CS5231: Systems Security

Lecture 7: Audit Applications

Main Applications (Continue from Last Week)

- Logging-based Applications
  - Intrusion Detection
  - Intrusion Recovery
  - Software Debugging
- One question to think: What to log?
  - Depends on applications
  - Need the right abstraction and amount of information

Binary-Level View of an Incident

Assembly code

```
myfunc:
push    {fp, lr}
add     fp, sp, #4
ldr     r0, helloworld
bl      <puts>
mov     r3, #0
mov     r0, r3
pop     {fp, pc}
```

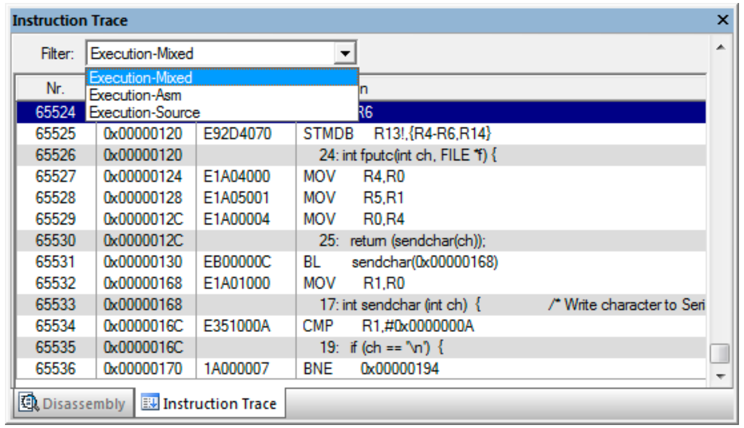
Source code

```
void myfunc (){
  printf("hello world");
}
```

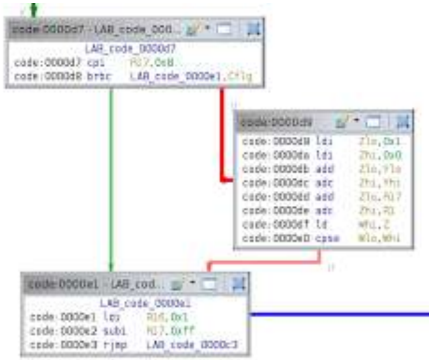
Binary code

```
00 48 2d e9 04 b0 8d e2
0c 00 9f e5 a5 ff ff eb
00 30 a0 e3 03 00 a0 e1
00 88 bd e8 d0 04 01 00
```

Instruction Trace



Control-flow Graph



Audit-Log-Level View

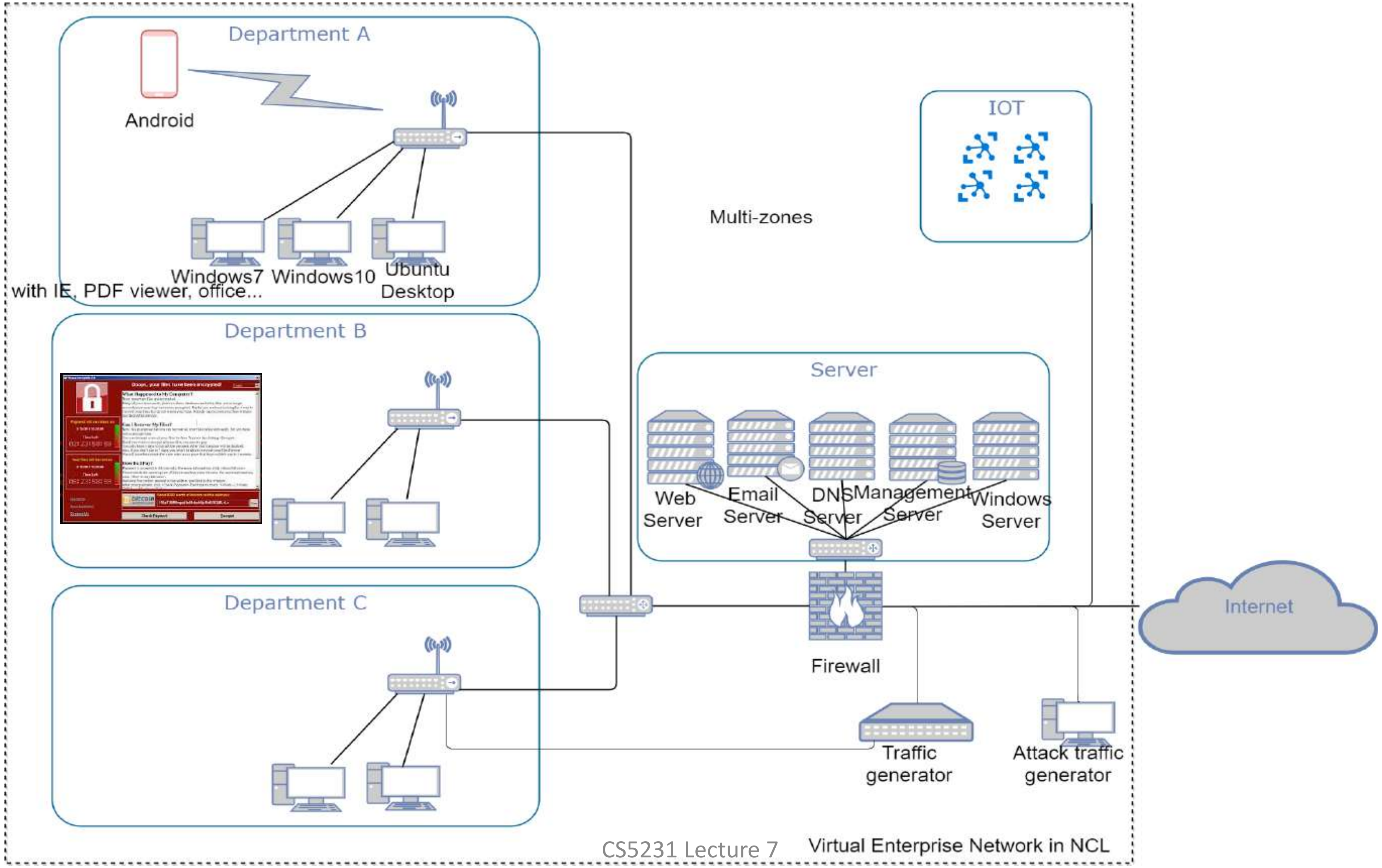
- **User-space** utilities (e.g., Auditd) collect system call records from kernel space through Netlink and write them to a log file under `/var/log/audit`
  - An Example of a read log entry in Auditd

```
----
type=PROCTITLE msg=audit(15/08/2019 14:37:30.522:61916019) : proctitle=sshd: junzeng [priv]
type=SYSCALL msg=audit(15/08/2019 14:37:30.522:61916019) : arch=x86_64 syscall=read
success=yes exit=52 a0=0x3 a1=0x7ffd69eecad0 a2=0x4000 a3=0x7ffd69ef0a60 items=0 ppid=5512
pid=5542 auid=junzeng uid=junzeng gid=junzeng euid=junzeng suid=junzeng fsuid=junzeng
egid=junzeng sgid=junzeng fsgid=junzeng ses=1805 comm=sshd exe=/usr/sbin/sshd key=(null)
----
```

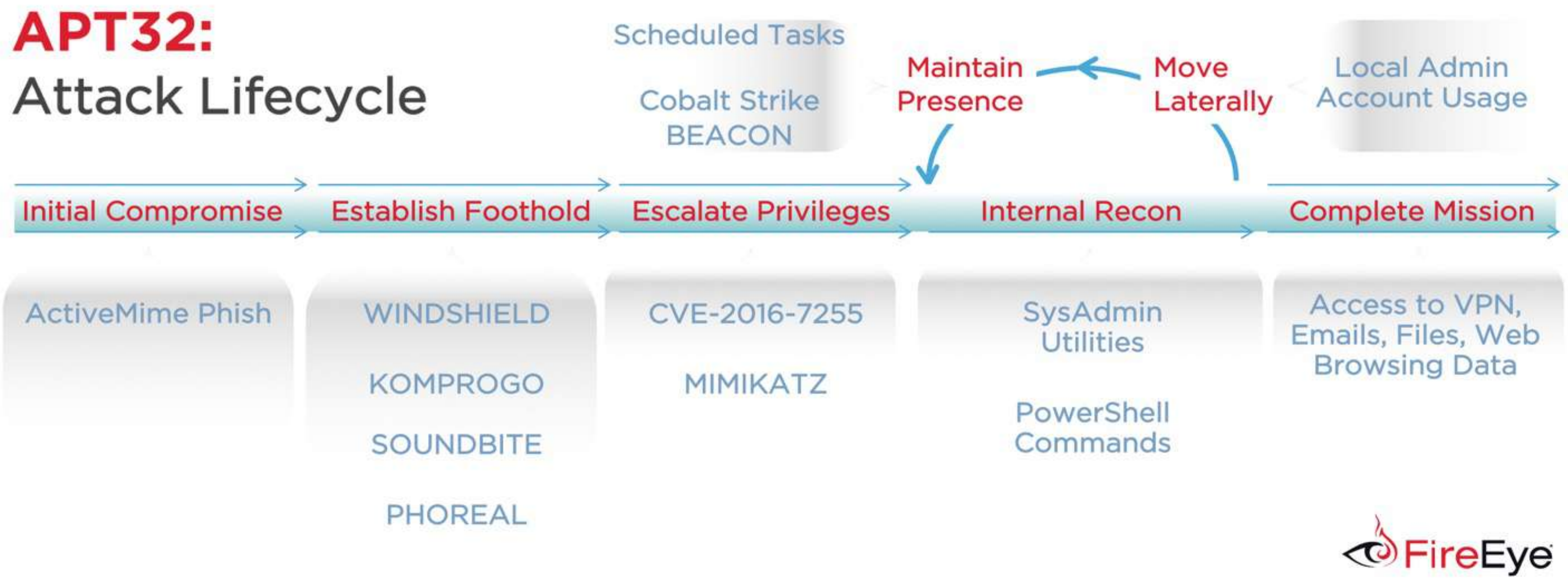
  - An Example of a read log entry in Auditbeat


```
{"@timestamp":"2020-11-04T14:27:14.666Z","@metadata":{"beat":"auditbeat","type":"doc",
"version":"6.8.12"},"auditd":{"sequence":989996,"result":"success","session":"1402","data":
{"a3":"20656c706f657020","tty":"(none)","a2":"1000","arch":"x86_64","syscall":"read",
"exit":"4096","a1":"5583baa77f70","a0":"5"}},"user":{"name_map":{"suid":"root",
"auid":"junzeng","egid":"root","euid":"root","fsuid":"root","gid":"root","sgid":"junzeng",
"fsgid":"root","uid":"root"},"euid":"0","fsgid":"0","fsuid":"0","suid":"0","gid":"0",
"sgid":"1000","egid":"0","auid":"1000","uid":"0"},"process":{"exe":"/usr/sbin/sshd",
"pid":"7959","ppid":"1689","name":"sshd"}}
```

Network and System View



High-level Report

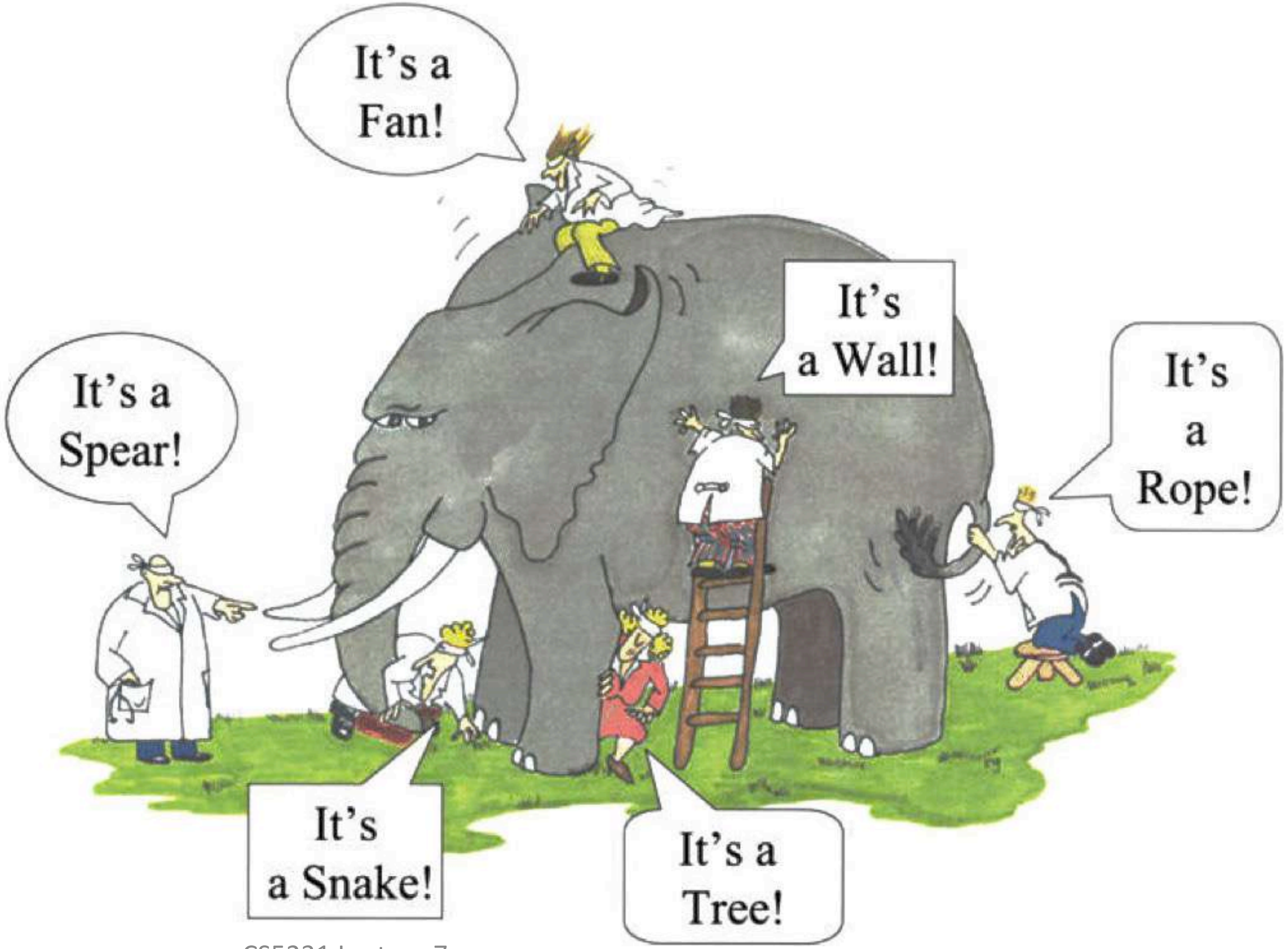


Levels of Understanding of Cyber Security Events

Transaction Level

System-call/Audit Level

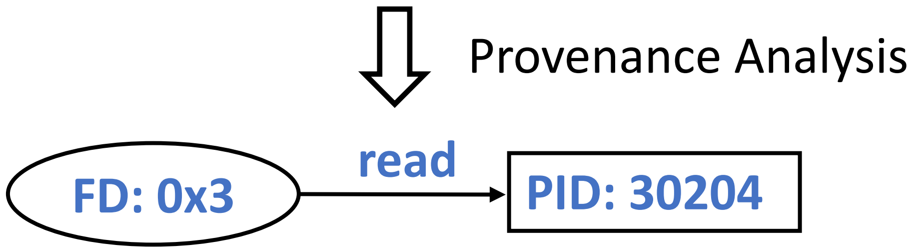
Program/Instruction Level



Building Dependency Graph

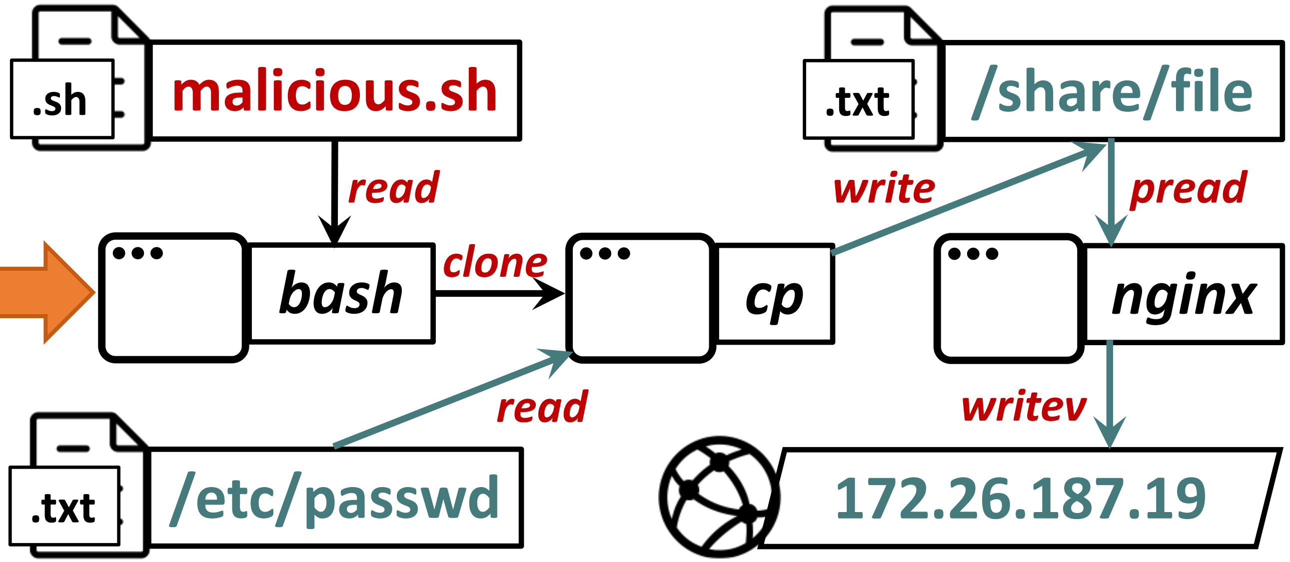
- Nodes
  - Files, processes, sockets
- Edges
  - System calls

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



Provenance Graph: a *representation* of *audit logs*

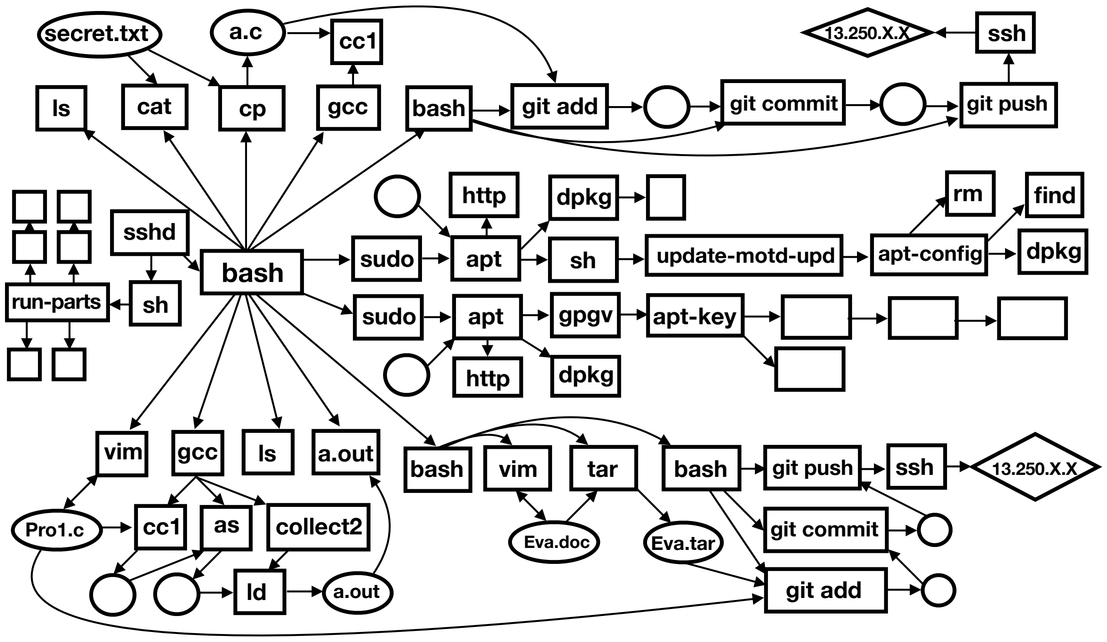
…
1. ***bash***, read, malicious.sh
2. ***bash***, clone, ***cp***
3. ***cp***, read, /etc/passwd
4. ***cp***, write, /share/file
5. ***nginx***, pread, /share/file
6. ***nginx***, writev, 172.26.187.19
…



✓***Provenance Graph*** constructs the ***overall attack scenario*** by ***combining*** historic audit logs***!***

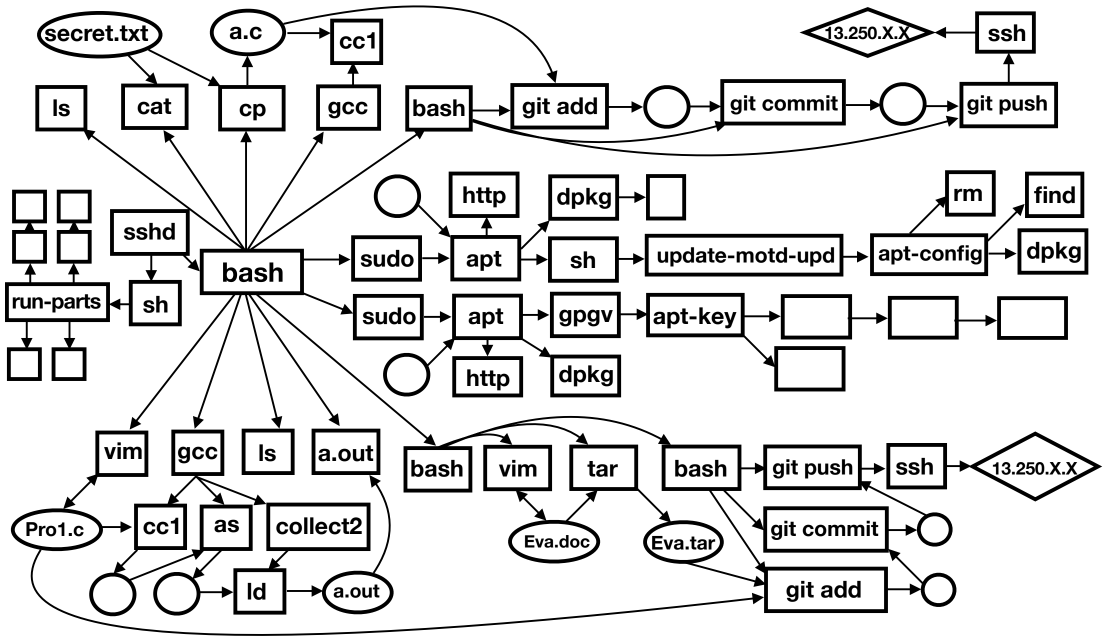
Analysis of Provenance Graph

- Dependency analysis
- Subgraph matching
- Deep learning and recommendation



Analysis of Provenance Graph

- Dependency analysis
- Subgraph matching
- Deep learning and recommendation



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 …]

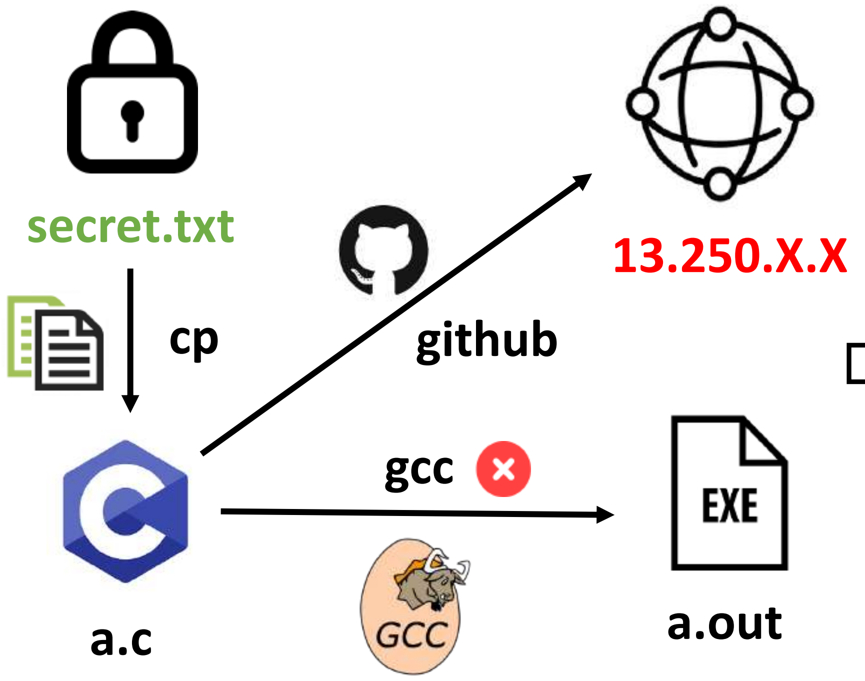
Can we automatically **abstract** high-level behaviors from low-level audit logs and **cluster** semantically similar behaviors before human inspection?

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

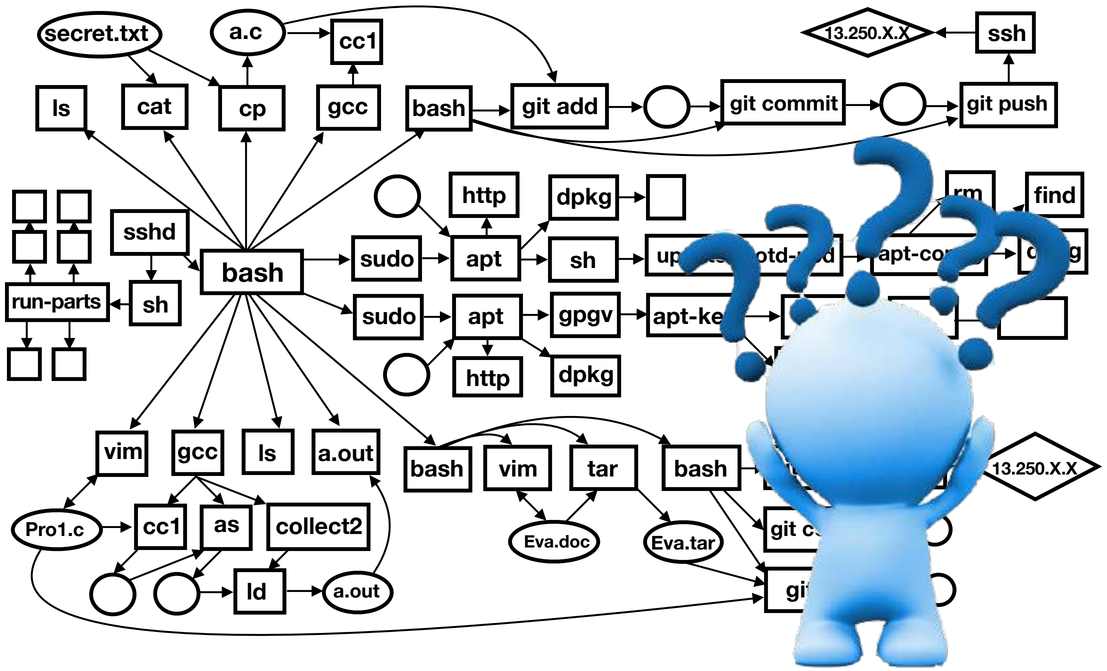
Behavior-specific rules heavily rely on domain knowledge (**time-consuming)**

Motivating Example

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



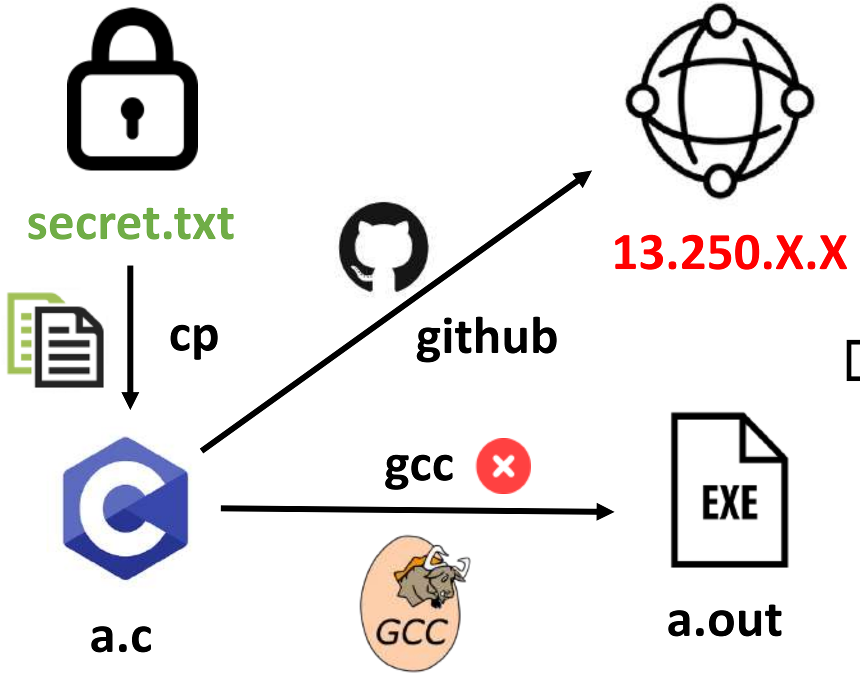
Data Exfiltration Steps



Motivating Example Logs

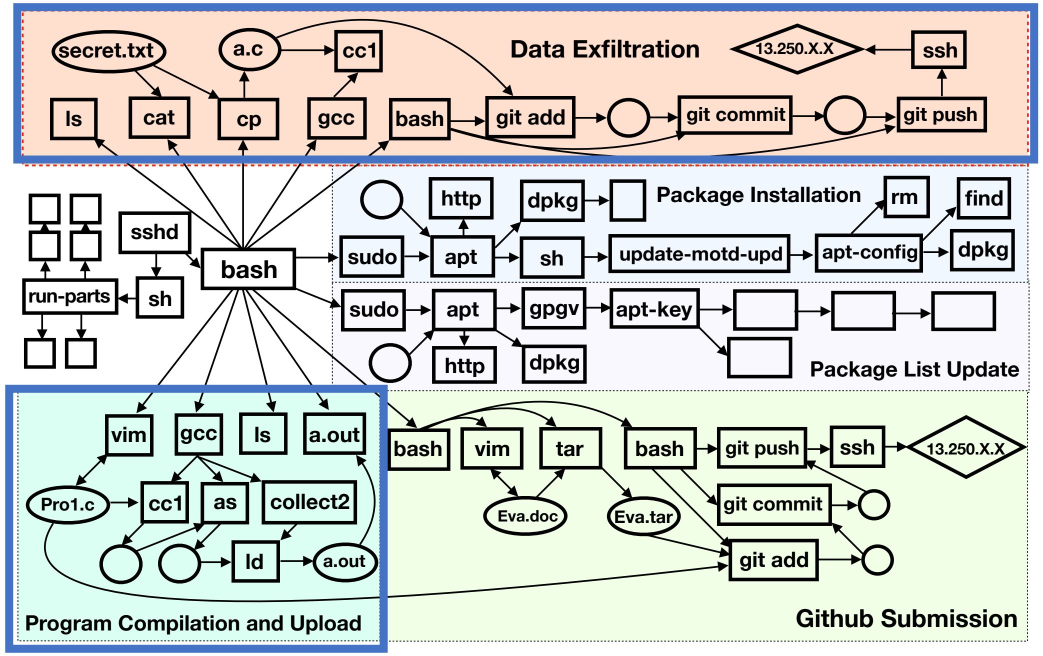
Motivating Example

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



Data Exfiltration Steps

Data Exfiltration

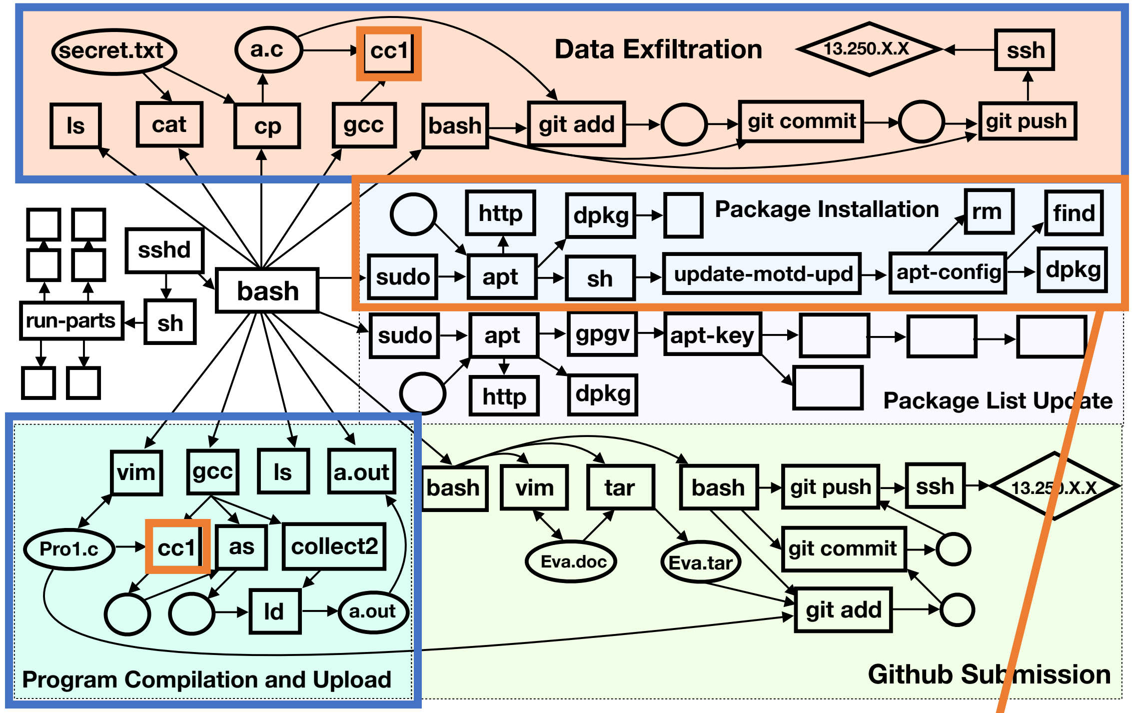


Program Compiling and Upload (cluster)

Motivating Example Logs

Challenges for Behavior Abstraction

Data Exfiltration



Program Compiling and Upload

Package Installation Events > 50,000

Event Semantics Inference:

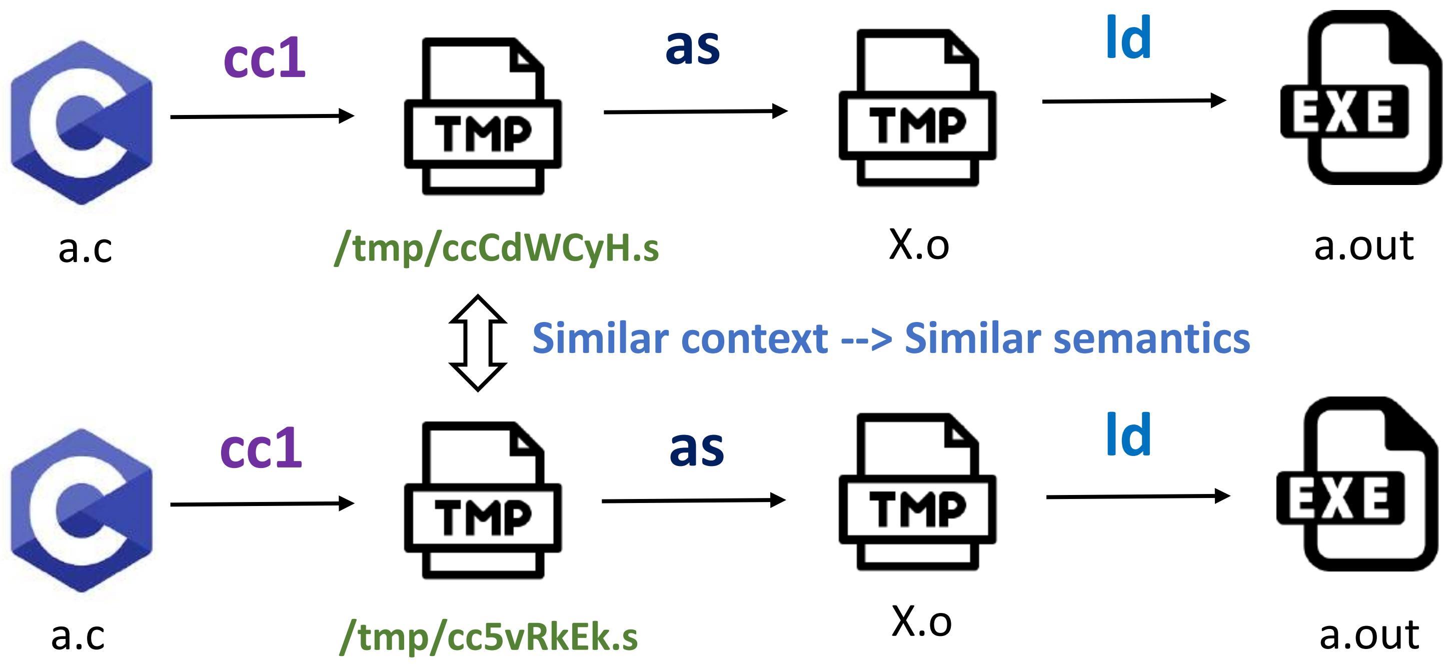
- Logs record **general-purpose** system activities but lack knowledge of **high-level semantics**

Individual Behavior Identification:

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

Our Insights

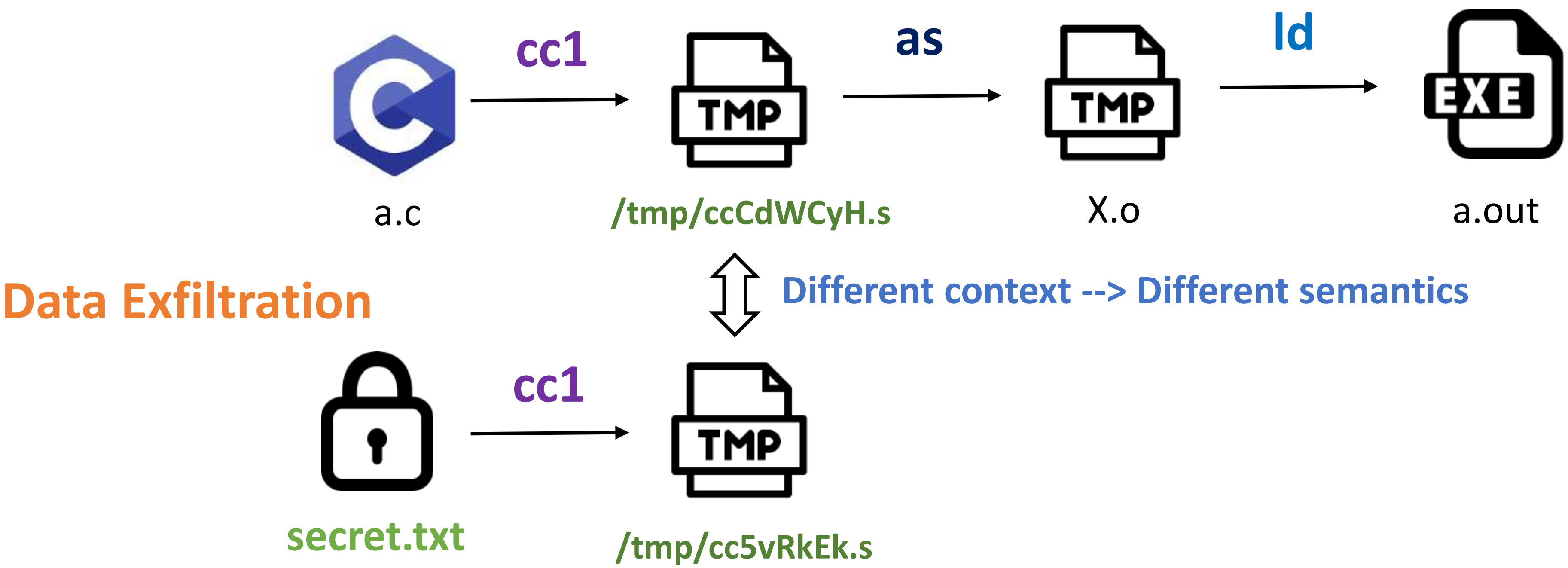
How do analysts manually interpret the semantics of audit events?



Compiling program using GCC

Our Insights

How do analysts manually interpret the semantics of audit events?

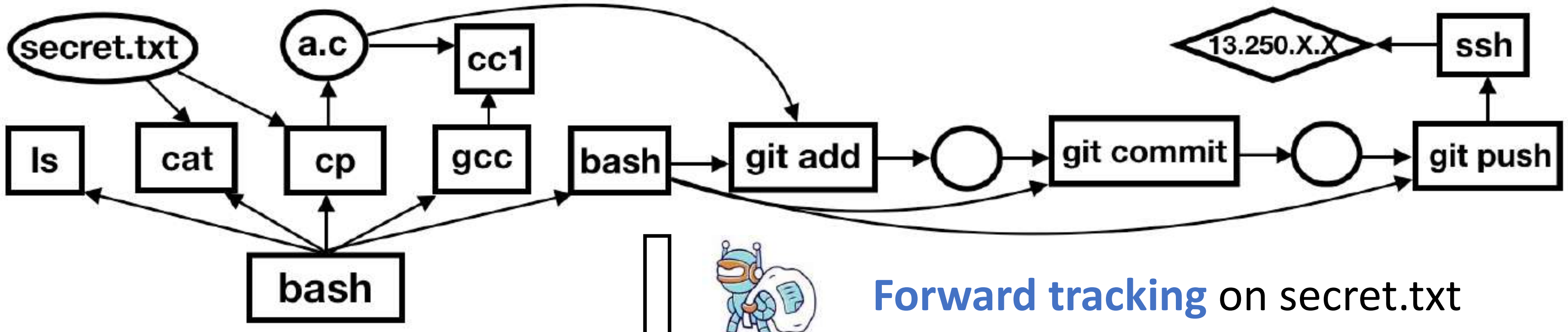


Compiling program using GCC

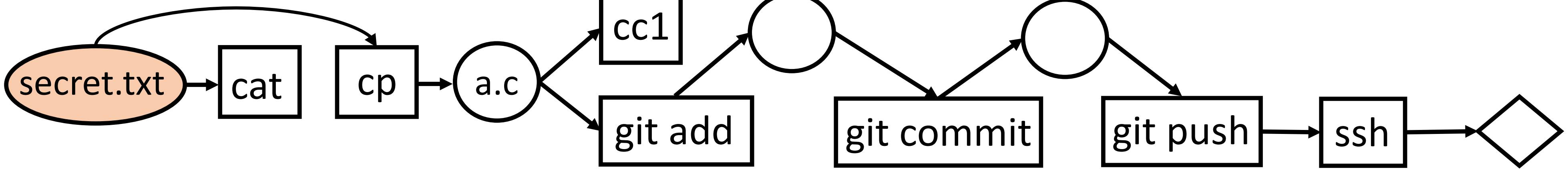
Reveal the semantics of audit events from their usage **contexts** in logs

Our Insights

How do analysts manually identify behaviors from audit events?



Forward tracking on secret.txt



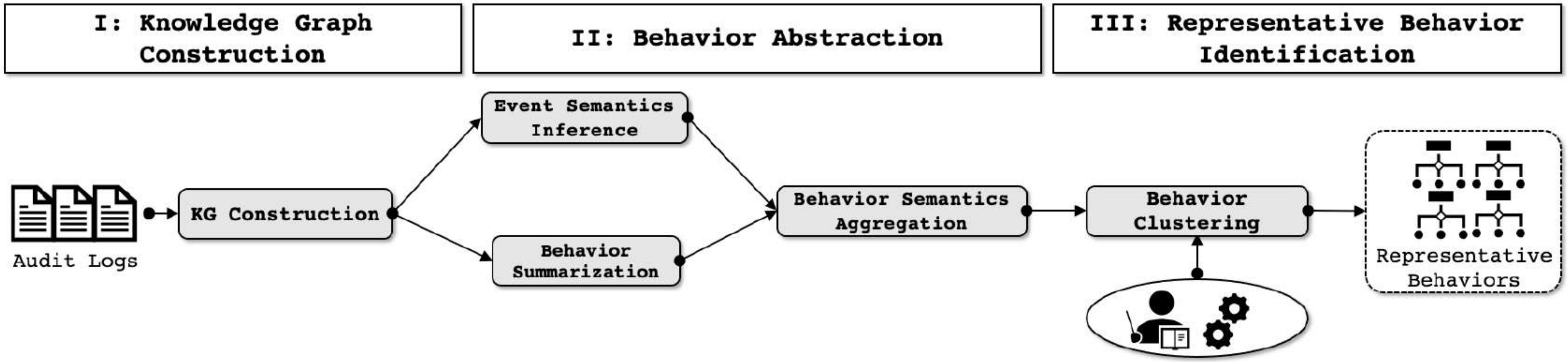
Data Exfiltration Behavior

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

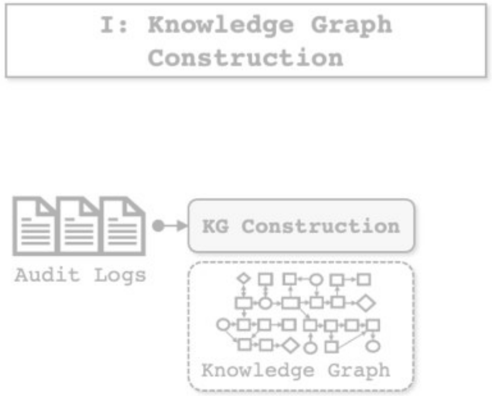
Knowledge Graph Construction

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):

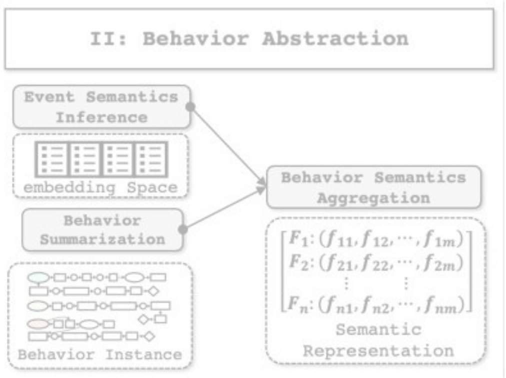
$$\mathcal{KG} = \{(h, r, t) | h, t \in \{Process, File, Socket\}, r \in \{Syscall\}\}$$

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



Event Semantics Inference

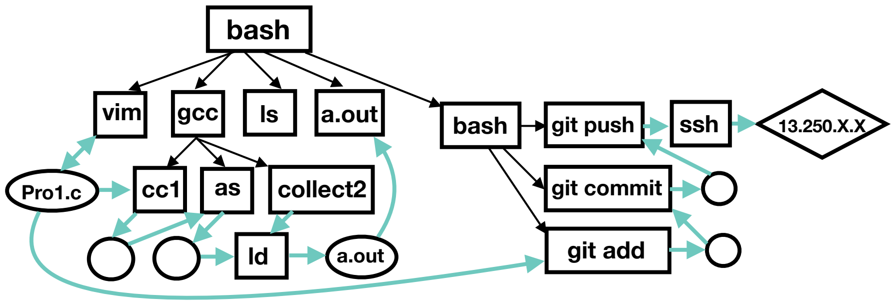
- Suitable **granularity** to capture contextual semantics
  - Prior work [CCS'17] studies log semantics using events as basic units.
  - Lose contextual information within events
  - Working on **Elements** (head, relation, and tail) preserves more contexts
- Employ an embedding model to extract contexts
  - 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

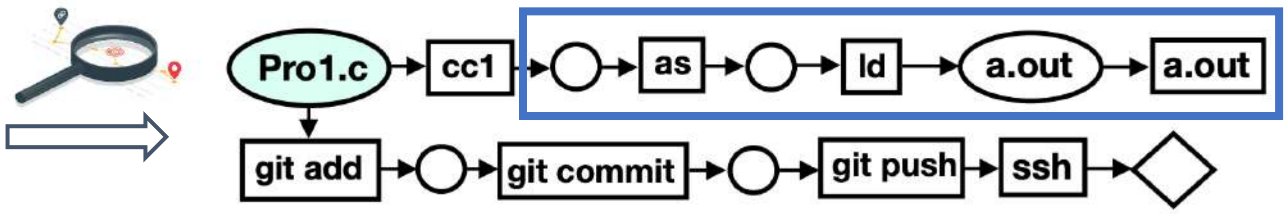
Individual behavior identification: Apply an adapted depth-first search (DFS) to track information flows rooted at a data object:

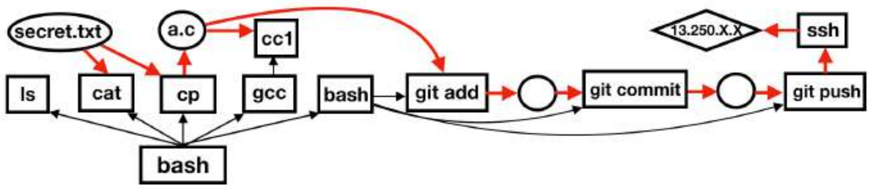
- Perform the DFS on every data object except libraries
- Two behaviors are merged if one is the subset of another



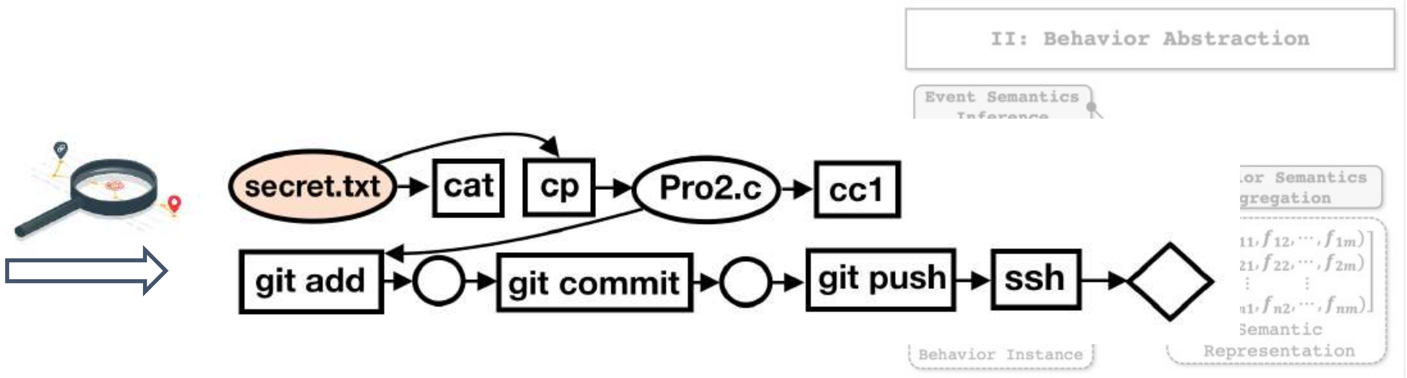
Program Compiling and Upload

Different!



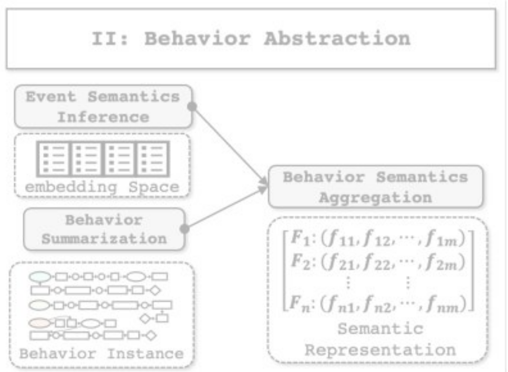


Data Exfiltration



Behavior Semantics Aggregation

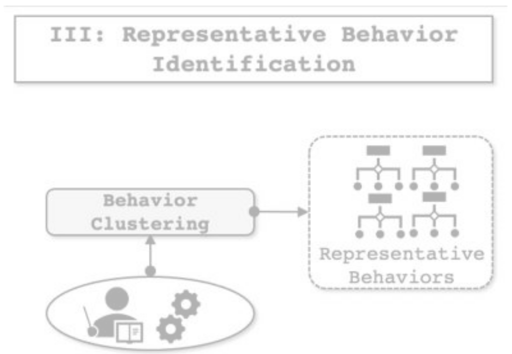
- How to aggregate event semantics to represent behavior semantics?
  - Naïve approach: Add up the semantics of a behavior's constituent events
  - Assumption: audit events equally contribute to behavior semantics 
- **Relative event importance**
  - Observation: behavior-related events are common across 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



Evaluation

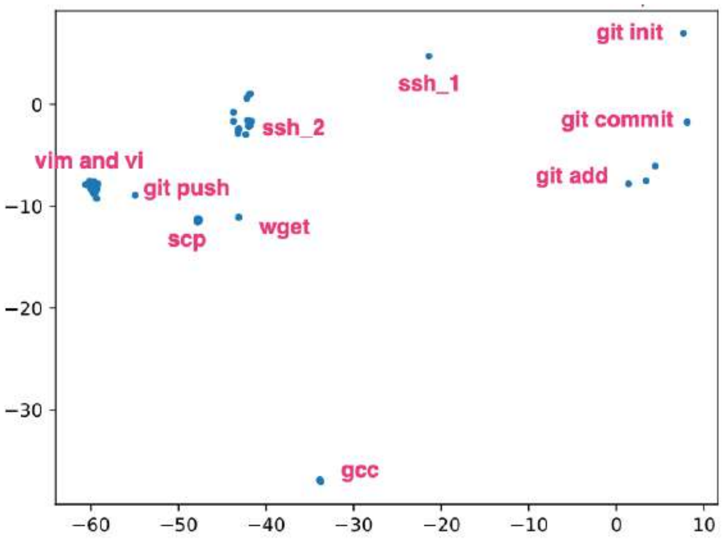
- **Experimental Setup:**
  - Simulated dataset: **275,863,292** events in 4,280 SSH sessions
  - DARPA Trace Dataset[2]: **726,072,596** events in 211 graphs

- **Behavior Abstraction Accuracy:**
  - Can WATSON cluster similar behaviors?
- **Event Semantics Explicability:**
  - Does inferred event semantics match our domain knowledge?
- **Efficacy in Attack Investigation:**
  - How many manual efforts can WATSON save?

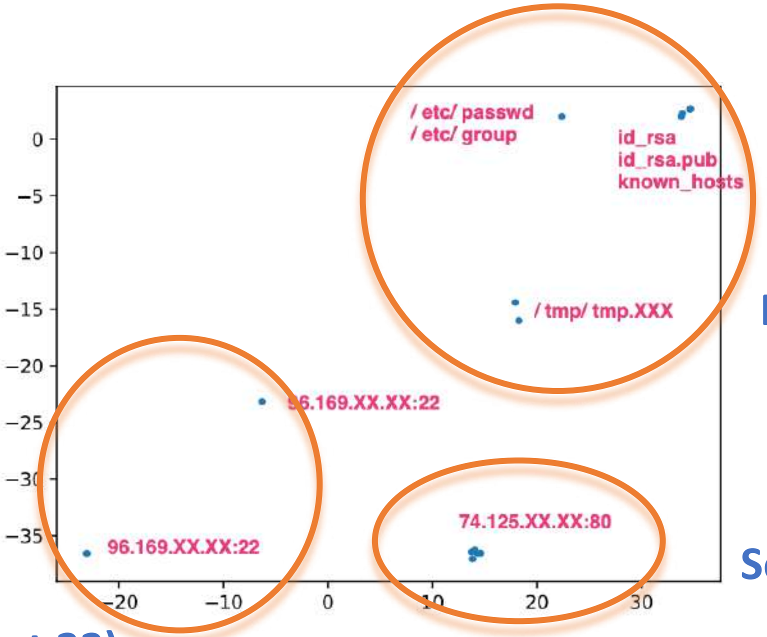
[2] Transparent Computing Engagement 3 Data Release. https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md

Event Semantics Explicability

Use t-SNE to project the embedding space (64 dimensional in our case) into a 2D-plane, giving us an intuition of embedding distribution



(a) 53 Program embeddings



Files

Socket (port 80)

Socket (port 22)

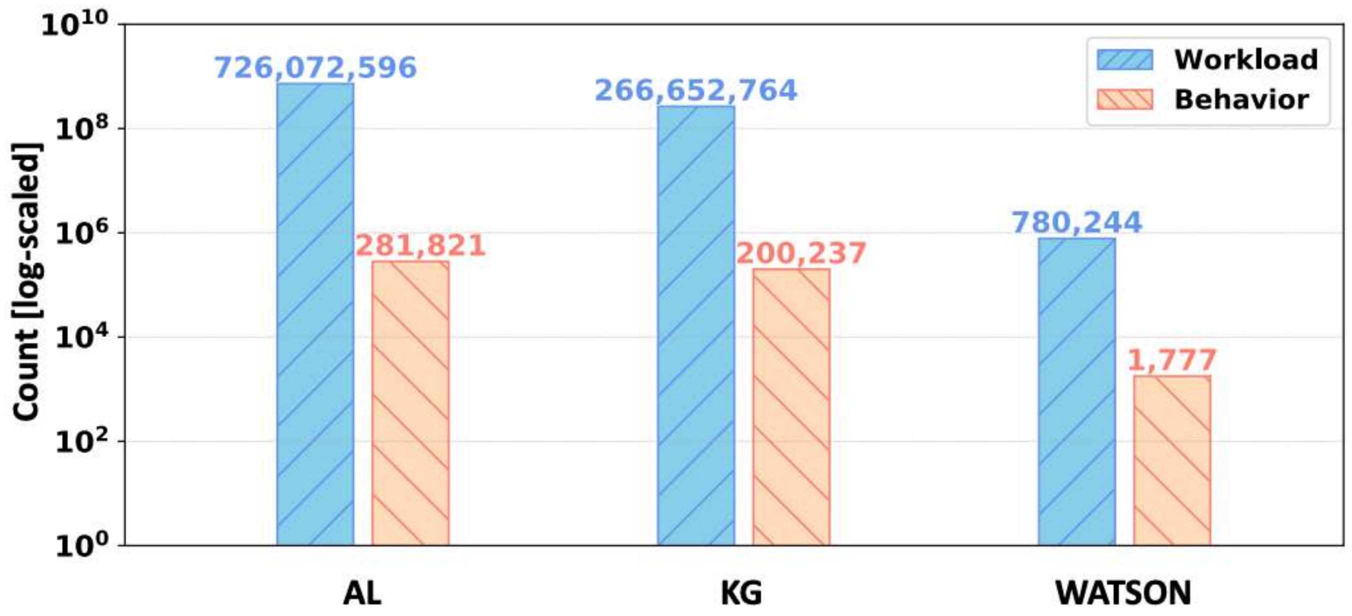
(b) 25 data object embeddings

Semantically similar system entities are **clustered** in the embedding space

Efficacy in Attack Investigation

Measure the **analysis workload reduction** of APT attack investigation in the DARPA TRACE dataset:

- Analysis workload: the number of events to recognize all behaviors



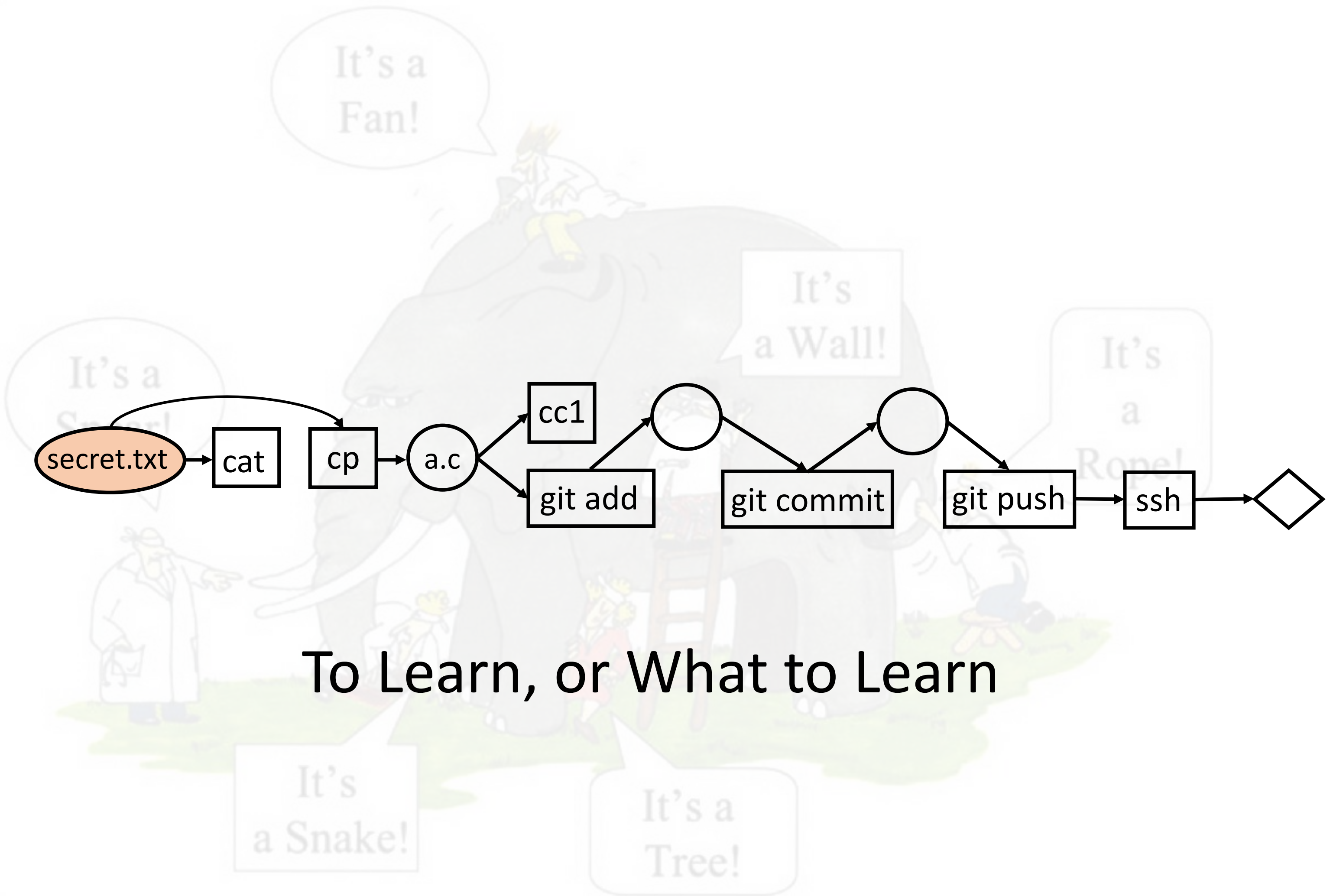
Two orders of magnitude reduction in analysis workload and behaviors

Gaps in Understanding of Cyber Security Events

Transaction Level

System-call/Audit Level

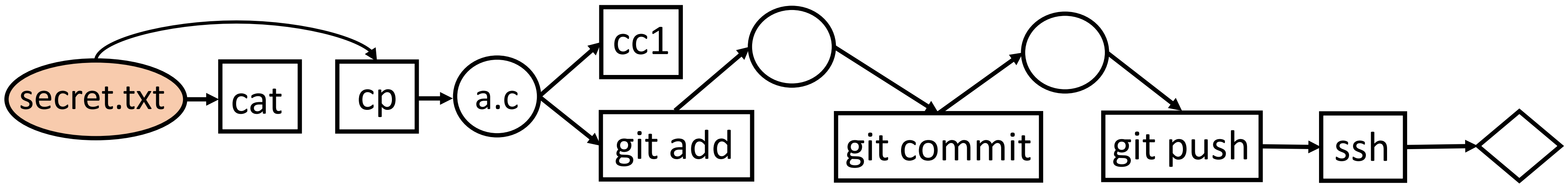
Program/Instruction Level



To Learn, or What to Learn

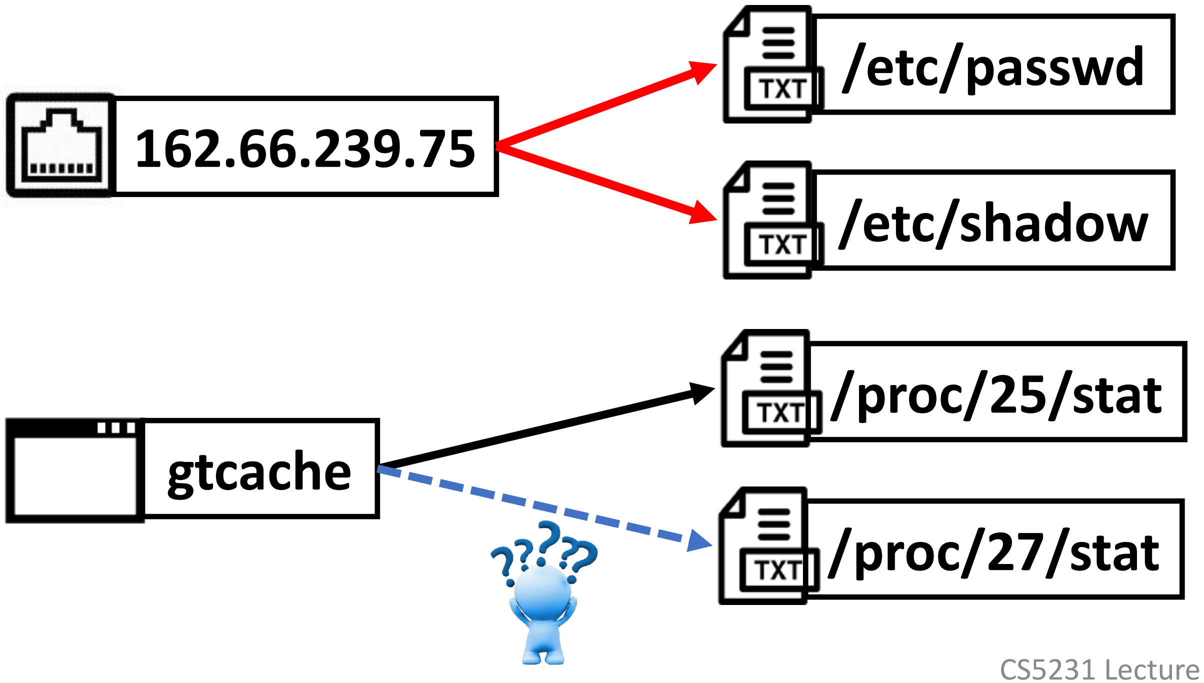
High-order Connection

- Related operations formed the context that defines the "meaning" of a sequence of system behaviors
  - High-order connection in a knowledge graph
  - Basis for many recommendation systems



Our Observation

- Cyber threats can be revealed by determining **how likely** a system entity would **interact** with another entity
  - Unlikely (or “Unintended”) interactions indicate cyber threats
  - Estimate such likelihood with **historical** system entity interactions



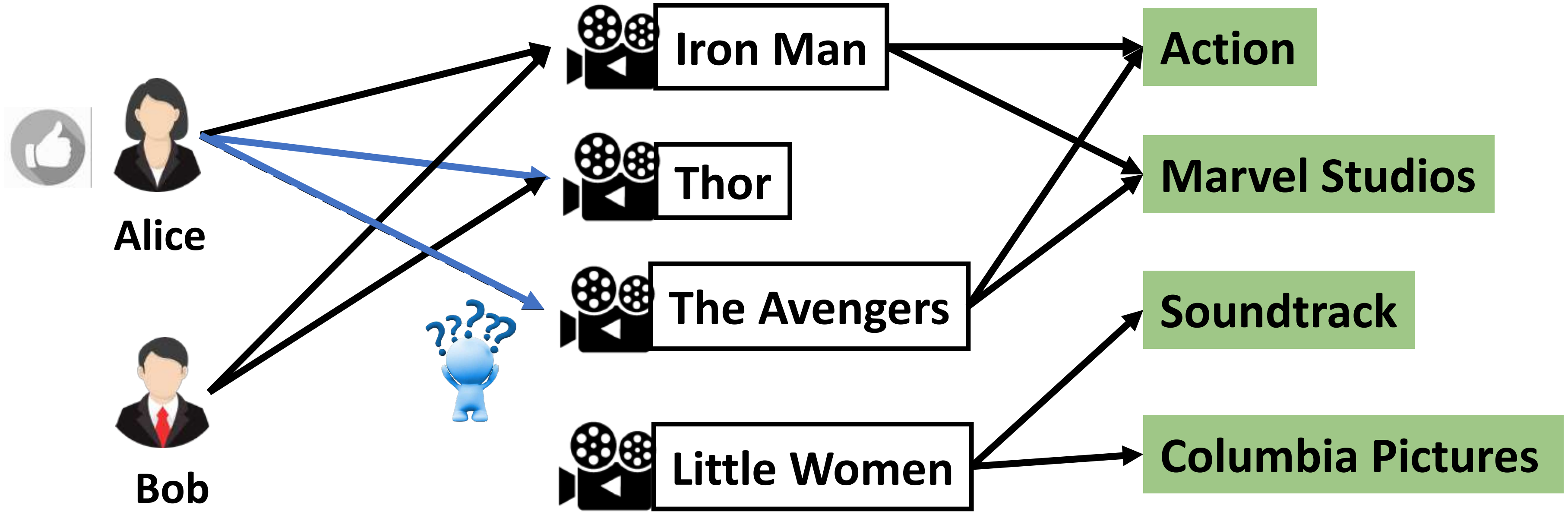
Sensitive files normally **do not** interact with public networks!

Should gtcache interact with /proc/27/stat? **Yes!**

Recommendation as a Similar Problem

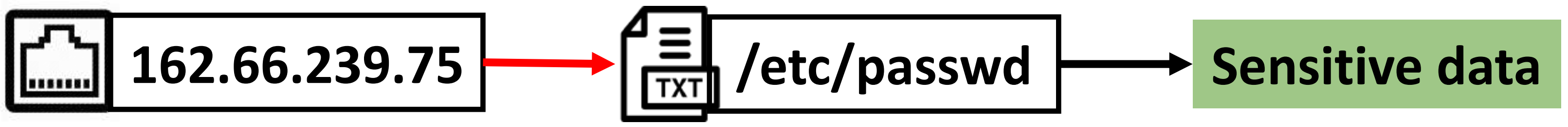
A similar problem has been explored in **Recommendation Systems**:

- Determine **how likely** a user would **interact** with an item
- **Similar** users share preferences on items: **historical** user-item interactions
- Item side information forms **high-order connectivity** that links **similar** items

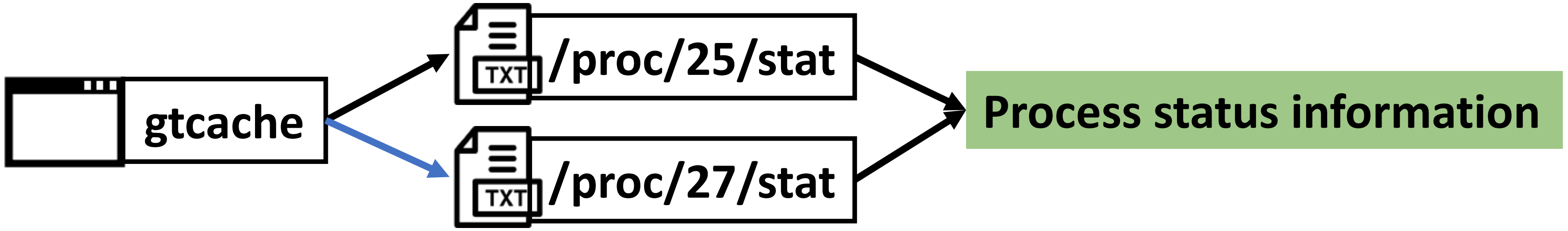


Recommendation-guided Cyber Threat Analysis

Observation: Similar system entities share preferences on interactions



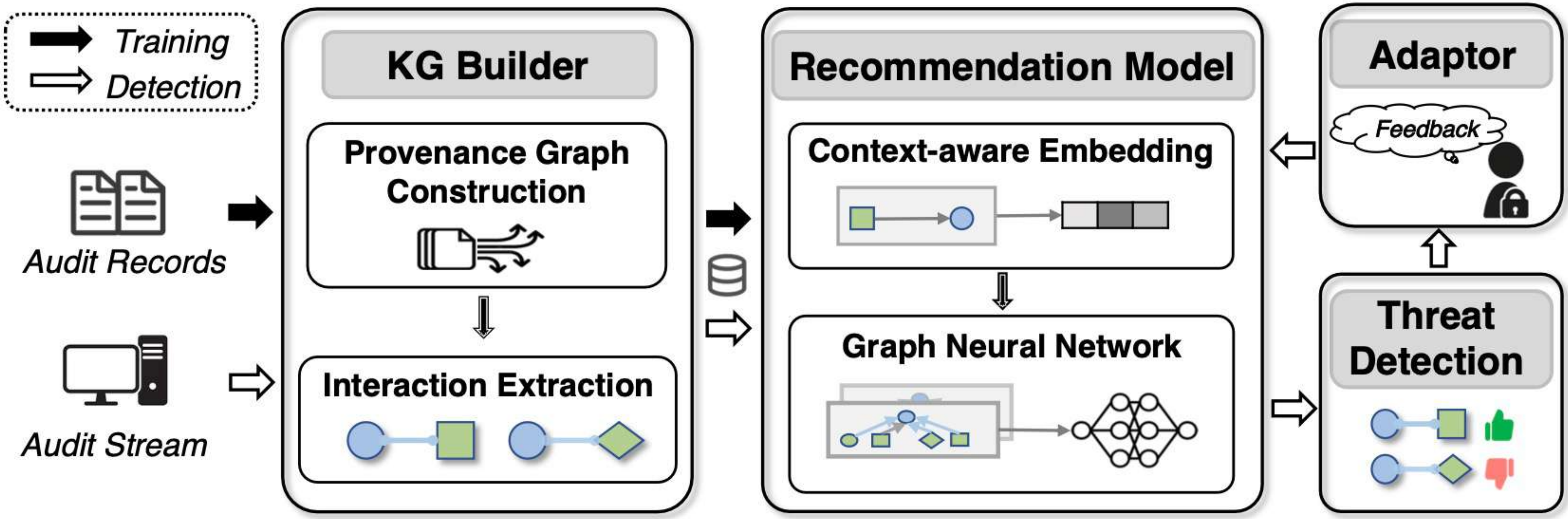
Insight: Identify high-order connectivity using side information of system entities to better uncover their similarities



We formulate cyber threat analysis as a recommendation task:

<u>**How likely** a system entity would **"prefer "** its interactive entities?</u>

ShadeWatcher: “Recommending” Malicious Behaviors



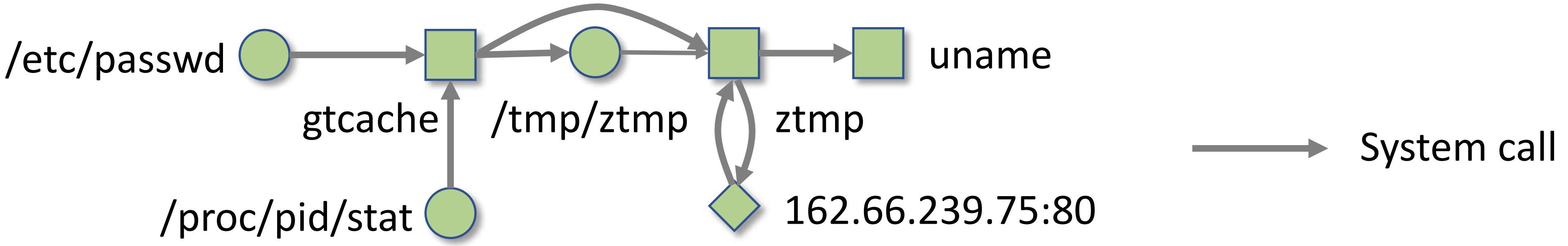
审堂下之阴，而知日月之行，阴阳之变

Sensing the movement of Sun and Moon from their shades [0]

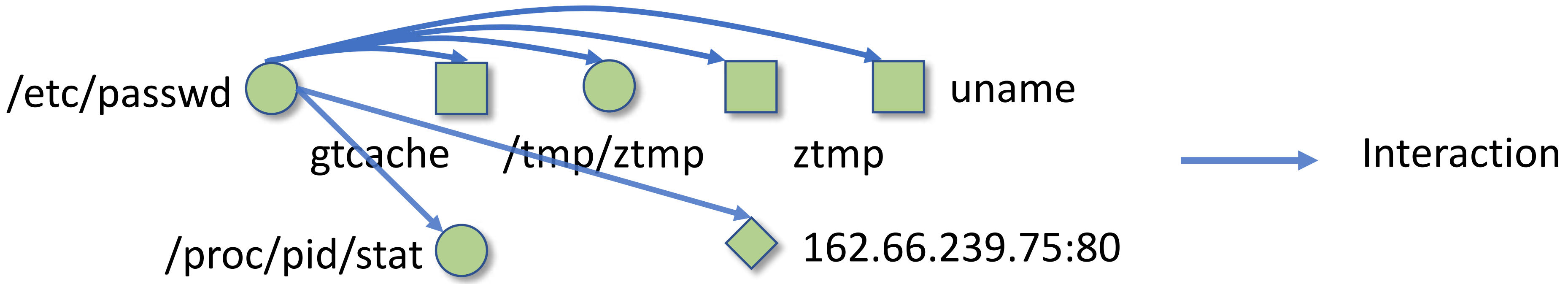
[0] Buwei Lv. *Master Lv’s Spring and Autumn Annals.* 239 BC

Knowledge Graph Builder

- Given audit records on end hosts, we parse them into a **provenance graph (PG)** and extract system entity interactions into a **bipartite graph (BG)**.



Provenance Graph

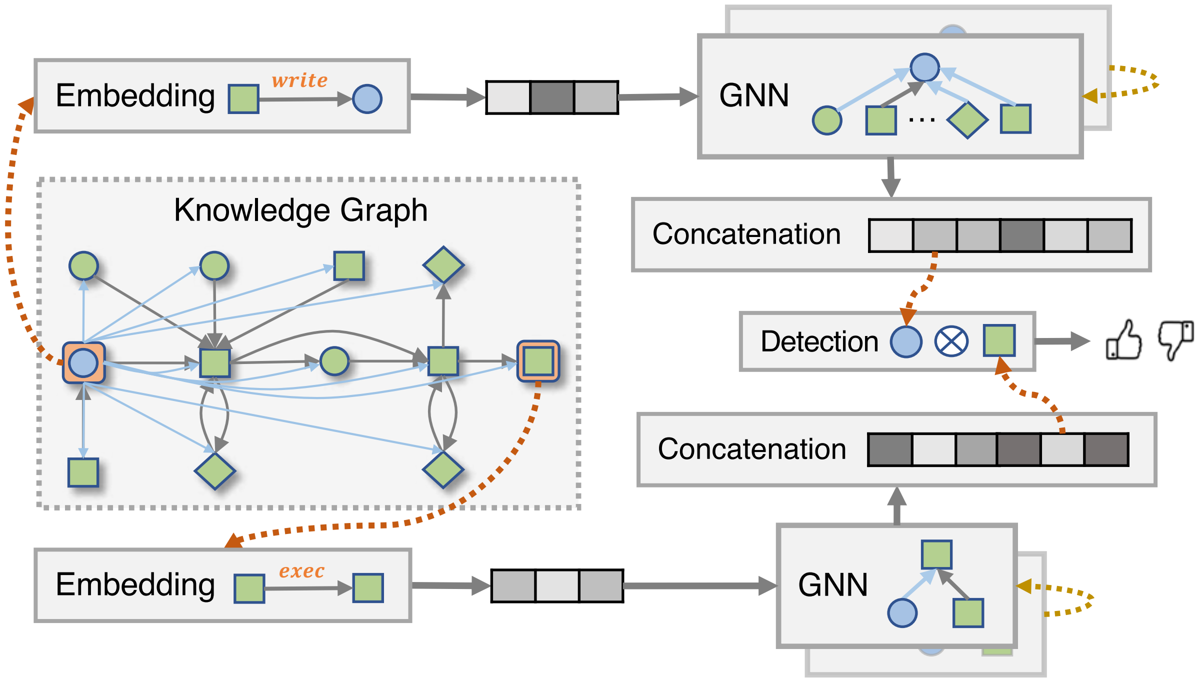


Bipartite Graph

Recommendation Model

Key Idea: use **different-order** connectivities in a KG to model the **likelihood** of system entity interactions, identifying anomalous ones as cyber threats

- Model first-order connectivity to parameterize system entities as embeddings (i.e., vectors)
- Model higher-order connectivity by propagating embeddings from neighbors via GNNs
- Classify system entity interactions into normal and anomalous



Evaluation

- **Experimental datasets:**
  - **Six real-world cyber-attacks** simulated in a testbed environment:

    Configuration Leakage, Content Destruction, Cheating Student, Illegal Storage, Passwd Gzip Scp, and Passwd Reuse
  - **Four APT attacks** from the DARPA Transparent Computing (TC) dataset

    Extension Backdoor, Firefox Backdoor, Pine Backdoor, and Phishing Executable
- **Evaluation aspects:**
  - How **effective** is SHADEWATCHER as a threat detection system?
  - To what extend do first-order and high-order information **facilitate** analysis?
  - How **efficient** is SHADEWATCHER in deployment?

Study of Recommendation-guided Analysis

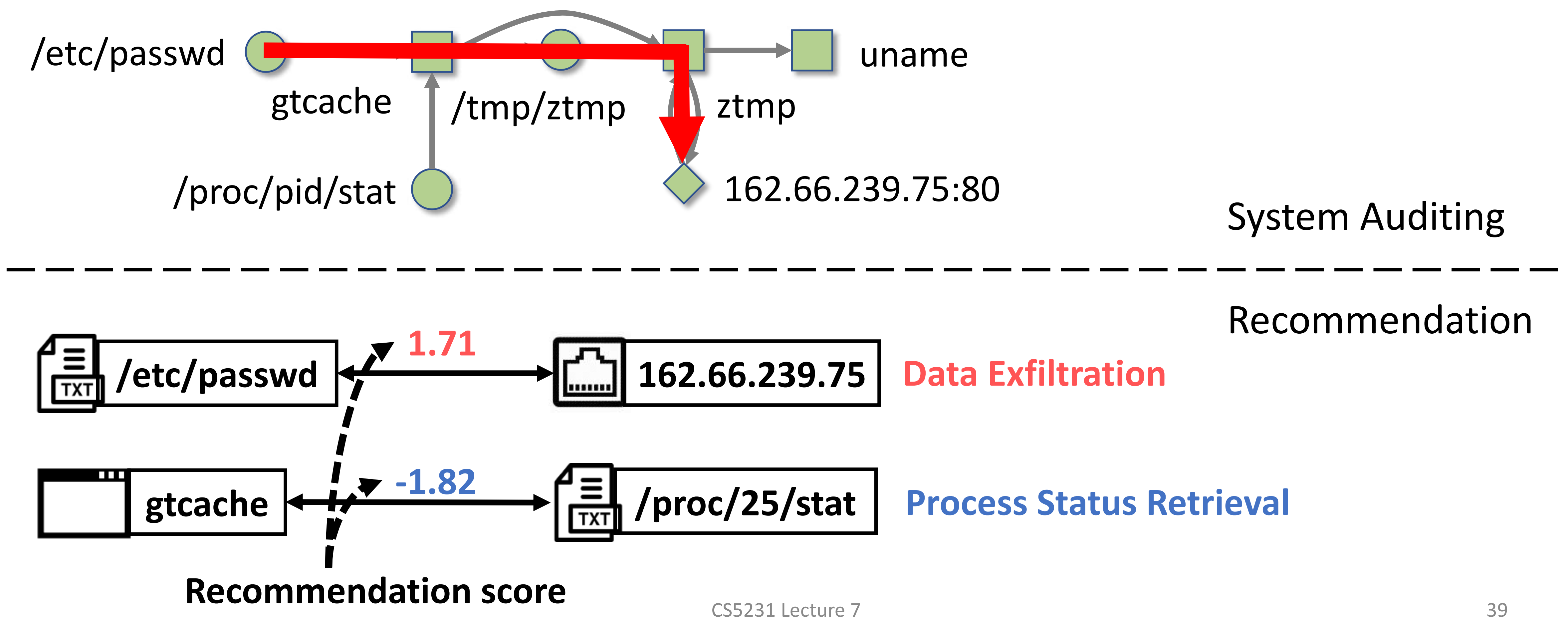
- Compare different KG embedding algorithms
- Study the importance of high-order information propagated by GNNs

KG Embedding	One-hot	TransE	TransH	TransR	TransR
GNN	✓	✓	✓	✗	✓
AUC Value	0.966	0.971	0.974	0.763	0.996

SHADEWATCHER (TransR + GNN) achieves the best performance (AUC):
- High-order information is **beneficial** to cyber threat analysis
- It is important to **distinguish** semantics under different relation contexts

Case Study

Extension Backdoor from the DARPA TC Dataset



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
  - Research work: Watson, ShadeWatcher