Dragon Star Summer School (III): Malware Analysis and Defense

Dawn Song

Computer Science Dept.

UC Berkeley

In-depth Malware Analysis

Given a piece of suspicious code sample,

What malicious behaviors will it have?

How to classify it?

- Key logger, BHO Spyware, Backdoor, Rootkit

What mechanisms does it use?

- How does it steal information?
- How does it hook?

Who does it communicate with? Where does it send information to?

Does its communication exhibit certain patterns?

Does it contain trigger-based behavior?

- Time bombs
- Botnet commands

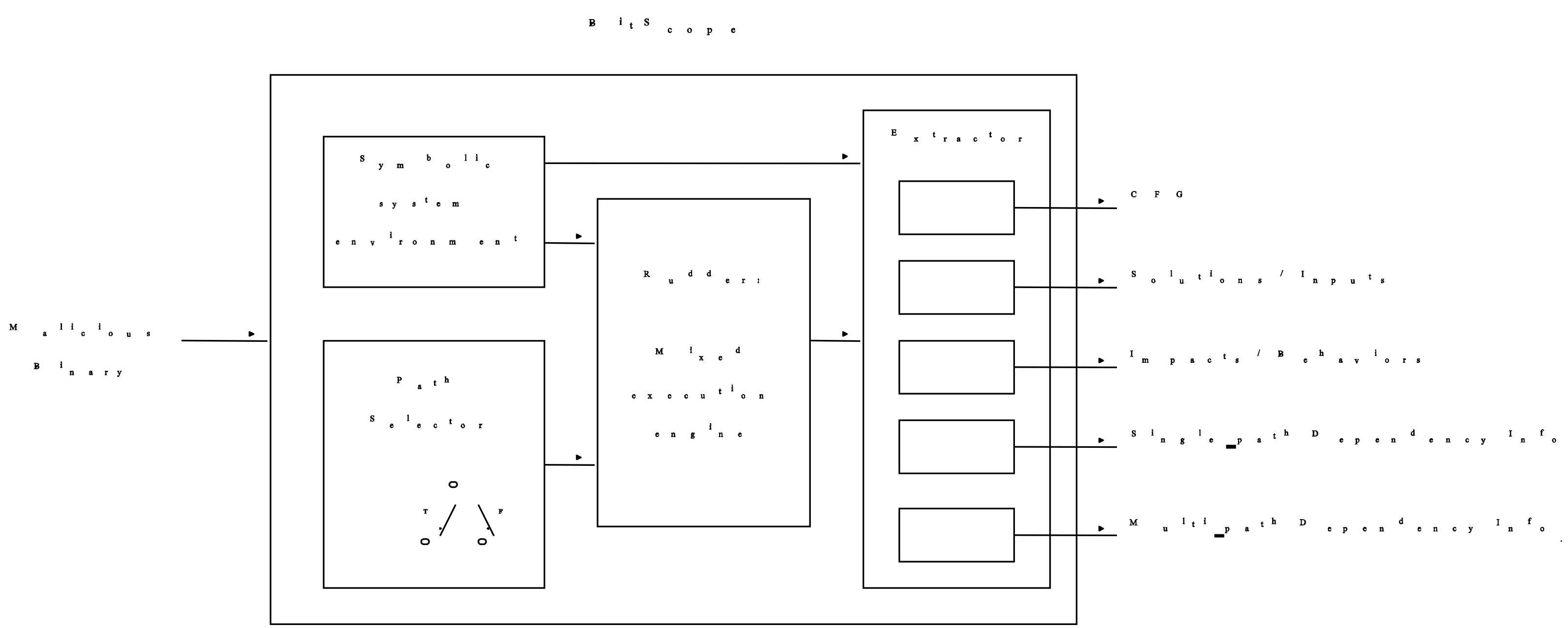
Can we design & develop a unified framework to answer these questions?

Our Approach: BitScope

Whole system dynamic binary instrumentation

Symbolic system environment introduces symbolic variables dynamically

Layered, panoptic symbolic execution



Illustrating Example (I): Privacy-Breaching Malware Detection

Privacy-breaching Malware

Spyware/adware, keyloggers, password thieves, network sniffers, backdoors, rootkits, etc

Creep into users' computers

Collect private information

Compromise system

Even software from reputable vendors

Google Desktop: spyware-like behavior in certain settings

Sony DRM player: contains Rootkit component

Key Observation

Intrinsic characteristics: abnormal information access and processing behavior:

They access, leak, or tamper with sensitive information

Examples:

Browser-based spyware: leak users' surfing habits

Keylogger: record users' keystrokes

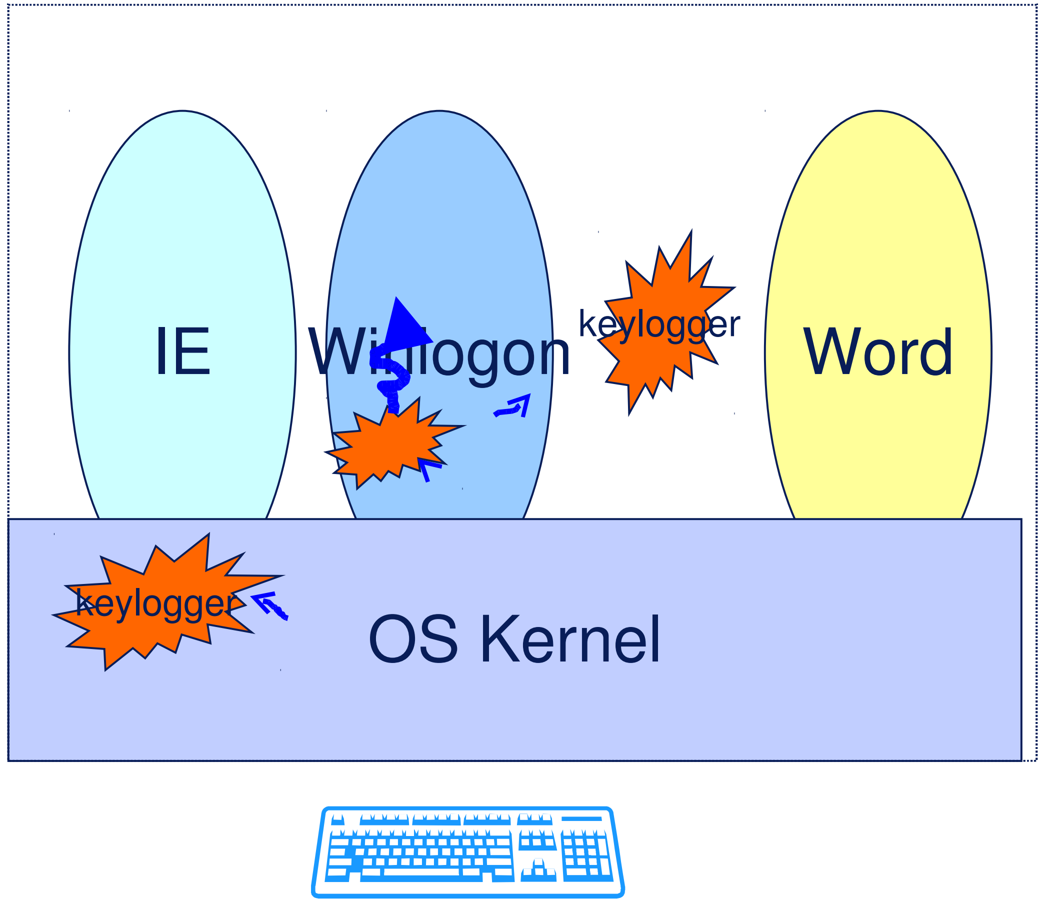
Password thief: steal users' passwords

Network sniffer: eavesdrop network traffic

Stealth backdoor: intercept network stack to establish a stealthy communication channel

Rootkit: tamper with critical system states

Key logger Example



tter what different manifestation keylogger takes, one invariant
mal information access and processing behavior.

Our Approach

Mark sensitive inputs as tainted

Monitor program execution to see how sensitive information flows

Fine grained (at instruction-level)

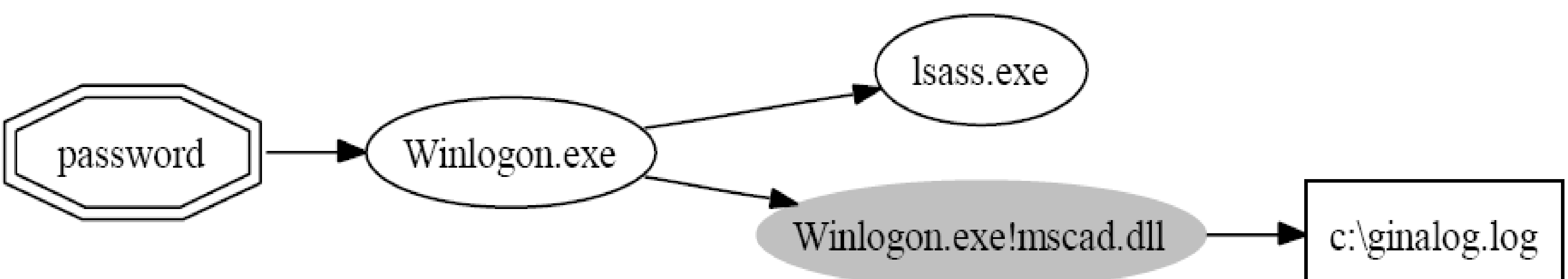
Whole system (including the kernel)

OS semantics aware

Obtain Taint graph

Dependency graph between taint sources and OS-level objects

- Taint sources: text, password, URL, ICMP, UDP, HTTP, document



Illustrating Example (II): Hook Detection

Malware needs to place hooks to achieve its malicious intents:

Rootkits: intercept and tamper with critical system states

Network sniffers: eavesdrop on incoming network traffic

Stealth backdoors: intercept network stack to establish stealthy communication channels

Spyware, keyloggers and password thieves, etc.

Previous work only detects known hooks

Challenges

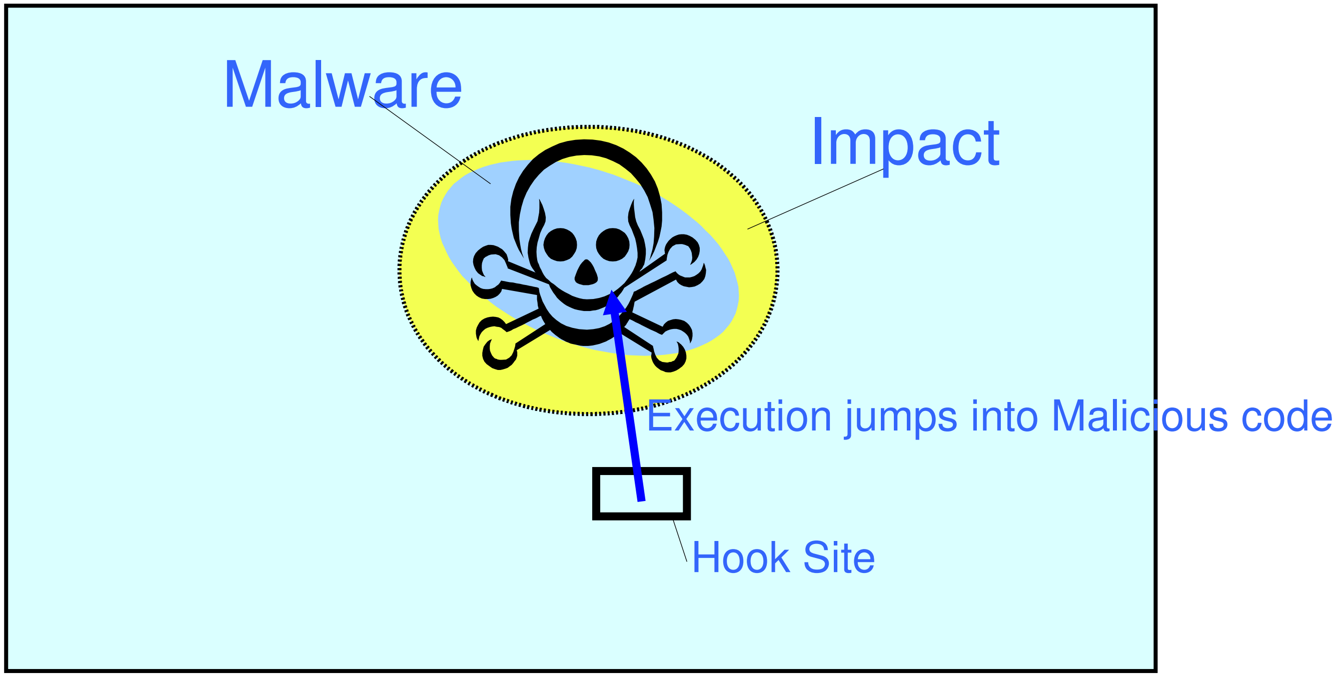
New hooks by malware

Different hooking mechanisms: code hooks & data hooks

Key Observation

A hook is an impact (*i.e., writes*) to the system by malware

This impact redirects the execution into the malicious code



Detect and analyze hooks by marking and tracking impacts

Our Approach

Hook Detection: Fine-grained Impact Analysis

Mark initial impacts (memory & register writes)

- By malware's module
- By malware's external function calls
- By malware's dynamically generated code

Track impacts propagation (and generate Impact Trace)

Detect when a hook is used

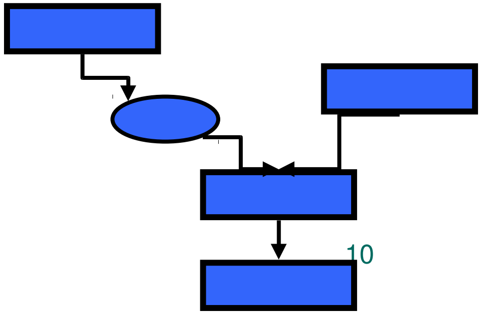
- Condition 1: Program counter (i.e, EIP in x86) is marked
- Condition 2: The execution jumps into the malicious code

Hook Analysis: Semantics-aware Impact Dependency Analysis

Backward data dependency analysis on Impact Trace

Combine OS-level semantics information

Generate a dependency graph: Hook Graph



Detecting Hooks in Sony Rootkit

Syntax: op src, dst

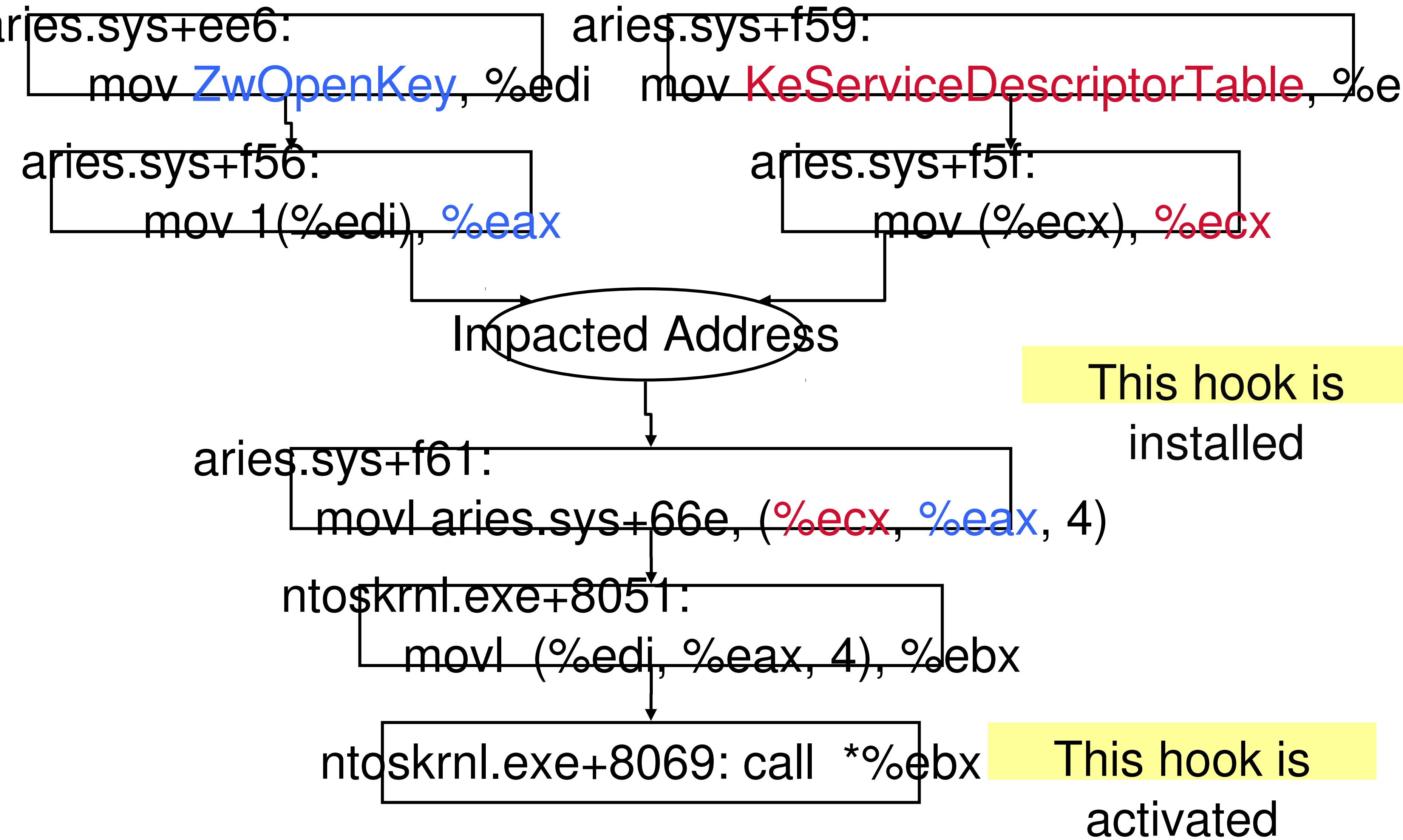
In Malicious Code

```
...
...
aries.sys+ee6:   mov ZwOpenKey, %edi
...
aries.sys+f56:   mov 1(%edi), %eax
aries.sys+f59:   mov KeServiceDescriptorTable, %ecx
aries.sys+f5f:   mov (%ecx), %ecx
aries.sys+f61:   movl aries.sys+66e, (%ecx, %eax, 4)
...
...
ntoskrnl.exe+8051: movl (%edi, %eax, 4), %ebx
ntoskrnl.exe+8069: call *%ebx
...
...
```

A hook is detected:

1) EIP is marked
2) The execution is redirected into aries.sys

Hook Graph for Sony Rootkit



Illustrating Example (III): Symbolic Execution to Detect Trigger-based Behavior

Trigger-based behaviors

Certain registry key set

Certain file exists

Mutex

Network connection

Time bomb

Commands in bot programs

Approach I: testing

Set up different environments

Test scripts simulate different system events

Challenge: difficult to satisfy trigger condition

TimeBomb Example

```
…
SystemTime time;
GetLocalTime(&time);

if (time.wMonth == 5 &&
    time.wDay   == 8) {
  DDoS();
} else {
  exit();
}
```

Our Approach

Return symbolic variable for malware's read from system environment

Symbolically execute instructions on symbolic variables

Compute path predicate for symbolic branches

Trigger conditions

Use solver to construct input satisfying path predicate

Trigger inputs

```
…
SystemTime time;
GetLocalTime(&time);

if (time.wMonth == 5 &&
    time.wDay == 8) {
  DDoS();
} else {
```

Symbolic variable

Symbolic branches

Path predicate:
time.wMonth == 5 &&
time.wDay == 8

BitScope: Unified Framework for In-depth Malware Analysis

Whole system dynamic binary instrumentation

Symbolic system environment introduces symbolic variables dynamically

Different types of symbols

- Taint symbol
- Dependency symbol
- Control symbol

Introduce symbols

- Keyboard, network inputs
- Memory read
- Function call return

Layered, panoptic symbolic execution

Taint symbol: keep track propagation chain

Dependency symbol: keep track symbolic formula for dependency

Control symbol: explore symbolic branches

BitScope: Extensible Architecture

User-defined symbolic system environment

What system inputs to make symbolic & what type of symbols

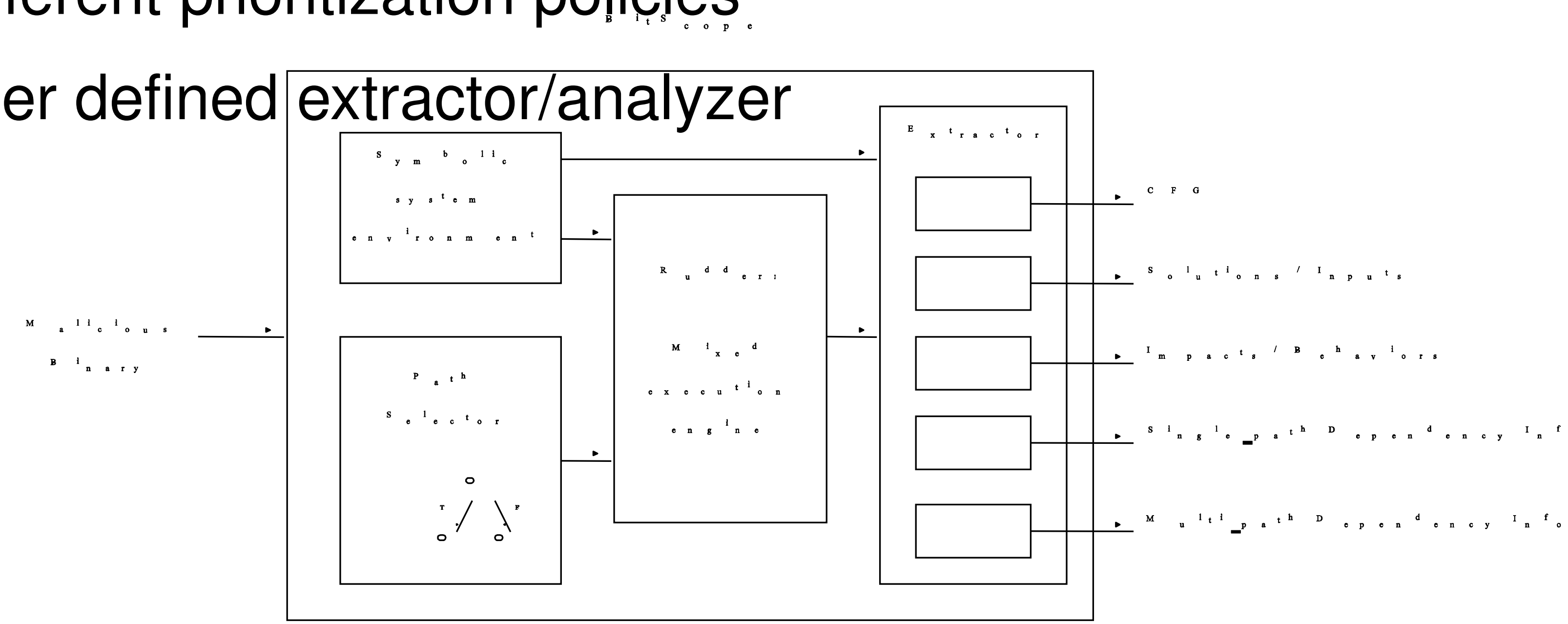
User-defined symbolic execution engine

How to perform symbolic execution for different types of symbols

User-defined path selector

Different prioritization policies

User defined extractor/analyzer



Extractors/Analyzers

Universal unpacker (Renovo): CFG

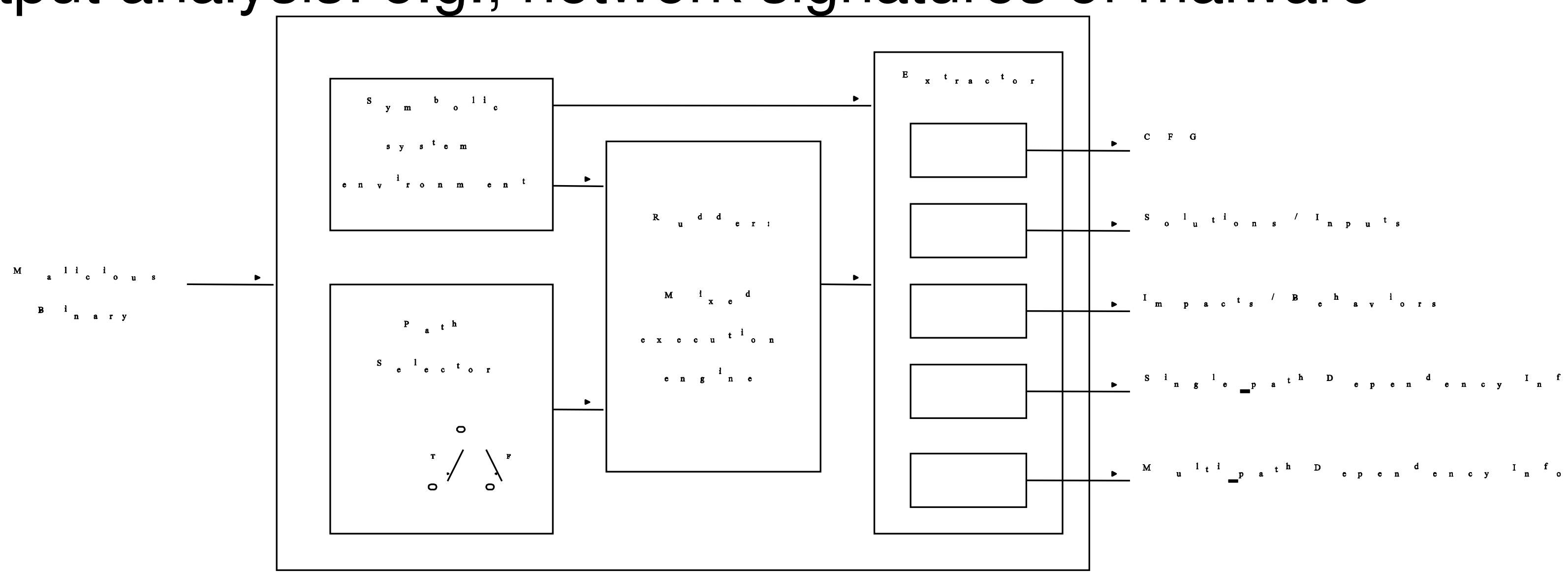
Privacy-breaching malware detector/analyzer (Panorama):

Whole system sensitive information flow

Hooking behavior analysis (HookFinder)

Trigger conditions & inputs

Output analysis: e.g., network signatures of malware



Experiment Results: Privacy-breach Malware

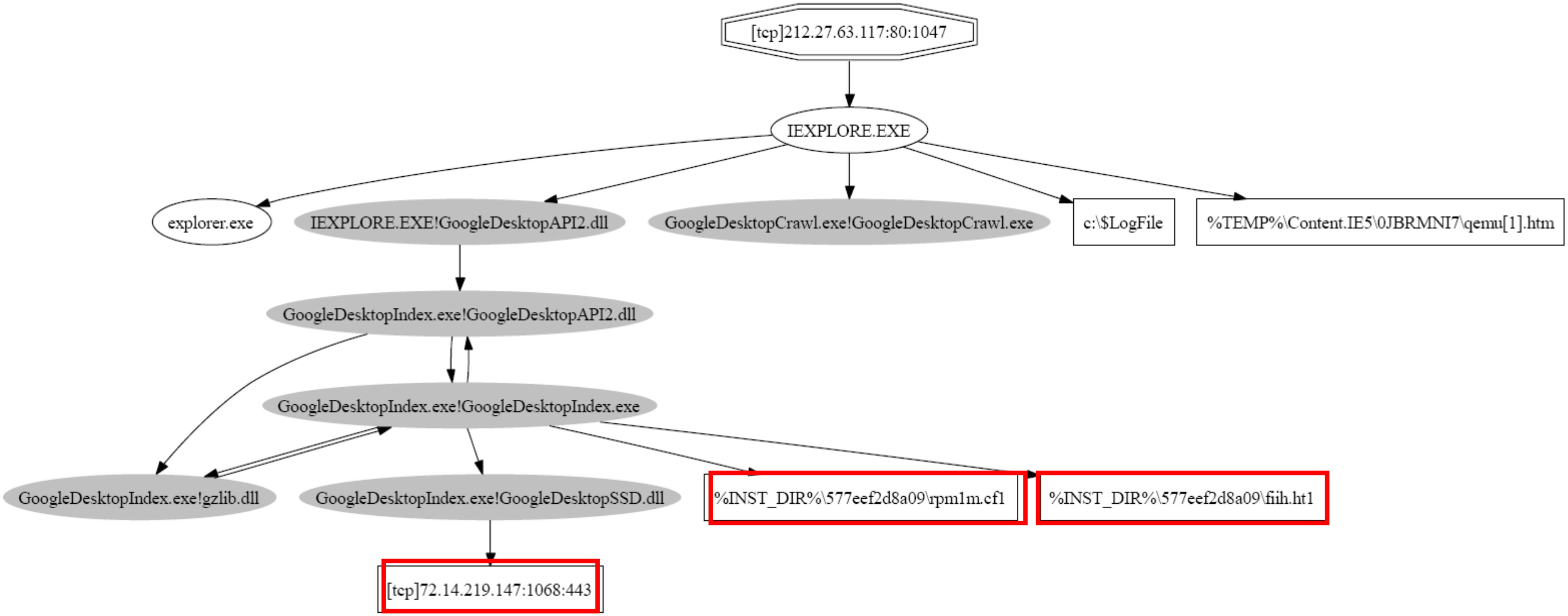
Category	Total	FNs	FPs
Keyloggers	5	0	-
Password thieves	2	0	-
Network sniffers	2	0	-
Stealth backdoors	3	0	-
Spyware/adware	22	0	-
Rootkits	8	0	-
Browser plugins	16	-	1
Multi-media	9	-	0
Security	10	-	2
System utilities	9	-	0
Office productivity	4	-	0
Games	4	-	0
Others	4	-	0
Sum	98	0	3

Browser Accelerator

Personal Firewall

Panorama correctly captures their information access and processing behaviors

Taint-graph for Google Desktop



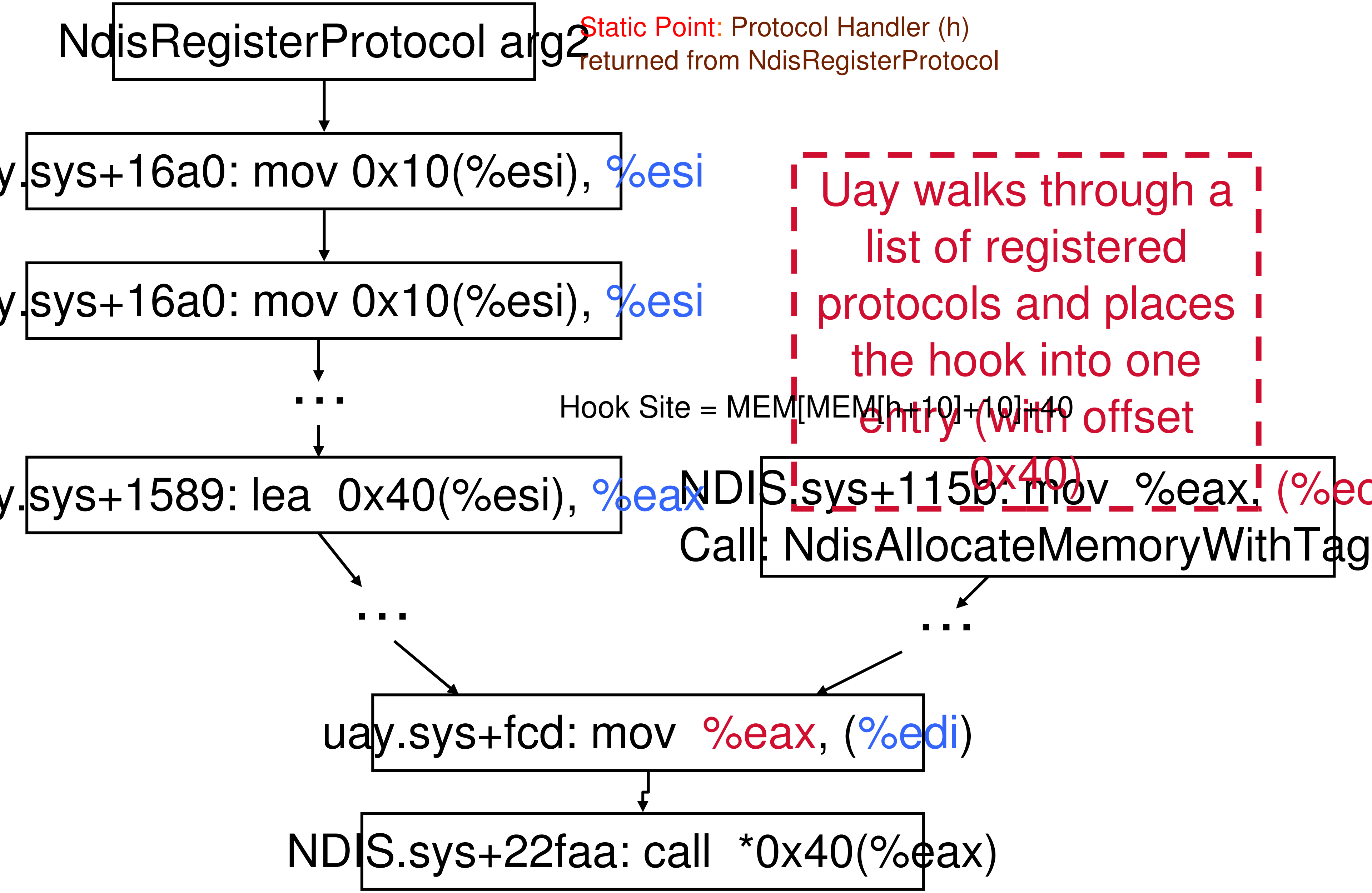
Google Desktop obtains incoming HTTP traffic, saves it into two index files, and then sends it out through an HTTPS connection, to a remote Google Server

Experiment Results: Hook Detection

Sample	Category	Runtime		Impact Trace	Hooks	
		Online	Offline		Total	Malicious
Troj/Keylogg-LF	Keylogger	6min	9min	3.7G	2	1
Troj/Thief	Password Thief	4min	<1min	143M	1	1
AFXRootkit	Rootkit	6min	33min	14G	4	3
CFSD	Rootkit	4min	2min	2.8G	5	4
Sony Rootkit	Rootkit	4min	<1min	25M	4	4
Vanquish	Rootkit	6min	12min	4.4G	11	11
Hacker Defender	Rootkit	5min	27min	7.4G	4	1
Uay Backdoor	Backdoor	4min	<1min	117M	5	2

Legitimate hooks: PsCreateSystemThread, CreateThread, CreateRemoteThread, StartServiceDispatcher

HookGraph of Uay



Experiment Results: Trigger-based Behavior Detection

Timebomb:

Blaster only sends SYN Flood during certain time

CodeRed only sends out exploits to propagate during certain time

Botnet commands:

SDBot is an extremely common IRC bot

- Unaided execution observes file copying, registry modification, and detection of internet access
- Symbolic execution discovered:
  - Input message format
  - 9 IRC commands
  - 72 bot commands

Trigger-based Behavior Detection

	Runtime	Behaviors Discovered	
		BitScope	Normal Env.
Trin00	569s	45	10
TFN2K	212s	39	16
SDBot 04b	~2hr	115	46
evilbot	127s	44	22
sdbot 2311	383s	234	66
ircbot 0045	186s	86	81
ircbot 004d	181s	93	58
q8bot	120s	53	25

Contact

http://bitblaze.cs.berkeley.edu

dawnsong@cs.berkeley.edu

BitBlaze team:
David Brumley, Juan Caballero, Ivan Jager, Cody Hartwig, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, Prateek Saxena, Heng Yin