*d*Anubis

Dynamic Device Driver Analysis
based on
Virtual Machine Introspection

Matthias Neugschwandtner
Paolo Milani Comparetti
Christian Platzer
Ulrich Bayer

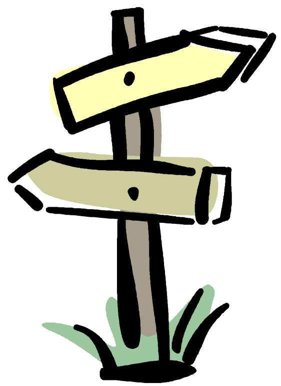
International Secure Systems Lab
Vienna University of Technology

Motivation

- Enormous number of new malware samples each day requires automated analysis
- Malware needs kernel-mode privileges to provide powerful functionality (e.g. Rootkits)
  - Stealth
  - Information gathering
- Aspect of device driver behavior has received less attention

Outline

- Overview
- *d*Anubis
- Evaluation
- Conclusion

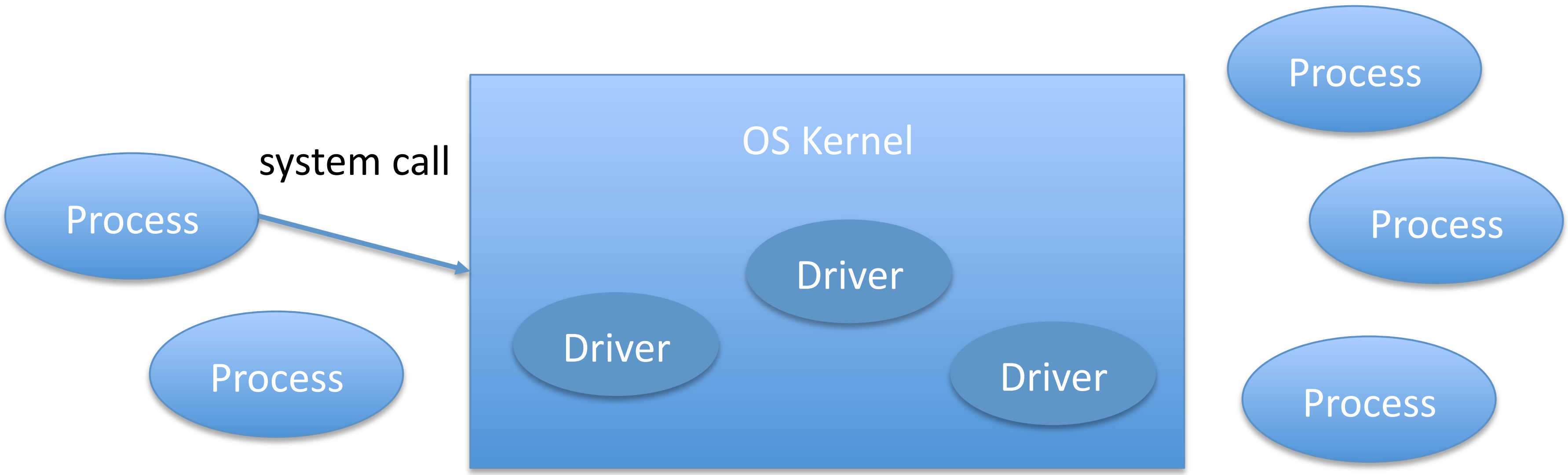


Our Approach

- Run malware in an emulated environment
- Monitor and evaluate analysis events
  - Executed code
  - Manipulated memory
- Reconstruct high-level semantics
- Generate human-readable analysis report

Process vs. Driver

	Process	Driver
Execution context	separate for each process	same as kernel
View on memory	unique page table directory	kernel memory space
Functionality	limited through well-defined system call interface	unlimited possibilities



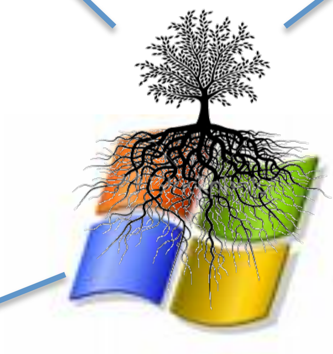
Rootkit: Goals

Inject code into the kernel

- Use a kernel exploit
- Load a device driver

Communicate with user mode application

- Use device communication



Provide stealth

Gather sensitive information

- Reroute kernel control flow
- Tamper with kernel data structures

Scenario: Process hiding

Application issues system call to list processes

Kernel system call dispatcher looks up system call in call table

replace entry call table hook

Kernel calls NtEnumerateProcess

apply runtime patch

direct kernel object manipulation

NtEnumerateProcess retrieves information from double-linked kernel process list

Information is returned to application

Scenario: File hiding

Application issues system call to list files in a directory

Kernel system call dispatcher looks up system call in call table

Kernel calls NtQueryDirectoryFile

NtQueryDirectoryFile requests information from the disk device

tamper with the device communication

Information is returned to application

Overview

- Dynamic analysis of Windows device drivers
- Virtual machine introspection using Qemu

- Derive high-level semantics of observed analysis events
- Provide driver context information for observed analysis events

- Perform first large scale study of kernel malware behavior

*d*Anubis

*d*Anubis

- Detect introduction of unknown device drivers
- Keep track of driver state
- Analyse
  - Generic behavioral aspects (e.g. called kernel functions)
  - Known typical Rootkit behavior (e.g. system call table hooks)
  - Licit OS - device driver interaction (e.g. device communication)

Challenges

- Bridging the semantic gap
  - Loss of semantic information when looking at memory from outside
  - Reestablish information by guest-view casting
  - Obtain necessary information from debugging symbols
- Tracking and attributing kernel-mode events
  - Code runs in arbitrary context
  - Identify event origin based on current program counter
  - Exact location of driver codebase in memory has to be known

Device Driver Analysis

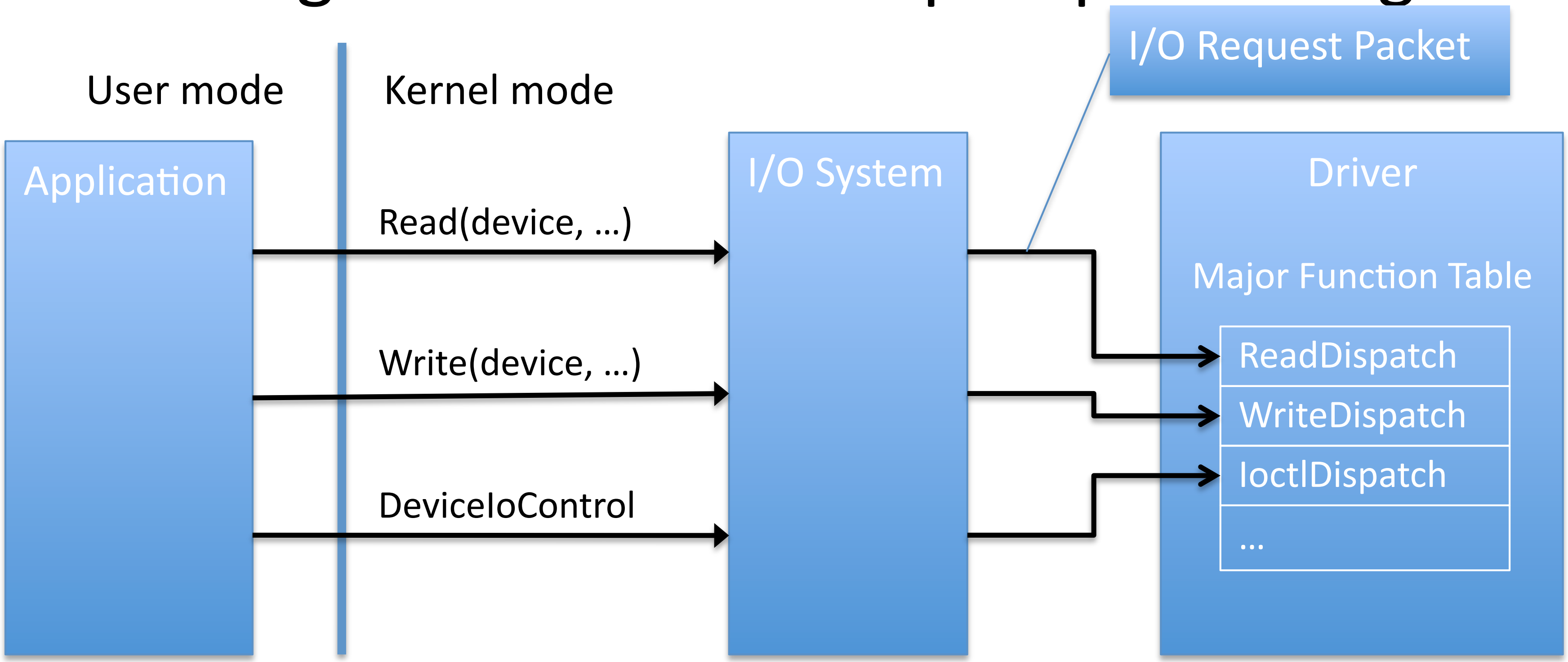
- Intercept lowest-level loading mechanisms
  - Get codebase location
  - Get offsets of the driver's exported functions
- Perform state-tracking
  - Assign analysis events to a driver's context

Device Driver Analysis

- Monitor driver activity
  - Record calls to exported Windows kernel functions
  - Taint string occurrences in the driver image
- Monitor driver communication
  - Creation of devices and attaching to driver stacks
  - Intercepting IRP traffic

Device Driver Interaction

- Devices as communication endpoints
- Stacking of drivers for complex processing



Memory Analysis

- Put certain memory regions under supervision
  - Kernel objects for process and driver bookkeeping
  - System call table
  - Kernel module codebases
- Track down and evaluate manipulations
  - Targets of call table manipulations
  - Consequences of kernel object manipulation
  - Detour patches of existing kernel code in memory
    - Determine which kernel function has been patched

Stimulation

- Rootkit functionality depends on external stimuli, e.g.
  - Keystrokes for keylogging
  - Process enumeration for process-hiding
- Stimulator component in the VM that repeatedly issues API calls

Evaluation

- Small evaluation with samples from www.rootkit.com to verify functionality
- Large-scale study
  - 64733 samples analyzed by Anubis in August 2009
  - 463 of these called NtLoadDeviceDriver

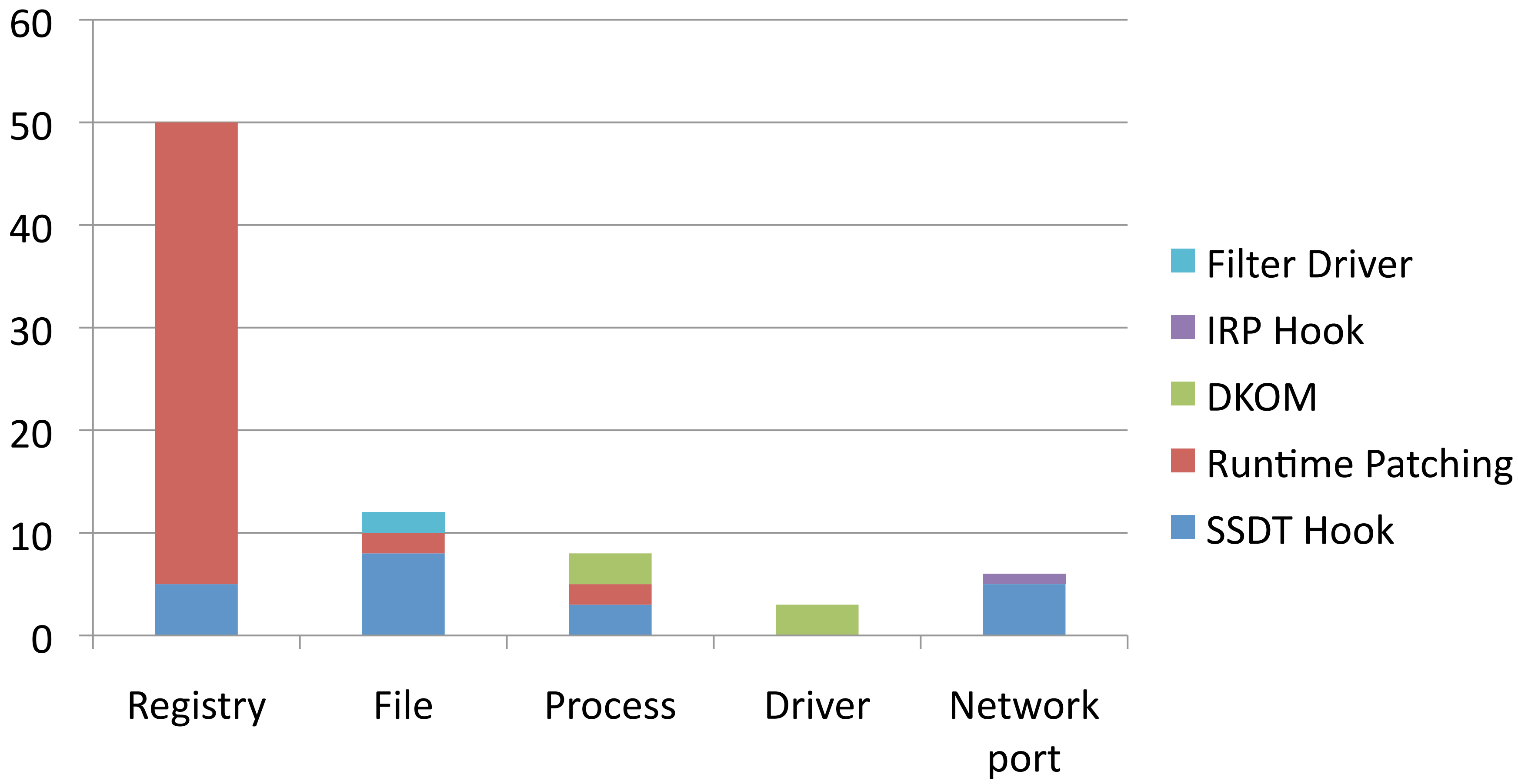
Results: High-level Activity

Driver activity	Number of samples exhibiting behavior
Device driver loaded	463
Windows kernel functions used	360
Windows device IO used	339
Strings accessed	300
Kernel code patched	76
Kernel call tables manipulated	37
MDL allocated	34
Kernel object manipulated	3

Results: Device Activity

Device activity	Number of samples
Device created	339
Driver’s device accessed from user mode	110
Driver‘s device invoked from user mode	86
Strings detected during communication	24
Attaches to device stack	2
Registers completion routine	2

Results: Stealth



Example Report A

Driver name	syssrv	
Created devices	\Device\MyDriver	
Rootkit activity	NtEnumerateKey	SSDT Hook
	NtQueryDirectoryFile	SSDT Hook
	svchost.exe	DKOM process hidden
Invoked major functions	CREATE	5x from user mode
	DEVICE_CONTROL	5x from user mode
	CLOSE	5x from kernel mode
Used strings	\WINDOWS\system32\mssrv32.exe	in DEVICE_CONTROL IRP
	\SOFTWARE\... \CurrentVersion\Run \mssrv32	in DEVICE_CONTROL IRP
Used kernel functions	ObReferenceObjectByName	during DEVICE_CONTROL
	PsLookupProcessByProcessID	during DEVICE_CONTROL
	NtEnumerateKey	during NtEnumerateKey hook
	wcslen, wcspz, wcscat	during NtEnumerateKey hook
	NtQueryDirectoryFile	during NtQueryDirectoryFile hook

Example Report B

Driver name	FILEMON701	
Created devices	\Device\Filemon701	
	two unnamed devices	
Attached to devices	sr	
	MRxSMB	
Completion routine	QUERY_VOLUME_INFORMATION for device sr	
Invoked major functions	CREATE	from user mode
	QUERY_VOLUME_INFORMATION	from kernel mode
	READ	from kernel mode
	CLEANUP, CLOSE	from kernel mode
	FastIoDeviceControl	
Used kernel functions	IoCreateDevice	during entry
	IoGetCurrentProcess	during entry
	IoCreateDevice	during FastIoControl
	IoAttachDeviceByPointer	during FastIoControl

Conclusions and Outlook

- *d*Anubis can provide a substantial amount of information on kernel-side malware
- Large scale analysis has given interesting first insight in the kernel-side malware landscape