Causality Analysis for Attack Investigation

杨帆

2018年6月28日

Background

- **A**dvanced **P**ersistent **T**hreat attacks are increasingly sophisticated
- APT attacks are conducted in multiple stages
  - Initial compromise/Establish foothold/Escalate privileges/Internal reconnaissance/Lateral movement/Maintain presence/Mission completion
- Investigating APT attacks is challenging, because:
  - APT attacks often leverage benign built-in software to avoid detection
  - *Low and slow*: attacks often span a long duration of time with a low profile
- An attack symptom may be detected at some of the stages
  - If you are fortunate enough ☺
- After that, the administrator can perform ***attack causality analysis***
  - To understand the attack, including its root cause and ramifications

Outline

- Introduction to Attack Causality Analysis
- Dependency Explosion and How To Avoid It
- Selected New Ideas/Topics
- Conclusions

Outline

- **Introduction to Attack Causality Analysis**
- Dependency Explosion and How To Avoid It
- Selected New Ideas/Topics
- Conclusions

Attack Causality Analysis: Intuitions

- Causality analysis is performed after the attack has happened
- The goal is to achieve a detailed understanding of a (detected) attack

- So the basic ideas are:

(1) Record what the attackers have done in a sequential manner

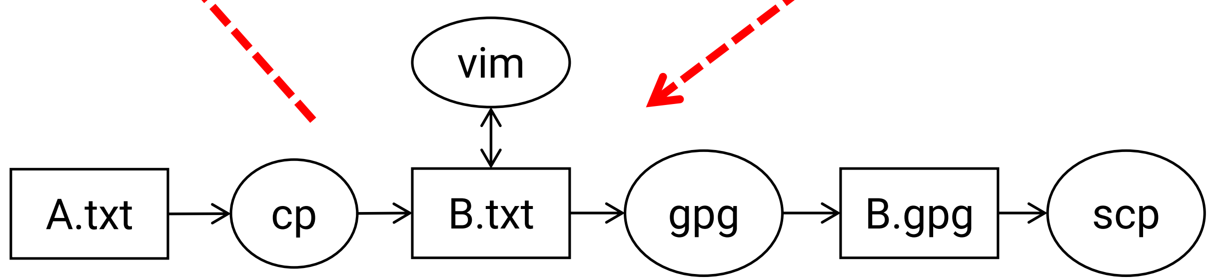
(2) Reconstruct the relationships between individual attack footprints

Attack Causality Analysis: Basic Ideas

- The attack behavior can be decomposed into primitive operations
  - e.g., file-related behavior can be decomposed into a small set of system calls
- We try to reconstruct the attack behavior from these recorded primitive operations later

An attacker opens a secret file, copies and pastes the file contents to a new file. Then, he edits the new file containing the secret. Later, he encrypts the new file, and sends it to outside.

```
...
[cp] a = open(“A.txt”, O_RDONLY);
[cp] b = creat(“B.txt”, 0644);
[cp] read(a, &buf, sizeof(buf));
[cp] write(b, &buf, sizeof(buf));
...
[scp] send(sockfd, &buf, sizeof(buf), 0);
```



Attack Causality Analysis: The Big Picture

- Collecting event traces via system interfaces or instrumentation
  - system interfaces: Linux Kernel Audit, Windows ETW
  - instrumentation: source code annotation

- Identifying the dependencies among subjects and objects
  - subjects: processes/sessions, objects: files/sockets

- Building and preprocessing the provenance graph for further analysis
  - pruning, refining, and prioritizing

- Extract useful attack information from the resulting graph
  - e.g., backtracking, prioritized searching, matching

Dependency Identification

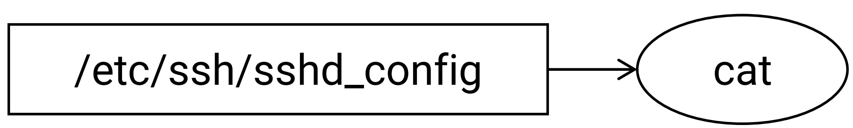
- Process/Process dependencies
  - Creating a new process: `clone/fork/execve`
  - Sharing memory with another process: `mmap`
  - Sending signal to another process: `signal`
- Process/File dependencies (file content and attribute)
  - file→process: `read/readv/execve/recv/open/unlink/fstat/…`
  - process→file: `write/writev/send/chown/chmod/utime/…`
- Process/Filename dependencies
  - `open/create/link/unlink/mkdir/rename/rmdir/stat/chmod/mount/…`

Backtracking Intrusions. ST King, PM Chen. TOCS 2005

Example: Linux Kernel Audit System

- Some log entries for the event `'cat /etc/ssh/sshd_config'`:

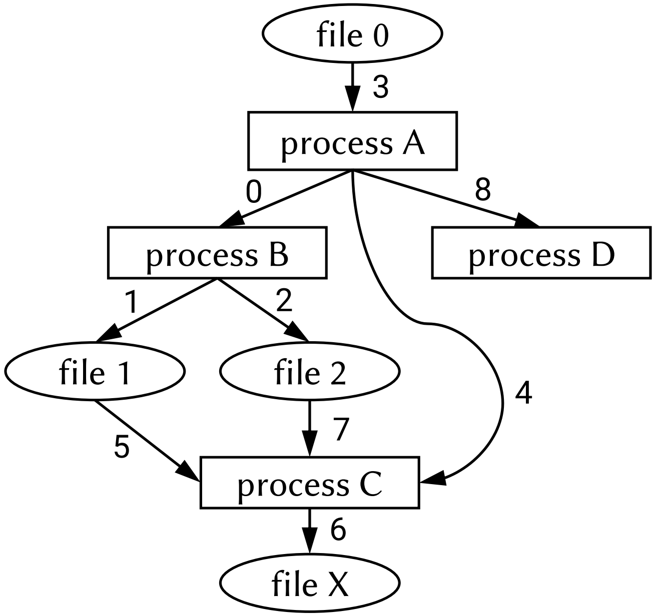
```
(1)type=SYSCALL msg=audit(1364481363.243:24287): arch=c000003e
   syscall=2 success=no exit=-13 a0=7fffd19c5592 a1=0
   a2=7fffd19c4b50 a3=a items=1 ppid=2686 pid=3538 auid=500 uid=500
   gid=500 euid=500 suid=500 fsuid=500 egid=500 sgid=500 fsgid=500
   tty=pts0 ses=1 comm="cat" exe="/bin/cat"
   subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
   key="sshd_config"

(2)type=CWD msg=audit(1364481363.243:24287): cwd="/home/shadowman"

(3)type=PATH msg=audit(1364481363.243:24287): item=0
   name="/etc/ssh/sshd_config" inode=409248 dev=fd:00 mode=0100600
   ouid=0 ogid=0 rdev=00:00 obj=system_u:object_r:etc_t:s0
```



Building the Provenance/Causal Graph

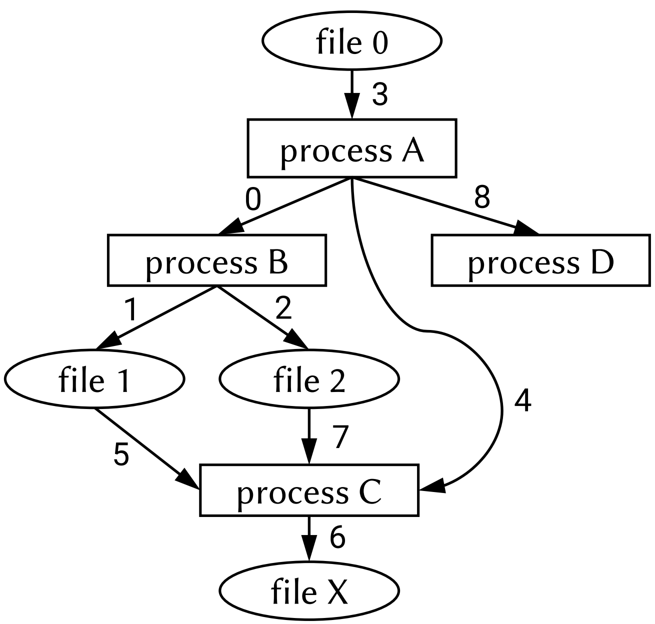
- After identifying all dependencies, building the graph becomes trivial
- We build a graph in which:
  - Nodes represent objects (files) and subjects (processes)
  - Directed edges represent dependencies
- Each edge is associated with the timestamp of the log entry

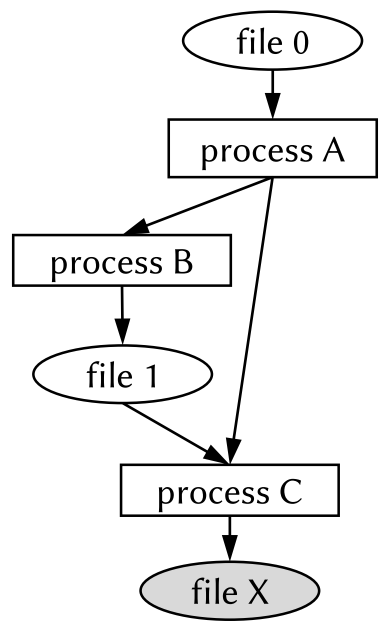
time 0: process A creates process B
time 1: process B writes file 1
time 2: process B writes file 2
time 3: process A reads file 0
time 4: process A creates process C
time 5: process C reads file 1
time 6: process C writes file X
time 7: process C reads file 2
time 8: process A creates process D



Basic Analysis: Backtracking and Forward Tracking

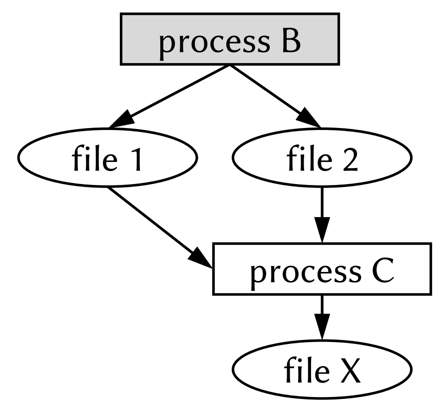
- Given a detection point (i.e., a node), we want to:
  - Retrieve the subgraph that causally affect the state of the detection point
    - find all **reverse-reachable** nodes from which there is path to the detection point
  - Retrieve the subgraph that are causally affected by the detection point
    - find all **reachable** nodes starting from the detection point





backtracking

forward tracking

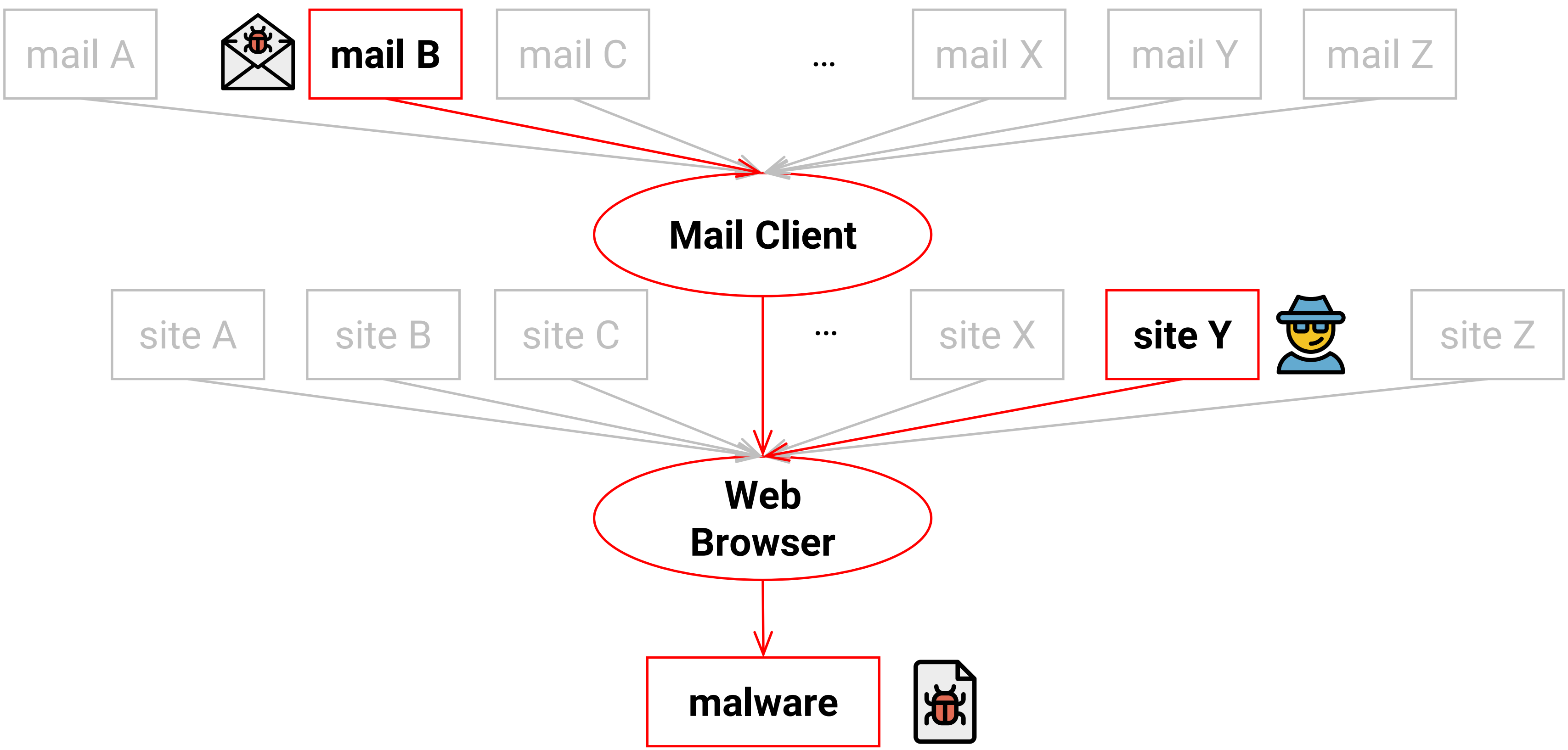


Outline

- Introduction to Attack Causality Analysis
- **Dependency Explosion and How To Avoid It**
- Selected New Ideas/Topics
- Conclusions

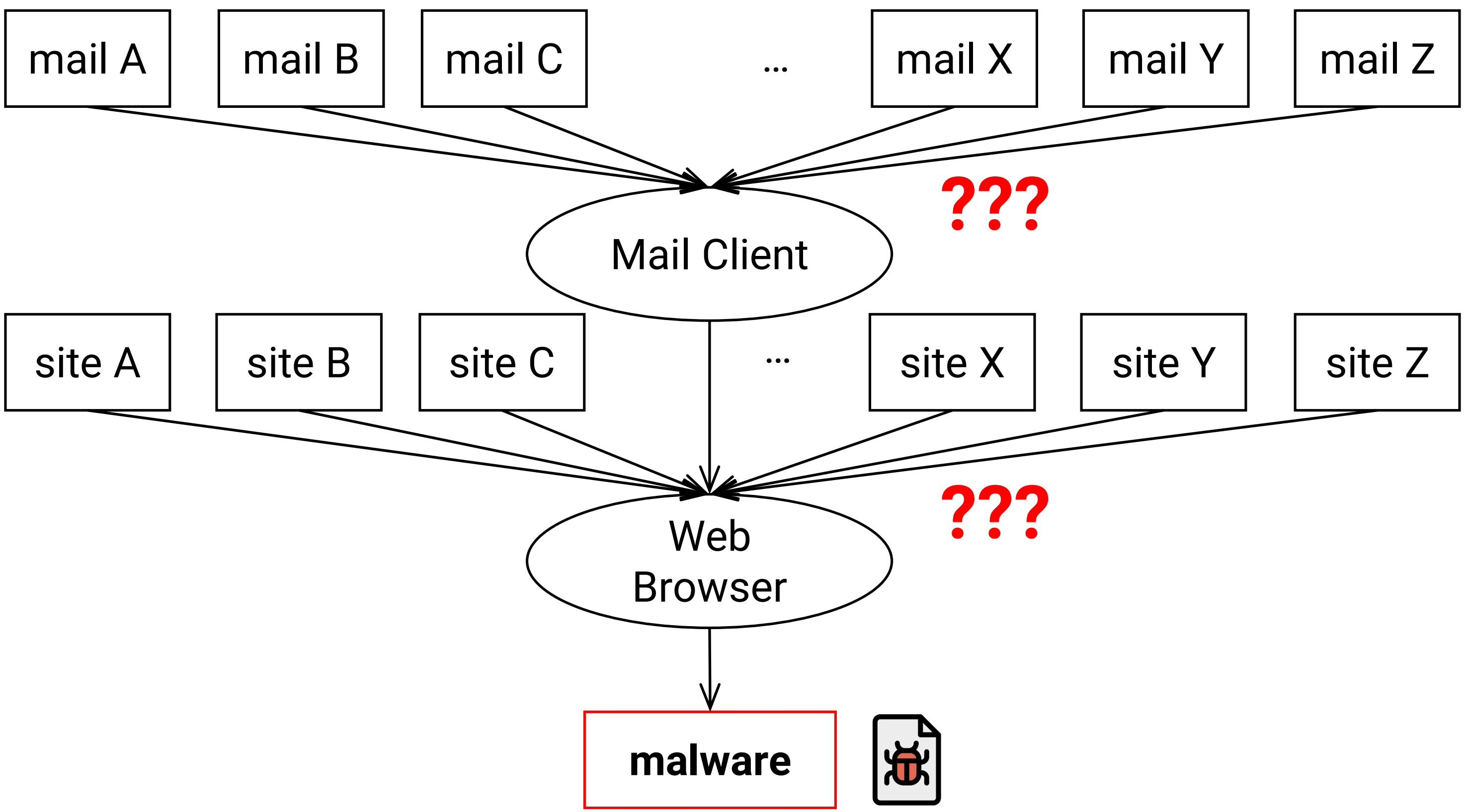
The Dependency Explosion Problem

- **Long running processes** interact with many other subjects and objects during their lifetime and generate many dependencies
  - Although only a small subset is attack related



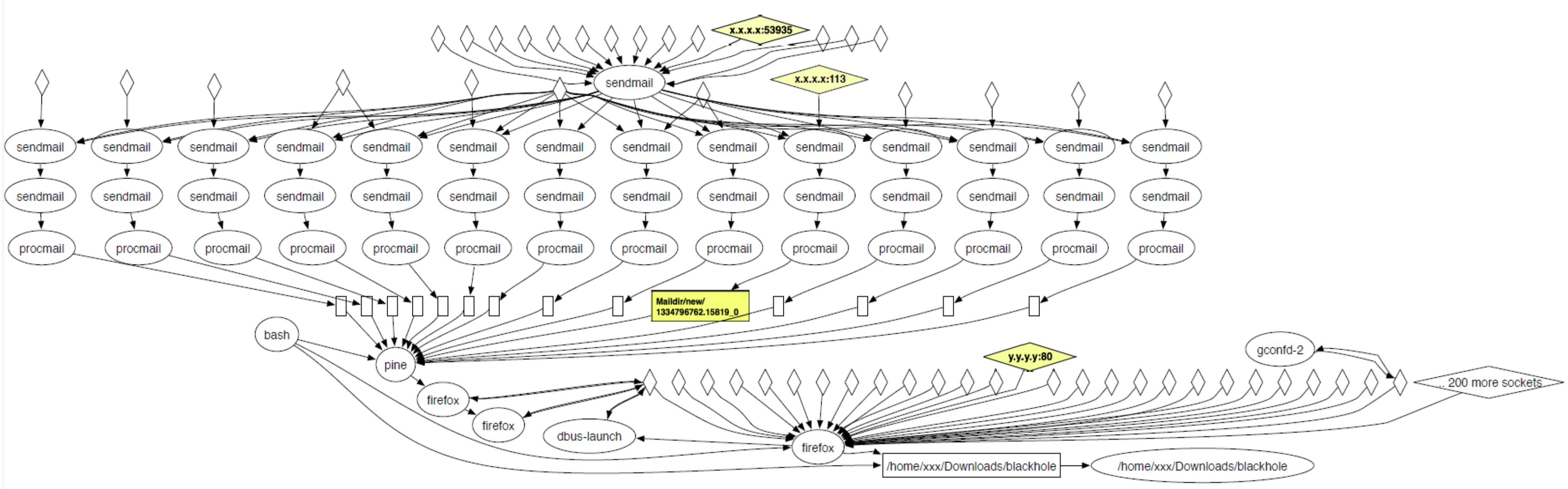
The Dependency Explosion Problem

- **Long running processes** interact with many other subjects and objects during their lifetime and generate many dependencies
  - Although only a small subset is attack related



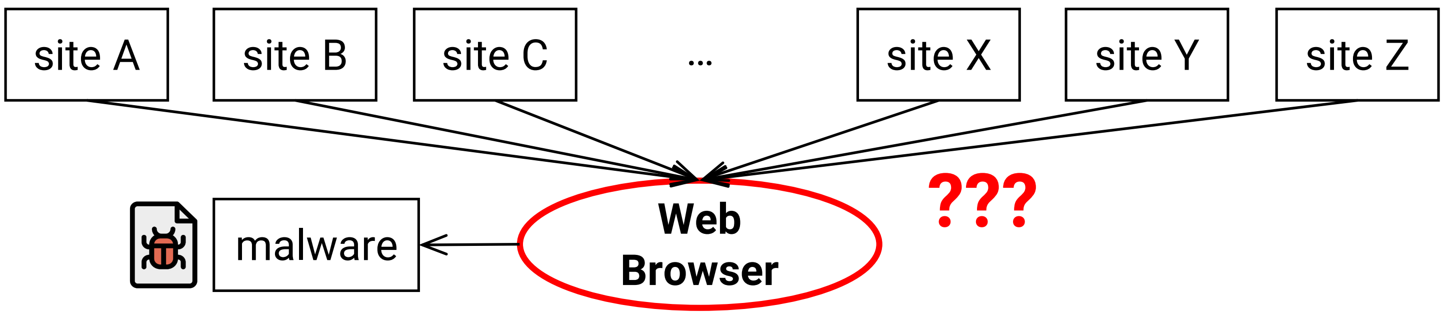
The Need for Eliminating False Dependencies

- Dependency explosion results in large and inaccurate causal graphs
- i.e., there are many **false dependencies**
  - Which make manual inspection extremely difficult
  - And prevent us from tracking the true attack path efficiently
- To generate cleaner graph, we have to eliminate these false deps

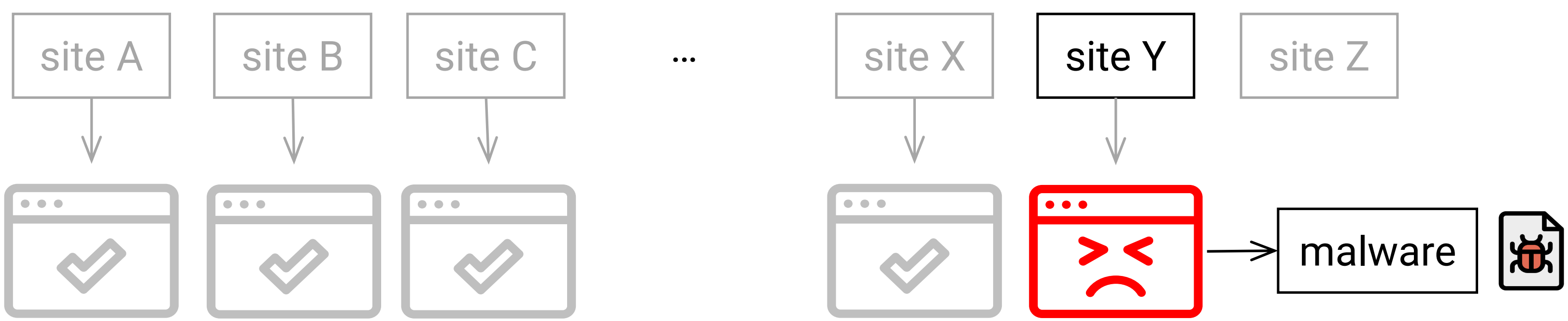


Eliminating False Deps: Basic Idea

- Key observation: false deps are mainly caused by the **coarse-grained** nature of long running processes

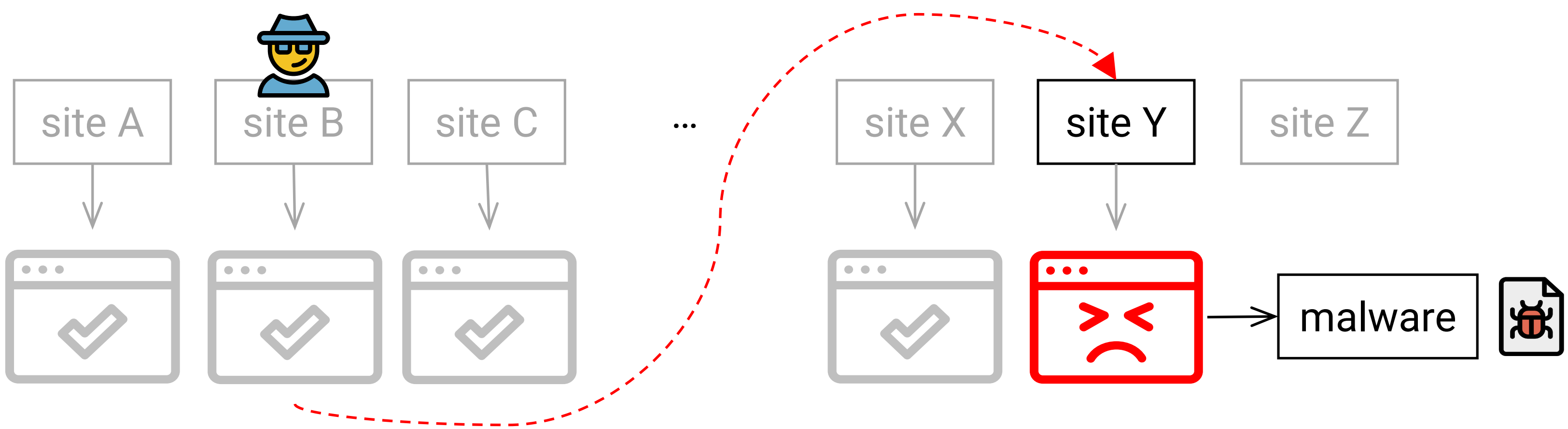


- Idea: break a long running process into **fine-grained** execution units
  - e.g., browser → browser tabs



Execution Partitioning

- In general, we can partition the execution into a chosen fineness level:
  - instruction < syscall < code block < function < thread/event loop < session < process
- It is crucial to capture the **inter-dependencies** across execution units
  - Simply partitioning is not enough, because it can introduce **false negatives**

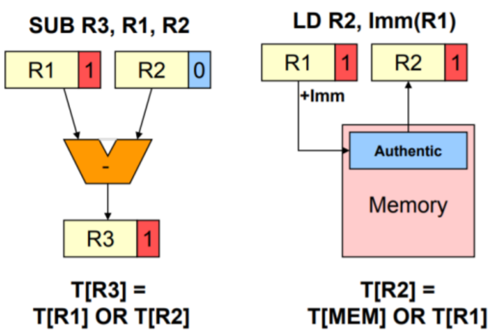


- There are trade-offs between partitioning and dependency detection
  - At finer level, dependency detection is generally harder and more expensive

Example #1: Instruction-Level Data Flow Tracking

- **Data/Information Flow Tracking**, a.k.a. ***taint analysis***
  - The process of *accurately* tracking the flow of *selected* data throughout the execution of a program
- Concepts: data sources, data tracking, data sinks
  - Sources: program or memory locations where data of interest enter
    - Data coming from these sources are **tagged** and tracked
  - Tracking: tagged data are tracked as they are copied/altered by instructions
    - Tags are **propagated** across data/control flow dependencies
  - Sinks: locations where one can check for the presence of tagged data

```
int authorized = 0;
...
n = read(fd, pwd, 32);
SHA1(pwd, n, hash);
if(0 == memcmp(hash, stored, 20))
    authorized = 1;
return authorized;
```



Secure Program Execution via Dynamic Information Flow Tracking. GE Suh, et al. SIGPLAN 2004
libdft: Practical Dynamic Data Flow Tracking for Commodity Systems. VP Kemerlis, et al. VEE 2012

Example #2: System Call Dependency Testing

- Instruction-level DFT usually introduces heavy runtime overhead
  - one instruction as an execution unit is too fine-grained
- LDX uses system calls as boundaries to partition the execution
- And tests the dependency of two system calls by ***dual execution***
  - Run two instances of a program
  - Give different inputs to the **source system call**
  - Check whether the **sink system calls** are different

```
"1234" = read(secret);        |   "abcd" = read(secret);
          ↓                    |             ↓
         ...                   |            ...
          ↓                    |             ↓
   send("1234");               |      send("abcd");
```

LDX: Causality Inference by Lightweight Dual Execution. Y Kwon, et al. ASPLOS 2016

Example #3: Event Loop Based Partitioning (1)

- BEEP observed that most long running programs are implemented in **event-handling loops**
  - Driven by external requests & dominated by event-processing handlers

	Category	Total Applications	Loop structured Applications
Servers	Web server	13	13
	Mail server	8	8
	FTP server	6	6
	SSHD server	2	2
	DNS server	9	9
	Database server	4	4
	Proxy server	2	2
	Media server	5	5
	Directory server	3	3
	Version control server	2	2
	Remote desktop server	2	2
UI Programs	Web browser	5	5
	E-mail client	5	5
	FTP client	5	5
	Office	2	2
	Text Editor	3	3
	Image tool	4	4
	Audio player	2	2
	Video player	4	4
	P2P program	6	6
	Messanger	2	2
	File manager	2	2
	Shell program	3	3

```
while(true) {
    req = accept();
    resp = handle(req);
    send(resp);
}
```

```
while(true) {
    ev = pending.pop();
    switch(ev.type) {
    case KeyPress:
        ...; break;
    case MouseClick:
        ...; break;
    }
}
```

Enter

One Unit

Exit

High Accuracy Attack Provenance via **B**inary-**b**ased **E**xecution **P**artition. KH Lee, et al. NDSS 2013

Example #3: Event Loop Based Partitioning (2)

- BEEP treats **one iteration of the event loop as an execution unit**
- Detection of inter-dependencies across units becomes crucial
  - Because there are many such dependencies in real-world programs
    - e.g., dispatcher/worker model in multi-threading programs
- To do this, BEEP instruments the memory access instructions
  - They use some heuristic rules to identify so-called *workflow objects*

Listener Thread

```
while(true) {
    req = accept();
    shared_queue.push(req);
}
```

Worker Thread #1

```
while(!exit) {
    req = shared_queue.pop();
    handle(req);
}
```

Worker Thread #2

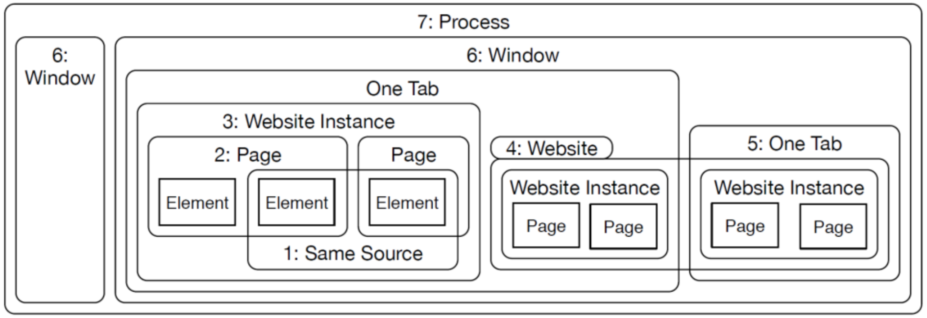
```
while(!exit) {
    req = shared_queue.pop();
    handle(req);
}
```

High Accuracy Attack Provenance via **B**inary-**b**ased **E**xecution **P**artition. KH Lee, et al. NDSS 2013

Example #4: Semantics Aware Partitioning

- MPI argues that event loop iterations are too low-level and may generate excessive execution units
  - e.g., mouse movement/click events
- *The units generated by a ideal partitioning scheme should precisely match with the (multi-perspective)* ***high level logic tasks***
  - e.g., sites/tabs/pages in web browsers, buffers in text editors
- Such high level tasks are usually represented by **data structures**
  - Especially in object-oriented programming languages
  - MPI tracks the runtime dependencies of these data structure instances

Partitioning Perspectives of Firefox



MPI: Multiple Perspective Attack Investigation with Semantics Aware Execution Partitioning. S Ma, et al. USENIX Security 2017

More Examples

- System call
  - MCI: Modeling-based Causality Inference in Audit Logging for Attack Investigation. Y Kwon, et al. NDSS 2018
- Event loop
  - Accurate, Low Cost and Instrumentation-Free Security Audit Logging for Windows. S Ma, et al. ACSAC 2015
- Combining fine-level and coarse-level partitioning
  - RAIN: Refinable Attack Investigation with On-demand Inter-Process Information Flow Tracking. Y Ji, et al. CCS 2017
  - ProTracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting. S Ma, et al. NDSS 2016
  - Kernel-Supported Cost-Effective Audit Logging for Causality Tracking. S Ma, et al. ATC 2018

Outline

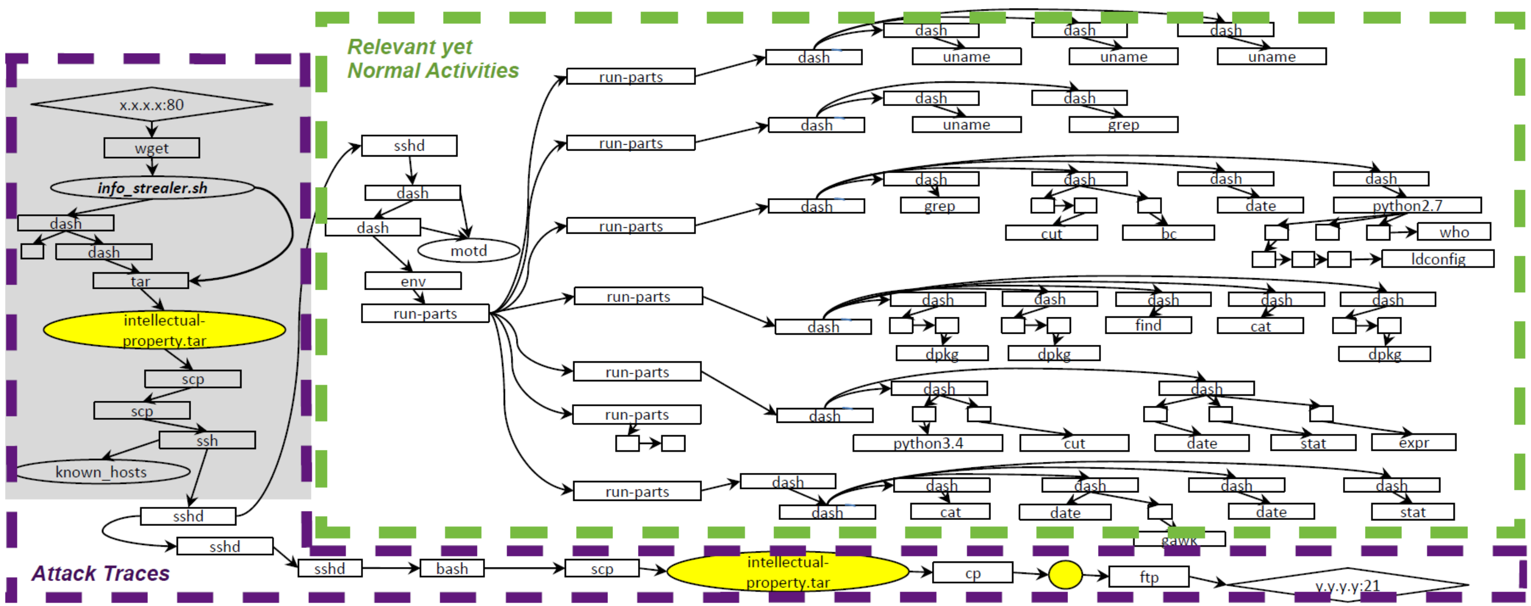
- Introduction to Attack Causality Analysis
- Dependency Explosion and How To Avoid It
- **Selected New Ideas/Topics**
- Conclusions

Selected New Ideas/Topics

- Timely attack causality analysis
  - Towards a Timely Causality Analysis for Enterprise Security. Y Liu, et al. NDSS 2018
- Provenance graph processing for microservices
  - Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs. WU Hassan, et al. NDSS 2018
- Deep learning for anomaly detection & diagnosis from logs
  - DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. M Du, et al. CCS 2017

(1) The Need for Timely Attack Causality Analysis

- It is impractical to fix the dependency explosion problem perfectly
  - Most techniques require source code modification
- There are still large parts of the graph that are just normal activities
  - Even though many false dependencies have been eliminated
- Fully analyzing the whole graph is time-consuming



(1) The Need for Timely Attack Causality Analysis

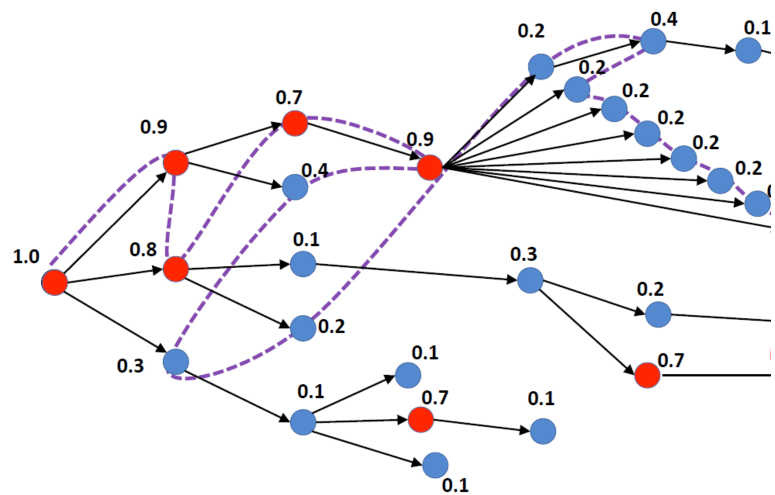
- Attack causality analysis is a considerably **time-sensitive** mission
  - The affected system requires complete cleanup before returning to its normal operation
  - APT attacks are performed in multiple stages. A detected point may not be the very end of attack sequence and the intrusion could further develop to cause more damage
- A **timely** attack causality analysis can:
  - Accelerate the discovery of all attack traces and reduce such recovery cost
  - Help us understand attack intentions and prevent future damage

- So practical causality analysis must take time limit into account and extract useful attack information in a timely manner

Towards a Timely Causality Analysis for Enterprise Security. Y Liu, et al. NDSS 2018

(1) Timely Attack Causality Analysis By Prioritizing

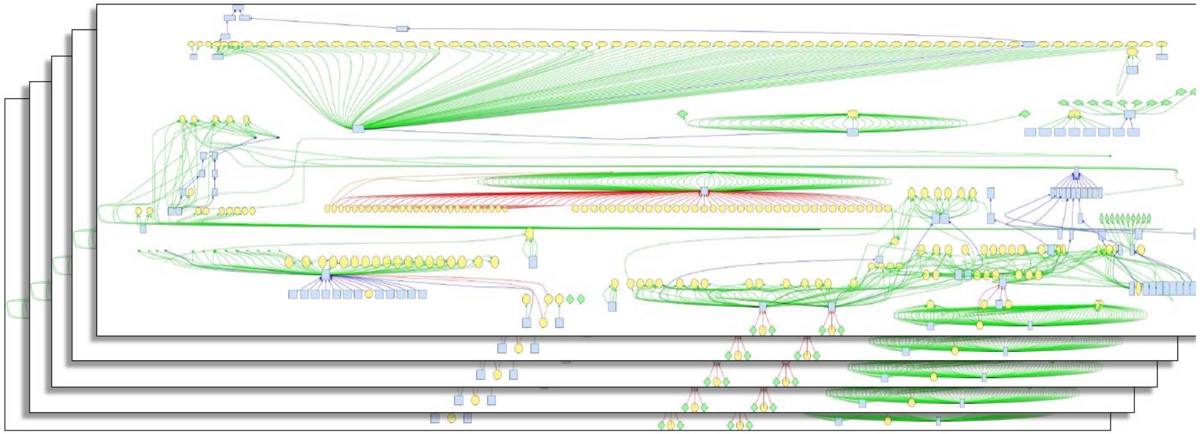
- Idea: **abnormal dependencies should be prioritized** during the exploration of provenance graph
- PrioTracker assigns priorities for edges during graph construction
- Priority Score = $\alpha \times$ rareness score + $\beta \times$ fanout score
  - Abnormal dependencies have higher rareness score
  - Fanout score encourages the searching procedure to expand the search area



Towards a Timely Causality Analysis for Enterprise Security. Y Liu, et al. NDSS 2018

(2) Homogeneous Provenance Graphs in Microservices

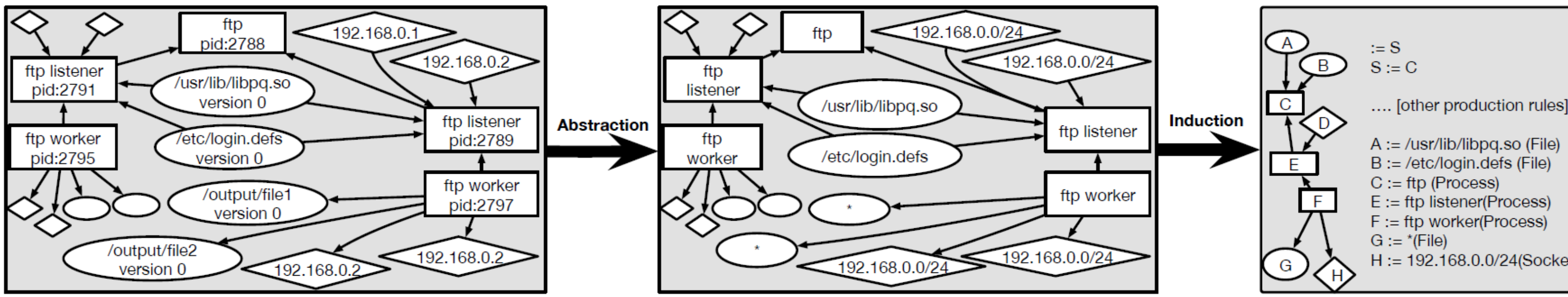
- Nowadays **microservice architecture** has been widely adopted
- Provenance graphs generated by microservice instances are **homogeneous** (i.e., highly redundant)
  - Except in the presence of anomalous activity
- Recognizing and abstracting the equivalent activities of multiply provenance graphs can be beneficial
  - Especially in large clusters for microservice deployment, e.g., public cloud



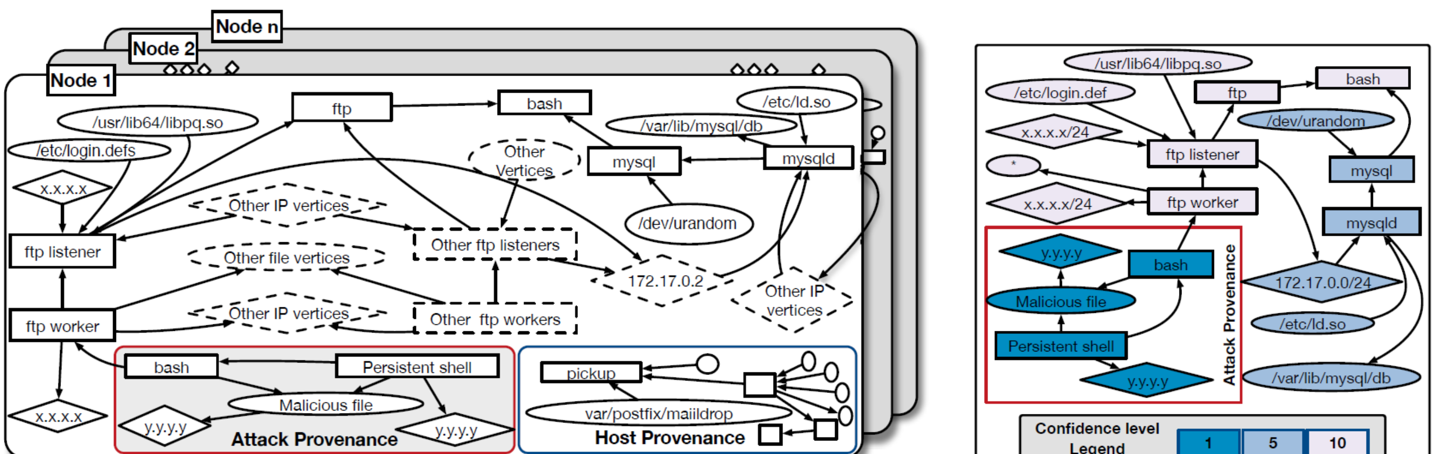
Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs. WU Hassan, et al. NDSS 2018

(2) Simplifying Homogeneous Provenance Graphs

- Remove or generalize **instance-specific information**
  - Using manually defined rules



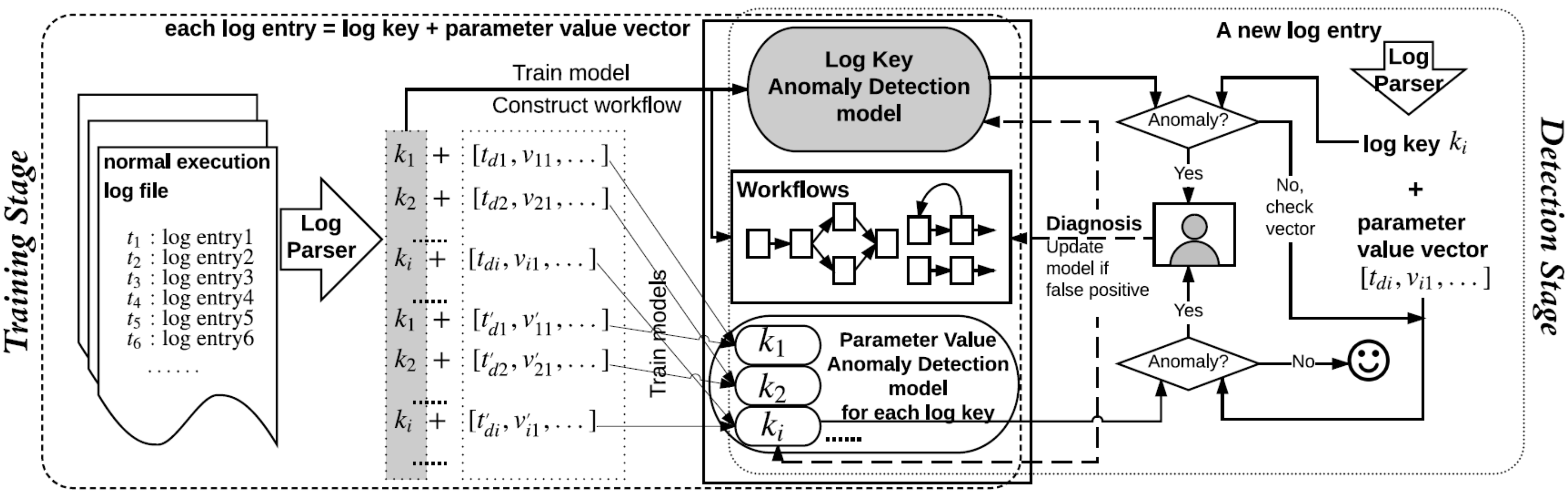
- Merge **repetitive subgraphs** using a grammar-based method



Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs. WU Hassan, et al. NDSS 2018

(3) Deep Leaning for Anomaly Detection & Diagnosis

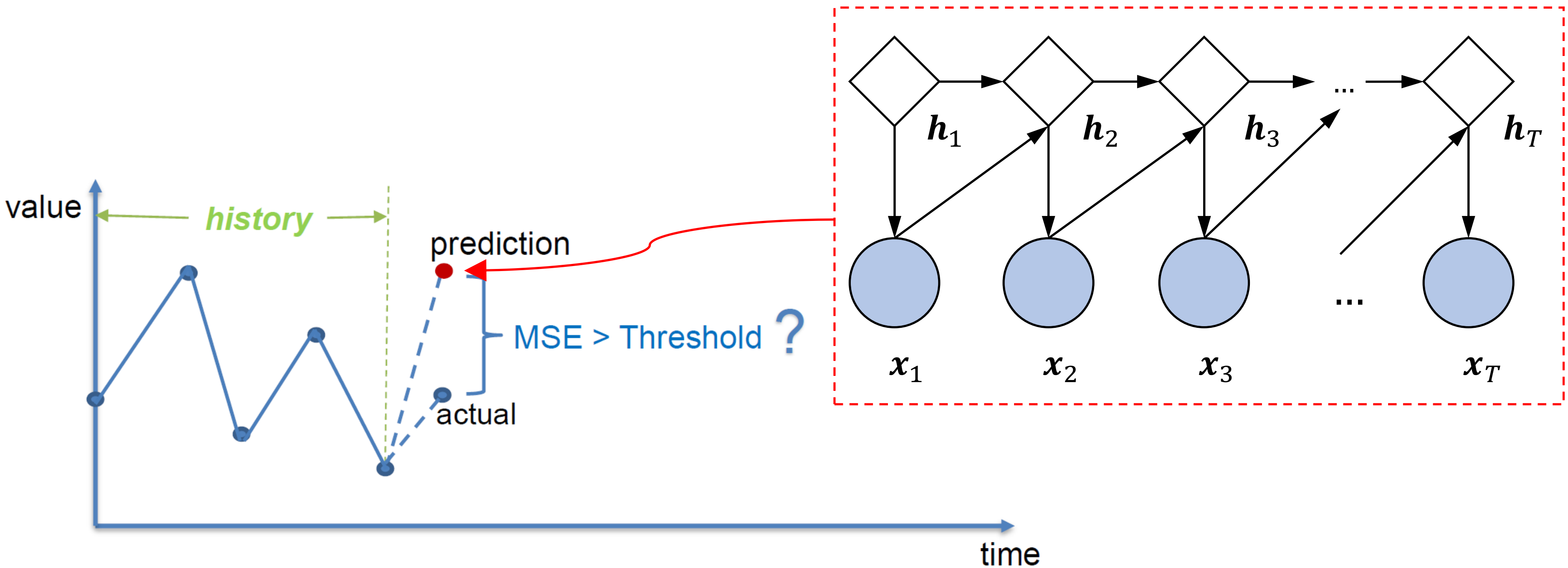
- Parse log files into a structured representation
  - Log key + time series of parameter value: k1 = [(t1, v1), (t2, v2), (t3, v3), ...]
- Train a **R**ecurrent **N**eural **N**etwork model for each log key
  - Using history logs
- Use the RNN model to detect anomalies online



DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. M Du, et al. CCS 2017

(3) Deep Leaning for Anomaly Detection & Diagnosis

- The RNN models are trained with history logs of normal execution
  - k1: [(t1, v1), (t2, v2), (t3, v3), …]
- The actual value is compared with the value predicted by RNN
- If the error ≥ a predefined threshold, it is treated as an anomaly



DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. M Du, et al. CCS 2017

Outline

- Introduction to Attack Causality Analysis
- Dependency Explosion and How To Avoid It
- Selected New Ideas
- **Conclusions**

Conclusions

- Causality analysis is an useful method for attack investigation
- The dependency explosion problem is a major challenge
  - Many solutions have been proposed
  - But there is still a large room for improvement
- Opportunities
  - New scenarios: container, microservices, browser, WebAssembly, …
  - Novel techniques: deep learning, Bayesian networks

References

[1] King, Samuel T., and Peter M. Chen. "Backtracking Intrusions." ACM SIGOPS Operating Systems Review 37.5 (2003): 223-236.

[2] Suh, G. Edward, et al. "Secure Program Execution via Dynamic Information Flow Tracking." ACM Sigplan Notices. Vol. 39. No. 11. ACM, 2004.

[3] Kemerlis, Vasileios P., et al. "libdft: Practical Dynamic Data Flow Tracking for Commodity Systems." Acm Sigplan Notices. Vol. 47. No. 7. ACM, 2012.

[4] Kwon, Yonghwi, et al. "Ldx: Causality Inference by Lightweight Dual Execution." ACM SIGOPS Operating Systems Review 50.2 (2016): 503-515.

[5] Lee, Kyu Hyung, et al. "High Accuracy Attack Provenance via Binary-based Execution Partition." NDSS. 2013.

[6] Ma, Shiqing, et al. "MPI: Multiple Perspective Attack Investigation with Semantics Aware Execution Partitioning." 26th USENIX Security Symposium, 2017.

[7] Liu, Yushan, et al. “Towards a Timely Causality Analysis for Enterprise Security”. NDSS 2018

[8] Hassan, Wajih Ul, et al. "Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs." NDSS. 2018.

[9] Kwon, Yonghwi, et al. "MCI: Modeling-based Causality Inference in Audit Logging for Attack Investigation." NDSS. 2018.

[10] Lee, Kyu Hyung, et al. "LogGC: Garbage Collecting Audit Log." CCS. 2013.

References

[11] Austin, Thomas H., and Cormac Flanagan. "Multiple Facets for Dynamic Information Flow." ACM Sigplan Notices. Vol. 47. No. 1. ACM, 2012.

[12] Capizzi, Roberto, et al. "Preventing Information Leaks through Shadow Executions." Computer Security Applications Conference, 2008.

[13] Baah, George K., et al. "Causal Inference for Statistical Fault Localization." Proceedings of the 19th International Symposium on Software Testing and Analysis. ACM, 2010.

[14] Kim, Dohyeong, et al. "Dual Execution for On The Fly Fine Grained Execution Comparison." ACM SIGARCH Computer Architecture News 43.1 (2015): 325-338

[15] Lee, Kyu Hyung, et al. "High Accuracy Attack Provenance via Binary-based Execution Partition." NDSS. 2013.

[16] Ma, Shiqing, Xiangyu Zhang, and Dongyan Xu. "ProTracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting." NDSS. 2016.

[17] King, et al. "Enriching Intrusion Alerts Through Multi-Host Causality." NDSS. 2005.

[18] Xu, Zhang, et al. "High Fidelity Data Reduction for Big Data Security Dependency Analyses." CCS. 2016.

[19] Zhao, Chunying, et al. "Program Behavior Discovery and Verification: A Graph Grammar Approach." IEEE Transactions on Software Engineering 36.3 (2010): 431-448.

[20] Du, Min, et al. "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning." CCS. 2017.

References

[21] Ji, Yang, et al. "Rain: Refinable Attack Investigation with On-demand Inter-Process Information Flow Tracking." CCS. 2017.

[22] Ma, Shiqing, et al. "Kernel-Supported Cost-Effective Audit Logging for Causality Tracking." USENIX ATC 2018.

[23] Ma, Shiqing, et al. "Accurate, low cost and instrumentation-free security audit logging for windows." ACSAC. 2015.

[24] Vadrevu, Phani, et al. "Enabling Reconstruction of Attacks on Users via Efficient Browsing Snapshots." NDSS. 2017.

[25] Gao, Peng, et al. "AIQL: Enabling Efficient Attack Investigation from System Monitoring Data." arXiv:1806.02290.

[26] Ming, Jiang, et al. "StraightTaint: Decoupled offline symbolic taint analysis." Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. ACM, 2016.

[27] Pei, Kexin, et al. "Hercule: Attack story reconstruction via community discovery on correlated log graph." CCS. 2016.

[28] Pasquier, Thomas, et al. "Practical whole-system provenance capture." SoCC. 2017.

[29] Sun, Xiaoyan, et al. "Using Bayesian Networks for Probabilistic Identification of Zero-Day Attack Paths." IEEE Transactions on Information Forensics and Security 13.10 (2018): 2506-2521.

References

[30] Pilato, Christian, et al. "TaintHLS: High-Level Synthesis For Dynamic Information Flow Tracking." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2018).

[31] Fu, William, Raymond Lin, and Daniel Inge. "TaintAssembly: Taint-Based Information Flow Control Tracking for WebAssembly." arXiv preprint arXiv:1802.01050 (2018).

Questions?