

Multi-level Observation and Understanding of Program Behaviors

Liang Zhenkai 梁振凯



Security Incidents Are on The Rise

TECH | TWITTER | CYBERSECURITY

Months la boggles m

Some of the biggest
By Jay Peters | @jaypeters | D

US & WORLD | TECH | HEA

1.5 millic Singapo

Local media say
By James Vincent | Jul 20



SINGHEALTH

WAS TAKEN?
RIC NUMBER,
GENDER, RACE
OF BIRTH OF
ON PATIENTS
OF OUTPATIENT
DISPENSED TO
000 PATIENTS

WAS NOT TAKEN?
ND TEST RESULTS
RS' NOTES
RE NOT AMENDED
DELETED

CHANNEL NEWSASIA

What happened? **Who** is affected? **How** to prevent?

Binary-Level View

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");  
}
```

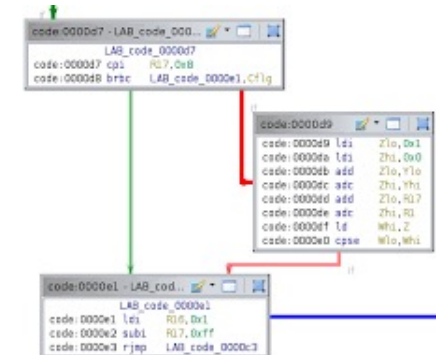
Binary code

```
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

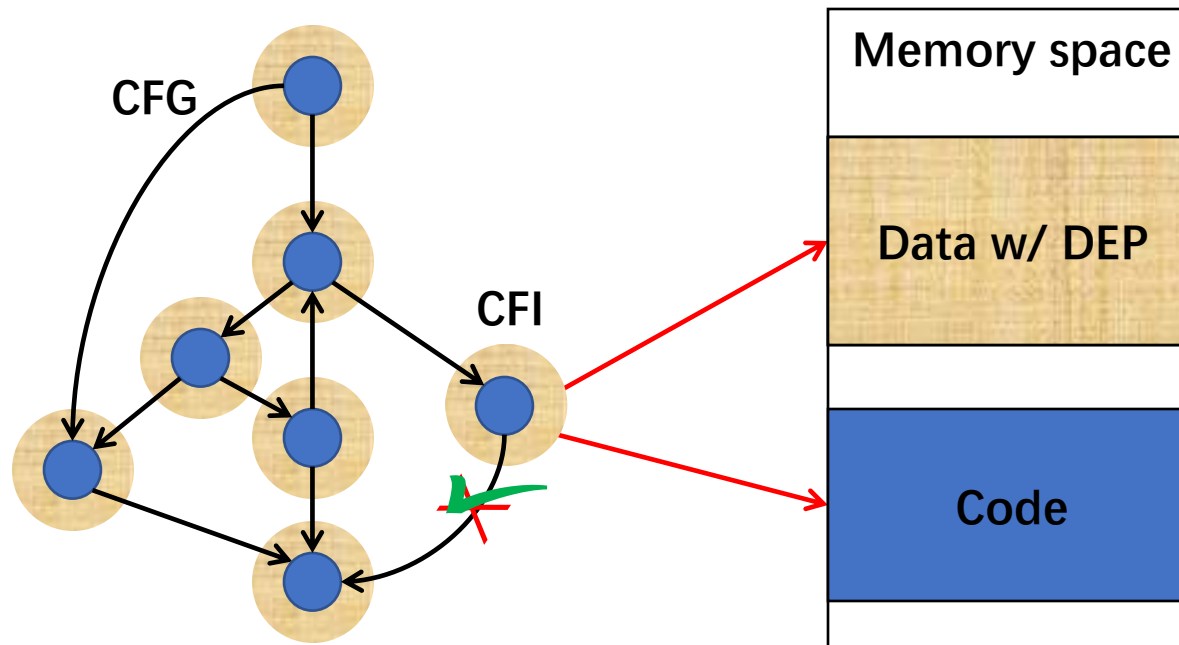
Nr.	Address	Disassembly	Source
65525	0x00000120	E92D4070 STMDB R13!,{R4-R6,R14}	
65526	0x00000120		24: int fputs(int ch, FILE *f) {
65527	0x00000124	E1A04000 MOV R4,R0	
65528	0x00000128	E1A05001 MOV R5,R1	
65529	0x0000012C	E1A00004 MOV R0,R4	
65530	0x0000012C		25: return (sendchar(ch));
65531	0x00000130	EB00000C BL sendchar(0x00000168)	
65532	0x00000168	E1A01000 MOV R1,R0	
65533	0x00000168		17: int sendchar(int ch) { /* Write character to Ser
65534	0x0000016C	E351000A CMP R1,#0x0000000A	
65535	0x0000016C		19: if (ch == '\n') {
65536	0x00000170	1A000007 BNE 0x00000194	

Control-flow Graph



Binary-Level Vulnerability, Attacks and Defenses

- Code injection ← Data Execution Prevention
- Code reuse ← Control Flow Integrity
 - return-to-libc
 - return-oriented programming (ROP)
- Data-oriented Programming (DOP)



Audit-Log-Level View

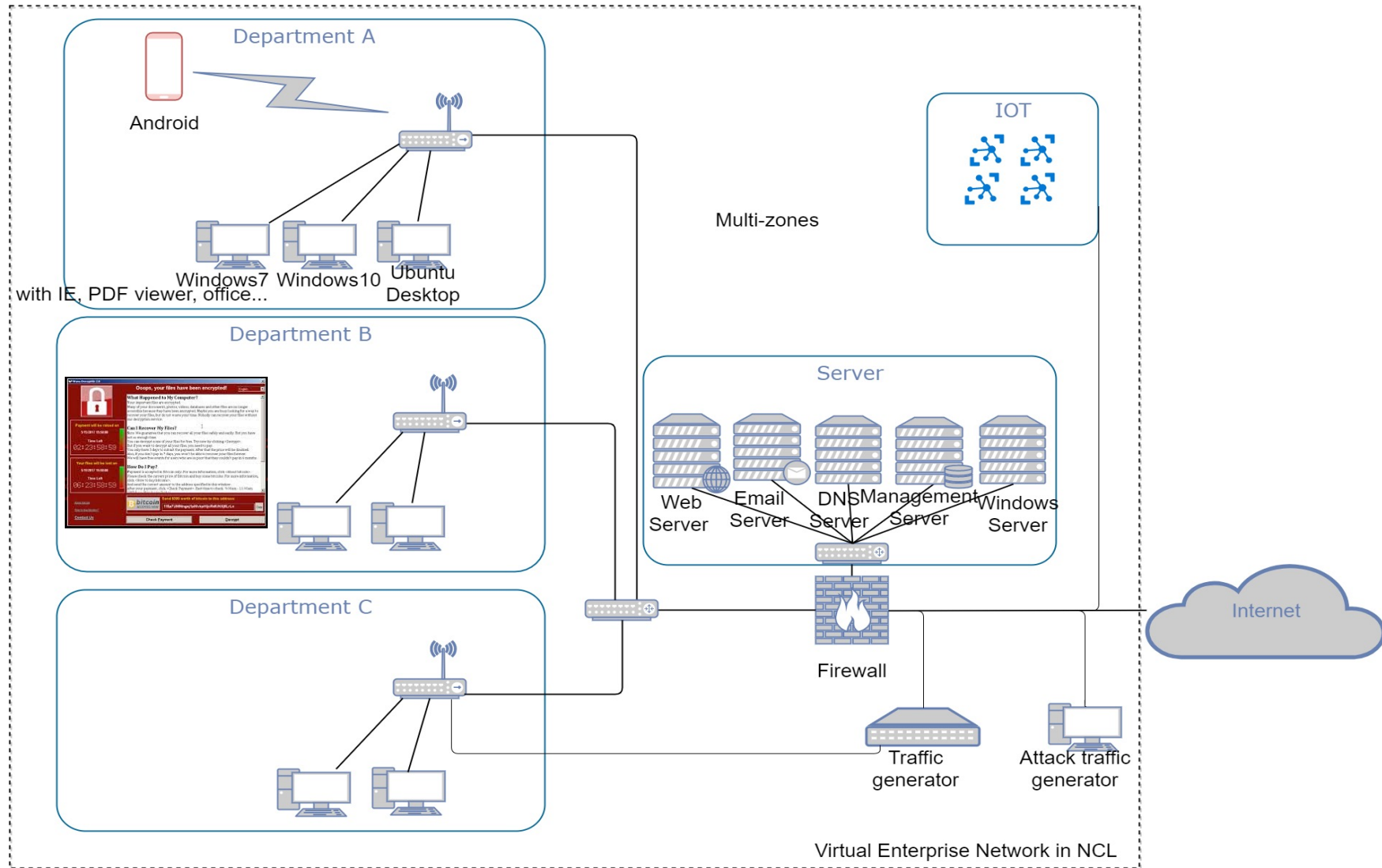
- **User-space** utilities (e.g., Auditd) collect system call records from kernel space through Netlink and write them to a log file under `/var/log/audit`
 - An Example of a read log entry in Auditd

```
-----  
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=0x7ffd69eecd0 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

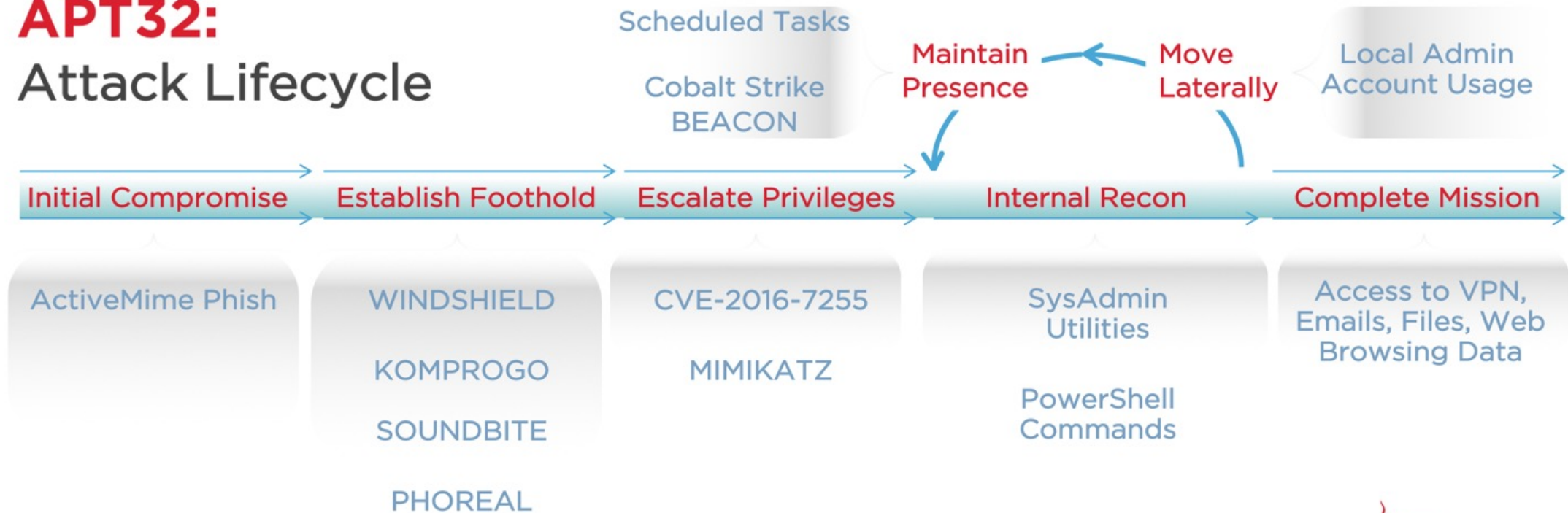
```
{"@timestamp":"2020-11-04T14:27:14.666Z","@metadata":{"beat":"auditbeat","type":"doc",  
"version":"6.8.12"},"auditd":{"sequence":989996,"result":"success","session":"1402","data":  
{"a3":"20656c706f657020","tty":"(none)","a2":"1000","arch":"x86_64","syscall":"read",  
"exit":"4096","a1":"5583baa77f70","a0":"5"}},"user":{"name_map":{"suid":"root",  
"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"}}
```

Enterprise Network View

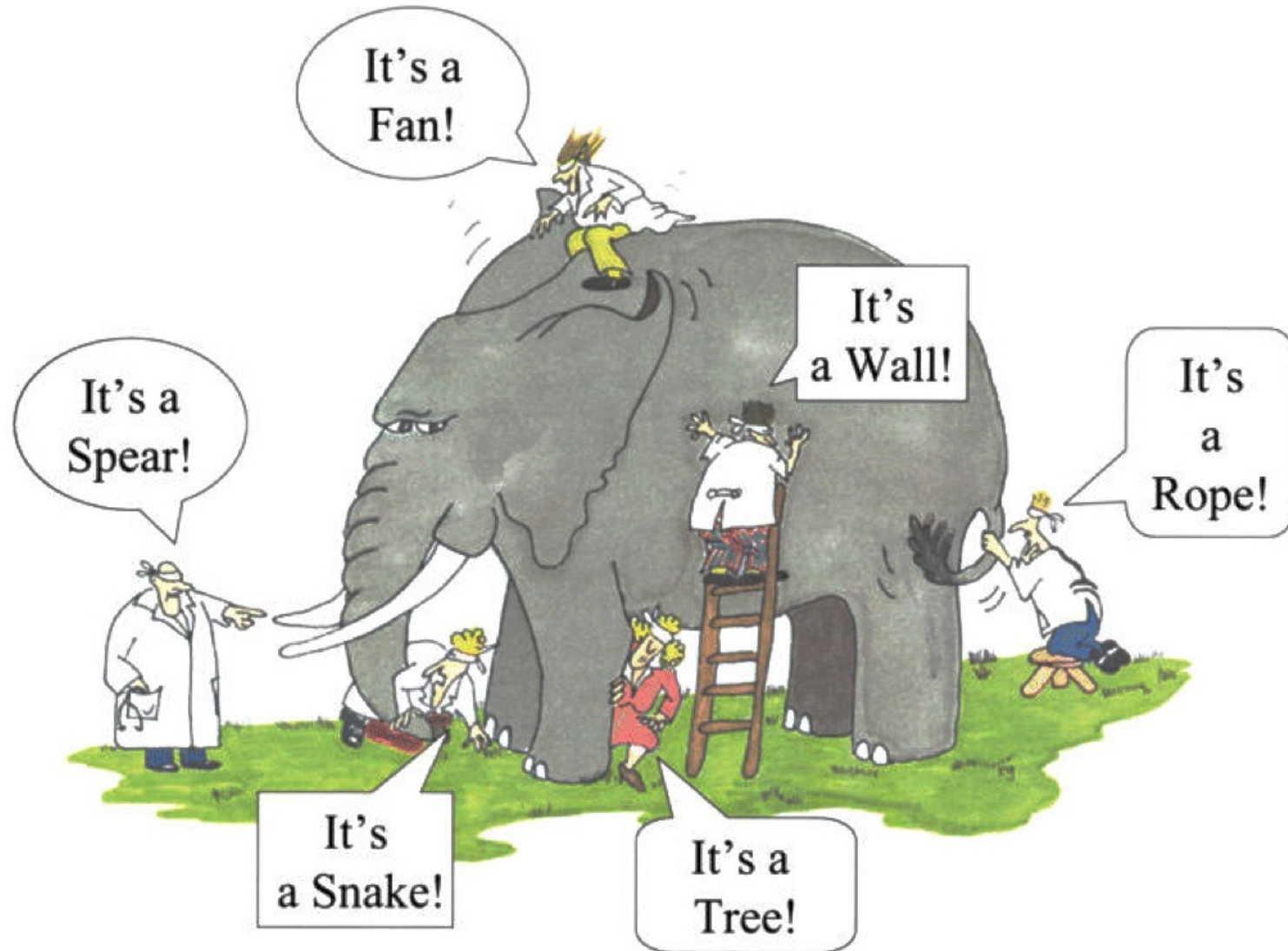


High-level Report

APT32: Attack Lifecycle



Understanding of Cyber Security Events



Endpoint Monitoring Solutions

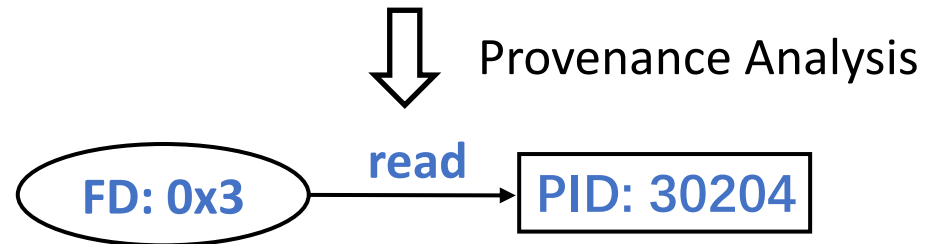
Endpoint monitoring solutions record **audit logs** for attack investigation



Audit logs:

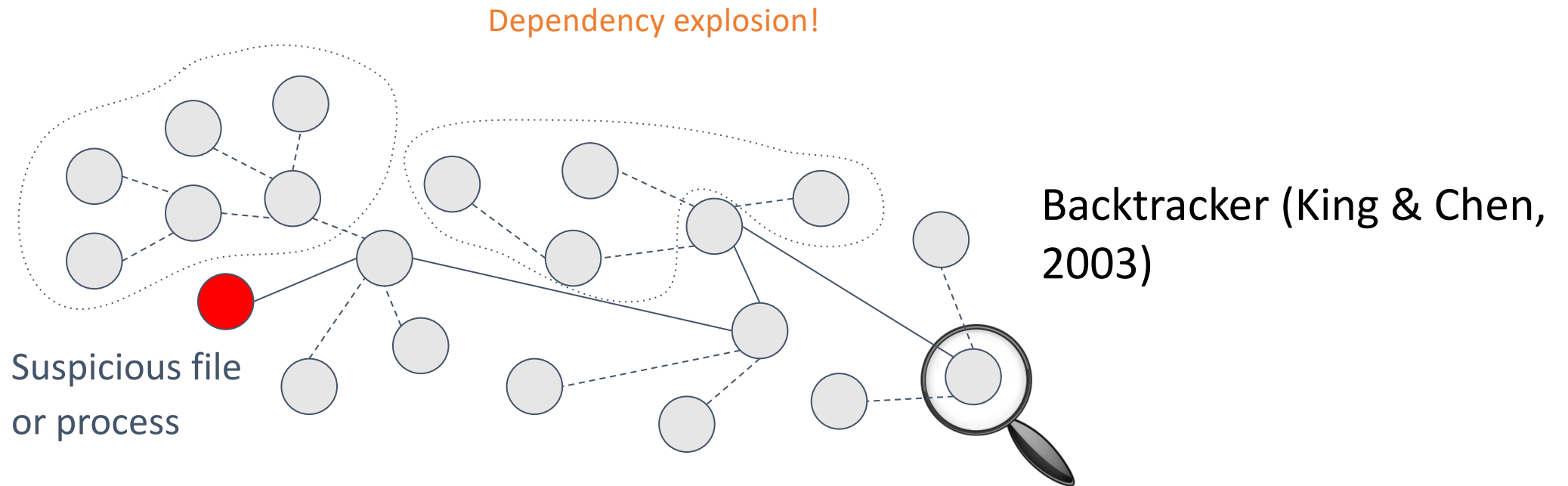
- A history of events representing OS-level activities
- Provide visibility into security incidents with data provenance

```
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
```



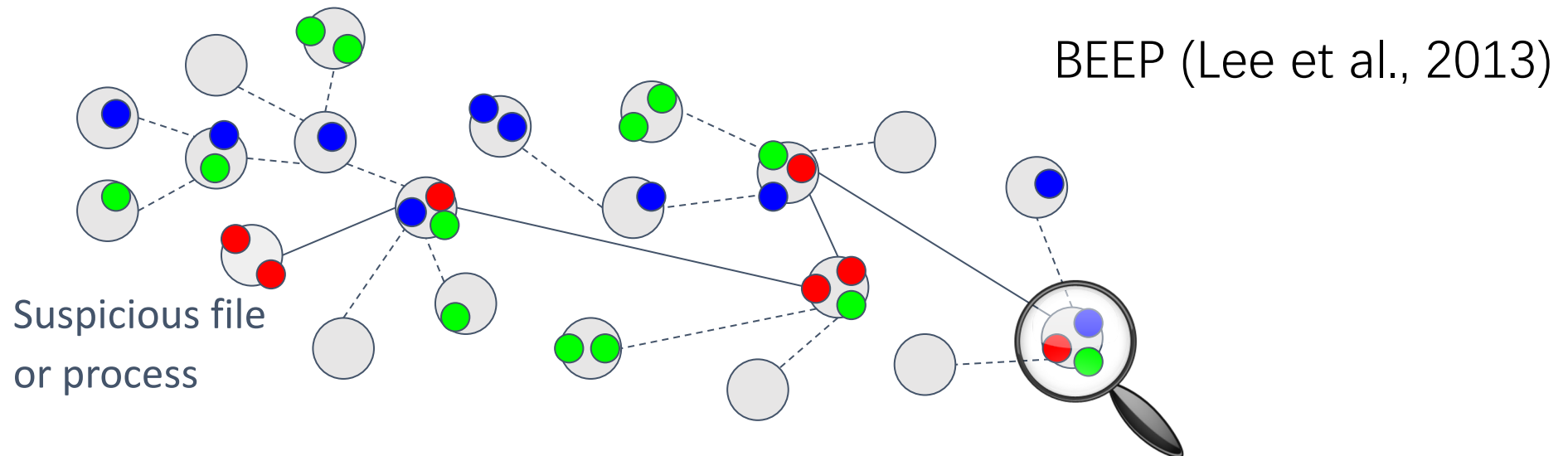
Audit Log Analysis

- 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



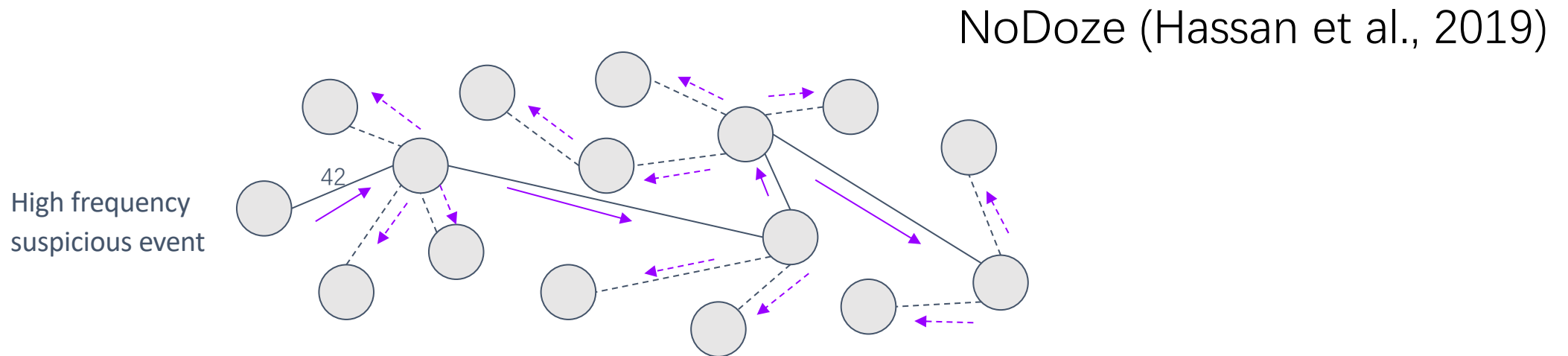
Audit Log Analysis

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



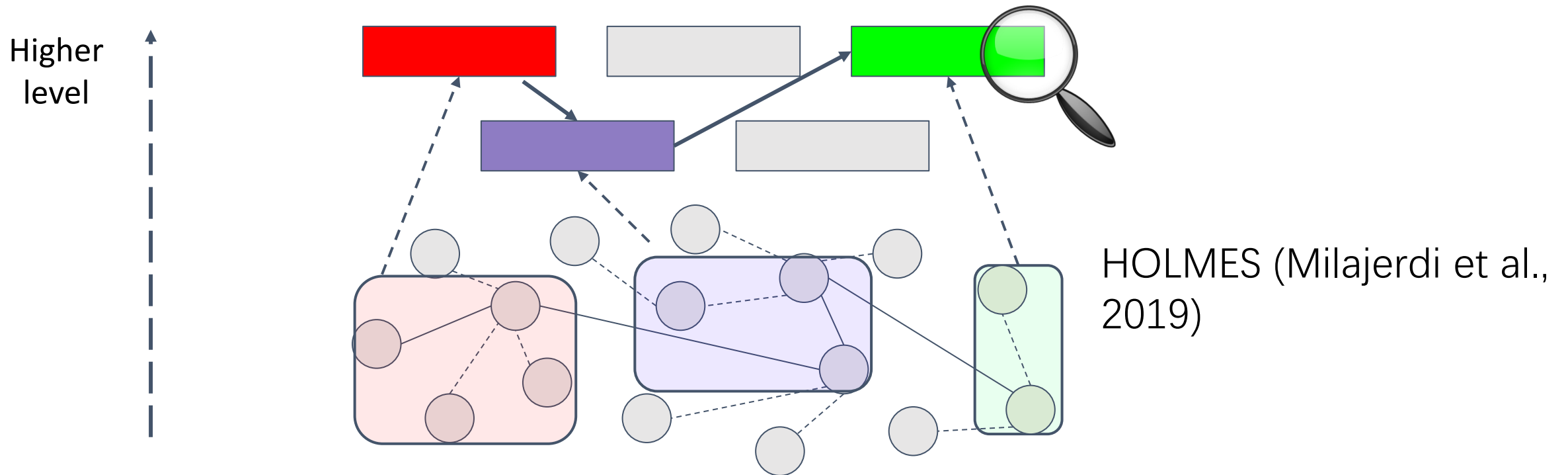
Audit Log Analysis

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



Audit Log Analysis

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



Related Work

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

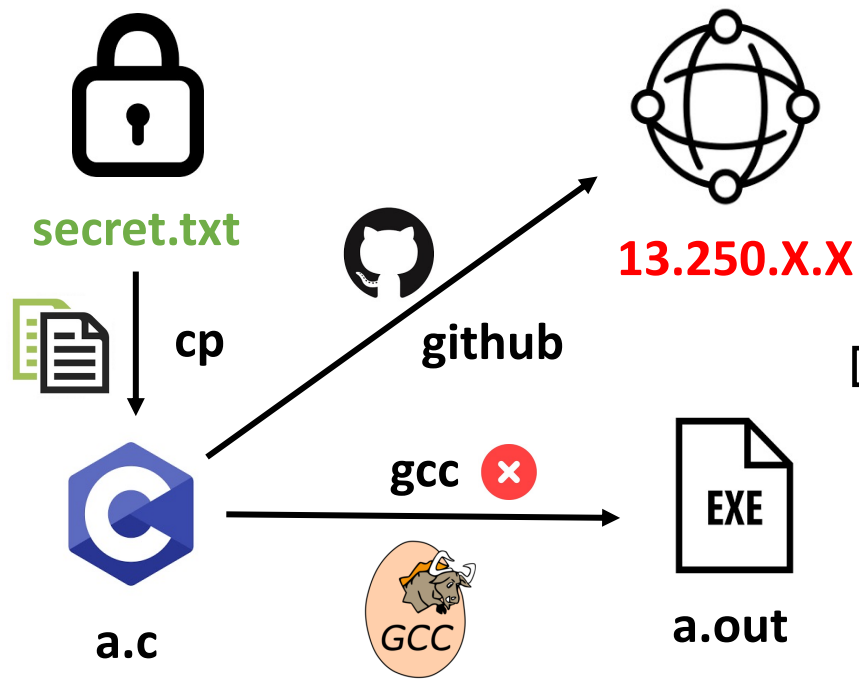
Can we automatically **abstract** high-level behaviors from low-level audit logs and **cluster** semantically similar behaviors before human inspection?

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

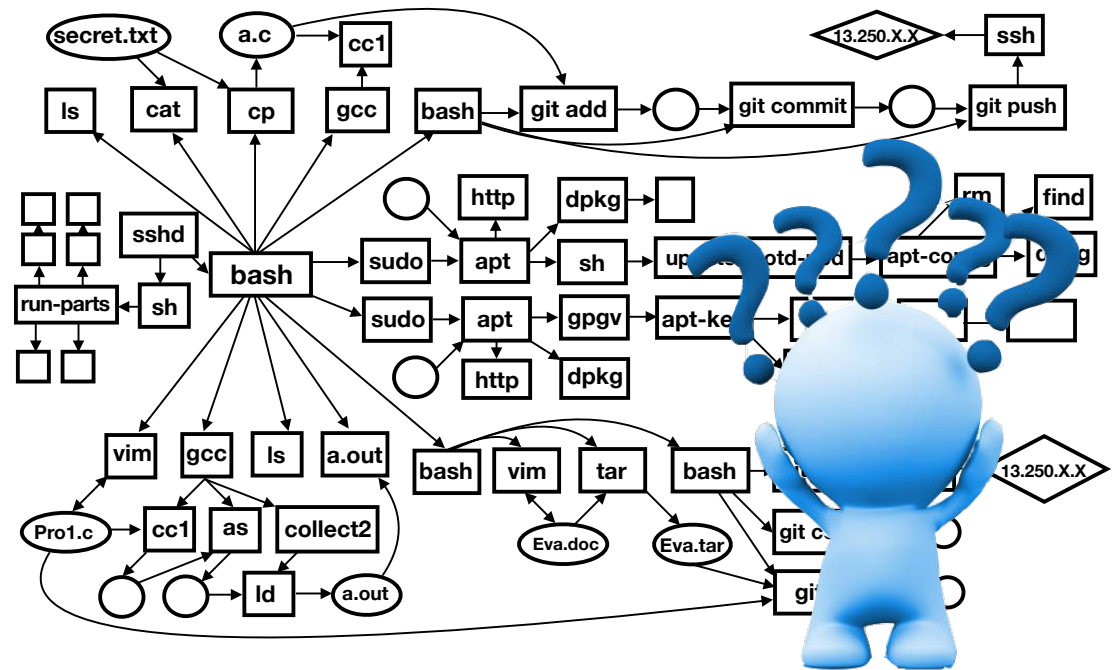
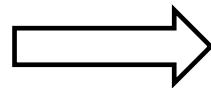
Behavior-specific rules heavily rely on domain knowledge (**time-consuming**)

Motivating Example

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



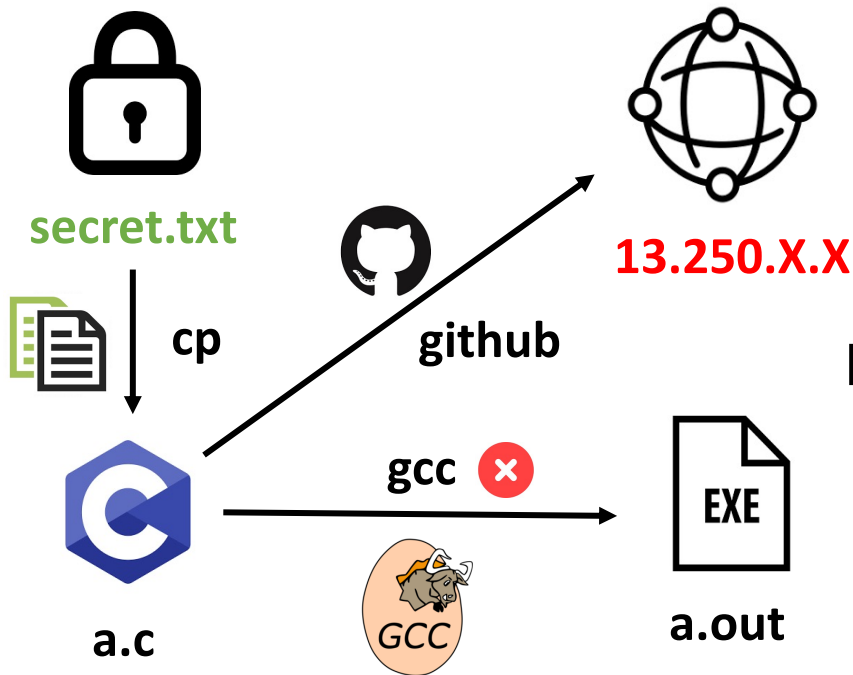
Data Exfiltration Steps



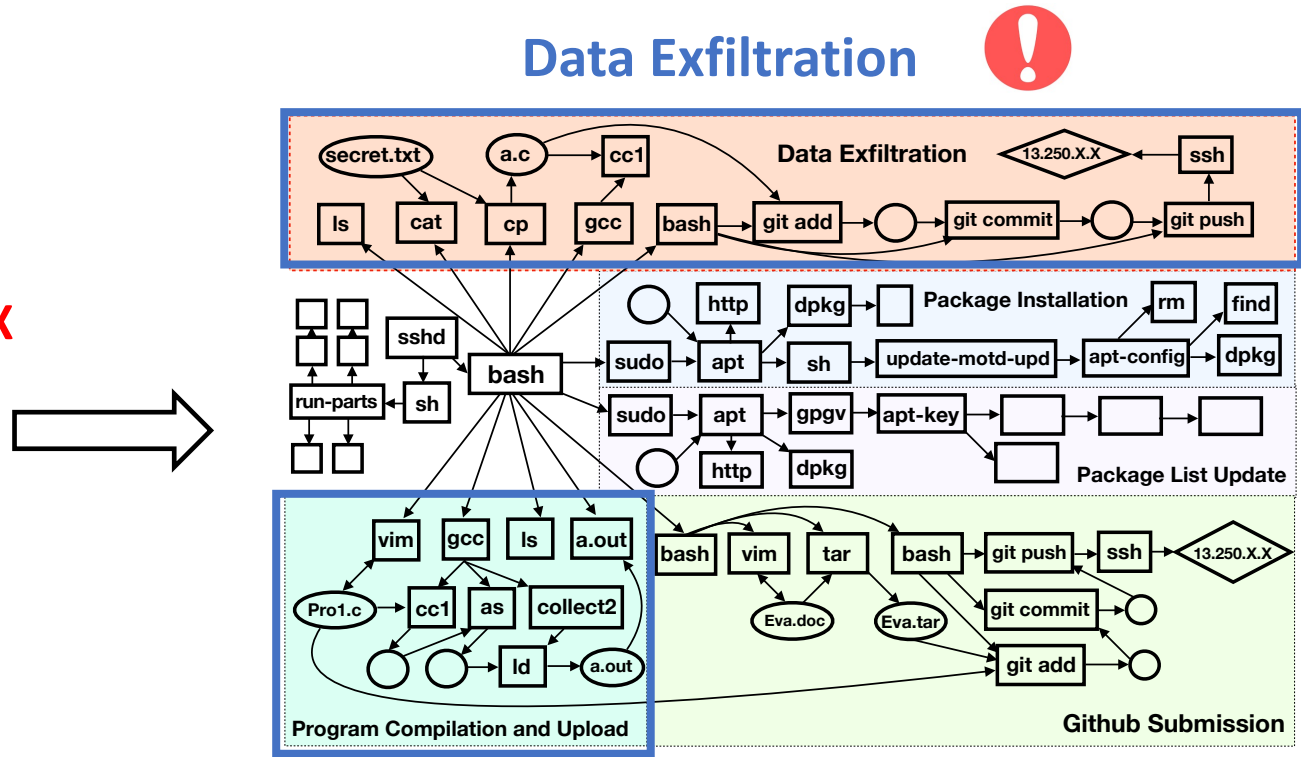
Motivating Example Logs

Motivating Example

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



Data Exfiltration Steps



Program Compiling and Upload (cluster)

Motivating Example Logs

Challenges for Behavior Abstraction

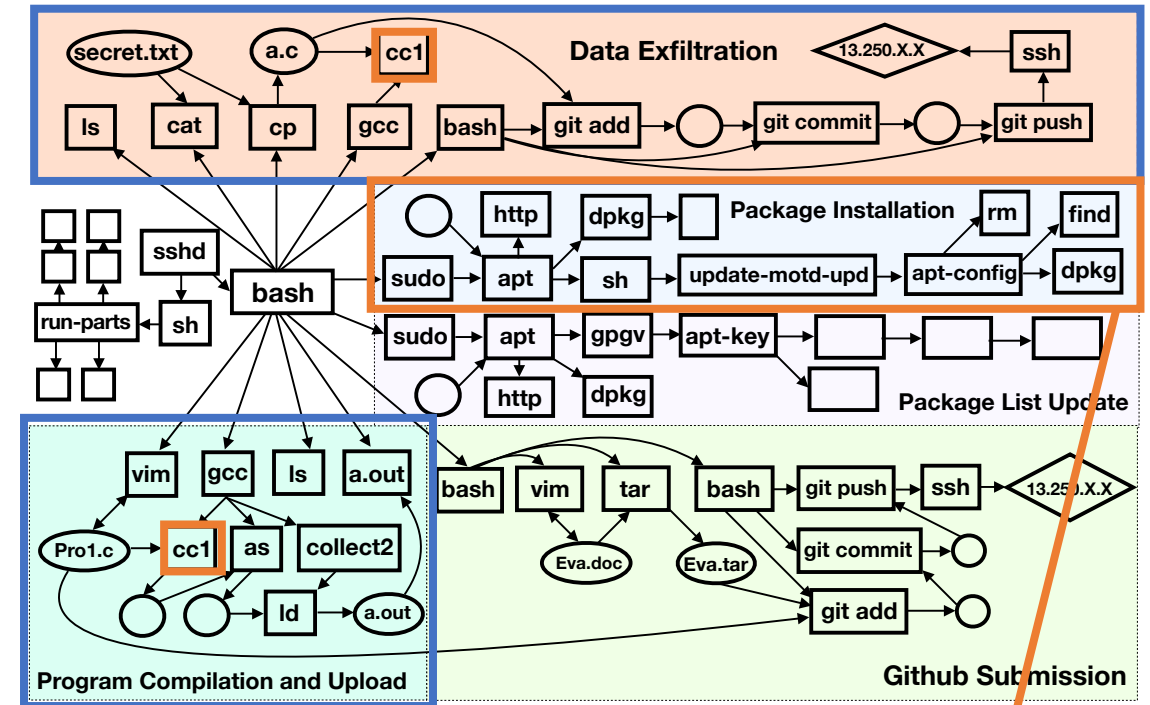
Data Exfiltration

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**

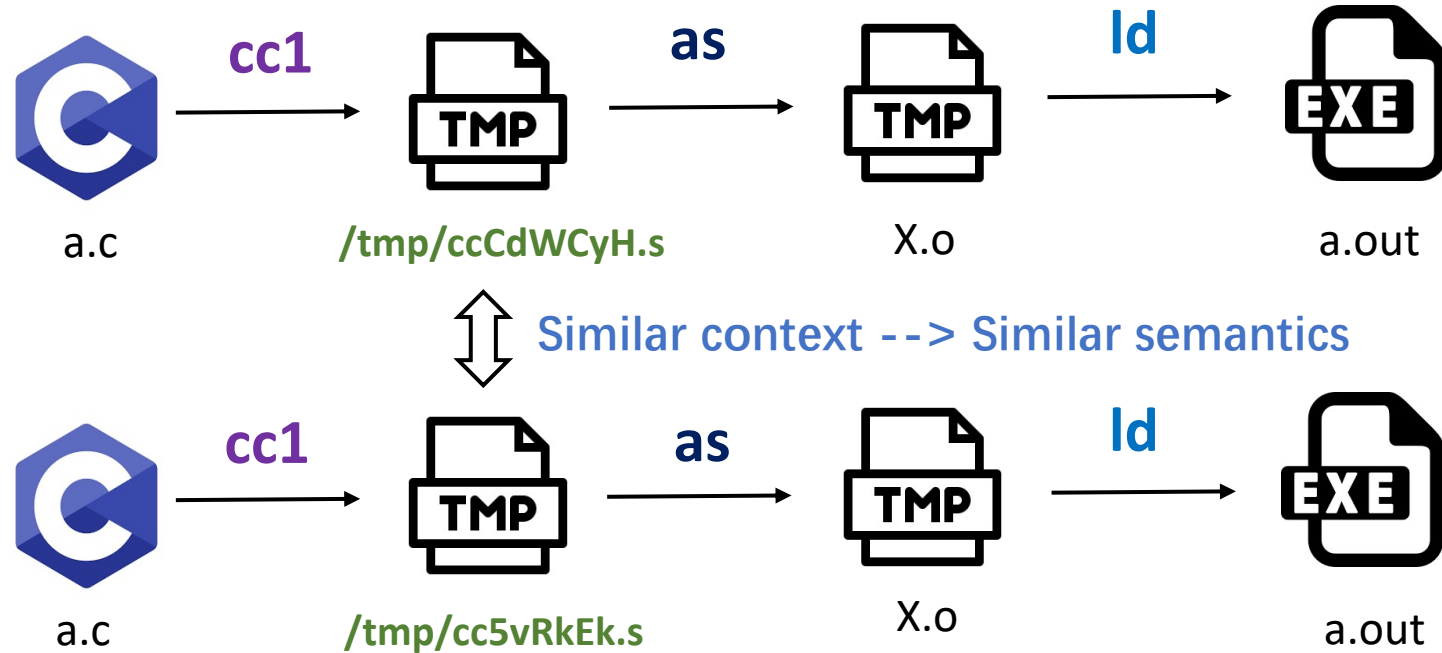


Program Compiling and Upload

Package Installation Events > 50,000

Our Insights

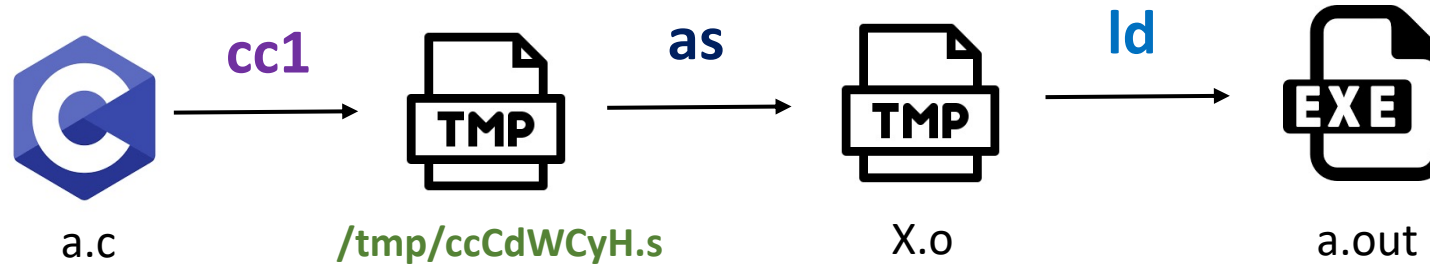
How do analysts manually interpret the semantics of audit events?



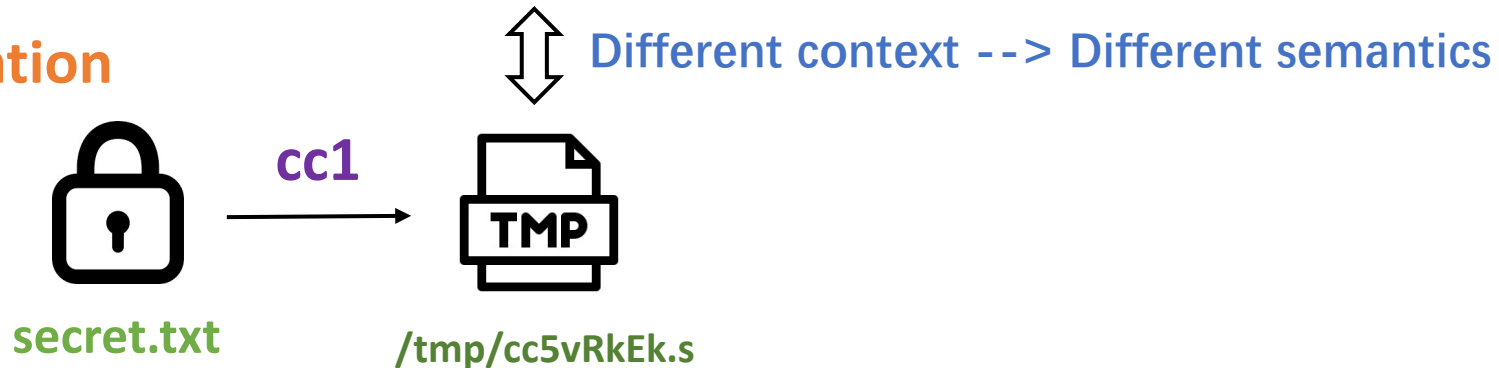
Compiling program using GCC

Our Insights

How do analysts manually interpret the semantics of audit events?



Data Exfiltration

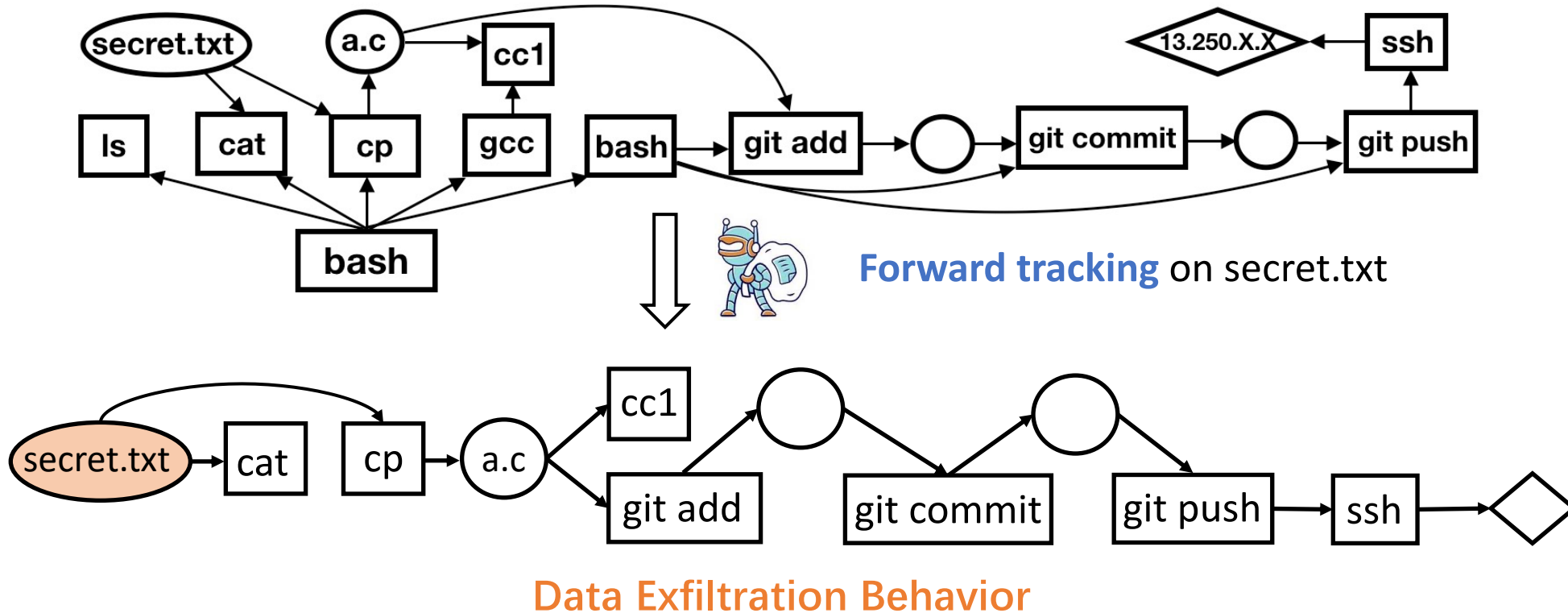


Compiling program using GCC

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

Our Insights

How do analysts manually identify behaviors from audit events?

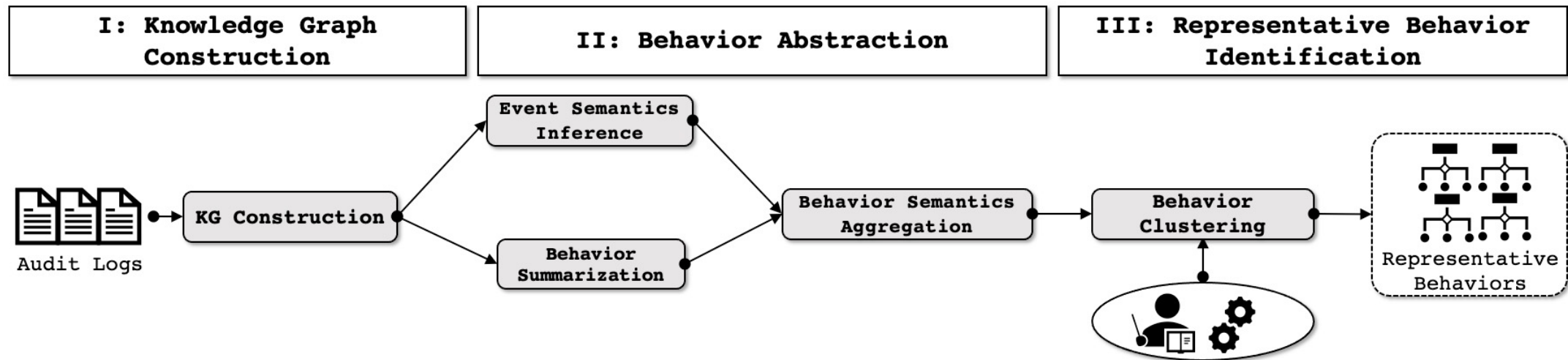


Summarize behaviors by tracking **information flows** rooted at **data objects**

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>.

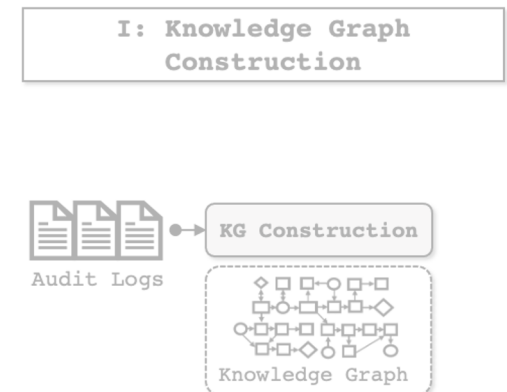
Knowledge Graph Construction

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):

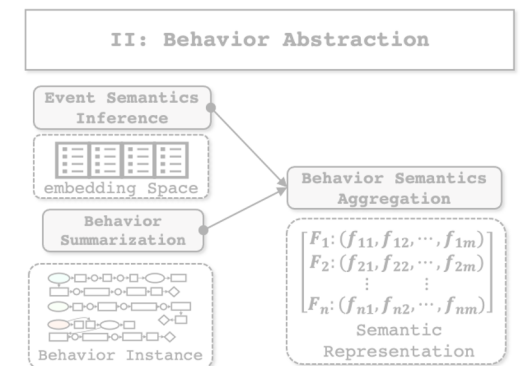
$$\mathcal{KG} = \{(h, r, t) | h, t \in \{Process, File, Socket\}, r \in \{Syscall\}\}$$

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



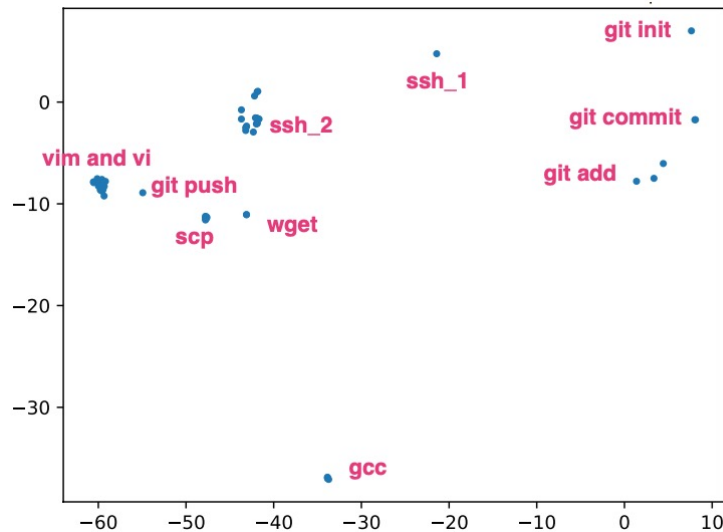
Event Semantics Inference

- 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 \approx Tail \rightarrow Context decides semantics**

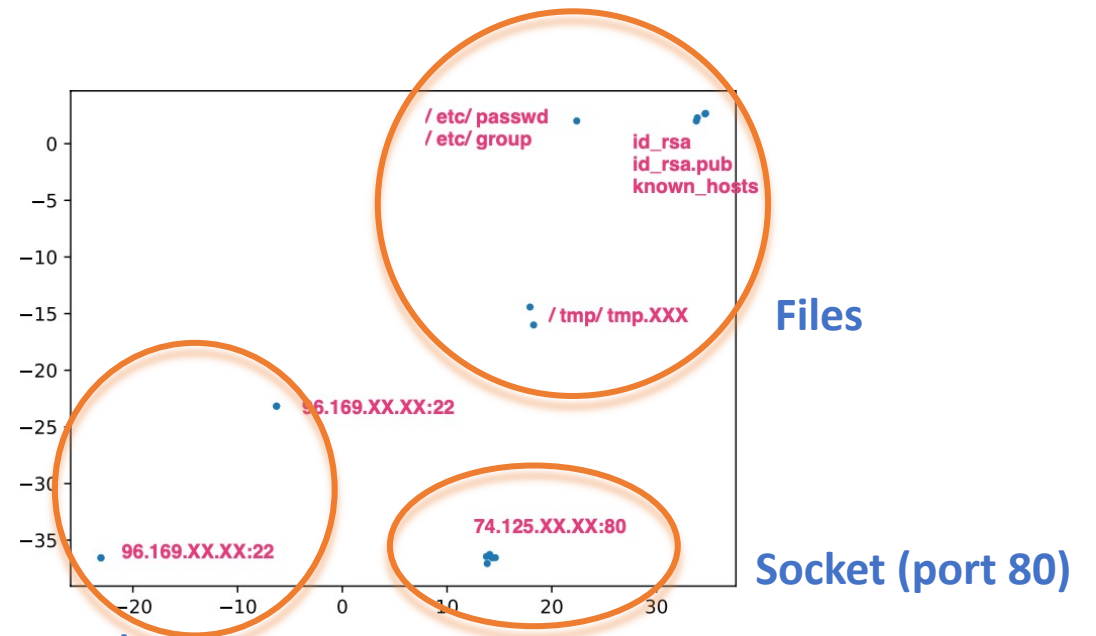


Event Semantics Explicability

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



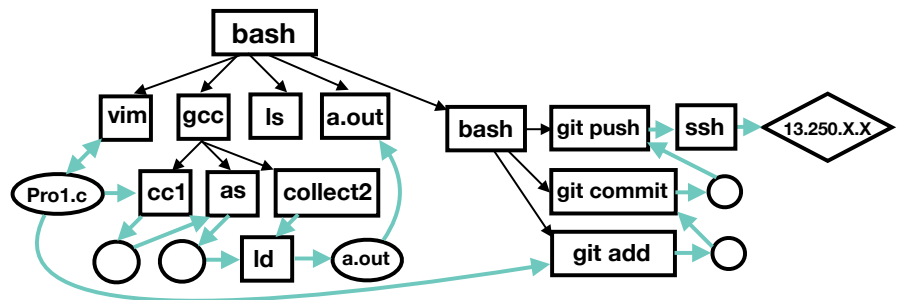
(b) 25 data object embeddings

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

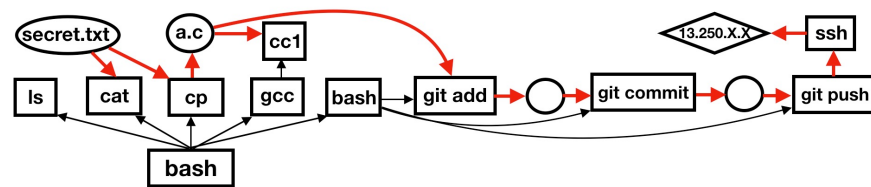
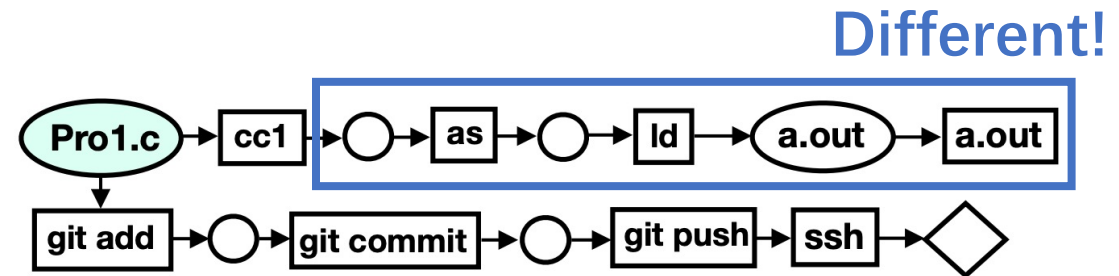
Behavior Summarization

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

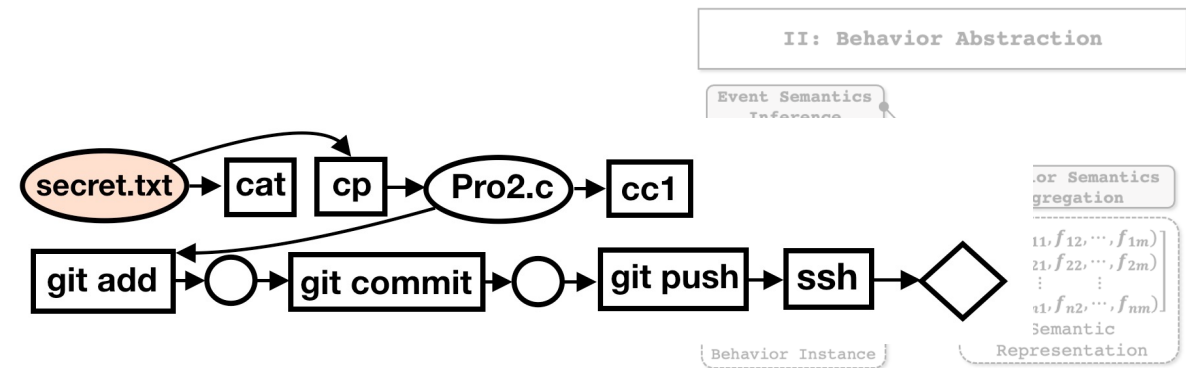
- Perform the DFS on every data object except libraries
- Two behaviors are merged if one is the subset of another




Program Compiling and Upload

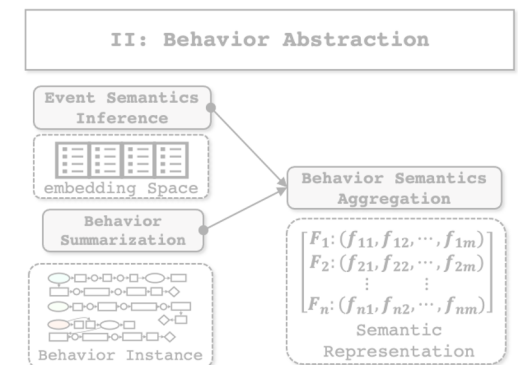


Data Exfiltration



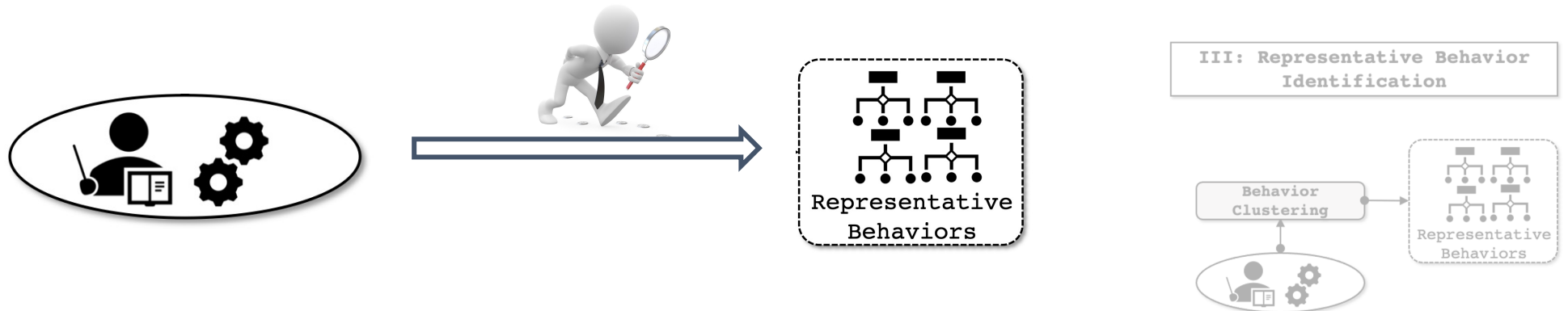
Behavior Semantics Aggregation

- 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

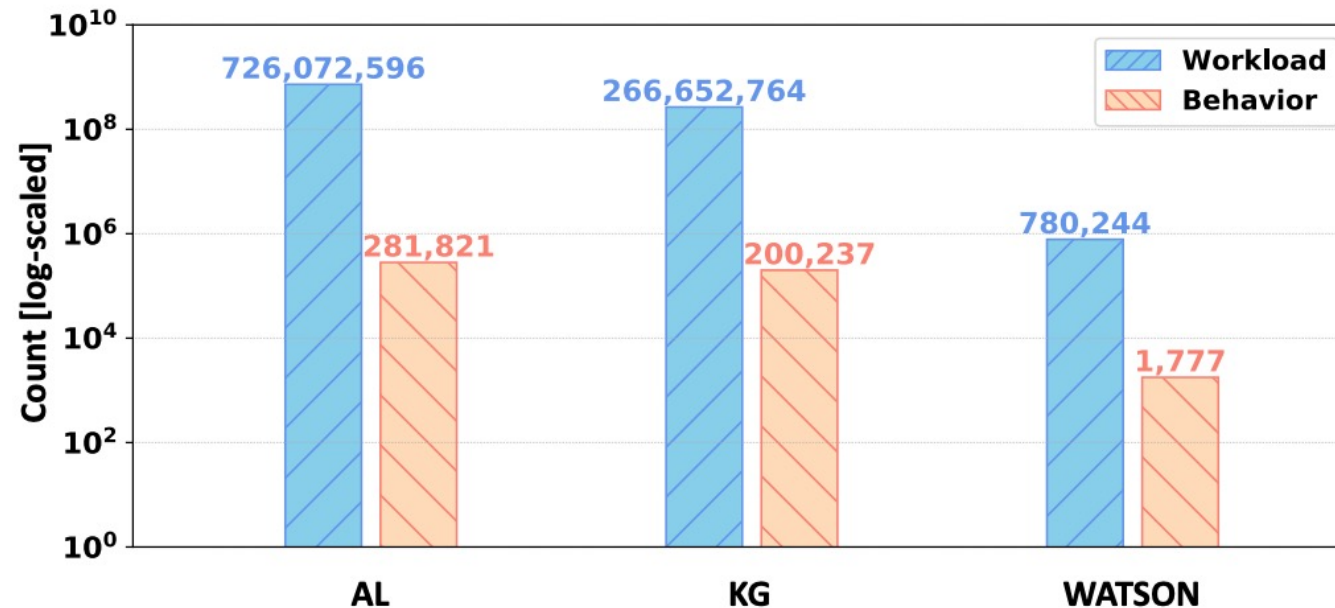
- Cluster semantically similar behaviors: **Agglomerative Hierarchical Clustering analysis (HCA)**
- Extract the most representative behaviors
 - Representativeness: Behavior's average similarity with other behaviors in a cluster
 - **Analysis workload reduction**: Do not go through the whole behavior space



Efficacy in Attack Investigation

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

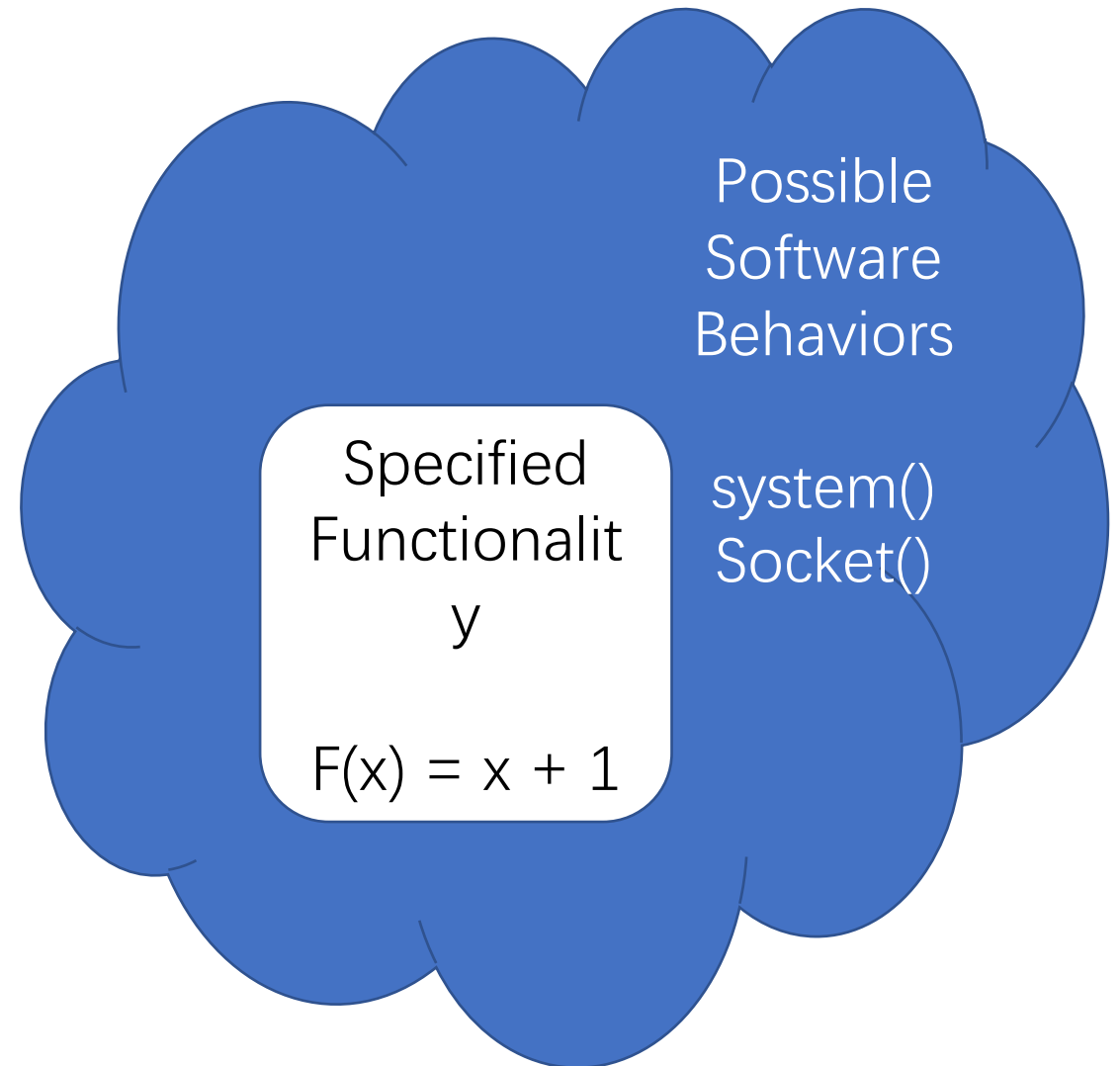
- Analysis workload: the number of events to recognize all behaviors



Two orders of magnitude reduction in analysis workload and behaviors

Functionality, Flexibility, and Security

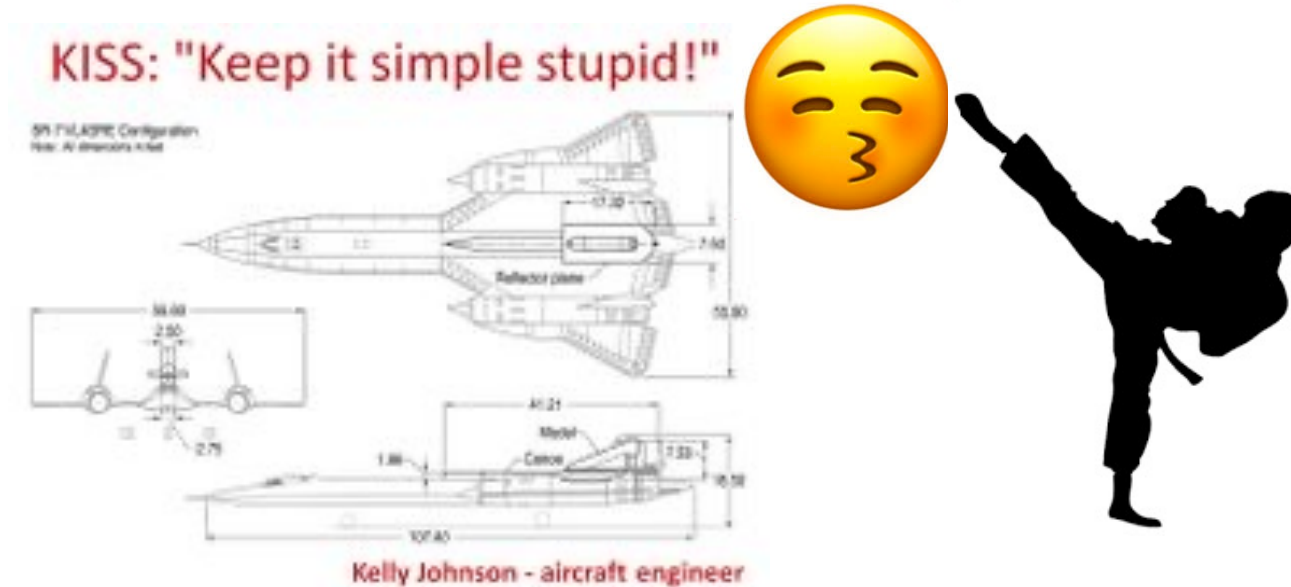
- Security is about “nothing else”
 - Specified functionality and **only** specified functionality
- Flexibility is the root of many security problems



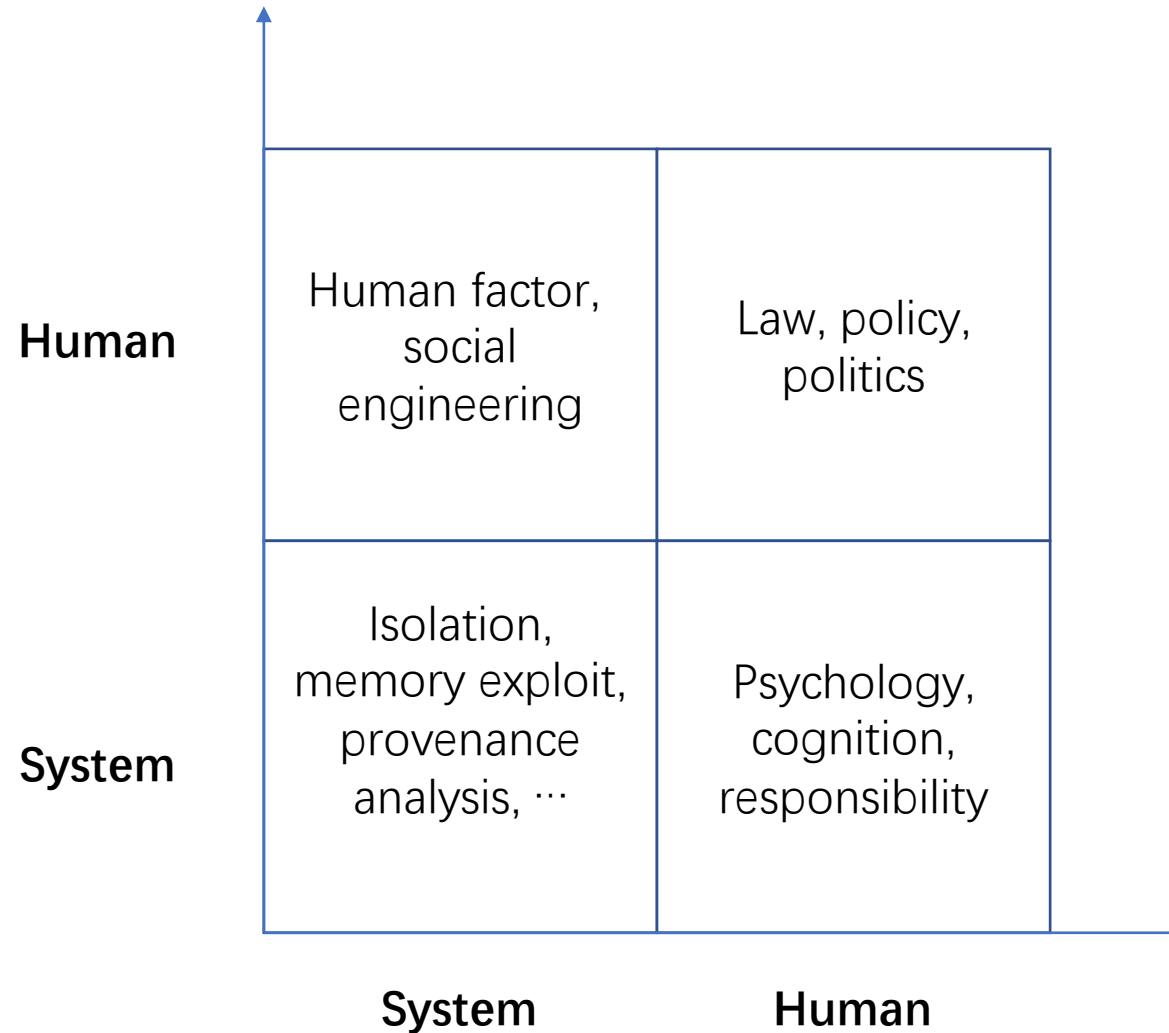
Simplicity in System Design

- KISS
(Keep It Simple, Stupid)

- KICS
(Keep It Complex & Smart)



Dimensions of System Research



Objective for Graduate Education

- Deep knowledge and skills in technology domain
 - Abstraction and presentation of thoughts
- Ability to think and analyze broadly
 - Especially in challenging times, calling for independent minds
- Understanding systems better
 - Order, flexibility, force, ...
- Form and live with a philosophy
 - Embrace trust
 - Life with minimal dependency

Research Areas and Range of Development



Views from different perspectives
Culture, social system, etc.



Top-ranked degree program and open culture
Top researchers in various fields



Economics, business, and technology
Fintech Institute

Trusted environment in Web/Mobile

System and security:
National Cybersecurity R&D Lab

Binary and System Analysis
Attack diagnosis and attribution

Behavior/Psychology and security:
CFPR in Arts and Social Science

Human behavior in cyber experimentation

- Multiple-level views of cyber incidents
- Our Insights in log analysis
 - Infer audit event semantics by usage contexts
 - Identify behaviors with information flows rooted at data objects
- On system research

Understand the movement of the sun and moon from traces of shadows under the roof.

审堂下之阴，而知日月之行，阴阳之变也



Understanding systems 理解系统
Abstracting knowledge 提炼知识
Connecting facts 参悟规律

Thank you!