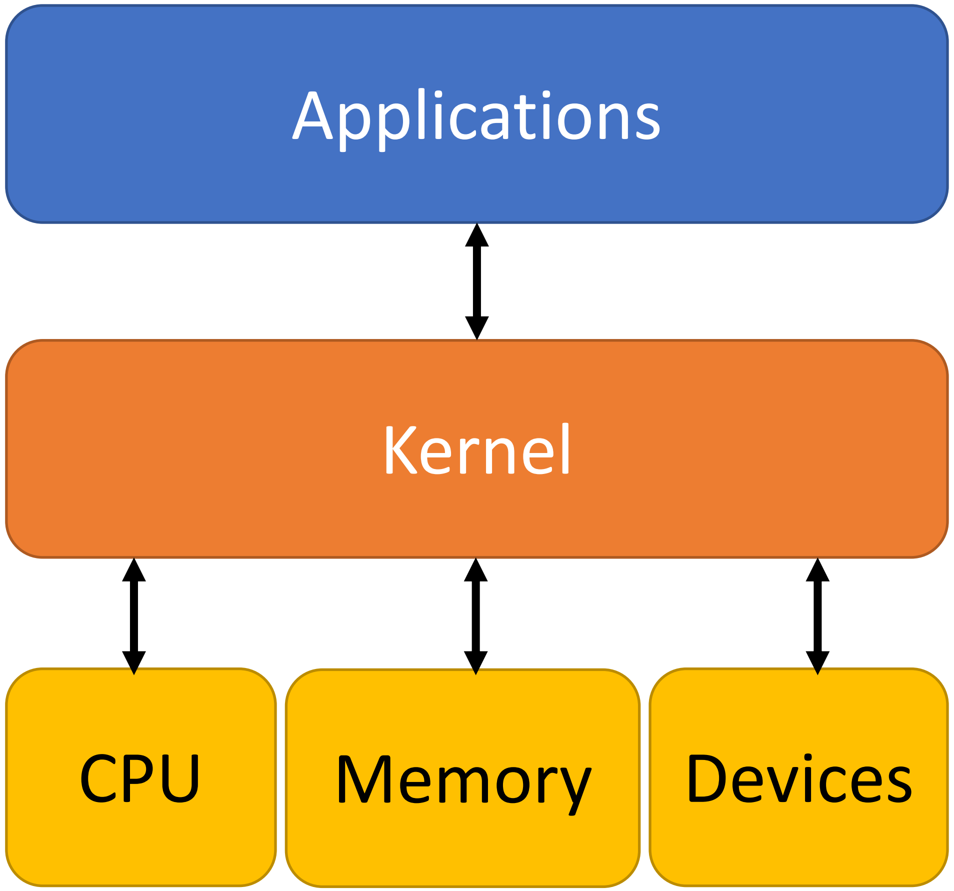
CS5231: Systems Security

Lecture 6: Kernel, Auditing, and System Provenance

Kernel Introduction

- Core of Operating System
- A software that interface between applications & hardware
- Runs in a different CPU privilege level than normal software



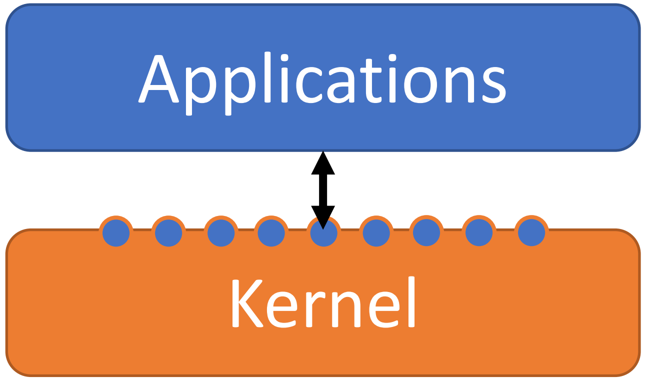
Under the Hood: Kernel Components

- Process Scheduler
  - Fair distribution of CPU time across processes
- Memory Management Unit (MMU)
  - Fair distribution of memory resources among processes
- Virtual File System (VFS)
  - Provide interface for accessing storage devices
- Networking
- Inter Process Communication (IPC)
- and others…
  - I/O scheduling & protection, buffering, caching, error handling, interrupt & event handling …

Types of Kernels

- Monolithic Kernel
  - Simple, common, and **<u>same address space</u>**
- Microkernel
  - Contain essential functionalities, hence smaller kernel
  - User & kernel have **<u>different address space</u>**
- Hybrid Kernel
  - Speed from monolithic & modularity from microkernel
- Nano Kernel
  - Small & bare minimum functionalities
- Exo Kernel
  - Distinct resource protection & management

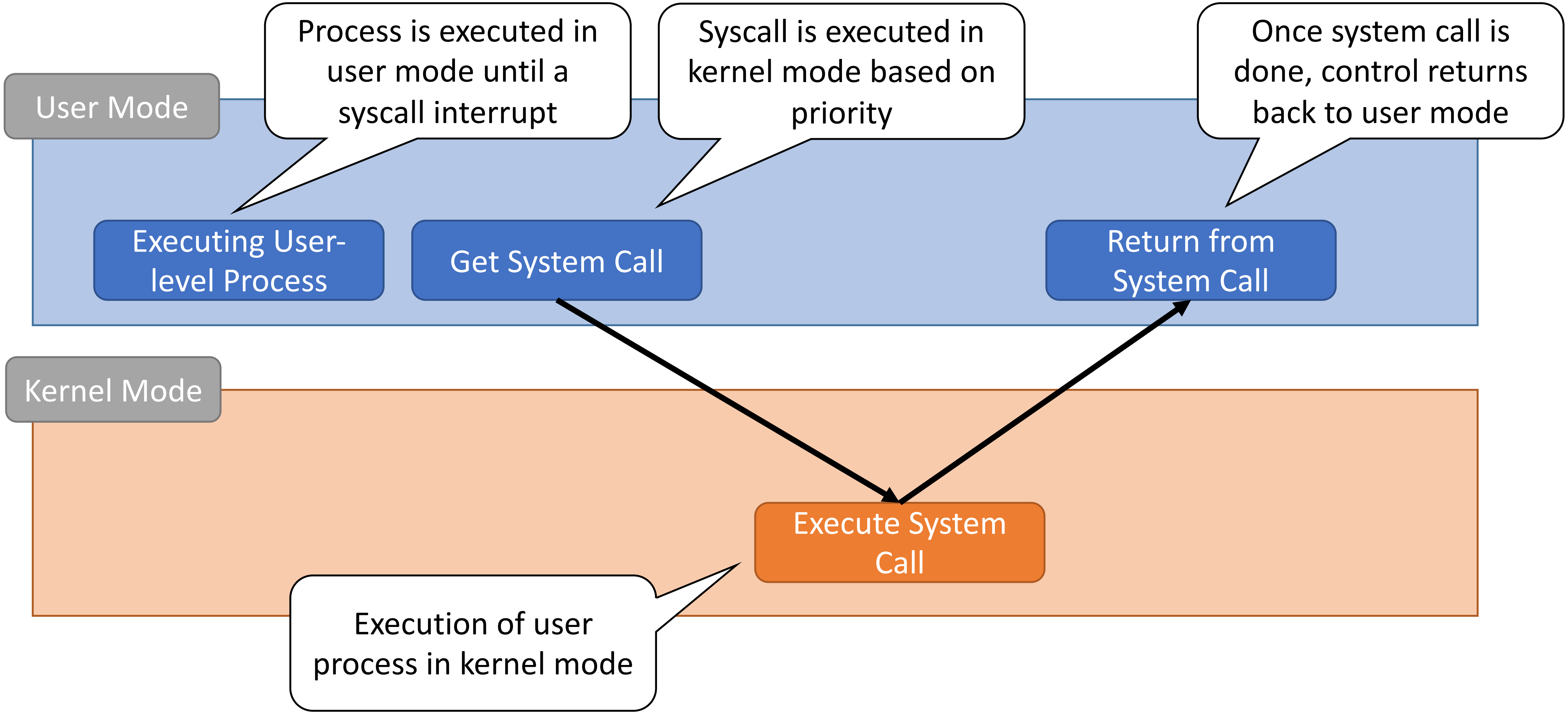
Entering the Kernel



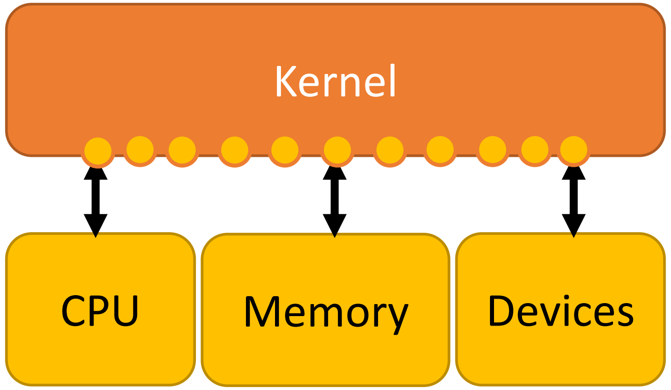
System Call

- Entry point into kernel
- Application can call *syscall* to perform task
  - Create process
  - Networking
  - File I/O
  - …
- All syscall in Linux manpage
  - https://man7.org/linux/man-pages/man2/syscalls.2.html

Syscall Process



Kernel Modules & Drivers



- Kernel module
  - Codes that can be loaded & unloaded into the kernel on-demand
  - Also known as *loadable kernel module*
- Kernel driver
  - Can be built statically in the kernel, or built as module for dynamic loading
  - Example of basic USB driver
    - https://github.com/muratdemirtas/Linux-Kernel-Examples/blob/master/Basic_USB_Driver/Basic_USB_Driver/Basic_USB_Driver_main.c

Kernel Module Example

```
//Hello World Example

//include needed libraries for building and printk() function
#include  <linux/module.h>
#include  <linux/init.h>

//start function for loading our module
int init_module(void)
{
    //kernel message, you can see with dmesg command
    printk(KERN_INFO "MOD: HELLO WORLD EXAMPLE LOADED.\n");
    //return 0 (success) if module was loaded correctly, a non 0 return means module failed.
    return 0;
}

//this function will execute when you remove this module from kernel.
void cleanup_module(void)
{
    //print debug message to kernel.
    printk(KERN_INFO "MOD: WORLD EXAMPLE LEAVING.\n");
}
```

printk is a C function that prints information to kernel log

https://github.com/muratdemirtas/Linux-Kernel-Examples/blob/master/Hello_World/Hello_World/Hello_World_main.c

Kernel Module Example

- Compile & Run
  - Need to have `Kbuild`, `Makefile`, and `module.c` before `make`
  - Use `insmod module.ko` & `rmmod module.ko` & `lsmod`
- View Log

```
# cat /var/log/syslog | tail -2
Oct 06 21:38:41 mysystem kernel: MOD: HELLO WORLD EXAMPLE LOADED.
Oct 06 21:38:47 mysystem kernel: MOD: WORLD EXAMPLE LEAVING.

# dmesg | tail -2
MOD: HELLO WORLD EXAMPLE LOADED.
MOD: WORLD EXAMPLE LEAVING.
```

System Auditing

Recording key events in the kernel

Endpoint Monitoring Solutions

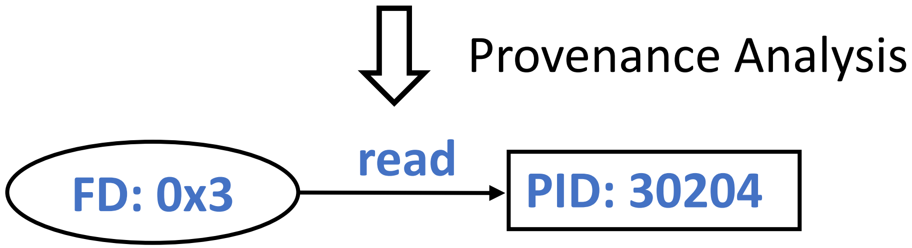
Endpoint monitoring solutions record **audit logs** for attack investigation

elastic splunk> LogRhythm

Audit logs:

- A history of events representing OS-level activities
- Provide visibility into security incidents with data provenance

type=SYSCALL msg=audit(**30/09/19 20:34:53**.383:98866813) : arch=x86_64 **syscall=read** exit=25 **a0=0x3** ppid=15757 **pid=30204** auid=junzeng sess=6309



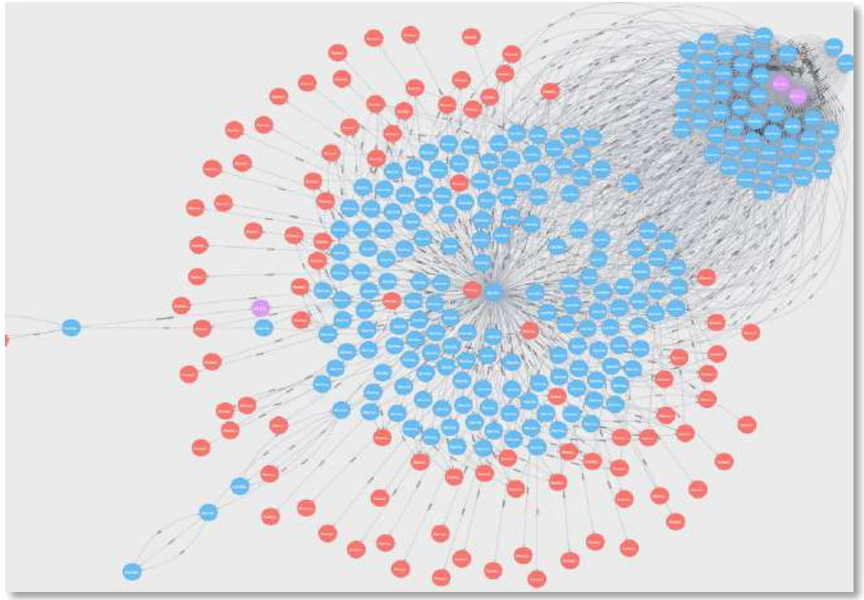
Investigation Using Audit Logs

Researchers use a **provenance graph** to navigate through audit logs:

- Nodes: system entities (e.g., process, file, and socket) & Edges: system calls
- **Backward/forward tracking** to find root cause of an attack and its ramifications

Real-world audit logs are always **large-Scale,** and provenance graphs are **Sophisticated!**

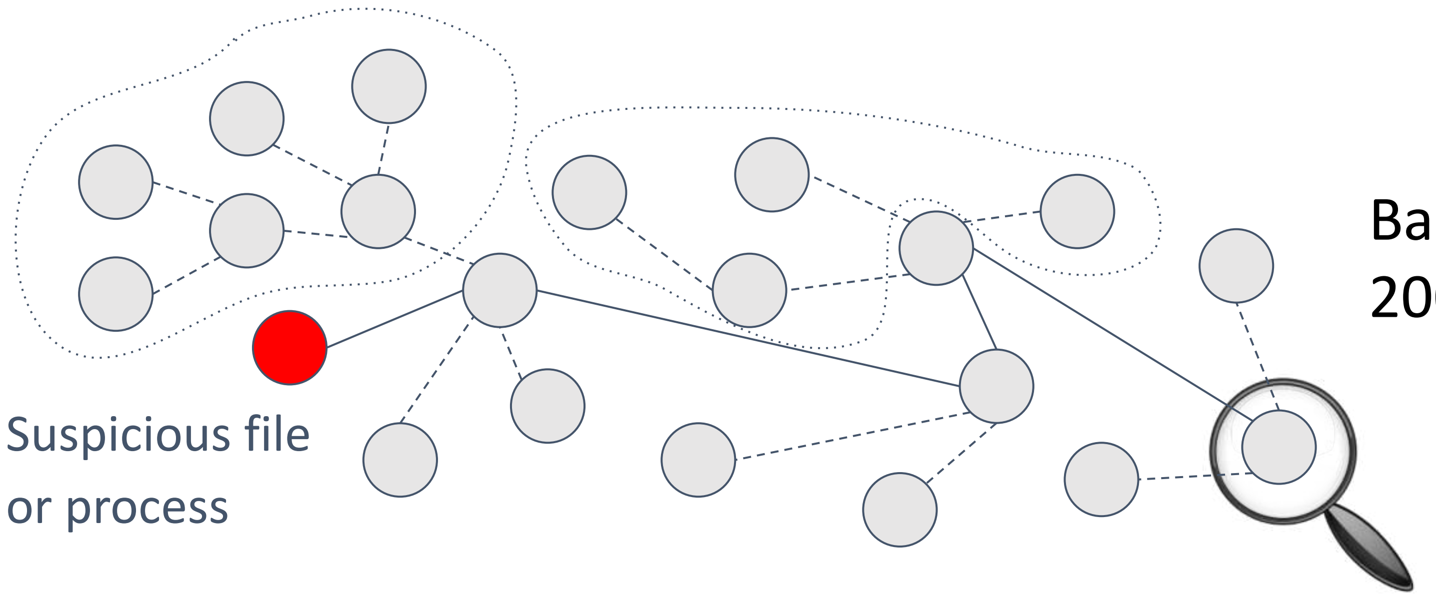
{"@timestamp":"2020-10-31T14:14:47.777Z","@metadata":{"beat":"auditbeat","type":"doc","version":"6.8.12"},"user":{"suid":"0","fsgid":"0","gid":"0","name_map":{"egid":"root","gid":"root","uid":"root","sgid":"root","suid":"root","auid":"yinfang","euid":"root","fsgid":"root","fsuid":"root"},"sgid":"0","uid":"0","euid":"0","egid":"0","fsuid":"0","auid":"1000"},"process":{"name":"sshd","exe":"/usr/sbin/sshd","pid":"18104","ppid":"1689"},"auditd":{"sequence":166719,"result":"success","session":"705","data":{"tty":"(none)","a3":"8","a0":"4","exit":"384","arch":"x86_64","syscall":"read","a2":"180","a1":"7ff97aeba120"}}}
{"@timestamp":"2020-10-31T14:14:47.777Z","@metadata":{"beat":"auditbeat","type":"doc","version":"6.8.12"},"user":{"auid":"1000","fsuid":"0","fsgid":"0","egid":"0","sgid":"0","suid":"0","uid":"0","euid":"0","name_map":{"fsgid":"root","sgid":"root","suid":"root","uid":"root","auid":"yinfang","egid":"root","euid":"root","fsuid":"root","gid":"root"},"gid":"0"},"process":{"pid":"18104","ppid":"1689","name":"sshd","exe":"/usr/sbin/sshd"},"auditd":{"data":{"a3":"8","a2":"180","arch":"x86_64","tty":"(none)","a0":"4","exit":"384","a1":"7ff97aeba120","syscall":"read"},"sequence":166720,"result":"success","session":"705"}}
{"@timestamp":"2020-10-31T14:14:47.777Z","@metadata":{"beat":"auditbeat","type":"doc","version":"6.8.12"},"user":{"egid":"0","auid":"1000","fsgid":"0","name_map":{"euid":"root","fsuid":"root","gid":"root","sgid":"root","auid":"yinfang","fsgid":"root","suid":"root","uid":"root","egid":"root"},"suid":"0","euid":"0","uid":"0","sgid":"0","fsuid":"0","gid":"0"},"process":{"name":"sshd","exe":"/usr/sbin/sshd","pid":"18104","ppid":"1689"},"auditd":{"sequence":166721,"result":"success","session":"705","data":{"a1":"7ff97aeba120","arch":"x86_64","a3":"8","exit":"384","syscall":"read","a2":"180","a0":"4","tty":"(none)"}}}



Audit Log Analysis

- Starting from a detection point, *Backtracker* does:
  - Events & objects identification related detection point
  - Generate dependency graph
  - Use rules to prune unrelated nodes in the dependency graph

Dependency explosion!

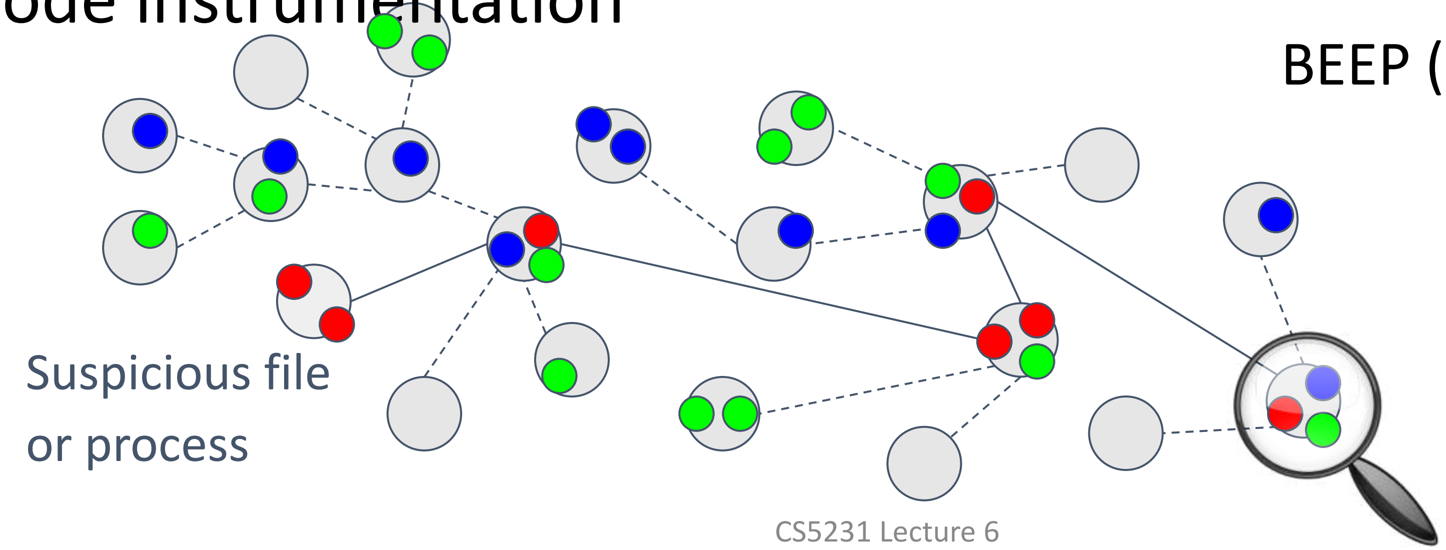


Backtracker (King & Chen, 2003)

Audit Log Analysis

- Resolve *dependency explosion* problem in a long running application
  - Fine-grained provenance tracing technique
  - Identifying unit boundaries & dependences
  - Partition into individual *unit*
  - Code instrumentation

BEEP (Lee et al., 2013)

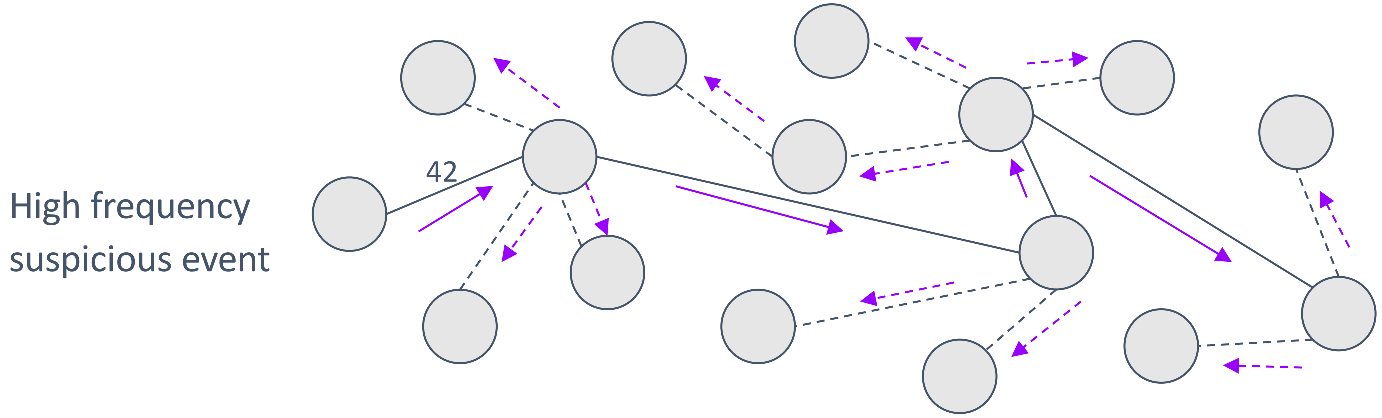


Audit Log Analysis

turn a security problem into a graph analysis problem

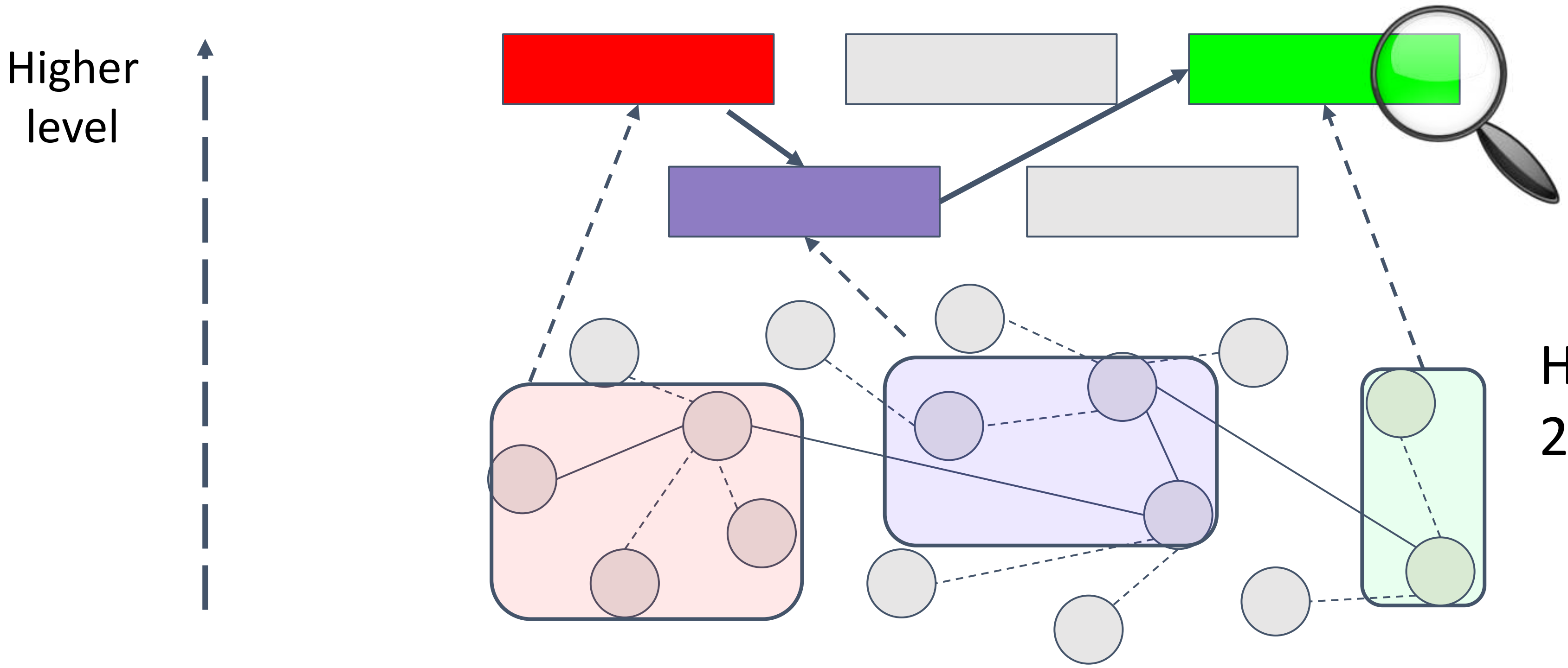
- Address *threat alert fatigue* during threat investigation
  - Assign anomaly scores to every edge in dependency graph
  - Based on frequency of events that have occured (historical & contextual information)
  - Propagated score through edges in the graph
  - Generate aggregated anomaly score for triaging

NoDoze (Hassan et al., 2019)



Audit Log Analysis

- Generate high-level graph during threat investigation
  - Develop robust & reliable detection signal
  - Correlate between suspicious information flow



HOLMES (Milajerdi et al., 2019)

What to be logged?

- It depends!
  - Application scenarios
  - Business purposes
- Questions
  - Universities may keep different kinds of logs
    - Student/staff record
    - Student/staff emails?

Why logging?

- Main goals
  - Record valuable data about applications, system and network activities
  - Provide clues for later investigation

- It must be performed by a trusted/ authorized entity …

A Taxonomy of Logs

- Application logs
  - Contain application-level events
  - e.g., access log / error log
- System logs
  - Contain system-related events
  - e.g., device driver loading/unloading
- Security logs
  - Contain security-related events
  - e.g., valid/invalid logon attempts

Application Log

- Mail server log
  - Connection status; SMTP queues;…
- FTP server log
  - Current logins; file uploaded/downloaded;…
- Database server log
  - Objects accessed; creation of new tables; …
- Web browser log
  - URLs visited; file downloaded; …

System Log

- System startup/initialization log
  - *dmesg*
- System running status
  - */proc*
- When a system goes wrong (e.g., crash)
  - What should be logged?
  - How to collect log?
    - Kernel dumps (egg vs. chicken)

Security Log

- Firewall log
  - Inbound/outbound packets
  - Packets dropped
  - TCP connections rejected
- IDS log
  - Unauthorized file modifications
  - Intrusion alerts
  - Suspicious system activities
  - Attack statistics

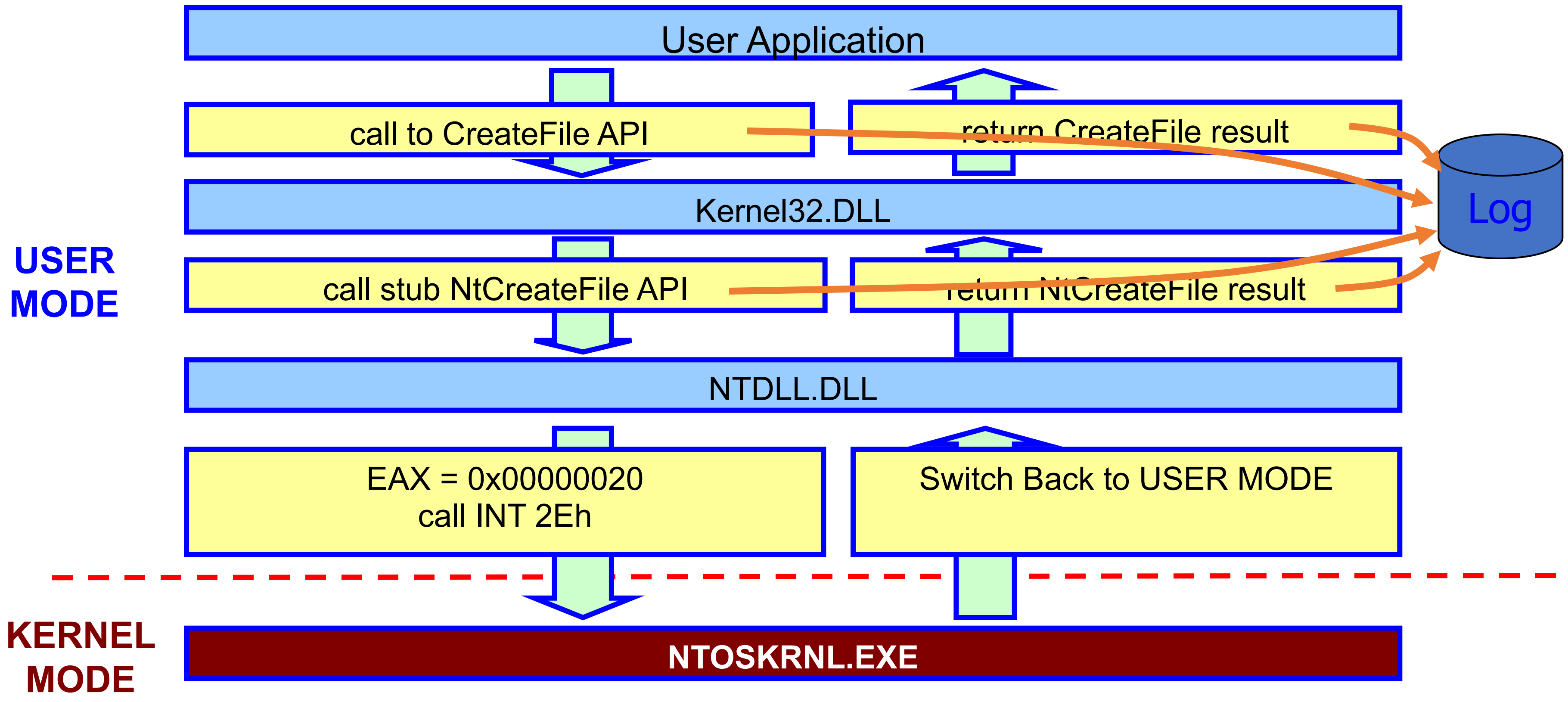
General Logging Mechanisms

- Application-level
  - Library wrapping / API hooking
- Kernel-level
  - Syslogd/klogd
  - System call interception
  - Linux security module
- Virtual Machine Monitor-level
  - System call interception

API Hooking

- Similar to functional overloading in programming language
- Commonly used for debugging purposes
- Platform-specific
  - Windows:
    - DLL injection/binary rewriting
  - Unix:
    - LD_PRELOAD trick

API Hooking


User Application
call to CreateFile API
return CreateFile result
Kernel32.DLL
Log
USER MODE
call stub NtCreateFile API
return NtCreateFile result
NTDLL.DLL
EAX = 0x00000020
call INT 2Eh
Switch Back to USER MODE
KERNEL MODE
NTOSKRNL.EXE

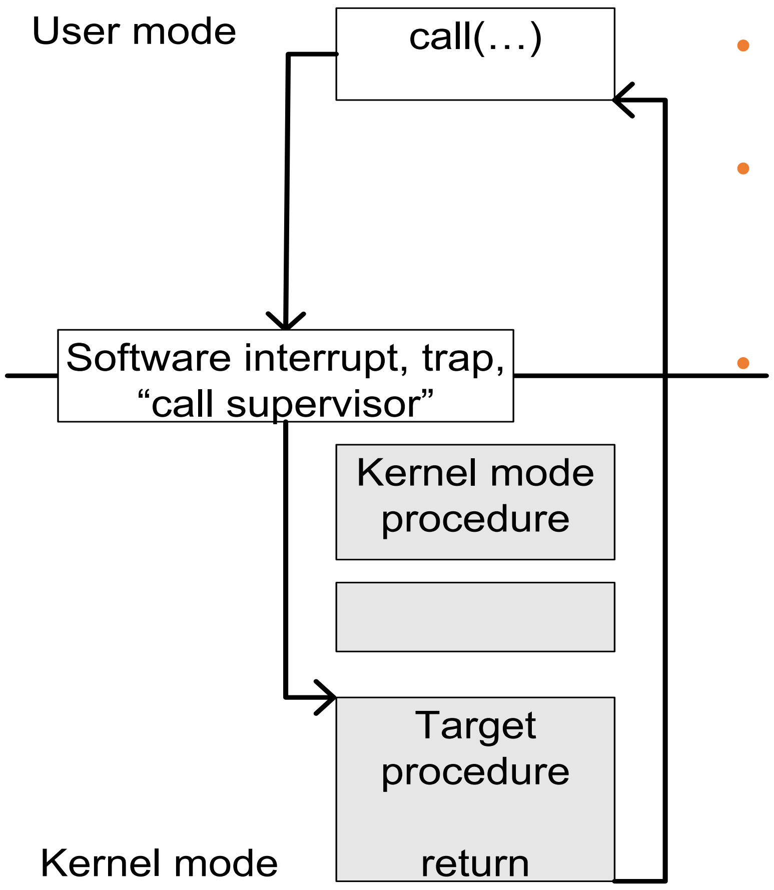
Example API Hooking Log

- iexplore.exe (268) : File::Write (C:\WINDOWS\Downloaded Program Files\ieloader.exe)
- iexplore.exe (268) : Sys::Execute (C:\ WINDOWS\Downloaded Program Files\ieloader.exe)
- ieloader.exe (1728) : Reg::SetValue (HKLM\Software\Microsoft\Windows\CurrentVersion\Run, FX)
- iexplore.exe (268) : Net::Connect (205.205.86.51, 80)
- ieloader.exe (1728) : File::Write (C:\WINDOWS\System32\uvbdcgrtjce.dll)
- ieloader.exe (1728) : Sys::Execute (C:\Program.exe)
- iexplore.exe (268) : File::Write (C:\Documents and Settings\HS\Local Settings\Temporary Internet Files\Content.IE5\EJ89IPOV\init[1].js)
- iexplore.exe (1812) : Reg::SetValue (HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders, Desktop)
- iexplore.exe (1812) : Reg::SetValue (HKCU\Software\Microsoft\Windows\ShellNoRoam\BagMRU, NodeSlots)
- iexplore.exe (1812) : Reg::SetValue (HKCU\Software\Microsoft\Windows\ShellNoRoam\BagMRU, MRUListEx)
- iexplore.exe (268) : File::Write (C:\WINDOWS\Downloaded Program Files\SET4A.tmp)
- iexplore.exe (268) : File::Write (C:\WINDOWS\Downloaded Program Files\ieloader.exe)
- iexplore.exe (268) : Sys::Execute (C:\WINDOWS\Downloaded.exe)
- ieloader.exe (1728) : Reg::SetValue (HKLM\Software\Microsoft\Windows\CurrentVersion\Run, FX)
- iexplore.exe (268) : Net::Connect (205.205.86.51, 80)
- ieloader.exe (1728) : File::Write (C:\WINDOWS\System32\uvbdcgrtjce.dll)
- iexplore.exe (268) : File::Write (C:\Documents and Settings\HS\Local Settings\Temporary Internet Files\Content.IE5\EJ89IPOV\init[1].js)

General Logging Mechanisms

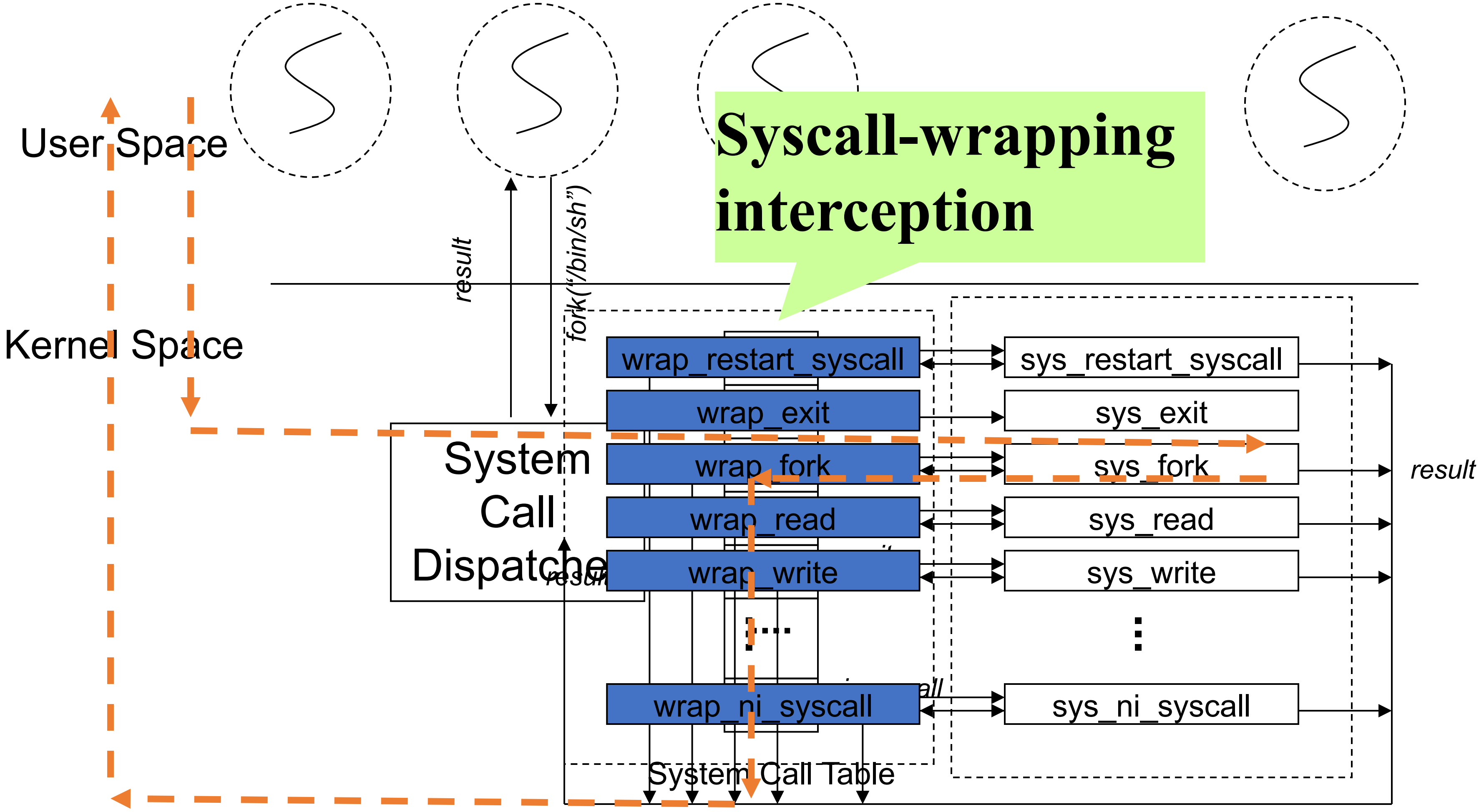
- Application-level
  - Library wrapping / API hooking
- Kernel-level
  - Syslogd/klogd
  - System call interception
  - Linux security module
- Virtual Machine Monitor-level
  - System call interception

System Call



- The parameters of the call are passed according to certain OS/hardware specific convention
- Switch to protected mode
  - E.g., int $0x80
- A special module takes over, that will analyze the parameters and the access rights; this module can reject the system call
- If accepted, then the corresponding routine from the operating system is executed and the result is returned to the user; upon return, the user mode is restored

System Call Interception



Example System Call Log

```
673["sendmail"]: 5_open("/proc/loadavg", 0, 438) = 5
673["sendmail"]: 192_mmap2(0, 4096, 3, 34, 4294967295, 0) = 1073868800
673["sendmail"]: 3_read(5, "0.26 0.10 0.03 2...", 4096) = 25
673["sendmail"]: 6_close(5) = 0
673["sendmail"]: 91_munmap(1073868800, 4096) = 0
...
2568["httpd"]: 102_accept(16, sockaddr{2, cbbdff3a}, cbbdff38) = 5
2568["httpd"]: 3_read(5, "\1281\1\0\2\0\24...", 11) = 11
2568["httpd"]: 3_read(5, "\7\0À\5\0\128\3\...", 40) = 40
2568["httpd"]: 4_write(5, "\132@\4\0\1\0\2\...", 1090) = 1090
...
2568["httpd"]: 4_write(5, "\128\19Ê\136\18\...", 21) = 2
```

Applications of Syscall Interception

- Logging
  - Syscalltrack, sebek, …
    - *http://syscalltrack.sf.net, …*
- Sandboxing
  - Systrace, Janus, …
    - *http://www.citi.umich.edu/u/provos/systrace/, …*
- Virtual Machine
  - UML, UMLinux
    - *http://user-mode-linux.sf.net, …*

General Logging Mechanisms

- Application-level
  - Library wrapping / API hooking
- Kernel-level
  - Syslogd/klogd
  - System call interception
  - Linux security module
- Virtual Machine Monitor-level
  - System call interception

Linux Security Module: Overview

- Motivation: Separate kernel from security features
  - Minimize the impact to kernel
  - SELinux motivated the creation of LSM

- Note LSM doesn't provide any security
  - Adds security fields to kernel
  - Provides interface to manage these fields

Linux Security Module (LSM)

- Security extension for kernel
- Reduce attack surface by enforcing access policies
- Mandatory Access Control
  - Subjects, objects, and operations

Unlike DAC in Linux's access control..!!

Introduce *LSM hooks*

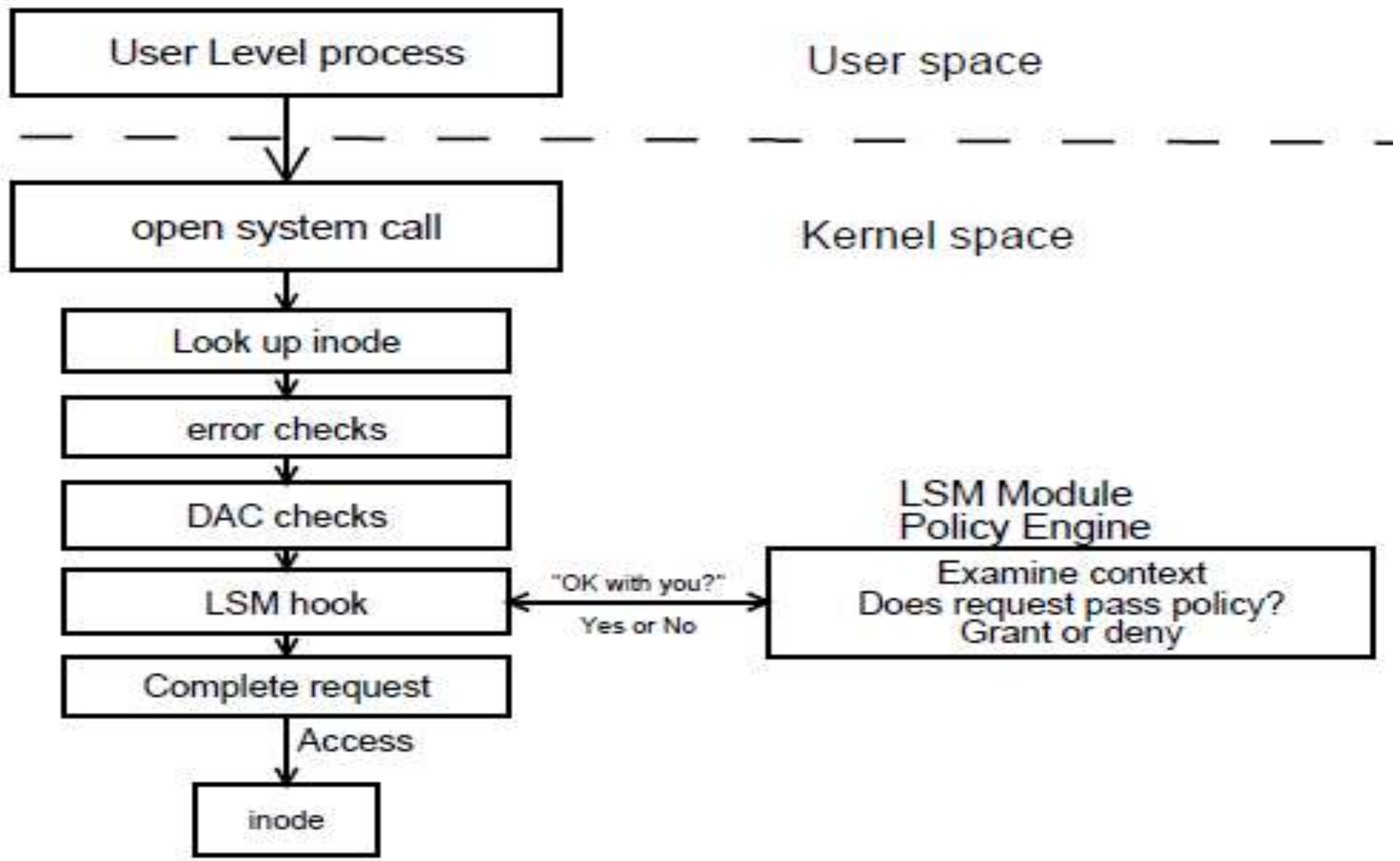
- Control access to kernel objects
  - E.g. inodes, files, credentials, devices, tasks, IPCs

"Can a subject S perform kernel Operation O on a kernel object A?"

Linux Security Module: Hooks

- Hooks
  - A set of functions to control operations on kernel objects and security fields in kernel data structures.
  - Management Hooks
    - used to manage security fields (e.g., file_alloc_security)
  - Control Hooks
    - used to perform access controls (e.g., selinux_inode_permission)

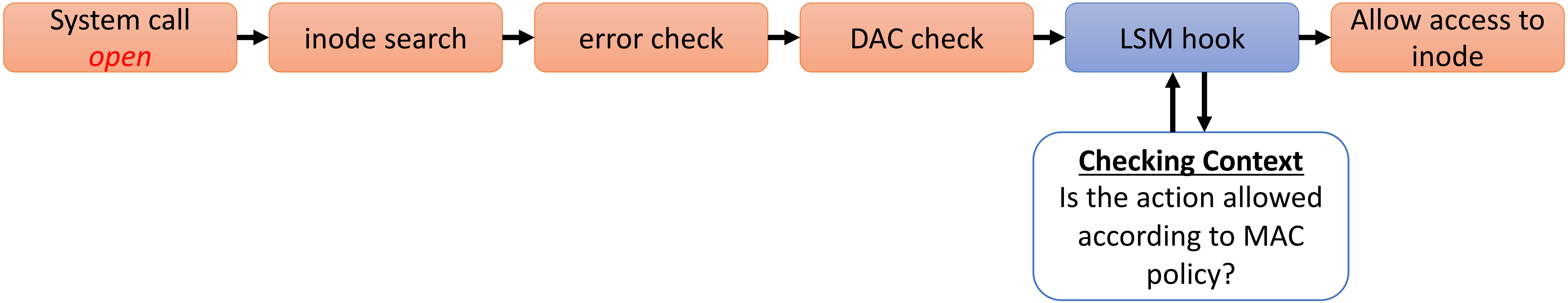
LSM Hook Architecture



Linux Security Module (LSM)

- LSM's MAC is checked only after DAC & other checks are performed
- LSM hooks are applied inline during kernel code execution
- Examples – AppArmor, SELinux, Smack, TOMOYO

<u>LSM workflow during open() syscall in kernel space</u>

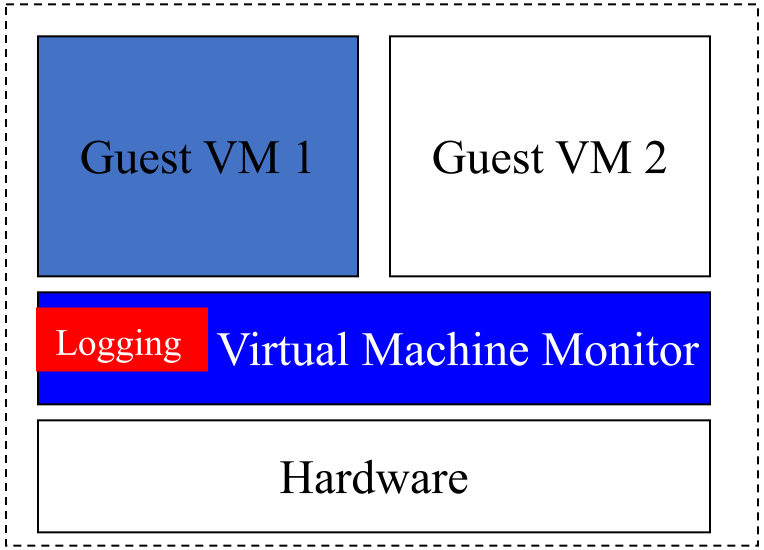


General Logging Mechanisms

- Application-level
  - Library wrapping / API hooking
- Kernel-level
  - Syslogd/klogd
  - System call interception
  - Linux security module
- Virtual Machine Monitor-level
  - System call interception

Virtual Machine Monitor-level

- Strength?
- Weakness?



Logging Storage Policies

- Reset log files at periodic intervals
- Rotate log files, keeping data for a fixed time
- Compress and archive to a tape or other permanent media

Main Applications

- Logging-based Applications
  - Intrusion Detection
  - Intrusion Recovery
  - Software Debugging

Intrusion Detection

- Intrusion Detection is the process of identifying and responding to malicious activity targeted at computing and networking resources
- Resources:
  - One computer, or
  - A local/wide area network

Models of Intrusion Detection

- Anomaly detection
  - What is usual, is known
  - What is unusual, is bad
- Misuse detection
  - What is bad, is known
  - What is not bad, is good
- Specification-based detection
  - What is good, is known
  - What is not good, is bad
- Goal → generating a **Detection Point**

Detection Point

- Suggests a possible intrusion
- Examples:
  - An anomaly log entry
    - e.g., a shell process launched
  - A suspicious system activity
    - e.g., an outbound TCP connection to a remote IRC server
  - An unauthorized modification to a critical configuration file
    - e.g., /etc/inetd.conf

After an Intrusion Is Identified

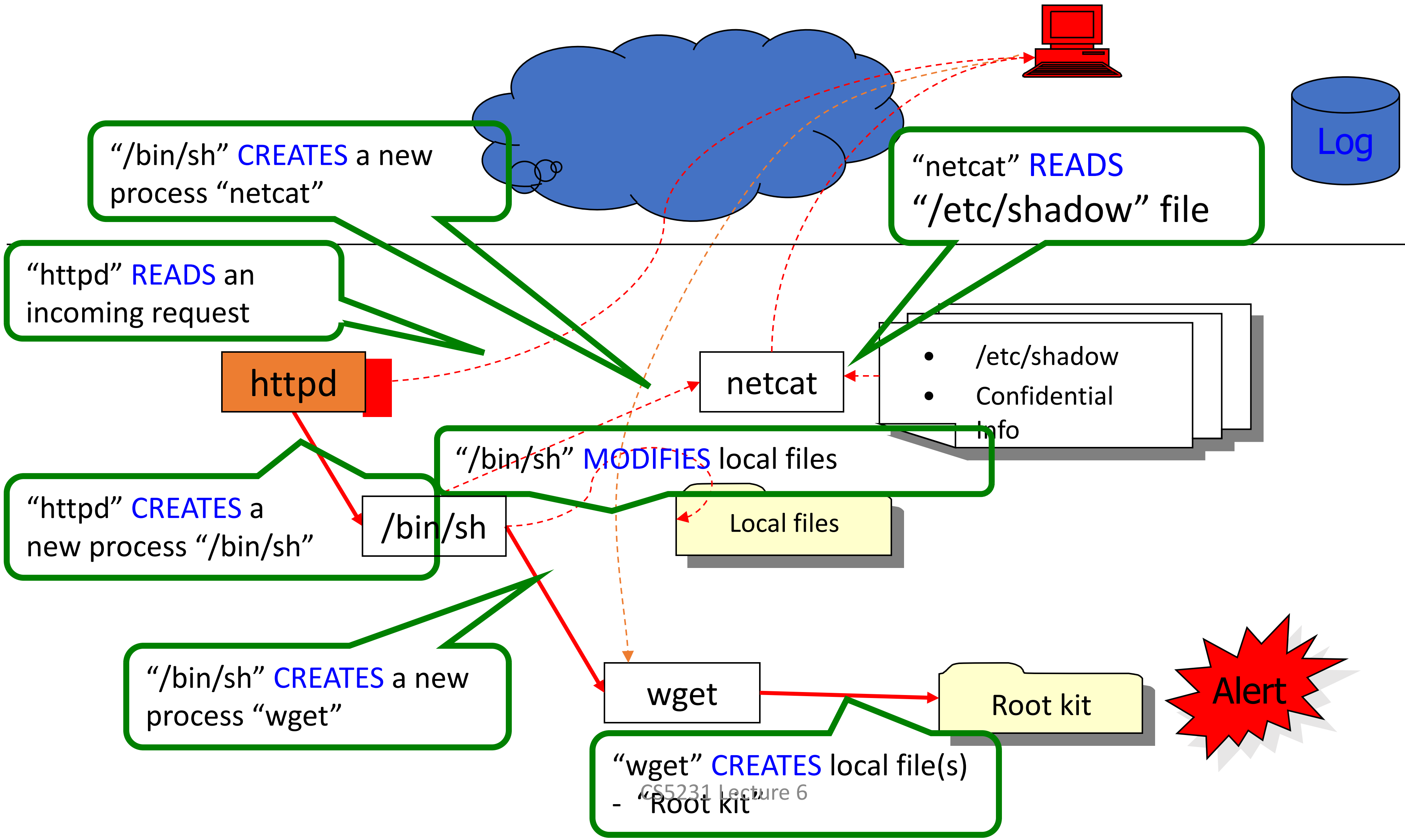
- For each intrusion, it is desirable to find out:
  - **Break-in Point:**
    - How did the attacker gain access to the system?
  - **Contaminations:**
    - What did the attacker do after the break-in?

Detection Point

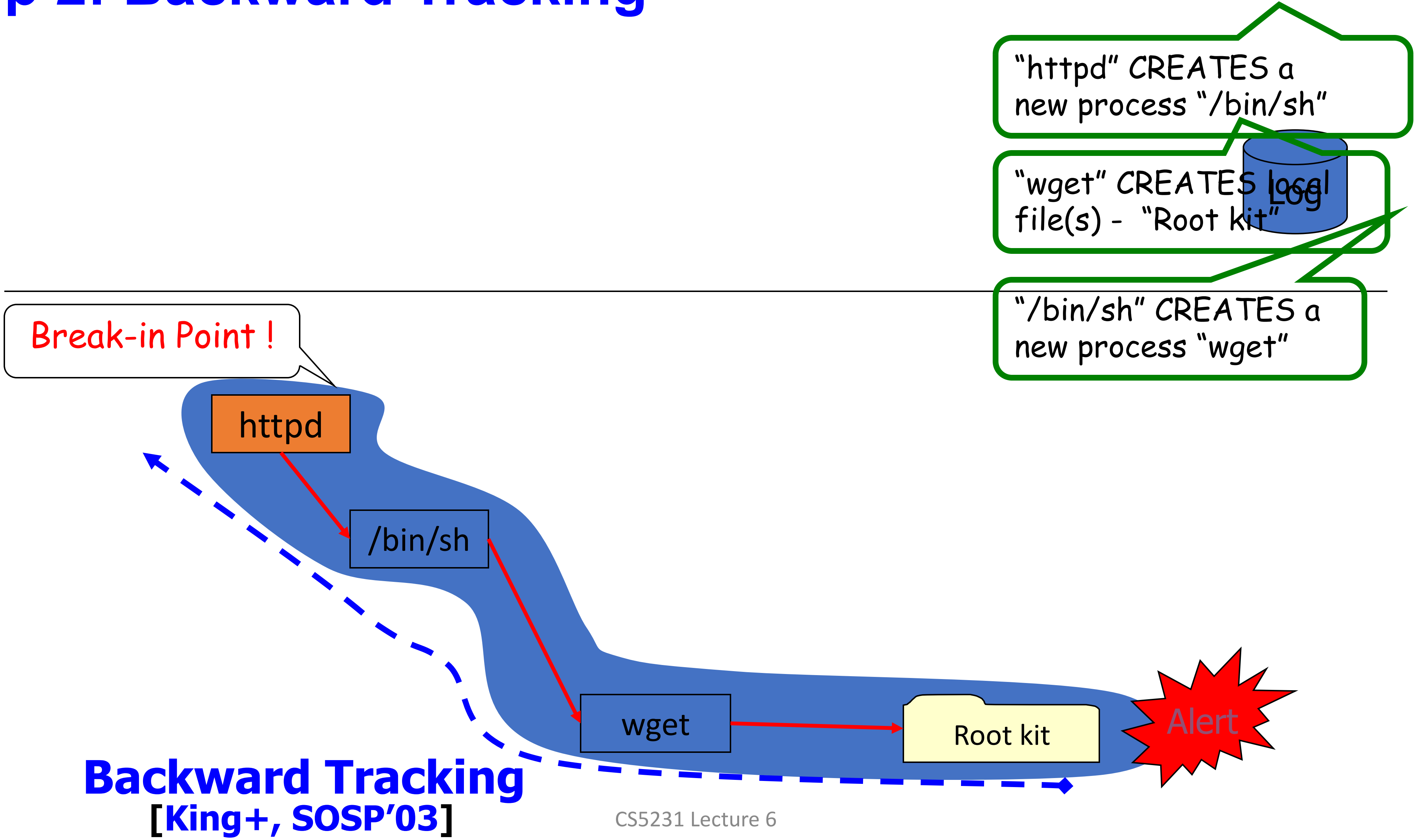
Intrusion Investigation

- Three Main Steps
  - Step 1: Online Log Collection
  - Step 2: Backward Tracking
  - Step 3: Forward Tracking

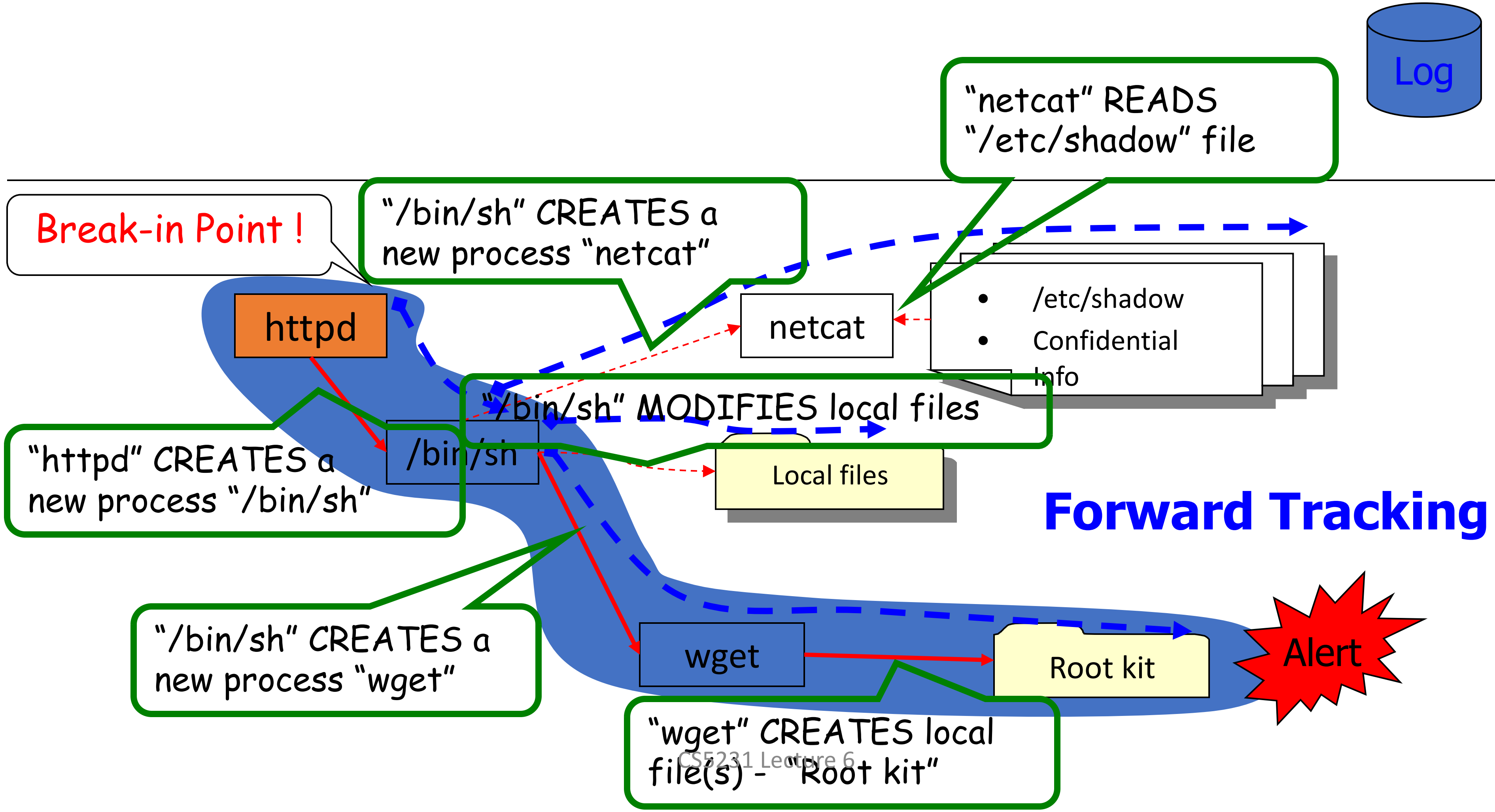
Step 1: Online Log Collection



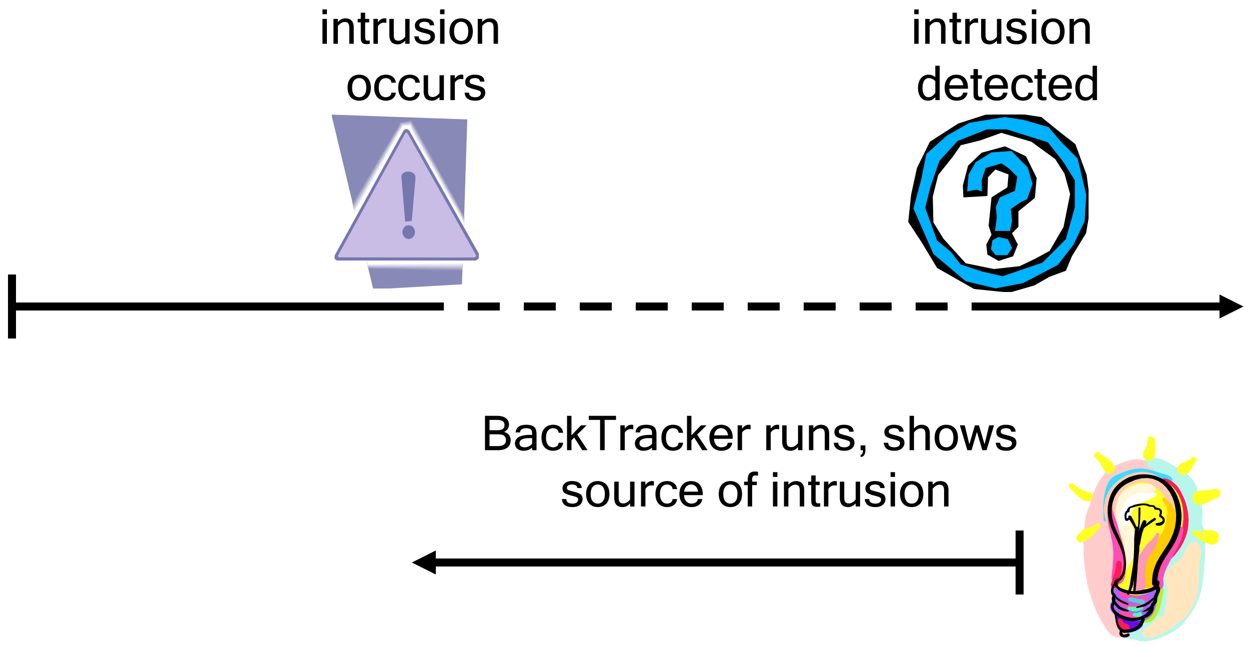
Step 2: Backward Tracking



Step 3: Forward Tracking



BackTracker



- Online component logs objects and events
- Offline component generates graphs

BackTracker Objects

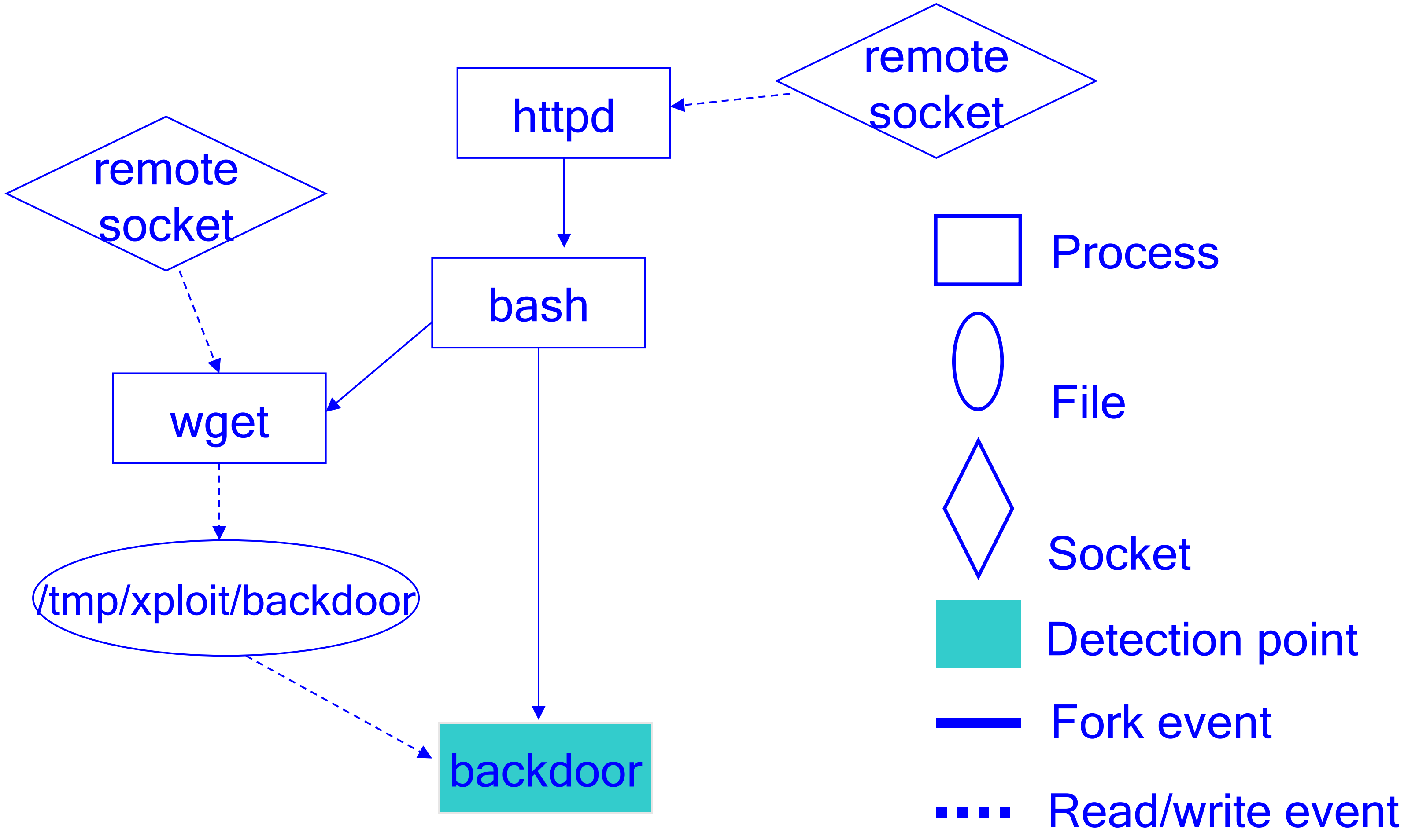
- Process
- File
- Filename

Dependency-Forming Events

- Process / Process
  - fork, clone, vfork
- Process / File
  - read, write, mmap, exec
- Process / Filename
  - open, creat, link, unlink, mkdir, rmdir, stat, chmod, …

- Dependency-tracking is an effective technique for highlighting actions of attacker

BackTracker Example

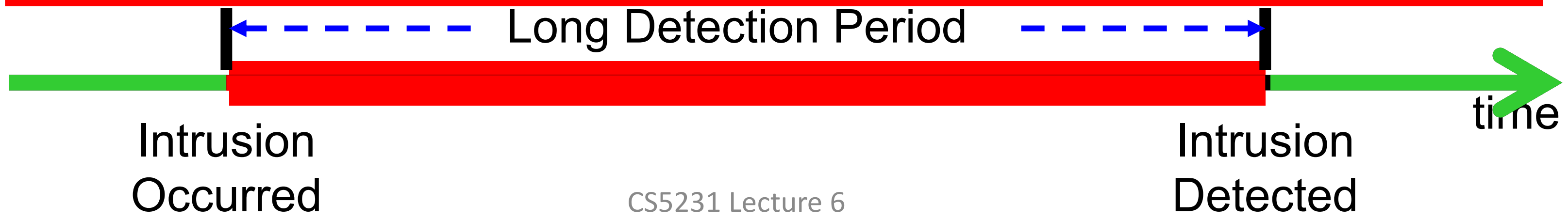


Challenge in Scalability

- Backward Tracking ➔ Break-in Point
  - Inputs: Detection Point, the Whole Log
- Forward Tracking ➔ Contaminations
  - Inputs: Break-in Point, the Whole Log

Analyze the whole log !

High Volume Log Data: 1.2 gigabytes per day for an intensive workload



Related Work

- Scale up provenance analysis:
  - Data reduction [NDSS'16, 18 …] & Query system [Security'18, ATC'18 …]
  - Recognizing behaviors of interest requires intensive manual efforts

A semantic gap between low-level events and high-level behaviors

- Apply expert-defined specifications to bridge the gap
  - Match audit events against domain rules that describe behaviors
  - Query graph [VLDB'15, CCS'19], Tactics Techniques Procedures (TTPs) specification [SP'19,20], and Tag policy [Security'17,18]

Behavior-specific rules heavily rely on domain knowledge (**time-consuming)**

Related Work

- Scale up provenance analysis:
  - Data reduction [NDSS'16, 18 …] & Query system [Security'18, ATC'18 …]

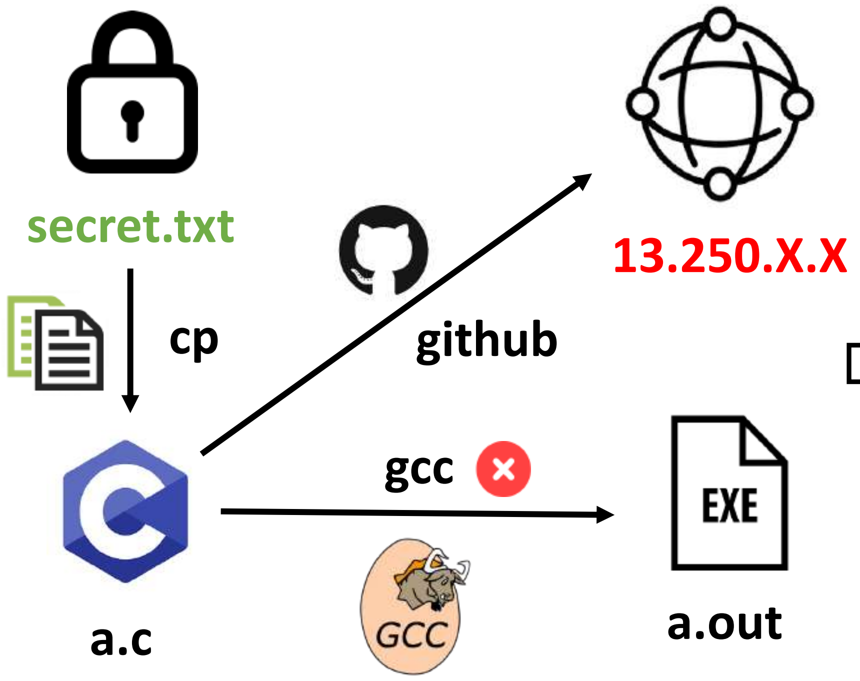
Can we automatically **abstract** high-level behaviors from low-level audit logs and **cluster** semantically similar behaviors before human inspection?

  - Query graph [VLDB'15, CCS'19], Tactics Techniques Procedures (TTPs) specification [SP'19,20], and Tag policy [Security'17,18]

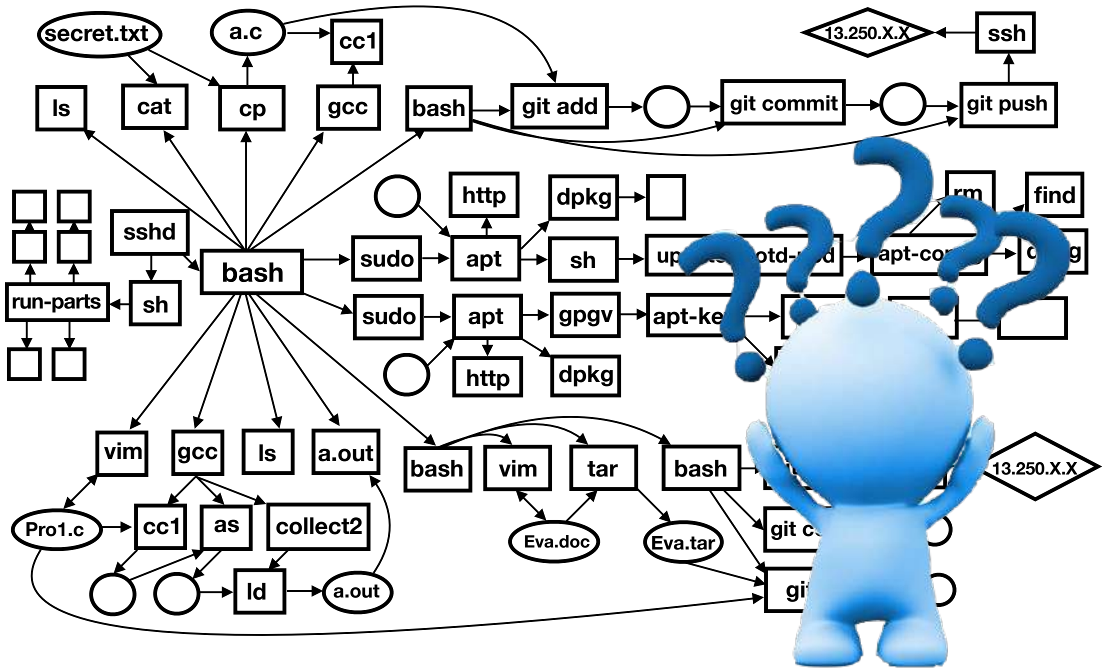
Behavior-specific rules heavily rely on domain knowledge (**time-consuming**)

Motivating Example

Attack Scenario: A software tester exfiltrates sensitive data that he has access to



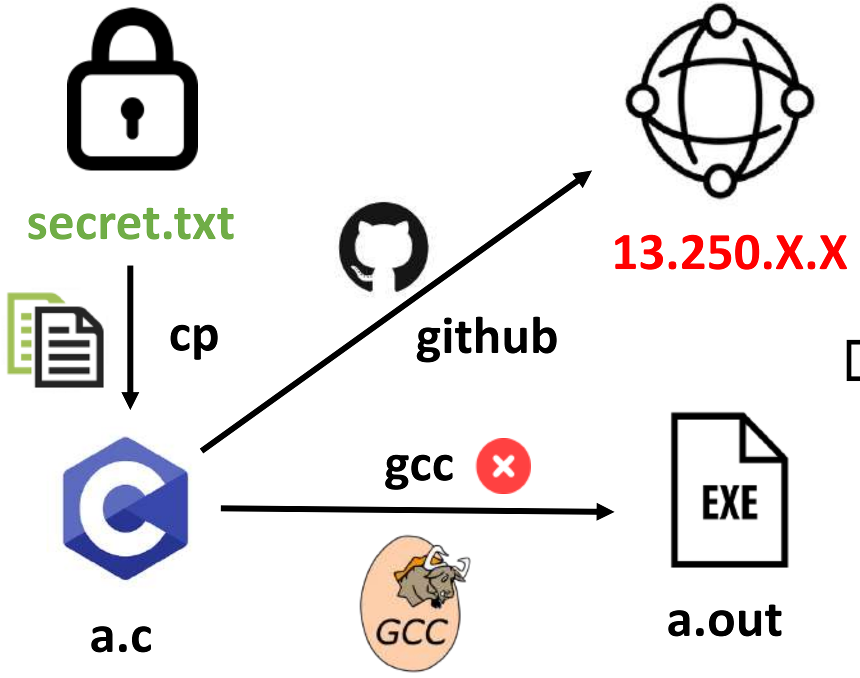
Data Exfiltration Steps



Motivating Example Logs

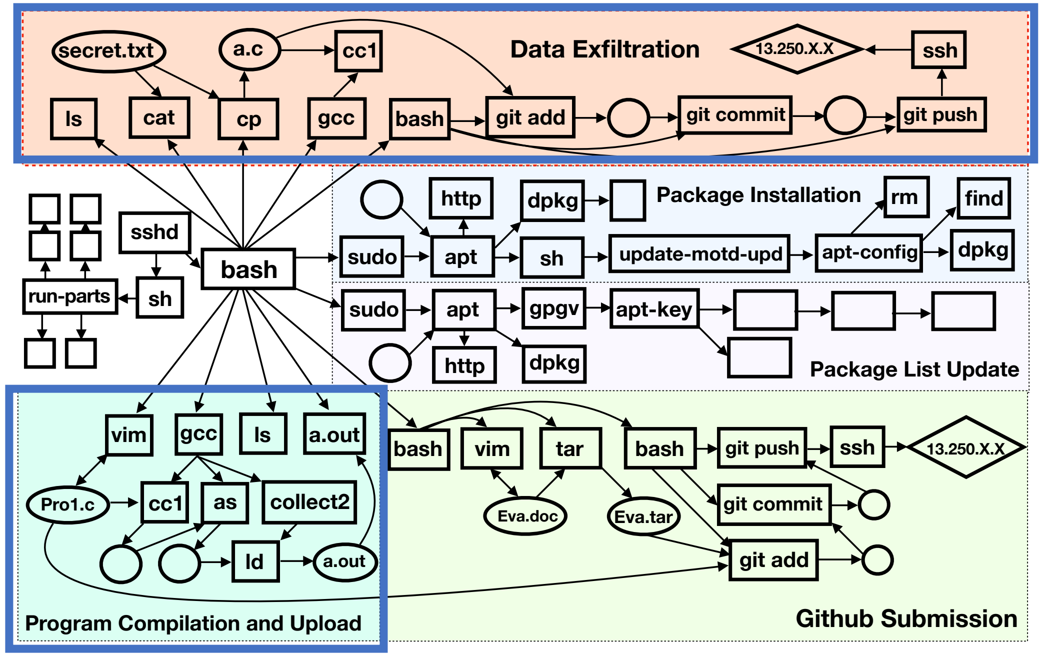
Motivating Example

Attack Scenario: A software tester exfiltrates sensitive data that he has access to



Data Exfiltration Steps

Data Exfiltration



Program Compiling and Upload (cluster)

Motivating Example Logs

Challenges for Behavior Abstraction

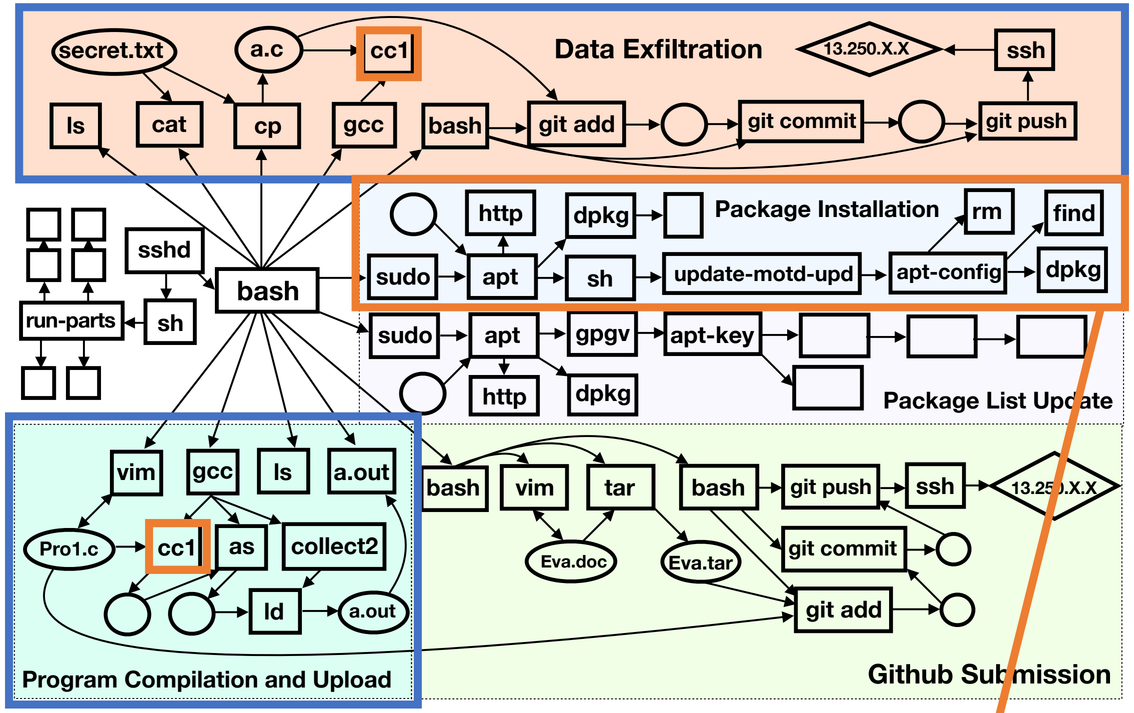
Event Semantics Inference:

- Logs record **general-purpose** system activities but lack knowledge of **high-level semantics**

Individual Behavior Identification:

- The volume of audit logs is **overwhelming**
- Audit events are **highly interleaving**

Data Exfiltration

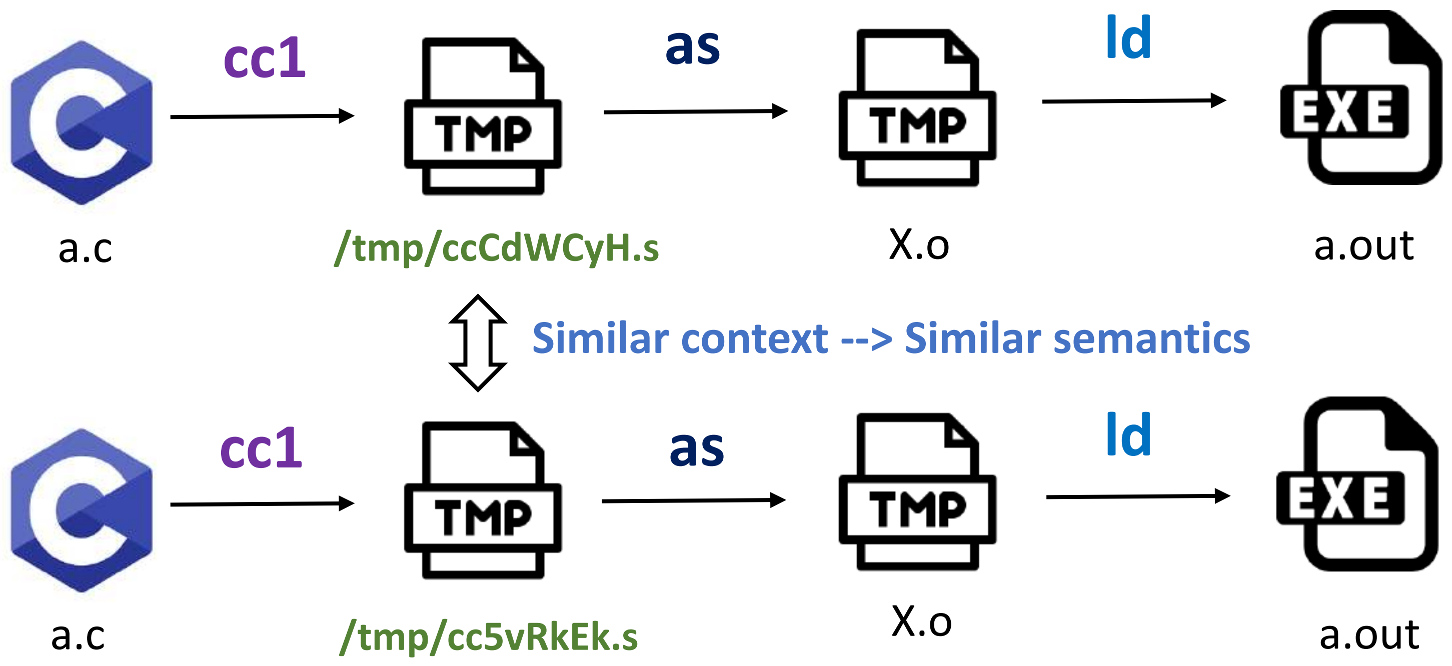


Program Compiling and Upload

Package Installation Events > 50,000

Our Insights

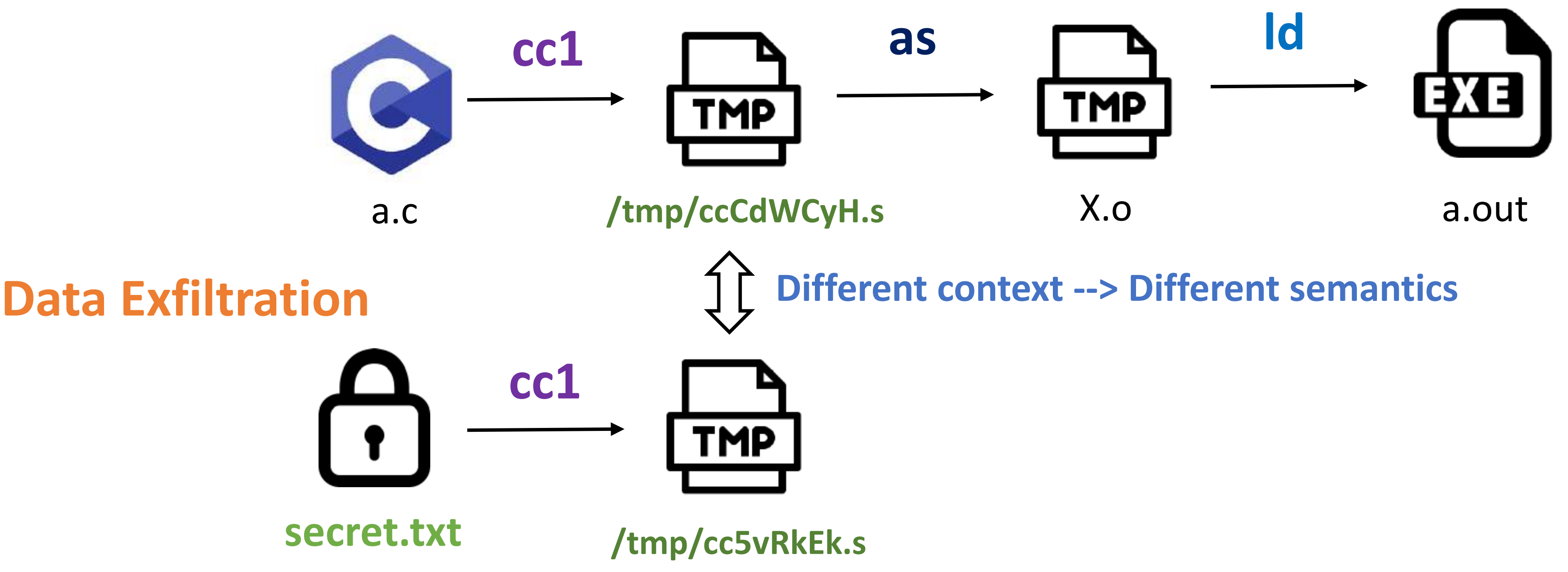
How do analysts manually interpret the semantics of audit events?



Compiling program using GCC

Our Insights

How do analysts manually interpret the semantics of audit events?

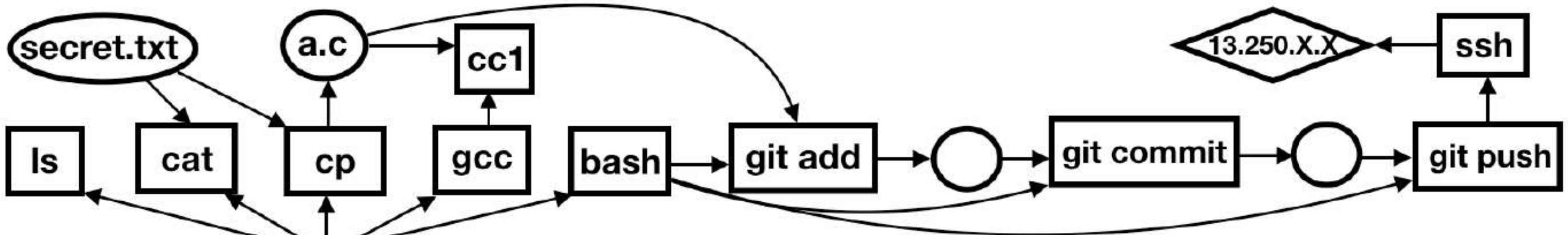


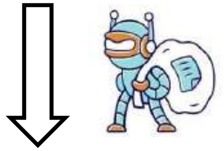
Compiling program using GCC

Reveal the semantics of audit events from their usage **contexts** in logs

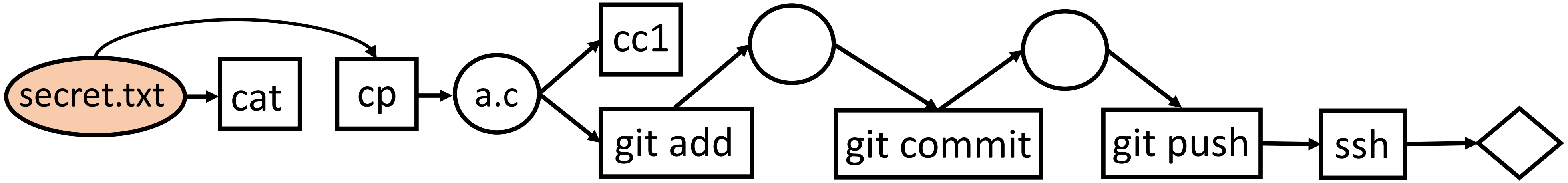
Our Insights

How do analysts manually identify behaviors from audit events?





Forward tracking on secret.txt



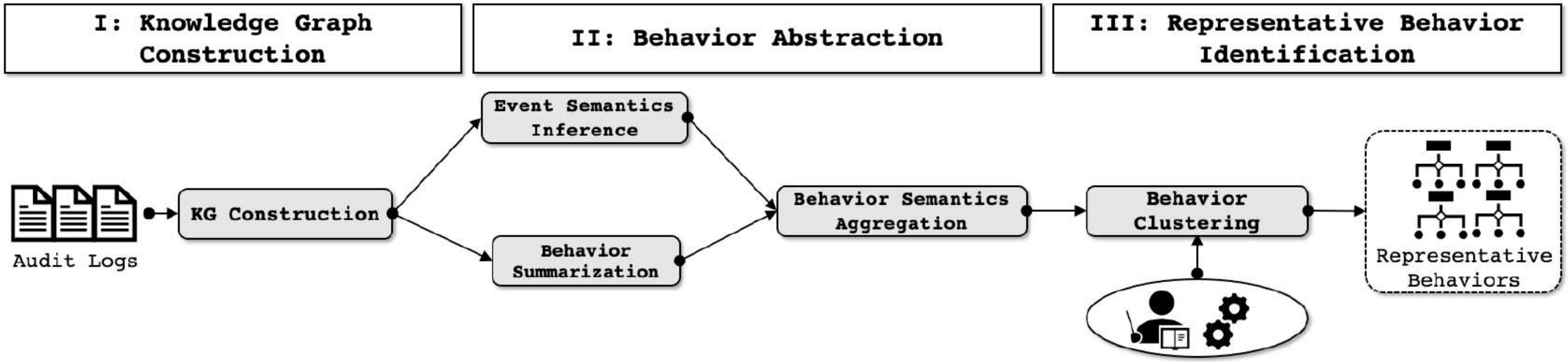
Data Exfiltration Behavior

Summarize behaviors by tracking **information flows** rooted at **data objects**

WATSON

An automated behavior abstraction approach that **aggregates the semantics of audit logs to model behavioral patterns**

- Input: audit logs (e.g., Linux Audit[1])
- Output: representative behaviors



[1] Linux Kernel Audit Subsystem. https://github.com/linux-audit/ audit-kernel

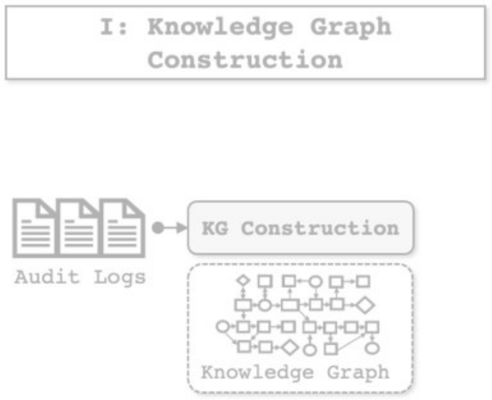
Knowledge Graph Construction

We propose to use a **knowledge graph** (KG) to represent audit logs:

- KG is a directed acyclic graph built upon triples
- Each triple, corresponding to an audit event, consists of three elements (head, relation, and tail):

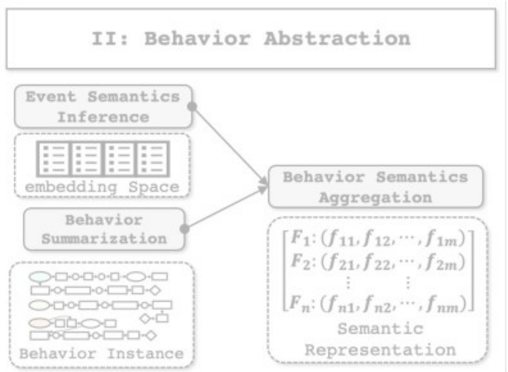
$$\mathcal{KG} = \{(h, r, t) | h, t \in \{Process, File, Socket\}, r \in \{Syscall\}\}$$

- KG unifies **heterogeneous** events in a **homogeneous** manner



Event Semantics Inference

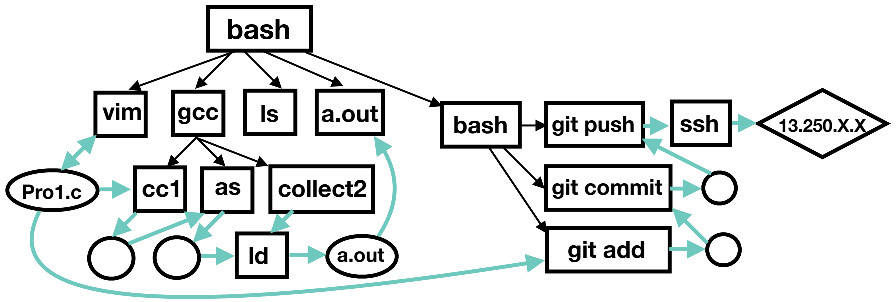
- Suitable **granularity** to capture contextual semantics
  - Prior work [CCS’17] studies log semantics using events as basic units.
  - Lose contextual information within events
  - Working on **Elements** (head, relation, and tail) preserves more contexts
- Employ an embedding model to extract contexts
  - Map elements into a vector space
  - Spatial distance represents semantic similarities
  - **TransE:** a translation-based embedding model
  - **Head + Relation ≈ Tail → Context decides semantics**



Behavior Summarization

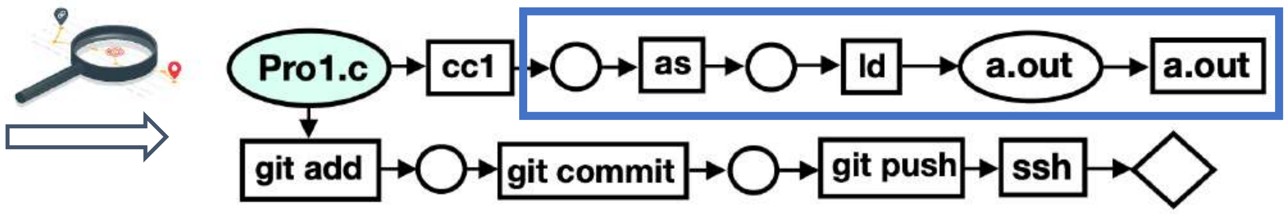
Individual behavior identification: Apply an adapted depth-first search (DFS) to track information flows rooted at a data object:

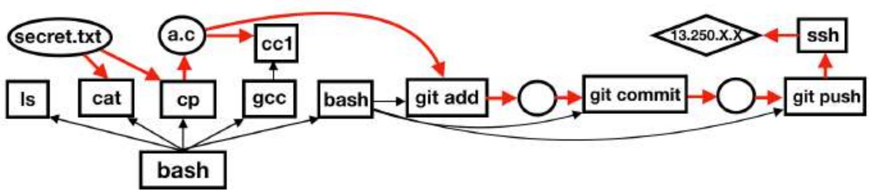
- Perform the DFS on every data object except libraries
- Two behaviors are merged if one is the subset of another



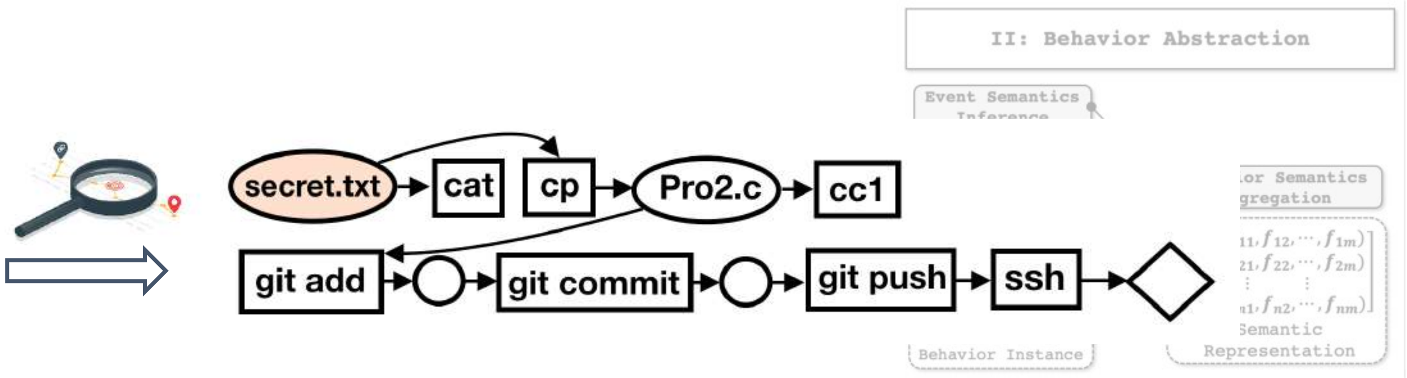
Program Compiling and Upload

Different!



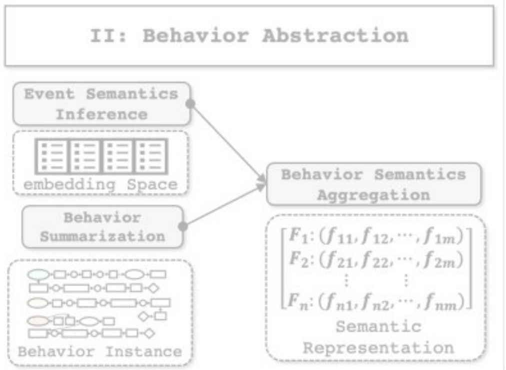


Data Exfiltration



Behavior Semantics Aggregation

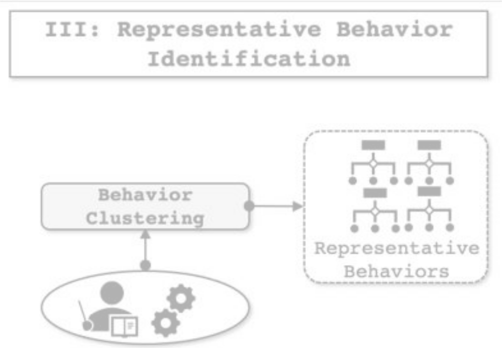
- How to aggregate event semantics to represent behavior semantics?
  - Naïve approach: Add up the semantics of a behavior's constituent events
  - Assumption: audit events equally contribute to behavior semantics 
- **Relative event importance**
  - Observation: behavior-related events are common across behaviors, while behavior-unrelated events the opposite
  - Apply frequency as a metric to define event importance
  - Quantify the frequency: **Inverse Document Frequency (IDF)**
- The presence of **noisy events**
  - Redundant events [CCS'16] & Mundane events



Representative Behavior Identification

- Cluster semantically similar behaviors: **Agglomerative Hierarchical Clustering analysis (HCA)**

- Extract the most representative behaviors
  - Representativeness: Behavior's average similarity with other behaviors in a cluster
  - **Analysis workload reduction**: Do not go through the whole behavior space



Summary

- Logging mechanisms
  - Application-level: Library wrapping / API hooking
  - Kernel-level: Syslogd/klogd, System call interception, Linux security module
  - Virtual Machine Monitor-level: System call interception
- Applications for auditing
  - Intrusion detection, recovery and investigation