Advanced Systems Security: Malware Detection

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University

Malware

- Attack code supplied by an adversary

Malware

- Attack code supplied by an adversary
  - What do you think of when you hear "malware"?

Example: Sirefef

PENNSTATE 1855

- Windows malware - Trojan to install rootkit
  - Technical details (see Microsoft)
  - And http://antivirus.about.com/od/virusdescriptions/a/What-Is-Sirefef-Malware.htm
- Attack: "Sirefef gives attackers full access to your system"
  - Runs as a Trojan software update (GoogleUpdate)
  - Runs on each boot by setting a Windows registry entry
  - Some versions replace device drivers
- Downloads code to run a P2P communication
  - Steal software keys and crack password for software piracy
  - Downloads other files to propagate the attack to other computers

Example: Sirefef

- Windows malware - Trojan to install rootkit
  - Technical details (see Microsoft)
  - http://antivirus.about.com/od/virusdescriptions/a/What-Is-Sirefef-Malware.htm
- Stealth: "while using stealth techniques in order to hide its presence"
  - "altering the internal processes of an operating system so that your antivirus and anti-spyware can't detect it."
    - Disable: Windows firewall, Windows defender
    - Changes: Browser settings
    - Join bot
- Microsoft: *"This list is incomplete"*

Malware

- Attack code supplied by an adversary
  - In ROP, an adversary may use existing code maliciously

Malware

- Attack code supplied by an adversary
  - How do we detect that a program contains malware?

Malware

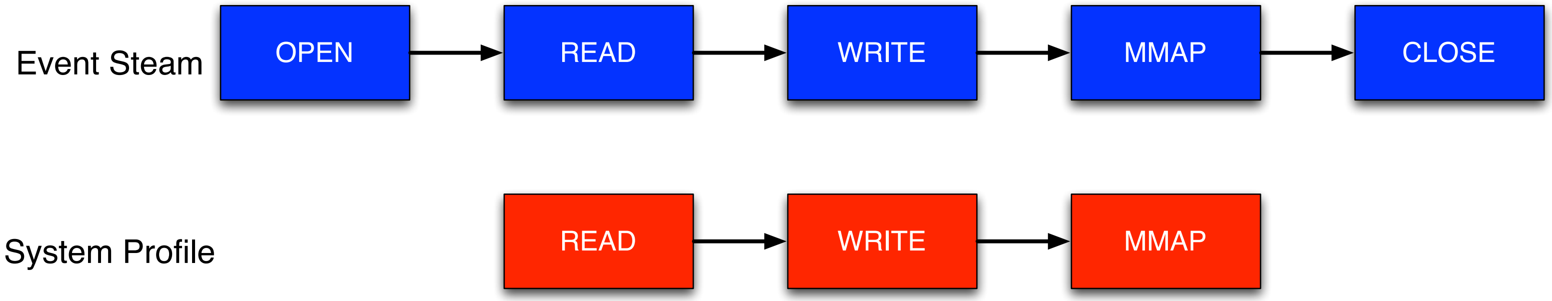
- Attack code supplied by an adversary
  - How do we detect that a program contains malware?
    - Two broad methods…
      - Anomaly and Misuse Detection

Anomaly Detection

- Detect that a program performs "anomalous" behavior
  - Out of the expected behavior for that program
  - How do we know what the "expected behavior" should be and how do we check that at runtime?

Sequences of System Calls

- Forrest et al. in early-mid 90s, attempt to understand the characteristics of an intrusion



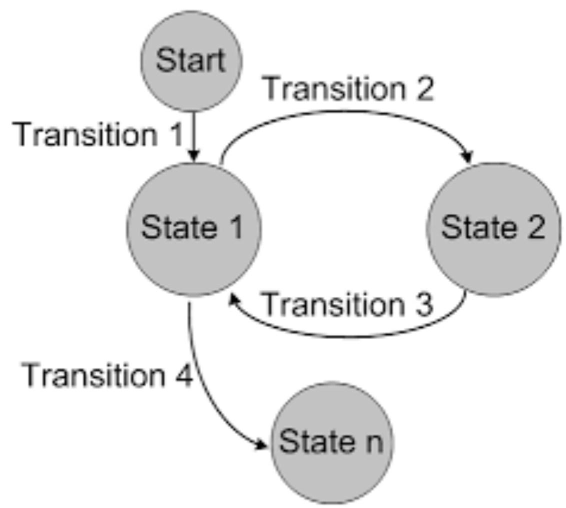
- Idea: match sequence of system calls with profiles
  - *n-grams* of system call sequences (learned)
    - Match sliding windows of sequences
    - Record the number of mismatches
    - Use n-grams of length ***5, 6, 11***.
- If found, then it is normal (w.r.t. learned sequences)

Compare Program Execution

- … to a state machine that describes all legal program executions [David Wagner, PhD thesis]
  - In terms of system calls
- Finite state automata
  - System calls (essentially) correspond to states and programs transition among them
- Pushdown automata
  - More accurate representation of the execution stack context in which system calls may occur

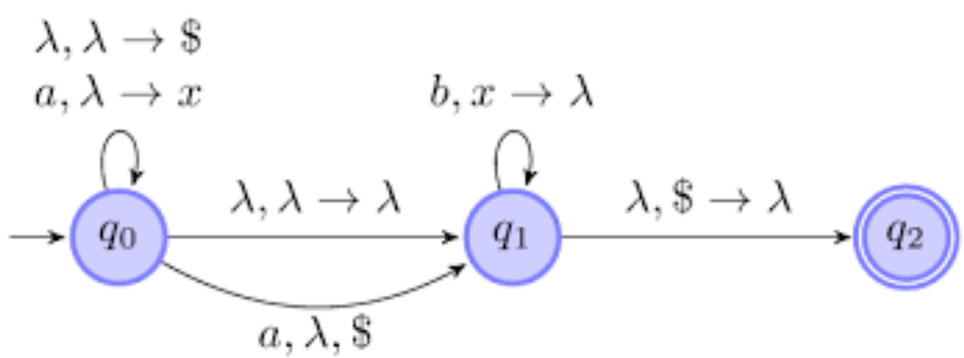
Finite State Automata Detection

- What system calls may ever follow system call X?
  - E.g., transitions from the state of system call X to each of the successor system calls
  - May use a sequence of system calls to indicate a transition



Pushdown Automata Detection

- What system calls may ever follow system call X in context (stack)?
  - There will be transitions from the state of system call X and call stack to the possible successor system calls from that context



Limitations

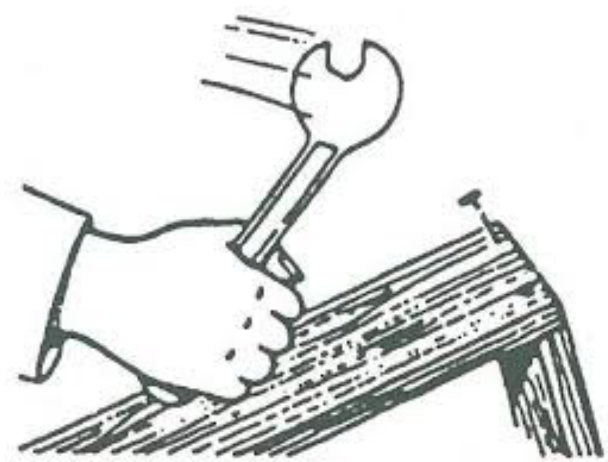
- How would you attack these anomaly detection methods?

Limitations

- How would you attack these anomaly detection methods?
- Mimicry [Wagner, CCS 2002]
  - Concoct malware that produces system call sequences that comply with state machines
  - Hard to predict argument values on advance, so can choose them
  - Or ignore results
- Possible to produce an ROP attack that mimics a state machine?

Misuse Detection

- Detect that a program performs "attack" behavior
  - Program performs malicious operations



Misuse Detection

- Classically found via signatures
  - Byte patterns present in malware
- What are some limitations of signatures?

Behavior Graphs

- Directed acyclic graphs consisting of a malware's system calls [Kolbitsch, USENIX 2009]
  - Constrain system call arguments
    - From where is the value derived – system call output
  - G = (V, E, F, ∂)
    - V: system calls; E: VxV
    - F: Function for each system call; ∂: function to arg map
  - Whenever an input argument a_i for system call *y* depends on the some output o_j produced by system call *x*, we introduce an edge from the node that corresponds to *x*, to the node that corresponds to *y*.

Behavior Graphs – I/O Function

- Use binary analysis to create a “function” that computes the output given the input
- Given input and code executed, could compute the argument value used in another system call
  - What if other program data is combined with that input?

Behavior Graphs – Effective?

- Training: Not possible to extract graphs for all

Name	Samples	Kaspersky variants	Our variants	Samples detected	Effectiveness
Allaple	50	2	1	50	1.00
Bagle	50	20	14	46	0.92
Mytob	50	32	12	47	0.94
Agent	50	20	2	41	0.82
Netsky	50	22	12	46	0.92
Mydoom	50	6	3	49	0.98
Total	300	102	44	279	0.93

Table 2: Training dataset.

- Detection: 92% of "known" samples

Name	Samples	Known variant samples	Samples detected	Effectiveness
Allaple	50	50	45	0.90
Bagle	50	26	30	0.60
Mytob	50	26	36	0.72
Agent	50	4	5	0.10
Netsky	13	5	7	0.54
Mydoom	50	44	45	0.90
Total	263	155	168	0.64

Table 3: Detection effectiveness.

Study Malware

- Malware is “in the wild”
  - Can’t we study it and learn its behavior and defenses against that behavior?

Anti-Reversing

- Art of Unpacking
  - Now malware developers actively develop their malware to evade analysis

Anti-Reversing

- Art of Unpacking
- Detect various side channels created when using tools to analyze malware
- E.g., Debuggers (Windows)
  - Software breakpoint
    - Modify code – rewrite instructions to trap to debugger
  - Hardware breakpoint
    - Debug registers are set

Anti-Reversing

- Art of Unpacking
- Detect various side channels created when using tools to analyze malware
- E.g., Debuggers (Windows)
  - Others
    - Slow the execution – can detect time delays (rdtsc)
    - Debugger privileges asserted
    - Parent process is different
    - Debug windows are created
    - Debugger processes are among tasks

Anti-Reversing

- Art of Unpacking
- Proactive defenses against analysis
  - Encryption
  - Compression
  - Permutation
  - Garbage code
- What is the benefit of garbage code to confusing the reverser?

Avoid Detection

PENNSTATE

- Modify debuggers
- Hide debuggers from the system (like malware hides processes)
- Don't use debuggers
- Avoid software and hardware breakpoints
- …

Reversing with SMM

- System management mode (SMM)
  - Sometimes called “ring -2”
  - Specific to Intel x86 processors
    - “all normal execution, including the operating system, is suspended and …” [Wikipedia]
    - “special separate software, which is usually part of the firmware or a hardware-assisted debugger, is executed with high privileges” [Wikipedia]
- Originally for power management and low-level systems management

Reversing with SMM

PENNSTATE

1855

- System management mode (SMM)
  - Can SMM configuration be interrogated by malware running at user-level?
  - …as opposed to a debugger that runs at the same privilege level

Malware Analysis in SMM

- Analyze malware at SMI (interrupt)
  - Can be asserted by software or hardware
    - Software: Write to Advanced Configuration and Power Interface (ACPI) port
      - I.e., add an instruction (out) to malware code – i.e., write code
    - Hardware: Two ways
      - (1) Serial interrupt: configuring the redirection table in I/O Advanced Programmable Interrupt Controller (APIC)
      - (2) Counter: set the corresponding performance counter (PerfCtr0) register to the maximum value

Malware Analysis in SMM

- Analyze malware at SMI (interrupt)
  - Can be asserted by software or hardware
    - Software: Write to Advanced Configuration and Power Interface (ACPI) port
      - Adversary can detect malware code modifications
    - Hardware: Two ways
      - (1) Serial interrupt: configuring the redirection table in I/O Advanced Programmable Interrupt Controller (APIC)
      - (2) Counter: Adversary can read performance counters from user space

Take Away

PENNSTATE 1855

- Problem: Detect malware before it is run
- In general, we can try to detect anomalies or misuse, but both have significant challenges
- Anomaly detection must detect that a running process really runs malware – model of expected
- Misuse detection must detect malice – and other examples of same malice – models of malice
- Malware writers now make reversing difficult
- Intrusion detection is hard to do accurately w/o causing false positives