DepComm: Graph Summarization on System Audit Logs for Attack Investigation

Zhiqiang Xu [1, 2], Pengcheng Fang [3], Changlin Liu [3], Xusheng Xiao [3]*, Yu Wen [1]*, Dan Meng [1, 2]
[1]Institute of Information Engineering, Chinese Academy of Sciences, China
[2]School of Cyber Security, University of Chinese Academy of Sciences, China
[3]Department of Computer and Data Sciences, Case Western Reserve University, USA
Email: {xuzhiqiang,wenyu,md}@iie.ac.cn
Email: {pxf109,cxl1029,xusheng.xiao}@case.edu

2022 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-1316-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/SP46214.2022.9833632

Abstract—**Causality analysis generates a dependency graph from system audit logs, which has emerged as an important solution for attack investigation. In the dependency graph, nodes represent system entities (e.g., processes and files) and edges represent dependencies among entities (e.g., a process writing to a file). Despite the promising early results, causality analysis often produces a large graph ($> 100,000$ edges) and it is a daunting task for security analysts to inspect such a large graph for attack investigation. To address challenges in attack investigation, we propose DepComm, a graph summarization approach that generates a summary graph from a dependency graph by *partitioning a large graph into process-centric communities and presenting summaries for each community*. Specifically, each community consists of a set of intimate processes that cooperate with each other to accomplish certain system activities (e.g., file compression), and the resources (e.g., files) accessed by these processes. Within a community, DepComm further identifies redundant edges caused by less-important and repetitive system activities, and perform compression on these edges. Finally, DepComm generates the summary for each community using the *InfoPaths* that represent the information flows across communities. These InfoPaths are more likely to capture a set of attack-related processes that work together to achieve certain malicious goals. Our evaluations on real attacks (~ 150 million events) demonstrate that DepComm generates 18.4 communities on average for a dependency graph, which is $\sim 70\times$ smaller than the original graph. Our compression further reduces the edges in each community to 32.1 on average. Compared with the 9 state-of-the-art community detection algorithms, on average, DepComm achieves a $2.29\times$ better F_1-score than these algorithms in detecting communities. Through cooperating with the automatic techniques HOLMES, DepComm can identify attack-related communities by a recall of 96.2%. Our case studies on the real attacks also demonstrate DepComm's effectiveness in facilitating attack investigation.**

Keywords-**attack investigation; system auditing; graph summarization; community detection**

I. Introduction

Recent cyber attacks have penetrated into many well-protected businesses, causing significant financial losses [1–7]. To counter these attacks, *causality analysis* [8–15] based on *ubiquitous system monitoring* has emerged as an important approach for performing attack investigation [8, 9, 12, 13, 16–19]. System monitoring observes system calls and generates kernel-level audit events as system audit logs. These logs enable causality analysis to identify entry points of intrusions (backward tracing) and ramifications of attacks (forward tracing), which have been shown to be effective in assisting attack investigation and timely system recovery [10, 11, 14, 20, 21].

While early results are promising for causality analysis, existing approaches require non-trivial efforts of manual inspection [14, 22], which hinders their wide adoption. Causality analysis approaches consider system entities (e.g., files, processes, and network connections) that are involved in the same system call event (e.g., a process reading a file) to have causal dependencies. Based on these dependencies, these approaches represent system-call events using a system dependency graph, where nodes represent system entities and edges represent dependencies derived from system events. Using a dependency graph, security analysts can investigate the contextual information of an attack by reconstructing the chain of events that lead to the POI (Point-Of-Interest) event (e.g., an alert event reported by intrusion detection systems [14, 23]). Such contextual information is effective in revealing attack-related events such as distinguishing benign uses of *ZIP* from ransomware [14, 24]. However, due to the dependency explosion problem [25–27], it is hard for security analysts to effectively extract the desirable contextual information from a huge graph (typically containing >100K edges [14, 22]).

Recognizing the challenges of using dependency graphs in attack investigation, recent techniques have been proposed to automatically filter irrelevant events and reveal attack-related events [12–15, 28]. While these techniques achieve promising results, manual attack investigation is still indispensable due to three major reasons. First, in spite of being rare, there are always residual risks in a system, which cannot be accurately revealed by these automation techniques, especially for techniques that heavily rely on system profiles [14]. Second, threats are continually evolving to evade defence techniques, such as emerging attack tactics and techniques lately developed by adversaries. Third, existing techniques mainly rely on heuristic rules that cause loss of information [8, 9] and intrusive system changes [13, 15] such as binary instrumentation, hindering their practical adoption.

Motivation. To effectively assist attack investigation, in this paper, **we aim to develop a graph summarization approach that preserves the semantics of system activities in a dependency graph while shrinking its size by hiding less-important details**. More specifically, we aim to generate a

*Corresponding Author

DOI 10.1109/SP46214.2022.00005

summary graph by dividing a dependency graph into a number of communities (i.e., sub-graphs) and presenting a succinct summary for each community. Each community contains only closely related processes, and they work together to accomplish certain system tasks (e.g., file compression). We then compute the summary using these processes and their accessed resources, which can represent high-level system activities that jointly outline the skeleton of the original dependency graph. Furthermore, our graph summarization can be combined with existing automatic investigation techniques [12–15, 28] to highlight attack-related communities.

Challenges. Graph summarization techniques [29–31] have been shown to be effective in managing large-scale graphs by generating a compact representation of a graph, i.e., *a summary graph*. However, little has been studied on the security implications of summary graphs for attack investigation. In particular, the unique characteristics of dependency graphs pose three major challenges for DEPCOMM to generate summary graphs for dependency graphs.

❶ A dependency graph is a type of heterogeneous graph, where nodes represent different types of system entities (i.e., processes, files, and network connections) and play different roles in attack steps. A general purpose summarization technique that treats each node equally cannot effectively detect the communities to represent major system activities. Additionally, a domain-specific technique of a non-security domain, even if it is designed for heterogeneous graphs, most likely leads to loss of attack information.

❷ Causality analysis [8, 9, 12, 14] relies on the event time to identify causal dependencies (e.g., a process reading a file after another process writing to it) and the dependency graph contains lots of less-important dependencies that represent irrelevant system activities such as chronicle system maintenance tasks and irrelevant web browsing. These dependencies become the majority parts of the dependency graphs and it is a challenging task to compress and hide these activities.

❸ Graph summarization techniques [29–31] mainly deal with the data stored in databases, and their schema and constraints play an important role in the generated summary graphs. But in dependency graphs, a sequence of edges that represent system activities should be the core of the generated summary graphs for attack investigation. How to summarize these edges becomes another challenge for DEPCOMM.

Contributions. To address the aforementioned challenges in graph summarization for dependency graphs, we propose DEPCOMM, *a graph summarization approach that detects process-centric communities, compresses the less-important edges inside each community, and summarizes each community using top-ranked paths that represent information flows among the communities*. DEPCOMM is a general approach that performs graph summarization on large-scale dependency graphs and can cooperate with various automatic investigation techniques [14, 32] to highlight and visualize attack-relevant communities. The design of DEPCOMM is driven by the following key insights.

First, in system audit logs, system activities (e.g., downloading a malicious script and executing the script) are often represented as a set of process nodes that have either strong correlation with each other or data dependencies through some resource nodes. For example, a process `tar` spawns a child process `bzip2` and they work together to compress a file. By carefully examining the cooperation of processes, we observe that these processes (1) either have parent-child relationships (i.e., a process spawning a set of children nodes) or (2) share the same parent process (i.e., sibling processes) and have data dependencies through some resources (e.g., files). We refer to this type of closely related processes as *intimate processes*. Thus, to address the challenge ❶, DEPCOMM *partitions a dependency graph into process-centric communities, where each community includes a group of intimate processes and the system resources accessed by these processes*.

Second, as shown in recent studies [26, 27, 33], there are many redundant edges caused by less-important and repetitive system activities, such as chronicle tasks and backup file updates. Thus, to addressing the challenge ❷, DEPCOMM identifies process-based and resource-based patterns and compresses the edges based on these patterns for each community. Rather than preserving dependencies as the existing work [27, 33], *our community detection allows aggressive compression among multiple processes inside a community*.

Third, by carefully inspecting the dependency graphs of various attacks [12, 13, 15, 18], major system activities (e.g., compressing files) and attack behaviors (e.g., leaking data) are often represented as information flows among attack-related processes, such as compressing sensitive data and leaking the compressed file. Such information flows are often represented as the paths from the input nodes to the output nodes in a community, referred to as *InfoPaths*. For example, a malicious script leaks a sensitive file by packaging, encrypting and uploading, and the corresponding InfoPath is: `../secret.doc`$\rightarrow$`tar`$\rightarrow$`../upload.tar`$\rightarrow$`bzip2`$\rightarrow$`../upload.tar.bz2`$\rightarrow$`gpg`$\rightarrow$`../upload.gpg`$\rightarrow$`curl`$\rightarrow$`xxx->xxx`. Moreover, there could be many InfoPaths from the inputs to the outputs in a community, and not all of them are related to major system activities. Thus, to address the challenge ❸, DEPCOMM *prioritizes the InfoPaths inside each community and ranks the InfoPaths that are more likely to represent attack steps and major system activities at the top*.

Approach. Based on these insights, DEPCOMM provides novel techniques to detect process-centric communities, performs compression inside the detected communities, and generates representative summaries for each community.

To detect process-centric communities (Section IV-C), DEPCOMM learns the behavior representations of a dependency graph's process nodes, and clusters the process nodes with similar representations into a community. Specifically, DEPCOMM performs random walks [34] on each process node to obtain walk routes and computes the behavior representation for each process node by vectorizing these walk routes. Particularly, as existing random walk algorithms [34–38] treat each node equally, they are less effective to generate similar behavior representations for intimate processes. Thus, we design *a series of novel hierarchical walk schemes*, which leverage both the information of the processes' local neighbors and the global process lineage trees [39] to choose the walk routes that are more likely to find intimate processes. With the learned representations for each process node, DEPCOMM clusters these process nodes into communities, and further classifies these processes' accessed resource nodes into the detected communities, producing process-centric communities.

To perform community compression (Section IV-D), DEPCOMM first computes a process lineage tree for each community, and associates each process node with the events that access resource nodes. By searching this tree, DEPCOMM can identify process-based patterns (e.g., a `bash` process spawning multiple `vim` processes) and resource-based patterns (e.g., multiple `vim` processes editing a source file). Based on the identified process-based and resource-based patterns, DEPCOMM merges all of the repeated edges and nodes to compress a community.

After compressing communities, DEPCOMM generates InfoPaths for each community, prioritizes the InfoPaths, and presents the top-ranked InfoPaths as the summary of a community (Section IV-E). To do that, DEPCOMM first identifies the input nodes and the output nodes of each community according to the information flows among communities, and then generates InfoPaths by finding paths for every pair of input and output nodes. Next, DEPCOMM assigns a priority score to each InfoPath based on its likelihood to represent major system activities in the community (e.g., containing the POI event). Finally, DEPCOMM ranks these InfoPaths based on the priorities and shows the top-ranked InfoPaths as the summary for a community. While top-2 InfoPaths can reveal attack behaviors for most communities (Section V-E), security analysts can decide the number of top InfoPaths shown in the summary of a community based on their needs.

Evaluation. We evaluate DEPCOMM on 6 real attacks performed in our lab setting and 8 attacks from the DARPA TC dataset [40]. In total, there are ~150 million system audit events and the generated dependency graphs consist of $1,302.1$ nodes and $7,553.4$ edges on average. In our evaluations, DEPCOMM generates 18.4 communities on average for a dependency graph, which is $\sim 70\times$ smaller than the original graph. These communities contain 43.1 nodes and 248.5 edges on average. Compared with the 9 state-of-the-art community detection algorithms [36, 38, 41–47], the F_1-score achieved by DEPCOMM (94.1%) is averagely 2.29 times better than those achieved by the algorithms. Next, DEPCOMM compresses the communities based on the detected process-based and resource-based patterns, and achieves a compression rate of 44.7% on average. The compressed communities have 15.7 nodes and 32.1 edges on average, which are reduced by 63.6% and 87.1%, respectively. Moreover, all the attacks can be effectively investigated by using the top-2 InfoPaths, i.e., 2 out of the 15.7 found InfoPaths on average (12.7%). These results show that these summary graphs require much less manual effort for attack investigation. Furthermore, the evaluation of cooperating with HOLMES [32] shows that all the attack-related communities except two ones are mapped to the steps in Kill Chain [48] (achieving a recall of 96.2%), and these two unrevealed communities can be easily recognized by considering the adjacent attack communities. Our implementation of DEPCOMM and the evaluation datasets are available at our project website [49].

II. BACKGROUND AND MOTIVATION

A. System Event and Causality Analysis

System Audit Event: Monitoring and analyzing kernel-level audit events are crucial for attack investigation and detection. System auditing events describe the interactions between two system entities, which are represented as 3-tuple $\langle$*subject, operation, object*$\rangle$. According to the previous work [8, 9, 12, 13, 16–19], subjects represent process entities, and objects represent process, file, or network entities. System audit events are categorized based on the types of their objects: *process events*, *file events*, and *network events*. *Process events* record the operations of processes, such as fork and clone. *File events* record the operations on files, such as files read, write, and rename. *Network events* record the operations of network accesses, such as send and receive messages from sockets.

Causality Analysis: Causality analysis has been widely applied to attack investigation and detection [8–15]. It infers the causal dependencies among system audit events, and organizes them as a dependency graph. A dependency graph is a directed graph, where nodes represent system entities (i.e., processes, files, and network connections) and edges represent system audit events. In a dependency graph $G(E, V)$, a system audit event is denoted as a directed edge $e(u, v)$, where $u \in V, v \in V$, $e \in E$, and the direction of the edge represents the direction of data flow (i.e., flowing from u to v). In addition, the edge records the start time ($e.st$) and the end time of the event ($e.et$). Given two nodes n_1 and n_2, n_2 has a causal dependency on n_1 if there exist two edges $e_1(n_1, v_1)$ and $e_2(v_2, n_2)$ such that $v_1 = v_2$ and $e_1.st \leq e_2.et$.

B. Motivating Example

We use a data exfiltration attack as an example to motivate DEPCOMM. In this attack, the attacker downloads and executes a backdoor program `bdoor` in a target system through an Apache Sever, and opens a terminal (i.e., `bash`) via exploiting the opened backdoor at the port 9999. The attacker then downloads an executable script `leak.sh`, and exploits the root access to run the script to collect sensitive files, which are sent to a suspicious remote host. All these activities among processes and OS resources are captured in the system audit logs. We construct the dependency graph by applying causality analysis from the suspicious event that sends the files to the remote IP (i.e., the POI event). Fig. 1 shows a part of the dependency graph. The complete dependency graph has 1,038 nodes and 4,039 edges, including attack-related and benign events. As we can see, it requires non-trivial efforts

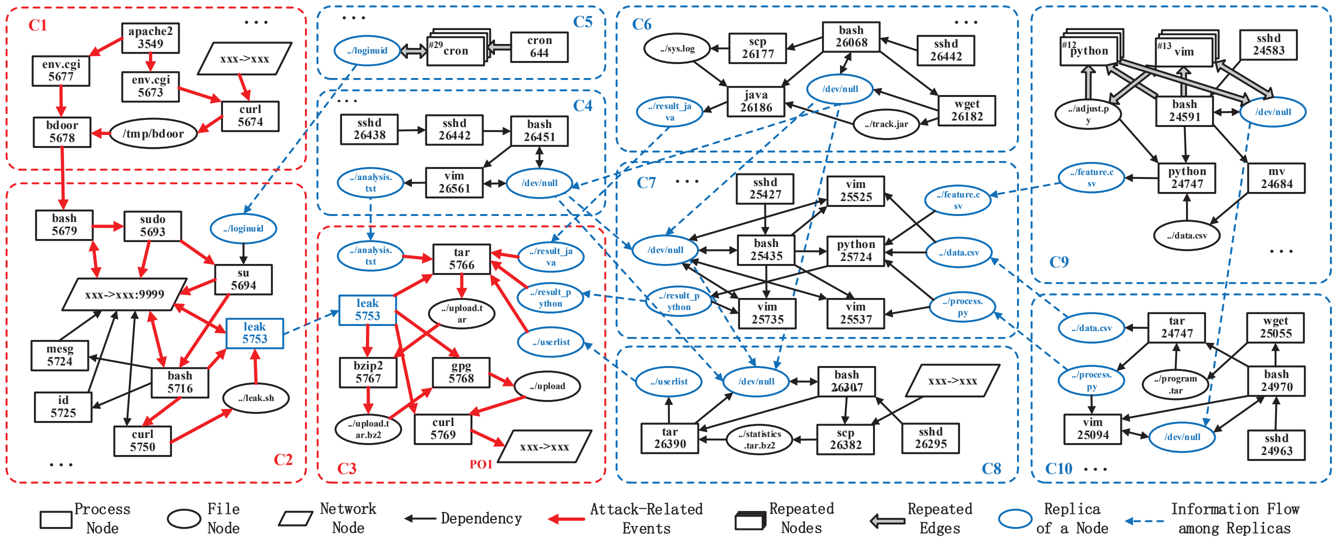


Fig. 1: Partial dependency graph for a data exfiltration attack. The complete dependency graph has 1,038 nodes and 4,039 edges. DEPCOMM partitions the dependency graph into 10 *process-centric communities*, where the red dashed frames are the communities with attack-related events (bold red edges), and the blue dashed frames are the communities with only normal events. For the nodes that represent the inputs and the outputs of communities, we create replicas of such nodes (blue nodes) and assign each copy to a community. These replicas are connected with directed edges (dashed blue arrows), where the direction indicates the direction of the information flow across communities.

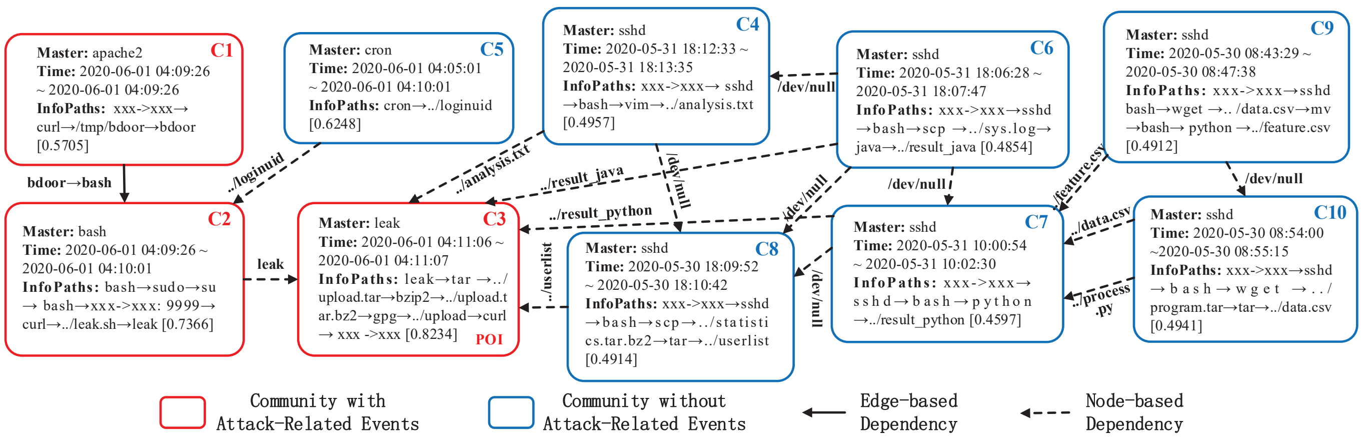

C1
Master: apache2
Time: 2020-06-01 04:09:26 ~ 2020-06-01 04:09:26
InfoPaths: xxx->xxx→ curl→/tmp/bdoor→bdoor [0.5705]
C5
Master: cron
Time: 2020-06-01 04:05:01 ~ 2020-06-01 04:10:01
InfoPaths: cron→../loginuid [0.6248]
C4
Master: sshd
Time: 2020-05-31 18:12:33 ~ 2020-05-31 18:13:35
InfoPaths: xxx->xxx→ sshd →bash→vim→../analysis.txt [0.4957]
C6
Master: sshd
Time: 2020-05-31 18:06:28 ~ 2020-05-31 18:07:47
InfoPaths: xxx->xxx→sshd →bash→scp →../sys.log→ java→../result_java [0.4854]
C9
Master: sshd
Time: 2020-05-30 08:43:29 ~ 2020-05-30 08:47:38
InfoPaths: xxx->xxx→sshd bash→wget →.. /data.csv→mv →bash→ python →../feature.csv [0.4912]
bdoor→bash
../loginuid
../analysis.txt
/dev/null
../result_java
/dev/null
/dev/null
/dev/null
../feature.csv
C2
Master: bash
Time: 2020-06-01 04:09:26 ~ 2020-06-01 04:10:01
InfoPaths: bash→sudo→su → bash→xxx->xxx: 9999→ curl→../leak.sh→leak [0.7366]
leak
C3
Master: leak
Time: 2020-06-01 04:11:06 ~ 2020-06-01 04:11:07
InfoPaths: leak→tar →../ upload.tar→bzip2→../upload.t ar.bz2→gpg→../upload→curl → xxx ->xxx [0.8234]
POI
../result_python
../userlist
C8
Master: sshd
Time: 2020-05-30 18:09:52 ~ 2020-05-30 18:10:42
InfoPaths: xxx->xxx→sshd →bash→scp→../statisti cs.tar.bz2→tar→../userlist [0.4914]
/dev/null
C7
Master: sshd
Time: 2020-05-31 10:00:54 ~ 2020-05-31 10:02:30
InfoPaths: xxx->xxx→ sshd→bash→python →../result_python [0.4597]
../data.csv
../process.py
C10
Master: sshd
Time: 2020-05-30 08:54:00 ~2020-05-30 08:55:15
InfoPaths: xxx->xxx→sshd →bash→wget →../ program.tar→tar→../data.csv [0.4941]
Community with Attack-Related Events
Community without Attack-Related Events
Edge-based Dependency
Node-based Dependency

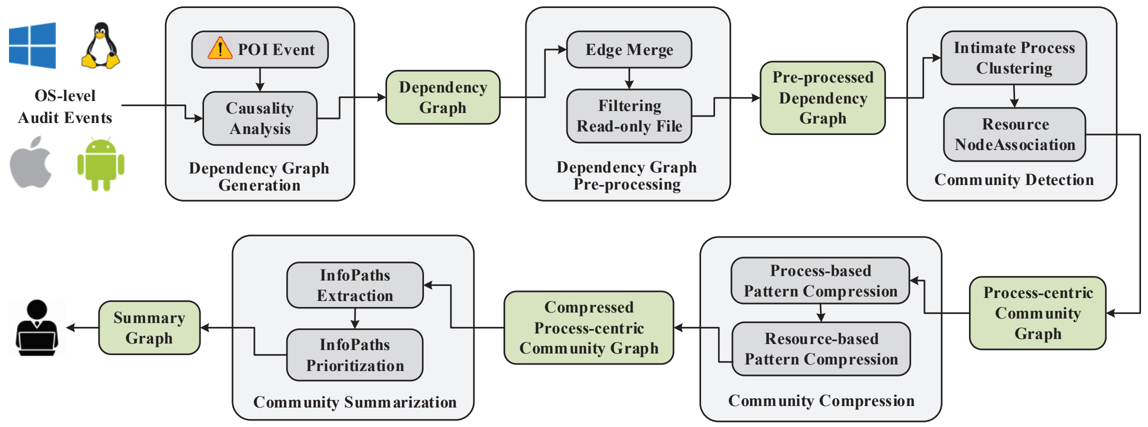
Fig. 2: Summary graph for the dependency graph in Fig. 1

for security analysts to understand the dependencies among nodes by inspecting such a large graph.

In this paper, we design DEPCOMM to summarize a large dependency graph into a compact graph that can facilitate attack investigation. DEPCOMM includes three key components: (1) community detection, (2) community compression, and (3) community summarization.

Community Detection. DEPCOMM first partitions the dependency graph into 10 process-centric communities (C1–C10), as shown in Fig. 1. Each community consists of a set of intimate process nodes and their accessed resource nodes. For example, in C3, `leak` spawns `tar`, `bzip2`, `gpg` and `curl`, and thus `leak` has parent-child relationships with these child process nodes. Moreover, the resource nodes `../upload.tar`, `../upload.tar.bz2`, `../upload` and `xxx->xxx` are accessed by these process nodes and thus are classified into C3. Additionally, dependencies betwen communities are either (1) *edge-based dependencies* that represent the inter-community edges (e.g., `bdoor→bash` between C1 and C2) or (2) *node-based dependencies* (blue ovals in Fig. 1) that indicate the input/output relationships between communities (e.g., `leak` in C2 and C3).

Community Compression. To further decrease the size of each community yet preserving their semantics, DEPCOMM compresses less-important and redundant dependencies in a community, including nodes and edges represented with the stacked shapes and the hollow arrows in Fig. 1. For example, in C9, `bash` repetitively spawns `python` (12 times) and `vim` (13 times) to read and write `../adjust.py` and `/dev/null`, which can be summarized as a process-based pattern (i.e., `bash` creating many `python` and `vim` nodes) and a resource-based pattern (e.g., `../adjust.py` accessed by many `python` and `vim` nodes). After compression, the number of edges of C9 decreases to 33 from 108 (69.4% compression rate). Similarly, C5 is compressed into 2 edges from 58 edges (96.5% compression rate).



OS-level Audit Events
POI Event
Causality Analysis
Dependency Graph Generation
Dependency Graph
Edge Merge
Filtering Read-only File
Dependency Graph Pre-processing
Pre-processed Dependency Graph
Intimate Process Clustering
Resource NodeAssociation
Community Detection
Process-centric Community Graph
Process-based Pattern Compression
Resource-based Pattern Compression
Community Compression
Compressed Process-centric Community Graph
InfoPaths Extraction
InfoPaths Prioritization
Community Summarization
Summary Graph

Fig. 3: The architecture of DepComm

Community Summarization. As shown in Fig. 2), for each community, DepComm generates a summary, which consists of three parts: (1) a master process node that is the source of a community's behaviors (e.g., `leak` in C3), (2) the time span between the start time of the earliest event and the end time of the latest event in a community, and (3) the top-ranked InfoPaths to show major information flows of a community. The top-ranked InfoPath is a dependency path from `leak` to `xxx->xxx`: `leak`→`tar`→`../ upload.tar`→`bzip2`→`../upload.tar .bz2`→`gpg`→`../upload`→`curl`→`xxx->xxx`, which has the highest priority because it includes the POI event `curl`→`xxx->xxx`.

Attack Investigation. We next show how to use the summary graph in Fig. 2 to investigate an attack from the POI event `curl` →`xxx->xxx` in C3. First, by inspecting all of the edges from other communities (C2, C4, C6, C7, C8) to C3, we find that the edge from C2 is more relevant to the master process node `leak` of C3 and the top-1 InfoPath of C3 than those from the other communities. Thus, C2 is considered to have attack-related events that lead to the POI event. Next, for the edges between C2 and other communities, we identify the edge `bdoor`→`bash` from C1 is more relevant than the edge from C5. Moreover, there exists no dependencies from other communities to C1. Therefore, C1 is likely to represent the initial steps of the attack. In summary, we identify 35 attack-related events by inspecting only 53 nodes and 8 InfoPaths, which shows a great reduction of manual efforts.

III. Overview and Threat Model

Fig. 3 shows the architecture of DepComm. DepComm consists of five components: (1) Dependency Graph Generation, (2) Dependency Graph Pre-processing, (3) Community Detection, (4) Community Compression, and (5) Community Summarization.

The dependency graph generation component leverages causality analysis to compute a dependency graph from system audit events (Section IV-A). The dependency graph pre-processing component processes the graph by merging the same types of edges between two nodes and filtering out read-only file nodes (Section IV-D). The community detection component partitions the graph into multiple process-centric communities and associates the resource nodes to the communities (Section IV-C). The community compression component compresses the nodes and the edges in each community based on the identified process-based patterns and resource-based patterns (Section IV-D). The community summarization component extracts InfoPaths for each community and prioritizes these InfoPaths. Finally, DepComm generates a summary graph with top-ranked InfoPaths (Section IV-E).

Threat Model: We follow the same threat model as the previous works on security investigation [8, 9, 25, 50, 51]. OS-level events are collected from the system kernel. We assume that the system kernel is trusted and not tampered by adversaries [52, 53]. Any kernel-level attacks that deliberately compromise security auditing systems are beyond the scope of this work, and existing software and kernel hardening techniques [50, 54–56] can be used to better protect log storage. We also do not consider the attacks performed using side channels or inter-procedural communications (IPC) that cannot be captured by the underlying provenance tracker. Finer-grained auditing tools that capture memory traces or side channel analysis techniques can be used to address these attacks and they are not the focus of this work.

DepComm clusters the system behaviors into communities and prioritizes InfoPaths that represent the information flows across communities. Thus, attackers who have full knowledge of DepComm's summarization approach may deliberately limit their attack within a few processes and files, minimizing their traces within a community and across communities. Such attacks typically compromise the processes by manipulating the memories of the processes (e.g., code reuse attacks [57]), and specialized techniques such as memory randomization [58, 59] can be applied to strengthen the memory protection. Attackers may also flood system audit logs by performing activities that generate a large amount of logs, e.g., creating lots of temporary files. To defend against such attacks, existing log compression techniques [26, 27, 33, 60] can be employed to compress system audit logs, and DepComm can work seamlessly on the compressed logs since these compressed log preserve the dependencies. Furthermore, anomaly detection techniques [14, 61] can be deployed to raise alerts for such unexpected spikes in log collection.

TABLE I: Attributes of system entities

Entity	Attributes
Process	PID, Name, User, Cmd
File	Name, Path
Network	IP, Port, Protocol

TABLE II: Attributes of system events

Event	Attributes
Process Event	Start time, End time, Subject ID, Object ID, Operations (execve, fork, clone)
File Event	Start time, End time, Subject ID, Object ID, Operations (write, read, rename, readv, writev)
Network Event	Start time, End time, Subject ID, Object ID, Operations (write, read, recvmsg, sendto, recvfrom)

IV. Design of DepComm

A. Dependency Graph Generation

DepComm uses system monitoring tools that run on mainstream operation systems (e.g., Windows, Linux, Mac OS and Android) to collect system audit events, including *process events*, *file events*, and *network events*. For each collected entity and event, DepComm records the attributes that are essential for security analysis (e.g., PID, file name and IP for entities; start time, end time and operation for events), as shown in Table I and Table II. Given a POI event (e.g., an alert about a file download), DepComm builds a dependency graph by performing backward causal analysis to track the dependencies. Starting from the POI event, the causal analysis iteratively finds the events that have dependencies to the POI event and happen before the POI event. These found events (i.e., edges) form the dependency graph for the POI event, such as the graph shown in Fig. 1.

B. Dependency Graph Pre-processing

Edge Merge: A dependency graph often has many parallel edges between a process node and a file/network node, indicating repetitive read/write operations in a short period. This is because OS typically performs a read/write task by distributing the data proportionally to multiple system calls. As shown in the recent study [26], these parallel edges do not offer extra useful information for attack investigation, and thus DepComm directly merges the parallel edges of the same operation type into one edge.

Filtering Read-only file nodes: As shown in recent studies [33, 51], a dependency graph has many read-only files, which are typically libraries, configuration files, and resources (e.g., `/lib64/libdl.so.2`) for process initialization that do not contain useful attack-related information [33]. Thus, DepComm filters out read-only files and retains the processes to preserve the semantics of major system activities.

C. Community Detection

DepComm identifies a group of intimate processes as a process-centric community. ***A process-centric community is a graph that contains (1) one master process node, (2) a set of child process nodes that represent a subset of the master process' spawned child processes such that these child processes have data dependencies among each other, and (3) a set of resource nodes accessed by the master processes and these child processes.*** For example, `leak` in

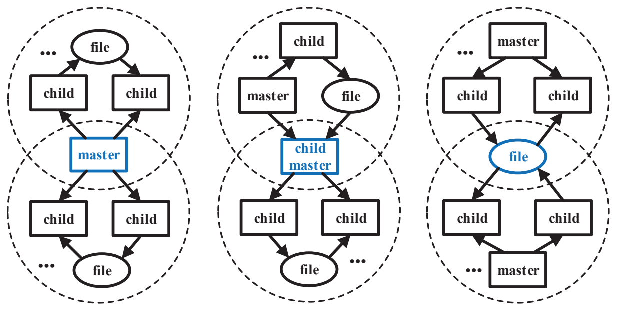


Fig. 4: Three types of overlapping nodes

Fig. 1 is the master process of C3, which spawns the child processes `tar`, `bzip`, `gpg`, and `curl` to compress and upload a file. These child processes have data dependencies with at least another child process, as reflected by the following path in the dependency graph: `tar`→`../upload.tar`→`bzip2` →`../upload.tar.bz2`→`gpg`→`../upload`→`curl`→`xxx->xxx`. Additionally, there are processes or resources that can belong to more than one communities and are referred as *overlapping nodes*. For example, in Fig. 1, `leak` first cooperates with `curl` to accomplish the execution of the script `leak.sh` in C2, and then spawns child processes `tar`, `bzip2`, `gpg` and `curl` to compress and upload a file in C3. In this case, `leak` is the overlapping node in both C2 and C3. We categorize overlapping nodes into three types as shown in Fig. 4:

ⓐ a process node that cooperates with different sets of child processes for different system activities;

ⓑ a process node that cooperates with its siblings to accomplish a system activity, and meanwhile spawns child processes to accomplish a different system activity;

ⓒ a resource node accessed by process nodes from different communities.

We next describe the two phases of the community detection component of DepComm: process-centric community detection and resource node association.

Process-Centric Community Detection. DepComm performs random walks on each process node based on our proposed hierarchical walk schemes to generate walk routes, and then applies a word2vec model [62] to learn the behavior representation based on the walk routes for each process node. Based on the behavior representations, DepComm clusters the process nodes with similar representations into the same communities. We next describe each step in detail.

1) Hierarchical Random Walk. A random walk rooted from a node v_1 generates a walk route of a specific length $\mathbb{W} = \{v_1, \cdots, v_l\}$, where $v_i \in \mathbb{W}$ is randomly chosen with a transition probability [34]. The transition probability from v_i to its neighbor node n is $Pr(v_i, n) = w(v_i, n)/W_{N(v_i)}$, where $w(v_i, n)$ denotes the walk weight from v_i to n, and $W_{N(v_i)}$ denotes the sum of walk weights among all the neighbors of v_i. Unlike existing random walk algorithms that treat neighbor nodes with equal probabilities [34], the walker in DepComm gives higher probabilities to v_i's neighbors that are more likely to be its intimate processes.

Specifically, the walker considers *both the processes' neigh-*

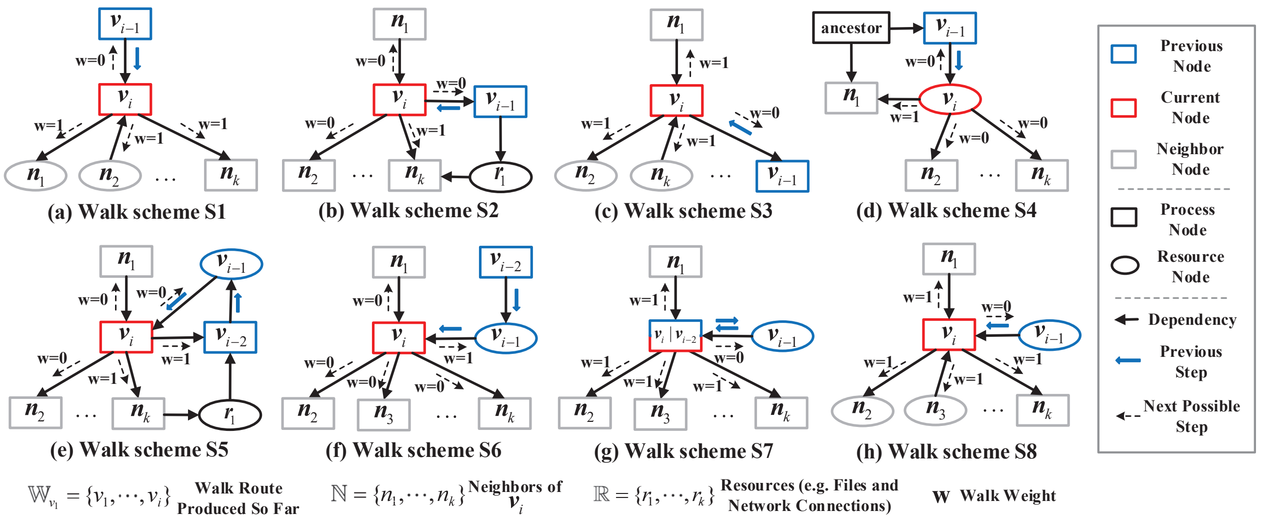


Fig. 5: Hierarchical walk schemes, where w denotes the walk weight.

bors and the global process lineage trees to ensure that intimate processes are more likely to be sampled into the same walk route, and thus they will have the similar contexts. For each process node p, DEPCOMM examines p's one-hop neighbor nodes, and associates p with: ① parent process node, ② child process nodes, and ③ accessed resource nodes. In particular, we observe that for a process p_r and its child process p_c, if p_c starts to spawn its own child processes (typically more than one child processes), p_c is very likely to initiate a new system task and the spawned child processes do not cooperate with p_r. Thus, the child processes of p_c should be in a different community from p_r's. To identify such creations of child processes, DEPCOMM searches the global process lineage trees and associates each process node with ④ its lowest ancestor that has multiple child processes.

With the information collected from the neighbors of the processes and the global process lineage trees (i.e.,①②③④), DEPCOMM employs 8 hierarchical walk schemes to generate walk routes. Specifically, when the walker starts at a process node v_1, it assigns equal weights to each of neighbors and randomly move to one of them. After this initial step, the walker chooses the next node based on 8 hierarchy walk schemes, as shown in Fig. 5. Without loss of generality, we assume that the walker is currently at the node v_i, the walk route produced so far is $\mathbb{W}_{v_1} = \{v_1, \cdots, v_i\}$, the neighbors of v_i is a set of nodes $\mathbb{N} = \{n_1, \cdots, n_k\}$, $\mathbb{R}(v)$ returns the resources accessed by a process node v, and $\mathbb{L}(v)$ finds the process node v's lowest ancestor that has multiple child processes. We next describe the walk schemes in detail:

- **Scheme S1:** Consider that v_{i-1} presents the parent process of v_i. If $\mathbb{N}$ contains other neighbor nodes except v_{i-1}, the walker will randomly walk to one of these neighbors, i.e., $\forall n_j \in \mathbb{N}, n_j \neq v_{i-1}, w(v_i, n_j) = 1, w(v_i, v_{i-1}) = 0$. If v_i has only one neighbor (i.e., v_{i-1}), to avoid the early termination of the walk, the walker will return to v_{i-1}, i.e., $w(v_i, v_{i-1}) = 1$.
- **Scheme S2:** Consider that v_{i-1} represents a child process of v_i. In this case, other child processes of v_i may not belong to the same community as v_{i-1}, unless they have data dependencies with v_{i-1}. Thus, if there are other child processes that access the same resources as v_{i-1}, the walker will walk to the child process nodes with a high probability, i.e., $\forall n_j \in \mathbb{N}, n_j \neq v_{i-1}, \mathbb{R}(n_j) \cap \mathbb{R}(v_{i-1}) \neq \emptyset, w(v_i, n_j) = 1, w(v_i, v_{i-1}) = 0$. Otherwise, the walker will return to v_{i-1}, i.e., $w(v_i, v_{i-1}) = 1, w(v_i, n_j) = 0$.
- **Scheme S3:** Consider that v_{i-1} represents a child process of v_i and v_{i-1} is the only child process of v_i. This indicates that v_i and v_{i-1} cooperate to process some data, and thus they belong to the same community. Thus, if there are other neighbors except v_{i-1}, the walker will continue to explore without return, i.e., $\forall n_j \in \mathbb{N}, n_j \neq v_{i-1}, w(v_i, n_j) = 1, w(v_i, v_{i-1}) = 0$. Otherwise, to avoid the early termination of the walk, the walker will return to v_{i-1}, i.e., $w(v_i, v_{i-1}) = 1$.
- **Scheme S4:** Consider that v_{i-1} is a process node and v_i is a resource node. The processes accessing v_i may belong to the v_{i-1}'s community if v_{i-1} and these processes have a common parent process. Thus, we let the walker walk to the neighbors that share the same parent process as v_{i-1} with a high probability, i.e., $\forall n_j \in \mathbb{N}, n_j \neq v_{i-1}, \mathbb{L}(n_j) = \mathbb{L}(v_{i-1}), w(v_i, n_j) = 1, w(v_i, v_{i-1}) = 0$. Otherwise, the walker will return to v_{i-1}, i.e., $w(v_i, v_{i-1}) = 1, w(v_i, n_j) = 0$.
- **Scheme S5:** Consider that v_{i-1} is a resource node, v_i is a process node with more than one child processes, and v_{i-2} is the child process of v_i. In this case, other child processes of v_i may not belong to the community as v_{i-2}, unless they have data dependencies with v_{i-2}. Thus, the walker will walk to v_{i-2} and the child process nodes that access the same resources as v_{i-2}, i.e., $\forall n_j \in \mathbb{N}, n_j \neq v_{i-1}, n_j \neq v_{i-2}, \mathbb{R}(n_j) \cap \mathbb{R}(v_{i-2}) \neq \emptyset, w(v_i, n_j) = 1, w(v_i, v_{i-2}) = 1$, and the weights of the other neighbors are set to 0.
- **Scheme S6:** Consider that v_{i-1} is a resource node, v_i is a process node with more than one child processes, and v_{i-2} is not the child process of v_i. In this case, we treat v_i as a master process of a community, and the child processes

of v_i and v_{i-2} do not belong to the same community. Thus, the walker will return to v_{i-1}, i.e., $\forall n_j \in \mathbb{N}, n_j \neq v_{i-1}, w(v_i, n_j) = 0, w(v_i, v_{i-1}) = 1$.

- **Scheme S7:** Consider that v_{i-1} is a resource node, v_i is a process node with more than one child processes, and $v_{i-2} = v_i$. This indicates that v_{i-1} is the end of an information flow. To increase the efficiency of sampling intimate processes, the walker will walk without return, i.e., $\forall n_j \in \mathbb{N}, n_j \neq v_{i-1}, w(v_i, n_j) = 1, w(v_i, v_{i-1}) = 0$.
- **Scheme S8:** Consider that v_{i-1} is a resource node and v_i is a process node with at most one child process. In this case, if v_i has other neighbor nodes except v_{i-1}, the walker will walk to the neighbor nodes without return, i.e., $\forall n_j \in \mathbb{N}, n_j \neq v_{i-1}, w(v_i, n_j) = 1, w(v_i, v_{i-1}) = 0$. If v_{i-1} is the only neighbor of v_i, to avoid the early termination of the walk, the walker will return to v_{i-1}, i.e., $w(v_i, v_{i-1}) = 1$.

2) Process Node Representation. We make an analogy by regarding nodes in a dependency graph as words and walk routes as ordered sequences of words. DepComm employs SkipGram [62], a widely-used word representation learning algorithm, to learn the behavior representation of process nodes in walk routes. More specifically, given a process node p and a contextual window size t, SkipGram extracts the sub-sequence $\mathbb{W}_p = \{v_{i-t}, \cdots, v_i, \cdots, v_{i+t}\}$ that consists of $v_i = p$ and its contextual nodes v_{i+k} ($k \in (-t, t)$) from each walk route containing p. Then, the d-dimension vector $\Phi(v_i)$ of v_i is learned by maximizing the log-probability of any node appearing in the sub-sequences, i.e., $logPr(\{v_{i-t}, \cdots, v_{i-1}, v_{i+1}, \cdots, v_{i+t}\}|\Phi(v_i))$. The optimization process aims to learn similar behavior representations for intimate process nodes with the similar contextual nodes. However, the optimization problem is NP-hard. To make the optimization problem tractable, we assume that the probabilities of choosing each node are conditional independent, and the objective function is converted into: $log \prod_{-t \leq k \leq t, k \neq 0} Pr(v_{i+k}|\Phi(v_i)$. Further, the objective function is modeled using the softmax function: $log \prod_{-t \leq k \leq t, k \neq 0} \frac{exp(\Phi(v_{i+k}) \cdot \Phi(v_i))}{\sum_{v \in V} exp(\Phi(v) \cdot \Phi(v_i))}$ [37]. However, it is still expensive to solve this optimization for a large graph, and thus we further use NEG (Negative Sampling) function [63] to approximate it. The model parameters for $\Phi(v_i)$ is adjusted using stochastic gradient ascent.

3) Process Node Clustering. To compute the overlapping clustering for process nodes based on their behavior representations, DepComm employs a soft clustering method, FCM (Fuzzy C-Means) [64]. Unlike the hard clustering method (i.e., K-means) that classifies a process node to only one cluster, FCM outputs the membership degree of each process node in each cluster by minimizing the objection function: $J = \sum_{i=1}^{|V_p|} \sum_{j=1}^{|C|} u_{ij}^2 ||v_i - c_j||^2$, where u_{ij} denotes the degree of a process node v_i belonging to a community c_j. v_i is classified to c_j, if u_{ij} is higher than a given threshold. Following the recent work [65], we set the threshold $\lambda = 0.8 \cdot max_j\{u_{ij}\}$. If a process node is labeled with

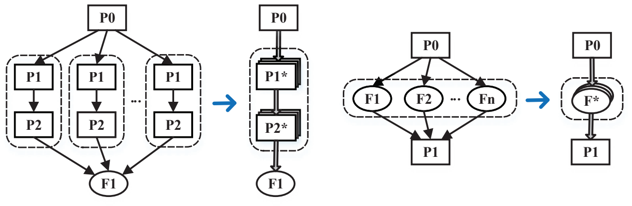


Fig. 6: Community compression based on (a) a process-based pattern and (b) a resource-based pattern

multiple communities (i.e., overlapping), we create multiple replicas of the node, and assign one replica to each community. In addition, DepComm determines the number of communities $|C|$ according to the fuzzy partition coefficient (FPC) $F(|C|) = 1/|V_p| \sum_{j=1}^{|C|} \sum_{i=1}^{|V_p|} u_{ij}^2$ [66], which is used to measure the cluster validity for different numbers of clusters. As a higher value of FPC indicates a better description for the data distribution, DepComm selects the number of clusters $|C|$ with the maximum FPC, i.e., $|C| = argmax(F(|C|))$.

Resource Node Association. Given a resource node r and a process node p, if they are connected by an edge, then v is associated to the community that p belongs to. If a resource node are connected with multiple process nodes from different communities, this resource node is an overlapping node, and we create replicas of the resource node and assign a replica to each community.

Dependencies across Communities. We categorize dependencies across communities as *edge-based dependencies* (i.e., the dependency represented by an inter-community edge between communities) and *node-based dependencies* (i.e., the dependency represented by overlapping nodes). As these nodes lack visible information flow directions for security analysts, DepComm creates a directed edge to connect the replicas (e.g., the blue dashed arrows in Fig. 1). Specifically, given two replicas $v_{1(1)}$ and $v_{1(2)}$ of a node v_1, where $v_{1(1)}$ is in the community C_i and $v_{1(2)}$ is in the community C_j, if $v_{1(1)}$ has an in-edge e_1, $v_{1(2)}$ has an out-edge e_2, and the start time of e_1 is earlier than the end time of e_2, then we create an directed edge from $v_{1(1)}$ to $v_{1(2)}$.

D. Community Compression

Process-based Patterns. Process-based patterns describe repetitive activities that spawn same set of processes to process some resources. Fig. 6(a) shows an example: a process `P0` repetitively spawns the child processes named `P1` and `P2` to write the file `F1`. But keeping repetitive activities do not provide extra values for security analysts. Thus, DepComm aims to identify such patterns and merge the repeated nodes and edges. This process includes the following four steps:

- **Step 1:** *Building process lineage tree.* Given a process-centric community, DepComm builds a process lineage tree rooted from the master process of the community by traversing the process nodes inside the community. The process lineage tree can present processes' spawning behaviors.
- **Step 2:** *Association with Accessed Resources.* To capture the resource usage of each process in the process lineage

tree, DepComm inspects the events inside the community to identify the resources accessed by these processes. Specifically, each process is associated with the representative attributes of the accessed resources (i.e., file names for files and IPs for network connections) and the operation types on these resources.

- **Step 3:** *Mining Process-based Patterns*. A process-based pattern in the process lineage tree is the repeated bottom-up sub-trees [67]. , where a bottom-up sub-tree includes a node and all its descendants. Unlike induced sub-tree and embedded sub-tree that have partial descendants [68], a bottom-up sub-tree can present a complete process spawning activity. Specially, DepComm uses the process lineage tree to generate a sub-tree for each process node. Then, DepComm encodes a sub-tree to a string by appending the associated resource attributes of the process nodes in the sub-tree, and identifies identical strings (i.e., repeated subtrees).
- **Step 4:** *Compression based on the Patterns*. The identified repeated sub-trees may have different parent nodes. To ensure the dependencies between the sub-trees and their parent nodes are not broken, DepComm selects only the repeated sub-trees having the same parent node, and merges the selected sub-trees into one sub-tree. The attributes of each node and edge in the merged sub-tree are the unions of the attributes of the original nodes and edges.

Resource-based Patterns. A resource pattern identifies resources that are repetitively accessed by a same set of processes. To identify such patterns, DepComm first associates the processes with their accessed resources, and then search each resource to identify the repetitive accesses. Based on the found patterns, the resource nodes are merged into one node, and the attributes of the merged nodes are the union of the attributes of the original resource nodes.

E. Community Summarization

For each community, DepComm generates a summary that consists of three parts: the master process, the time span, and the top-ranked InfoPaths, as shown in Fig. 2). The master process represents the root process for the system activities in the community. The time span is computed using the earliest start time and the latest end time among all the events in the community (i.e., $c.st = min_{e_i \in c}\{e_i.st\}$ and $c.et = max_{e_i \in c}\{e_i.et\}$), which provides the timing information for tracing certain activities. InfoPaths indicate the information flows from the inputs to the outputs in the community, representing the major activities in the community.

InfoPaths Extraction. Given a process-centric community, DepComm first identifies its input and output nodes. An input node represents the incoming information flow for a community, which are the target node of an inter-community edge (e.g., `leak` and `../analysis.txt` of C3 in Fig. 1) and network nodes with outgoing edges (e.g., `xxx->xxx` of C1 in Fig. 1, representing the external IP which a community receives files from). In addition, for the communities without incoming edges (e.g., C5 in Fig. 1), we select the master process as the input node. An output node represents the outgoing information flow of a community, which are the source node of an inter-community edge (e.g., `leak` of C2 in Fig. 1), network nodes with incoming edges (e.g., `xxx->xxx` of C3 in Fig. 1, representing the external IP which a community sends files to), and POI nodes. Then, for each pair of input and output nodes, DepComm uses Depth First Search (DFS) algorithm to find a longest path without duplicate nodes as an InfoPath. Such a path generally covers more activity information than the shorter ones.

InfoPaths Prioritization. A community often contains multiple inputs and outputs, and thus has multiple InfoPaths. DepComm priorities the InfoPaths based on their likelihoods to represent major activities (e.g., attack behaviors). The priority score of an InfoPath $P_k : v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_{|P_k|-1}$ is computed based on the following four key features:

(a) *POI Event* (f_{poi}). An InfoPath that contains the POI event is directly related to the attack. Thus, f_{poi} is 1 if an InfoPath contains the POI event, and is 0 otherwise.

(b) *Input/Output Type* (f_{iot}). As processes drive the attack execution, an security analyst is more likely to find another attack stage through process nodes. For example, in Fig. 1, tracking the attack from C3 to C2 can be done through the input process node `leak` but not through the input file node `../analysis.txt`. Thus, f_{iot} gives an InfoPath whose input or output node is a process a higher priority:

$$f_{iot} = \frac{1}{2}(\delta(v_0) + \delta(v_{|P_k|-1})) \tag{1}$$

where $\delta(v_i)$ is 1 if v_i is a process and 0 otherwise.

(c) *Event Uniqueness* (f_{uni}). File events that appear in fewer communities are more likely to represent the major activities in the community, such as the event `vim write ../analysis.txt` that occurs only in C4, while file events that are frequently observed in different communities often represent irrelevant chronicle tasks running in the background. Based on this observation, we design the feature f_{uni} to measure the uniqueness of file events:

$$f_{uni} = \frac{1}{|Evt(P_k)|} \sum_{e_i \in P_k, e_i \in Evt_f} \frac{1}{|Comm(e_i)|} \tag{2}$$

where $|Evt(P_k)|$ denotes the number of file events in P_k, $e_i \in Evt_f$ denotes file events, and $|Comm(e_i)|$ denotes the number of communities in which e_i occurs. f_{uni} has a larger value when $|Comm(e_i)|$ is smaller.

(d) *Time Span* (f_{span}). Intuitively, an InfoPath whose time span is similar to the time span of the community is more likely to represent the major activities in the community. We design the feature f_{span} to model this intuition:

$$f_{span} = \frac{e(v_{|P_k|-2}, v_{|P_k|-1}).et - e(v_0, v_1).st}{c.et - c.st} \tag{3}$$

where the numerator denotes the time span of the InfoPath, and the denominator denotes the time span of the community.

TABLE III: Statistics of the attacks' dependency graphs

Attack Cases	Dep. Graph		Pre-processed Dep. Graph		Attack		$\|C\|$
	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$	
A1: Email Penetration	527	24,470	201	476	42	65	8
A2: Compile Crash	369	4,675	160	351	9	9	17
A3: Files Tamper	5,810	387,086	787	1,613	63	94	16
A4: Data Exfiltration	1,038	24,094	203	620	23	35	10
A5: Password Crack	557	10,917	43	82	39	77	4
A6: VPN Filter	22,358	275,917	518	1,424	79	130	12
D1: Phishing Email (C.S.)	7,724	2,174,649	2,545	6,483	5	6	48
D2: Phishing Email (F.D.)	2,311	1,007,062	815	18,858	12	17	20
D3: Firefox Backdoor (F.D.)	7,645	1,598,642	5,210	34,047	14	18	43
D4: Browser Extension (F.D.)	9,533	1,900,715	7,056	38,419	9	18	45
D5: Browser Extension (Theia)	3,302	37,109	172	750	12	15	11
D6: Firefox Backdoor (Theia)	3,501	37,468	205	819	13	17	13
D7: Phishing Email (Theia)	2,745	29,987	123	559	5	6	8
D8: Pine Backdoor (Trace)	2,945	133,890	192	1,247	16	23	7
Average	5,026.1	546,191.5	1,302.1	7,553.4	24.3	38.3	18.7

Based on these features, we compute the priority score of P_k by giving equal weights to each feature. According to the assigned priority scores, we sort the InfoPaths and select the top-n paths as the summary, where security analysts have the flexibility to choose the value of n.

V. Evaluation

DepComm includes ~15k lines of python code and is deployed on a server with two Intel Xeon E5-2630 v3 2.4GHz CPUs (32 processors) and 128GB memory. The evaluation dataset includes 6 attacks performed in our test environment deployed with system monitoring tools and 8 attacks in the DARPA TC (Transparent Computing) dataset [40]. In the evaluations, we aim to answer the following research questions:

- **RQ1**: What is the overall effectiveness of DepComm in summarizing dependency graphs?
- **RQ2**: How does DepComm cooperate with the automatic investigation technique HOLMES [32]?
- **RQ3**: How effective is DepComm in community detection, compared with other state-of-the-art approaches?
- **RQ4**: How effective is DepComm in community compression?
- **RQ5**: How effective is DepComm in generating community summaries using top-ranked InfoPaths?
- **RQ6**: What is the turnaround performance of DepComm in summarizing dependency graphs?

A. *Evaluation Setup*

Attack Dataset: We adopt Sysdig [69] to collect the attack dataset from 6 Linux hosts that have 10 active users. Routine system tasks on these hosts include web browsing, text editing, code development, and some other services (e.g., databases). On these hosts, we performed 6 multi-step attacks based on the known exploits [12, 13, 26, 70] and Kill Chain [71]. The collected dataset contains $\sim$ 100 million events for three days. Below are the details of these attacks:

- **A1: Penetration into Email Server.** An attacker inside a corporation inserts malicious code into a normal software and uploads this modified software to the corporation's resource server. One employee downloads this modified software and executes it, and the malicious code creates a connection to the attacker's host that allows the attackers to easily hijack emails.

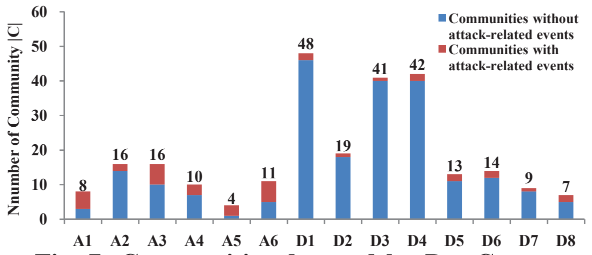


Fig. 7: Communities detected by DepComm

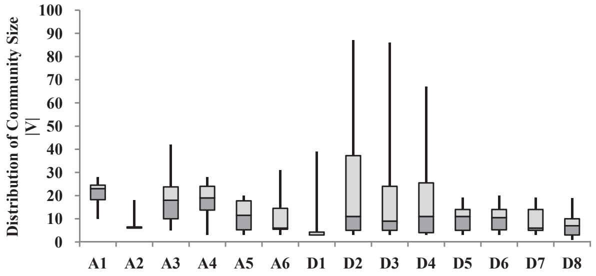


Fig. 8: Community sizes

- **A2: Crashing Compiler.** An insider attacker uploads a malicious C code to the internal resource server. When an employee downloads and compiles the source code, the malicious code causes the compiler to crash, and at last the compiler generates an incorrect executable file.
- **A3: Tampering Sensitive Files.** During a three-day period, an insider attacker logs into an employee's host using the stolen password several times, and then collects and tampers some sensitive files. Finally, the attacker sends these sensitive files back by email.
- **A4: Data Exfiltration.** An attacker penetrates into a victim host via exploiting the Shellshock vulnerability [72] to set up a backdoor, and exflitrate sensitive data by installing malware into the host. This attack is shown in Section II-A.
- **A5: Cracking Password.** After the shellshock penetration, the attacker downloads a password cracker payload from the C&C server, and then obtains the root's password by running the cracker. The attacker then penetrates to other hosts inside the same network using the root privilege obtained via the cracked passwords.
- **A6: VPN Filter** To launch a more persistent and stealthy attack, the attacker uses a more sophisticated multi-stage VPN Filter malware [73] After the shellshock penetration, the attacker downloads the Stage 1 executable from the C&C server. When triggered, the Stage 1 executable will download the Stage 2 executable, which will gather sensitive documents and establish a stealthy connection to the C&C server for data exfiltration.

DARPA TC Dataset: DARPA TC dataset [40] is an effort to develop forensics analysis and detection of Advanced Persistent Threats (APT) [74–76]. This dataset records the attack traces of various vulnerability exploits on different operating systems (e.g., Linux and Windows). Based on the attack descriptions, we exclude the failed attacks and use 8 attacks in our evaluations ($\sim$ 50 million events).

Labeling Ground Truth: We build system dependency graphs via the cross-host backward causality analysis [9] from

TABLE IV: Statistics of edges generated by DEPCOMM and NoDoze

	A1	A2	A3	A4	A5	A6	D1	D2	D3	D4	D5	D6	D7	D8	Average
Nodoze	538	157	2,403	1,641	612	198	727	3,135	5,337	7,160	2,752	2,631	2,257	1,744	2,235.14
DEPCOMM (Top-1)	74	68	86	70	27	47	77	69	122	127	44	45	36	33	66.07
DEPCOMM (Top-2)	93	82	123	91	30	56	109	94	171	180	68	65	45	40	89.07
DEPCOMM (Top-3)	115	84	147	114	32	68	121	119	202	208	76	81	55	52	105.28

the POI events. We use the attack scripts and the attack descriptions to identify the POI events and the attack-related events for the attacks performed in our test environment and DARPA TC dataset, respectively. The detailed statistics of the dependency graphs are shown in Table III. Column "Dep. Graph" shows the number of nodes and edges of original dependency graphs. Column "Pre-processed Dep. Graph" shows the number of nodes and edges of graphs after the pre-processing (i.e., merging edges and filtering read-only file). Column "Attack" shows the number of attack-relevant nodes and edges. Finally, we manually partition each dependency graph into communities. Specially, we first identify the processes that are created by the identical parent processes and are related to each other by checking whether they have data dependencies through resources. Each group of the related processes are put into a community. We then label the parent process as the master process of the community, and associate the resource nodes to the found communities according to their dependencies with the process nodes. To further obtain more objective ground truths, three independent experts are asked to verify our ground truths. These experts have Ph.D. degrees of computer sciences and have been conducting research in computer system field for more than ten years. We revise our ground truths if at least two experts consider certain nodes should belong to different communities. All these results are available at our project website [49]. Column "$|C|$" shows the number of communities that are manually partitioned.

B. *RQ1: Overall Effectiveness of* DEPCOMM

We applied DEPCOMM to generate summary graphs for the dependency graphs shown in Table III, and measured the number of the detected communities and their sizes to demonstrate the effectiveness of DEPCOMM. Fig. 7 shows the results of the detected communities. We can see that DEPCOMM partitions the dependency graphs into 18.4 communities on average. Compared with the original dependency graphs, which have $1,302.1$ nodes on average, it is 70.7 times smaller. These results indicate that with the much smaller number of communities, it is feasible to visualize all the communities for security analysts to easily see the overview of all the related system activities. We can also see that the largest number of communities is 48 for Phishing Email (C.S), which includes different system tasks (e.g., browsing web pages in Firefox, sending or receiving E-mail and calendar service).

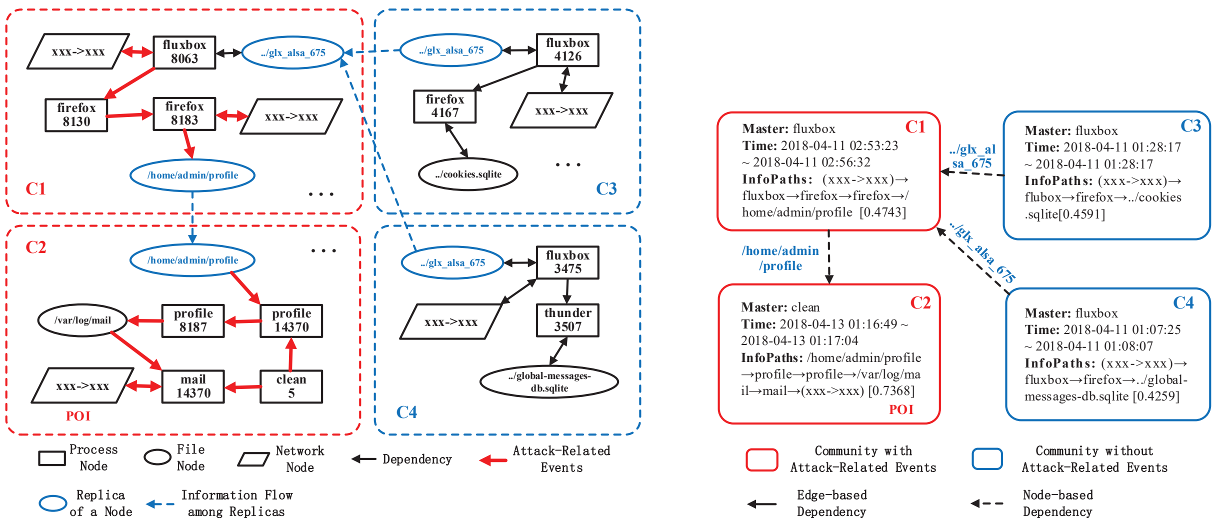
We next show the distributions of community sizes (the number of nodes in each community) for the 14 attacks in Fig. 8. As we can see, the community sizes are relatively small (15.7 nodes on average), which greatly reduce security analysts' efforts in inspecting each community. Compared with the original dependency graphs, these results also show that the community compression is quite effective in compressing the redundant edges, reducing 216.4 redundant edges on average for each community. Furthermore, the summary graphs need only 2.26MB on average to store the summary graphs, while the original dependency graphs need 344.32MB on average.

We also compare against the state-of-the-art dependency graph reduction approach, NoDoze [14]. Nodoze learns an execution profile from benign system behaviors and reduces a dependency graph based on the anomaly scores computed using the profile for each path in the dependency graph. We use the events collected when the hosts are not under attack to generate the execution profile. We compare the number of events in the top-1, top-2, and top-3 InfoPaths for all the communities with the events identified by NoDoze, as shown in Table IV. Top-3 InfoPaths of DEPCOMM have averagely $\sim 21\times$ less edges than NoDoze. NoDoze achieves poor performances since its effectiveness heavily depends on whether an execution profile can cover all the benign events and is representative, which is very difficult due to the versatility of runtime environment of most systems. Thus, the execution profile learned from one system is difficult to generalize to other systems. DEPCOMM does not suffer from the same limitations as DEPCOMM does not require extra execution profiles.

Case Study. We here illustrate how summary graphs can be used to facilitate attack investigation. Fig. 9 shows the summary graph generated by DEPCOMM for the attack D5 in the DARPA dataset. DEPCOMM partitions the dependency graph into 13 process-centric communities. Fig. 9(a) shows 4 communities (C1-C4), and Fig. 9(b) shows the corresponding summary graph, where the top-1 InfoPath is used as the community summary. C2 contains the POI event, and thus is an attack-related community. From the 8 events in C2, we can easily identify 8 attack events (red edges). These attack events represent the attack behaviors that open an backdoor to the attacker's console using `mail`. Based on the InfoPaths of C1 and C2, we can see that C1 is another attack-related community. Similarly, it is easy to identify another 7 attack-related events from 17 events in C1, which represents the attack behavior of downloading the malicious file `/home /admin/profile`. By inspecting the InfoPaths, we can further identify C3 and C4 via their dependencies on C1. However, the outputs of their InfoPaths are not the same as the input of C1's InfoPaths. Thus, C3 and C4 are not attack-related communities. In summary, we reveal the attack-related events of the attack D5 by inspecting only 25 events out of the $37,109$ events in the original graph.

C. *RQ2: Cooperation with HOLMES*

We next illustrate how DEPCOMM cooperates with one of the state-of-the-art investigation technique, HOLMES [32]. HOLMES builds a high-level scenario graph (HSG) that


xxx->xxx
fluxbox 8063
../glx_alsa_675
firefox 8130
firefox 8183
xxx->xxx
/home/admin/profile
C1
...
../glx_alsa_675
fluxbox 4126
firefox 4167
xxx->xxx
../cookies.sqlite
...
C3
C2
/home/admin/profile
...
/var/log/mail
profile 8187
profile 14370
xxx->xxx
mail 14370
clean 5
POI
../glx_alsa_675
fluxbox 3475
xxx->xxx
thunder 3507
../global-messages-db.sqlite
C4
Process Node
File Node
Network Node
Dependency
Attack-Related Events
Replica of a Node
Information Flow among Replicas
Master: fluxbox
C1
Time: 2018-04-11 02:53:23 ~ 2018-04-11 02:56:32
InfoPaths: (xxx->xxx)→ fluxbox→firefox→firefox→/ home/admin/profile [0.4743]
../glx_al sa_675
Master: fluxbox
C3
Time: 2018-04-11 01:28:17 ~ 2018-04-11 01:28:17
InfoPaths: (xxx->xxx)→ flubox→firefox→../cookies .sqlite[0.4591]
/home/admin /profile
../glx_alsa_675
Master: clean
C2
Time: 2018-04-13 01:16:49 ~ 2018-04-13 01:17:04
InfoPaths: /home/admin/profile →profile→profile→/var/log/ma il→mail→(xxx->xxx) [0.7368]
POI
Master: fluxbox
C4
Time: 2018-04-11 01:07:25 ~ 2018-04-11 01:08:07
InfoPaths: (xxx->xxx)→ fluxbox→firefox→../global- messages-db.sqlite [0.4259]
Community with Attack-Related Events
Community without Attack-Related Events
Edge-based Dependency
Node-based Dependency

(a) Process-centric communities **(b) Summary graph**

Fig. 9: Communities and summary graph for the attack D5

TABLE V: Kill Chain Steps for attack-related communities

Attack Case	Kill Chain Steps for Attack-Related Communities (AC)
A1: Email Penetration	AC1: Initial Compromise (Top-2); AC2: -; AC3: Complete Mission (Top-1); AC4: Complete Mission (Top-1); AC5: Complete Mission (Top-1)
A2: Compile Crash	AC1: Initial Compromise (Top-1); AC2: -; AC3: Complete Mission (Top-1)
A3: Files Tamper	AC1: Initial Compromise (Top-1); AC2: Internal Recon (Top-2); AC3: Internal Recon (Top-2); AC4: Internal Recon (Top-2); AC5: Internal Recon (Top-1); AC6: Complete Mission (Top-1)
A4: Data Exfiltration	AC1: Initial Compromise (Top-1); AC2: Establish Foothold (Top-1), Privilege Escalation (Top-1); AC3: Internal Recon (Top-2), Complete Mission (Top-1)
A5: Password Crack	AC1: Initial Compromise (Top-1); AC2: Establish Foothold (Top-1), Privilege Escalation (Top-1); AC3: Complete Mission (Top-1)
A6: VPN Filter	AC1: Initial Compromise (Top-1); AC2: Establish Foothold (Top-1), Privilege Escalation (Top-1); AC3: Privilege Escalation (Top-1), Internal Recon (Top-1); AC4: Initial Compromise (Top-1); AC5: Complete Mission (Top-1); AC6: Internal Recon (Top-2), Complete Mission (Top-1)
D1: Phishing Email (C.S.)	AC1: Initial Compromise (Top-1), Establish Foothold (Top-1)
D2: Phishing Email (F.D.)	AC1: Initial Compromise (Top-1), Establish Foothold (Top-1)
D3: Firefox Backdoor (F.D.)	AC1: Initial Compromise (Top-1), Establish Foothold (Top-1), Internal Recon (Top-2)
D4: Browser Extension (F.D.)	AC1: Initial Compromise (Top-1), Complete Mission (Top-1)
D5: Browser Extension (Theia)	AC1: Initial Compromise (Top-1); AC2: Establish Foothold (Top-1)
D6: Firefox Backdoor (Theia)	AC1: Initial Compromise (Top-1); AC2: Establish Foothold (Top-1)
D7: Phishing Email (Theia)	AC1: Initial Compromise (Top-1), Establish Foothold (Top-1)
D8: Pine Backdoor (Trace)	AC1: Initial Compromise (Top-1); AC2: Establish Foothold (Top-1)

integrates the TTP (Tactics, Techniques, and Procedures) [77], an important indicator for describing the steps of Advanced Persistent Threats (APT), and uses the HSG to map the low-level event information flows to the steps in the Kill Chain [48]. In this evaluation, we first build the HSGs for the 14 attack cases, and then use the HSG to map the top-ranked InfoPaths to the steps in the Kill Chain.

Table V shows the mapping results for attack-related communities. We can observe that the Top-2 InfoPaths are sufficient to find the Kill Chain. We also manually inspect these InfoPaths to create the mappings, and confirm that most of the mappings found by the HSGs agree with our manual mappings. In total, HOLMES identifies 35 out of the 37 attack-related communities, achieving a recall of 96.2%. HOLMES fails to map two attack-related communities (AC2 of A1 and AC2 of A2) to the Kill Chain. The attack behavior in AC2 of A1 is to download a modified software application to the host of a victimized employee from the corporation's resource web and executing the software application, where the resource server has a trusted IP and the modified software application has a trusted file name. Therefore, the activities in AC2 cannot be captured by the TTP rule of HOLMES. Nevertheless, while HOLMES's rules fail to map AC2, the InfoPaths computed by DEPCOMM can be used to complement HOLMES's rules. In fact, AC2 has an InfoPath from the attack-related community AC1, which represents the attack behavior of uploading the modified software application to the resource web, and has another InfoPath that leads to another attack-related community AC3, which represents the attack behavior of tampering the system files `/etc/mail.rc` for email server. Thus, the activities in AC2 form an indispensable step of the attack A1, and can be easily inferred as an attack-related community after inspecting these InfoPaths. HOLEMS fails to map the attack behaivor in AC2 of A2 for the similar reason. AC2 of A2 describe the actitivies that a compiler `cc1` reads an anomaly file that is created by AC1 and generates an incorrect binary file that is the input node of AC3. Unfortunately anomaly file access activities are not included in the TTP rules of HOLMES, and thus AC2 cannot be mapped. Similarly, AC2 has two InfoPaths that connect to two attack-related communities, AC1 and AC3, which makes the activities in AC2 an indispensable step of the attack A2 as well. These results show that DEPCOMM can easily cooperate with other automatic techniques to highlight the attack-related communities and help security analysts to recognize residual attack-related communities missed by the automatic techniques.

TABLE VI: Results of community detection for 14 attack cases

Attack Cases	NISE 2016		EgoSpliter 2017		NMNF 2017		DANMF 2018		PMCV 2019		CGAN 2019		VGRAPH 2019		CNRL 2019		DeepWalk 2014		DEPCOMM	
	F_1	$\|C\|$	F_1	$\|C\|$	F_1	$\|C\|$	F_1	$\|C\|$	F_1	$\|C\|$	F_1	$\|C\|$	F_1	$\|C\|$	F_1	$\|C\|$	F_1	$\|C\|$	F_1	$\|C\|$
A1: Email Penetration	0.459	5	0.327	6	0.553	5	0.696	6	0.509	5	0.302	8	0.354	8	0.521	5	0.674	5	**0.928**	**8**
A2: Compile Crash	0.492	9	0.273	9	0.376	7	0.320	8	0.274	10	0.219	17	0.343	17	0.730	8	0.413	7	**0.952**	**16**
A3: Files Tamper	0.301	7	0.242	7	0.455	9	0.522	11	0.359	8	0.174	16	0.204	16	0.672	12	0.645	11	**0.958**	**16**
A4: Data Exfiltration	0.426	6	0.516	8	0.509	9	0.684	7	0.327	5	0.329	10	0.300	10	0.642	7	0.647	7	**0.937**	**10**
A5: Password Crack	0.666	3	0.622	3	0.756	3	0.847	3	0.711	4	0.505	4	0.487	4	0.910	4	0.988	4	**1.0**	**4**
A6: VPN Filter	0.629	7	0.705	7	0.675	6	0.604	6	0.343	5	0.305	12	0.315	12	0.686	7	0.650	7	**0.915**	**11**
D1: Phishing Email (C.S.)	0.164	6	0.244	10	0.109	7	0.251	10	0.0	1	0.113	48	0.452	48	0.234	9	0.384	10	**0.944**	**48**
D2: Phishing Email (F.D.)	0.235	6	0.168	5	0.206	6	0.314	8	0.187	6	0.136	20	0.161	20	0.353	7	0.266	6	**0.955**	**19**
D3: Firefox Backdoor (F.D.)	0.357	7	0.203	8	0.293	8	0.399	9	0.227	6	0.175	43	0.128	43	0.316	9	0.391	9	**0.930**	**41**
D4: Browser Extension (F.D.)	0.242	6	0.212	7	0.213	9	0.298	9	0.297	8	0.091	45	0.139	45	0.447	12	0.403	11	**0.923**	**42**
D5: Browser Extension (Theia)	0.464	6	0.356	5	0.453	7	0.406	8	0.413	7	0.179	11	0.221	11	0.600	8	0.508	8	**0.911**	**13**
D6: Firefox Backdoor (Theia)	0.396	6	0.321	5	0.393	6	0.485	8	0.388	7	0.194	13	0.234	13	0.608	8	0.529	9	**0.887**	**14**
D7: Phishing Email (Theia)	0.307	5	0.458	6	0.505	5	0.507	6	0.286	4	0.231	8	0.283	8	0.694	6	0.698	6	**0.966**	**9**
D8: Pine Backdoor (Trace)	0.537	4	0.623	5	0.449	4	0.481	4	0.318	3	0.340	7	0.418	7	0.773	5	0.758	5	**0.971**	**7**

D. RQ3: Comparison of Community Detection

We compare DEPCOMM with other state-of-the-art community detection algorithms to show the effectiveness of DEPCOMM's community detection technique. Considering the overlapping nature of dependency graphs, we select 9 typical overlapping community detection algorithms as the baselines, including NISE (2016) [36], EgoSpliter (2017) [41], NMNF (2017) [42], DANMF (2018) [43], PMCV (2019) [44], CGAN (2019) [45], VGRAPH (2019) [46], CNRL (2019) [47] and DeepWalk (2014) [38]. We use F_1-score [78] to evaluate the overall correspondence between the detected communities and the ground-truth communities labeled by us.

Table VI shows the F_1-score and the number of detected communities $|C|$ for DEPCOMM and the baselines. The results show that F_1-score achieved by DEPCOMM is averagely 2.29 times higher than those achieved by the baselines. This shows that our community detection algorithm is effective to detect the process-centric communities, while the other baselines have poorer performance due to the following reasons: (1) they mainly focus on homogeneous graphs, and are oblivious to the types of system events. Thus, they cannot effectively distinguish process nodes and resource nodes, and mix these nodes in a community, causing a community to contain multiple irrelevant system activities or spread a system activity across multiple communities; (2) they depend on a common assumption that edges inside a community are more than the edges linking with the nodes of other communities. Thus, they fail to split two master process nodes connected with information flows into two communities, even though the two processes represent distinct system activities.

Furthermore, even though DEPCOMM and DeepWalk both use SkipGram to learn the node representation from the walk routes, DEPCOMM outperforms DeepWalk by $1.65\times$ on average. This shows that DEPCOMM's hierarchical walk schemes are more effective than the random walk scheme adopted by DeepWalk, which treats each node equally.

E. RQ4: Effectiveness of Community Compression

To evaluate the effectiveness of community compression, we compute the compression rates as $\gamma = 1 - Size_{post}/Size_{pre}$, where $Size_{pre}$ denotes the number of nodes or edges of a community before applying compression and $Size_{post}$ is the number of nodes or edges after using compression. The box plots in Fig. 10 and Fig. 11 show the distributions of the compression rates for the nodes and the edges, respectively. We can see that for a community, the number of nodes and the number of edges are reduced averagely by 38.4% and 44.7%, respectively, with the maximum reduction being 97.3% for the nodes and 98.9% for the edges. In addition, we verify that the InfoPaths are not changed after compression. The reason is that the repeated activities have a same information flow that often enters the subgraph formed by the repeated activities through a single node and leave the subgraph via another single node, and thus compressing the repeated activities will not change the events inside InfoPaths. In a word, compressing these repeated activities still preserves the semantics for the task represented by a community.

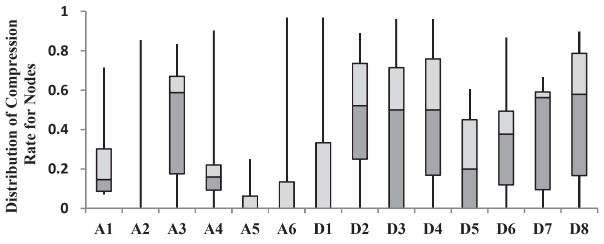


Fig. 10: Community compression rate for nodes

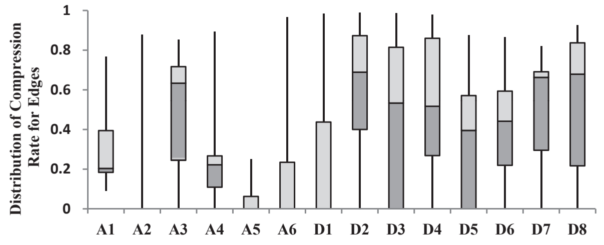


Fig. 11: Community compression rate for edges

F. RQ5: Effectiveness of InfoPath Ranking

For each community, DEPCOMM extracts InfoPaths based on its input and output nodes. On average, a community has 4.3 input nodes and 3.9 output nodes, forming 15.7 InfoPaths. We manually inspect the top-3 InfoPaths for each community and confirm that the top-2 InfoPaths are sufficient to represent system activities and attack behaviors. That is, we only need to inspect 12.7% of the extracted InfoPaths.

TABLE VII: Top 3 InfoPaths of the community C3 with attack-related events and C8 without attack-related events

	InfoPaths	Priority Score
C3	Top-1: `leak→tar→../upload.tar→bzip2→../upload.tar.bz2→gpg→../upload→cur→xxx->xxx`	0.8234
	Top-2: `../analysis.txt→tar→../upload.tar→bzip2→../upload.tar.bz2→gpg→../upload→cur→xxx->xxx`	0.7141
	Top-3: `../userlist→tar→../upload.tar→bzip2→../upload.tar.bz2→gpg→../upload→cur→xxx->xxx`	0.7137
C8	Top-1: `xxx->xxx→sshd→bash→scp→../statistics.tar.bz2→tar→../userlist`	0.4914
	Top-2: `xxx->xxx→scp→../statistics.tar.bz2→tar→../userlist`	0.3839
	Top-3: `/dev/null→bash→scp→../statistics.tar.bz2→tar→../userlist`	0.1830

We next use two communities to illustrate the effectiveness of the top-ranked InfoPaths. Table VII shows the top 3 InfoPaths of an attack-related community C3 that contains attack behaviors and a community without attack-related events C8 for the attack A4. The events in C3 show that an attacker runs a malicious script to compress, encrypt, and upload the sensitive files to a remote server. We can see that these attack behaviors can be effectively represented by using the top-1 InfoPath whose priority score is 0.8234. While the top-2 and the top-3 can also cover the behaviors, the input node of the top-1 InfoPath is a malicious script process (i.e., `leak`) and it is easier to help security analysis further trace the community that creates the malicious script. The events in C8 show that a user logs into a host using sshd, transfers a compressed file from a server to the host, and decompresses the file. We can see that the top-1 InfoPath with the highest priority score (0.4914) can represent all these activities, while the top-2 InfoPath lacks the events for sshd login, and the top-3 InfoPath lacks the sshd login event and contains a file event (`/dev/null→bash`) that appears in many communities.

G. RQ6: Turnaround Time Performance of DepComm

To understand the turnaround time performance of DepComm, we measure the turnaround time of each phase in DepComm for the 14 attack cases. As the hierarchical walks and vectorization in the community detection phase are independent to each other, it is feasible to parallelize all the hierarchical walks and vectorization. We use multi-processes (20 processes) in a host to realize the parallelization. The results are shown in Table VIII. On average, DepComm takes $1,148.90$s to generate a summary graph, which is $\sim 6\times$ faster than running in a single process. More specifically, dependency graph construction uses 32.12s, dependency graph pre-processing uses 256.72s, and community detection uses 858.48s. For community detection, hierarchical walks uses 581.28s and vectorization uses 269.26s, which are $\sim 7\times$ and $\sim 4\times$ faster than running in a single process. Finally, community compression uses 1.41s and community summarization uses 0.17s. We can observe that (1) the community detection phase takes up most of the time due to the walking sampling and representation learning, and they can be accelerated by parallelization; (2) DepComm takes less time to compress process-centric communities due to the highly efficient frequent pattern mining algorithm; (3) the community summarization phase requires the lest time because of the small community sizes after compression. In a word, the turnaround time performance of DepComm can be further improved by parallelizing the hierarchical walks and vectorization.

VI. Discussion

Cooperation with Other Investigation Techniques. Besides highlighting attack-related communities, visualization techniques can be applied on the summary graphs generated by DepComm to show the overview of system activities, and provide on-demand zoom in (zoom out) functionality to show (hide) the detailed events in the communities. Additionally, by integrating with other causality analysis techniques [12, 14, 28], DepComm can generate a heat map that highlights the communities that are likely to contain suspicious behaviors.

Forensics of Real-World Attacks. Recent real-world attacks, such as Advanced Persistent Threat (APT) [75, 76], are sophisticated (multi-step attacks that exploit various vulnerabilities) and stealthy (staying dormant for a long period). With the advances of log compression techniques [26, 27, 33, 60] and the continuing decreases of storage costs, it is affordable to store system audit logs for months or even years. Furthermore, recent distributed database solutions [18, 19, 79, 80] show promising results to improve the search performance of the logs, which can be used to generate dependency graphs for massive amount of logs. By working together with these solutions, DepComm can be applied on the generated dependency graphs to detect communities, and integrate with other detection techniques [14, 32] to highlight attack-related communities.

Analysis Turnaround Time. Our current implementation of DepComm takes averagely $1,148.90$s (in Table VIII) to generate a summary graph. As the hierarchical walks and vectorization are independent to each other, it is feasible to parallelize all the hierarchical walks and vectorization [63]. By working with intrusion detection systems [11, 23] that can provide real-time alerts and defenses, DepComm can be applied to identify the attack entry points and impacts, enabling quicker turnaround time for system recovery and preventing future compromises.

Limitations of DepComm. Hierarchical graph embedding is a novel graph embedding technology for system dependency graphs. However, there are still some hyper-parameters (e.g., the walking length, the window size for sub-sequence extraction, and the dimension of vectors) that need to be set manually. We adjust them based on the existing sensitivity analysis [38], where the walking length is set to 200, the window size is 20, and the dimensions is 20. There are still some less-important events that cannot be compressed by DepComm, such as some interactions with system files (e.g., `bash→/dev/null`). By inspecting the communities, these events can be found in different communities, and thus mining discriminative patterns [81, 82] may help identify such patterns.

TABLE VIII: Turnaround Time of DEPCOMM

Attack Cases	Dep. Graph Construction(s)	Dep. Graph Preprocess(s)	Community Detection(s)	Community Compression(s)	Community Summarization(s)	Total(s)
A1: Email Penetration	1.15	7.14	186.72	0.032	0.054	195.09
A2: Compile Crash	0.23	1.4	78.38	0.0096	0.089	80.11
A3: Files Tamper	30.64	147.13	740.28	0.029	0.13	918.21
A4: Data Exfiltration	1.11	6.55	172.24	0.027	0.074	180.00
A5: Password Crack	0.80	3.19	20.54	0.0075	0.0085	24.54
A6: VPN Filter	13.42	111.9	1,463.45	0.049	0.17	1,588.99
D1: Phishing Email (C.S.)	141.63	1,028.32	110.77	0.0052	0.060	1,280.78
D2: Phishing Email (F.D.)	60.72	486.57	2,247.12	4.53	0.31	2,799.25
D3: Firefox Backdoor (F.D.)	83.38	780.93	2,968.01	7.36	0.47	3,840.15
D4: Browser Extension (F.D.)	93.06	953.06	2,846.95	7.70	0.41	3,901.18
D5: Browser Extension (Theia)	5.87	10.21	135.69	0.017	0.16	151.95
D6: Firefox Backdoor (Theia)	5.76	10.4	159.11	0.025	0.16	175.45
D7: Phishing Email (Theia)	5.86	8.27	105.28	0.011	0.11	119.53
D8: Pine Backdoor (Trace)	6.06	38.97	784.18	0.022	0.17	829.40
Average	32.12	256.72	858.48	1.42	0.17	1,148.90

VII. RELATED WORK

Causality Analysis via System Audit Logs. Causality analysis was initially proposed by King et al. [8, 9], which aims to automatically reconstruct a series of events that represent attack steps. As causality analysis suffers from the dependency explosion problem [26, 27, 33, 51], recent research has proposed techniques to perform fine-grained causality analysis [13, 15, 25, 83, 84] and prioritize dependencies [12, 14]. Also, Gui et al. [85] proposed an approach that presents updates of causality analysis periodically and involves human in the loop to provide heuristics in reducing the generated dependency graphs. Unlike these techniques that aim to reveal attack-related events, DEPCOMM generates a summary graph from the dependency graph, and can work with these techniques to highlight the attack-related communities.

Behavior Analysis via System Audit Logs. Gao et al. [18, 19] proposed domain-specific languages that query system audit logs for efficient attack investigation. Milajerdi et al. [32] proposed to rely on the correlation of suspicious information flows to detect ongoing attack campaigns, and used the knowledge from cyber threat intelligence (CTI) reports to align the attack behaviors recorded in system audit logs [86]. Pasquier et al. [87] proposed a runtime analysis of provenance by combining runtime kernel-layer reference monitor with a query module. Hossain et al. [28] proposed a tag-based technique to perform real-time attack detection and reconstruction from system audit logs. The summary graphs generated by DEPCOMM can be integrated with these techniques to facilitate the understanding of attack behaviors and provide better defenses. Furthermore, recent approaches [14, 88] leverage alerts from threat detection systems or software applications' runtime logging activities [89] to generate compact graphs. Unlike these approach whose quality heavily depend on the detected alerts and the generalizability of the learned system profiles, DEPCOMM is a general approach that leverages only the information inside dependency graphs to detect communities, and can easily cooperate with various automatic investigation techniques [32] to detect attack-related communities.

Community Detection. NISE [36] is a local-expansion algorithm, which expands a initialized seed set into clusters with overlaps. EgoSplitter [41] first build a node-decoupling graph through splitting nodes into multiple replicas, then applies some classic methods for disjoint community detection to the build graph. NMNF [42] uses non-negative matrix factorization to learn node representation with mesoscopic community structure. DANMF [43] proposed a novel deep NMF model for overlapping community detection, which models the non-negative matrix factorization process by autoencoder network. PMCV [44] detects overlapping communities through searching and joining adjacent k-cliques sharing k-1 nodes. CGAN [45] uses the Generative Adversarial Nets to learn the membership strength of nodes to communities. VGRAPH [46] uses neural networks to model the generation of node neighbors, which joins community detection and node representation learning. CNRL [47] applies the Latent Dirichlet Allocation model (LDA) to the random walk sequences to learn the community membership. DeepWalk [38] joins random walk schemes and word2vec to learn node representation with community structure. These existing algorithms mainly focus on homogeneous graphs and treat each node equally, while DEPCOMM gives priorities for neighbor nodes that are more likely to represent intimate processes.

Graph Summarization. Graph summarization produces a compact representation of a large-scale graph, facilitating the identification of structure and meaning in data [29–31]. It has extensive applications, such as clustering, classification, community detection, and outlier detection. Unlike these techniques whose target data is mainly stored in databases, DEPCOMM processes dependency graphs, a type of heterogeneous graphs where process nodes and other resource nodes represent different steps of system activities.

VIII. CONCLUSION

We have presented DEPCOMM, which clusters intimate processes that cooperate with each other to accomplish certain system tasks into a community and compresses the repeated events inside each community. For each community, DEPCOMM further identifies InfoPaths that represent the information flows across communities, and ranks these InfoPaths based on their likelihoods to reveal attack behaviors. The top-ranked InfoPaths are then used as the summary for each community. Our evaluations on real attacks demonstrate the effectiveness of DEPCOMM in detecting process-centric communities, compressing repeated events, and prioritizing InfoPaths to assist attack investigation.

Acknowledgments

We would like to thank the anonymous reviewers for the constructive comments and suggestions to improve the manuscript. Zhiqiang Xu, Yu Wen, and Dan Meng are partially supported by the Strategic Priority Research Program of Chinese Academy of Sciences, Grant No.XDC02010300. Pengcheng Fang, Changlin Liu, and Xusheng Xiao are partially supported by the National Science Foundation under the grants CNS-2028748 and CCF-2046953.

References

[1] Ebay, “Ebay Inc. to ask Ebay users to change passwords,” 2014, http://blog.ebay.com/ebay-inc-ask-ebay-users-change-passwords/.

[2] CNN, “OPM government data breach impacted 21.5 million,” 2015, http://www.cnn.com/2015/07/09/politics/office-of-personnel-management-data-breach-20-million.

[3] New York Times, “Target data breach incident,” 2014, http://www.nytimes.com/2014/02/27/business/target-reports-on-fourth-quarter-earnings.html?_r=1.

[4] NPR, “Home Depot Confirms Data Breach At U.S., Canadian Stores,” 2014, http://www.npr.org/2014/09/09/347007380/home-depot-confirms-data-breach-at-u-s-canadian-stores.

[5] Techcrunch, “Yahoo discloses hack of 1 billion accounts,” 2016, https://techcrunch.com/2016/12/14/yahoo-discloses-hack-of-1-billion-accounts/.

[6] Federal Trade Commission, “The equifax data breach,” 2017, https://www.ftc.gov/equifax-data-breach.

[7] Federal Trade Commission, “The marriott data breach,” 2018, https://www.consumer.ftc.gov/blog/2018/12/ marriott-data-breach.

[8] S. T. King and P. M. Chen, “Backtracking intrusions,” in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*. ACM, 2003, pp. 223–236.

[9] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen, “Enriching intrusion alerts through multi-host causality,” in *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*. ISOC, 2005.

[10] A. Goel, K. Po, K. Farhadi, Z. Li, and E. de Lara, “The taser intrusion recovery system,” in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*. ACM, 2005, pp. 163–176.

[11] T. Kim, X. Wang, N. Zeldovich, and M. F. Kaashoek, “Intrusion recovery using selective re-execution,” in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 2010, pp. 89–104.

[12] Y. S. Liu, M. Zhang, D. Li, K. Jee, Z. C. Li, Z. Y. Wu, J. Rhee, and P. Mittal, “Towards a timely causality analysis for enterprise security,” in *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*. ISOC, 2018.

[13] Y. Kwon, F. Wang, W. H. Wang, K. H. Lee, W. C. Lee, S. Q. Ma, X. Y. Zhang, D. Y. Xu, S. Jha, G. F. Ciocarlie, A. Gehani, and V. Yegneswaran, “MCI : Modeling-based causality inference in audit logging for attack investigation,” in *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*. ISOC, 2018.

[14] W. U. Hassan, S. J. Guo, D. Li, Z. Z. Chen, K. K. Jee, Z. C. Li, and A. Bates, “Nodoze: Combating threat alert fatigue with automated provenance triage,” in *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*. ISOC, 2019.

[15] S. Q. Ma, X. Y. Zhang, and D. Y. Xu, “Protracer: towards practical provenance tracing by alternating between logging and tainting,” in *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*. ISOC, 2016.

[16] X. Jiang, A. Walters, D. Xu, E. H. Spafford, F. Buchholz, and Y.-M. Wang, “Provenance-aware tracing of worm break-in and contaminations: A process coloring approach,” in *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2006, pp. 38–38.

[17] S. Q. Ma, J. Zhai, Y. Kwon, K. H. Lee, X. Y. Zhang, G. F. Ciocarlie, A. Gehani, V. Yegneswaran, D. Y. Xu, and S. Jha, “Kernel-supported cost-effective audit logging for causality tracking,” in *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*. USENIX Association, 2018, pp. 241–254.

[18] P. Gao, X. S. Xiao, Z. C. Li, F. Y. Xu, S. R. Kulkarni, and P. Mittal, “AIQL: Enabling efficient attack investigation from system monitoring data,” in *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*. USENIX Association, 2018, pp. 113–125.

[19] P. Gao, X. S. Xiao, D. Li, Z. C. Li, K. K. Jee, Z. Y. Wu, C. H. Kim, S. R. Kulkarni, and P. Mittal, “SAQL: A stream-based query system for real-time abnormal system behavior detection,” in *Proceedings of the USENIX Security Symposium (USENIX Security)*. USENIX Association, 2018, pp. 639–656.

[20] G. P. Spathoulas and S. K. Katsikas, “Reducing false positives in intrusion detection systems,” *Computers & Security*, vol. 29, no. 1, pp. 35–44, 2010.

[21] T. Pietraszek, “Using adaptive alert classification to reduce false positives in intrusion detection,” in *Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*. Springer, 2004, pp. 102–124.

[22] W. U. Hassan, M. Lemay, N. Aguse, A. Bates, and T. Moyer, “Towards scalable cluster auditing through grammatical inference over provenance graphs,” in *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*. ISOC, 2018.

[23] C. Kruegel, F. Valeur, and G. Vigna, *Intrusion Detection and Correlation - Challenges and Solutions*, ser. Advances in Information Security. Springer, 2005, vol. 14.

[24] A. Kharraz, S. Arshad, C. Mulliner, W. K. Robertson, and E. Kirda, “UNVEIL: A large-scale, automated approach to detecting ransomware,” in *Proceedings of the USENIX Security Symposium (USENIX Security)*, T. Holz and S. Savage, Eds. USENIX Association, 2016, pp. 757–772.

[25] K. H. Lee, X. Y. Zhang, and D. Y. Xu, “High accuracy attack provenance via binary-based execution partition,” in *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*. ISOC, 2013.

[26] Z. Xu, Z. Y. Wu, Z. C. Li, K. K. Jee, J. Rhee, X. S. Xiao, F. Y. Xu, H. N. Wang, and G. F. Jiang, “High fidelity data reduction for big data security dependency analyses,” in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2016, pp. 504–516.

[27] Y. T. Tang, D. Li, Z. C. Li, M. Zhang, K. Jee, X. S. Xiao, Z. Y. Wu, J. Rhee, F. Y. Xu, and Q. Li, “Nodemerge: Template based efficient data reduction for big-data causality analysis,” in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2018, pp. 1324–1337.

[28] M. N. Hossain, S. M. Milajerdi, J. A. Wang, B. Eshete,

R. Gjomemo, R. Sekar, S. D. Stoller, and V. N. Venkatakrishnan, "SLEUTH: real-time attack scenario reconstruction from COTS audit data," in *Proceedings of the USENIX Security Symposium (USENIX Security)*. USENIX Association, 2017, pp. 487–504.

[29] Y. K. Liu, T. Safavi, A. Dighe, and D. Koutra, "Graph summarization methods and applications: A survey," *ACM Computing Surveys*, vol. 51, no. 3, pp. 1–34, 2018.

[30] K. A. Kumar and P. Efstathopoulos, "Utility-driven graph summarization," *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, vol. 12, no. 4, pp. 335–347, 2018.

[31] J. Ko, Y. Kook, and K. Shin, "Incremental lossless graph summarization," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2020, pp. 317–327.

[32] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "HOLMES: real-time APT detection through correlation of suspicious information flows," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2019, pp. 1137–1152.

[33] M. N. Hossain, J. A. Wang, R. Sekar, and S. D. Stoller, "Dependence-preserving data compaction for scalable forensic analysis," in *Proceedings of the USENIX Security Symposium (USENIX Security)*. USENIX Association, 2018, pp. 1723–1740.

[34] F. Spitzer, *Principles of random walk*. Springer Science & Business Media, 2013, vol. 34.

[35] J. R. Xie, B. K. Szymanski, and X. M. Liu, "Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process," in *IEEE International Conference on Data Mining Workshops (ICDM)*. IEEE, 2011, pp. 344–349.

[36] J. J. Whang, D. F. Gleich, and I. S. Dhillon, "Overlapping community detection using neighborhood-inflated seed expansion," *IEEE Transactions on Knowledge & Data Engineering (TKDE)*, vol. 28, no. 5, pp. 1272–1284, 2016.

[37] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2016, pp. 855–864.

[38] B. Perozzi, R. Al Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2014, pp. 701–710.

[39] Symantec, "Process lineage," 2020, https://help.symantec.com/cs/ATP_SAAS/ATP/v128100935_-v128439210/Process-lineage-example?locale=EN_US.

[40] DARPA, "Transparent computing engagement 3 data release," https://github.com/darpa-i2o/Transparent-Computing.

[41] A. Epasto, S. Lattanzi, and R. Paes Leme, "Ego-splitting framework: From non-overlapping to overlapping clusters," in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2017, pp. 145–154.

[42] X. Wang, P. Cui, J. Wang, J. Pei, and S. Q. Yang, "Community preserving network embedding," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. AAAI, 2017, pp. 203–209.

[43] F. Ye, C. Chen, and Z. Zheng, "Deep autoencoder-like nonnegative matrix factorization for community detection," in *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*. ACM, 2018, pp. 1393–1402.

[44] N. Kasoro, S. Kasereka, E. Mayogha, H. T. Vinh, and J. Kinganga, "Percomcv: A hybrid approach of community detection in social networks," in *The International Conference on Ambient Systems, Networks and Technologies (ANT)*. ELSEVIER, 2019, pp. 45–52.

[45] Y. T. Jia, Q. Q. Zhang, W. N. Zhang, and X. P. Wang, "Communitygan: Community detection with generative adversarial nets," in *In Proceedings of the 2019 World Wide Web Conference (WWW)*. ACM, 2019, pp. 784–794.

[46] F. Y. Sun, M. Qu, H. Jordan, C. W. Huang, and J. Tang, "vgraph: A generative model for joint community detection and node representation learning," in *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*. MIT Press, 2019.

[47] C. C. Tu, X. K. Zeng, H. Wang, Z. Y. Zhang, Z. Y. Liu, M. S. Sun, B. Zhang, and L. Y. Lin, "A unified framework for community detection and network representation learning," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 31, no. 6, pp. 1051–1065, 2019.

[48] "Cyber Kill Chain," http://www.lockheedmartin.com/us/ what-we-do/information-technology/cybersecurity/ tradecraft/cyber-kill-chain.html.

[49] Anonymous, "Project website for depcomm," 2021, https://github.com/ieeesp2021sub/depcomm.

[50] A. M. Bates, D. Tian, K. R. B. Butler, and T. Moyer, "Trustworthy whole-system provenance for the linux kernel," in *Proceedings of the USENIX Security Symposium (USENIX Security)*. USENIX Association, 2015, pp. 319–334.

[51] K. H. Lee, X. Y. Zhang, and D. Xu, "Loggc: garbage collecting audit log," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2013, pp. 1005–1016.

[52] Microsoft, "ETW events in the common language runtime," 2017, https://msdn.microsoft.com/en-us/library/ff357719(v=vs.110).aspx.

[53] Redhat, "The linux audit framework," 2017, https://github.com/linux-audit.

[54] S. A. Crosby and D. S. Wallach, "Efficient data structures for tamper-evident logging," in *Proceedings of the USENIX Security Symposium (USENIX Security)*. USENIX Association, 2009, pp. 317–334.

[55] R. Paccagnella, P. Datta, W. U. Hassan, A. Bates, C. Fletcher, A. Miller, and D. Tian, "Custos: Practical tamper-evident auditing of operating systems using trusted execution," in *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*. ISOC, 2020.

[56] R. Paccagnella, K. Liao, D. Tian, and A. Bates, "Logging to the danger zone: Race condition attacks and defenses on system audit frameworks," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2020, pp. 1551–1574.

[57] F. Schuster, T. Tendyck, C. Liebchen, L. Davi, A. R. Sadeghi, and T. Holz, "Counterfeit object-oriented programming: On the difficulty of preventing code reuse attacks in c++ applications," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2015, pp. 745–762.

[58] K. J. Lu, W. K. Lee, S. Nürnberger, and M. Backes, "How to make aslr win the clone wars: Runtime re-randomization." in *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*. ISOC, 2016.

[59] K. J. Lu, C. Y. Song, T. Kim, and W. K. Lee, "Unisan: Proactive kernel memory initialization to eliminate data leakages," in

Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS). ACM, 2016, pp. 920–932.

[60] N. Michael, J. Mink, J. Liu, S. Gaur, W. U. Hassan, and A. Bates, “On the forensic validity of approximated audit logs,” in *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*. IEEE, 2020, pp. 189–202.

[61] F. C. Liu, Y. Wen, D. X. Zhang, X. H. Jiang, X. Y. Xing, and D. Meng, “Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2019, pp. 1777–1794.

[62] T. Mikolov, K. Chen, and J. Dean, “Efficient estimation of word representations in vector space,” in *arXiv:1301.3781*, 2013.

[63] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*. MIT Press, 2013, pp. 3111–3119.

[64] J. C. Dunn, “A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters,” *Cybernet*, vol. 3, no. 3, pp. 32–57, 1973.

[65] S. H. Zhang, R. S. Wang, and X. Zhang, “Identification of overlapping community structure in complex networks using fuzzy-means clustering,” *Physica A: Statistical Mechanics and its Applications*, vol. 374, no. 1, pp. 483–490, 2007.

[66] E. Trauwaert, “On the meaning of dunn’s partition coefficient for fuzzy clusters,” *Fuzzy Sets and Systems*, vol. 25, no. 2, pp. 217–242, 1988.

[67] F. Luccio, L. Pagli, A. M. Enriquez, and P. O. Rieumont, “Bottom-up subtree isomorphism for unordered labeled trees,” *International Journal of Pure and Applied Mathematics*, vol. 38, no. 3, pp. 325–343, 2007.

[68] Y. Chi, R. R. Muntz, S. Nijssen, and J. N. Kok, “Frequent subtree mining — an overview,” *Fundamenta Informaticae*, vol. 66, no. 1–2, pp. 161–198, 2004.

[69] Sysdig, “Sysdig,” 2017, https://sysdig.com/.

[70] Exploit Database, “Exploit Database,” 2017, https://www.exploit-db.com/.

[71] L. Martin, “cyber kill chain,” https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html.

[72] “CVE-2014-6271: bash: specially-crafted environment variables can be used to inject shell commands.” 2014, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271.

[73] “VPNFilter: New Router Malware with Destructive Capabilities,” 2018, https://symc.ly/2IPGGVE.

[74] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, “Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains,” *Leading Issues in Information Warfare & Security Research*, vol. 1, p. 80, 2011.

[75] Fireeye, “Anatomy of Advanced Persistent Threats,” 2017, https://www.fireeye.com/current-threats/anatomy-of-a-cyber-attack.html.

[76] Symantec, “Advanced Persistent Threats: How They Work,” 2017, https://www.symantec.com/theme.jsp? themeid=apt-infographic-1.

[77] “Adversarial tactics, techniques and common knowledge,” https://attack.mitre.org/wiki/Main Page.

[78] G. Rossetti, L. Pappalardo, and S. Rinzivillo, “A novel approach to evaluate community detection algorithms on ground truth,” in *Proceedings of the Workshop on Complex Networks (CompleNet)*. Springer, 2016, pp. 133–144.

[79] P. Gao, X. Xiao, D. Li, K. Jee, H. Chen, S. R. Kulkarni, and P. Mittal, “Querying streaming system monitoring data for enterprise system anomaly detection,” in *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 1774–1777.

[80] P. Gao, X. S. Xiao, Z. C. Li, K. Jee, F. Y. Xu, S. R. Kulkarni, and P. Mittal, “A query system for efficiently investigating complex attack behaviors for enterprise security,” *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, vol. 12, no. 12, pp. 1802–1805, 2019.

[81] Z. Bo, X. S. Xiao, Z. C. Li, Z. Y. Wu, Z. Y. Qian, X. F. Yan, A. K. Singh, and G. F. Jiang, “Behavior query discovery in system-generated temporal graphs,” *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, vol. 9, no. 4, pp. 240–251, 2015.

[82] X. F. Yan, P. S. Yu, and J. W. Han, “Graph indexing based on discriminative frequent structure analysis,” *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 4, pp. 960–993, 2005.

[83] Y. Ji, S. H. Lee, E. Downing, W. R. Wang, M. Fazzini, T. Kim, A. Orso, and W. K. Lee, “Rain: Refinable attack investigation with on-demand inter-process information flow tracking,” in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2017, pp. 377–390.

[84] Y. Ji, S. H. Lee, M. Fazzini, J. Allen, E. Downing, T. Kim, A. Orso, and W. K. Lee, “Enabling refinable cross-host attack investigation with efficient data flow tagging and tracking,” in *Proceedings of the USENIX Security Symposium (USENIX Security)*. USENIX Association, 2018, pp. 1705–1722.

[85] J. P. Gui, D. Li, Z. Z. Chen, J. Rhee, X. S. Xiao, M. Zhang, K. Jee, Z. C. Li, and H. F. Chen, “APTrace: A responsive system for agile enterprise level causality analysis,” in *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 1701–1712.

[86] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, “Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting,” in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2019, pp. 1795–1812.

[87] T. Pasquier, X. Y. Han, T. Moyer, A. Bates, O. Hermant, D. Eyers, J. Bacon, and M. Seltzer, “Runtime Analysis of Whole-system Provenance,” in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2018, pp. 1601–1616.

[88] W. U. Hassan, A. Bates, and D. Marino, “Tactical provenance analysis for endpoint detection and response systems,” in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2020, pp. 1172–1189.

[89] W. U. Hassan, M. A. Noureddine, P. Datta, and A. Bates, “Omegalog: High-fidelity attack investigation via transparent multi-layer log analysis,” in *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*. ISOC, 2020.