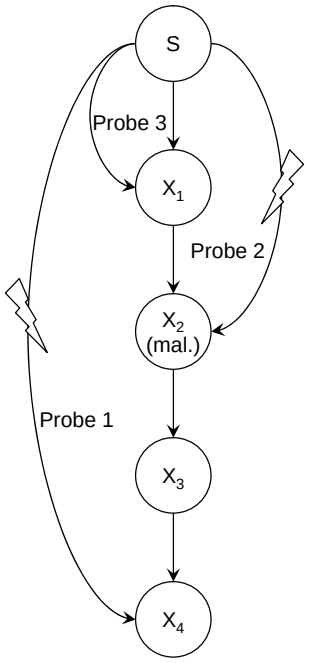


Figure 5: Binary Probing

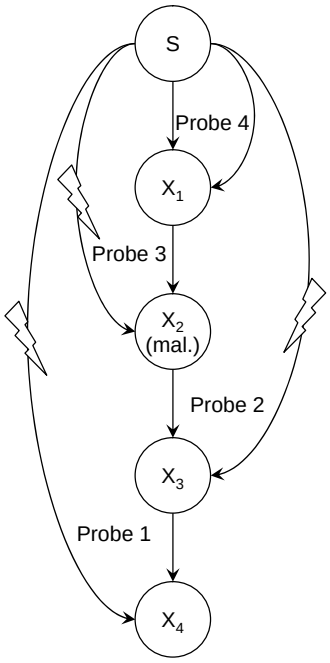


Figure 6: Iterative Probing

command field and not the whole packet.

In normal operation (that is while it receives packets from X_n as a reply to the packets it sent to X_n) there is no need for probing. But when S hasn't received a packet from X_n for a certain amount of time t, it will sent a probe packet to X_n. If there is no reply within a certain timeout, it will sent a probe to X_{n-1} and so on until it receives a reply from a node or reaches X_1. This is called *iterative probing* and shown in figure 6.

Iterative Probing has one advantage over binary probing: a selfish node only knows of an ongoing probing when it is its turn to answer a probe. So it is not able to blame any nodes on an arbitrary position later in the path by selectively filtering out or forwarding probe packets. Instead, there are only two possibilities: it can reply to the probe or it can discard it. All later probe packets are sent to nodes earlier in the path and cannot be manipulated any more. But there is still one problem remaining.

Let X_j be the first node from which S receives an acknowledge. There are two possibilities now. Either is X_{j+1} the selfish node dropping all packets. In this case X_{j+1} is also dropping probe packets and X_j is working properly. Or X_j is the selfish node dropping packets. But before dropping a packet, X_j checks if it is a probe addressed to himself. In order to be harder to detect, X_j will then reply to the probe.

So albeit the iterative probing sensor is harder to fail than the binary probing, it cannot distinguish which of the two nodes is actually the malicious one. We call this problem the

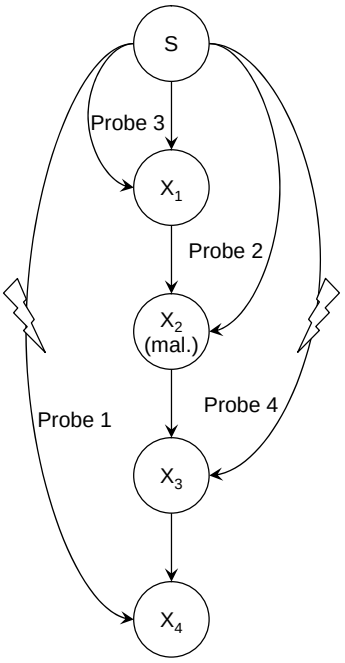


Figure 7: X_2 answers probes: possible selfish nodes $\{X_2, X_3\}$

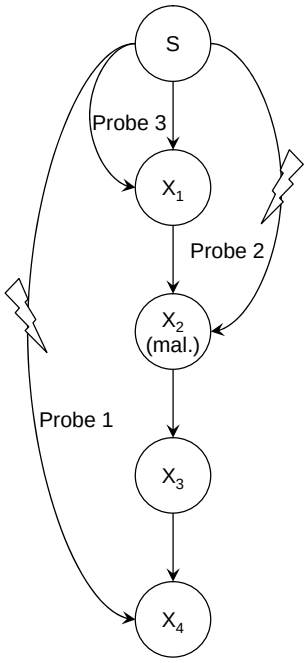


Figure 8: X_2 answers not: possible selfish nodes $\{X_1, X_2\}$

probing dilemma. In the next section we will present an approach to prevent this. But first we give an analysis of the iterative probing.

Figure 9 (left side) shows the simulation results for the iterative probing sensor facing the standard adversary – a selfish case-3.1 node. Even for 10 selfish nodes we still have an average of 4.9 nodes detecting each selfish node. The false-positives are negligibly low. So probing is an efficient way of detecting selfish nodes.

4.3 Unambiguous-Probing

As indicated above, the probing techniques described so far face a serious problem: probing can not unambiguously detect a selfish node. Even worse, the standard probing described in [AHNRR02] allows a malicious node to make another arbitrary node look selfish. Our iterative-probing can narrow the potential adversary nodes down to two nodes. In order to clearly identify one of these nodes as being responsible for the dropped data packets, we can combine the iterative probing with overhearing. Let X_j and X_{j+1} be the nodes that are suspicious of dropping packets like described above. Now we can verify if X_j is dropping the packet by asking X_{j-1} to check if he can overhear the forwarding of a

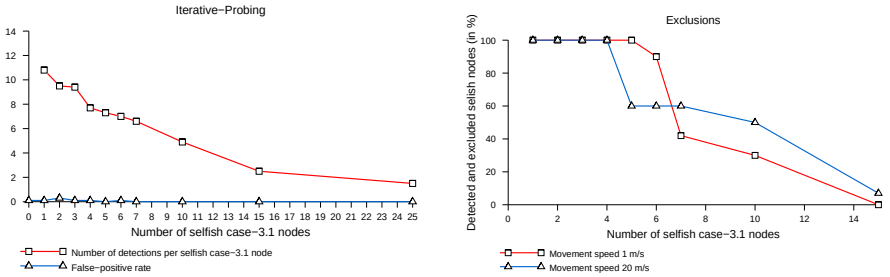


Figure 9: Iterative Probing and exclusion of selfish nodes

following probe packet by node X_j. If this probe fails and X_{j-1} can't hear X_j forwarding the packet, then it is very likely that X_j is dropping the packets, otherwise X_{j+1} is the node responsible for the packet drop.

4.4 Overall Detection Rate

MobIDS combines all presented sensors in order to make a decision on excluding nodes from the network. Our simulation results show that the detection of misbehaving nodes is very accurate and we have practically no false accusations. Figure 9 (right side) shows the percentage of discovered and excluded selfish nodes at different movement speeds. In this scenario, three different nodes were needed to detect another node as selfish in order to exclude it from the network. In the simulations, we used combined-overhearing, unambiguous-probing and route-request scanning sensors in parallel. The last sensor was not presented here due to space limitations. It takes information from the routing protocol and detects nodes that are not forwarding route requests properly. As you can see, up to about 5 selfish nodes, all were detected and excluded reliably. At around 10 selfish nodes, detection rate drops below 50%. As we have already discussed in the section on overhearing sensors, 10 selfish nodes are actually 20% of all nodes. As these nodes do not work as sensors, i.e. they do not contribute to the MobIDS detection system, it is obvious that detection results get worse. So MobIDS requires the consensus of a significant majority of all nodes that they want to cooperate and form an ad hoc network. Given that, detection of a few selfish nodes is very reliable.

5 Conclusion and Future Work

As we have seen, the construction of sensors to detect selfish or malicious nodes in ad hoc networks is a complex task. In this paper we have presented a number of different sensors that can detect different kinds of selfish nodes with a good confidence as shown in our

simulation results. If multiple sensors are active in parallel and a selfish node is detected by a number of these sensors, then this is a good indication for excluding the node from the network.

One remaining problem with our current simulations is that all thresholds need to be set manually in order to get good detection results. So in the future we will try to find ways how these values can be set and adjusted automatically during operation. Possible candidates might be some kind of an adjustment algorithm or a self-learning system using neural networks. Furthermore we plan to develop and test additional sensors that will e.g. use topology information from the routing protocol in order to detect selfish nodes.

References

[AHNRR02] Awerbuch, B., Holmer, D., Nita-Rotaru, C., und Rubens, H.: An On-Demand Secure Routing Protocol Resilient to Byzantine Failures. In: *ACM Workshop on Wireless Security (WiSe)*. Atlanta, Georgia. September 2002. auch verfgbar unter `http://citeseer.nj.nec.com/article/awerbuch02demand.html`.

[AK96] Anderson, R. und Kuhn, M.: Tamper Resistance - a Cautionary Note. In: *Proceedings of the Second Usenix Workshop on Electronic Commerce*. S. 1–11. November 1996. auch verfgbar unter `http://citeseer.nj.nec.com/article/anderson96tamper.html`.

[AK97] Anderson, R. und Kuhn, M.: Low cost attacks on tamper resistant devices. In: *IWSP: International Workshop on Security Protocols, LNCS*. 1997. auch verfgbar unter `http://citeseer.nj.nec.com/anderson97low.html`.

[BB02a] Buchegger, S. und Boudec, J.-Y. L.: Nodes Bearing Grudges: Towards Routing Security, Fairness, and Robustness in Mobile Ad Hoc Networks. In: *Proceedings of the Tenth Euromicro Workshop on Parallel, Distributed and Network-based Processing*. S. 403–410. Canary Islands, Spain. January 2002. IEEE Computer Society. http://citeseer.nj.nec.com/article/buchegger02nodes.html.

[BB02b] Buchegger, S. und Boudec, J.-Y. L.: Performance Analysis of the CONFIDANT Protocol: Cooperation Of Nodes - Fairness in Distributed Ad-hoc Networks. In: *Proceedings of IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC)*. Lausanne, CH. June 2002.

[BH01] Buttyn, L. und Hubaux, J.-P.: Nuglets: a Virtual Currency to Stimulate Cooperation in Self-Organized Mobile Ad Hoc Networks. Technical Report DSC/2001/001. EPFL-DI-ICA. January 2001.

[BH03] Buttyn, L. und Hubaux, J.-P.: Stimulating Cooperation in Self-Organizing Mobile Ad Hoc Networks. *ACM/Kluwer Mobile Networks and Applications*. 8(5). October 2003.

[GA02] Guerrero Zapata, M. und Asokan, N.: Securing Ad hoc Routing Protocols. In: *Proceedings of the 2002 ACM Workshop on Wireless Security (WiSe 2002)*. S. 1–10. September 2002. auch verfgbar unter `http://doi.acm.org/10.1145/570681.570682`.

[Gu02] Guerrero Zapata, M.: Secure Ad hoc On-Demand Distance Vector Routing. *ACM Mobile Computing and Communications Review (MC2R)*. 6(3):106–107. July 2002. auch verfgbar unter `http://doi.acm.org/10.1145/581291.581312`.

[HPJ02] Hu, Y.-C., Perrig, A., und Johnson, D. B.: Ariadne: A secure On-Demand Routing Protocol for Ad hoc Networks. In: *Proceedings of MobiCom 2002*. Atlanta, Georgia, USA. September 2002.

[JMHJ03] Johnson, D. B., Maltz, D. A., Hu, Y.-C., und Jetcheva, J. G. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR). `http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-09.txt`. April 2003.

[Ka03] Kargl, F.: *Sicherheit in Mobilen Ad hoc Netzwerken*. PhD thesis. University of Ulm. Ulm, Germany. 2003. also available as `http://medien.informatik.uni-ulm.de/~frank/research/dissertation.pdf`.

[MGLB00] Marti, S., Giuli, T. J., Lai, K., und Baker, M.: Mitigating routing misbehavior in mobile ad hoc networks. In: *Mobile Computing and Networking*. S. 255–265. 2000. auch verfgbar unter `http://citeseer.nj.nec.com/marti00mitigating.html`.

[MM] Michiardi, P. und Molva, R. Prevention of Denial of Service attacks and Selfishness in Mobile Ad Hoc Networks. http://www.eurecom.fr/ michiard/pub/michiardi_adhoc_dos.ps.

[MM02] Michiardi, P. und Molva, R.: A Collaborative Reputation mechanism to enforce node cooperation in Mobile Ad Hoc Networks. In: *Proceedings of the 6th IFIP Communication and Multimedia Security Conference*. Portorosz, Slovenia. September 2002.

[Pe01] Perkins, C. E. (Hrsg.): *Ad Hoc Networking*. Addison-Wesley. 2001.

[PH02a] Papadimitratos, P. und Haas, Z. J.: Secure Routing for Mobile Ad hoc Networks. In: *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*. San Antonio, TX. January 2002. auch verfgbar unter `http://wnl.ece.cornell.edu/Publications/cnds02.pdf`.

[PH02b] Papadimitratos, P. und Haas, Z. J. Secure Routing for Mobile Ad Hoc Networks. Working Session on Security in Wireless Ad Hoc Networks, EPFL, (published in Mobile Computing and Communications Review, vol.6, no.4). June 2002.

[PH02c] Papadimitratos, P. und Haas, Z. J.: Securing Mobile Ad Hoc Networks. In: Ilyas, M. (Hrsg.), *Handbook of Ad Hoc Wireless Networks*. CRC Press. 2002.

[PH03] Papadimitratos, P. und Haas, Z. J.: Secure Link State Routing for Mobile Ad Hoc Networks. In: *IEEE Workshop on Security and Assurance in Ad hoc Networks, in conjunction with the 2003 International Symposium on Applications and the Internet*. Orlando, FL. January 2003.

[PHS02] Papadimitratos, P., Haas, Z. J., und Samar, P. The Secure Routing Protocol (SRP) for Ad Hoc Networks. draft-papadimitratos-secure-routing-protocol-00.txt. December 2002.

[Sc99] Schneier, B.: Modeling security threats. *Dr Dobb's Journal*. December 1999. auch verfgbar unter `http://www.ddj.com/documents/s=896/ddj9912a/9912a.htm`.

[SDL+02] Sanzgiri, K., Dahill, B., Levine, B. N., Shields, C., und Belding-Royer, E. M.: A Secure Routing Protocol for Ad Hoc Networks. In: *Proceedings of 2002 IEEE International Conference on Network Protocols (ICNP)*. November 2002. auch verfgbar unter `http://signl.cs.umass.edu/pubs/aran.icnp02.ps`.

[ZCY03] Zhong, S., Chen, J., und Yang, Y. R.: Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. In: *Proceedings of IEEE Infocom '03*. San Francisco, CA. April 2003.

[ZL00] Zhang, Y. und Lee, W.: Intrusion detection in wireless ad-hoc networks. In: *Mobile Computing and Networking*. S. 275–283. 2000. auch verfgbar unter `http://citeseer.nj.nec.com/zhang00intrusion.html`.

[ZLH03] Zhang, Y., Lee, W., und Huang, Y.-A.: Intrusion Detection Techniques for Mobile Wireless Networks. *to appear in ACM Wireless Networks (WINET)*. 9. 2003. auch verfgbar unter `http://www.wins.hrl.com/people/ygz/papers/winet03.pdf`.

Aktive Strategien zur Schutzzielverletzungerkennung durch eine kontrollierte Machtteilung in der Zugriffskontrollarchitektur

Joerg Abendroth*
Distributed Systems Group
Trinity College, Dublin
Joerg@Abendroth.info

Abstract: Zugangskontrolle und Intrusion Detection werden oft separat behandelt. Zur Erkennung von Schutzzielverletzungen wird hauptsächlich der Datenverkehr eines Netzwerkes ausgewertet - dies geschieht in einer passiven Weise.
Aktive Strategien zum Erkennen von Angriffen sind nur durch neue Zugriffskontrollsysteme und kontrollierte Machtteilung möglich.
In dem vorliegenden Beitrag wird eine neue Kategorisierung von Zugriffskontrollsysteme vorgestellt und insbesondere auf die kontrollierte Machtteilung eingegangen. Es wird gezeigt wie neue aktive Strategien zur Schutzzielverletzungserkennung möglich werden. Diese Strategien erlauben auch den Missbrauch berechtigter Benutzer, die ihre Befugnisse missbrauchen, zu erkennen. Ebenso können nun bekannte Ideen aus der SPAM Bekämpfung in die Zugangskontrolle übernommen werden. Ein Prototyp wurde mittels der ASCap Architektur implementiert und zeigt, dass die vorgestellten Techniken einsetzbar sind.

1 Zugangskontrollsysteme und Schutzzielverletzungserkennung

Auf dem Gebiet der Zugangskontrollsysteme werden oft Neuerungen vorgestellt, welche die Verwaltung oder Anpassung an bestimmte Anwendungen erleichtern. Als Beispiel mag RBAC[1] [FK92] dienen, das die Verwaltung durch die Einführung einer Abstraktionsschicht, den sogenannten Rollen, erleichtert. Doch wenige Ansätze berücksichtigen ein Hauptproblem der Praxis - nämlich, dass fast jede Anwendung ein Einzelsystem darstellt und applikations- oder verwaltungsbereichübergreifende Zugangsregeln nur schwer realisiert werden können. Wenn dieses Problem auf der Ebene der Zugangskontrolle gelöst ist, dann eröffnen sich interessante Möglichkeiten auf der Ebene der Intrusion Detection Systeme (IDS). Verknüpfte Systeme, die es dem IDS erlauben in Zugriffskontrollmechanismen einzugreifen, ermöglichen nicht nur passive, sondern aktive Strategien zur Angriffserkennung.

*Der Author ist durch ein Forschungsstipendium der IONA Technologies PLC unterstützt.
[1] Role Based Access Control

Traditionell gibt es auf dem Gebiet der IDS zwei Hauptrichtungen: Schutzzielverletzungen werden entweder anhand ihrer Angriffssignaturen erkannt [ER88, De87] oder durch eine Abweichung von dem historisch bekannten, angriffsfreien Datenverkehr [LJ88]. Bei beiden Ansätzen wird der Datenverkehr passiv beobachtet. Intrusion Prevention Systeme (IPS) erweitern die bekannten IDS Systeme um ein aktives Element. Diese Systeme können in den Datenverkehr aktiv eingreifen. So kann ein als Gateway eingesetzes IPS System den Datenverkehr von OSI Schicht 7 auswerten und sogar verändern. Ein wichtiger Anwendungsfall sind große Firmennetze, die Softwarepatches nicht zeitnah einspielen können. IPS Systeme können etwaige Sicherheitslücken am Netzübergang stopfen.

Analysiert man die Strategien mit denen die Intrusion Prevention Systeme entscheiden ob ein Angriff vorliegt, dann findet man wieder die gleichen zwei *passiven* Techniken. *Aktive* Systeme hingegen würden nicht nur beobachten, sondern bei Alarmzeichen oder als "routinemäßigen Rundgang" das System so verändern, dass Eindringlinge bemerkt werden. Strategien aktiver IDS befassen sich nicht mit den Anwendungen und deren Schwachstellen, sondern dem Angreifer und dessen Eigenschaften. Für diese aktiven Strategien ist eine Verknüpfung der Zugriffskontrollarchitektur (ZKA) und IDA nötig. Diese verknüpften Systeme müssen Zugriffspolitiken haben, die eine kontrollierte Machtteilung erlauben.

Die in [AJ03] vorgestellte ZKA ist universell in Anwendungen einsetzbar. Ziel dieses Entwurfes war eine flexible und bei Bedarf dynamisch umkonfigurierbare ZKA. Die Zugangsentscheidungsfunktion ist aus Unterelementen, den Regelmodulen, zusammengesetzt und wird durch einen lokalen Vermittleragenten dem jeweiligen Anwendungsdienst zugänglich gemacht. Diese Objektorientierung erlaubt eine kontrollierte, teilweise Machtteilung auf dem Gebiet der Verwaltung. Weitere Elemente der Architektur sind externe Sicherheitsserver, deren Verhalten verschiedenen Nutzungsebenen zugeordnet sein kann. So ist es beispielweise möglich, dass ein externer Sicherheitsserver ein Bindeglied zum IDS darstellt oder sein Verhalten durch deren Einflussnahme ändert.

Ist eine Verknüpfung von IDS und ZKA möglich, können grundlegend neue Strategien in dem IDS eingesetzt werden. Passives Erkennen von auftretenden Alarmzeichen kann durch ein aktives Provozieren derselben ersetzt werden. Strategien können beispielweise in Verdachtsmomenten das Zugangssystem so umkonfigurieren, dass Angriffe deutlicher sichtbar werden. Eine solche Alarmzustandskonfiguration kann jedoch einen Mehraufwand für legitime Systembenutzer bedeuten (zusätzliche Passwortabfragen z.B. per SMS, etc.).

Einen weiteren Vorteil der Verknüpfung von ZKA und IDS stellt das Auslagern der letzteren dar. Ein Security Audit in heutigen Praxis erfolgt überwigend durch einer einmaligen Zustandsüberprüfung. Schwachstellen werden zu dem Zeitpunkt der Prüfung erkannt, doch ein dauerhafter Überwachen wäre nur durch das Installieren schwer überprüfbarer Programme möglich, so dass der Kunde der externen Firma vollständig vertrauen muss. Die vorgestellte Architektur ermöglicht es jedoch, im Rahmen der ZKA die benötigten Daten an den externen Dienstleister zu liefern oder diesem teilweise Einfluss zu gewähren. Diese kontrollierte Abgabe von Verwaltungsmacht stellt eine Neuerung auf dem Gebiet der ZKA dar und ermöglicht auch die bereits erwähnte Verknüpfung mit einem IDS.

In dem vorliegenden Beitrag wird zuerst der Stand der Technik diskutiert. Danach wird in Abschnitt 3 die ASCap Architektur vorgestellt. Abschnitt 4 erörtert eine Kategorisierung der geteilten Zugriffsverwaltung. Diese ist die Grundlage um beispielweise externe Dienstleister teilweise an der Zugriffskontrolle zu beteiligen. Dann stellt Abschnitt 5 Strategiebeispiele vor, die mit der ASCap Architektur möglich werden. Zuletzt fasst Abschnitt 6 die Ergebnisse zusammen und diskutiert weiterführende Forschungsansätze.

2 Verwandte Ansätze

Auf dem Gebiet der Access Control Mechanism wurde die ASCap Architektur von den “active capabilities” [QL96] und dem “proxy principle” [Sh86] beeinflusst. Entgegen den ersten Capability basierten Betriebssystemen [Le84] geht die ASCap Architektur von keinem Referenzmonitorbereich auf dem Clientrechner aus.

Außerdem haben Forschungsergebnisse von Kühnhauser auf dem Gebiet der Metapolitiken [Kü95] die Frage aufgeworfen, wie eine kontrollierte Machtteilung in der Zugangskontrolle möglich wird. Kühnhauser stellt mit [Kü99] gezielt eine Lösung für überschneidende Verwaltungsgebiete vor, in der durch etwaige Konfliktsituationen aufgelöst werden können. Eine Erweiterung um aktive Elemente wäre möglichen, auch wenn dies nicht in den vorliegenden Forschungsbeiträgen behandelt wird.

Aktuelle IDS Systeme sind meist passiver Natur; es werden Datenströme beobachtet und Anzeichen für Verstöße gesucht (z.B. snort [We04d]). Die IPS können aktive in den Datenverkehr eingreifen (z.B. um weitere Einbruchsaktivitäten zu verhindern), doch verwenden auch nur passive Datenauswertungskomponenten. Bei IPS systemen wird die IDS mit der firewall verknüpft. Im Gegensatz dazu verknüpft die ASCap Architektur das IDS System mit der ZKA, was auch proaktives Handeln zulässt.

Die Forschung auf dem Gebiet der aktiven Netzwerke versucht nicht nur die Konfiguration und Wartung von Netzwerken zu vereinfachen, sondern stellt auch systemübergreifende Schnittstellen zur Verfügung. Hess, Jung und Schäfer [JS03] stellen mit FIRDAN eine Architektur vor, die IDS und Reaktion auf Angriffe kombiniert. Sie binden z.B. Honeypots [2] in die Verteidigung ein, indem sie Netzwerkverkehr dorthin umleiten, um mehr Informationen über den Eindringling zu sammeln. Die Komponent von FIRDAN, die Alarme auswertet, kann das System Verändern, falls ein Angreifer den eigentlichen Angriff unter einer hohe Anzahl von Falschalarmen verstecken will. So wird es möglich, einen echten Angriff trotz Täuschungsmanöver zu erkennen. Es bleibt jedoch offen, ob FIRDAN es erlaubt, mittels “Wachrundgänge” noch unentdeckte Eindringlinge zu finden.

[2] Designierte Dummyrechner, die Angreifer anlocken sollen und dann einen Alarm auslösen.

Die IDA Architektur[3] [AOTG99] sucht Anzeichen eines erfolgreichen Einbruchs und entsendet dann einen Agenten, der den Vorfall genauer untersucht. Da die Befugnisse des Agenten weitreichend sein können, wäre eine Interaktion mit der lokalen Zugriffskontrolle denkbar. In dem Fall, in dem die IDA von einem externen Dienstleister betrieben wird, erhält dieser weitreichende Kontrolle über das interne Firmennetz. Die ASCap Architektur erlaubt hier durch Einsatz der kontrollierten Machtteilung ein differenzierteres Vorgehen.

3 Die ASCap Zugriffkontrollarchitektur

Als Basis der vorgestellten Intrusion Detection Strategien dient die ASCap[4] ZKA. In diesem Abschnitt stellen wir die Hauptmerkmale vor, die für ein Verständnis zwingend notwendig sind. Für eine umfassende Beschreibung sei auf [AJ03] verwiesen.

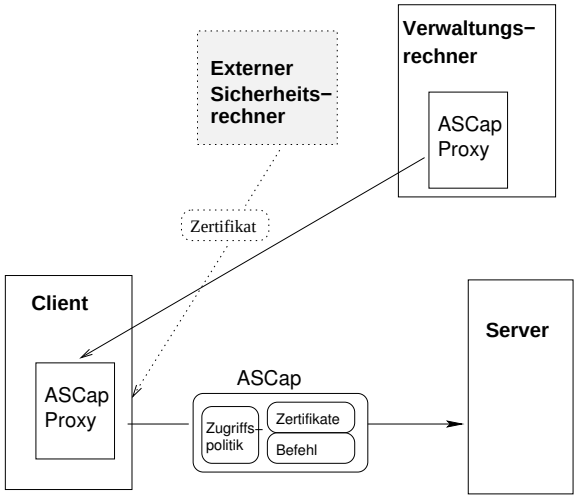


Abbildung 1: Überblick über die ASCap Zugriffskontrollarchitektur

Abbildung 1 zeigt die Komponenten der ASCap Architektur. Die vier rechteckigen Kästen symbolisieren jeweils einen eigenständigen und vernetzten Rechner. Der *Verwaltungsrechner* ist die Administrationskomponente. Sie stellt den *ASCap Proxy* zur Verfügung, der wiederum das Verhalten des Zugriffskontrollsystems bestimmt. Da es möglich ist, dass der ASCap Proxy erst kurz vor dem Versenden konfiguriert wird, ist ein dynamisches Anpassen an verschiedene Szenarien der IDS denkbar. Der *Client* stellt den Benutzerrechner da, der auf den *Server* zugreifen will. Vor Zugriffen kann eine Kommunikation mit einem externen Sicherheitsrechner vorgeschrieben sein. Der externe Sicherheitsrechner kann das Verhalten des Zugriffskontrollsystems erweitern, beispielweise ein Auswerten von Daten

[3] Intrusion Detection Agent System
[4] Active Software Capabilities

(IDS).
In fast allen Fällen liefert der externe Sicherheitsrechner ein Zertifikat zurück. Der ASCap Proxy sammelt die Zertifikate externer Sicherheitsrechner und interner Clientquellen und fasst diese mit der *Zugriffspolitik*[5] zur ASCap zusammen. Interne Clientquellen sind der genaue Zugriffswunsch (aufgeführt als *Befehl* in der Abbildung) und z.B. ein Passwort des Benutzers. Die ASCap beinhaltet somit alle Informationen des Clients, die zum Entscheiden der Zugriffsanfrage nötig sind.
Benötigt ein System beispielweise einen zeitnahen Revokation Service, können weitere externe Sicherheitsanfragen von Serverseite erfolgen. Der Server wertet die Zertifikate mit der angegebenen Zugriffspolitik aus. Diese kann entweder in ausführbarer Form (kryptographisch mittels Signatur geschützt) in der ASCap vorliegen oder per Referenz von einer gesicherten Datenbank auf dem Server geladen werden.

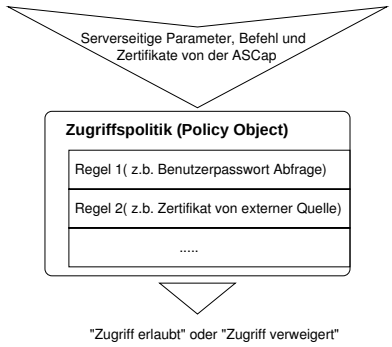


Abbildung 2: Die Zugriffspolitik stellt die Zugriffsentscheidungsfunktion dar.

Die Abbildung 2 zeigt zwei Dinge: Wie die Zugriffsentscheidung durch die Zugriffspolitik erreicht wird und den inneren Aufbau der Zugriffspolitik.
Die Zugriffspolitik stellt quasi die Zugriffsentscheidungsfunktion dar: sie erhält Parameter vom Server, sowie die Zertifikate der ASCap als Eingabeparameter. Die Ausgabe der Zugriffspolitik ist ein “Zugriff erlaubt” oder “Zugriff verweigert”. Auf der Abstraktionsebene der Zugriffspolitikimplementierung besteht eine Zugriffspolitik aus verschiedenen Zugriffsregeln. Jede Regel liefert ein Einzelergebnis, das ähnlich wie bei Kühnhausers Metapolitiken unterschiedlichen Einfluss auf die endgültige Zugriffsentscheidung haben. Ein negatives Ergebnis einer Regel, kann, einen positiven Einfluss auf die Zugriffsentscheidung haben.
Um eine Vielzahl von Zugriffsmodellen zu unterstützen, kann die ASCap Architektur in zwei Modi betrieben werden. Der erste Modus (Referenz-Modus) erlaubt eine erhöhte Sicherheitseinstufung und Bearbeitungsgeschwindigkeit. In der ASCap wird anstelle der vollen Zugriffspolitikimplementierung nur eine Referenz zu einer schon auf dem Server vorhandenen Zugriffspolitik gegeben. Dies schützt nicht nur vor Angriffen auf den kryptographischen Schutz (Signatur) der Zugriffspolitik, sondern erlaubt auch auf dem Server

[5] im engl. policy object

die verwendbaren Zugriffspolitiken zu limitieren. Der zweite Modus, “active capability policy” genannt, erlaubt es, die gesamte Zugriffspolitikimplementierung mittels der ASCap dem Server zu liefern. Dies bietet eine erhöhte Flexibilität und ermöglicht dynamische Zugriffsmodelle. Die Sicherheit des zweiten Modus ist durch die Verwendung kryptographischer Verfahren zur Integritätswahrung der Zugriffspolitik gewährleistet. Jedoch muss der Server der Quelle der Zugriffspolitik vertrauen, falls Quelle und Server keine Verwaltungseinheit bilden. Die Problematik wird in Abschnitt 4 genauer diskutiert.
Die ASCap Architektur wurde ursprünglich konzipiert, um verschiedene Zugriffsmodelle mit derselben Zugriffskontrollarchitektur und Andwendungsschnittstelle darstellen zu können. Dies bedeutet, dass sowohl hoch sichere als auch sehr flexibele Modelle dargestellt werden können. Um die allgemeine Sicherheit und Flexibilität zu zeigen, wurde ein Verhaltensmodell mittels eines Prozesskalkulus modelliert. Der π-Kalkulus erlaubt mittels Verhaltenskongruenzen sowohl verschiedene Zugriffsmodellimplementierung zu vergleichen, als auch zu zeigen, dass die von der ASCap Architektur eingeführten Zusatzelemente (z.B. der ASCap Proxy oder die externen Sicherheitsrechner) nicht die Zugriffsmodelle beeinflussen. Der formale Beweis ist in [Ab03] zu finden.

4 Geteilte Zugriffsverwaltung

In den aktuellen Anwendungen wird die Zugriffsverwaltung meist zentral nach dem Referenzmonitorprinzip mittels eines Domaincontrollers implementiert. Ansätze dezentraler Verwaltungsstrukturen bedienen sich meist der PKI[6] [Va02]. Diese Ansätze können jedoch nur drei der von uns identifizierten Kategorien der geteilten Zugriffsverwaltung darstellen. Die vierte Kategorie, die kontrollierte Machtteilung[7], ist nicht darstellbar. Nachfolgend werden die Kategorien der geteilten Zugriffsverwaltung anhand von Beispielen der ASCap Architektur vorgestellt.

4.1 Kategorie α: Einfache Verwaltung, Intern

Diese Kategorie kann als Basisklasse angesehen werden; es findet keine Machtteilung statt und beide Rechner (Verwaltungsrechner und Server) gehören der gleichen Verwaltungseinheit an. Abbildung 3 zeigt diese Situation. Der Kreis um den Server deutet die Verwaltungseinheit einer Firma an. Es wird davon ausgegangen, dass in einer Verwaltungseinheit die einzelnen Teilkomponenten dem Ziel der Verwaltungseinheit wohlgesonnen sind, d.h. keine Angriffe gegen andere Mitglieder derselben Verwaltungseinheit stattfinden. Nachdem der Client den *ASCap Proxy* erhalten hat, wird angenommen, dass er den Regeln der Verwaltungseinheit folge leistet (dargestellt durch die gestrichelte Linie). Doch auch im

[6] public key infrastructure
[7] im engl. ’partial outsourcing’ genannt

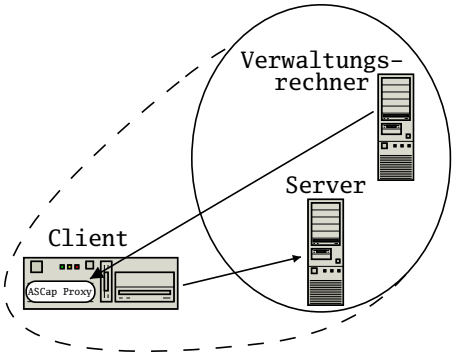


Abbildung 3: Kategorie α: Keine Machtteilung, einfache Verwaltung

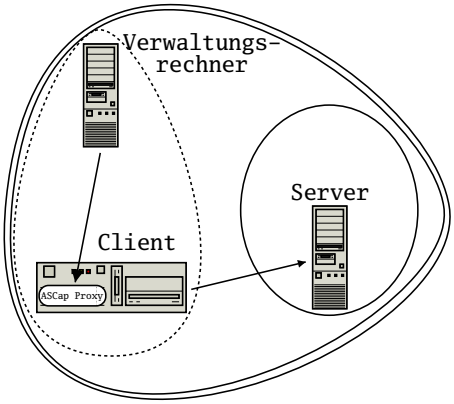


Abbildung 4: Kategorie β: Volle Machtabgabe, beispielweise zu einem externen Dienstleister

Falle eines Verstoßes (z.B. bei Manipulation eines Zertifikates) kann dieser immer noch durch die verwendeten kryptographischen Signaturen erkannt werden.
Sicherheitstechnisch ist diese Kategorie einfach zu analysieren, da es nur eine Verwaltungseinheit gibt. Sowohl Authentifizierung als auch Zugriffsbefugnisse werden von einem einzigen (firmeninternen) Verwaltungsrechner kontrolliert.

Vorteile sind die, die eine zentrale Verwaltung heutzutage bietet.

Nachteile beinhalten die Tatsache, dass die volle Verwaltung von der firmeneigenen IT Abteilung getragen werden muss. Ein externer Dienstleister kann die firmeneigenen Abteilungen beraten, ist jedoch in den Prozess der Zugriffsverwaltung in keiner Weise eingebunden.

Anwendungsbeispiel: Zugangskontrollarchitekturen der heutigen Anwendungen, z.B. ein *Windows Domain Controller*, dessen Passwort nur Angehörigen der eigenen Firma bekannt ist.

Möglichkeiten für die Schutzzielverletzungserkennung: Die interne Firma kann eigene Erkennungsanwendungen einsetzen, die aber separat von der ZKA operieren. Eine externe Firma kann Überwachungsanwendungen im Verwaltungsbereich der internen Firma installieren, doch dies stellt eine zusätzliche Sicherheitsschwachstelle dar.

4.2 Kategorie β: Einfache Verwaltung, Extern

Abbildung 4 zeigt das Gegenbeispiel zu Kategorie α - die Verwaltung wird vollständig von einer externen Verwaltungseinheit betreut. Wiederum stellt der durchgezogene Kreis um den Server die Verwaltungseinheit dar, der ein natürliches Vertrauen zugeordnet ist. Diesmal deutet die gestrichelte Linie an, dass der Client den Regeln des Verwaltungsrechners folgen muss und somit beide eine eigene Verwaltungseinheit darstellen. Der doppelte Kreis weist darauf hin, dass für einen funktionierenden Zugriff der Verwaltungsrechner und Server ein Vertrauensverhältnis haben müssen.
Die Sicherheit hängt vollständig von dem Verhalten des extern administrierten Verwaltungsrechner ab. Dies erlaubt einer Firma, voll von dem Wissen und Möglichkeiten eines externen Dienstleisters zu profitieren, erfordert jedoch, dass diesem Dienstleister auch voll vertraut wird. Zudem muss die auf Erkennen von Schutzzielverletzungen spezialisierte externe Firma alltägliche Arbeiten wie das Zurücksetzen von Passwörtern übernehmen.

Vorteil ist die vollständig ausgelagerte Arbeit.

Nachteil ist die Tatsache, dass der externen Firma vertraut werden muss.

Anwendungsbeispiel: Ähnlich wie in Kategorie α, doch diesmal installiert und beaufsichtigt von der externen Firma.

Möglichkeiten für die Schutzzielverletzungserkennung: Die externe Firma kann ähnlich der internen Firma Dienste zur IDS betreiben. Die extern installierte Überwachungssoftware stellt nun nicht mehr eine so große Sicherheitsschwachstelle dar. Dennoch können auch in dieser Kategorie nur passive Strategien zur Angriffserkennung eingesetzt werden.

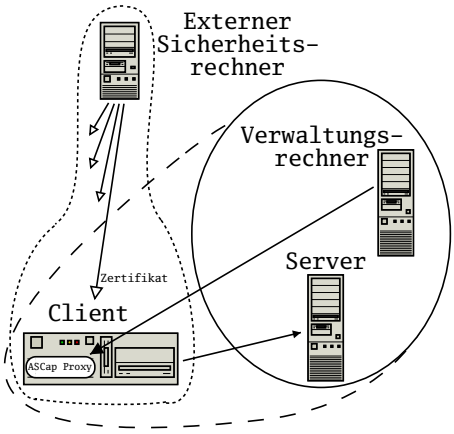


Abbildung 5: Kategorie γ: Machtteilung durch externe Sicherheitsrechner

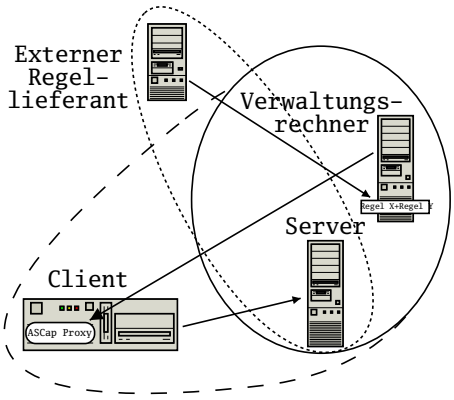


Abbildung 6: Kategorie δ: Kontrollierte Machtteilung durch externe Regellieferanten

4.3 Kategorie γ: Machtteilung durch externe Sicherheitsrechner

In dieser Kategorie wird, wie der Kreis in Abbildung 5 zeigt, die Zugriffsverwaltung von der Firma selbst erledigt. Die Zugriffspolitik erfordert jedoch, dass der Benutzerrechner ein Zertifikat von einem externen Sicherheitsrechner anfordert. Dieser externe Sicherheitsrechner wird von einem externen Dienstleister betrieben und kann Anfragen von einer Vielzahl von Firmen bearbeiten. Da der Benutzerrechner sich ggf. bei dem externen Sicherheitsrechner authentifizieren muss, besteht ein gewisses Vertrauensverhältnis, das durch den gepunkteten Ellipse angedeutet ist. Der erweiterte Kreis zeigt wiederum an, dass der Benutzerrechner den Regeln der Zugriffspolitik folge leisten muss. Es ist kein großer Kreis um den externen Sicherheitsrechner und Server gezogen, da kein zwingendes Vertrauensverhältnis zwischen den Beiden besteht. Dies bedeutet, dass der externe Sicherheitsrechner nicht wahlfrei Zugänge zu dem Server vergeben kann.

Wertet man die Sicherheit dieser Kategorie aus, zeigt sich, dass die interne Firma die endgültige Kontrolle über die Zugangsentscheidungen hat. Dies schließt die Entscheidung ein, welche externen Sicherheitsrechner kontaktiert werden müssen und welchen Einfluss deren Zertifikate erhalten. Aus der Perspektive der Sicherheit kann ein feindlich gesonnener externer Sicherheitsrechner maximal eine zeitweise Dienstverweigerung bewirken.

Schwachstelle dieses Szenarios ist der externe Sicherheitsrechner, der ständig erreichbar sein muss.

Vorteil dieser Kategorie ist die Möglichkeit, dass das Zertifikat der externen Firma erst nach verschiedenen Sicherheitstests ausgehändigt werden kann. Somit kann die Expertise des externen Dienstleisters genutzt werden, ohne ihm Kontrolle über das firmeneigene Netzwerk zu geben.

Anwendungsbeispiel: Diese Kategorie kann durch die externen Sicherheitsrechner der ASCap Architektur implementiert werden. Der Benutzerrechner muss ein oder mehrere Zertifikate abrufen und diese in der ASCap zu dem Server senden.

Möglichkeiten für die Schutzzielverletzungserkennung: Die externe Firma ist so in die Zugriffskontrolle eingebunden, dass sie nur die notwendigen Informationen erhält. Im Vergleich zu einem installierten Überwachungsprogramm kann der Kunde mittels der eingesetzten Zugriffspolitik das zu übermittelte Wissen selbst selektieren. Das IDS kann nun auch übergeordnete Muster erkennen, wie beispielweise einen Benutzer, der eine Vielzahl Zugriffsrechte ansammelt. In einem anderen Szenario händigt ein externer Sicherheitsrechner nur dann ein Zertifikat aus, wenn der Client die volle Transaktionssequenz vorab mitteilt, so dass auch anwendungsspezifische Schutzzielverletzungen erkannt werden, deren Einzelschritte jeweils unbedenklich sind.

Diese Kategorie bietet mit den externen Sicherheitsrechnern ein nützliches Bindeglied zwischen ZKA und IDS.

4.4 Kategorie δ: Kontrollierte Machtteilung durch externe Regellieferanten

Abbildung 6 zeigt die letzte Kategorie. Der externe Regellieferant stellt dem Verwaltungsrechner Einzelregeln zur Verfügung. Der Verwaltungsrechner fügt die verschiedenen Regeln (hier Regel X und Regel Y) zuerst zur vollständigen Zugriffspolitik und dann dem *ASCap Proxy* zusammen. Die gestrichelte Linie um den externen Regellieferanten und den Server zeigt eine natürliche Vertrauensbeziehung an, da der Server nun auch Regeln von dem externen Regellieferanten ausführt.

Schwachstelle: Das System scheint nun eine sicherheitstechnische Schwachstelle mit diesen extern erzeugten ausgeführten Regeln zu haben. Diese ist jedoch minimiert indem die Regelimplementierungen nicht direkt zu dem Server gesendet werden, sondern zuerst den Verwaltungsrechner passieren. Dort können mittels Techniken des Programmbeweises [HR02] und Sandkastentestsystemen ein unbedenkliches Verhalten nachgewiesen werden, bevor die Regel zum Einsatz kommt. Wie bei Kategorie γ kann vom Verwaltungsrechner der Einfluß der extern bereitgestellten Regel kontrolliert werden, so dass keine vollständige Machtabgabe stattfindet.

Vorteil Im Gegensatz zu der vorherigen Kategorie bedient die externe Firma nicht die Clientrechner, sondern die wesentlich kleinere Gruppe der Verwaltungsrechner - dies erlaubt eine bessere Skalierbarkeit.
Die Flexibilität dieser Kategorie kann gezeigt werden, indem der Regellieferant eine Regel bereitstellt, die verlangt einen externen Sicherheitsserver zu kontaktieren - so können alle Kategorie γ Systeme implementiert werden.

Anwendungsbeispiel: Eine Firma betreibt ihren eigenen Verwaltungsrechner, der eine Regelimplementierung von externen Beraterfirmen bezieht. Diese Regeln werden zu einer Zugriffspolitik in einem ASCap Proxy zusammengefasst. Die ASCap ZKA ermöglicht mittels des *active capability* Modus die zeitliche Einführung des neuen ASCap Proxy.

Möglichkeiten für die Schutzzielverletzungserkennung: Ein externer Dienstleister verfolgt die bekannten security Mailing Listen (z.B. CERT[We04a],bugtraq[We04c]). Wird eine neue Sicherheitslücke bekannt, generiert er eine Regel, die etwaige Zugriffsanforderungen auf Angriffsanzeichen prüft. Diese Regel kann den Angriff abwehren ohne einen Informationsfluß zu dem externen Dienstleister zu zulassen.

Weitere Beispiele insbesondere der aktiven Intrusion Detection werden im nachfolgenden Abschnitt besprochen.

4.5 Aktive Intrusion Detection durch eine neue Zugriffskontrollarchitektur

Durch die neu entwickelte *kontrollierte Machtteilung* auf dem Gebiet der ZKA, sowie die nun mögliche Verknüpfung von Zugriffskontrolle und IDS werden neue *aktive* Strategien

möglich. Die aktiven SZVE-Strategien werden Eigenschaften des Benutzers überprüfen, die ihn leichter zuortbar machen, die aber nicht in normalen Arbeitsabläufen auftreten. Ein Beispiel ist ein reguläre Angestellter, der eine Abrechnung eines Projektes erstellen will. Wird ihm mitgeteilt, dass die Daten dieses Projektes zeitweise nicht verfügbar sind, wird er wahrscheinlich andere, nicht onlinegestützte Arbeiten des gleichen Projektes erledigen. Ein Eindringling wird eher dazu übergehen, andere noch verfügbare Projekte auszuspionieren.
Erst eine kontrollierte Machtteilung erlaubt es, externe Expertisefirmen in solchen aktiven IDS-Strategien einzusetzen. Ihnen ist es jedoch nicht möglich, ihre Macht für eigene Einbrüche zu missbrauchen. Dieser Vorteil wird wichtig für große Dienstleister, die zentral Daten sammeln. *Dshield* [We04b] ist ein beliebter Logbuch Auswertungsdienst, der von der hohen Datenmenge profitiert, um unbekannte und schwer erkennbare Angriffssignaturen zu erlernen. Dieser Dienst konnte sich etablieren, da er keine aktiven Elemente in den teilnehmenden Netzen zwingend benötigt. Ebenso erlaubt die Kategorie δ das Einbeziehen externer Dienstleister in die Zugriffskontrolle, ohne ihnen eine pauschale Machtfreiheit über die Systeme zu geben.

5 Strategiebeispiele

Erlaubt die ZKA eine Verknüpfung mit dem IDS wird es möglich, eine neue Klasse von Erkennungsstrategien zu verwenden. In diesem Abschnitt werden drei Gruppen von neuen Erkennungsstrategien zusammen mit der Umsetzung in der ASCap Architektur besprochen:

1. Passives Erkennen kombiniert mit aktivem Eingreifen
2. Aktive Erkennungsstrategien
3. Aktive automatisierte Eindringling Suchstrategien

5.1 Passives Erkennen kombiniert mit aktivem Eingreifen

Diese Gruppe entspricht den heutigen Intrusion Prevention Systemen. Eine kontrollierte Machtteilung erlaubt darüber hinaus den Überblicksvorteil einer globalen Zentrale zur Intrusion Detection auszunutzen. Bei der SPAM Bekämpfung ermöglicht ein Überblick über eine große Anzahl von Mailkonten, SPAM leicht zu identifizieren. So kann ähnlich dem Dshield Projekt [We04b] in einer verknüpften Zugriffskontrolle ein externer Anbieter die erreichten Zugriffsrechte der Benutzer überwachen und beim Auftreten einer Anomalie (z.B. ein Benutzer hat ungewöhnlich viele Zugangsrechte) einen Alarm oder eine Reaktion auslösen.
Umsetzung durch die ASCap Architektur: Mittels des Mechanismus der externen Sicherheitsrechner von Kategorie γ ist es möglich, einer übergeordneten Datenauswertung die notwendigen Informationen zu liefern.

5.2 Aktive Erkennungsstrategien

Geht man von einer Monokultur im Bereich der Zugriffskontrolle aus, so ist es dem einfachen Angreifer möglich, eine einmal entdeckte Nachlässigkeit beliebig auszunutzen. Wenn man, um die Sicherheit zu erhöhen, als Schutzmassnahme alle bekannten Techniken gleichzeitig anwendet, würde dies normale Arbeitsabläufe unnötig erschweren. Als Lösung bietet sich an, periodisch die verwendeten Techniken zu ändern, um Eindringlingen das Leben zu erschweren. Aktive Erkennungsstrategien erweitern dies, indem sie das Verhalten der einzelnen Anwender überwachen. Für jedes Zugriffskontrollszenario werden die Statistiken getrennt geführt. Es wird angenommen, dass rechtmäßige Benutzer in allen Szenarien vollen Zugang haben und dass die Verhaltensmuster gleich sind. Fällt ein Benutzer durch Inaktivität in einem Szenario auf, kann ein Eindringling gefunden worden sein. Die aktiven Erkennungsstrategien rufen nun die Zugriffstechnikwechsel gezielt hervor, um erkannte Muster zu bestätigen und Falschalarme zu vermeiden.
Umsetzung mittels der ASCap Architektur: Verschiedene Zugriffspolitiken mit Regelbausteinen wie verschiedene Sicherheitsabfragen (Passwort, Geburtsdatum etc.) oder der Zuhilfenahme von externen Hilfsmitteln (z.B. SMS die beantwortet werden muss) sind von der internen Firma in Abstimmung mit dem Dienstleister vordefiniert. Die externe Firma kann mittels einer dynamisch änderbaren Regel (Kategorie δ) die verwendete Zugriffspolitik im Einzelfall wählen. Werden Alarmzeichen erkannt, kann die Wahl der Zugriffspolitik so beeinflusst werden, dass Eindringlinge leichter erkannt werden.

5.3 Aktives Suchen

Nach dem Beispiel der manuellen Einzelfallanalyse kann das aktive Suchen aus dem Überprüfen von Annahmen über den Angreifer bestehen. Angreifer verbinden sich oft über Zwischenrechner, die sie beliebig auswählen können. Die Annahme, dass ein normaler Benutzer nur über einen Einzelplatzrechner und begrenzte Einwahlmöglichkeiten verfügt, können zu der Erkennungsstrategie führen, dem verdächtigen Benutzer mitzuteilen, dass die von ihm genutze IP Region zeitweise gesperrt werden muss. Ein hartnäckiger Angreifer wechselt nun leicht den Zwischenrechner und wird so enttarnt.
Eine weitere Eigenschaft eines Spiones kann sein, dass er Daten beliebiger Quelle sammeln will. Arbeitet ein Benutzer außerordentlich lange an einem Projekt und erregt damit aufsehen, so ist es möglich, diesen Bereich zeitweise für ihn zu schließen. Ändert er sein Interessengebiet und arbeitet ebenso intensive an anderen Projekten, so kann ein fleißiger Angreifer gefunden worden sein.
Aktive Strategien zur Angriffserkennung der Gruppe “Aktives Suchen” werden einzelne Eigenschaften des Angreifers annehmen und dann das System zeitweise so verändern, dass diese Eigenschaften deutlicher testbar werden. Damit sind diese Strategien der passiven Datenauswertung deutlich überlegen.
Umsetzung mittels der ASCap Architektur: Nachdem die vermuteten Eigenschaften der Angreifer spezifiziert sind, können Tests entwickelt werden, die diese Eigenschaften überprüfen. Dann ist es möglich an den erforderlichen Stellen in dem System Einstellpunkte

anzubringen, die je einen Test darstellen. In der ZKA werden externe Sicherheitsrechner (Kategorie γ) und dynamisch änderbare Regeln (Kategorie δ) zum Einsatz kommen.

6 Zusammenfassung und Ausblick

In diesem Beitrag wurden aktive Strategien zur Intrusion Detection vorgestellt, die es erlauben, nicht nur nach bekannten Angriffsignaturen im auftretendem Datenverkehr zu suchen, sondern im Falle von allgemeineren Alarmsignalen gezielt Benutzer auf verdächtige Eigenschaften zu prüfen. Diese Tests werden möglich, wenn die IDS enger mit der Zugriffskontrollarchitektur zusammenarbeiten. Hierzu wurden zuerst verschiedene Kategorien der Zugriffskontrollverwaltung vorgestellt. Eine bisher unbekannte Kategorie der kontrollierten Machtteilung wurde aufgezeigt und die ASCap Architektur vorgestellt. Die ASCap Architektur implementiert die kontrollierte Machtteilung und erlaubt so, IDS mit der ZKA zu verknüpfen. Verschiedene Beispiele von Angriffserkennung wurden diskutiert, die mit den verknüpften Systemen angewendet werden können. Es wurde aufgezeigt wie bekannte Techniken der SPAM-Bekämpfung auf die Zugriffskontrolle und Schutzzielverletzungserkennung übertragen werden können.

Das Aufzeigen der Möglichkeit aktiven Strategien zu verwenden wird als Ergebniss unserer Forschung angesehen. Ein passives auf bekannte Angriffssignaturen warten kann durch ein aktives Provozieren von verdächtigem Verhalten erweitert werden. Dabei wechselt der Fokus vom Angriff zum Angreifer. Die Strategie des “Aktives Suchen” erlaubt es einen fleissigen Arbeiter von einem Industriespion zu unterscheiden. Die in diesem Beitrag beschriebenen Strategiebeispiele beruhen auf einfache Annahmen und Teil der weiterführenden Forschung wird es sein, die hier vorgestellten information-technische Infrastruktur zu nutzen um weitere Eigenschaften und Strategien zu finden. Hier sehen wir Chancen auch interdizplinär beispielweise von verhaltenspsychologische Ergebnisse wie [Tu99](Kapitel 25) zu profitieren.

Literatur

[Ab03] Abendroth, J.: Applying π-calculus to practice: An example of a unified security mechanism. Technical Report RS-03-39. Basic Research In Computer Science, University of Aarhus. 2003. 35 pp.

[AJ03] Abendroth, J. und Jensen, C. D.: A unified security mechanism for networked applications. In: *Proceedings of 18th Symposium on Applied Computing (SAC2003)*. S. 351–357. ACM. March 2003.

[AOTG99] Asaka, M., Okazawa, S., Taguchi, A., und Goto, S.: A method of tracing intruders by use of mobile agents. In: *INET’99*. 1999.

[De87] Denning, D. E.: An intrusion-detection model. *IEEE Trans. Softw. Eng.* 13(2):222–232. 1987.

[ER88] E, S. M. S. E. H. und R., W.: Expert system in intrusion detection: A case study. In: *Proceedings of the 11th National Computer Security Conference*. S. 85–91. October 1988.

[FK92] Ferraiolo und Kuhn: Role based access control. In: *Proceedings of 15th National Computer Security Conference*. S. 554–563. October 1992.

[HR02] Huth, M. R. A. und Ryan, M. D.: *Logic in Computer Science*. Cambridge University Press. The Edinburgh Building, Cambridge, DB2 2RU, UK. 2002.

[JS03] Jung, A. H. M. und Schaefer, G.: Combining multiple intrusion detection and response technologies in an active networking based architecture. In: *DFN-Arbeitstagung ueber Kommunikationsnetze, Duesseldorf, Germany*. June 2003.

[Kü95] Kühnhauser, W. E.: On paradigms for security policies in multipolicy environments. In: *Proceedgins of 11th International Information Security Conference (IFIP/SEC'95), Cape Town, South Africa*. 1995.

[Kü99] Kühnhauser, W. E.: Metapolitiken (german). Technical Report RS-AiS-1999-13,ISBN3-88457-362-4. Informationstechnik GmbH. Sankt Augustin. 1999.

[Le84] Levy, H. M.: *Capability-Based Computer Systems*. Digital Press. Bedford, Massachusetts. 1984.

[LJ88] Lunt, T. F. und Jagannathan, R.: A prototype real-time intrusion-detection expert system. In: *IEEE Symposium on Security and Privacy*. S. 59–66. October 1988.

[QL96] Qian, T. und Liao, W.: Active capability: An application specific security and protection model. Technical report. University of Illinois at Urbana-Champaign. 1996.

[Sh86] Shapiro, M.: Structure and encapsulation in distributed systems: The proxy principle. In: *Proceedings of the 6th International Conference on Distributed Computer Systems*. S. 198–204. Cambridge, Massachusetts, U.S.A. 1986.

[Tu99] Turvey, B.: *Criminal Profiling: An Introduction To Behavioral Evidence Analysis*. Academic Press. 24-28 Oval Road, London NW1 7DX,UK. 1999.

[Va02] Various. Open source pki book, http://opensourcepkibook.sourceforge.net. 1.12.2002.

[We04a] Website. http://www.cert.org. 31.1.04.

[We04b] Website. http://www.dshield.org. 31.1.04.

[We04c] Website. http://www.securityfocus.com/archive/1. 31.1.04.

[We04d] Website. http://www.snort.org. 31.1.04.

A Honeynet within the German Research Network – Experiences and Results

Helmut Reiser
Munich Network Management Team
Ludwig Maximilian University Munich
helmut.reiser@ifi.lmu.de

Gereon Volker
Munich Network Management Team
Technical University Munich
gereon.volker@mytum.de

Abstract: A honeynet is a special prepared network which is not used in normal business. It is a kind of playground to watch and learn the tactics of crackers. The only purpose of a honeynet is to be probed, attacked or compromised. During the operation other systems may not be harmed by an attack originated within the honeynet. In this paper the design, realization and operation of a honeynet built within the German Research Network (DFN) will be described. Concepts for continuously monitoring and securing the honeynet are introduced. A selection of the results of the operation phase will be presented as well.

1 Introduction

Due to increasing number and severity of attacks it is important for security administrators to know about tactics, motives and preferred targets of their adversaries. In this paper the design, operation and analysis of a honeynet built within the German Research Network will be described. A honeynet is a screened and controlled network of honeypots. A **honeypot** is a system whose single purpose is to be probed, attacked, or compromised, by attackers. Learning the tactics, tools, and motives of attackers is one reason to build honeypots. The other reason is to slow down an attack by engaging the attacker with a system whose only purpose is to be attacked. The honeypot should be sufficiently interesting for the attacker, to extend his efforts and spend a lot of time at this system. A **honeynet** is not a single system but a network. Within this honeynet different types of honeypots can be placed. The whole honeynet with all its systems is not used productively in regular business. Therefore, all traffic within this network must be originated by an attacker. The honeynet is located therefore behind a filter (the honeywall) where all inbound and outbound data is captured. This data is then analyzed to learn about the techniques of attackers [Ho01].

The honeynet described here has been set up at the Leibniz–Supercomputing Center within the German Research Network. The German Research Network is the Internet backbone for all universities and research institutes in Germany. In German it is called ”Deutsches Forschungsnetz (DFN)”. DFN is the national research network with upstream connections to the European research network Géant, to various commercial providers and the global

Internet. DFN connects all research facilities and provides them Internet access and additional services (e.g. mail, WWW, Video-Conferencing, etc.). DFN is one of the largest Internet providers in Germany. In 2003 data transmitted per month, exceeded the petabyte barrier. The Leibniz–Supercomputing Center (”Leibniz Rechenzentrum”, LRZ) in Munich operates the Scientific Network in Munich (MWN) and research facilities in the south part of Bavaria and is directly connected to one of the core routers of the DFN. The honeynet which will be described in this paper has been built at the LRZ. Its operation time lasted from July 15th until September 12th 2003.

Section 2 describes the requirements, the design, the architecture and the operation of the honeynet. Alarming mechanisms as well as analysis tools are described. The results of the deployment are pointed out in section 3. The paper is concluded by a “lessons learned” section and a view to further work.

2 Design, Operation and Analysis

To deploy a honeynet, three basic tasks must be carried out: data capture, data control and data analysis. Data capture deals with the recording of all traffic which arrives or leaves the honeynet. This must be done on several layers and on all systems within the honeynet. For this reason a few tools are installed on the honeywall and the honeypots which are described in the following.

Honeynets must be configured in a way that other systems cannot be harmed or attacked if an attacker breaks into the honeynet. This is the aim of data control. Data analysis means efficient analysis of the collected data. Therefore, tools are required which are able to help the administrator to analyze fast growing log files (up an even more than 100 MB a day) and to extract relevant information out of data noise.

In the following, we describe hardware and software architecture design to meet these requirements.

2.1 Honeynet Software– and Hardware–Architecture

Based on the ideas of Lance Spitzner, one of the founders of the honeynet project [Honb], a second-generation (Gen II) honeynet (see figure 1) has been selected for this work [Sp03]. A Gen I honeynet is placed behind a simple firewall with a capturing component. In a Gen II honeynet the “copy of the production net” is placed behind a honeynet sensor which is also called honeywall. The only way to reach a honeypot is throughout the honeywall.

A honeynet is a smaller copy of the network and system infrastructure within a certain organization. Therefore it should contain systems similar to those regularly used within the organization. In our case Linux and Windows honeypots have been placed in the honeynet.

The honeywall is a more complex gateway, it is a bridging firewall with an intrusion detec-

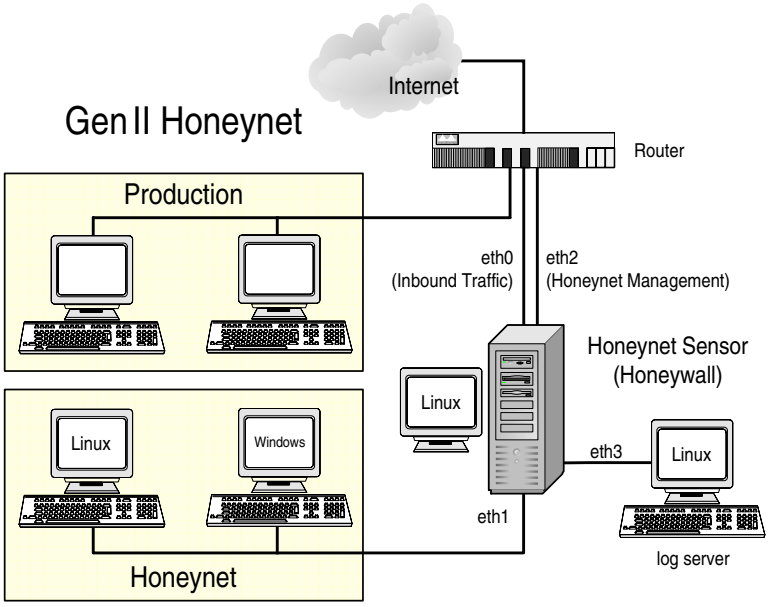


Figure 1: Gen II honeynet with additional log server

tion system. It has to realize two properties: The attacker should not be able to detect the existence of the honeywall and for the operator of the honeynet it is an efficient analyzing, filtering and controlling tool.

From the attackers point of view the honeywall acts like a bridge which means that the honeywall is nearly “invisible” to the attacker. There is no decrement of the time-to-live (TTL) field in the IP–Header, there is no packet routing (bridges operate on layer two of the ISO/OSI model) and there are no MAC addresses to identify. Bridges usually use the spanning tree protocol to detect loops. This protocol is deactivated at the honeywall during operation. The only way to recognize the bridge respectively another system between the attacker and a honeypot is to capture the network traffic on one of the honeypots. If the attacker exceeds the outgoing connection limit or starts new attacks which get dropped by the controlling component of the honeywall he might wonder why his attacks do not reach the destination system. However in this case the attacker has already gained access to a honeypot and left behind lots of information about his attack. Even if he realizes that he has been bluffed, his traces can be used to reconstruct the way the attacker compromised the honeypot.

From the honeynet operators point of view the honeywall works like an extended firewall. All of the data passing the honeywall can be captured, and filtered. As the firewall component does not analyze any content of packets an Intrusion Detection System (IDS) and a controlling component is installed. The intrusion detection system is realized with snort [snoa]. Snort controls the traffic from the Internet to the honeypots. Known attacks (es-

pecially attacks on web servers) can be easily detected: snort compares each packet with its internal database of signatures. A signature describes a kind of a patten of a known attack. If a packet matches one of these signatures an alert is generated or an action can be triggered. To simplify the analysis self defined signatures can be added. This feature has been used during the analysis phase to separate data noise from interesting traffic and new attacks. Data noise in this case is traffic which is already known and analyzed, e.g. probes which can be seen regularly, known mass attack–attempts from certain sources or senseless attacks like testing Windows IIS exploits on a Linux honeypot.

Snort_inline [snob] is a modified version of snort. It is intended as a protection for foreign hosts. It must be prevented that a honeypot is used as a platform for further attacks on other systems in the Internet. Therefore the firewall forwards outgoing packets from a honeypot to snort_inline (via the queue target of iptables [ipta]). The signatures are similiar to snort and all recognized attacks from one of the honeypots are dropped by snort_inline. Even if a worm (e.g. like Blaster or MS Slammer) or another kind of "automatic" attack reaches one of the honeypots this will not harm other systems. One deficiency of snort is that this kind of signature based IDS is unable to detect new attacks which are not known and therefore not represented in the signature database. In consequence a concept is necessary for protecting the "rest of the world" from unknown attacks.

For that purpose a controlling component has been installed to restrict the number of outgoing connections. The firewall which counts all outgoing connections could be used as such a controlling component on the honeywall. In our case each honeypot is allowed to have 15 outgoing connections per day (in each case for TCP, UDP and ICMP). This number is quite restrictive and the asymmetry between inbound and outbound traffic bandwidth might give an attacker a hint that he is within an honeynet. We are aware of that problem, however our most important requirement is to protect foreign systems from damage caused by our honeynet.

Besides this protection and control system a data capturing concept has been implemented. Data capture is done on all systems and on several layers:

1. All network traffic (inbound and outbound) will be dumped on the honeywall. To log binary dumps of all packets tcpdump has been used.

2. Firewall logfiles are more general than network dumps. Several tools exist to analyze these files. Analyzing the tcpdump log files is very time-consuming so it's much easier to get an overview of the events by firewall logfiles and select the most interesting binary dumps with this information.

3. A mechanism to dump attacker's keystrokes on each honeypot is required. Only with shell inputs the attacker's approach could be analyzed. Members of the Honeynet Project developed some tools to dump the keystrokes: for Windows platforms exists a tool called ComLog [com] on Unix systems Sebek [seb] is used. This tool is a modified rootkit based on the Adore rootkit [ado]. It forwards keystrokes and even SSH connections to a specified host. In this case these messages are forwarded to the honeywall. The messages can't be recognized by the attacker because Sebek puts the data directly to the network device driver and not via the socket interface

and the TCP/IP stack of the kernel [Mc03]. The attacker does not see any suspicious network connections. Even if he puts the network interface in promiscuous mode he is not able to see related outgoing packets.

4. On each honeypot the local logfile entries are forwarded to a central log server. This prevents from modifications of local logs concealing the attacker if a honeypot has been taken over. To ensure the forwarding to the log server and to camouflage forwarding a second syslog daemon (with a hidden configuration file) is installed and configured on the Linux honeypot. The Windows eventlog is forwarded with Eventlog to syslog [eve] to this log server.

Especially for forwarding of local logs a central log server is needed. The first idea was to place it together with the honeypots within the honeynet, but in this configuration the log server becomes another honeypot and might also be a potential aim for an attack which is not intended. For this reason the honeywall is equipped with a fourth network interface card where the log server is connected to.

The resulting architecture consists of a honeywall two honeypots (operating systems Windows 2000 and SuSE Linux 8.0) which are connected to the honeywall (cf. figure 1). Two interfaces on the honeywall are used for inbound and outbound honeynet traffic (eth0 and eth1), one is a dedicated management interface to administrate the architecture remotely (eth2) and the fourth is needed for the log server (eth3). The IP address of the management interface is placed in a different and additionally protected subnet which can not be reached directly from the honeynet. The log interface is protected by the firewall.

The Windows honeypot is installed without any Service Packs or patches for IIS or Windows on the Linux honeypot a default installation with X environment (also without any patches) was chosen.

The honeypots must be configured in a way that attackers should not notice the existence of the honeynet and the honeypots must be as interesting that they will be selected as a target for an attack. For this reason several services are set up and “attractive” names for the systems must be found to suggest productive systems. One honeypot is named internal.lrz-muenchen.de to indicate a host where confidential information is stored. The other one got the name tivoli.lrz-muenchen.de to pretend a running network management host.

The following services are configured:

- Web servers: On both honeypots are web servers installed, an Internet Information Server (IIS) on the Windows honeypot, which is included with Microsoft Windows 2000 Professional. There are no web pages configured to pretend the installation was done by a careless administrator who forgot this installed service. An Apache web server with PHP is configured on the Linux honeypot. To generate content every day two statistics of the daily traffic amount are generated by the accounting server of the LRZ and copied to the Linux honeypot. A Perl script randomizes the IP-addresses and stores the timestamp of the generation in a MySQL database to implement a history function.

Name	Function	CPU/RAM	Operating System
gway.lrz-muenchen.de	Honeynet sensor	PIII 500 MHz 128 MB RAM	SuSE Linux 8.3
tivoli.lrz-muenchen.de	Honeypot I	Pentium 200MHz 256 MB RAM	Microsoft Windows 2000 (without any Service Pack)
internal.lrz-muenchen.de	Honeypot II	Pentium 200MHz 64 MB RAM	SuSE Linux 8.0 (default installation)
logging.lrz-muenchen.de	Log server	Pentium 200MHz 128 MB RAM	SuSE Linux 8.3

Table 1: Systems building the honeynet

- FTP servers: Both honeypots have a FTP server installed. Download-able data (a Knoppix distribution) is available on the Windows honeypot.

- SSH server: A SSH server is installed per default on the Linux honeypot.

- Database server: The MySQL database which is used for the generation of web pages is also directly accessible from the Internet.

In addition, users with weak passwords have been created on both honeypots to provoke password guessing and account cracking. Table 1 summarizes the installed systems within the honeynet.

2.2 Alarming

Operation of a honeynet is a time consuming and critical task. During operation it is necessary to keep the administrator informed about the ongoing incidents in the honeynet. Therefore different notification mechanisms were set up on the honeypots and on the gateway. Notifications via email was one way to inform the administrator.

In the honeynet a program called swatch [swa] watches the firewall logfile. If a new entry was added to the logfile swatch sends out an email to predefined addresses. To restrict the traffic and prevent a mailbox overflow limits were set: Maximum ten emails were sent within an hour. Additionally all entries within one hour were counted. If the value of this counter exceeds more than 25 an additional email informed the administrator. Besides notifications via email Short Message Service (SMS) was used as a further alarming mechanism which allows the administrator greater mobility. As SMS is pretty expensive the notification via SMS was highly restricted. A maximum of two SMS per hour and only if outgoing connections were registered were sent.

2.3 Analysis Tools

During and after operation the collected data must be analyzed. There are three main tasks: Logfile Analysis, binary packet analyzing and investigation of the source of the attack and the tools used. Some tools are suitable to do an offline analysis; especially web based tools are better for online analysis. The following tools were used:

- Logfile Analysis:
  The honeynet administrator had to investigate snort logs, firewall logs and honeypot logs. To have efficient searching and querying possibilities all logfile entries were additionally stored in a database (e.g., MySQL). To analyze snort logs ACID [aci] was used, which is a PHP based web frontend for snort. ACID allows, for example, to generate charts, summarize alerts, select time frames or to take a look at the content of selected packets. All logs during the whole operation time of the honeynet have to be accessible, therefore ACID has to cope with huge amounts of data (cf. figure 2). The performance of the hosting system is the determining factor for ACID response times.

  The firewall was implemented using Linux iptables. To evaluate the logfiles, iptables log, a PHP/MySQL based web frontend was used [iptb]. This tool stores every entry of the iptables logfile in a database. With the database approach entries are easier to read, easier to group and easier to analyze than the logfile itself.

  Snort_inline [snob] logs all outgoing attacks and drops them. These logs are important if a successful attack hit one of the honeypots because it's possible to gain information about further propagation of the attack. Also ACID could be used here.

  This kind of analysis is very suitable for online analysis. If the administrator gets informed about an incident he can easily take a look at ACID or iptables log via a web browser.

- Binary packet analysis:
  All inbound and outbound traffic was dumped via tcpdump [tcp]. Packetyzer [pac] and Ethereal [eth] are efficient and handy tools to cope with tcpdump logfiles and support decoding of several protocols. Packetyzer (for Windows platforms only) is easier to handle and allows searching for patterns within captured packets. Ethereal is recommended for UNIX platforms. Normally binary packet analysis is done offline.

- Investigation of the source of the attack:
  One of the interesting questions during an analysis is the source (subnetwork or domain) of an attack. An IP address doesn't contain any information about the geographical position of the system the attacker uses. If nslookup returns a hostname of the IP address a rough guess of the location is possible. With traceroute transit systems between the honeynet and the attacker's system might be determined. Unfortunately traceroute doesn't determine the country where the system is located. Visualroute [vis] shows results of traceroute on a world map using known locations

of a lot of transit systems. This approach is far from being perfect, however in quite a lot of cases it reveals helpful and useful information.

By these tools it is only possible to determine the name and the rough location of the attacker's system. P0f [p0f] allows to identify the operating system of the attacker's host by using a technique called passive fingerprinting. Unlike active fingerprinting where packets are sent to the foreign host, passive fingerprinting uses the passing network traffic and analyzes the TCP options [Do].

3 Results

The honeynet was in operation between July 15th and September 12th 2003. At no time existence of the new subnet was propagated actively and no connections were made from the honeynet to other sites, e.g. there was no mail, news or www traffic. The honeynet was brought online at 8:55 am (GMT+1) on July 15th and two minutes later the first successful attack — a CodeRed2 on Microsoft IIS — hit the Windows honeypot. The honeynet was frequented quite often enabling us to collect a high amount of data (cf., figure 2).

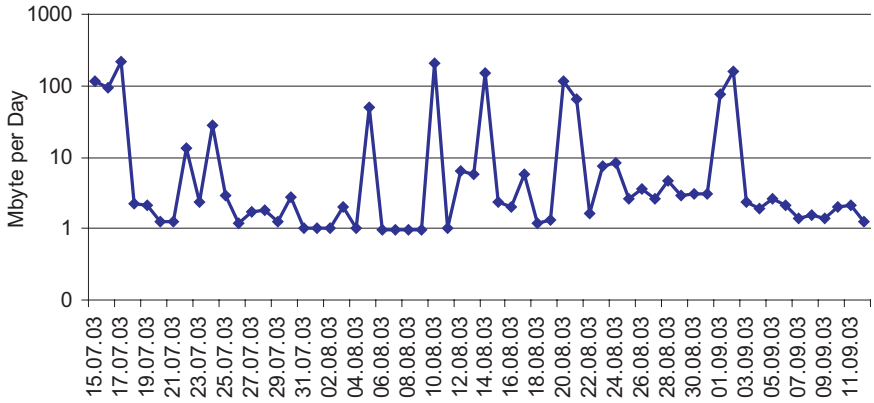


Figure 2: Honeynet traffic

Most attacks targeted the Windows systems. Regarding the services being attacked (cf., figure 3) one can observe that more than half of all attacks (57 % in sum) tried to exploit one of the plenty Windows NetBIOS vulnerabilities. The second biggest group, 36 % of the traffic, targeted Web servers. Incidents presented in the following can be classified in four main categories: Attacks against web servers, worm attacks, spoofing attacks and noise traffic.

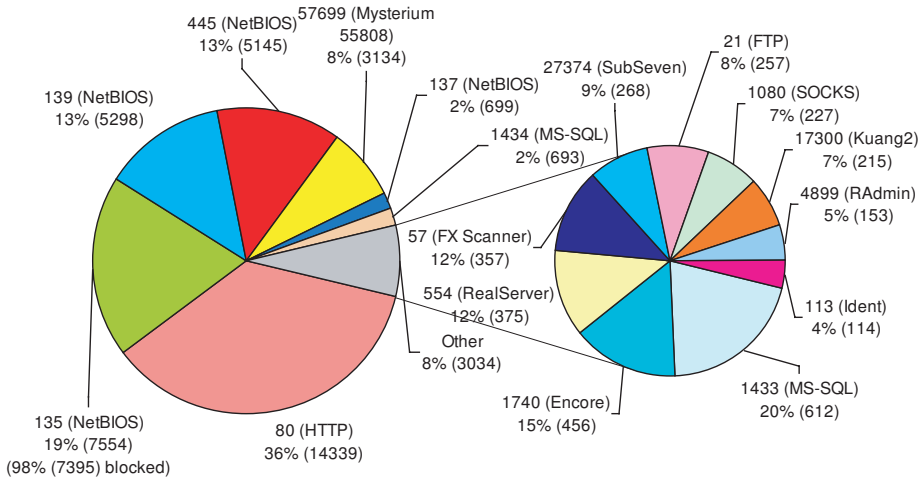


Figure 3: Attacked ports (services) and frequency

3.1 General Observations

At the start–up of the honeynet (July 15th), additional port filters had already been enabled on the DFN-Gateway. This filters block destination ports especially for avoiding well-known and easy exploitable bugs in different Windows NetBIOS system (i.e. ports 135, 137, 138, 139, 445 and 593 are blocked; see [lrz]). During the operation phase it was decided to open these ports to observe all attacks. On August 12th these filters were disabled for the honeynet which resulted in a heavy rise of traffic per day (figure 4).

In figure 5 the number of attacks against port 80 and port 139 for both honeypots are shown. The lines are mostly “parallel” for both systems. Even if a honeypot is not vulnerable the attackers tried to probe both systems. This suggest heavy tool usage and broad scans of Script–Kiddies. Often tools have been seen which probe the kind of OS and than stop on systems which are not vulnerable. This behavior explains the heavy differences in the number of attacks regarding both honeypots.

An analysis on source addresses of packets arriving at the honeynet revealed interesting aspects regarding the distribution of source domains (cf., figure 6). The transformation used were reverse DNS lookups directly on the firewall. The vast majority of attacks were carried out from `t-dialin.net`. Addresses of digital subscriber line (DSL) of “Deutsche Telekom” are bound to this domain. For the second largest group (36%) a reverse DNS lookup (promptly triggered by iptables) did not succeed, giving that group the name “unknown”. IP–spoofing or IP–addresses from private networks may be possibly the reason for this fact, although some IP–addresses simply might have no reverse lookup defined by intention.

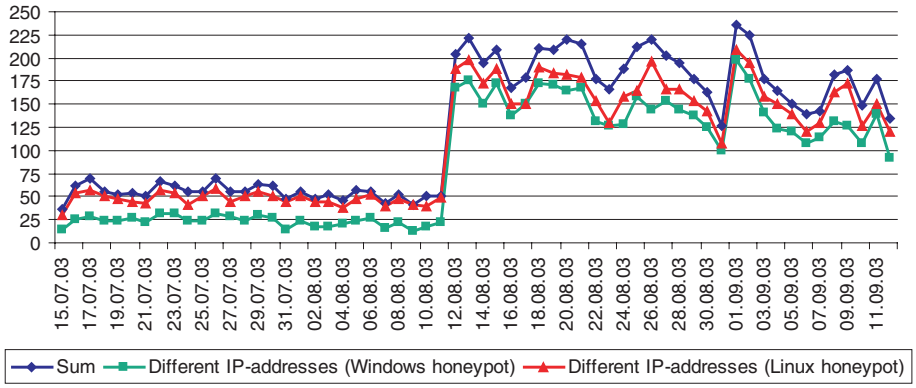


Figure 4: Number of different registered IP–addresses per day

3.2 Web attacks

Web servers (especially Microsoft IIS based ones) frequently become victims of worm code because of plenty and well known (but mostly unfixed) bugs in the software, e.g. vulnerability to buffer overflows.

The IIS webservers in our honeynet was successfully hit by an attack identified as Code Red II [dfn] only two minutes after being online. Even on the Linux honeypot, this attack was recognized eight times during the two month of operation. This fact raises the assumption that script kiddies are randomly trying to attack systems without determining the type of the running web server software in advance.

Some attackers tried to harm the web servers by buffer overflow attacks sending large HTML requests. In most cases the request length was 4096 characters.

3.3 Worm attacks

During the honeynet's operation time the Blaster worm (also known as Lovsan or W32Blaster [cer]) spread out over the globe. It's first appearance at the honeynet was recognized on August 11th at 10:56 pm. A client within the Munich Research Network (MWN) — most likely a notebook — sent requests to port 135 over the whole network, also hitting the Windows honeypot. Further dissemination of the worm was circumvented by snort_inline blocking outgoing TFTP request. Such a connection would have been necessary for the worm to retrieve a file called `msblaster.exe` which was needed to spread further.

The worm's impact on the Windows honeypot was to kill the remote procedure call subsystem (RPC), causing messages in the eventlog (one approximately every 2 minutes).

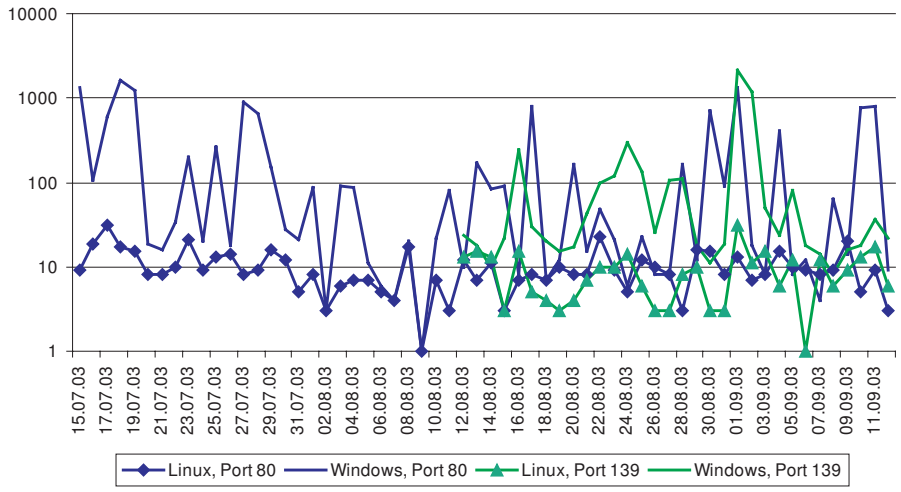


Figure 5: Number of attacks on ports 80 and 139 comparing both honeypots

The system was restored to a clean state by rebooting. Finally, on August 12th at 12:30 am a new rule for dropping requests to port 135/TCP had to be added to the honeywall, as at that time every two minutes the Windows honeypot was hit by a new attack.

On August 20th a variant of the Blaster worm (GaobotAA) [sym] hit the Windows honeypot. This worm also uses the RPC vulnerability but it connects via ports 139 and 445. The execution of the worm code (a file called `winhlpp32.exe` (about 58 KB) was dropped in %WindDir%\system32) inducing a connection to the worm's source system on destination port 22226 or 22227. Via that connection data could be exchanged. However GaobotAA was not able to download code because of the former mentioned new blocking rule for port 135. After September 7th no more attacks of this worm were recognized.

3.4 DoS-Attacks

Between August 26th and September 12th nameservers of different providers in the USA probably became victims of denial of service (DoS) attacks. The queries' source address was set to one of the honeypots' IP–address, the destination port was 53 or 80. The hosts replied to the given IP–address sending response packets with set SYN/ACK–Flag. The honeypots itself replied with a RST-Flag because the source port of the original query was 1740 and this port was closed by default. The honeypots received only the spoofed answers and were not among the intended victims.

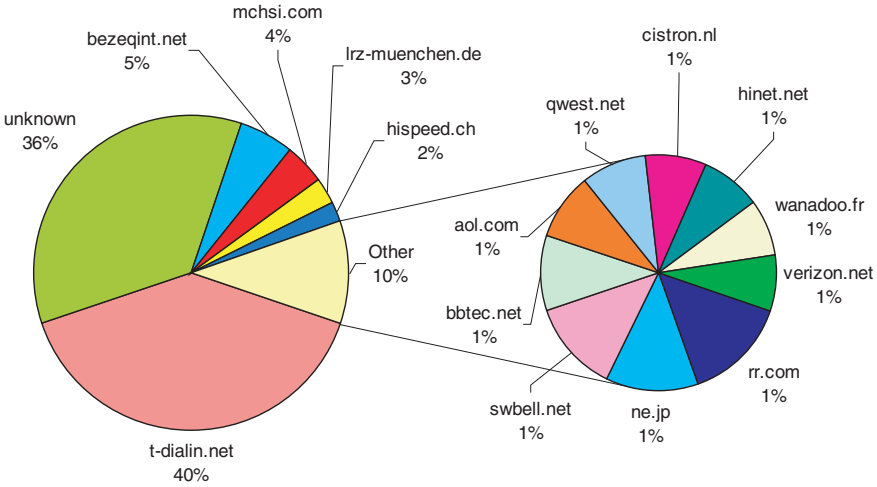


Figure 6: Distribution of DNS domains (more than 250 incoming connections)

3.5 Stumbler/“Mysterium 55808”

Since the beginning of operation, packets with a very high window size were recognized only on the Linux honeypot. The window size was set to 55808 and destination port was always set to 57669. Intrusec [int] and ISS [iss] called the causing trojan “Stumbler” and investigated these packets. The characteristic of Stumbler is its window size and a spoofed source IP–addresses. The trend of this activity is shown in figure 7.

3.6 “Noise”

Lots of connection requests were registered which tried to set up a connection to known trojan ports like Skydance respectively IRC (Port 4000), RAdmin (Port 4899), NetBus (Port 12345), Kuang 2 (Port 17300) and SubSeven (Port 27374). None of these ports were in listen state so all requests have been confirmed with RST. Nearly the same number of requests were noticed on both honeypots. This is also an evidence that script kiddies tried to find vulnerable systems by doing random scans. The same situation showed up with the classic proxy ports (3128, 8080) and the SOCKS port (1080).

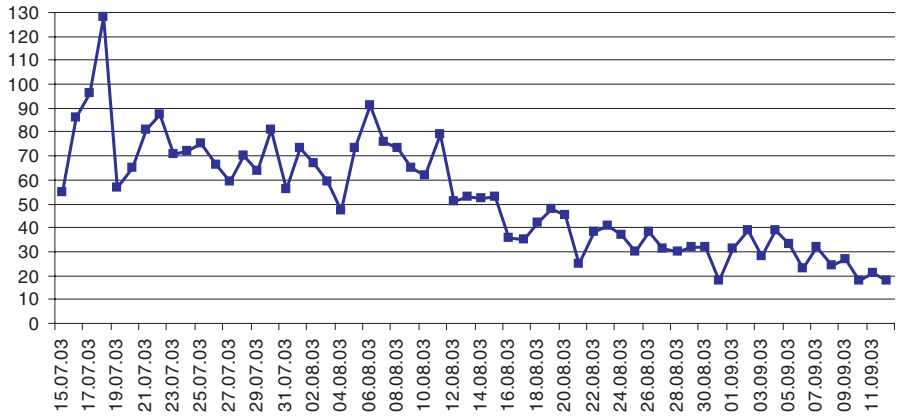


Figure 7: Received Packets per day with window size 55808

4 Lessons learned and future work

This work showed that computer systems which are placed in the Internet without publishing their exitance are under attack extremely fast. The honeynet got online and only two minutes later the first attack took place. “Unfortunately” no “real” or “clever” hostile take-over happened; neither to windows nor to linux. Nevertheless, it was very interesting to watch and analyze the ongoing events in the honeynet, especially the propagation of the Blaster worm.

The honeynet project presents the results and traces of clever attacks. However our experience was, that such attacks are quite seldom. This implies that more than 90 % of the attacks can be defended quite easily by applying a patch for vulnerable software.

The Windows platform was the favorite target of our attackers, 69 % of the attacks aim at this platform. On the Windows honeypot one third of all connections tried to attack the Web server. On the Linux honeypot however we could not detect any serious or successful attack.

A lot of our attackers can be categorized as Script–Kiddies. They are low skilled and mostly use scripts written by more sophisticated crackers. We infer a Script–Kiddie based on the following reasons: A lot of attacks were web attacks using tools which are available in the Internet e.g. FX Scanner [fxs]. These tools are suited for automated attacks over a wide range of IP-addresses and subnets (brute-force scans) without knowing anything about existing systems and their vulnerabilities. Scans have been seen, trying to exploit a certain OS–dependent bug, on both honeypots regardless the used operating system.
Another big group of unintentional “attackers” became infected by a worm or virus, which started to spread out and hit the honeynet casually.

Regarding the sources of the attacks: More than 40% could be assigned to systems which use the Deutsche Telekom as ISP (based on the IP-addresses and domain names). For the attacker such a big provider might be used as a kind of “anonymizer”. Each dial in results in a changed IP address. Without judicial authorization it is nearly impossible to find further informations about the “real” source of the attack.

To sum it up: Most attackers are low skilled, the favorite target of attacks was Windows especially the IIS. In most cases the security level could be improved quite easily. A firewall blocking services which are a cinch to exploit and a patch management which operates continuously, consequently and rapidly are the two most important building blocks for higher security.

Deploying a honeynet the way described here and especially its operation requires a lot of time (supervisioning and analyzing the collected data) and a lot of knowledge about operating systems. However, a honeynet can give hints to ongoing mass attacks and very important information about propagation and effects of such attacks (e.g. worms). A honeynet can slow down an attack. It shows which systems used in the production network are mostly attacked and which services are preferred. This insights can influence the security policy of an organization. Honeynets are very flexible and can be deployed for a multiplicity of scenarios.

During the operation of this work several new ideas were developed to enhance and achieve new insights e.g. how honeynets can help to fight spam. Therefore faked open proxies (e.g. Bubblegum proxypot [pro]) are configured which pretend to anonymize the spammers’ data but this data is captured. A possible way to fight relay spam with a special sendmail configuration is pointed out in [fig].

We are interested to compare a virtual honeynet (Honeyd [hona]) with a dedicated system. In the virtual honeynet the honeypots are not represented as physical hosts but emulated on a single host.

A honeynet might also be used as an Intrusion Response System (IRS) or even as an Intrusion Prevention System (IPS). Detection of an attack can be coupled with active defense operations (e.g., closing ports or filtering certain sources). We are aware that this can be double-edged sword. However defending against extremely fast automatic attacks only an automatic defense system may help.

Acknowledgment

The authors wish to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments on previous versions of the paper. The MNM Team directed by Prof. Dr. Heinz-Gerd Hegering is a group of researchers of the University of Munich, the Munich University of Technology, and the Leibniz Supercomputing Center of the Bavarian Academy of Sciences. The web server of the MNM Team is located at `http://wwwmnmteam.informatik.uni-muenchen.de`.

References

[aci] ACID. http://acidlab.sourceforge.net/.

[ado] Adore rootkit. https://www.team-teso.net/releases/adore-0.39b4.tgz.

[cer] CERT Advisory CA-2003-20 W32/Blaster worm. http://www.cert.org/advisories/CA-2003-20.html.

[com] ComLog. http://www.geocities.com/floydian_99/comlog.html.

[dfn] Code Red2 Analyse und Gegenmassnahmen. http://www.dfn-cert.de/dfn/bt/2001/bt-2001-codered.pdf.

[Do] Doyle, B. Passive Fingerprinting Utilizing the Telnet Protocol Negotiation data. http://www.sans.org/resources/idfaq/fingerp_telnet.php.

[eth] Ethereal. http://www.ethereal.com/.

[eve] Eventlog to Syslog Utility. https://engineering.purdue.edu/ECN/Resources/Documents/UNIX/evtsys.

[fig] Fighting Relay Spam the Honeypot Way. http://www.tracking-hackers.com/solutions/sendmail.html.

[fxs] FX Scanner. http://www.der-klan.de/tools/scanner.html.

[HAN99] Hegering, H.-G., Abeck, S., und Neumair, B.: *Integrated Management of Networked Systems — Concepts, Architectures and their Operational Application*. Morgan Kaufmann Publishers, ISBN 1–55860–571–1. January 1999. 651 p.

[Ho01] Honeynet Project (Hrsg.): *Know Your Enemy: Revealing the Security Tools, Tactics, and Motives of the Blackhat Community*. Addison-Wesley. 2001.

[hona] Honeyd. http://www.honeyd.org.

[Honb] The Honeynet Project. www.honeynet.org.

[Honc] The Honeynet Project: Frequently Asked Questions. http://www.honeynet.org/misc/faq.html.

[int] Intrusec Alert: 55808 Trojan Analysis. http://www.intrusec.com/55808.html.

[ipta] iptables. http://www.netfilter.org/.

[iptb] iptables log. http://www.gege.org/iptables/.

[iss] “Stumbler” Distributed Stealth Scanning Network. http://www.iss.net/issEn/delivery/xforce/alertdetail.jsp?oid=22441.

[lrz] Einschränkungen und Regeln im Netzbetrieb. http://www.lrz-muenchen.de/services/netz/einschraenkung/.

[Mc03] McCarty, B.: The Honeynet Arms Race. *IEEE Security and Privacy*. 1(6). December 2003.

[p0f] P0f. http://lcamtuf.coredump.cx/p0f.shtml.

[pac] Packetyzer. http://www.networkchemistry.com/products/packetyzer/.

[pro] Bubblegum proxypot. http://world.std.com/~pacman/proxypot.html.

[seb] Know Your Enemy: Sebek2. http://www.honeynet.org/papers/sebek.pdf.

[snoa] Snort - The Open Source Network Intrusion Detection System. http://www.snort.org.

[snob] Snort_inline. http://sourceforge.net/projects/snort-inline/.

[Sp03] Spitzner, L.: The Honeynet Project: Trapping the Hackers. *IEEE Security and Privacy*. 1(2). March 2003.

[swa] Swatch. http://swatch.sourceforge.net/.

[sym] W32.HLLW.Gaobot.AA. http://www.symantec.com/avcenter/venc/data/w32.hllw.gaobot.aa.html.

[tcp] tcpdump. http://www.tcpdump.org/.

[vis] Visualroute. http://www.visualroute.com/.

Ermittlung von Verwundbarkeiten mit elektronischen Ködern

Maximillian Dornseif
Institut für Strafrecht, Universität Bonn

Felix C. Gärtner Thorsten Holz
Lehr- und Forschungsgebiet Informatik 4, RWTH Aachen

Abstract: Als elektronische Köder (*honeypots*) bezeichnet man Netzwerkressourcen, deren Wert darin besteht, angegriffen und kompromittiert zu werden. Oft sind dies Computer, die keine spezielle Aufgabe im Netzwerk haben, aber ansonsten nicht von regulären Rechnern zu unterscheiden sind. Köder können zu Köder-Netzwerken (*honeynets*) zusammengeschlossen werden. Sie sind mit spezieller Software ausgestattet, die die Forensik einer eingetretenen Schutzzielverletzung erleichtert. Durch die Vielfalt an mitgeschnittenen Daten kann man deutlich mehr über das Verhalten von Angreifern in Netzwerken lernen als mit herkömmlichen forensischen Methoden. Dieser Beitrag stellt die Philosophie der Köder-Netzwerke vor und beschreibt die ersten Erfahrungen, die mit einem solchen Netzwerk an der RWTH Aachen gemacht wurden.

1 Einleitung

> „Angenommen", sagte er [Winnie Pu] zu Ferkel, „*du* willst *mich* fangen, wie würdest Du das machen?"
>
> „Tja", sagte Ferkel, „ich würde es so machen: Ich würde eine Falle bauen, und ich würde einen Topf Honig in die Falle stellen, und Du würdest den Honig riechen, und Du würdest in die Falle gehen, und …"
>
> [Mi87, S. 64]

Der englische Begriff *honeypot* bezeichnet für gewöhnlich einen Gegenstand, von dem eine gewisse Attraktivität ausgeht, die bestimmte, nicht nur tierische, Interessenten anzulocken vermag [Lo01]. Sie eignen sich demnach als Köder, um Aufmerksamkeit auf einen Gegenstand zu lenken. Wissenschaftler haben neuerdings begonnen, das Prinzip der Köder auch im Bereich der IT-Sicherheit anzuwenden. Hier werden *elektronische Köder* ausgelegt, um das Verhalten von Angreifern leichter zu studieren. Elektronische Köder sind Netzwerkressourcen (Computer, Router, Switches), deren Wert darin besteht, angegriffen und kompromittiert zu werden [Sp02]. Die Köder haben keine spezielle Aufgabe im Netzwerk, sind aber ansonsten nicht von regulären Komponenten zu unterscheiden. Sie sind mit spezieller Software ausgestattet, die die anschließende Forensik eines Angriffs

deutlich erleichtert. Im Gegensatz zu einer herkömmlichen forensischen Untersuchung erlauben beispielsweise gezielte Veränderungen im Betriebssystem das direkte Mitschneiden aller Aktivitäten eines Angreifers. Durch die Vielfalt der so gewonnenen Daten kann man schneller und genauer dessen Angriffswege, Motive und Methoden erforschen. Auch kann man die benutzten Angriffswerkzeuge, die normalerweise nach erfolgter Kompromittierung gelöscht werden, sofort sicher stellen. Schon relativ früh hatte es Versuche gegeben, mit entsprechenden Modifikationen die Aktivitäten von Angreifern aufzuzeichnen [St88, Ch90]. Diese Versuche erfolgten jedoch nicht systematisch, sondern aus der aktuellen Situation heraus, in der ein Einbruch beobachtet wurde.

Der *honeypot*-Ansatz erscheint sowohl aus praktischer als auch aus theoretischer Sicht interessant für den Bereich der IT-Sicherheit. Aus praktischer Sicht erlauben detaillierte Informationen über das Verhalten von böswilligen „Hackern“ (den sogenannten *blackhats*), die Abwehrmaßnahmen in unterschiedlichen Umgebungen effizienter und effektiver zu gestalten. Effizienter werden Abwehrmaßnahmen dadurch, dass man sich auf *relevante* Angriffe konzentrieren kann (Netzwerke von *honeypots*, so genannte *honeynets*, können hierzu empirische Beiträge liefern). Effektiver werden Abwehrmaßnahmen, indem mit Hilfe der *honeypots neue* Schwachstellen und Verwundbarkeiten entdeckt und schnell analysiert werden können.

Aus theoretischer Sicht bilden *honeypots* eine interessante Instanz des *dual use*-Prinzips der IT-Sicherheit: Man bekämpft Angreifer mit ihren eigenen Methoden. Die Fähigkeiten und die Werkzeuge, die ein Verteidiger anwendet, unterscheiden sich kaum von denen der Angreifer. So entstand beispielsweise die *honeynet*-Software Sebek [Th03b] aus einem verbreiteten Hackerwerkzeug (einem *rootkit*). Außerdem versucht der Betreiber eines *honeypot* genau das, was ein erfahrener Angreifer ebenfalls tun würde, nämlich: seine Existenz im System mit allen Mitteln zu verbergen.

Am Lehr- und Forschungsgebiet Informatik 4 (Verlässliche Verteilte Systeme) der RWTH Aachen wird im Rahmen eines durch die DFG geförderten Projektes der Einsatz von *honeypots* zur Gefahrenabwehr in der IT-Sicherheit untersucht. Dieser Beitrag berichtet von den ersten Erfahrungen, die beim Aufsetzen eines elektronischen Köder-Netzwerks gesammelt werden konnten. Die hierbei gewonnenen Einsichten deuten darauf hin, dass *honeynets* eine wertvolle Unterstützung für die IT-Sicherheitsforschung sein können.

Abschnitt 2 erläutert Ursprünge, Ziele und Erfolge bisheriger *honeynet*-Forschungen. In Abschnitt 3 wird der Aufbau des *honeynet* an der RWTH Aachen näher beschrieben. Daran schließt sich in Abschnitt 4 ein erster Bericht über die bisher gemachten Erfahrungen an. Die ethischen und rechtlichen Aspekte beim Betrieb eines *honeynet*, die in Deutschland beachtet werden müssen, werden in Abschnitt 5 näher beschrieben. Abschnitt 6 gibt einen Überblick über die geplanten Arbeiten in diesem Themengebiet.

2 Das globale Honeynet-Projekt

2.1 Motivation

Unter dem Namen *The Honeynet Project* [hon04a] haben sich Forscher aus der ganzen Welt zusammengeschlossen, die sich mit dem Thema IT-Sicherheit beschäftigen. Ziel des Projekts ist, die Programme, Vorgehensweisen und Motive von böswilligen Angreifern (*blackhats*) kennenzulernen und das so gewonnene Wissen allen Interessierten zur Verfügung zu stellen. Das Augenmerk liegt vor allem auf folgenden Punkten:

- **Bewusstsein schaffen**

  Es soll bewusst gemacht werden, in welchem Ausmaß Bedrohungen und Sicherheitslücken in Netzwerken bestehen. Dazu werden Computersysteme „echten“ Angriffen aus dem Internet ausgesetzt und Daten über die Geschwindigkeit und den Grad der Schwierigkeit (oder vielmehr der Leichtigkeit) einer Kompromittierung veröffentlicht.

- **Lernen über bekannte Angriffswege**

  Durch die Erfahrungen mit *honeynets* kann man genaue Rückschlüsse ziehen auf die Art und Weise, wie Angreifer vorgehen, sowie auf die Werkzeuge, die sie dabei verwenden. Diese Informationen erlauben einem Systemadministrator, sich effizient gegen Verletzlichkeiten abzusichern. Effizient bedeutet hierbei, dass man den Einsatz von Ressourcen für die Gefahrenabwehr gezielter planen kann.

- **Lernen über unbekannte Angriffswege**

  Honeynets ermöglichen erstmals, Angreifern bei ihrer Arbeit „über die Schulter zu schauen“ und so Bedrohungen und Angriffe in einer neuen Qualität zu erforschen. Sehr leicht läßt sich beispielsweise die zeitliche Reihenfolge rekonstruieren, in der ein Angreifer vorgegangen ist. Dies ist bei herkömmlichen *a posteriori* forensischen Untersuchungen (wenn überhaupt) nur mit großem Aufwand möglich. Außerdem gewähren *honeynets* neue Einblicke in die Motive und Kommunikationswege von Angreifern. In diesem Kontext sollen Methoden und Techniken entwickelt werden, die das Sammeln und das Verbergen von Informationen im Rechnernetz ermöglichen.

- **Aktive Verteidigung**

  Honeynets dienen direkt dem Schutz von Netzwerken. Als *soft target* in einem besonders gefährdeten Netz sollen sie wie die Magnesium-Anode beim kathodischen Korrosionsschutz die Angreifer von den schützenswerten Zielen ablenken. Weiterhin wird versucht, Angriffe zu erkennen und diese, für den Angreifer transparent, vom Produktionssystem auf einen *honeypot* umzulenken.

2.2 Vorgehensweise

Im Rahmen des Honeynet-Projekts werden Werkzeuge für den Einsatz von *honeypots* entwickelt. Hierzu gehört die spezielle Überwachungssoftware Sebek [Th03b], die ihre Existenz auf dem *honeypot* verbirgt. Sebek ist ein Client/Server-System: Auf dem *honeypot* selbst befindet sich der Sebek-Client, der, für einen Angreifer nicht wahrnehmbar, alle Aktivitäten mitschneidet. Diese Informationen werden (ebenfalls unmerklich) über das Netzwerk an den Sebek-Server geschickt, der die Daten für die Auswertung speichert und aufbereitet.

Der *honeypot* selbst ist in der Regel ein Computersystem, das keine konventionelle Aufgabe im Netzwerk hat und mit dem deshalb in der Regel niemand interagiert. Diese Annahme vereinfacht die Entdeckung eines Angriffes deutlich: Findet eine Interaktion mit diesem System statt und werden dort Pakete empfangen, deutet dies auf ein Sammeln von Informationen, einen Angriff oder einen Kompromittierung des Systems hin. Deshalb wird sämtlicher Netzwerkverkehr zwischen dem *honeypot* und dem übrigen Netz für die spätere Analyse gesammelt.

Ein *honeynet* ist ein Netzwerk von mehreren *honeypots* und stellt also die Verallgemeinerung des Prinzips dar. Hierbei werden meist verschiedene Typen von *honeypots* in einem Netz aufgesetzt, wodurch man Informationen über Angriffe auf verschiedene Plattformen erhalten kann.

2.3 Bisherige Erfahrungen

Der Einsatz von *honeynets* hat bereits wertvolle Informationen über die Vorgehensweise und das Verhalten von *blackhats* gebracht. Die verwendeten Programme und Toolkits (insbesondere *rootkits* und verwendete *exploits*) konnten aufgezeichnet und analysiert werden. Beispielsweise wurde im Januar 2002 mittels der aufgezeichneten Daten eines *honeypot* erstmals ein *exploit* gefunden, der eine vorher schon bekannte Schwachstelle in der `dtspcd`-Software (CDE Subprocess Control Service) zum Angriff nutzte [CE02].

Durch das Mitschneiden aller Aktivitäten kann man das typische Vorgehen nach einem erfolgreichen Angriff besser nachvollziehen. Darüber hinaus liefern mitgeschnittene Konversationen im *Internet Relay Chat* (IRC) Informationen über das soziale Verhalten und die Kommunikationswege von Angreifern. Das Azusa Pacific University Honeynet Research Project entdeckte vor kurzem mittels eines *honeynet* ein sogenanntes *botnet*, i.e. ein Netzwerk von kompromittierten Rechnern, auf denen ein IRC-Client installiert ist, über den der jeweilige Computer eingeschränkt ferngesteuert werden kann. Dieses *botnet* hatte eine Größe von mehr als 15 000 Rechnern [Mc03b]. Große *botnets* wie dieses werden häufig eingesetzt, um *Denial of Service*-Angriffe auf IRC-Server oder Webserver durchzuführen und stellen deshalb eine große Bedrohung dar.

Die Untersuchung dieses *botnet* hat zu weiteren interessanten Erkenntnissen über das soziale Verhalten von Angreifern geführt. Die Forscher entdeckten etwa eine Reihe von IRC-Foren (*channels*), in denen organisierter Handel mit Informationen über Kreditkarten

stattfand [Mc03a]. Sie fanden automatisierte Tools, mit deren Hilfe Kreditkartenbetrug vereinfacht wurde und erhielten einen Einblick in die Szene.

Außerdem können aus den aufgezeichneten Informationen von *honeynets* Abschätzungen über die Verwundbarkeit von Computersystemen (*Vulnerability Assessment*) und deren erwartete Zuverlässigkeit gewonnen werden [hon04b].

3 Architektur an der RWTH

Am Lehr- und Forschungsgebiet Informatik 4 (Verlässliche Verteilte Systeme) an der RWTH Aachen wurde im Januar 2003 ein *honeynet* aufgesetzt, dessen grundsätzlicher Aufbau in Abbildung 1 zu sehen ist.

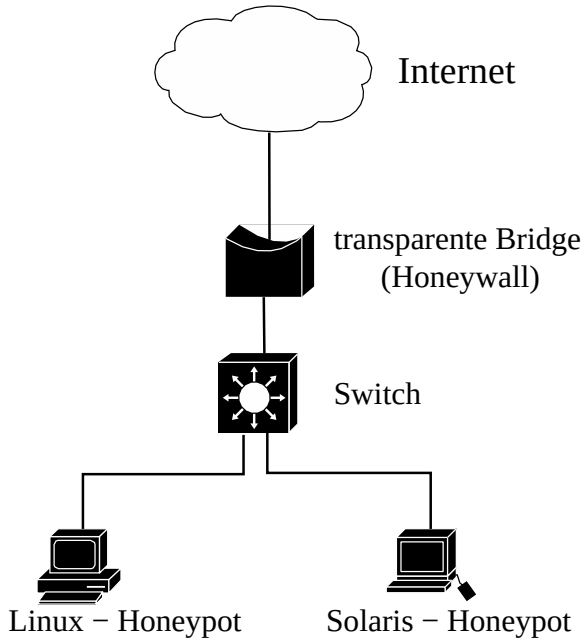


Abbildung 1: Aktueller Aufbau des *honeynet* an der RWTH Aachen.

Auf dem Linux-*honeypot* läuft eine SuSE Linux 8.0 Professional-Installation, die die Dienste HTTP (Apache 1.3.23 inklusive PHP 4.1.0), FTP (vsFTPd 1.0.1) und SSH (OpenSSH 3.0.2p1) anbietet. Außerdem wurde PHP-Nuke [php04] in Version 5.0 sowie MySQL 3.23.53 installiert, um beobachten zu können, ob Angreifer auch spezielle Webapplikationen angreifen. Gegenüber einer Standardinstallation wurden nur wenige Änderungen durchgeführt: Die wichtigste Änderung war die Installation des Sebek-Client. Zudem wurden einige sogenannte *honeytokens* [Th03a] auf dem System hinterlegt. Dies sind verschiedene Arten von Daten (beispielsweise Mails, Tabellenkalkulations- oder verschlüsselte

Dateien), die das Interesse des Angreifers wecken und die den Ablauf der Informationsgewinnung nach einem erfolgreichen Angriff nachzuvollziehbar machen sollen. Desweiteren wurde das System so eingerichtet, dass es einem „normalen“ System mit drei Benutzern gleicht.

Der Solaris-*honeypot* ist eine Sun Ultra 1 mit Solaris 8 als Betriebssystem. Dieses System bietet ebenfalls die Dienste HTTP (Apache 1.3.12), FTP (wuftpd-2.6.1) und SSH (OpenSSH 3.02p1) an. Der Sebek-Client sorgt auch hier dafür, dass sämtliche Aktivitäten eines Angreifers auf diesem System aufgezeichnet werden können. Außer der Deaktivierung vieler Dienste des `inetd` (etwa `telnet`, `echo` oder `talk`) wurden keine anderen Modifikationen durchgeführt.

Die beiden *honeypots* sind an einen Switch angeschlossen und von dort aus über eine transparente Bridge mit dem Internet verbunden. Diese transparente Bridge, genannt *honeywall*, ist eine der wichtigsten Komponenten des *honeynet*. Die *honeywall* basiert auf Debian GNU/Linux 3.0r2 und stellt Möglichkeiten zur Steuerung des Datenflusses und Speicherung der aufgezeichneten Informationen bereit. Mittels der `netfilter` / `iptables`-Funktionalität des Linux-Kernels wird der ausgehende Netzwerkverkehr des *honeynet* limitiert. Beispielsweise dürfen pro Tag nur 15 TCP-Verbindungen je *honeypot* initiiert werden.

Zusätzlich läuft auf der *honeywall* die Snort_inline-Software, ein so genanntes *Intrusion Prevention System*. Dieses sorgt in Verbindung mit `iptables` dafür, dass ein Angreifer nach erfolgter Kompromittierung eines *honeypot* nicht von dort aus noch weitere Systeme angreifen kann. Snort_inline basiert auf Snort, einem quelloffenen *Intrusion Detection System*, und erweitert dessen Funktionalität um Methoden zur Datensteuerung speziell für *honeynets*. Dazu werden die Snort-Regeln so umgeschrieben, dass bekannte Angriffe auf andere Systeme nicht geblockt, sondern modifiziert und somit effektiv verhindert werden. Abbildung 2 zeigt ein Beispiel für eine Snort_inline-Regel, mit der Pakete, die Shellcode für die x86-Architektur enthalten, so verändert werden, dass sie auf einem angegriffenem System keinen Schaden anrichten.

```
alert ip $HONEYNET any -> $EXTERNAL_NET any
(msg:"SHELLCODE x86 stealth NOOP"; rev:6; sid:651;
content:"|EB 02 EB 02 EB 02|";
replace:"|24 00 99 DE 6C 3E|";)
```

Abbildung 2: Snort_inline-Regel zur Modifikation von Paketen mit x86-Shellcode

Die Modifizierung sorgt im Gegensatz zur Blockierung von Paketen dafür, dass der Angreifer versuchen kann, andere Systeme anzugreifen, aber dennoch scheitert. Dies ermöglicht, die Vorgehensweise eines Angreifers so gering wie möglich zu beeinflussen, jedoch ohne dass von ihm weiterer Schaden ausgeht. Mittels Snort_inline kann man das Risiko eines Angriffs vom *honeynet* auf andere Systeme zwar verringern, dennoch besteht aber natürlich weiterhin die Gefahr, dass mittels bisher unbekannter Schwachstellen oder veränderter *exploits* andere Systeme angegriffen werden. Dieses Restrisiko bleibt beim Einsatz eines *honeynet* naturgemäß bestehen.

Außer der Steuerung des Datenflusses hat die *honeywall* die Aufgabe, für die Speicherung aller gesammelter Informationen innerhalb des *honeynet* zu sorgen. Dazu wurde auf diesem System der Sebek-Server installiert, der die von den Sebek-Clients gesendeten Datenpakete sammelt und in eine Datenbank schreibt. Mittels Sebek können sämtliche Daten, die von den *honeypots* verarbeitet werden (insbesondere auch verschlüsselte SSH- oder HTTPS-Verbindungen), aufgezeichnet und später analysiert werden. Zusätzlich sorgen die Speicherung aller von Snort_inline gesammelten Datenpakete in einer Datenbank und die Log-Dateien von `iptables` für eine gewisse Redundanz der gesammelten Informationen.

Darüber hinaus wurden einige Programme wie etwa ACID [aci] oder Swatch [swa04] installiert, um die gesammelten Daten leichter zu verarbeiten und eine Alarmierung bei entdeckter Kompromittierung auszulösen. Außerdem wurde eine Härtung des Systems durchgeführt, um das Risiko eines Angriffs auf die *honeywall* zu minimieren.

4 Statusbericht

Das Aufsetzen des *honeynet* verlief weitgehend reibungslos. Lediglich ein automatisches Laden des Sebek-Client-Moduls beim Systemstart und anschließendes „Verstecken“ dieses Moduls bereitet noch Probleme. Dies soll allerdings durch ein Umschreiben des Sebek-Clients gelöst werden. Insgesamt erforderte die Realisierung der in Abbildung 1 dargestellten Konfiguration weniger als eine Personenwoche Arbeitszeit. Beim Testen der Unsichtbarkeit des Sebek-Client-Moduls wurde desweiteren eine einfache Möglichkeit entdeckt, mit der ein *blackhat* selbst ohne privilegierte Rechte die Existenz von Sebek auf dem *honeypot* entdecken kann. Der Entwickler des Sebek-Clients wurde darüber informiert. Eine systematische Analyse des Fehlers findet im Moment statt.

Eine größere Hürde bildeten verständlicherweise die für Netzwerksicherheit verantwortlichen Mitarbeiter des Rechenzentrums der RWTH Aachen, die den ein- und ausgehenden Netzwerkverkehr zentral kontrollieren. Die dort anfänglich vorherrschende Skepsis ist natürlich verständlich, da ein *honeynet* auch Gefahren für die Netzinfrastruktur und für andere Computer innerhalb des Netzwerkes in sich birgt. Nach einer erfolgten Kompromittierung eines *honeypot* kann ein Angreifer versuchen, von dort aus weitere Systeme im internen Netz der RWTH anzugreifen. Generell gelten Systeme innerhalb des gleichen Subnetzes als einfacher angreifbar, da in der Regel keine Firewall den Netzwerkverkehr filtert. Snort_inline sorgt zwar dafür, dass bereits bekannte Angriffe vereitelt werden, allerdings liefert dies eben keinen garantierten Schutz.

Durch konstruktive Gespräche konnte allerdings eine für beide Seiten annehmbare Lösung gefunden werden. Dem *honeynet* wurde ein Subnetz mit 64 IP-Adressen zugegewiesen. Eine dedizierte Leitung vom Rechenzentrum hin zum *honeynet* ermöglicht den reibungslosen Betrieb und sorgt für eine Absicherung des internen Netzes der RWTH, da keine physische (Kabel-)Verbindung in dieses besteht.

Seit Mitte Februar 2004 ist der Linux-*honeypot* vom Test- in den Produktionsbetrieb übergegangen und sammelt seitdem Daten über den Netzwerkverkehr. Bisher fand noch keine

Kompromittierung des *honeynet* statt, allerdings gewährt eine Auswertung der bis zum 15. April 2004 aufgezeichneten Daten bereits interessante Rückschlüsse auf die Aktivitäten von *blackhats*:

- In den ersten beiden Monaten wurden etwa 61 MB an Netzwerkverkehr mitgeschnitten; dies entspricht etwa 425.000 Datenpaketen. Insgesamt wurden mehr als 9.500 verschiedene IP-Adressen ermittelt, die Datenpakete zu den *honeypots* verschickten; dabei wurden vermutlich auch per *IP spoofing* gefälschte Adressen beobachtet.

  Auch die passive Existenz eines Computersystems im Internet, d.h. eine Anbindung ohne ausgehenden Netzwerkverkehr, lässt es also durchaus zum Angriffsziel werden.

- Der Zugriff über das Netz fand schon etwa zehn Minuten nach dem Anschluss des *honeynet* an das Internet statt. Das System wurde systematisch auf Schwachstellen hin untersucht (*port scan*). Ein ungepatchter Internet Information Server (IIS) von Microsoft wäre nach dieser kurzen Zeitspanne schon kompromittiert gewesen.

- Es wurden vor allem *scans* nach `cmd.exe` durchgeführt. In diesen beiden ersten Monaten fanden knapp 10.000 *scans* dieses Typs statt. Dies deutet auf eine immer noch hohe Aktivität von *script kiddies* und Würmern hin, die nach den Hintertüren von *Code Red* oder Verwundbarkeiten des Internet Information Server suchen. *Code Red* selbst scheint in verschiedenen Versionen noch aktiv zu sein, ca. 1.500 mal wurde ein *scan* nach einer solchen Schwachstelle registriert.

- Die restlichen vom *honeynet* angebotenen Dienste wie FTP, SSH und PHP-Nuke wurden nur selten beachtet. Lediglich ein *scan* nach FTP-Servern, die jederman das Ablegen von Daten erlauben, konnte mehrfach beobachtet werden. Bei den beiden anderen Diensten gab es jeweils nur weniger als ein Dutzend Verbindungsversuche.

Desweiteren konnte gezeigt werden, dass es für einen versierten Angreifer möglich ist, Sebek zu entdecken, zu deaktivieren und zu umgehen [DHK04].

5 Ethische und Rechtliche Aspekte

Dieser Abschnitt bietet einen Überblick über die ethischen Aspekte und die rechtlichen Rahmenbedingungen, die beim Betrieb eines *honeynet* insbesondere in Deutschland beachtet werden müssen. Dies kann natürlich keine umfassende juristische Analyse ersetzen, da diese sehr stark vom konkreten Einzelfall abhängt, reicht aber für eine erste Einschätzung des juristischen Risikos.

Die Verantwortlichkeit des *honeynet*-Betreibers ist in zwei Bereichen besonders diskussionswürdig:

- Wie ist das *honeynet* in Bezug auf das gesamte Internet und wie sind insbesondere Angriffe von *honeypots* aus auf andere Systeme zu beurteilen? Hierbei sind ethische, straf- und zivilrechtliche Aspekte von Interesse.

- Wie ist es zu bewerten, dass die *blackhats* ohne ihr Wissen zum Teil eines Experimentes gemacht werden? Hierbei kommen insbesondere datenschutzrechtliche Aspekte zum Tragen.

5.1 Ethische Aspekte

Durch die Installation eines mit dem Internet verbundenen *honeynet* wurden dem Internet Systeme hinzugefügt, die wider besseren Wissen nicht dem Stand der Sicherheitstechnik entsprechen. Man kann argumentieren, dass dadurch grundsätzlich die Gesamtsicherheit des Internets verringert wurde. Für den *honeynet*-Betreiber würde dies bedeuten, dass er eine besondere Verantwortung gegenüber Personen übernimmt, deren Systeme durch *blackhats* von dem *honeynet* aus angegriffen werden. Betrachtet man jedoch das aktuelle Sicherheitsniveau von Systemen im Internet, scheint dieses dermaßen niedrig zu sein, dass die *honeypot*-Systeme möglicherweise sogar *über* dem durchschnittlichen Sicherheitsniveau liegen und somit die Gesamtsicherheit des Internet *verbessern*.

Bedenkt man weiterhin, dass (1) die Systeme des *honeynet* mit ihrer mangelnden Sicherheit die Population potentieller Opfer im Internet vergrößert und damit das Risiko des einzelnen verringert, Opfer einer Attacke zu werden, und (2) gleichzeitig auch durch die strenge Kontrolle des ausgehenden Verkehrs der Anteil an Rechnern verringert wird, die ohne weiteres als Plattform für weitere Angriffe genutzt werden können, kann durchaus davon ausgegangen werden, dass das *honeynet* allein durch seine Existenz das Risiko dritter, zum Opfer eines Angriffs aus dem Internet zu werden, leicht verringert. Weiterhin besteht die berechtigte Hoffnung, dass die mit dem *honeynet* gesammelten Ergebnisse mittelfristig zur Steigerung der Gesamtsicherheit des Internet beitragen. Ethisch scheint daher der Betrieb eines *honeynet* gegenüber der Gesellschaft vertretbar zu sein.

5.2 Strafrechtliche Aspekte

Rechtlich könnte eine straf- oder zivilrechtliche Haftung bestehen, wenn das *honeynet* Teil eines Angriffs gegen Dritte ist. Strafrechtlich könnte der Betrieb eines *honeynet* gemäß § 27 StGB Beihilfe zu Straftaten sein, die von einem *blackhat* über das *honeynet* verübt werden. Beihilfe kann jedoch nur durch eine *vorsätzliche* Hilfeleistung erfolgen. Dies ist nicht gegeben. Stattdessen wird ja versucht, mittels Snort_inline und `iptables` Beeinträchtigungen anderer Systeme von dem *honeynet* aus zu verhindern. Weiterhin ist der Betrieb eines Rechners mit dem Sicherheitsniveau der Honeypots völlig sozialadäquat, d.h. er lädt nicht etwa *blackhats* zum Missbrauch ein. Der Sicherheitsstandard der Systeme liegt sogar über dem vieler anderer am Internet angeschlossener Systeme. Strafrechtlich ist der Betrieb des *honeynet* daher unbedenklich.

5.3 Zivilrechtliche Aspekte

Eine zivilrechtliche Haftung für durch den Angreifer vom *honeynet* aus gegenüber anderen verursachten Schäden käme primär aus § 823 I BGB in Betracht. Dafür müsste aber ein von dem *blackhat* verursachter Schaden den Betreibern des *honeynet* zurechenbar sein. Da die Betreiber des *honeynet* jedoch nicht aktiv andere Rechner schädigen, kommt nur eine Haftung aus dem Unterlassen der Absicherung des *honeynet* in Betracht. Dazu müsste der Betreiber insbesondere eine Garantenstellung innehaben, die ihn verpflichtet, Gefahren aus dem *honeynet* einzuschränken. Diese könnte sich aus einer Verkehrssicherungspflicht ergeben. Die allgemeine Verkehrssicherungspflicht besagt, dass jedermann, der in seinem Verantwortungsbereich Gefahren schafft oder andauern lässt, die notwendigen Vorkehrungen treffen muss, die im Rahmen des wirtschaftlich zumutbaren geeignet sind, Gefahren von Dritten abzuwenden.

Welche Maßnahmen notwendig und zumutbar sind, ist eine Wertungsfrage. Bisher scheint es die Gesellschaft inklusive der Rechtsprechung abzulehnen, Betreiber von mit dem Internet verbundenen Systemen für Schäden zur Verantwortung zu ziehen, die durch die Unsicherheit dieser Systeme entstanden sind. Daher ist davon auszugehen, dass es als nicht notwendig oder als nicht zumutbar angesehen wird, mit dem Internet verbundene Systeme abzusichern oder zu überwachen. Das *honeynet* wird jedoch umfangreich überwacht. Insbesondere wird besonderer Aufwand betrieben, um Schaden von Dritten abzuwenden. Damit hat der Betreiber des *honeynet*, selbst wenn man eine Verkehrssicherungspflicht für mit dem Internet verbundene Systeme annähme, diese erfüllt.

5.4 Datenschutzrechtliche Aspekte

Bei der Frage, wie der Betrieb des *honeynet* gegenüber den eindringenden *blackhats* zu bewerten ist, ist vor allem fraglich, in wie weit ein angreifender *blackhat* nicht unwissend zum Teil eines Experiments gemacht wird. Der *blackhat*–Community ist durchaus bewusst, dass zum einen *honeynets* existieren und dass zum anderen Systemadministratoren soweit sich entsprechende Gelegenheiten bieten, alle Schritte von *blackhats* aufzeichnen und umfangreiche forensische Untersuchungen durchführen. Deshalb kann davon ausgegangen werden, dass *blackhats* billigend in Kauf nehmen, dass ihr Handeln aufgezeichnet und untersucht wird. Auch rechtlich bestehen aus dieser Hinsicht keine Bedenken. Insbesondere ist nicht davon auszugehen, dass das *honeynet* mit personenbezogenen Daten in Kontakt kommt. Somit ist der Betrieb aus datenschutzrechtlicher Sicht unbedenklich.

6 Zukünftige Arbeiten

Nach einer Testphase mit dem in Abschnitt 3 beschriebenen *honeynet* soll in einer folgenden Phase eine Erweiterung um ein virtuelles *honeynet* auf Basis von User-mode Linux [use04] oder VMware [vmw04] erfolgen. Bei einem virtuellen *honeynet* werden virtuelle

Maschinen benutzt, um *honeypots* aufzusetzen. So erreicht man eine einfachere Administration und Wartbarkeit, da man auf einem Computer mehrere *honeypots* verwalten kann.

User-mode Linux [use04] ist eine Portierung von Linux auf Linux, die es ermöglicht, mehrere Instanzen des Linux-Kernels auf einem Linux-Rechner zu starten. Damit kann man ohne großen Aufwand mehrere *honeypots* simulieren. Allerdings ist man mit dieser Lösung auf Linux als simuliertes Betriebssystem beschränkt. Dies kann man durch Verwendung von VMware [vmw04] vermeiden, einem kommerziellen System zur Emulation von Computern. Mit VMware kann auf den virtuellen Maschinen unter anderem Microsoft Windows (Windows 3.1x bis Windows Server 2003), Linux (Kernel-Versionen 2.2, 2.4 und 2.6.) und Novell NetWare betrieben werden. Allerdings sind hierbei die Anforderungen an die Hardware wesentlich höher und die Lizenzkosten relativ hoch. Aufgrund der erhöhten Aktivität von *blackhats* in Bezug auf den Internet Information Server von Microsoft scheint die Einrichtung eines *honeypot* mit Microsoft Windows eine sinnvolle Erweiterung des *honeynet* zu sein. Daher wird dies als nächstes angestrebt.

Abbildung 3 zeigt die geplante Erweiterung um virtuelle Maschinen. Die genaue Zusammensetzung des virtuellen *honeynet* ist noch in Planung, und deshalb ist noch keine Aussage zur genauen Konfiguration oder den verwendeten Plattformen möglich.

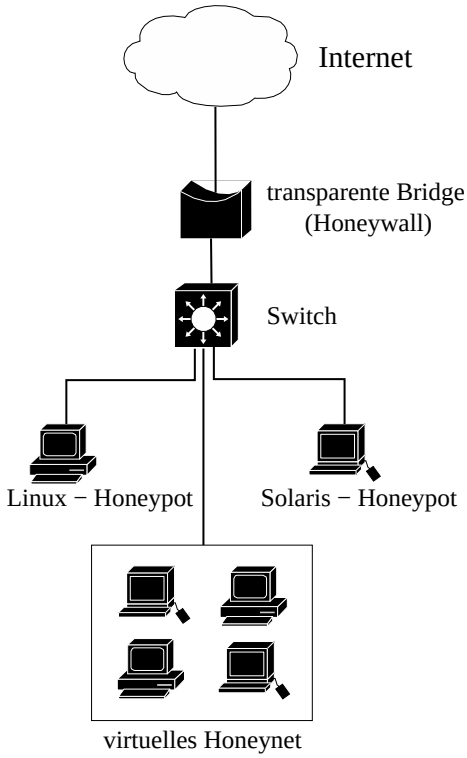


Abbildung 3: Geplante Konfiguration des *honeynet* an der RWTH Aachen.

Außer der Erweiterung des bestehenden *honeynet* um virtuelle *honeypots* sollen in nächster Zeit Programme entwickelt werden, mit denen die Analyse der gesammelten Daten vereinfacht werden kann. Es existiert zwar ein Web-Interface, mit dessen Hilfe die Daten von Sebek ausgewertet werden können, aber eine Weiterentwicklung dieses System sowie die Erstellung weiterer Programme zur Datenanalyse wird angestrebt.

Zudem ist auch eine Weiterentwicklung der schon bestehenden Programme zur Datensteuerung und -sammlung (insbesondere Snort_inline und Sebek) angedacht. Die Veröffentlichung des Quellcodes der verwendeten Programme führt nämlich dazu, dass *blackhats* Methoden entwickeln können, um *honeynets* zu entdecken. Dies ist in der Vergangenheit mehrmals dokumentiert worden [Co03, Co04]. Gleichzeitig führt die Weiterentwicklung von *rootkits* dazu, dass in Sebek immer neue Ideen einfließen, um das Programm noch transparenter in das System zu integrieren. Es findet also ein ständiger Wettlauf um die neuesten Methoden zur Entdeckung und Verschleierung von *honeynets* statt, der eine Weiterentwicklung auf diesem Gebiet unerlässlich macht.

Die gegenwärtig bei *honeynets* verwendete Methodik schränkt das untersuchbare Verhalten der *blackhats* sehr ein. Es werden praktisch nur Angriffe auf mäßig gesicherte Unix-Serversysteme ohne Anwendungen mit komplexen Benutzerschnittstellen und ohne aktive Nutzung untersucht. Weiterhin können praktisch nur „zufällige“ Angreifer detektiert werden: Es handelt sich also in der Regel entweder um automatisierte Angriffe von Würmern und dergleichen oder um Angriffe von *blackhats*, die ihre Ziele mehr oder weniger nach dem Zufallsprinzip aussuchen. Dass ein *blackhat*, der seine Ziele planmäßig auswählt, Systeme im *honeynet* angreifen wird, scheint unwahrscheinlich.

Die Frage, wie die Bereiche der abgedeckten *blackhat*-Aktivitäten ausgedehnt werden können, muss weiter diskutiert werden. Ein erster Ansatz besteht darin, auf den *honeypots* Software wie PHP-Nuke [php04] zu installieren, die es ermöglicht, Angriffe auf Webapplikationen zu beobachten. Problematisch ist hierbei allerdings die Frage, ob und in welcher Hinsicht die Simulation von Benutzeraktivität sinnvoll ist. Wie mit Hilfe von *honeynet*-Technologie andere Aktivitäten beobachtet werden können, insbesondere Ressourcendiebstahl in Form von *spam relaying* oder Proxy-Mißbrauch und Angriffe auf Desktop Systeme in Form von Viren, Würmern, *spyware*, Remote Access Trojanern und Dialern, ist noch ungeklärt.

Danksagungen

Die Autoren sind den Verantwortlichen des Rechenzentrums der RWTH Aachen, Jens Hektor und Christian Bischof, zu großem Dank verpflichtet. Dank geht ebenfalls an Lexi Pimenidis für die Unterstützung beim Aufsetzen des Linux-*honeypot* und an Ulrike Freiling für den Hinweis auf [Mi87] und weitere wertvolle Verbesserungsvorschläge zur Textgestaltung. Thorsten Holz wurde unterstützt als Forschungsstudent im Rahmen des DFG-Graduiertenkollegs „Software für mobile Kommunikationssysteme“ an der RWTH Aachen.

Literatur

[aci] ACID Homepage. Internet: `http://acidlab.sourceforge.net/`.

[CE02] CERT/CC. CERT Advisory CA-2002-01: Exploitation of Vulnerability in CDE Subprocess Control Service. Januar 2002. `http://www.cert.org/advisories/CA-2002-01.html`.

[Ch90] Cheswick, W. An evening with berferd in which a cracker is lured, endured, and studied. USENIX proceedings, Jan 20, 1990. 1990.

[Co03] Corey, J. Local Honeypot Identification. September 2003. `http://www.phrack.org/fakes/p62/p62-0x07.txt`.

[Co04] Corey, J. Advanced Honey Pot Identification. Januar 2004. `http://www.phrack.org/fakes/p63/p63-0x09.txt`.

[DHK04] Dornseif, M., Holz, T., und Klein, C. N.: Nosebreak – attacking honeynets. In: *Proceeding of 5th Annual IEEE Information Assurance Workshop*. United States Military Academy, West Point, New York. Juni 2004.

[hon04a] The Honeynet Project. Internet: `http://www.honeynet.org/`. 2004.

[hon04b] The Honeynet Project Statistics. Internet: `http://www.honeynet.org/papers/stats/`. 2004.

[Lo01] Longman Group Ltd. (Hrsg.): *Longman Dictionary of Contemporary English*. Langenscheidt-Longman GmbH. München. Dritte, erweiterte auflage. 2001.

[Mc03a] McCarty, B.: Automated identity theft. *IEEE Security & Privacy*. 1(5):89–92. 2003.

[Mc03b] McCarty, B.: Botnets: Big and bigger. *IEEE Security & Privacy*. 1(4):87–90. 2003.

[Mi87] Milne, A. A.: *Pu der Bär*. Cecilie Dressler Verlag. Hamburg. 1987. Aus dem Englischen übersetzt von Harry Rowohlt.

[php04] PHP-Nuke Homepage. Internet: `http://phpnuke.org/`. 2004.

[Sp02] Spitzner, L.: *Honeypots: Tracking Hackers*. Addison-Wesley. 2002.

[St88] Stoll, C.: Stalking the wily hacker. *CACM*. 31(5):484–497. 1988.

[swa04] Swatch Homepage. Internet: `http://swatch.sourceforge.net/`. 2004.

[Th03a] The Honeynet Project. Honeytokens: The Other Honeypot. Juli 2003. `http://www.securityfocus.com/infocus/1713`.

[Th03b] The Honeynet Project. Know your Enemy: Sebek. November 2003. `http://www.honeynet.org/papers/sebek.pdf`.

[use04] User-Mode Linux. Internet: `http://user-mode-linux.sourceforge.net/`. 2004.

[vmw04] VMware Homepage. Internet: `http://www.vmware.com/`. 2004.

Foundations for Intrusion Prevention

Shai Rubin, Ian D. Alderman, David W. Parter, and Mary K. Vernon
University of Wisconsin, Madison

Abstract: We propose an infrastructure that helps a system administrator to identify a newly published vulnerability on the site hosts and to evaluate the vulnerability's threat with respect to the administrator's security priorities. The infrastructure foundation is the vulnerability semantics, a small set of attributes for vulnerability definition. We demonstrate that with a few attributes it is possible to define the majority of the known vulnerabilities in a way that (i) facilitates their accurate identification, and (ii) enables the administrator to rank the vulnerabilities found according to the organization's security priorities. A large scale experiment demonstrates that our infrastructure can find significant vulnerabilities even in a site with a high security awareness.

1 Introduction

To date, worms and other widespread network-based attacks have gained unauthorized access to many organizations' hosts by exploiting known vulnerabilities. For example, the 'Code Red' worm—which spread in July and August 2001—exploited a known buffer overflow vulnerability on thousands of hosts that were running Microsoft servers [CCZ02, SPW02]. The damage could have been avoided if the host administrators had installed the software patch that was announced by Microsoft approximately one month before the worm was released.

While intrusion detection systems (e.g., [Ro99]) can detect an attack such as the Code Red, they usually cannot prevent an attack from occurring. To prevent attacks, we envision an *Intrusion Prevention Infrastructure* (IPI), depicted in Figure 1. As soon as a new vulnerability is published, the IPI detects the vulnerability on every system host, estimates the vulnerability threat with respect to the site security priorities, and aids the administrator in repairing the vulnerability. The infrastructure core is a vulnerability database which provides a vulnerability definition that facilitates and integrates the other infrastructure components: an accurate audit tool, a site-customizable threat analyzer, and a repair tool that derives repair options from the vulnerability definition.

This paper focuses on two IPI components: the vulnerability database, with its language for defining vulnerabilities, and the threat analyzer. More particularly, we make the following contributions:

Vulnerability semantics (Section 4). We propose a formal language for vulnerability definition. The language core comprises two sets of attributes: the *presence* attributes define the characteristics of a vulnerable host and the *threat* attributes define the severity of the

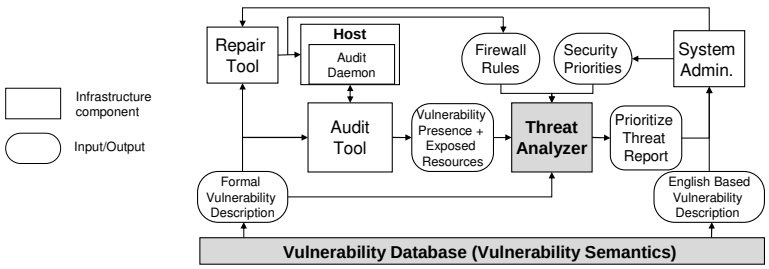


Figure 1: Intrusion Prevention Infrastructure. In this paper, we develop the threat analyzer and the vulnerability semantics.

vulnerability and the difficulty of exploiting it. Since our language is formal, it facilitates unambiguous vulnerability definition and accurate audit; since it is simple, repair options can be derived from the vulnerability description by nullifying the presence or threat conditions. Unlike other languages (e.g., [MIb]), our attributes do not rely on specific host or operating-system features, so they support an efficient audit. At the same time, our language enables threat definition that depends on the host configuration. For example, one can define a conditional threat: "if file F exists, the threat is high, else the threat is low".

Threat analyzer (Section 5). We develop a *site-customizable* threat analyzer that ranks vulnerabilities according to the site security preferences. Our first threat measure is the *attack severity*, a quantitative measure of the resources a vulnerability exposes. To identify these resources, the analyzer uses both the vulnerability semantics and the audit results; to quantify them, it uses a resource ranking which is customized according to the site security preferences. The second threat measure is the *attack difficulty*, an estimate of the attacker ability to launch a successful attack. We split the difficulty definition into two components. The *inherent* component represents the vulnerability difficulty by comparing the vulnerability to others that are similar. The *site-specific* difficulty reflects the difficulty of exploiting the vulnerability on a particular host.

IPI case study (Section 6). We study the IPI feasibility, its necessity, and its impact on security. We partially implemented the IPI audit and threat tools, and we used the tools to conduct a long term experiment on a real site. Our site has 1500 hosts, maintains strong configuration management (only software that is authorized by the site administrator can be installed), and has a security officer who devotes his entire time to ensuring the site security. The results demonstrate the IPI's capability of finding long-lived vulnerabilities, even for a site with such a high security awareness. The study demonstrates the value of a site-customizable threat analyzer for consistently measuring threat levels over a long period of time. The threat analyzer measures both the impact of repairing vulnerabilities and the threat levels of vulnerabilities that were not repaired because they are allowed by the site security policy.

2 Related Work

Vulnerability databases. There are several vulnerability databases that contain hundreds [CE, De] to thousands [Th, Se, In] of vulnerability descriptions. These databases primarily use a natural language (i.e., English) to describe vulnerabilities. As we illustrate in the next section, such descriptions are subject to different interpretations by different people. Furthermore, a natural language description is difficult to use by programs that can automate the audit, repair, and threat analysis processes. In contrast, our language is formal, so it enables unambiguous vulnerability definition and facilitates the use of automatic tools.

Vulnerability definitions and languages. Cuppens et al. [CO00] propose a formal language to describe vulnerabilities. However, they focus on vulnerability composition to detect a series of exploits that form a significant attack [CM02]. Their language does not include semantics that support accurate audit or repair of a vulnerability, and is limited in the threat characterization for each vulnerability.

Parallel to our work, MITRE corporation is developing the Open Vulnerability Assessment Language (OVAL) [MIb]. OVAL uses SQL to define a host-dependent test that determines whether or not a host is vulnerable. Unlike OVAL, our higher-level definition supports threat analysis and vulnerability repair in addition to testing for the presence of the vulnerability. Furthermore, our presence attributes facilitate a network-based audit technique that is host independent. Network-based auditing is identical for all hosts and operating systems, and is needed for efficient testing.

Quantitative threat analysis. Existing databases and audit tools specify, for each vulnerability, a *fixed* quantitative threat level like 'high', or 'low' (the CERT database uses fixed numeric ratings). As far as we can tell, the meaning of these different levels is not precisely defined in any of these tools and there are inconsistencies among the tools (e.g., the threat of CVE-2000-0614 is 'high' by ICAT [Th] but 'medium' by ISS [In]). In our threat model, the threat level depends on the identity of the resources the vulnerability exposes rather than on a fixed value given by the vulnerability definer. This means that (i) a vulnerability can yield different threat levels when found on different vulnerable hosts (e.g., web server vs. user workstation), and (ii) different vulnerabilities that expose the same resources have the same threat level.

Ortalo et al. [ODK99] define a vulnerability threat based on the difficulty of exploiting the vulnerability; they assign a fixed difficulty level to each vulnerability. Our threat definition is broader: it is based on the resources the vulnerability exposes in addition to the vulnerability difficulty. Furthermore, our difficulty definition is customizable: it takes into account site-dependent factors such as host configuration and firewall protection.

Audit and testing technologies. Two types of audit technologies have been developed: (i) host configuration checkers that search for host configuration flaws that might be exploited by attackers [FPA98, FS90, KS94, Or99, SSH93, ZL96], and (ii) remote network-based vulnerability scanners which test for vulnerabilities exclusively using the network protocols [De, Mu95, In, To]. Since neither of these techniques alone can conclusively identify all vulnerabilities, these tools produce many false positives. To increase accuracy, the IPI audit methodology is based on precise semantics, and uses information from both host and

network audit techniques. However, we leave the detailed implementation of this audit methodology to the future.

3 Motivation: Imprecise Vulnerability Definition

We demonstrate that a natural-language vulnerability definition—used by all contemporary vulnerability databases—lacks the ability to define a vulnerable host and the vulnerability threat. Consider the natural language description from the ICAT database [Th]:

***GuestBook* vulnerability (CAN-1999-1053 [MIa]).** The *guestbook.pl* is a CGI script that enables visitors to sign an online guest book. A security hole, when guestbook.pl is run on Apache 1.3.9 and possibly other versions, enables users to insert, instead of their names, shell commands—called Server-Side Includes (SSI). Consequently, attackers can execute arbitrary commands on the host. However, such commands will be executed only if the Apache web server enables SSI, and guestbook.pl is configured to accept HTML tags.

This description leads to three possible definitions of a vulnerable host. A *first* definition could be: (i) the Apache web server is running, (ii) the Apache web server is configured to enable SSI, (iii) at least one user is using the guestbook.pl script, and (iv) HTML tags are enabled in that guestbook.pl script. A *second,* broader definition can ignore the third and the fourth conditions. After all, since any user at any time can install the guestbook.pl script, a host can be viewed as vulnerable even if the script is not currently installed. Lastly, one might claim that the *GuestBook* vulnerability represents a family of vulnerabilities which can be described by a *third* definition broader than the first and narrower than the second—conditions (i), (ii), and (iii'): there exists a script accessible by Apache that does not sanitize SSI directives in its input.

Natural language description is not only ambiguous with respect to the presence of the vulnerability, but also imprecise with respect to the vulnerability threat. First, the commands the attacker inserts are executed under the privileges of the web server account. If Apache is running with 'root' privileges, the threat is higher than if it is running with 'nobody' privileges. Second, the threat depends on the conditions that define the presence of the vulnerability. *GuestBook* poses an immediate threat according to the *first* and *third* definitions above, but only a potential threat according to the *second* definition.

The *GuestBook* description lacks two other important properties a vulnerability definition must possess. First, a definition must facilitate accurate audit. We cannot use a vague phrase like 'possible other versions' to implement an accurate audit procedure. Second, to facilitate rapid repair, the definition must include simple and accurate repair options.

4 Vulnerability Semantics

A system administrator considers a vulnerability in the context of a system; an administrator may choose not to repair the *GuestBook* vulnerability, because the vulnerable host

is behind a firewall. The goal of the vulnerability semantics is to specify how various attributes influence the meaning of a vulnerability, and enable the audit tool and threat analyzer to fill in the attributes with values. For example, the semantics specifies the vulnerability protocol, and the audit tool fills in whether this protocol is blocked by a firewall for a given host, or, in the *GuestBook* case, the semantics specifies that the threat depends on the account that runs the Apache process, while the audit tool determines who the account owner is (root vs. nobody).

To achieve flexibility in definition, the vulnerability semantics contains four attribute sets. The *identification* attributes specify key characteristics of the vulnerability in a human-readable form. The *presence* attributes define sufficient and necessary conditions to determine whether a host is vulnerable. The *threat* attributes define system resources compromised by the vulnerability and the difficulty of exploiting it. Lastly, the *repair* attributes define how to immunize a vulnerable host; this set is beyond the scope of this paper.

4.1 Vulnerability Identification

This is a set of attributes that provides a human-readable description of the vulnerability; it is not a complete set of properties that characterize a vulnerable host. The attributes in this set are (Table 1): (i) *Name*: provides a vulnerability-unique identifier (e.g., CVE name [MIa]) (ii) *Operating system*: provides a list of operating systems that are known to be vulnerable; (iii) *Vulnerable unit*: provides a list of the software components that should be repaired; (iv) *Configuration*: provides an informal description of configuration

Identification	Name	Apache GuestBook (CAN-1999-1053)	Telnet cleartext passwords (CAN-1999-0619)
	Operating system	ANY	ANY
	Vulnerable unit	Apache version 1.3.9, guestbook.pl.	Telnet
	Configuration	Server Sides Include (SSI) on.	
	Protocol,Port	RFC: 2616/HTTP,80+'any'	RFC: 854/TELNET,23
Presence	Condition Set	P_1: *serviceRunning()* P_2: $package{=}Apache$ P_3: *content*(*config_file*, $[Includes\|XBitHack]$)	P_1: *serviceRunning()*
	Verification Hints	$H_{3,UNIX}$: (*config_file*= '/etc/httpd/conf/httpd.conf')	
Threat	Exposed Resources	if (version=1.3.9) then <*CIA,SA*> else UNKNOWN	<*CIA,NPA*>

	Expected Time to Exposure (days)	if (access(guestbook.pl) or *content(guestbook.pl, 'html=1'))* then 0;
		else
		TimeUntil(guestbook.pl installed)=30
			TimeUntil(a user uses TELNET from sniffed network)=7.

	Expected Time to Attack (days)	7	0.5

Table 1: Examples of vulnerabilities definitions.

settings of the vulnerable units, like Apache with SSI turned on; and (v) *Protocol/Port*: specifies the name and RFC number of the vulnerability's application-level protocol (e.g., HTTP/2616) and the port number the vulnerability uses. The port is either an integer (e.g., 21 for FTP), or the word "any" if the protocol standard does not dictate a specific port. In the latter case, we also specify the default port (e.g., 80 for HTTP).

4.2 Presence Semantics

The vulnerability presence semantics has two components. The first is a set of predicates, the *condition set*, that specifies necessary, sufficient, and verifiable conditions that must hold in order for a host to be vulnerable. The second is a set of *verification hints* the audit tool can use to verify these conditions.

4.2.1 Condition Sets.

Out of the hundreds of vulnerabilities we reviewed, 90% of them can be defined using the predicates in Table 2. We use predicates that assert configurational properties only, functional properties only, or both.

Since audit tools usually use a functional test—a test that uses the means the attacker uses [De]—to verify functional predicates, using functional predicates in the vulnerability presence definition is preferable. When this is not feasible, because a functional test is either unsafe (e.g., requires buffer overflow exploitation) or inefficient, configurational predicates can be used. For example, the most efficient way to verify that the SSH service enables Kerberos authentication is by checking the SSH configuration file (P_4 in *SSH* in Table 5). To facilitate flexible and efficient audit, we permit two condition sets; in the *WebSitePro* vulnerability (Table 5) we specified two (logically equivalent) sets, and the audit tool can select the set that it believes is more efficient to check.

The condition set relies on the assumption that the site uses an official software release. If

Predicate	**Evaluated to true if and only if**	**Type**
serviceRunning()	the service specified by the Protocol attribute is running.	functional
access(application, p)	*application* is accessible using the service specified by the Protocol attribute **given** the predicate p is true.	
$Login(user, password)$	login with the pair (*user,password*) succeeded.	
content(file, regExp)	regular expression *regExp* is found in $file$.	config.
version() < v	version of the network service package is smaller than v. Other binary operators (e.g., $=, >, \leq$) are supported.	
package() = name	the name of the software package that provides the network service is $name$.	
hostResponse(command,regExp)	after executing command com the respond contains the regular expression $regExp$.	config. + functional
netResponse(message, regExp)	after sending the message msg the respond contains the regular expression $regExp$.	

Table 2: Predefined basic and compound predicates for presence and threat semantics.

the site changed the software (e.g., by modifying the version number, service banner, etc.), we classify such changes as user software that is not widely distributed and is not covered by the standard IPI audit.

4.2.2 Verification Hints.

To help the audit tool verify the predicates in the condition set, we add verification hints to the vulnerability presence definition. We define two categories of verification hints: *informational* and *logical*.

An informational hint suggests the location of a resource that the audit tool can use to verify predicates. For example, $H_{3,UNIX}$ in the *GuestBook* definition (Table 1) points to the Apache configuration file required by predicate P_3 (an additional hint can be provided for a Windows machine). A logical hint specifies either a *netResponse* or a *hostResponse* predicate that can be used to verify the predicates in the condition set. For example, to verify the predicates P_1 and P_2 in *Perl-In-CGI* (Table 5), the audit tool can use $H_{1,2}$ that specifies a *netResponse* predicate that if it holds, P_1 and P_2 hold too. Note that $H_{1,2}$ is a *sufficient* hint only: if it fails, P_1 and P_2 may still be true. We can specify *necessary and sufficient* hints; for example, $H_{2,3,UNIX}$ in the *LPRng* example (Table 5) specifies a *hostResponse* predicate which holds if and only if both P_2 and P_3 hold. To increase the audit flexibility, it is possible to specify more than one hint for predicates. For example, $H_{2,3,UNIX}$ (a host dependent hint) and $H_{2,3}$ (a host independent hint) can be used to verify P_2 and P_3 in the *SSH* example (Table 5).

There is no clear border between necessary-and-sufficient hints and predicates in the condition set; the vulnerability definer has the power to 'upgrade' such hints into predicates. However, our guideline is to leave the condition set, to the extent possible, host independent.

4.3 Threat Semantics

The threat semantics has two components: the *vulnerability severity* and the *vulnerability difficulty*. A novel feature of our semantics is the use of the same predicates that define presence to define the threat attributes.

4.3.1 Vulnerability Severity.

The vulnerability severity is the extent to which an attacker gains unauthorized privileges on various system resources. Hence, the *Exposed Resources* attribute specifies privileges on resources that the attacker gains. In the *Telnet* example (Table 1), the attacker gains Confidentiality, Integrity and Availability (CIA) privileges (i.e., reading, modifying, and blocking access, respectively) to all resources associated with a particular non-privileged user. In *WebSitePro* (Table 5), the attacker gains only confidentiality privileges to the directory names of the web server.

Reg. Expression	Resource Definition
[*'PA'* | *'NPA'* | *'SA'* | *UserName*]	Denotes resources associated with user accounts. The strings '*PA*' and '*NPA*' distinguish between privileged accounts (e.g., root or administrator) and regular user accounts. '*SA*' denotes an account of the owner of the vulnerability network service (e.g., *SA=root* if the web server is running with the root privileges). *UserName* is used to specify a specific user account (e.g., 'john' or 'guest').
File	Specifies a regular expression for a set of files or directories.
HostIp(.*RFC*)	Denotes the physical device associated with an IP address *HostIp* (e.g., a workstation, a printer). *RFC* is optional and specifies the protocol of the compromised service (e.g., 2616 for HTTP).
root_directoy	The root directory associated with the vulnerability network service (e.g., root directory of a web server).
[*'user_data'*| *'file_data'*| *'directory_data'*]	Denotes compromised *data* about users' files, or directories.
config_file	Denotes the configuration file of a vulnerable network service.

Table 3: Predefined resources.

We used the resources listed in Table 3 to define hundreds of vulnerabilities. Besides facilitating the customizable threat model, our 'exposed resources' approach has two other advantages. First, it enables us to specify resources in a fine-grained manner. For example, <*CIA,john*> specifies *CIA* privileges of the account with the name *john*, while <*CIA,/home/john/**> specifies *CIA* privileges to all files under the directory '*/home/john*'. Second, using presence predicates to express conditions on the exposed resources attribute further increases our definition expressiveness. In the *GuestBook* example, it is not clear whether Apache versions other than 1.3.9 are vulnerable too (Section 3). So, we put the *version* predicate in the exposed resources attribute rather than in the condition set (Table 1). The result is a vulnerability definition that is not limited to a particular version when it is unknown whether other versions are vulnerable too (as more knowledge becomes available, the definition may change).

4.3.2 Attack Difficulty.

Our difficulty definition is based on two observations. First, some vulnerabilities are intrinsically more difficult to exploit than others. The *GuestBook* can be exploited by any user familiar with a few UNIX commands, whereas exploiting a buffer overflow on an SNMP daemon (e.g., CAN-2002-0017) requires programming and network expertise. Second, the vulnerability difficulty is also host dependent. For example, it is very difficult to exploit a host behind a firewall. We use two attributes to define the intrinsic and host-specific difficulty.

The *Expected Time to Exposure* (ETE)—which captures the host-dependent difficulty—is the time it takes for a host to be vulnerable. In most cases, the ETE is 0 if all the presence predicates hold (e.g., *LPRng* in Table 5). In cases where additional user activity is required before a host becomes vulnerable, the ETE gets a non-zero value. For example, a user must install the *guestbook.pl* script before the *GuestBook* attack can take place. In such cases, the attribute value is '*timeUntil*(user-activity-description)=default' (Table 1). The

IPI database contains the default value for the *timeUntil* predicate, but the administrator can customize the value according to her familiarity with her system.

The *Expected Time to Attack* (ETA)—which captures the intrinsic vulnerability difficulty—is the expected time until an attacker gains unauthorized privileges, given that the host is vulnerable (i.e., no more user activity is required for the host to become vulnerable). In the case of *LPRng* (Table 5), the ETA is the time it takes to develop code that exploits the buffer overflow. For password sniffing (*Telnet* in Table 1), it is the time it takes to login *after* a user/password pair is known, which is essentially zero. For *GuestBook* (Table 1), it is the time until the guest book owner will upload the guest book and the attacker code will be executed (estimated as a week). We give examples of assigning the ETA value to the vulnerabilities we found in Section 6.1.

The vulnerability definer assigns both the ETA and ETE values (e.g., 1, 7 days). These values can be conservative, and can be refined from measures of exploit activity on the Internet (e.g., [BAMF01]). However, it is important to be consistent, and to assign similar values to vulnerabilities with similar difficulties.

4.4 Repair Semantics

The repair semantics is beyond the scope of this paper. However, since the presence semantics defines a set of necessary and sufficient conditions a vulnerable host must possess, to repair the host is to nullify at least one of the conditions. For example, software upgrade usually changes the software version number, so it nullifies the version predicate. This observation makes it easier to present to the system administrator the repair options, as we illustrate in Section 6.2.

5 Threat Analysis

In the previous section we presented the threat semantics for defining the vulnerability threat. Here, we discuss how the threat analyzer uses the semantics to estimate the vulnerability threat according to the site security priorities.

5.1 Ranking Severity

We assign to every resource in the system (e.g., file, host, printer) an 'exposure cost' that represents the damage to the organization if the resource is compromised (e.g., the cost for the company if its front web page is hacked). We assume that the audit tool reveals the exposed resources of each vulnerability, so we can map each exposed resource to its cost. We define the total severity of a vulnerability as the total cost of the resources the vulnerability exposes. More formally, the total severity of vulnerability v, TS_v, is defined as:

$$TS_v = \sum_{h \text{ vulnerable to } v} EC(h, <P, R>)$$

where h is a system host (e.g., server, printer), $<P, R>$ is a privileges-resource pair (Section 4.3.1) the attacker gains after exploiting v on h, and EC is the site *Exposure-Cost* function which maps the exposed resource $<P, R>$ on the host h to its exposure cost. The following properties make TS_v a good candidate for measuring severity:

1. The EC map counts for both the type of the vulnerable host and the resources the vulnerability exposes. This enables threat definition that is relative to both the vulnerability characteristics and the type of the vulnerable host. For example, EC can assign a higher cost to resources on an AFS server than to resources on a user workstation, so the threat of a given vulnerability is higher on the AFS server.
2. Usually, an administrator repairs a vulnerability by applying a patch to *all* vulnerable hosts. Since TS_v sums the costs over all hosts, it concisely shows to the administrator the 'security benefit' from his actions.
3. TS_v ensures that a resource is protected only if it is not exposed from any host. If v exposes the same resource from two different hosts, a situation that occurs in the presence of a shared file system, TS_v counts the resource twice. One might claim that this 'overestimates' v's severity, but it still reflects that removing the vulnerability necessitates repairing more than one host.
4. TS_v facilitates a simple and intuitive comparison between vulnerabilities. If v_1 and v_2 are vulnerabilities that expose the same resources on the same hosts, the definition ensures that TS_{v_1}=TS_{v_2}. Furthermore, if v_1 and v_2 expose different resources but still TS_{v_1}=TS_{v_2}, then even though v_1 and v_2 are different, their severity is still the same (with respect to the EC chosen). Similarly, because TS_v is additive, the administrator can easily understand that if TS_{v_1}=$100TS_{v_2}$ then v_1 is 100 times more severe than v_2.

5.2 Building an Exposure Cost Map.

We build the EC map by (i) qualitatively ranking sets of hosts according to the site security priorities, (ii) qualitatively ranking sets of exposed resources according to the site security priorities, and (iii) combining and quantifying the two rankings in a consistent way.

The IPI has a repository of rankings for hosts and exposed resources; any host ranking can be combined with any resource ranking. So, the administrator can select the rankings that come close to her system characteristics, refine them, and quickly build an EC function according to her security preferences. This process is simple enough to be done from scratch by the administrator, if desired.

To rank hosts, we create the $Hosts$ partial order which *qualitatively* ranks sets of hosts according to their exposure cost as defined by the site security priorities. Such a simple qualitative ranking—which fits many organizations—is illustrated in Figure 2(a). The ranking contains three levels, where the organizations' servers have the highest exposure cost and the employees' workstations the lowest.

To rank resources, we create the *Exposed Resources* (ERs) partial order which ranks sets

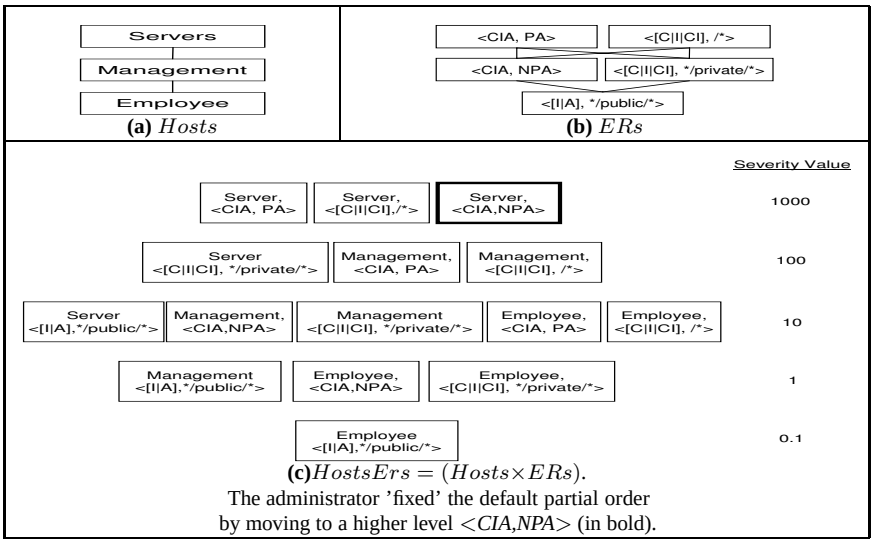


Figure 2: Building severity cost function.

of exposed resources according to the site security priorities, without considering the resource host component. For example, a site that emphasizes confidentiality and integrity can rank exposed resources using the ERs from Figure 2(b). At the lowest cost level, the site ranks availability and integrity of files (i.e., vulnerabilities that expose those files) in *public* directories. Next, the site ranks vulnerabilities that compromise non-privileged accounts. Since, according to the site policy, compromising user's private files is equivalent to compromising the user's account, the site ranks also into the second level vulnerabilities that expose either confidentiality or integrity (but not availability) of files in private directories. Last, the site assigns the highest exposure cost to vulnerabilities that expose privileged accounts, or expose the confidentiality or integrity of files which do not fall under the previous categories[1].

Both *Hosts* and ERs are neither unique nor as detailed as possible. As the system administrator becomes more familiar with the process, he can further partition the $Hosts$ or ERs. For example, he can add more server levels to $Hosts$, or move a specific exposed resource, like the availability of the company's main web page, to a higher level in ERs.

The next step is to build the product partial order: $Hosts \times ERs = HostsERs$. $HostsERs$ preserves the consistency of both $Hosts$ and ERs. But, it may not reflect exposure rankings that depend on a combination of hosts and resources. For example, servers are considered more valuable than user workstations, and in many cases compromising a non-privileged account on a server may cause the same damage as compromising a privileged account on the server. Hence, the administrator can move the exposure level of non-privileged accounts on servers to a higher level in the $HostsERs$ order (Figure 2(c)).

[1]To keep the example simple, we assume that all users' files are either under *public* or *private* directories. The example can be easily refined to include files that are not under one of these directories (e.g., by using the resource */home/*).

Note that the modified $HostsERs$ (Figure 2(c)) 'breaks' the consistency of the ERs partial order. That is, it is no longer true that the cost of any privileged account vulnerability is higher than any non-privileged account vulnerability. However, the $HostsERs$ helps to maintain the consistency of both $Hosts$ and ERs in the majority of the cases. The administrator will override the consistency only when security priorities dictate the inconsistency. In a realistic case, where $Hosts$ and ERs contain many more sets of hosts and resources, it is difficult to build an EC function which is consistent with both. In such cases, the importance of the $HostsERs$ order and the methodology to consistently build it from a product of two partial orders increases.

Since every level in both $Hosts$ and ERs represents the same exposure cost, it makes sense to assign to each level in $HostsERs$ the same cost too. A consistent way to do so is to assign higher exposure costs to higher levels in $HostsERs$. For example, we assign exposure costs with different orders of magnitude to each level in the $HostsERs$ from Figure 2(c). As we demonstrate in Section 6, such a simple scheme is very effective in pinpointing vulnerabilities with high severity.

5.2.1 Ranking Difficulties.

To quantify attack difficulty, we use a sum of the ETE and ETA values. Other (e.g., non-linear [ODK99]) measures are left for future work.

To account for visibility (i.e., firewall), the vulnerabilities could again be partitioned (within each difficulty ranking) by their visibility ranking. If there are many firewalls, the system administrator could assign a "trustworthiness"rating to the users of each firewalled subnet, where this rating is used as a multiplier for the expected time to attack (ETA) measure defined for the wide area Internet.

6 Pilot Study

We investigate two questions regarding the IPI. First, to what extent the IPI is needed. Second, to what extent the threat model can express the site security priorities, and whether such a model is useful as a quantitative security measure. To answer the first question, we test whether a 'repeated audit' approach is capable of identifying security holes in a site with relatively high security awareness. To answer the second question, we perform a field test of our threat model.

6.1 Experimental Methodology

We simulated the IPI, over a period of six months, on a site that contains more than 60 dedicated servers, and almost 1500 hosts running a variety of operating systems. The site administrators monitor mailing lists and web sites for security issues, and apply secu-

rity patches as quickly as possible. Most importantly, the site security practices emphasize *strong configuration management* (SCM)—all software installations are identical and users cannot install their own software on their workstations. Under SCM settings, the administration can focus security efforts on a (relatively) small set of software packages, so it is relatively easy to identify and patch vulnerabilities. Finding long-lived vulnerabilities in such a situation points to the necessity of the IPI.

We used Nessus [De] to audit the system over a period of six months. During the first five months we did not share the audit results with the administrators, but we modeled the vulnerabilities using our presence semantics and semi-automatically applied our threat analysis:

1. We analyzed each vulnerability Nessus reported and expressed its presence using our predicates (Section 4.2). Then, we analyzed the Nessus script that attempts to identify the vulnerability. If the script did not verify all the predicates necessary for the vulnerability identification, we classified this Nessus alarm as a false positive and removed the alarm from the audit results. For each vulnerability, we also specified the repair options we derived from the predicates (Section 4.4).

2. We assigned exposed resources (Section 4.3.1) to each vulnerability. According to our difficulty definition, we assigned an ETA value (intrinsic difficulty) to each vulnerability: 30 days to exploits that require large computational effort (e.g., dictionary attack), 7 days to vulnerabilities that require considerable programming effort (e.g., buffer overflows without a known exploit), and 1 day to exploits that are easily found on the web (e.g., buffer overflow with a known exploit) [2].

3. We removed from the audit results all vulnerabilities that cannot be exploited due to site firewall protection.

4. Together with the system administrators we built the $Hosts$, the ERs, and the final $HostsERs$ partial orders; an order that is similar in nature to the one illustrated in Figure 2(c). Then, we calculated the total threat (TS_v) of each true positive vulnerability.

The prioritized audit report for the fifth month was presented to the system administrators. To measure the impact of the audit results on the site security, we performed one additional audit a month later.

6.2 Results

Per month, Nessus reported about 190 different vulnerabilities on hundreds of different hosts. After we analyzed the reported vulnerabilities, we conclude that 75% of the alarms are false positives caused by one of the following reasons:

[2] We do not claim that these are the difficulties the IPI should use—future work should consider such issues. But, these values illustrate to the system administrator the differences in the inherent difficulty of the vulnerabilities found.

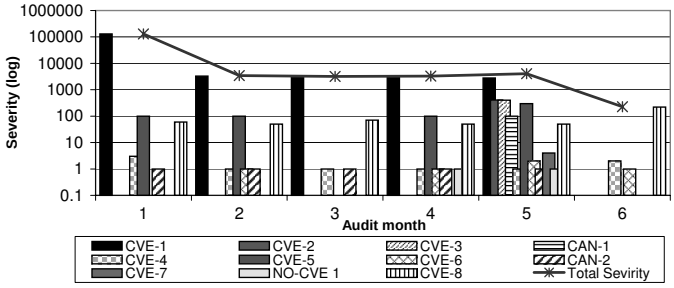


Figure 3: Severity of persistent vulnerabilities. Severity calculated by TS_v.

1. Inaccurate test. A presence predicate is not verified properly. For example, Nessus identifies the *TELNET* vulnerability (Table 1) by checking for the existence of the Telnet banner. However, the existence of a banner does not mean that Telnet login is possible. Indeed, the site administrators disabled Telnet, but to notify users about this, left a modified banner in place.
2. Incomplete test. One of the predicates in the condition set is not verified at all. For example, Nessus cannot verify predicates that require host configuration information as P_4 in the *SSH* vulnerability (Table 5).
3. Firewall protection. Nessus does not filter out vulnerabilities that cannot be exploited due to firewall protection[3].

We believe that the IPI eliminates these types of false positives. First, inaccurate tests are less likely to happen because the IPI audit stems from a precise semantics (Table 2). Second, configuration predicates, that are verified using a host-based audit daemon, eliminate false positives that require host configuration information. Lastly, we indent to incorporate firewall information into the threat analyzer.

Figure 3 presents the severe vulnerabilities found during our six month audit. To maintain the site confidentiality, we do not specify the vulnerability names or the $HostsERs$ partial order. Three observations should be noted:

1. Vulnerabilities of which the system administrator was unaware did pop up, even though the system used SCM. CVE-2, CVE-3, CVE-7, and CAN-1 appeared in the fifth month. During the experiment period CVE-6 appeared, was fixed (without the use of the audit results) and reappeared.
2. High severity vulnerabilities could pass undetected even in a site that used SCM. Between the first and second months, the administrator removed a severe vulnerability (CVE-1) from almost all hosts. This system repair was done without any audit information and reduced the severity level of the vulnerabilities in the system by two orders of magnitude. Nevertheless, two servers and three user workstations were inadvertently left exposed to CVE-1. During this time, despite CVE-1 existence on

[3]Nessus can identify these by performing two audits: one inside the firewall and one from outside the firewall. Clearly, this approach doubles the audit effort.

Name	Service	Threat Definition			Threat Analysis			Repair			Comments
		Exposed Resources	ETA (days)	ETE (days)	Servers	Non-Servers	Severity	Update	Reconfig	Block	
CVE-2	ftp	<*CIA,SA*>	1	0	7	11	410	√	-	√	buffer overflow
CVE-5	ftp	<*CIA,PA*>	1	0	-	3	300	√	-	√	buffer overflow
CVE-4	oracle tnslsnr	<*CIA,SA*>	1	0	-	1	1	√	√	-	easy password
NO-CVE 1	HTTP	<*user_name,C*>	1	0	-	1	1	-	-	√	misconfiguration
CVE-1	ssh	<*CIA,PA*>	7	0	2	8	2800	√	√	√	buffer overflow
CVE-3	ftp	<*CIA,NPA*>	7	0	3	-	400	√	-	√	buffer overflow
CAN-1	finger	<*CIA,PA*>	30	0	-	1	100	√	-	√	easy password
CVE-8	X11	<*CIA,NPA*>	30	0	-	5	50	-	√	√	easy password
CVE-7	HTTP	<*HostIp.2616A*>	30	0	-	4	4	√	-	√	buffer overflow
CVE-6	ldap	<*Host,A*>	30	0	-	2	2	√	-	√	buffer overflow
CAN-2	lpd	<*Host,A*>	30	0	-	1	1	-	√	√	easy password

Table 4: The threat analyzer report after the audit of the fifth month. Vulnerabilities are prioritizes first by their difficulty and then by their severity.

only a few machines, it imposed a risk that was an order of magnitude higher than all the other vulnerabilities combined. CVE-1 and other undetected vulnerabilities (e.g., CAN-2, CVE-5) were fixed only after the administrator got the audit results.

3. Our threat analysis accurately captured the security notion of the system administrator. After the administrator fixed the majority of the vulnerabilities in the audit report (Table 4), the total severity level dropped one order of magnitude (between the fifth and the sixth months in Figure 3), a decrease that was confirmed by the system administrator. The administrator reviewed the raw Nessus report and assured us that the vulnerabilities in Table 4 were the most severe ones.

   Other state of the art audit tools do not capture this major change in the threat level. For example, the Nessus report after the fifth month included 'high severity' vulnerabilities on 45 hosts and the sixth month report included 44. Due to vulnerable hosts that popped up before the sixth month audit (CVE-4 and CVE-8), the number of vulnerable hosts remained almost the same. However, the threat level was reduced considerably because the administrator considered the vulnerabilities that were fixed (i.e., CVE-1) much more severe than the vulnerabilities that popped up.

Table 4 presents the prioritized audit report (of the fifth month) our threat analyzer produces from the audit results. A few observations should be noted.

1. The severity of privileged-account vulnerabilities is not always higher than that of non-privileged-account ones. The severity of CVE-3, a non-privileged-account vulnerability found on three sensitive servers, is higher than CVE-5, a privileged-account vulnerability found only on user workstations.
2. Not all vulnerabilities with the same exposed resources (e.g., privileged account) have the same severity. Although CVE-1, CVE-5, and CAN-1 have the same exposed resources, their severity levels are considerably different.

3. Our audit report includes precise repair options that are derived from the vulnerability presence definition.

The above two sets of observations reveal the advantages of the proposed IPI. First, the IPI is based on the 'frequent audit' approach which is necessary to identify severe vulnerabilities even in a site that uses strong configuration management. Second, because our threat definition is based on exposed resources and because our threat analysis is sensitive to the number and type of vulnerable hosts, we can accurately capture the administrator security preferences. The analysis not only enables us to differentiate between vulnerabilities that traditionally had the same severity level (e.g., CVE-1 and CVE-5), but it also demonstrates that sometimes the severity of 'highly severe' vulnerabilities is actually less severe than that of 'less severe' ones.

7 Conclusion

Much remains to be done: automating the audit and repair processes, extending the threat model to fully support difficulty and visibility site-dependent parameters, and developing the vulnerability semantics to support all these activities. Building a robust IPI is a significant challenge, and we hope that the vision we have outlined here will inspire both academics and industry professionals to continue this work.

References

[BAMF01] Browne, H. K., Arbaugh, W. A., McHugh, J., and Fithen, W. L.: A trend analysis of exploitations. In: *IEEE Symposium on Security and Privacy*. Oakland, CA. May 2001.

[CCZ02] Cliff Changchun Zou, D. T., Weibo Gong: Code red worm propagation modeling and analysis. In: *ACM Conference on Computer and Communications Security*. Washington DC. November 2002.

[CE] CERT Coordination Center. Vulnerabilities, Incidents & Fixes. Available at `http://www.cert.org/`.

[CM02] Cuppens, F. and Miege, A.: Alert correlation in a cooperative intrusion detection framework. In: *IEEE Symposium on Security and Privacy*. Oakland, CA. May 2002.

[CO00] Cuppens, F. and Ortalo, R.: LAMBDA: A language to model a database for detection of attacks. In: *International Symposium on Recent Advances in Intrusion Detection (RAID)*. Toulouse, France. October 2000.

[De] Deraison, R. Nessus Security Scanner. Available at `http://www.nessus.org/`.

[FPA98] Farmer, D., Powell, B., and Archibald, M.: TITAN. In: *Large Installation System Administrator's Conference (LISA)*. Boston, MA. December 1998.

[FS90] Farmer, D. and Spafford, E. H.: The COPS security checker system. In: *USENIX Technical Conference*. Anaheim, CA. June 1990.

[In] Internet Security Systems. The Internet Security Scanner. Available at `http://www.iss.net/`.

[KS94] Kim, G. H. and Spafford, E. H.: Experiences with tripwire: Using integrity checkers for intrusion detection. In: *USENIX Applications Development Symposium*. Toronto, Canada. April 1994.

[MIa] MITRE Corporation. CVE: Common Vulnerabilities and Exposures. Available at `http://www.cve.mitre.org/`.

[MIb] MITRE Corporation. OVAL: Open Vulnerability Assessment Language. Available at `http://oval.mitre.org/`.

[Mu95] Muffett, A.: WAN – hacking with AutoHack – Auditing security behind the firewall. In: *USENIX Security Symposium*. Salt Lake City, UT. June 1995.

[ODK99] Ortalo, R., Deswarte, Y., and Kaâniche, M.: Experimenting with quantitative evaluation tools for monitoring operational security. *IEEE Transactions on Software Engineering*. 25(5). September/October 1999.

[Or99] Ortalo, R.: Implementation of a distributed security monitoring tool: ESOPE. Technical Report 99506. Laboratory for Analysis and Architecture of Systems (LAAS). December 1999.

[Ro99] Roesch, M.: Snort – lightweight intrusion detection for networks. In: *Large Installation System Administrator's Conference (LISA)*. Seattle, WA. November 1999.

[Se] SecurityFocus Inc. Vulnerabilities database. Available at `http://www.securityfocus.com/bid/`.

[SPW02] Staniford, S., Paxson, V., and Weaver, N.: How to own the internet in your spare time. In: *USENIX Security Symposium*. San Francisco, California. August 2002.

[SSH93] Safford, D. R., Schales, D. L., and Hess, D. K.: The TAMU security package: An ongoing response to Internet intruders in an academic environment. In: *USENIX Security Symposium*. Santa Clara, CA. October 1993.

[Th] The National Institute of Standards and Technology (NIST). The ICAT Metabase. Avaliable at `http://icat.nist.gov/`.

[To] Todd, B. Sara: The security auditor's research assistant. Available at `http://www-arc.com/sara/`.

[ZL96] Zerkle, D. and Levitt, K.: NetKuang–A multi-host configuration vulnerability checker. In: *USENIX Security Symposium*. San Jose, CA. July 1996.

Identification	Name	Perl in CGI directory (CAN-1999-0509)	LPRng buffer-overflow (CVE-2000-0917)	SSH buffer-overflow (CAN-2002-0575)	Windows FTP backdoor (CAN-1999-0200)	WebSitePro root directory (CAN-2000-0066)
	Operating system	ANY	<RedHat,Linux,7.0>	<OpenBSD,OpenBSD, 3.1>	<MS,Windows,*>	<MS,Windows,*>
	Vulnerable unit	Perl	<Patrick_Powell,LPRng, <3.6.24>		Windows FTP	<OReilly Software, WebSitePro, <2.4.9>
	Configuration			Kerberos Authentication On	There exist an account with the name 'guest'.	
	Protocol Port	RFC: 2616/HTTP 80+'any'	RFC: 1179/LPDP 515+'any'	[a]/SSH 22+'any'	RFC: 959/FTP 21	RFC: 2616/HTTP 80+'any'
Presence	Condition Set	$P_1 : serviceRunning()$ $P_2 : access(perl)$	$P_1 : serviceRunning()$ $P_2 : version < 3.6.24$ $P_3 : package = LPRng$	$P_1 : serviceRunning()$ P_2:*version* < 3.2 P_3:*package* =*OpenSSH* P_4:*content*(*config_file*, AFSTokenPassing)	P_1 :*Login(USER=*, PASSWARD=*)*	Set 1: $P_1 : serviceRunning()$ $P_2 : version < 2.4.9$ $P_3 : package =$ *WebSitePro* Set 2: P_1:*netResponse(* ('GET /HTTP1.0 ', 'WebSitePro*404*: ||*windows_dir_name*'[b]
	Verification Hints	$H_{1,2,UNIX}$: *netResponse*('/cgi-bin/perl?-v', 'version') $\rightarrow (P_1 \wedge P_2)$	$H_{2,3,UNIX}$: *hostResponse*('lp -V', 'LPRng-'|| <'3.6.24')[c] $\leftrightarrow (P_2 \wedge P_3)$	$H_{2,3,UNIX}$: *hostResponse*('sshd -V', 'OpenSSH_'|| <'3.2') $\leftrightarrow (P_2 \wedge P_3)$ $H_{4,UNIX}$: *config_file*= '/etc/ssh/sshd_config') $H_{2,3}$: *netResponse*(*serviceRunning()*, 'OpenSSH_'|| <'3.2') $\leftrightarrow (P_2 \wedge P_3)$		
Threat	Exposed Resources	<*CIA,SA*>	<*CIA,PA*>	<*CIA,NPA*>	<*CIA,ftpguest*>	<*C,directory_data*>
	Expected Time to Exposure (days)	0	0	0	0	0
	Expected Time to Attack (Days)	1	1	1	0	1

[a] No RFC number yet for SSH.
[b] Put here a regular expression for a windows directory name.
[c] To conserve space we use the shorthand < to denote 'all versions smaller than', alternatively in reality all versions can be specified.

Table 5: Examples of vulnerability definitions.

Structural Comparison of Executable Objects

Halvar Flake

halvar@blackhat.com

Abstract: A method to heuristically construct an isomorphism between the sets of functions in two similar but differing versions of the same executable file is presented. Such an isomorphism has multiple practical applications, specifically the ability to detect programmatic changes between the two executable versions. Moreover, information (function names) which is available for one of the two versions can also be made available for the other .

A framework implementing the described methods is presented, along with empirical data about its performance when used to analyze patches to recent security vulnerabilities. As a more practical example, a security update which fixes a critical vulnerability in an H.323 parsing component is analyzed, the relevant vulnerability extracted and the implications of the vulnerability and the fix discussed.

1 Introduction

While programs that compare different versions of the same source code file have been in widespread use for many years, very little focus has so far been placed on the importance of detecting and analyzing changes between two versions of the same executable.

Without an automated way of detecting source code changes in the object code resulting from compilation, the party prompted with the task of reverse engineering the changes from the object code is at a disadvantage: It takes relatively little work to change source code and recompile, while the analysis of the object code will have to be completely redone to detect the changes. Both virus authors of high-level-language virus families (such as SoBig) and closed-source software vendors try to exploit this asymmetry: The authors of SoBig intend to create large quantities of work for the antivirus researchers to have more time to use the infrastructure built by their worm, whereas closed source vendors hope their customers will have time for installing patches because possible attackers presumably need a lot of time to reverse-engineer the relevant changes from object-code-only security updates.

This paper presents a novel approach which corrects the abovementioned asymmetry: Given two variants of the same executable A called A' and A'', an one-to-one mapping between all the functions in A' to all functions in A'' is created. The mapping does not depend on the specific assembly-level instructions generated by the compiler but is more general in nature: It maps *control flow graphs* of functions, thus ignoring less-aggressive optimization such as instruction reordering and changes in register allocation.

This allows porting of information (such as function names from symbolic debug information or prior analysis) from one executable to another. Furthermore, due to the approach taken, functions that have changed their functionality significantly will not be mapped, allowing the easy detection of functional changes to the program.

Detecting programmatic changes between two versions of the same executable is relevant to security research as it allows for quick analysis of security updates (”patches”) to extract detailed information about the underlying security vulnerabilities. This allows for quick assessment of the risk posed by a particular problem and can be used to prevent vendors from fixing security issues ”silently”, e.g. without notifying their customers about the security problem.

2 Previous Work

Automatically analyzing and classifying changes to source code have been studied extensively in literature before, and listing all relevant papers seems to be out of scope for this paper. Most of this research focuses on treating the source code as a sequence of lines, and applying a sequence-comparison algorithm [Hir77][HS77].

The problem of matching functions in two executables to form pairs has been studied in [ZW00, ZW99], although focused on reuse of profiling information which allowed the assumption of symbols for both executables being available. Other work has been done with focus on efficient distribution of binary patches [Poc] [BM99]. Both approaches, while finding differences between two binaries, are incapable of dealing with aggressive link-time profiling-information-based optimizations and will generate a lot of superfluous information in case register allocation or instruction ordering has changed. A bytecode-centric approach to find sections of similar JAVA-code is studied in [BM98].

Recently another approach to binary comparison also dealing with graph isomorphisms was discussed in [Tod]: Starting from the entry points of an executable basic blocks are matched one-to-one based on instructions present in them. If no matching is possible, a change must have occured. Due to the reliance on comparing actual instructions, a significant number of locations is falsely identified as changed - the paper mentions that about 3-5 % of all instructions change between two versions of the same executable.

3 Graph-Centric analysis

Instead of focusing on the concrete assembly level instructions generated by a compiler, we focus on a *graph-centric* analysis, neglecting as much of the assembly as possible and instead analyzing only structural properties of the executable.

3.1 An executable as Graph of Graphs

We analyze the executable by regarding it as a *graph of graphs*. This means that our executable consists of a set of functions $F := \{f_1, ..., f_n\}$. They correspond to the disassembly of the functions as defined in the original C sourcecode. The *callgraph* of the program is the directed graph with $\{f_1, \ldots, f_n\}$ as nodes. The edges of this graph represent function calls: An edge from f_i to f_k implies that f_i contains a subfunction call to f_k.

Every function $f_i \in F$ itself can be represented as a *control flow graph* (or short *cfg*) consisting of individual basic blocks and their branch relations. Thus one can represent an executable as a *graph of graphs*, e.g. a directed graph (the *callgraph*) in which each node itself corresponds to a *cfg* of the corresponding function.

3.2 Control Flow Graphs

The concept discussed here is well-known in literature on compilers and code analysis [AVA99]. Every function in an executable can be treated as a directed graph of special shape. Every node of the graph consists of assembly instructions that imply the execution of the following instruction in memory if and only if the previous instruction in memory was executed. To clarify this: Let i_k, i_{k+1} be addresses of two assembly-level instructions which are adjancent in memory. These instructions belong to the same basic block if the execution of i_k at n steps of execution implies execution of i_{k+1} at $n + 1$ steps, and the execution of i_{k+1} at $n + 1$ implies execution of i_k at step n.

Control flow graphs have a few special properties:

1. Every *cfg* has a unique entry point, meaning a unique node that is not linked to by any other node.
2. Every *cfg* has one or more exit points, meaning nodes that do not link to any other node.

Figures 1 through 4 show a simple C function (figure 1), it's assembly-level counterpart (figure 2), a full *cfg* containing the assembly-level instructions (figure 3) and finally just the *cfg* of the function.

3.3 Retrieving the information

In order to retrieve these graphs from an executable, a good disassembly of the binary is needed. The industry standard for disassembly is [Dat], mainly due to its excellent cross-platform capabilities coupled with a programming interface that allows retrieval of

```
int foo( int a, int b ) {
  while( a-- ) {
    b++;
  }
  if( b > 100 )
    return 1;
  else
    return 0;
}
```

Figure 1: The C function

the needed information without knowledge of the underlying CPU or its assembly. This facilitates implementing the described algorithms only once but testing them on multiple architectures.

3.4 Indirect calls and disassembly problems

In many cases creating a complete callgraph (which represents all possible relations between the different functions) from a binary is not trivial. Specifically indirect subfunction calls through tables (very common for example in C++ code that uses virtual methods) are hard to resolve statically.

In the presented approach, such indirect calls whose targets cannot be resolved statically, are simply ignored and treated as a regular assembly-level instruction. In practice, this does not yield many problems: The question whether a certain call is made directly or not is not answered by the optimizer but by the code that is being compiled, and thus does not change between different builds of the same program without a source code change.

4 Structural matching

The general idea explored in this paper is matching the functions in two executables by utilizing both information derived from the *callgraph* and the respective *cfg*'s instead of relying on instructions or instruction patterns. In this section two versions of the same executable will be considered: A and B as well as their callgraphs $\mathfrak{A} := \{\{a_1, \ldots, a_n\}, \{a_1^e, \ldots, a_m^e\}\}$ and $\mathfrak{B} := \{\{b_1, \ldots, b_l\}, \{b_1^e, \ldots, b_k^e\}\}$ which consist of their respective nodes (functions) and edges (c_i^e is a 2-tuple containing two nodes, and thus describes an edge in the graph).

Ideally, we want to create a bijective mapping $p : \{a_1, \ldots, a_n\} \rightarrow \{b_1, \ldots, b_m\}$. In the general case, this mapping does not exist due to different cardinalities of the two sets (if functions have been added or removed). Furthermore, properly embedding $\mathfrak{B}$ into $\mathfrak{A}$ seems to be an excessively expensive operation, specificially considering the possibility of

```
  push    ebp
  mov     ebp, esp
  push    esi
  push    edi
  jmp     short loc_40126A
loc_401267:
  inc     [ebp+arg_4]
loc_40126A:
  mov     edi, [ebp+arg_0]
  mov     esi, edi
  sub     esi, 1
  mov     [ebp+arg_0], esi
  cmp     edi, 0
  jnz     short loc_401267
  cmp     [ebp+arg_4], 64h
  jle     short loc_401287
  mov     eax, 1
  jmp     short loc_40128C
loc_401287:
  mov     eax, 0
loc_40128C:
  pop     edi
  pop     esi
  pop     ebp
  retn
```

Figure 2: The assembly code

$m < n$.

A different iterative approach to creating an approximation of p is taken: An initial mapping p_1 is created which maps elements of $A_1 \subset \{a_1, \ldots, a_n\}$ to elements of $B_1 \subset \{b_1, \ldots, b_n\}$. The mapping is then used to iteratively create a sequence of mappings $p_2, \ldots, p_h$ with $A_1 \subset A_2 \subset \cdots \subset A_h$ and $B_1 \subset B_2 \subset \cdots \subset B_h$.

4.1 A simple matching heuristic

Comparing undirected graphs is well-known to be excessively expensive, and even the restricted directed graphs can be quite expensive to compare. A relatively simple (and very imprecise) heuristic for telling whether two graphs are isomorphic is to compare the number of nodes and edges. If they do not match, it can be said with certainty that no isomorphism exists. The initial partial mapping p_1 is constructed by associating every $b_i \in \{b_1, \ldots, b_m\}$ with a 3-tuple $(\alpha_i, \beta_i, \gamma_i)$ where α_i is the number of basic control blocks in b_i, β_i is the number of edges in b_i and γ_i is the number of edges in the calltree

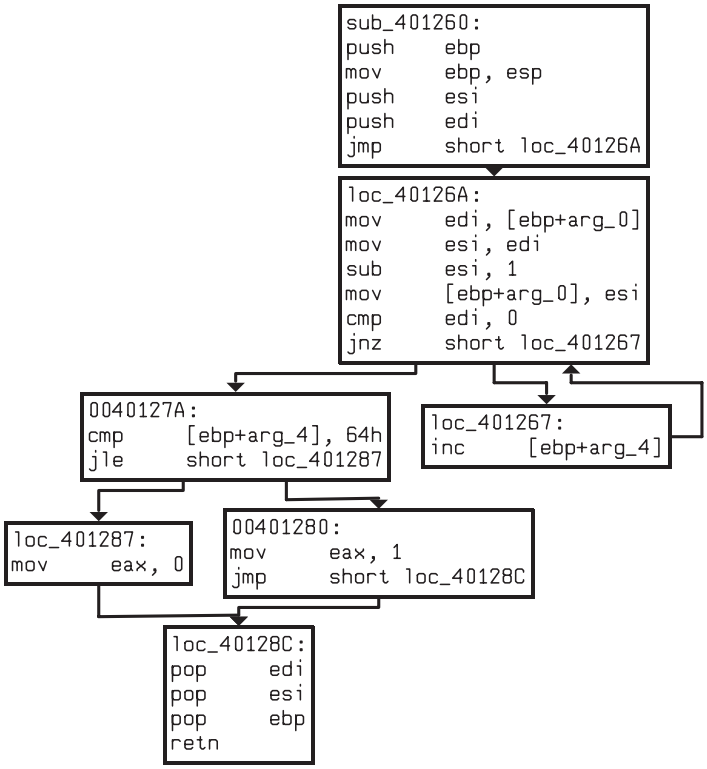


Figure 3: *cfg* with assembly

originating at b_i. We denote the mapping that maps a function in a callgraph $\mathfrak{C}$ to it's 3-tuple with $s : \mathfrak{C} \rightarrow \mathbb{N}^3$ and define the inverse function $s^{-1} : \mathbb{N}^3 \rightarrow \mathfrak{P}(\{c_1, \ldots, c_o\})$ that retrieves the set of functions which map to a certain tuple.

The mapping p_1 is constructed by examining all 3-tuples generated from A and B as follows: The functions $a_i \in A$ and $b_j \in B$ are mapped to each other if and only if they map to the same tuple and no other element exists in $\{a_1, \ldots, a_n\}$ or $\{b_1, \ldots, b_l\}$ which maps to the same tuple. More formally:

$$p_1(a_i) = b_j \Leftrightarrow |s^{-1}(s(a_i))| = 1 = |s^{-1}(s(b_j))| \wedge s(a_i) = s(b_j) \tag{1}$$

If the cardinalities of the sets $s^{-1}(s(a_i))$ and $s^{-1}(s(b_j))$ are both equal to one and both a_i and b_j map to the same tuple, p_1 maps a_i to b_j.

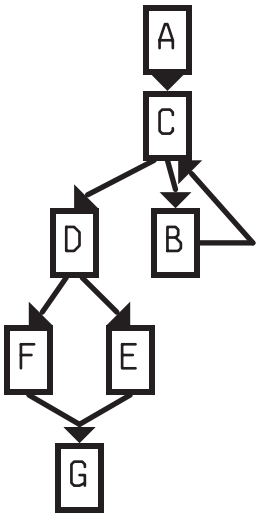


Figure 4: *cfg* without assembly

4.2 Improving p_1

The above heuristics yield only relatively small subsets of $A := \{a_1, \ldots, a_n\}$ and $B := \{b_1, \ldots, b_l\}$ that can be successfully matched initially. In general, smaller functions are a lot less likely to be successfully matched. This is mainly due to the special form that *cfg*'s take: Most basic blocks have exactly two "children" - this means that the odds that two randomly chosen *cfg*'s with the same node count have the same number of edges decrease as *cfg*'s grow. Smaller functions tend to have fewer subfunction calls, furthermore increasing the likelyhood that $|s^{-1}(s(a_i))| \neq 1$ occurs. It is intuitively clear that smaller sets A and B would reduce the odds of such collisions.

The improved mappings p_i are constructed by taking advantage of the information gained from p_{i-1} and using them to create small subsets $A'_j \subset A$ and $B'_j \subset B$ which are used for improving the mapping as explained above. The algorithm for constructing p_i from p_{i-1} works as follows:

1. Take the i^{th} element a_i from A_{i-1} and retrieve $p_{i-1}(a_i)$
2. Let A'_i be the set of all functions a_k that have edges originating from a_i leading to a_k in $\mathfrak{A}$ and B'_i the set of all functions b_o that have edges originating from $p_{i-1}(a_i)$ leading to b_o in $\mathfrak{B}$
3. Construct $p'_i : A'_i \rightarrow B'_i$ in the same way as p_1 was constructed
4. $p_i(a_j) := p_{i-1}(a_j)$ if $a_j \in A_{i-1}$. If $a_j \notin A_{i-1}$ and the construction of p'_i yielded a match, $p_i(a_j) := p'_i(a_j)$. If the construction of p'_i did not yield a match and $a_j \notin A_{i-1}$ then $p_i(a_j)$ is undefined.

5. A_i and B_i are the domain and image of p_i

Once p_k has been constructed where $|A_k| = k$, the iteration is finished and cannot yield improved results.

4.3 Graph restructuring

Compilers (and optimizing linkers) tend to change *cfg*'s in ways that do not truly change the logical structure of the function. Oftentimes, a single control block in a *cfg* is split up into several smaller ones that are linked with an unconditional branch instruction [1]. Since these operations will change the node and link count a way to easily and quickly undo the changes is needed. A simple graph-restructuring algorithm is applied before generating the 3-tuples which removes superfluous nodes generated by these optimizations:

```
for x ∈ {c_1, ..., c_o}:
    if (number of edges to x) = 1:
        y_x^e := edge to x
        y := source node of y_x^e
        {x_i^e, ..., x_j^e} := set of edges originating in x
        remove edge from y to x from graph
        remove x from graph
        for x^e ∈ {x_i^e, ..., x_j^e} :
            add edge from y to target of x^e to graph
            remove x^e from graph
```

5 Practical results

An implementation of the described methods has been created as an extension to the commercial debugger IDA Pro.

5.1 Name porting between databases

Two libraries that come with every standard install of Windows were examined in different versions: wininet.dll and msgsvc.dll. The versions of wininet.dll were those of Windows XP SP1/SP2 respectively, the versions of msgsvc.dll those pre/post MS03-48. Both have been heavily fragmented by aggressive link-time optimizations and pose significant problems to signature-based function matching.

[1] This seems to be a specialty of Microsoft's optimizing linker

File	File Size	# functions	# mapped	Runtime in seconds
msgsvc.dll pre MS03-48	35.600	134	100	< 5
msgsvc.dll post MS03-48	34.064	129	100	< 5
wininet.dll SP1	599.040	2310	1522	183
wininet.dll SP2	588.288	2321	1522	183

5.2 Analysis of security patches

5.2.1 H.323 Parser

After the NISCC published information about vulnerabilities in multiple H.323 parsers, the question arose where the relevant mistake in Microsofts ISA Server product was. Microsoft refuses to publish detailed information about the vulnerability they fix. According to the NISCC report, the problem was located in ASN.1 decoding.

Both the pre- and post-patch versions of H323ASN1.DLL were analyzed, with the result that 11 functions in the unpatched version could not be mapped to the patched version, and 8 functions in the patched version could not be mapped to any function in the unpatched version.

Address	# Nodes	# Links	# Children
40f627	26	46	21
40f837	19	32	12
41d012	10	16	7
41ed06	8	12	2
428d36	8	12	2
42b9e2	8	12	2
42bc90	8	12	2
42bd85	8	12	2

Patched Version

Address	# Nodes	# Links	# Children
40f4bb	22	40	21
40f697	14	24	12
41cd73	9	15	8
41ce7d	8	13	7
425595	4	5	4
425728	4	5	4
428b72	7	10	2
42b98e	7	10	2
42bbd2	7	10	2
42bcbf	7	10	2

Unpatched Version

A manual inspection of these functions yielded the result that the first three functions in both tables are in fact the same with the only change being an added range check. In all three cases, the old version retrieves an unsigned 32-bit integer from an ASN.1 PER encoded stream by means of a function called ASN1PERDecU32Val().

This 32-bit integer is passed on to ASN1PERDecZeroTableCharStringNoAlloc() as second argument. The patched variant on the other hand introduces a range check to make sure this second argument is smaller than 129.

A closer inspection of ASN1PERDecZeroTableCharStringNoAlloc() reveals that the function calculates the size of memory allocation based on the formerly untrusted value – an attacker was able to set this value in a manner that the calculation would exceed

MAXUINT and thus be of very small size. The subsequent copy-operation would then corrupt the heap, allowing an attacker to gain control in the next round of heap consolidation. Instead of fixing the issue at the core (e.g. in the MSASN1.DLL library), a range check was added into the calling application (H323ASN1.DLL).

The update thus disclosed to an examining party that every call to ASN1PERDecZeroTableCharStringNoAlloc() needs to have argument checking done *before* the call is issued. A short system-wide scan was conducted to see if other applications besides ISA Server use the relevant function in dangerous way. Two other instances were found: The Windows-internal H.323 Multimedia Provider Library (which allows arbitrary applications to easily process H.323 data) and Microsoft's Video Conferencing Software Netmeeting. Neither does proper range checking on the function in question.

The result was that the update to H323ASN1.DLL fixed one bug but alerted anyone with the capability to analyze patches to two further remotely exploitable vulnerabilities which were not fixed at the time.

Microsoft was contacted and the issues were fixed a few months later, in MS04-11.

The total analysis took less than 3 hours time.

5.2.2 SSL/PCT Parser

In April, Microsoft issued an update to SCHANNEL.DLL, the library responsible for handling SSL communication. According to their security bulletin, they removed a security problem that allowed attackers to take full control of any computer running an SSL-based server. No technical details were provided, except that the problem itself lay in a part of the library responsible for parsing PCT packets [2].

More than 20 changed functions were detected in total, but only one with a name that implied it was involved with PCT parsing. An examination of the function Pct1SrvHandleUniHello() revealed that the old version had taken a string, NOT'ed every character and appended it to the original string. The new version was changed in such a manner that it ensured the combined string would not exceed 32 characters.

Detecting and understanding the vulnerability (a vanilla stack-smash with EIP overwrite) took less than 30 minutes. Subsequently, code was constructed to reach the appropriate location in the binary. Within 5 hours, EIP could be overwritten with an arbitrary value, and within 10 hours of the start of the analysis, a program that reliably exploited the vulnerability was created.

6 Comparison to other methods

In comparison to other methods for reverse engineering changes to a binary, the presented method has a few distinctive advantages as well as a few significant disadvantages.

[2] PCT is a legacy-protocol that was obsoleted by TLS and is supported for legacy browsers

6.1 Few False positives

The presented method performs significantly better than [Tod] in terms of false positives: The instruction-based approach suffers from 3-5 % of all instructions being marked as changed. Unless heavy, structure-changing optimizations are performed (such as the inlining of complex functions), the presented method is free of false positives: A functions whose flowgraph has changed has undergone a change. While testing the method on a multitude of different programs, no function pair was found that had not changed but was marked as changed. This drastically reduces the human work involved when trying to detect the significant changes in a security update.

6.2 CPU-independence

The presented method is almost completely independent of the underlying CPU architecture as long as a good disassembly with cross-references is available. The only CPU-dependent function that has to be available in addition to flow information is the capability to distinguish between a subfunction-call and a non-subfunction call. Successful tests were ran examining differences between MIPS-based ROM images and SPARC-based Solaris ELF executables in addition to the x86-based PE files discussed above.

Instruction-based approaches contain large amounts of CPU-dependent code which makes creating a multi-platform analysis tool significantly more complex.

6.3 Possible False Negatives

The downside obviously is the presence of possible false negatives if the program logic itself is not changed but constants or buffer sizes are. It is easy to imagine that a software vendor will fix a security vulnerability not by adding a range check but by enlarging the size of a buffer, which in the current method will go unnoticed. This is where [Tod] is clearly superior, as any change in buffer sizes or constants will be detected. This is bought by the cost of having to examine a significantly larger number of detected changes. Empirical evidence suggests that security updates which change constants but not program flow are very rare. Nonetheless, this is a region in which improvements on the proposed method are desirable.

7 Summary

It has been shown that nondisclosure of vulnerability information is not a promising deterrent to would-be-attackers and that security updates can be reverse engineered in relatively little time (given the right tools). It has furthermore been shown that special care has to be

taken when releasing security updates, as the information in the patch has to be assumed to be public. An incomplete bugfix can do more harm than good by disclosing the existence of other (unfixed) bugs along with the fix.

The presented work furthermore implies that the common practice of leaving one or two weeks between the publication of a security update and installing the patch is highly dangerous.

Leaving the politics of vulnerability disclosure out, it has been shown that analysis of binaries based only on structural properties of the code is a promising field of research, as it allows analysis of executable code without the need to abstract to an intermediate language or CPU-specific analysis engines.

8 Future work

Many things have to be improved and worked on to make the proposed method truly useful. Fast heuristics that can tell that two graphs are not isomorphic which are better than the current version are needed and would greatly improve the matching statistics. Furthermore, intraprocedural difference analysis would be useful: Given two *cfg*'s a_1, b_1 which are different, but expected to belong to the same function due to their position in the callgraph or due to other heuristics, an algorithm that constructs a partial isomorphism between subgraphs of a_1 and b_1 would allow quicker analysis of changes. Given two versions of the same large function, finding a relatively small change still has to be done manually.

A separation of the function-matching for name porting and function-matching for binary difference analysis will be needed sooner or later: Function-matching benefits from relaxed heuristics, while binary difference analysis does not want to miss changes.

More in-depth study of the effects of heavily optimizing/inlining compilers would be desirable, as well as more studies on the applicability of the presented methods to other CPU-architectures.

Detecting changes in buffer sizes and changes in (certain) constants would be desirable goals in the immediate future.

9 Acknowledgements

The author would like to thank the anonymous reviewers for many constructive comments. Valuable comments were also provided by Josh Anderson, Brandon Baker, John Pincus, Felix Lindner und Jan Muenther.

References

[AVA99] Jeffrey D. Ullmann Alfred V. Aho, Ravi Sethi. *Compilerbau*. Oldenburg Verlag, München Wien, 2 edition, 1999.

[BM98] Brenda S. Baker and Udi Manber. Deducing Similarities in Java Sources from Bytecodes. pages 179–190, 1998.

[BM99] Brenda S.Baker, Udi Manber and Robert Muth. Compressing Differences of Executable Code. In *ACMSIGPLAN Workshop on Compiler Support for System Software (WCSS)*, pages 1–10, 1999.

[Dat] DataRescue. IDA Pro Disassembler *http://www.datarescue.com/idabase*.

[Hir77] Daniel S. Hirschberg. Algoritms for the Longest Common Subsequence Problem. *J. ACM*, 24(4):664–675, 1977.

[HS77] James W. Hunt and Thomas G. Szymanski. A fast algorithm for computing longest common susequences. *Commun. ACM*, 20(5):350–353, 1977.

[Poc] Pocket Soft Inc. RTPatch – Software Update Tool *http://www.pocketsoft.com/whitepapers/whitepaper.html*.

[Tod] Todd Sabin. Comparing binaries with graph isomorphisms *http://razor.bindview.com/publish/papers/comparing-binaries.html*.

[ZW99] Scott McFarling Zheng Wang, Ken Pierce. BMAT - A Binary Matching Tool. *2nd ACM Workshop on Feedback-Directed Optimization*, November 1999.

[ZW00] Scott McFarling Zheng Wang, Ken Pierce. BMAT - A Binary Matching Tool for Stale Profile Propagation. *The Journal of Instruction-Level Parallelism (JILP)*, 2, May 2000.

Anti-Patterns in JDK Security and Refactorings

Marc Schönefeld

schonef@acm.org

Abstract: This paper underlines the importance of security awareness whilst programming Java applications. Several problems in current JDK implementations are demonstrated that allow to undermine the security of Java applications. Coding errors and quality problems in current Java distributions create possibilities to create covert channels, cause resource blocking and denial-of-service attacks. To make things worse Java components are often deployed according to the AllPermissions antipattern with non-restrictive security settings, which allows bugs on the system layer to be exploited by attackers. Coping with this antipattern from the user side is connected with the definition of adequate permission sets. A tool that automates this time consuming task is presented as a refactoring for the AllPermission antipattern.

1 Introduction

Java is one of the few programming languages that was designed from the beginning with security goals in mind[Go99b]. In the first versions (1.0 and 1.1) a simple sandbox model was available that allowed containment for remote code. Current editions of Java extend this model to a fine-grained architecture that extends the locality based approach of the first sandbox model with more trust-establishing parameters such as the codebase of the current instruction, the signature of the code signer and with the Java Authentication and Authorization System (JAAS) framework also the identity of the current user. An application can be secured with a policy file to define what actions are allowed according to the actual trust level of the current user and executed code parts.
As with all security architecture specifications, hackers typically do not try to attack the system through the front door, they seek errors in the implementations and try to exploit covert channels and triggers to the layer below[Go99a] to bypass security mechanisms.
The following sections will show that although equipped with language based security features such as type safety, antipatterns such as covert channels can be found in the Java architecture that allow to exploit logical errors in the code of trusted libraries. In addition false assumptions in the packaging the of the classes in the trusted system libraries of the JDK offer a harmful range of ready-to-call functionality to the attacker.

2 Java Security

Traditional requirements towards programming languages are reliability, performance, flexibility, abstraction and broad applicability. Induced by the growing importance of distributed middleware models like CORBA[Ob01] and the Java 2 Enterprise Edition (J2EE[Sh]) the requirements towards security gained importance. This is due to the fact, that code could not be considered as trusted as it is downloaded on demand from unknown remote sites which trustworthiness is typically unknown.

The experiences with traditional programming languages like C and C++ were the basis to design Java [GJSB00]from scratch with security goals in mind and avoid the potential vulnerabilities of direct memory access, pointer arithmetic and arbitrary type casts. Such concepts are found with an emphasis on system security in the Java 2 Standard Edition (J2SE[Sua]) as platform independent framework for desktop applications as well as with an additional emphasis on access control in J2EE. Java is based on the principles of language based security which is enforced by the trusted kernel to provide code safety. These principles are control flow safety, memory safety, stack safety which support safety of the Java type system [Ko99]. Java code is stored in platform-independent bytecode format that is verified in accordance to the subdisciplines of type safety by the trusted kernel.

2.1 Secure Coding and Antipatterns

As the maintainer of the Java programming language Sun Microsystems has published a set of coding guidelines for secure Java programming [Suc]. The coding guidelines give hints when dealing with the following issues:

R1 Refrain from using non-final public static variables

R2 Reduce scope

R3 Refrain from using public variables

R4 Protect packages

R5 Make objects immutable if possible

R6 Never return a reference to an internal array that contains sensitive data

R7 Never store user-supplied arrays directly

R8 Serialization

R9 Native methods

R10 Clear sensitive information

Violation of these guidelines typically results in security antipatterns. The negative effects of ignoring the guidelines R1, R2, R3, R4 , R9 and combinations of them is discussed the following sections.

According to Brown et al. [BMIM98]

> the essence of an AntiPattern is two solutions, instead of a problem and a solution for ordinary design patterns. The first solution is problematic. It is a commonly occurring solution that generates overwhelmingly negative consequences. The second solution is called the refactored solution. The refactored solution is a commonly occurring method in which the AntiPattern can be resolved and reengineered into a more beneficial form.

In the following the negative consequences of the integration of the `org.apache.*` classes into JDK 1.4.x are demonstrated.

2.2 Non-final public static methods and fields in the JDK-Packages

The packages in the JDK are subdivided in the classes that form the core Java language (`java.lang.*`), supporting classes (`java.*`), implementation dependent `sun.*` prefixed classes and other packages. Beginning with version 1.4.x of the JDK the ”other” packages contain several classes from the Apache Xalan and crimson libraries to provide functionality to process XML and XSLT data [Sub]. A typical Java class consists of instance related methods and fields and of class related methods and fields, which are identified via the `static` keyword. It is common that data structures that are designed to be available globally are implemented with `public static final` modifiers, which makes them available throughout the application (`public`), binds them to the class (`static`) and makes them writable only once (`final`).

According to secure coding guideline `R1` declaration of non-final public static methods and fields is harmful, this is especially true when executing code from untrusted sources.

The Java Plugin [Su03a] from Sun Microsystems is designed to execute Java code from untrusted sources in an Internet browser like the Microsoft Internet Explorer or Mozilla. It starts a single JVM and creates an instance of the applet classloader class for each loaded applet. The Java virtual machine uses private `ClassLoader`objects to create separate address spaces between processes to provide an environment of code confinement. Any user class may be loaded multiple times if it is loaded by private class loaders. In contrast classes residing in the `rt.jar` (JDK boot system classes) are loaded once and their static variables are strict singletons[GHJ95]. The classes of the Apache XML and XSLT utility packages (`org.apache.*`) expose several public static fields and property values that can be set from untrusted user code. These static variables become risk and threat to integrity when they can be modified by untrusted code in such a way that they modify system behavior. As Sun did not rename these packages with a `sun.*` prefix the XML utility

classes technically became part of the public Java interface available to all code types including applets, although the `org.apache` namespace is not mentioned in the official JDK documentation. The applet sandbox[Go99b] permission set allows access to packages and classes of the public interface, which enables class loaders to define and access these classes, in contrast to the unaccessible classes in private `sun.*` packages as such as `sun.security.util.PropertyExpander` class. These classes are restricted from definition and direct access by the default security manager policy settings located in `jre/lib/security/java.policy`.

2.3 Covert Channel and Triggers Antipatterns

According to Bishop[Bi00] a Covert channel is

> a path of communication that was not designed to be used for communication

An attacker can find out potential covert channels in the JDK classes by analyzing the communication and calling paths between the classes. Those classes are packed in a Java archive (jar file), which is an extension of the Zip-Format by a manifest which holds Java specific meta information.

An attacker may use bytecode engineering techniques [Sc02] to perform the following analysis while scanning the jar file:

1. Acquisition of a list of the public classes in the jar file as these are accessible via the reflection API to outer Java scripting such as Beanshell, Javascript or stored procedures in several JDBC drivers.
2. Scan these classes for public, static non-final fields and methods. The acquired fields can be used to establish covert channels for processes running inside the same VM. The acquired methods can be used to trigger actions in the mentioned script environments.

3 Covert channels and triggers

Two misuse cases will be shown that exhibit the danger of exploiting these shared resources and functionality. The scenarios are:

1. **Covert channels** that allow unsigned applets to communicate to other signed and unsigned applets
2. **Covert triggers** that allow to execute arbitrary programs on the machine running the VM via remote JDBC calls

According to the security pattern framework by Yoder and Barcalow[YB97] a single access point limits the entrance to critical functions and resources of applications. This pattern is typically undermined by covert channels that bypass the ”single entrance” premise. The Java security manager concept that enforces the applet sandbox is an implementation of such a single access point. It technically intercepts the critical calls by referring to the policy in place prior to resource access. The strict default applet policy to protect the integrity of the users workplace when working with mobile contents loaded from untrusted sources.

3.1 Covert channels in the JDK

3.1.1 Applet covert channel

One of the restrictions enforced by the default applet security manager is the limitation of package access, limiting access to methods and fields of classes from a package that not prefixed by a `sun` top-level package name.

3.1.2 Detection techniques

In order to find the potential covert channel and triggers inside the JVM, the class files residing in the Java runtime libraries (`rt.jar`) were inspected. This was done with the help of the Bytecode Engineering Library (BCEL) which is part of the Apache Jakarta project[Daa]. Scanning the public classes in `rt.jar` of the Java version 1.4.2_03 resulted in 91 public static non-final fields and 4286 public static methods. These were exported to XML files via an XMLEncoder object to allow optional further automatic processing.

3.1.3 Proof-of-concept

In order to demonstrate the danger of covert JDK channels with a proof-of-concept implementation the public and static field `LANGUAGE` from the `XSLTProcessorVersion` class inside the `org.apache.xalan.processor` package was chosen, which resides in `rt.jar`. Then an applet was constructed that tested read and write access this field. This applet was distributed to three different remote web sites. The three identical applets were then loaded into a single web browser in different frames, which means sharing the same VM by all applets. During runtime applet access to the variable from the three applets was tested. As expected all three applets were allowed to modify this static variable and exchange data and serializable (Guideline `R8`) objects which were serialized via an `ObjectOutputStream` in `String` object and therefore violated the sandbox restrictions. The risk potential for this behavior ranges from denial-of-service of the XSLT functionality to sandbox escape by bypassing containment through covert channel communication. Additionally signed applets may leak information to unsigned applets which may circumvent the Bell LaPadula[BL] privacy considerations intended by the applet developer.

To improve quality of the Java runtime environment the issue was submitted to the Java bug database and was labeled with the internal number 236774. The bug was considered to be new and will be made visible to the public in the database after a refactored version is available, which will be case with the 1.4.2_05 version of the Sun JDK.

3.1.4 Memory reading applet

A covert channel to physical system memory was found by the author [Su03b] in the Java Media Framework (JMF), which is a toolset that allows to play multimedia elements such as music, movies and other stream data within pure Java applications. As the JMF is concerned with access to system hardware functionality such as the sound card and graphics equipment and uses several native codecs, performance-oriented and therefore platform dependent code has to be accessed by the JMF libraries. The JMF libraries are installed as endorsed library to the `lib/ext` directory of the JRE. Libraries in this directory are loaded by the boot class loader and are trusted fully by the applet security manager (which equals an `AllPermission` setting). A result of bytecode analysis of the classes of the `jmf.jar` in version 2.11.c a covert channel was identified that allowed indirect access to the system memory which is a violation of the strict containment premise enforced by the permission sets of the Java sandbox. The `NBA` class (NativeBlockAccessor) is responsible to provide a specified communication area between the components in pure Java and the native memory storage. The cause for the problem is the inappropriate guarding of an internal variable of the NBA class inside the JMF. This is a violation of the `R2` (reduce scope) secure coding guideline and causes the memory access antipattern. The field `data` holds the pointer to a native memory block. By subclassing the `NBA` class, the information stored in `data` was made available to arbitrary Java applets, which after using conversion routines were able to map Java byte values to the exposed system memory.

3.2 Covert trigger

A **covert trigger** in analogy to covert channels is defined here as

> A trigger of actions that was not designed to trigger actions

3.2.1 Applet floppy hardware attack antipattern

A containment problem can be raised with a covert trigger. Due to a implementation logic error in the Java virtual machine for the Windows platform, the security manager check is called after the physical check whether a floppy drive is available in the disk drive. When running the `createXmlDocument` of the `XmlDocument` class of the `org.apache.crimson.tree package` in an endless loop the machine (tested with the Java Plugin in IE6) stops working because the floppy drive is busy with antagonistic accesses to the disk. The hardware stress applet can lead to overheating in the floppy drive which might cause physical damage to the drive and the other components of the affected

PC,but even on systems without floppy drives the applet allows a simple denial-of-service attack by accessing other blocking device types via their file names. This antipattern was created by executing privileged code (triggering physical floppy access) without proper access control checks (`FilePermission`).

3.2.2 JDBC macros and covert triggers via remote command injection

JDBC is a Java centric standard to establish client-server database applications. Java clients use services of a database server via remote JDBC calls. Three 100% pure java databases were tested for vulnerabilities in regards of the remote command injection antipattern. The databases were:

HSQLDB aka Hypersonic SQL[HS], an open-source SQL database bundled with JBOSS 3.x

Pointbase DB 4.6 [Dab], a commercial SQL database system, is bundled with the J2EE 1.4 reference implementation

Cloudscape SQL [IB] from IBM is a standalone 100% pure java database and is also available bundled with the Websphere application server

By the time period of penetration testing local installations of all three databases, these products were not designed to run with a Java security manager. As a consequence they have been found to be vulnerable to remote command injection, information disclosure. A simple JDBC SQL statement was sufficient to start an arbitrary executable on the host running a SQL database, which open a covert trigger. With a SecurityManager in place this would only be feasible with an explicit ”execute”FilePermission.

As a demonstration the following SQL statement injects Java code in the address space of the server VM.

```
CREATE FUNCTION COMPDEBUG (IN P1 boolean) returns VARCHAR(100)
LANGUAGE JAVA NO SQL EXTERNAL NAME
"org.apache.xml.utils.synthetic.JavaUtils::setDebug"
PARAMETER STYLE SQL;
SELECT COMPDEBUG(true) FROM SYSUSERS;
CREATE FUNCTION SETPROP (IN P1 VARCHAR(100),
IN P2 VARCHAR(100)) returns VARCHAR(100)
LANGUAGE JAVA NO SQL EXTERNAL NAME
"java.lang.System::setProperty" PARAMETER STYLE SQL;
SELECT SETPROP('org.apache.xml.utils.synthetic.javac',
'cmd.exe') FROM SYSUSERS;
CREATE FUNCTION COMPILE (IN P1 VARCHAR(100),
IN P2 VARCHAR(100)) returns VARCHAR(100)
LANGUAGE JAVA NO SQL EXTERNAL NAME
"org.apache.xml.utils.synthetic.JavaUtils::JDKcompile"
PARAMETER STYLE SQL;
SELECT COMPILE('', '/c notepad.exe') FROM SYSUSERS;
```

The example (in Pointbase SQL syntax) starts a `notepad.exe` on the host executing the JDBC database as a proof of concept. As every other more harmful executable such as a remote shell could be started as well. The SQL statement is equal in functionality to the following Java code:

```
{
 org.apache.xml.utils.synthetic.JavaUtils.setDebug(true);
 System.setProperty("org.apache.xml.utils.synthetic.javac","cmd.exe");
 org.apache.xml.utils.synthetic.JavaUtils.JDKcompile("","/c notepad.exe");
}
```

The syntax between the databases differ in small details but the possibility of injection was shown for HSQLDB, Pointbase and Cloudscape SQL, when running on a Sun JDK 1.4.x virtual machine.

The code does the following:

- It sets a debug mode
- Then an internal variable of the Xerces classes is set that defines the default Java compiler to an arbitrary executable program
- Finally it calls the compile function, which invokes an executable file (`cmd.exe`).

Due to the individual mapping mechanism of the SQL data types to the Java data types which is different for each database product. The smallest set of available functionality was restricted by the mapping of HSQLDB. Via bytecode engineering candidate methods in rt.jar were retrieved that have a `public static void` signature with primitive (such as boolean) or `java.lang.String` input values. This set was scanned whether the member calls privileged code parts such as file operations and shell execution. Beside the presented examples other functionality can be called that can be misused for log manipulation or abnormal program termination such as demonstrated in the next SQL statement which calls a vulnerable JVM routine in the sun packages which causes an immediate JVM crash in the remote JDBC server. This vulnerability was communicated to Sun by the author in 2002 but is not fixed until today.

```
CREATE FUNCTION CRASH5(IN P1 VARCHAR(20)) RETURNS VARCHAR(20)
 LANGUAGE JAVA
 NO SQL
 EXTERNAL NAME "sun.misc.MessageUtils::toStderr"
 PARAMETER  STYLE SQL;
SELECT CRASH5(null) from SYSUSERS;
```

The SQL statement is equal in functionality to the following Java code.

```
{
  sun.misc.MessageUtils.toStderr(null);
}
```

The enhanced problem with Hypersonic SQL which was deployed with JBOSS application server 3.2.1 was the lack of a security manager. It opened a listing TCP port 1701 that

accepted anonymous JDBC calls, which allowed to inject arbitrary remote commands into the J2EE process as the JDBC DB was running in the same VM as the J2EE server process. By closing the open port and switching default configuration to the internal JVM communication mode the vulnerability has been fixed by the JBOSS developers after a vendor notification. The Pointbase DB has also been refactored with version 4.8 after a vendor notification by deploying an optional security manager. IBM admits that is a good idea to apply a security manager. As the secure coding guidelines `R1, R2, R4` are violated when allowing JDBC macros,

4 Structural Refactoring of the AllPermission antipattern

Refactorings are a means to turn anti-patterns in a good solution by applying well-known and scalable patterns. The problems identified above were problems of security unaware coding which in combination with unlimited access rights is very harmful. This is typically caused by an `AllPermissions` or equivalent settings, that is the default case when there is no instance of a SecurityManager is installed with a virtual machine. Running a system with a minimal set of privileges is therefore an admirable goal as it enhances the assurance level. Even when a security manager is installed determining the range of a minimal permission set is a time-consuming task. Therefore jChains has been developed. The functionality it provides was first used when a minimal set of rules was needed to refactor the JDBC macros antipattern.

4.1 jChains

jChains is a custom security manager framework records the permissions needed for the codebases (jars) of J2SE applications running under the access control enforcement of the Java security manager. This allows security unaware applications to be run under a security manager. The resulting policy file is recorded while running the program and is useful as a starting point when developing a security policy for a Java application. When run against libraries when source is not available it is useful for reverse engineering, revealing the permission needed to use the libraries. This is helpful when a developer does not trust the jar , and do not want to grant it the AllPermission free ride ticket. jChains is designed to acquire a policy set from a Java application by doing a training mode and later enforce this policy set in a production mode.

4.1.1 Refactoring JDBC macros with jChains

A typical use case for jChains was evaluating a useful set of rules for the Pointbase database when providing the vendor with a possible security-related refactoring suggestion.

```
grant codeBase "file:${pointbase.lib}${file.separator}-" {
permission java.net.SocketPermission "*:1024-", "connect,resolve,accept";
permission java.io.FilePermission "<<ALL␣FILES>>", "read,write,delete";
permission java.util.PropertyPermission "*", "read";
};
```

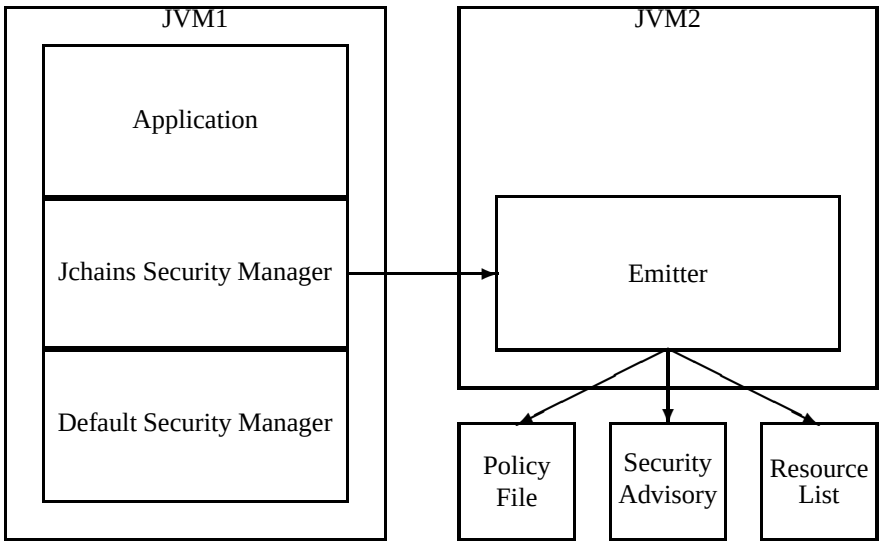


Figure 1: jChains-Architecture

jChains [Sc] is either configurable in a local mode or can communication to a remote CORBA server. CORBA was chosen in favor of RMI because of it's independence from a concrete implementation language. The distributed CORBA mode allows to decouple permission recording from permission evaluation. This is useful for remote permission training scenarios, e.g. when there is no direct user access to the system hosting the virtual machine. As benefit inherited from CORBA location transparency jChains may also operate locally.

5 Conclusion

It has been shown that the Java platform is not free of security related issues although providing a large scale of default security precautions. But these mechanisms cannot help in a case when the attacker attacks the system on a semantical level below the mechanism implementation, which typically is the case when antipatterns such as logical errors and packaging structures are exploited. These coding related antipatterns are often caused by circumventing the assumptions of the secure coding guidelines. To provide a refactoring for the AllPermissions antipattern jChains has found acceptance under developers (is a published freshmeat project and is hosted at the `java.net` site. In addition jChains

was useful for refactoring the permission set of the Pointbase SQL server product and is currently in evaluation by a major german banking group to adjust the appropriate policy set for the J2EE thin-clients of a banking client-server system.

References

[Bi00] Bishop, M.: *Computer Security*. Addison-Wesley. 2000.

[BL] Bell, D. und LaPadula, L.: Secure computer systems. Technical report. Air Force Elec. Syst. Div.

[BMIM98] Brown, W. J., Malveau, R. C., III, H. W. S. M., und Mowbray, T. J.: *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley & Sons. 1998.

[Daa] Dahm, M. BCEL manual. `http://jakarta.apache.org/bcel/manual.html`.

[Dab] Data Mirror Software. Pointbase product homepage. `http://www.pointbase.com`.

[GHJ95] Gamma, E., Helm, R., und Johnson, R.: *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley. 1995.

[GJSB00] Gosling, J., Joy, B., Steele, G., und Bracha, G.: *The Java Language Specification Second Edition*. Addison-Wesley. Boston, Mass. 2000. `http://citeseer.ist.psu.edu/gosling00java.html`.

[Go99a] Gollmann, D.: *Computer Security*. Wiley & Sons. 1999.

[Go99b] Gong, L.: *Inside Java 2 Platform Security*. Addison-Wesley. 1999.

[HS] HSQLDB Development Team. Hsqldb product homepage. `http://hsqldb.sourceforge.net`.

[IB] IBM Corporation. Cloudscape product homepage. `http://www-306.ibm.com/software/data/cloudscape`.

[Ko99] Kozen, D.: Language-based security. In: *Mathematical Foundations of Computer Science*. S. 284–298. 1999. `http://citeseer.nj.nec.com/kozen99languagebased.html`.

[Ob01] Object Management Group: *The Common Object Request Broker: Architecture and Specification*. Object Management Group. 2.5. September 2001.

[Sc] Schoenefeld, M. jChains homepage. `http://jchains.org`.

[Sc02] Schoenefeld, M.: Security Aspects in Java Bytecode Engineering. In: *Blackhat USA 2002 Proceedings*. 2002.

[Sh] Shannon, B. Java™2 Platform Enterprise Edition Specification, v1.4. `http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf`.

[Sua] Sun Microsystems. Java 2 Platform, Standard Edition v 1.4 Datasheet. `http://java.sun.com/j2se/1.4/datasheet.1_4.html`.

[Sub] Sun Microsystems. Java™API for XML processing release notes. http://java.sun.com/webservices/docs/1.2/jaxp/ReleaseNotes.html.

[Suc] Sun Microsystems: *Security Code Guidelines*. http://java.sun.com/security/seccodeguide.html.

[Su03a] Sun Microsystems. Java Plug-in 1.4.2 Developer Guide. 2003. http://java.sun.com/j2se/1.4.2/docs/guide/plugin/developer_guide/contents.html.

[Su03b] Sun Microsystems. Security Advisory on Java Media Framework. 2003. http://sunsolve.sun.com/pub-cgi/retrieve.pl?doc=fsalert%2F54760.

[YB97] Yoder, J. und Barcalow, J. Architectural patterns for enabling application security. 1997. http://citeseer.nj.nec.com/yoder98architectural.html.

LIV - The Linux Integrated Viruswall

Teobaldo A. Dantas de Medeiros

GEINF – CEFET/RN
Centro Federal de Educação Tecnológica
Av. Senador Salgado Filho 1559, Tirol, Natal – RN - BRAZIL
59015-000
teobaldo@cefetrn.br

Paulo S. Motta Pires

DCA/UFRN
Universidade Federal do Rio Grande do Norte
Centro de Tecnologia – Natal – RN - BRAZIL
59.072-970
pmotta@dca.ufrn.br

Abstract This paper presents a system developed in Linux aiming the protection of local area networks containing Windows workstations against malicious agents. The developed solution, named LIV - Linux Integrated Viruswall, besides filtering **SMTP**, **HTTP** and **FTP** traffic destined to the protected network, is capable of detecting malicious agents propagation in the local area network using a technique that we call "sharing-trap". Compromised workstations are isolated from the network and their users are notified, stopping the malicious agent's spread. Results collected from a network protected by LIV, containing thousands of Windows workstations, are presented and discussed. This paper includes information about the recent incident caused by MyDoom worm.

1 Introduction

Malicious agents can be defined as computer programs that operate on behalf of a potential intruder, aiding it on the activity of attacking a system or a network [1]. Once limited to damages in the compromised systems, modern malicious agents acquired new characteristics, as the capability of transmitting private information to the program author, the possibility of remotelly control infected machines and the use of a compromised group of computers on a distributed denial of service (DDoS) attack.

During the year of 2001, the economical impact caused by malicious agents was estimated at $13.2 billion [2]. In 2003, only W32/Sobig.F [3], W32/Nachi [4], W32/Blaster [5] and W32/Slammer [6] worms were responsible for economical losses estimated at $3.25 billion. These values show the growing importance of the adoption of actions that reduce damage caused by the malicious agents on systems and on computer networks.

The traditionally proposed model for protection against malicious agents [7] consists of three protection layers: the first of them acting on the Internet gateway, the second protecting the file and mail servers and the third protecting the workstations. In spite of the immunity propitiated by this model, a basic vulnerability persists: The propagation speed of new malicious agents, unknown by the antiviruses that work on three layers of the model, allows that apparently "protected" networks and workstations continue being infected [8].

In this work, we present an Internet gateway solution that aim to protect local area networks against malicious agents. The solution, named LIV, Linux Integrated Viruswall, is endowed with features implemented in other products [9,10], such as **SMTP**, **HTTP** and **FTP-**traffic filtering [11,12,13], and also incorporates new functionalities. Among those new functionalities, we highlight: the use of the sharing-trap to detect malicious agents spreading in the local area network, analysis of the network traffic generated by the workstations to determine the infected ones, isolation of compromised workstations from the network and the use of a proxy server as a communication channel between the protection system and the users of the workstations. Therefore, LIV is not limited to merely preventing the contamination of the protected network. In the case where a malicious agent gets to enter in the network, deceiving the traditional defense mechanisms, LIV will act limitating its propagation on the LAN.

LIV is constituted of a group of 10 processes, named **ISPAMA** - Integrated System for Protection Against Malicious Agents. **ISPAMA** coordinates the behavior of **CIFS** (Common Internet File System [14]), **SMTP** , **HTTP** and proxy servers. The operation of an antivirus scanner is also controlled by **ISPAMA**. LIV uses the firewall functions of the Linux kernel, via iptables [15], and a database manager for the storage and information exchanging among the various **ISPAMA** processes. The use of a **CIFS** server made possible the creation of a network sharing (sharing-trap), published and made available, without access restrictions, to Windows workstations. The sharing-trap aims to cause the replication of malicious agents to the LIV. Machines that transmit malicious agents to the sharing-trap are considered infected and are isolated from the network. The **SMTP** server acts in the analysis of the e-mail attachments, preventing the entrance of known malicious agents, as well as putting suspect files in quarantine. The **HTTP** server is used for the LIV configuration and, together with the proxy server, acts in the analysis of the downloads made by users of the protected network. The **HTTP** server and the proxy server are also used as a communication channel between the LIV and the users of the network, allowing, for instance, notification of users of the compromised machines in case of infection. The Linux firewall acts in the isolation process of the compromised machines and in the generation of logs related to the workstations' traffic. These logs will be stored later in the LIV database and analyzed by

the **ISPAMA** processes. The log analysis is another method used by the LIV to discover the infected workstations in the network.

After this introduction, we present in the Section 2 the architecture of the LIV. In Section 3, we describe the operation of the **ISPAMA** processes. Section 4 details the operation of **CIFS** server and the sharing-trap. Section 5 discusses the results obtained by LIV in the recent W32/MyDoom [16] incident, reserving to Section 6 the conclusions of the present work.

2 LIV Architecture

The current version of LIV is implemented in Slackware 9.0 Linux distribution [17]. However, there is no known incompatibility with the implementation of LIV in other distributions, since servers and programs that were used in the solution are also available on these other distributions. It is important to emphasize that not all packages used by LIV are present in Slackware 9.0 distribution. Furthermore, some of these packages were substituted by newer versions, or recompiled to support options not available in the distribution version.

At this moment, LIV is protecting a network containing thousands of Windows workstations. The protected network is connected to a single 6 Mbps Internet link. In this particular case, LIV was implemented on a monoprocessed CISC server, with 512Mb of RAM, presenting a quite satisfactory performance. Some results collected from this network configuration are presented on Section 5.

Ten processes run in the LIV server, and these processes are responsible for the implementation of the protection against malicious agents, sharing information amongst themselves by the use of the LIV database. The group of ten processes plus the database is denominated **ISPAMA** - Integrated System for Protection Against Malicious Agents. The specific operation of each **ISPAMA**'s process will be discussed in the Section 3.

Besides the **ISPAMA** processes, several other servers are executed in the LIV machine. **ISPAMA** coordinates the behavior of these servers and, when necessary, activates the functions of the Linux firewall to limit the spread of malicious agents throughout the local area network. A scanner is used to make verifications on e-mail attachments and on downloads.

The applications managed by **ISPAMA** are available for several Linux distributions. These programs are a **CIFS** server, implemented by Samba [18], a **SMTP** server, implemented by Sendmail [19], a **HTTP** server, implemented by Apache [20], and a proxy server, implemented by Squid [21]. It is also necessary to activate the firewall functions of the Linux kernel. Figure 1 shows the general architecture of LIV.

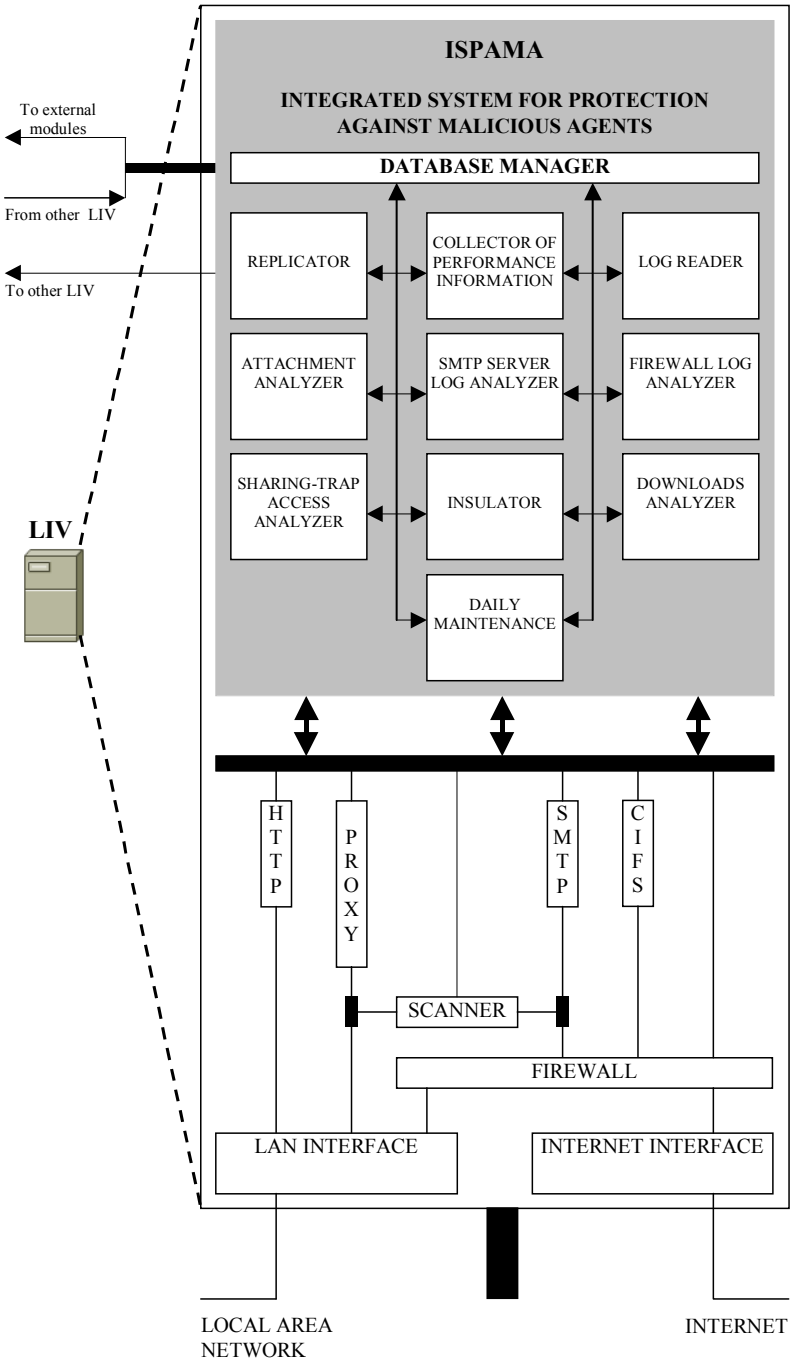


Figure 1: LIV Architecture

As shown on Figure 1, one LIV server can exchange information with another. The use of a larger number of LIV servers make it possible to increase the capacity of limiting malicious agents spread throughout the LAN, in the case where some of them get to enter in the protected network. If a machine is contaminated, LIV will isolate it from the network, preventing that the malicious agents access shares of other workstations or the mail servers of the organization and continue the propagation process on the network. The isolation is implemented in the LIV servers by reconfiguring the Linux firewall. Additionally, LIV allows the use of external insulating modules, that are responsible for the programming of departmental routers of the organization. The external modules instruct routers to filter the packages originated from infected workstations.

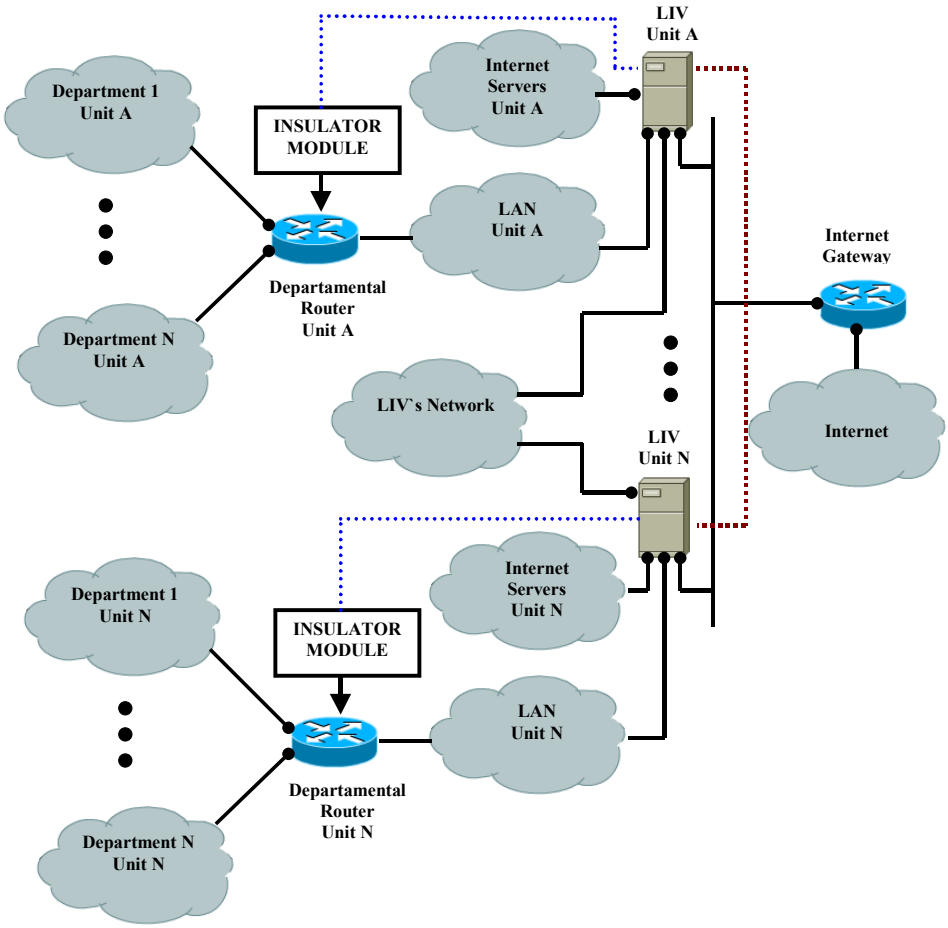


Figure 2: Typical network topology using LIV.

LIV can be used in simple or in complex network topologies. In the simplest network topology, the organization has no Internet servers and only one single LIV is used. In this case, only two network interfaces (LAN and Internet) are needed [22]. Figure 2 shows a more complex situation. In this example, an organization is subdivided in units

and departments. The organization has a single Internet link shared by all units, and each unit has its own Internet servers, including mail servers, and a LIV server. The LIV server is connected to the unit's LAN and to the Internet servers hosted on that unit. Additionally, the unit's LIV is connected to the Internet gateway network and to a communication network constituted of all LIV servers of the organization. This LIV network is used to share information concerning infected workstations and banned e-mail address.

In the next section, we will describe **ISPAMA** processes in detail.

3 ISPAMA Processes

ISPAMA controls all decisions taken by LIV. The **ISPAMA** processes are responsible for reading logs generated by Linux and inserting the records in the LIV database. Based in log analysis, the LIV tries to determine if there are infected workstations in the protected network. Accesses to the sharing-trap and the e-mail attachments verification are also used by LIV to identify infected workstations. Some other processes perform tasks such as replication control, isolation and collection of performance data.

3.1 Log Reader

The log reader stores on the database information generated by the firewall Linux and by the mail server. Only relevant information to the isolation decision is stored. LIV controls the number of log reader processes running according to the amount of records generated by Linux.

3.2 Firewall Log Analyzer

LIV examines the traffic generated by the workstations based on rules defined by its administrator. The rules define the port and the protocol that LIV will monitor in the network, allowing the log reader to configure the firewall so that it will start registering the packets related to the new rules. Besides the port and the protocol, the rules contain other attributes that are periodically anallyzed by the firewall log analyzer. When one of the defined attribute values is exceeded, the workstation that generated these packets will be isolated of the network. Table I summarizes the attributes used in firewall rules.

Attribute	Explanation
Destination Port (*PORT*)	Destination ports of the connections (TCP protocol) or datagrams (UDP protocol) that will be examined by the rule.
Protocol	Protocol used by the packets examined in this rule (UDP or TCP).
Limit of Connections destined to LIV Server (*LCLIV*)	Defines a limit to the number of connections destined to the LIV server that a workstation is allowed to establish using stipulated port/protocol in the interval specified by the rule
Limit of Connections destined to Intranet (*LCIN*)	Defines a limit to the number of connections destined to Intranet addresses that a workstation is allowed to establish using stipulated port/protocol in the interval specified by the rule
Limit of Connections destined to Internet (*LCOUT*)	Defines a limit to the number of connections destined to Internet addresses that a workstation is allowed to establish using stipulated port/protocol in the interval specified by the rule
Limit of Periodical Accesses (*LPA*)	Defines a limit to the number of periodical connections that a workstation is allowed to establish using stipulated port/protocol in the interval specified by the rule
Interval	Time interval used to restrict queries sent to LIV database. Only log records generated in the rule interval are computed when verifying the traffic generated by a workstation

Table I: Firewall rules attributes.

3.3 SMTP Server Log Analyzer

The **SMTP** server log analysis is similar to that described in the previous section for the firewall log analysis. The main difference between them consists on the type of address analyzed. Firewall analysis works with IP addresses of the workstations, and the **SMTP** server analysis works with e-mail addresses. The attributes of the mail server rules also differ from the firewall ones and are presented in Table 2.

Attribute	Explanation
Distinct Originator Addresses (DOA)	Defines how many distinct origin e-mail addresses one workstation can use in the defined rule interval
Recipients Limit (RLM)	Defines the maximum number of distinct recipients in all the e-mail messages sent by a workstation in the rule interval
Messages per Recipient (MPR)	Defines how many messages can be sent by one workstation to the same recipient in the rule interval
External Domain Limit (EDL)	Defines the maximum number of messages sent by a workstation using an external domain in the originator address
Interval	Time interval used to restrict queries sent to LIV database. Only log records generated in the rule interval are computed when verifying the traffic generated by a workstation

Table II: **SMTP** server rules attributes.

If the administrator wishes, some of the rule's attributes can be ignored by LIV in the log analysis processes.

3.4 Insulator

The insulator process is responsible for configuring the Linux firewall whenever a workstation is isolated or reintegrated into the network. The firewall is also configured if the LIV's rules have been changed. The isolation filters the traffic generated by infected workstation, allowing only accesses to essential services, such as name resolution, World Wide Web (**WWW**) to intranet servers and access to proxy ports on LIV servers. Besides interacting with firewall, the insulator alters the proxy server configuration so that it will start blocking the access of infected workstations to the Internet. Whenever an Internet access is denied, the proxy server will inform the workstation user about the infection and the isolation of his machine. The last insulator function is to alter the **SMTP** server configuration so that it will temporarily reject the reception of e-mail sent by addresses that are transmitting malicious agents to the protected network.

3.5 Attachment Analyzer

Each e-mail originated or destined to the protected network is analyzed by LIV. This analysis will initially verify the existence of malicious agents in attachments. If some malicious agent is found, the action taken by LIV will depend on the address of the sender of the message. If the sender's IP address belongs to the intranet, the workstation will be isolated from the network. Otherwise, the sender's e-mail address will be put in a **SMTP** server rejection list. If LIV does not find malicious agent in attachments, it will perform a second analysis, that consists of removing dangerous existing attachments in the message. The LIV's administrator defines which kind of files will be accepted or refused in attachments. Usually, executables, batch files and similar ones should be refused.

3.6 Other ISPAMA Process

The five remaining **ISPAMA** processes are the following: Sharing-trap access analyzer, downloads analyzer, replicator, collector of performance information and daily maintenance. The sharing-trap access analyzer will be discussed in the next section and the operation of the other four processes will be summarized now.

The download analyzer performs a scan operation in files downloaded via the proxy server, being an incumbency of the LIV's administrator to determine which file extensions will be examined and which will not. The replicator process periodically sends information about infected workstations and about the banned e-mails addresses to LIV partners in the protected network. The function of the process collector of performance information is to obtain data about the CPU usage and memory resources

on the LIV server. The collected data are presented graphically in the LIV WEB interface. Finally, the daily maintenance process accomplishes tasks like exclusion of old log records stored in LIV database and the removal of files put in quarantine on server's disk.

4 The Sharing-trap

The sharing-trap is a technique used by LIV to detect workstations compromised by malicious agents that are capable of spreading themselves throughout local area network. Any workstation can access the sharing-trap. There are no access control, therefore LIV will accept access independently of the network credentials informed to the **CIFS** server. Additionally, if a workstation searches for a sharing name inexistent in the **CIFS** server, LIV will map this access to the sharing-trap. Figure 3 illustrates how a Windows workstation sees the sharing-trap in the network.

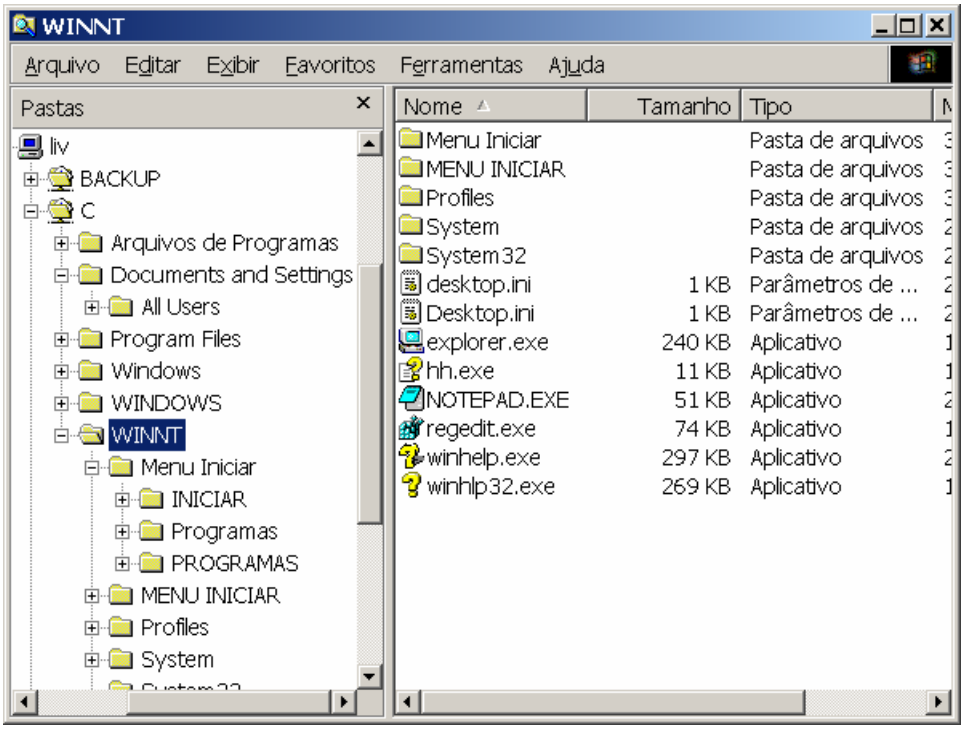


Figure 3 Sharing-trap accessed using a Windows Workstation.

When an access to the sharing-trap is concluded, **CIFS** server runs a LIV process that scans for malicious agents. LIV will isolate the workstation that accomplished the last access to the sharing-trap if some malicious agent is found in the scan. The files of the sharing-trap are restored in the case of some alteration is detected during the analysis

process. Figure 4 shows the key configuration of the Samba server to implement the sharing-trap.

```
1.  [global]
2.      workgroup = EXAMPLE
3.      netbios name = LIV
4.      server string = Linux Integrated Viruswall
5.      interfaces = 192.0.2.0/255.255.252.0
6.      bind interfaces only = Yes
7.      security = DOMAIN
8.      encrypt passwords = Yes
9.      map to guest = Bad Password
10.     password server = EX_DC1, EX_DC2
11.     username map = /etc/samba/private/smbalias
12.     deadtime = 2
13.     wins server = EXAMPLE:192.0.2.10
14.     default service = C
15.     remote announce = 192.0.2.10/EXAMPLE 192.0.2.11/EXAMPLE
16.
17. [C]
18.     comment = LIV Sharing Trap
19.     path = /usr/local/liv/armadilha
20.     admin users = nobody
21.     read only = No
22.     guest ok = Yes
23.     root postexec = /usr/local/liv/cifs %I &
24.     volume = LIV
25.     fstype = FAT
26.     dos filemode = Yes
27.     dos filetimes = Yes
28.     dos filetime resolution = Yes
```

Figure 4. Key configuration of the Samba server to implement the sharing-trap.

Figure 4 shows the case of a LIV server that is member of the Microsoft domain EXAMPLE. Line 9 of the configuration file instructs the Samba server to map invalid user accesses to a guest account. This guest account can access the sharing-trap "C", configured by the lines 18-28. Line 14 redirects accesses to inexistent shares names to the sharing-trap. Line 22 allows guest access to the sharing-trap, and line 20 grants administrative privileges to the guest account. Line 23 states that sharing-trap access analyzer is activated after each share access.

In the next section, we will discuss the results obtained by LIV in two months of operation protecting a network composed of thousands of Windows workstations.

5 Results

The results presented in this section were obtained in a network containing more than 6,000 Windows workstations distributed by approximately 180 remote places in the Brazilian state of Rio Grande do Norte. The network topology is similar to that presented in Figure 2. The data were collected in the period from January 14, when LIV was implanted in the network, until February 26, 2004. During this period, LIV analyzed 691,184 e-mails, removing 6,658 malicious agents found in attachments. The amount of scanned downloads has summed 40,467, on which 38 were infected. Figure 5 presents

incidents involving all known variants of MyDoom, Bagle [23] and NetSky [24] in the protected network with a period of one day for each interval. In all these cases, LIV removed the malicious agents from the message, replacing the attachment with a warning message. After that, the warning message was sent for the sender and for the recipient of the e-mail.

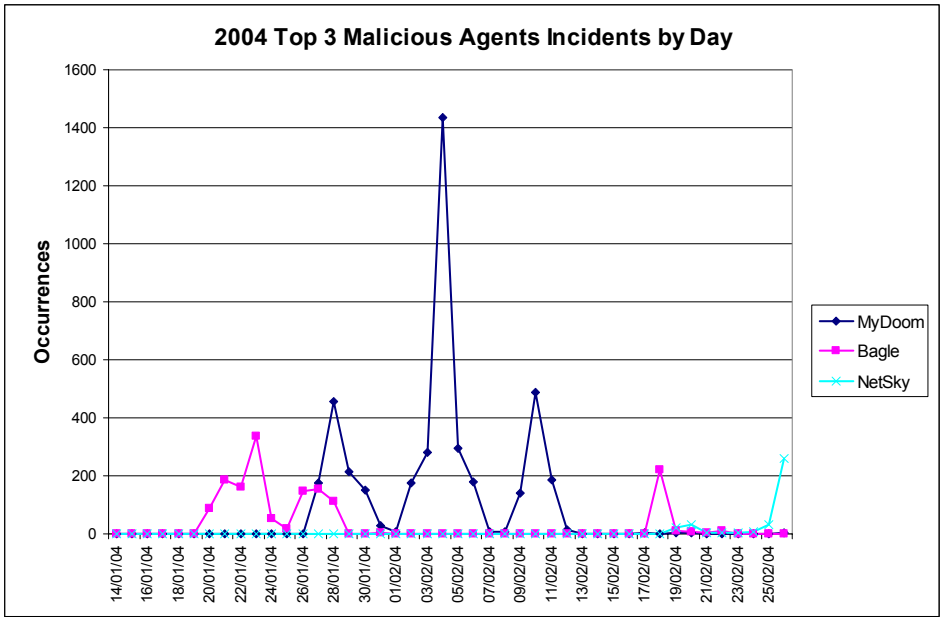


Figure 5. MyDoom, Bagle and NetSky removed from e-mail attachments with a daily period.

Table III presents some data obtained from LIV database about MyDoom.

GENERAL DATA	
MyDooms removed from attachments	4253
Number of infected machines in the protected network	31 (0.52 %)
Propagation peak	04/02/2004
Percentage of infected e-mails in the peak of propagation	6.05 %
SOURCE OF E-MAIL MESSAGES CONTAINING MYDOOM	
LAN (sent by the 31 infected machines before isolation)	3275 (77 %)
Internet	978 (23 %)
→ Average number of e-mails sent from infected machines before the isolation (with an maximum interval of 5 minutes):	105.65
SOURCE OF E-MAIL MESSAGES CONTAINING MYDOOM AND COMING FROM THE INTERNET	
Brazil	881 (90,1%)
Other Countries	97 (9,9%)

Table III. Information concerning MyDoom collected from LIV database.

An inquiry made in some of the 31 stations infected by MyDoom in the protected network had demonstrated that these machines were used in others networks (most of them are notebooks) or that they have also been connected to the Internet via dial-up. In some cases, it was reported that machines were compromised when reading infected messages in Internet webmails, placed outside the LIV protection perimeter. This occurs because some webmails are incompatible with the download protection method used by LIV.

Another significant result of LIV refers to the performed isolation operations. The network where LIV is operating had hundreds of infected machines prior to the LIV implantation. On January 14, when LIV has started its operation, practically all of them were immediately isolated. From January 14 until February 26, LIV accomplished 672 isolation operations. In 376 of these isolations, the network support team of the organization was able to determine with reliability whether the machine was or not infected. In 91.76% of these cases, the machine was really infected. The 8.24% remaining constitute the false positives, when LIV isolates a machine that is not infected. A false positive can occur due to wrong network configuration at workstations, or due to improper LIV parameters setting. Typically, when LIV server is configured so that it will be able to detect and isolate a higher number of infected workstations, it will also cause more false positives.

In the next Section we present the conclusions of this work.

6 Conclusions

We presented a system for protection against malicious agents that acts on the Internet gateway of the network. The solution, named LIV - Linux Integrated Viruswall, is capable of preventing malicious agents entrance in the protected network as well as detecting already infected workstations. Compromised workstations containing malicious agents are isolated from the network and their users are notified. LIV introduces new features in comparison with other solutions. First, LIV uses the sharing-trap technique to detect malicious agents spread through LAN. Additionally, LIV analyzes the network traffic generated by workstations to verify if they are compromised. Another innovative feature is the use of the proxy server as a communication channel with the users of the infected machines, making it possible to inform them when a malicious agent infect their machines even when there is no local antivirus installed.

After 44 days of its implantation on a network composed of more than 6,000 Windows workstations, LIV had already blocked the entrance of approximately 6,700 malicious agents in the protected network. LIV had also detected the infection of at least 345 machines, isolating them from network, and avoiding the malicious agents spread. In the analyzed period, 368 attached files had been put in quarantine. Many of these files contained malicious agents still unknown at the time in which they were analyzed by the LIV's scanner.

The features incorporated to LIV, together with the results obtained, demonstrate that it is possible to significantly increase the security of a computer network against malicious agents by using a regular computer with no special hardware, acting in the gateway of the protected network.

References

[1] Zelser, Lenny: The Evolution of Malicious Agents. Online publication, 2000. Available at http://www.zeltser.com/agents, may 2004 .

[2] Computer Economics: Malicious Code Attacks Had $13.2 Billion Economic Impact in 2001. Online publication, 2002. Available at http://www.computereconomics.com/article.cfm?id=133, may 2004.

[3] CERT: CERT® Incident Note IN-2003-03. W32/Sobig.F Worm. Online publication, 2003. Available at http://www.cert.org/incident_notes/IN-2003-03.html, may 2004.

[4] Symantec: W32.Welchia.Worm. Online publication, 2003. Available at http://securityresponse.symantec.com/avcenter/venc/data/w32.welchia.worm.html, may 2004.

[5] CERT: CERT® Advisory CA-2003-20 W32/Blaster worm. Online publication, 2003. Available at http://www.cert.org/advisories/CA-2003-20.html, may 2004.

[6] CERT: CERT® Advisory CA-2003-04 MS-SQL Server Worm. Online publication, 2003. Available at http://www.cert.org/advisories/CA-2003-04.html, may 2004.

[7] Trend Micro: Virus Protection Across The Enterprise. Online publication, 2003. Available at http://www.trendmicro.com/en/products/gateway/gatelock5000/evaluate/whitepaper.htm, may 2004

[8] CERT: CERT® Incident Note IN-2003-01. Malicious Code Propagation and Antivirus Software Updates. Online publication, 2003. Available at http://www.cert.org/incident_notes/IN-2003-01.html, may 2004.

[9] AMaViS Team: AMaViS - A Mail Virus Scanner. Online publication, 2004. Available at http://www.amavis.org/, may 2004.

[10] Trend Micro: InterScan VirusWall - Features. Online publication, 2004. Available at http://www.trendmicro.com/en/products/gateway/isvw/evaluate/features.htm, may 2004.

[11] Postel, J: Simple Mail Transfer Protocol. Online publication, 1982. Available at http://www.ietf.org/rfc/rfc0821.txt, may 2004.

[12] Fielding, R. et al.: Hypertext Transfer Protocol - HTTP/1.1. Online publication, 1999, Available at http://www.ietf.org/rfc/rfc2616.txt, may 2004.

[13] Postel, J et al.: File Transfer Protocol. Online publication., 1985. Available at http://www.ietf.org/rfc/rfc959.txt, may 2004.

[14] Hertel, C: IMPLEMENTING CIFS - The Common Internet File System, Prentice Hall Professional Technical Reference (PTR), 2003.

[15] Russel, Rusty et al.: THE NETFILTER/IPTABLES PROJECT. Online publication, 2004. Available at http://www.netfilter.org/, may 2004.

[16] CERT: CERT® Incident Note IN-2004-01 - W32/Novarg.A Virus. Online publication, 2003. Available at http://www.cert.org/incident_notes/IN-2004-01.html, may 2004.

[17] Slackware Linux, inc.: The Slackware Linux Project. Online publication, 2003. Available at http://www.slackware.com, may 2004.

[18] Samba Team: The Samba Web Pages. Online publication, 2004. Available at http://www.samba.org, may 2004.

[19] Sendmail: Sendmail Home Page. Online publication, 2004. Available at http://www.sendmail.org, may 2004.
[20] Apache Software Foundation: The Apache Httpd Server Project. Online publication, 2004. Available at http://httpd.apache.org, may 2004.
[21] Team Squid: Squid Web Proxy Cache. Online publication, 2004. Available at http://www.squid-cache.org, may 2004.
[22] Dantas de Medeiros, Teobaldo A.; Motta Pires, Paulo S. : LIV: LINUX INTEGRATED VIRUSWALL – UMA ESTRATÉGIA PARA A PROTEÇÃO DE ESTAÇÕES DE TRABALHO CONTRA CÓDIGOS MALICIOSOS EXECUTÁVEIS: Proc. of IV Simpósio de Segurança em Informática – ITA/CTA, São José dos Campos – SP – Brazil, 2002; p 38-43 (in portuguese).
[23] Network Associates: W32/Bagle.b@MM. Online publication, 2004. Available at http://vil.nai.com/vil/content/v_101030.htm, may 2004.
[24] Computer Associates: Virus Information Center - Win32.Netsky.B. Online publication, 2004. Available at http://www3.ca.com/virusinfo/virus.aspx?ID=38332, may 2004.

Risiken der Nichterkennung von Malware in komprimierter Form

Heiko Fangmeier, Michel Messerschmidt, Fabian Müller, Jan Seedorf

antiVirusTestCenter
Fachbereich Informatik, Universität Hamburg
Vogt-Kölln-Straße 30
22527 Hamburg
5fangmei@informatik.uni-hamburg.de
uni@michel-messerschmidt.de
9fmuelle@informatik.uni-hamburg.de
seedorf@informatik.uni-hamburg.de

Abstract: Maliziöse Software (Malware) gefährdet die Vertraulichkeit, Integrität und die Verfügbarkeit von Informatiksystemen auf verschiedene Art und Weise. In diesem Beitrag wird der Frage nachgegangen, inwiefern durch Malware in komprimierter Form Risiken entstehen. Ausgewählte Risiken werden anhand eines Szenarios veranschaulicht und analysiert.

Ein Standard-Schutzmechanismus vor Malware ist Anti-Malware-Software. Es wird eine Testmethodik vorgestellt, mit der systematisch die Güte der Erkennung von Malware in komprimierter Form durch Anti-Malware Software getestet werden kann. Abschließend werden mit dieser Methodik erlangte Testergebnisse vorgestellt.

1 Risiken durch Malware in komprimierter Form

1.1 Einleitung

Maliziöse Software (Malware) gefährdet die Vertraulichkeit, Integrität und die Verfügbarkeit von Informatiksystemen auf verschiedene Art und Weise. Ein etablierter Schutzmechanismus gegen Malware ist der Einsatz von Anti-Malware Software. Diese kann gegenüber einer Testmenge mit maliziöser Software getestet werden, um eine Aussage über den gewährten Schutz zu ermöglichen.

In diesem Beitrag wird der Frage nachgegangen, inwiefern Risiken der Umgehung des durch Anti-Malware Software gebotenen Schutzes durch Malware in komprimierter Form bestehen. Um die Risiken zu verdeutlichen, wird in einem Beispielszenario aufgezeigt, wie der Schutz durch komprimierte Malware umgangen werden kann. Im zweiten Abschnitt wird eine Testmethodik vorgestellt, mit der die Güte der Erkennung

komprimierter Malware von Anti-Malware Software gemessen werden kann. Abschließend werden Ergebnisse eines mit der vorgestellten Methodik durchgeführten Tests präsentiert.

Prinzipiell besteht ein Risiko der Umgehung einer Anti-Malware Software immer dann, wenn ein Anti-Malware Programm ein bestimmtes Kompressionsformat nicht unterstützt. Dann kann mit diesem Format komprimierte Software unerkannt auf einen eigentlich geschützten Rechner gelangen. Dieses Risiko kann allerdings dadurch gemindert werden, dass die Anti-Malware Software den Rechner im on-access Modus schützt[1]. Hierbei wird jeder ausführbare Code beim Zugriff durch eine im Hintergrund aktive Anti-Malware Software überprüft. So kann zwar komprimierte Malware unerkannt auf einen Rechner gelangen, aber bei der Ausführung (nach einer Dekompression) wird sie erkannt und eine Infektion verhindert.

Ein zusätzliches Risiko besteht bei laufzeit-komprimierter Malware[2], da diese erst während der Ausführung dekomprimiert wird. Wird das entsprechende Kompressionsformat von der Anti-Malware Software nicht unterstützt, kann die Aktivierung der Malware auch im on-access Modus in der Regel nicht verhindert werden. Im durchgeführten Test konnte die Erkennung von laufzeit-komprimierter Malware aus Komplexitätsgründen jedoch nicht getestet werden.

Es sind jedoch auch Szenarien denkbar, zum Beispiel in zentral administrierten Netzwerken, in denen nicht jedes Rechensystem durch eine lokal installierte Anti-Malware Software im on-access Modus geschützt wird. Hier besteht nicht nur das Risiko der Verbreitung von unerkannter maliziöser Software, sondern auch das Risiko der Ausführung und damit der Infektion des betroffenen Systems.

Ein weiteres Risiko besteht durch bestimmte, stark komprimierte Archive, die beim Entkomprimieren potentiell die Anti-Malware Software zum Absturz bringen und somit deren Verfügbarkeit gefährden [Bi03].

1.2 Ein Beispielszenario

Abbildung 1 zeigt ein beispielhaftes Szenario zur Veranschaulichung von potentiellen Risiken. In diesem Szenario als einzelne Rechner oder Server bezeichnete Symbole können entweder für einen einzelnen Computer dieses Typs oder aber für eine gesamte Gruppe dieser Rechner stehen (Cluster). Dieses ist für die Betrachtung in Bezug auf Malware nicht relevant, da ein Rechnercluster als infiziert betrachtet werden kann, sobald ein Rechner dieses Clusters infiziert ist.

[1] Anti-Malware Programme können bösartige Software (Malware) grundsätzlich in zwei verschiedenen Betriebsarten erkennen: Im on-demand Modus wird die Software bei Bedarf (engl. "demand") aktiviert und zur Überprüfung (Scannen) von Dateien oder Verzeichnissen eingesetzt. Im on-access Modus ist die Software im Hintergrund aktiv und überwacht die Aktivitäten auf dem Rechner. Sie überprüft bei jedem Dateizugriff (engl. "access") automatisch im Hintergrund die entsprechende Datei auf bösartige Software.

[2] Eine laufzeit-komprimierte (engl. „runtime compression“) Software besteht in der Regel aus einer kurzen Dekompressionsroutine gefolgt von dem komprimierten Code, der erst direkt vor der Aktivierung im Speicher dekomprimiert wird. Für die Erkennung von Malware sind dabei auch Verfahren zu berücksichtigen, bei denen der ausgeführte Code niemals komplett dekomprimiert vorliegt.

Rechner B, Rechner C und der Gateway-Server A befinden sich in einem lokalen Netzwerk (LAN). Rechner B ist ein Laptop, der sowohl gelegentlich im Firmennetz angeschlossen ist, als auch eine direkte Verbindung ins Internet haben kann (bei Einsatz außerhalb des betrachteten LANs). Rechner C entspricht einem typischen Büroarbeitsplatz. Er ist über das Gateway A an das Internet angeschlossen. Server/Gateway A ist der zentrale Punkt der Netzanbindung.

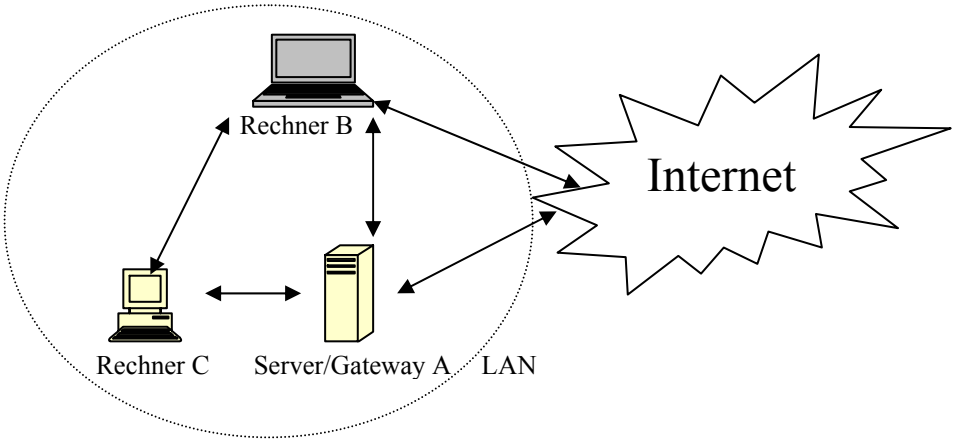


Abbildung 1: Beispielszenario

Generell besteht in einem lokalen Netzwerk das Risiko, dass auf einem mit Anti-Malware Software geschützten Rechner Malware in komprimierter Form abgelegt aber nicht ausgeführt wird (vgl. 1.1), und diese dann von einem nicht geschützten Rechner des Netzwerkes aufgerufen bzw. kopiert und ausgeführt wird. Genauso kritisch ist ein Ausfall eines Anti-Malware Produktes auf einem der Rechner. Im Einzelnen sind zum Beispiel folgende Möglichkeiten der Ausbreitung von maliziöser Software und der Infektion von Rechnern in einem lokalen Netzwerk denkbar:

Wird auf dem Gateway-Rechner A, der das Netzwerk schützen soll, ein Anti-Malware Produkt im on-access Modus eingesetzt, das ein bestimmtes Kompressionsformat nicht unterstützt, so kann mit diesem Format komprimierte Malware unentdeckt zu Rechner B und C gelangen. In diesem Fall würden alle Rechner ohne Anti-Malware Schutz infiziert. Selbst beim Einsatz eines Anti-Malware Produktes auf Rechner C im on-access Modus ist eine Infektion denkbar, falls dieses Produkt die (durchgelassene) Malware auch unkomprimiert nicht erkennt. Dies kann der Fall sein, wenn auf Rechner C ein anderes Produkt als auf dem Gateway eingesetzt wird, oder wenn die Software auf Rechner C aus anderen Gründen nicht schützt (veraltete Signaturen, Absturz, Fehlkonfiguration).

Ein weiteres Risiko ist denkbar, wenn Rechner B außerhalb des Netzwerkes eingesetzt wird (zum Beispiel auf Reisen), und komprimierte Malware auf den Rechner gelangt, deren Kompressionsformat von der auf Rechner B laufenden Anti-Malware Software nicht unterstützt wird. In diesem Fall kann die Malware auf Rechner C gelangen und

dort zur Infektion führen (weil gar kein Anti-Malware Schutz besteht oder aus in obigem Beispiel genannten Gründen). Dies kann sogar dann passieren, wenn die auf dem Gateway-Rechner eingesetzte Anti-Malware Software Malware und Kompressionsformat erkennen würde, da eine direkte Verbindung im Netzwerk von Rechner B zu Rechner C besteht.

Aus den obigen Ausführungen ist zu ersehen, dass sich kein Risiko durch komprimierte Malware manifestieren kann, solange jede Komponente des Netzwerkes durch Anti-Malware Software im on-access Modus geschützt ist[3]. Wenn allerdings, intentional oder unabsichtlich, mindestens eine Komponente im Netzwerk nicht geschützt ist (z.B. aus Performance-Gründen), kann eine Nichterkennung von komprimierter Malware zu einer Infektion des Netzwerkes führen.

2 Testen der Erkennung von komprimierter Malware durch Anti-Malware Software

Ein Standard-Schutzmechanismus - auch gegen komprimierte Malware - ist Anti-Malware Software. Es stellt sich die Frage, inwiefern aktuelle Anti-Malware Programme ausreichend guten Schutz gegen die aufgezeigten Risiken bieten, indem sie Malware auch in komprimierter Form erkennen. Im anti Virus Test Center (aVTC) der Universität Hamburg wird seit Jahren regelmäßig Anti-Malware Software getestet. Um die Frage der Güte von Anti-Malware Programmen hinsichtlich der Erkennung von Malware in komprimierter Form zu untersuchen, wurde die im aVTC eingesetzte Methodik so angepasst, dass im Rahmen eines Tests von Anti-Malware Software folgende Fragen beantwortet werden können:

- Welches Produkt unterstützt welches Komprimierungsformat ?
  Um eine Aussage zu machen, welches Anti-Malware Programm welchen Schutz hinsichtlich komprimierter Malware bietet, gilt es zu prüfen, welches Produkt welche Formate bei der Erkennung von maliziösem Code unterstützt.

- Mit welcher Güte werden die Formate unterstützt ?
  Es wird nicht nur getestet, welches Produkt welche Formate unterstützt, sondern auch die Qualität der von den Produkten jeweils eingesetzten Dekomprimierungsroutine[4], um nachzuprüfen, ob die getestete Anti-Malware Software auch zuverlässig alle Malware in einem von ihr unterstützen Format erkennen kann, so sie diese Malware denn auch unkomprimiert erkennt.

[3] sofern diese die Malware unkomprimiert erkennt und es sich nicht um eine Laufzeit-Komprimierung handelt

[4] Es wird davon ausgegangen, dass die getesteten Produkte im Regelfall die komprimierte Malware dekomprimieren und dann mit ihren Signaturen vergleichen. Dies muss nicht der Fall sein: die Produkte können auch für jede Kombination aus Malware und Kompressionsformat eine eigene Signatur erstellen.

- Werden alle Versionen eines Archivformates erkannt ?
  Erfahrungen zeigen [Br03], dass sich einige Komprimierungsformate substantiell in unterschiedlichen Versionen des jeweiligen Formats unterscheiden. Es gilt zu verifizieren, dass ein Produkt auch Malware in allen Versionen eines Formates erkennt. Ist dies nicht der Fall, entsteht die Gefahr, dass sich der Benutzer der Software zu Unrecht auf die Unterstützung eines bestimmten Komprimierungsformates durch die eingesetzte Anti-Malware Software verlässt.

- Unterstützen die getesteten Anti-Malware Programme auch unterschiedliche Modi eines Komprimierungsformates ?
  Neben unterschiedlichen Versionen bieten viele Komprimierungsprogramme auch die Kompression in unterschiedlichen Modi an (z.B. selbstextrahierende Archive, solide Archive, u.ä.). Damit sich der Benutzer auf die eingesetzte Anti-Malware Software verlassen kann, gilt es zu verifizieren, dass auch alle Modi eines Komprimierungsformates unterstützt werden.

- Wie reagieren die getesteten Anti-Malware Programme bei Problemen mit komprimierten Dateien ?
  Sollte ein Anti-Malware Programm nicht in der Lage sein, eine komprimierte Datei zu überprüfen, muss dies dem Benutzer gemeldet werden, damit dieser nicht fälschlicherweise die betreffende Datei als nicht-maliziös erachtet.

- Erkennen die getesteten Anti-Malware Programme auch Malware in mehrfach rekursiv gepackten Archiven ?
  Ein ausreichender Schutz durch Anti-Malware Software besteht nur dann, wenn diese auch mehrfach rekursiv gepackte Archive zuverlässig erkennt, da sonst der Schutz durch Anti-Malware Software leicht umgangen werden kann.

Im Rahmen der vorgestellten Testmethodik wird nicht systematisch geprüft, inwiefern die Verfügbarkeit der getesteten Produkte durch denial-of-service Attacken mit bestimmten Archiven gefährdet werden kann (vgl. 1.1, [Bi03]).

2.1 Angewandte Testmethodik

Im anti Virus Test Center (aVTC) der Universität Hamburg werden auf der Grundlage von ethischen Grundsätzen [Br01] in einer abgeschotteten Testumgebung Tests von Anti-Malware Software durchgeführt. Dazu wird eine Testmenge mit Malware auf einem Server bereitgestellt, an der die Testprodukte gemessen werden, indem per Netzwerkzugriff im on-demand Modus[5] die Testmenge überprüft wird. Durch Auswertung der dabei erzeugten Meldungen (Logdateien) werden die Erkennungsrate (Anteil der erkannten Malware), die Erkennungsgenauigkeit (gleiche Identifikation gleicher Varianten) sowie die Erkennungszuverlässigkeit (zuverlässige Erkennung aller infizierten Samples einer Malwarevariante) berechnet.

[5] Um die Güte der Kompressionsunterstützung zu testen, werden im Rahmen der vorgestellten Testmethodik Tests im on-demand Modus durchgeführt. [Si02] hat nachgewiesen, dass die so erzielten Ergebnisse auch für den Einsatz der Produkte im on-access Modus angenommen werden können.

Die Vergleichbarkeit der Produkte wird durch einen festgelegten Stichtag für die Produktversionen und Signaturen erreicht. Damit die erzielten Testergebnisse möglichst objektiv, nachvollziehbar und reproduzierbar sind, werden bei sämtlichen aVTC-Tests alle verwendeten Einstellungen, die verwendete Hardware, die Testumgebung und die Testmethodik ausführlich dokumentiert und veröffentlicht [AV04]. So können die erzielten Ergebnisse jederzeit reproduziert werden.

Die Verzeichnisstruktur im aVTC ordnet die Musterdateien hierarchisch: <Testmenge> \ <Plattform> \ <Bezeichnung> \ <Variante> \ <Objekt>. Zur automatischen Auswertung der von den Testprodukten erzeugten Protokolldateien wird im aVTC die Skriptsprache Perl verwendet. Die Verarbeitung einer Protokolldatei erfolgt in mehreren Schritten:

- Protokolldatei in einheitliches Format umwandeln (Trennung des Protokolls in Pfad des getesteten Objektes, Meldung des Testproduktes, gemeldete Malwarebezeichnung für getestetes Objekt)
- Protokolldatei aufteilen (infiziert gemeldete Dateien, nicht infiziert gemeldete Dateien, übrige Zeilen)
- Erkennungsrate und andere Kriterien ermitteln
- Überprüfung des Protokolls

Ergeben sich bei der Auswertung der Protokolldateien Unstimmigkeiten oder hat ein Produkt nicht alle Objekte der getesteten Testmenge im Protokoll gemeldet, werden diese Objekte bis zu zwei Mal erneut getestet und ausgewertet. Durch dieses wiederholte Testen haben die Testprodukte eine weitere Chance, auch auf diese Objekte zuzugreifen, und andere potentielle Fehlerquellen können minimiert werden.

Aufgrund der speziellen Fragestellungen (s.o.) zum Test von komprimierter Malware wurde die aVTC-Methodik angepasst. Statt der Erkennungsrate ist ausschließlich der Einfluss der Kompressionsformate auf die Erkennungsrate von Interesse, um Aufschluss über die Güte der Unterstützung von Kompressionsformaten zu gewinnen. Deshalb werden die als Testdaten verwendeten Malwaredateien sowohl unkomprimiert (Referenztestmenge) als auch in verschiedenen Kompressionsformaten überprüft. Aus der Differenz zwischen der Erkennungsrate unter Anwendung der jeweiligen Kompression und der unkomprimierten Referenzergebnisse ergibt sich die Qualität der Unterstützung des jeweiligen Kompressionsformats. Die Minderung der Erkennungsrate (in Prozentpunkten) pro Produkt und Kompressionsformat berechnet sich wie folgt:

$$\text{Minderung der Erkennungsrate}_{\text{Produkt, Format}} = \text{Erkennungsrate Referenztestmenge}_{\text{Produkt}} - \text{Erkennungsrate}_{\text{Produkt, Format}}$$

Die Kompressionsformate werden (soweit möglich) auf folgende Arten verwendet, um auch weniger offensichtliche Schwächen der Anti-Malware Produkte zu identifizieren:

- Standard-Kompression[6]
- Die gesamte Referenztestmenge (inkl. Verzeichnisstrukturen) in einem Archiv
- Archivdateien werden umbenannt (generischer Name ohne Dateiendung)
- Erzeugung selbst-extrahierender Archive
- Erzeugung von passwort-geschützten Archiven
- Rekursive Archive: Jeder Komprimierungsvorgang wird 2x bzw. 9x durchgeführt

Unterschiedliche Formatversionen und Spezialmodi einzelner Kompressionsformate[7] werden im Rahmen des hier vorgestellten Tests als einzelne Kompressionsformate betrachtet.

2.2 Testdurchführung

Der Test wurde auf einem isolierten Rechner mit einer lokal verfügbaren Testmenge auf Windows 2000 durchgeführt. Auf die Hypothese, dass die erzielten Testergebnisse auf andere Win32-Betriebssysteme übertragen werden können (z.B. Windows XP), wird im Rahmen dieses Beitrags nicht eingegangen. Entsprechend der Testmethodik des aVTC wurde vor jedem Produkttest ein System-Image des frisch installierten Betriebssystems neu aufgespielt, um Wechselwirkungen der verschiedenen Anti-Malware Produkte auszuschließen.

Insgesamt wurden 25 Anti-Malware Produkte auf die Erkennung in 32 Kompressions- und Archiv-Formaten (inklusive verschiedener Format-Versionen) getestet. In dem hier beschriebenen Test wurden als Referenztestmenge „in-the-wild“ Fileviren[8] verwendet, die in der Wildlist für Oktober 2001 [We01] aufgeführt sind. Durch die Verwendung dieser sich seit 2001 „in-the-wild“ befindlichen Viren ist die Annahme gerechtfertigt, dass nahezu alle Anti-Malware Produkte diese Samples in unkomprimierter Form einwandfrei erkennen können (vgl. [Br02]).

3 Testergebnisse

Im Folgenden sollen einige Ergebnisse des durchgeführten Tests kurz vorgestellt werden. Eine komplette Auflistung aller erzielten Testergebnisse findet sich unter [AV04].

---

[6] Standardkompression des jeweiligen Formates, jeweils alle infizierten Samples einer Malwarevariante in einem Archiv

[7] z.B. Rar: solide Archive

[8] Die Gesamt-Testmenge des aVTC ist nach Plattformen, auf denen die maliziöse Software lauffähig ist, unterteilt. Dementsprechend gibt es Makro-, Skript-, File- und Bootviren als Testmenge. Für den vorgestellten Test von komprimierter Malware wurden File-in-the-wild Viren als Referenztestmenge verwendet. Dem liegt die nicht überprüfte Hypothese zugrunde, dass die getesteten Produkte die maliziöse Software zuerst dekomprimieren und die Dekomprimierungsroutinen sich bei anderen Testmengen (z.B. Skriptviren) ähnlich verhalten.

3.1 Unterstützung von Kompressionsformaten durch die getesteten Produkte

Abbildung 2 zeigt die getesteten Produkte und Kompressionsformate; Abbildung 3 zeigt einen Überblick über die Unterstützung von Kompressionsformaten durch die getesteten Produkte[9]. Eine weiße Zelle bedeutet keine Abnahme der Erkennungsrate gegenüber der unkomprimierten Malware in der Referenztestmenge, das heißt, das entsprechende Format wird ohne Fehler unterstützt und alle unkomprimiert erkannte Malware wird auch unter diesem Format komprimiert erkannt. Das Gegenteil ist bei den dunkelgrauen Zellen der Fall: hier ist die Erkennungsrate um 100 Prozentpunkte abgefallen, d. h. das entsprechende Format wird nicht unterstützt.

Produkte		Kompressionsformate	
ANT Antivir	PER Per Antivirus	7Z_ 7-Zip	RA1 Rar v1
AVA Avast!	PRO Protector	AC2 Ace v2	RA2 Rar v2
AVG AVG Antivirus System	QHL QuickHeal	ACE Ace v1	RA3 Rar v3
AVK Antiviren Kit	RAV RAV Antivirus	ARC Arc	RAR Rar v3 (solid comp.)
AVP Kaspersky Antivirus	SCN McAfee ViruScan	ARJ Arj	SHA Shell Archive
BDF BitDefender	SWP Sophos Anti Virus	B64 MIME Base64	SQZ Squeeze It
CMD Command Antivirus	VBR VirusBuster	BH_ Black Hole	TAR Tape Archive
DRW Dr. Web	VSP VirScanPlus	BZ2 Bzip2	UC2 Ultra Compressor 2
FIR Fire Anti-virus Kit		CAB Cabinet File	UUE UUEncode
FPR F-Prot for Windows		CMS MS Compress	ZI2 PkZip 6.0 (zip2.04 compatible)
FSE F-Secure		GZ_ Gzip	ZI6 PkZip 6.0
GLA Gladiator Antivirus		HA_ Ha	ZIB PkZip 6.0 (bzip2 comp.)
IKA Ikarus Virus Utilities		HAP Hap	ZID PkZip 6.0 (DCLimplode comp.)
INO eTrust Antivirus		JAR Jar	ZIE PkZip 6.0 (Deflate64 comp.)
NAV Symantec Antivirus		JAV Java Archive	ZIP InfoZip 2.3
NVC Norman Virus Control		LHA Lha	ZOO Zoo
PAV Power Antivirus		PAK Pak	

Abbildung 2: Getestete Produkte und Kompressionsformate

Bei den Ergebnissen in den hell- und mittelgrauen Zellen beträgt die Minderung der Erkennungsrate weder 0 noch 100 Prozentpunkte. Das bedeutet, dass diese Formate von den jeweiligen Produkten zwar prinzipiell unterstützt werden und auch damit komprimierte Malware erkannt wird. Allerdings werden in komprimierter Form weniger Dateien als unkomprimiert erkannt. Dies deutet auf eine schlechte Implementierung der Dekompression in den betroffenen Programmen hin und zeigt eine Schwäche in der Abwehr gegenüber so komprimierter Malware. Auffällig ist, dass kein Produkt im Test alle Modi des Zip-Formates vollständig unterstützt. Bei einigen Modi dieses Formates (ZIB, ZID) haben alle Produkte im Test Probleme bei der Dekompression.

[9] Standard-Kompression, Kompressionsformate in der linken Spalte, getestete Produkte in der ersten Zeile

Minderung der Erkennungsrate								
	0		0,1 - 20		20,1 – 99,9		100	

	ANT	AVA	AVG	AVK	AVP	BDF	CMD	DRW	FIR	FPR	FSE	GLA	IKA	INO	NAV	NVC	PAV	PER	PRO	QHL	RAV	SCN	SWP	VBR	VSP
7Z_																									
AC2																				0,9	92,1				
ACE																				0,9	6,6				
ARC				30,3																	32,8				
ARJ																				0,9					
B64		0,2											34,0						97,7						
BH_																									
BZ2																					2,9				
CAB																				0,9					
CMS							0,9			0,9															
GZ_	99,8																								99,2
HA_																					69,2				
JAR																									
JAV															60,0					0,9					
LHA														6,3	0,7		0,5								
PAK						97,9																			
RA1							95,0			95,0				99,5						0,9					
RA2																				0,9					
RA3																				0,9					
RAR																				0,9					
SHA					55,2		89,3			89,1				55,7		56,2	55,2						55,3		
SQZ																									
TAR		0,5				8,9																	2,7		
UC2																									
UUE														0,7											
ZI2			1,4																	0,9					
ZI6			1,4																	0,9					
ZIB		97,5	97,5	97,5		97,4						72,2			97,5			99,3	97,5		97,5	97,5	97,5		
ZID		82,0	82,0	81,7	90,5	81,1	90,9	90,5		90,5	90,5	54,4		90,5	81,7	82,3	90,5	86,0	82,2	91,3	81,7	81,7	82,3		
ZIE			99,5	99,5											80,1				99,5		99,5	99,5	99,5		
ZIP			2,3												22,2					0,9					
ZOO																									

Abbildung 3: Minderung der Erkennungsrate bei Standard-Kompression

3.2 Probleme und Auffälligkeiten

Beim Testen komprimierter Dateien ohne Dateiendung zeigte sich, dass einige Produkte Viren nicht erkennen, wenn die Archivdateien lediglich umbenannt wurden (betrifft AVG bei allen unterstützten Formaten sowie CMD und FPR bei „LHA"-Archiven). Bei den passwort-geschützten Archiven interessieren nicht die Erkennungsraten (die wie erwartet durchgängig bei 0% Erkennung liegen), sondern die Meldungen der Anti-Malware Produkte. Um eine realistische Einschätzung des Risikos zu ermöglichen, sollte zumindest ein Hinweis gegeben werden, dass diese Archivdateien nicht geprüft werden konnten (darüber hinaus ist auch eine Begründung wünschenswert, warum diese Dateien nicht geprüft werden konnten, etwa „password protected"). Ein erheblicher Anteil der getesteten Anti-Malware Produkte ist dazu aber nicht in der Lage und meldet jede dieser Dateien nur als „nicht infiziert" bzw. als „Ok" (betrifft ANT, AVG, GLA, IKA, INO, PER, PRO, RAV, SCN, VBR, VSP).

Zusätzlich zeigten sich technische Probleme. Viele Anti-Malware Produkte konnten Archivinhalte in der Log-Datei nicht vollständig benennen (ANT, BDF, FPR, GLA, INO, NAV, QHL, SCN). Diese Schwäche zeigte sich insbesondere bei Archiven, die Verzeichnisstrukturen enthielten sowie bei mehrfach rekursiv komprimierten Archiven. Bei einigen Produkten war dieses Verhalten generell zu beobachten (ANT, BDF, GLA, NAV).

Beim Überprüfen der rekursiv komprimierten Archive waren deutliche Stabilitätsprobleme festzustellen (FSE, NVC, PRO, SCN stürzten wiederholt ab). Von den wenigen Produkten die Archive im „HA"-Format unterstützen, benötigten einige sehr lange (in der Größenordnung mehrerer Stunden) zum Scannen eines „HA"-Archivs (betrifft RAV und AVK[10]).

4 Zusammenfassung

Durch Nichterkennung von maliziöser Software in komprimierter Form können Risiken entstehen, auch beim Einsatz von Anti-Malware Software im on-access Modus. Um eine Aussage über die Erkennungsgüte von Anti-Malware Produkten hinsichtlich komprimierter Malware machen zu können, wurde eine hierfür entwickelte Testmethodik vorgestellt.

Bei der Durchführung eines Tests von Anti-Malware Software mit der vorgestellten Methodik zeigt sich, dass erhebliche Schwächen bei fast allen getesteten Produkten bestehen. Insbesondere beunruhigen die Testergebnisse, bei denen ein Format von einem Produkt zwar unterstützt wird, jedoch eine Minderung der Erkennungsrate gemessen wurde. Zusätzlich wurden Mängel bei einem Großteil der getesteten Software bei der Meldung von nicht überprüften komprimierten Dateien festgestellt sowie diverse andere Schwachpunkte aufgezeigt (vgl. [AV04]).

[10] Da AVK intern u.a. die Scan-Engine von RAV verwendet, handelt es sich vermutlich um eine einzelne Schwäche dieser Scan-Engine.

Literaturverzeichnis

[AV04] anti Virus Test Center, Hompage, 2004, www.avtc.info

[Bi03] Bieringer, P.: bzip2 bomb vulnerability of antivirus decompression engines, 2003, http://www.aerasec.de/security/advisories/txt/bzip2bomb-antivirusengines.txt

[Br01] Brunnstein, K.: avtc code of conduct, 2001, ftp://agn-www.informatik.uni-hamburg.de/pub/CodeConduct/CoC-016.txt

[Br02] Brunnstein, K.: avtc test report 2002-12, 2002, http://agn-www.informatik.uni-hamburg.de/vtc/en0212.htm

[Br03] Brunnstein, K.: avtc test report 2003-04, Evaluation WinXP, 2003 ftp://agn-www.informatik.uni-hamburg.de/pub/texts/tests/pc-av/2003-04/7evalwxp.txt

[Si02] Siekierski, U.: methods and procedures for quality assessment of AntiMalware products, especially on access scanners, Diplomarbeit, Fachbereich Informatik, Universität Hamburg, 2002

[We01] Wells, J.: PC Viruses in the Wild – Sep/Oct 2001, http://www.wildlist.org/WildList/200110.htm

Author Index

GI-Edition Lecture Notes in Informatics

P-1 Gregor Engels, Andreas Oberweis, Albert Zündorf (Hrsg.): Modellierung 2001.

P-2 Mikhail Godlevsky, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications, ISTA'2001.

P-3 Ana M. Moreno, Reind P. van de Riet (Hrsg.): Applications of Natural Language to Information Systems, NLDB'2001.

P-4 H. Wörn, J. Mühling, C. Vahl, H.-P. Meinzer (Hrsg.): Rechner- und sensorgestützte Chirurgie; Workshop des SFB 414.

P-5 Andy Schürr (Hg.): OMER - Object-Oriented Modeling of Embedded Real-Time Systems.

P-6 Hans-Jürgen Appelrath, Rolf Beyer, Uwe Marquardt, Heinrich C. Mayr, Claudia Steinberger (Hrsg.): Unternehmen Hochschule, UH'2001.

P-7 Andy Evans, Robert France, Ana Moreira, Bernhard Rumpe (Hrsg.): Practical UML-Based Rigorous Development Methods - Countering or Integrating the extremists, pUML'2001.

P-8 Reinhard Keil-Slawik, Johannes Magenheim (Hrsg.): Informatikunterricht und Medienbildung, INFOS'2001.

P-9 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Innovative Anwendungen in Kommunikationsnetzen, 15. DFN Arbeitstagung.

P-10 Mirjam Minor, Steffen Staab (Hrsg.): 1st German Workshop on Experience Management: Sharing Experiences about the Sharing Experience.

P-11 Michael Weber, Frank Kargl (Hrsg.): Mobile Ad-Hoc Netzwerke, WMAN 2002.

P-12 Martin Glinz, Günther Müller-Luschnat (Hrsg.): Modellierung 2002.

P-13 Jan von Knop, Peter Schirmbacher and Viljan Mahnič (Hrsg.): The Changing Universities – The Role of Technology.

P-14 Robert Tolksdorf, Rainer Eckstein (Hrsg.): XML-Technologien für das Semantic Web – XSW 2002.

P-15 Hans-Bernd Bludau, Andreas Koop (Hrsg.): Mobile Computing in Medicine.

P-16 J. Felix Hampe, Gerhard Schwabe (Hrsg.): Mobile and Collaborative Business 2002.

P-17 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Zukunft der Netze –Die Verletzbarkeit meistern, 16. DFN Arbeitstagung.

P-18 Elmar J. Sinz, Markus Plaha (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2002.

P-19 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3.Okt. 2002 in Dortmund.

P-20 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3.Okt. 2002 in Dortmund (Ergänzungsband).

P-21 Jörg Desel, Mathias Weske (Hrsg.): Promise 2002: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen.

P-22 Sigrid Schubert, Johannes Magenheim, Peter Hubwieser, Torsten Brinda (Hrsg.): Forschungsbeiträge zur "Didaktik der Informatik" – Theorie, Praxis, Evaluation.

P-23 Thorsten Spitta, Jens Borchers, Harry M. Sneed (Hrsg.): Software Management 2002 - Fortschritt durch Beständigkeit

P-24 Rainer Eckstein, Robert Tolksdorf (Hrsg.): XMIDX 2003 – XML-Technologien für Middleware – Middleware für XML-Anwendungen

P-25 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Commerce – Anwendungen und Perspektiven – 3. Workshop Mobile Commerce, Universität Augsburg, 04.02.2003

P-26 Gerhard Weikum, Harald Schöning, Erhard Rahm (Hrsg.): BTW 2003: Datenbanksysteme für Business, Technologie und Web

P-27 Michael Kroll, Hans-Gerd Lipinski, Kay Melzer (Hrsg.): Mobiles Computing in der Medizin

P-28 Ulrich Reimer, Andreas Abecker, Steffen Staab, Gerd Stumme (Hrsg.): WM 2003: Professionelles Wissensmanagement - Erfahrungen und Visionen

P-29 Antje Düsterhöft, Bernhard Thalheim (Eds.): NLDB'2003: Natural Language Processing and Information Systems

P-30 Mikhail Godlevsky, Stephen Liddle, Heinrich C. Mayr (Eds.): Information Systems Technology and its Applications

P-31 Arslan Brömme, Christoph Busch (Eds.): BIOSIG 2003: Biometric and Electronic Signatures

P-32 Peter Hubwieser (Hrsg.): Informatische Fachkonzepte im Unterricht – INFOS 2003

P-33 Andreas Geyer-Schulz, Alfred Taudes (Hrsg.): Informationswirtschaft: Ein Sektor mit Zukunft

P-34 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 1)

P-35 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 2)

P-36 Rüdiger Grimm, Hubert B. Keller, Kai Rannenberg (Hrsg.): Informatik 2003 – Mit Sicherheit Informatik

P-37 Arndt Bode, Jörg Desel, Sabine Rathmayer, Martin Wessner (Hrsg.): DeLFI 2003: e-Learning Fachtagung Informatik

P-38 E.J. Sinz, M. Plaha, P. Neckel (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2003

P-39 Jens Nedon, Sandra Frings, Oliver Göbel (Hrsg.): IT-Incident Management & IT-Forensics – IMF 2003

P-40 Michael Rebstock (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2004

P-42 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Economy – Transaktionen und Prozesse, Anwendungen und Dienste

P-43 Birgitta König-Ries, Michael Klein, Philipp Obreiter (Hrsg.): Persistance, Scalability, Transactions – Database Mechanisms for Mobile Applications

P-44 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): Security, E-Learning. E-Services

P-45 Bernhard Rumpe, Wofgang Hesse (Hrsg.): Modellierung 2004

P-46 Ulrich Flegel, Michael Meier (Eds.): Detection of Intrusions and Malware & Vulnerability Assessment – DIMVA 2004

The titles can be purchased at:

Köllen Druck + Verlag GmbH
Ernst-Robert-Curtius-Str. 14
53117 Bonn
Fax: +49 (0)228/9898222
E-Mail: druckverlag@koellen.de