Selecting and Improving System Call Models for Anomaly Detection
(or, 30 minutes before CIPHER 5’s CTF results)

Alessandro Frossi    Federico Maggi    Gian Luigi Rizzo
Stefano Zanero

July 10, 2009
Topic of this talk
System Call Based Anomaly Detection

Detecting intrusions using system call flows w/ data models

Let's take a look at a simple, generic example.
System Call Based Anomaly Detection
Detecting intrusions using system call flows w/ data models

Let's take a look at a simple, generic example.
System Call Based Anomaly Detection

A set of models is created based on certain features of the system calls.
System Call Based Anomaly Detection

Models estimate feature values observed in “good” system calls

Example of models
- string length
- number of arguments
- function name
- string character distribution
System Call Based Anomaly Detection

Estimations become more accurate as more system calls are analyzed

Applications

Kernel

Models of good system calls

Models of good "behaviors"
System Call Based Anomaly Detection

Also, models based on sets of system calls can be constructed.
System Call Based Anomaly Detection

Knowledge about system calls’ context is learned

Applications

Kernel

Models of good system calls

Models of good "behaviors"
System Call Based Anomaly Detection

In detection mode, the same models can be used to spot malicious system calls...
System Call Based Anomaly Detection

...or malicious execution contexts

Training and detection may be more complex, but the basic idea is the same.
The systems we analyzed
Different Approaches: Deterministic vs. Stochastic

We analyzed two anomaly detectors based on different approaches.
Different Approaches: Deterministic vs. Stochastic

We analyzed two anomaly detectors based on different approaches

**FSA-DF [IEEE S&P 2006]**

- Deterministic
- Control-flow: FSA
- Data-flow: unary/binary relations
Different Approaches: Deterministic vs. Stochastic

We analyzed two anomaly detectors based on different approaches.

**FSA-DF [IEEE S&P 2006]**
- Deterministic
- Control-flow: FSA
- Data-flow: unary/binary relations

**S²A²DE [IEEE TODS 2009]**
- Stochastic
- Control-flow: Markov-chain
- Data models: clusters
Deterministic Data-flow Anomaly Detection

The system calls generated by each process are examined.

```plaintext
5866 8052110  unlink("/usr/local/var/proftpd/test.sock") = 0
5866 808ec46  capget(DONT_CARE, DONT_CARE) = 0
5866 80a0b7f  timer_gettime (DONT_CARE, DONT_CARE) = 3
```
Deterministic Data-flow Anomaly Detection

Different PCs means different process states

5866 8052110  unlink("/usr/local/var/proftpd/test.sock") = 0
5866 808ec46  capget(DONT_CARE, DONT_CARE) = 0
5866 80a0b7f  timer_gettime (DONT_CARE, DONT_CARE) = 3
Deterministic Data-flow Anomaly Detection

A system call changes the process’ state...

```
unlink("/usr/local/var/proftpd/test.sock") = 0

808ec46 capget(DONT_CARE, DONT_CARE) = 0

80a0b7f timer_gettime (DONT_CARE, DONT_CARE) = 3
```
Deterministic Data-flow Anomaly Detection

...and so forth

```
5866 8052110  unlink("/usr/local/var/proftpd/test.sock") = 0
5866  808ec46  capget(DONT_CARE, DONT_CARE) = 0
5866  80a0b7f  timer_gettime (DONT_CARE, DONT_CARE) = 3
```
Deterministic Data-flow Anomaly Detection

This analysis is repeated until termination

```
5866 8052110   unlink("/usr/local/var/proftpd/test.sock") = 0
5866 808ec46   capget(DONT_CARE, DONT_CARE) = 0
5866 80a0b7f   timer_gettime (DONT_CARE, DONT_CARE) = 3
```
Deterministic Data-flow Anomaly Detection

A network of unary/binary data-flow relations on top of the process’ FSA
Deterministic Data-flow Anomaly Detection

A network of unary/binary data-flow relations on top of the process’ FSA

\begin{align*}
FD_3 = & \text{open}(F_3, M_3) \\
M_3 & \text{equal } O \\
isDirectory F_8' & \Rightarrow F_8' \text{isWithinDir } F_6 \\
\text{isDirectory } F_8 & \Rightarrow F_8 \text{isWithinDir } I \\
FD_{11} = & \text{open}(F_{11}, M_{11}) \\
F_{11} & \text{equal } F_8 \\
\end{align*}
Deterministic Data-flow Anomaly Detection

Other types of relations

- **Unary**: capture properties of a single argument.
  - equal
  - elementOf
  - subsetOf
  - range
  - isWithinDir
  - hasExtension

- **Binary**: capture relations between two arguments.
  - equal
  - isWithinDir
  - contains
  - hasSameDirAs
  - hasSameBaseAs
  - hasSameExtensionAs
Major Drawback: False Positives

Mostly due to the deterministic relations
Major Drawback: False Positives
Mostly due to the deterministic relations

\[
\text{open}(\text{"/tmp/php1553"}, 0, 0x1b6) = 5
\]

unary \text{elementOf}(\{\text{/tmp/php1553, /tmp/php9022}\})

\[
\text{open}(\text{"/tmp/php9022"}, 0, 0x1b6) = 5
\]
Major Drawback: False Positives
Mostly due to the deterministic relations

\[
\text{open}("/tmp/php1553", 0, 0x1b6) = 5
\]

unary \text{elementOf} \{/tmp/php1553, /tmp/php9022\}

\[
\text{open}("/tmp/php9022", 0, 0x1b6) = 5
\]

What if "/tmp/php1990" is found?
Major Drawback: False Positives
Mostly due to the deterministic relations

\[
\text{open(“/tmp/php1553”, 0, 0x1b6) = 5}
\]

unary \textbf{elementOf}(\{/tmp/php1553, /tmp/php9022\})

\[
\text{open(“/tmp/php9022”, 0, 0x1b6) = 5}
\]

What if “/tmp/php1990” is found?
These false positives occur pretty often.
Stochastic Behavior Profiling of Processes
Clusters of similar system calls interconnected by Markov-chains
Stochastic Behavior Profiling of Processes

Clusters of similar system calls interconnected by Markov-chains

Finding similar system calls

— anomaly scores [ACM TISSEC 2006] based on certain features of the arguments

— distance metrics [ACM TODS 2009] used to cluster similar system calls

— each application's process creates different clusters
Stochastic Behavior Profiling of Processes
Clusters of similar system calls interconnected by Markov-chains
Stochastic Behavior Profiling of Processes
Clusters of similar system calls interconnected by Markov-chains
Stochastic Behavior Profiling of Processes

Clusters of similar system calls interconnected by Markov-chains

write1

open1

chmod1
Stochastic Behavior Profiling of Processes

Clusters of similar system calls interconnected by Markov-chains
Stochastic Behavior Profiling of Processes

Clusters of similar system calls interconnected by Markov-chains
Stochastic Behavior Profiling of Processes
Clusters of similar system calls interconnected by Markov-chains

Markov-chains encode process behavior
— transitions between system calls can occur with different probabilities [ACM TODS 2009]
— a call is anomalous if either there is no matching state (i.e., cluster) or transition probability is violated
Major Drawbacks: False Negatives
Mostly due to the stochastic nature of Markov-chains
Major Drawbacks: False Negatives
Mostly due to the stochastic nature of Markov-chains

Clustering

- clustering depends on configuration parameters
- different parameters $\rightarrow$ different results
Major Drawbacks: False Negatives
Mostly due to the stochastic nature of Markov-chains

Clustering
- clustering depends on configuration parameters
- different paramenters $\rightarrow$ different results

Markov-chains (example)
Major Drawbacks: False Negatives
Mostly due to the stochastic nature of Markov-chains

Clustering
- clustering depends on configuration parameters
- different parameters → different results

Markov-chains (example)

Threshold = 0.5 * 1 * 0.5 = 0.25
Major Drawbacks: False Negatives
Mostly due to the stochastic nature of Markov-chains

Clustering
- clustering depends on configuration parameters
- different parameters → different results

Markov-chains (example)

Threshold = 0.5 \times 1 \times 0.5 = 0.25
Major Drawbacks: False Negatives
Mostly due to the stochastic nature of Markov-chains

**Clustering**
- clustering depends on configuration parameters
- different parameters → different results

**Markov-chains (example)**

\[
\text{Threshold} = 0.5 \times (1 \times 0.5)^n \times 0.5 \rightarrow 0
\]
Major Drawbacks: False Negatives

Mostly due to the stochastic nature of Markov-chains

**Clustering**
- clustering depends on configuration parameters
- different parameters → different results

**Markov-chains (example)**

![Markov Chain Diagram]

Threshold = \(0.5 \times (1 \times 0.5)^n \times 0.5 \rightarrow 0\)

No valid threshold can be found if cycles are not of fixed length.
For instance, DoS attacks may not be detected.
Pros and cons of the two approaches
## Pros and cons of the two approaches

<table>
<thead>
<tr>
<th></th>
<th>FSA-DF</th>
<th>S(^2)A(^2)DE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FSA</strong></td>
<td>• Perfectly models a software behavior</td>
<td>• Introduces a statistical approach</td>
</tr>
<tr>
<td></td>
<td>• No False Negatives</td>
<td>• False Negatives</td>
</tr>
<tr>
<td></td>
<td>• Doesn’t allow deviations</td>
<td>• Few False Positives</td>
</tr>
<tr>
<td><strong>Relations</strong></td>
<td>• Deterministic approach</td>
<td>• Stochastic approach</td>
</tr>
<tr>
<td></td>
<td>• No new input adaptation</td>
<td>• Can adapt to new inputs</td>
</tr>
<tr>
<td></td>
<td>• Prone to False Positives</td>
<td>• Few False Positives</td>
</tr>
<tr>
<td></td>
<td>• No False Negatives</td>
<td>• False Negatives</td>
</tr>
<tr>
<td><strong>Control Flow</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Flow</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Clusters</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
First contribution:
combination of the two approaches
## Combining Complementary Approaches

Deterministic control-flow + stochastic data models

### Hybrid IDS

<table>
<thead>
<tr>
<th>FSA-DF</th>
<th>S$^2$A$^2$DE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FSA</strong></td>
<td><strong>Control Flow</strong></td>
</tr>
</tbody>
</table>
| • Perfectly models a software behavior  
  • No False Negatives  
  • Doesn’t allow deviations | **S$^2$A$^2$DE** | • Stochastic approach  
  • Can adapt to new inputs  
  • Few False Positives  
  • False Negatives |
Combining Complementary Approaches

The learning algorithm is similar to that used in FSA-DF

- \( \forall \text{couple} \langle \text{syscall}_{i-1}, \text{syscall}_i \rangle \in \{ \text{TrainingSet} \} \)
  - make state
  - learn relations
Combining Complementary Approaches

The learning algorithm is similar to that used in FSA-DF

- \( \forall \text{couple}(\text{syscall}_{i-1}, \text{syscall}_i) \in \{ \text{TrainingSet} \} \)
  - make state
  - learn relations
    - equal
    - elementOf
    - subsetOf
    - range
    - isWithinDir
    - hasExtension
    - isWithinDir
    - contains
    - hasSameDirAs
    - hasSameBaseAs
    - hasSameExtensionAs
Combining Complementary Approaches
The learning algorithm is similar to that used in FSA-DF

- \( \forall \text{couple} \langle \text{syscall}_{i-1}, \text{syscall}_i \rangle \in \{ \text{TrainingSet} \} \)
  - make state
  - learn relations
    - equal save model of similar strings
    - elementOf save model of similar strings
    - subsetOf
    - range
    - isWithinDir
    - hasExtension
    - isWithinDir
    - contains save model of similar strings
    - hasSameDirAs
    - hasSameBaseAs
    - hasSameExtensionAs
Combining Complementary Approaches
The learning algorithm is similar to that used in FSA-DF

- learn string domains
- \( \forall \text{couple} \langle \text{syscall}_{i-1}, \text{syscall}_i \rangle \in \{ \text{TrainingSet} \} \)
  - make state
  - learn relations
    - equal
    - elementOf
    - subsetOf
    - range
    - isWithinDir
    - hasExtension
    - isWithinDir
    - contains
    - hasSameDirAs
    - hasSameBaseAs
    - hasSameExtensionAs
Paths And Filenames

How to find groups of good strings into execve/open/read/... args?

/var/log/http.0 ... /etc/ftp.conf ... /tmp/php1231
... /var/run/nfsd.pid ... /etc/smb/samba.conf
... /opt/local/lib/libncurses.a ... /usr/lib/libkmod.a
... /tmp/uscreens/427.ttys000 ... /var/db/ntp.drift ...
Paths And Filenames
How to find groups of good strings into execve/open/read/... args?

/var/log/http.0 . . . /etc/ftp.conf . . . /tmp/php1231
 . . . /var/run/nfsd.pid . . . /etc/smb/samba.conf
 . . . /opt/local/lib/libncurses.a . . . /usr/lib/libkmod.a
 . . . /tmp/uscreens/427.ttys000 . . . /var/db/ntp.drift . . .

Self-Organizing Map
Type of artificial neural network, trained using unsupervised learning to
produce a multi dimensional discretized representation of the input space
of the training samples, called map.

Idea

▶ SOM to capture classes of good strings.
▶ Model of good strings → nodes.
▶ Similar strings → neighbor nodes.
Paths And Filenames
How to find groups of good strings into execve/open/read/... args?

/var/log/http.0 ... /etc/ftp.conf ... /tmp/php1231
... /var/run/nfsd.pid ... /etc/smb/samba.conf
... /opt/local/lib/libncurses.a ... /usr/lib/libkmod.a
... /tmp/uscreens/427.ttys000 ... /var/db/ntp.drift ...

Self-Organizing Map
Type of artificial neural network, trained using unsupervised learning to produce a multi dimensional discretized representation of the input space of the training samples, called map.

Idea
- SOM to capture classes of good strings.
- Model of good strings → nodes.
- Similar strings → neighbor nodes.
OK, they look pretty nice. But why SOMs?
OK, they look pretty nice. But why SOMs?
Paths And Filenames
Integration in Hybrid IDS - algorithm

▶ create SOM of all paths
  ▶ SOM initialization with linux directory structure.
  ▶ Extract all the paths from the syscalls
  ▶ SOM training [Kohonen 2004] with a randomized subset of the paths.

▶ \( \forall \text{couple} (\text{syscall}_{i-1}, \text{syscall}_i) \in \{ \text{TrainingSet} \} \)
  ▶ make state
  ▶ learn relations
    ▶ if \( \text{syscall}_{i-1} \) contains a path argument
      find BMU from the SOM
      add BMU to the edge
    ▶ subsetOf
    ▶ range
    ▶ isWithinDir
    ▶ hasExtension
    ▶ isWithinDir
    ▶ hasSameDirAs
    ▶ hasSameBaseAs
    ▶ hasSameExtensionAs
Second contribution:
improved system call models
Improved System Call Models

New models to reduce false detections

▶ Goal 1: Resilience to spurious strings in the datasets.
Goal 1: Resilience to spurious strings in the datasets.
- Long/short strings in the training data can bias interval based models.
Improved System Call Models
New models to reduce false detections

- **Goal 1: Resilience to spurious strings in the datasets.**
  - Long/short strings in the training data can bias interval based models.

- **Goal 2: Detect simple DoS attacks.**
Improved System Call Models
New models to reduce false detections

- **Goal 1: Resilience to spurious strings in the datasets.**
  - Long/short strings in the training data can bias interval based models.

- **Goal 2: Detect simple DoS attacks.**
  - i.e., process forced to execute the same code region until crash.
Argument Length Using Gaussian Intervals

Yields to less false positives
Argument Length Using Gaussian Intervals
Yields to less false positives

**Statistics:** to estimate the distribution of args length

- $|\text{args}| = X_{\text{args}} \sim \mathcal{N} (\mu, \sigma^2)$
- Sample Mean, Sample Variance.

**Model precision parameter:**

- Kurtosis $\hat{\gamma}_X = \frac{\hat{\mu}_X^4}{\hat{\sigma}_X^4} - 3$
- If $\gamma_{X_{\text{args}}} < 0$ the sample is spread on a big interval

**Anomaly threshold:** percentile $T_{\text{args}}$ centered on the mean.
create SOM of all paths

∀ couple⟨syscall_{i-1}, syscall_i⟩ ∈ \{ TrainingSet \}

- make state
- learn relations
  - save BMU
  - subsetOf
  - range: save string length or num. value
  - isWithinDir
  - hasExtension
  - isWithinDir
  - hasSameDirAs
  - hasSameBaseAs
  - hasSameExtensionAs
Mitigating DoS Using Edge Frequency Models

Yields to less false negatives
Mitigating DoS Using Edge Frequency Models

Yields to less false negatives

**Given that:**
- each FSA edge is traversed a variable number of times over multiple executions
- the traversal frequency has a range

**Idea:** estimate a validity interval to detect DoS attacks.
Edge Traversal Frequency

The Model

S1 → S3 → S2 → S1 → S3 → S2 → S1 → S3 → S2 → S1 → S3 → S2 → S1 → S3 → S2 → S1 → S3 → S2 → ... multiple executions ...
Edge Traversal Frequency

The Model

\[ \text{freqs} = X_{\text{freqs}} \sim \text{Beta}(\alpha, \beta) \]

- Estimated \( \alpha \) and \( \beta \)
- Interval from \( x \)-th percentile
- **Not** estimated if few values available
Mitigating DoS Using Edge Frequency Models
Integration in Hybrid IDS - algorithm

- create SOM of all paths
- $\forall$ couple \( syscall_{i-1}, syscall_i \in \{\ TrainingSet\ \} \)
  - make state
  - learn relations
    - save BMU
    - subsetOf
    - save string length or num. value
    - isWithinDir
    - hasExtension
    - isWithinDir
    - hasSameDirAs
    - hasSameBaseAs
    - hasSameExtensionAs
- save edge traverse count
How We Built The Evaluation Dataset

- "normal" execution of 5 tools (420509 syscalls)
- recent exploits from CVE (140415 syscalls)

<table>
<thead>
<tr>
<th>Normal Attacks</th>
<th>Training (420509 syscalls)</th>
<th>Detection (140415 syscalls)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sing</td>
<td>(1528)</td>
<td></td>
</tr>
<tr>
<td>mt-daapd</td>
<td>(9832)</td>
<td></td>
</tr>
<tr>
<td>proftpd</td>
<td>(18114)</td>
<td></td>
</tr>
<tr>
<td>sudo</td>
<td>(3157)</td>
<td></td>
</tr>
<tr>
<td>bitchx</td>
<td>(107784)</td>
<td></td>
</tr>
</tbody>
</table>
How We Built The Evaluation Dataset

- “normal” execution of 5 tools (420509 syscalls)
- recent exploits from CVE (140415 syscalls)
How We Built The Evaluation Dataset

- “normal” execution of 5 tools (420509 syscalls)
- recent exploits from CVE (140415 syscalls)
Accuracy Evaluation

No false negatives (deterministic control-flow) + almost-zero false positives (stochastic data models)

<table>
<thead>
<tr>
<th></th>
<th>sing</th>
<th>mt-daapd</th>
<th>proftpd</th>
<th>sudo</th>
<th>BitchX</th>
<th>mcweject</th>
<th>bsdtar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traces</td>
<td>22</td>
<td>18</td>
<td>21</td>
<td>22</td>
<td>15</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>Syscalls</td>
<td>1528</td>
<td>9832</td>
<td>18114</td>
<td>3157</td>
<td>107784</td>
<td>75</td>
<td>102</td>
</tr>
<tr>
<td>$S^2A^2$DE</td>
<td>10.0%</td>
<td>0%</td>
<td>0%</td>
<td>10.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>8.7%</td>
</tr>
<tr>
<td>FSA-DS</td>
<td>5.0%</td>
<td>16.7%</td>
<td>28%</td>
<td>15.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Hybrid IDS</td>
<td>0.0%</td>
<td>0%</td>
<td>0%</td>
<td>10.0%</td>
<td>0.0%</td>
<td>S$^2$A$^2$DE</td>
<td>SOM-S$^2$A$^2$DE</td>
</tr>
</tbody>
</table>

Table 2. Comparison of the FPR of $S^2A^2$DE vs. FSA-DF vs. Hybrid IDS and $S^2A^2$DE vs. SOM-$S^2A^2$DE. Values include the number of traces used. Accurate description of the impact of each individual model is in Section 4.2 (first five columns) and 4.3 (last two columns).
Accuracy Evaluation
No false negatives (deterministic control-flow) + almost-zero false positives (stochastic data models)

<table>
<thead>
<tr>
<th></th>
<th>sing</th>
<th>mt-daapd</th>
<th>profdtpd</th>
<th>sudo</th>
<th>BitchX</th>
<th>mcweject</th>
<th>bsdtar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traces</td>
<td>22</td>
<td>18</td>
<td>21</td>
<td>22</td>
<td>15</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>Syscalls</td>
<td>1528</td>
<td>9832</td>
<td>18114</td>
<td>3157</td>
<td>107784</td>
<td>75</td>
<td>102</td>
</tr>
<tr>
<td>$S^2A^2DE$</td>
<td>10.0%</td>
<td>0%</td>
<td>0%</td>
<td>10.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>8.7%</td>
</tr>
<tr>
<td>FSA-DS</td>
<td>5.0%</td>
<td>16.7%</td>
<td>28%</td>
<td>15.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Hybrid IDS</td>
<td>0.0%</td>
<td>0%</td>
<td>0%</td>
<td>10.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

Table 2. Comparison of the FPR of $S^2A^2DE$ vs. FSA-DF vs. Hybrid IDS and $S^2A^2DE$ vs. SOM-$S^2A^2DE$. Values include the number of traces used. Accurate description of the impact of each individual model is in Section 4.2 (first five columns) and 4.3 (last two columns).

- sing - write on arbitrary file (data-flow).
- proftpd - arbitrary command execution (data-/control-flow).
- sudo - arbitrary command execution (control-flow).
- bitchx - arbitrary code execution (control-flow + DoS).
Performance Evaluation

Not-so-negligible overhead, but mostly due to ptrace

<table>
<thead>
<tr>
<th>System calls</th>
<th>sing</th>
<th>sudo</th>
<th>BitchX</th>
<th>mcweject</th>
<th>bsdtar</th>
<th>Avg. speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>S^2A^2DE</td>
<td>0.4</td>
<td>0.8</td>
<td>1.9</td>
<td>0.1</td>
<td>0.1</td>
<td>8463</td>
</tr>
<tr>
<td>FSA-DF</td>
<td>1.3</td>
<td>1.5</td>
<td>1.2</td>
<td>-</td>
<td>-</td>
<td>7713</td>
</tr>
<tr>
<td>Hybrid IDS</td>
<td>29</td>
<td>5.8</td>
<td>27.7</td>
<td>-</td>
<td>-</td>
<td>1067</td>
</tr>
<tr>
<td>SOM-S^2A^2DE</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8.8</td>
<td>19</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 3. Detection performance measured in “seconds per system call”. The average speed is measured in system calls per second (last column).
Conclusions and Future Works

Solve performance issues due to SOMs

- **deterministic** models accurately capture the control-flow
- **stochastic** models accurately capture data-flow features
- a **hybrid** approach lowers false detections
- performance issues:
  - the optimization of BMUs lookup is the first item on our TODO list
  - the use of a faster system call interceptor the second one ;)
