CONnecting the EXtra doTS (CONTEXTS): Correlating External Information about Point of Interest for Attack Investigation

Sareh Mohammadi[*✉], Hugo Kermabon-Bobinnec[*], Azadeh Tabiban[†], Lingyu Wang[*✉], Tomás Navarro Múnera[*], Yosr Jarraya[‡]

[*]*CIISE, Concordia University, Montreal, Canada, {sareh.mohammadi, hugo.kermabonbobinnec, lingyu.wang, tomas.navarromunera}@concordia.ca*

[†]*University of Manitoba, Winnipeg, Canada, azadeh.tabiban@umanitoba.ca*

[‡]*Ericsson Security Research, Montreal, Canada, yosr.jarraya@ericsson.com*

Abstract*—Provenance analysis is one of the go-to solutions today for human analysts to investigate security incidents. To assist analysts in managing the sheer size of provenance graphs, many pruning solutions have been proposed. Such solutions rely on graph-theory features, anomaly detection, and other techniques to identify nodes and edges that are irrelevant to the detected incident. Despite differences in their methodologies, those solutions typically share a common approach when it comes to the detected incident, i.e., they merely regard the incident as an abstract starting point, without tapping into it further. However, we observe that this may lead to missed opportunities for pruning, since the incident is typically associated with external information, e.g., knowledge about the exploit or the vulnerability, which may provide extra contextual insights for effective pruning. Based on such an observation, we propose CONTEXTS, a solution that complements existing pruning approaches by leveraging external information about the incident. Specifically, the solution extracts contextual information from external sources, maps such information to provenance graph nodes, and then correlates those nodes to form a subgraph relevant to the incident. Our implementation and experiments based on real-world attacks demonstrate its effectiveness, e.g., working as the pre-processor of an existing pruning approach, it helps to reduce the false positives from more than 150k to less than ten, and as a standalone pruning solution, CONTEXTS achieves 100% TPR for 19 out of 20 attacks, with an FPR below 0.6% for 16 out of 20 attacks. Finally, its real-world practicality is illustrated through a user study where 94.4% of participants agreed with its usefulness in attack investigation.

1. Introduction

Investigating a detected security incident to identify the attacker's activities is usually a prerequisite for understanding the root cause and consequently deploying mitigation solutions. To this end, system auditing tools (e.g., [1], [2]) provide necessary inputs such as system logs, although those tools typically lack an explicit representation of the dependency between logged activities [3]. To address this, provenance analysis provides an established solution for identifying the causal relationships between logged activities in the form of a graph, namely, provenance graph (PG). PGs have seen various applications in security, such as attack detection [4], [5], [6], threat hunting [7], [8], and forensic analysis [9], [10], [11], [12], [13]. In particular, to assist analysts in attack investigation, the fine-grained node/edge-level anomaly detection approaches [14], [15] reconstruct APTs by connecting the identified anomalous nodes or edges, while the threat hunting approaches [8], [16], [17], [18] search PGs for predefined sequences of signatures (a.k.a. Indication of Compromise). Other approaches focus on in-depth analysis of individual alerts to identify the underlying vulnerabilities [9], [10], [11], [19], [20]. These latter approaches typically consider the alert as a Point of Interest (POI) to extract a dependency graph from the PG, either by identifying paths leading to the POI (e.g., [10], [19], [20], [21]) or through backward/forward searches from the POI (e.g., [3], [9], [12], [22], [23]).

A key challenge here is that the extracted dependency graph is usually still too large for human analysts to understand. Therefore, a crucial next step is to prune the dependency graph, e.g., based on anomaly scores [9], [10], [12], [13], data flows [9], [12], [19], [23], node degrees [9], [10], [12], merging parallel edges [10], [19], [20], [22], and aggregating nodes [20], [24]. However, existing studies indicate that those pruning solutions may face various challenges in practice: (i) A backward search may cause dependency explosion [22], [25], whereas limiting paths to specific lengths may lead to missing activities in the dependency graph [10], [21]. (ii) The accuracy of anomaly scores inherently relies on the completeness of benign activities used for training [9], [20]. (iii) Graph-theory features may not capture the semantics of attacks [19], [20]. (iv) Many approaches require numerous traversals of large dependency graphs, which may become computationally expensive for large PGs [25].

Our key observation is that, despite the differences in their methodologies, existing pruning methods mostly adopt a similar approach when it comes to the POI, i.e., *regarding the POI as an abstract node(s) in the PG*. However, this may lead to missed opportunities for pruning since it ignores the fact that, external to the provenance graph, the incident

✉ Corresponding authors.

alert behind the POI is typically associated with external information from various sources, e.g., knowledge about the attack captured in the detection rule that generates the alert, or knowledge about the exploited vulnerability. Such information can usually provide extra contextual insights to enable additional pruning, as demonstrated in our work.

Based on this observation, we propose CONTEXTS, a solution that can enhance existing provenance-based attack investigation approaches through providing additional pruning capabilities. Our key idea is threefold. First, in addition to regarding the POI as a starting point, we tap into it further by collecting external information from various sources about the incident alert behind the POI. Second, the collected information is then mapped to the PG to establish additional points of interest, namely, *waypoints*. Third, as the original POI and the waypoints represent different perspectives of the same attack (e.g., knowledge about the exploit versus knowledge about the vulnerability), correlating them in the PG may achieve a "triangulation" effect to effectively narrow down the scope of investigation to a smaller subgraph containing relevant nodes and edges.

More specifically, CONTEXTS first builds a knowledge base offline to store external information related to known alerts. Second, upon receiving an incident alert, it automatically extracts the related external information from the knowledge base, and generates queries to label corresponding PG nodes as the POI and waypoints. Third, to correlate those labeled nodes, it leverages both graph-theoretic techniques such as path finding and vicinity expansion, and semantic approaches such as retrieving process nodes' ancestor/descendant hierarchy, and limiting the expansion at frequent system activities. We focus on vulnerabilities affecting the kernel and its associated utilities in this paper due to their relative importance and the need highlighted in recent works [26], [27], although the methodology is applicable to other types of vulnerabilities. In summary, our main contributions are as follows.

- We address the research gap that external information about the detected incident is not leveraged in existing pruning approaches. By leveraging such external information in a systematic way, CONTEXTS can work as both a standalone solution for attack investigation, and a pre-processor to provide significantly smaller inputs (than original PGs) to existing pruning solutions for enhancing their effectiveness.
- We provide detailed methodology for CONTEXTS to (i) build its knowledge base of external information, (ii) map such information to PG elements, (iii) correlate those PG elements to form the output subgraph, (iv) apply the output subgraph for attack investigation, root cause derivation, and enriching intrusion detection system (IDS) rule-sets.
- We implement and evaluate CONTEXTS based on real-world security incidents. Our experiments demonstrate that it can significantly improve the effectiveness of existing pruning approaches, e.g., reducing the number of false positives from thousands to less than 50 in all cases. Additionally, as a standalone pruning solution, CONTEXTS achieves 100% TPR for 19 out of 20 attacks, with an FPR below 0.6% for 16 out of 20 attacks. Furthermore, its performance is robust against noises in the PG, e.g., the number of edges in the output subgraph remains below 200 as the PG size increases by thirty times (with more than 180,000 edges). Finally, our user study shows its real-world practicality, where the participants reported with an average certainty of 4.85 (out of 5) that CONTEXTS is effective in reducing the scope of analysis, and 94.4% reported CONTEXTS helped them to identify the most relevant attack activities.

2. Preliminaries

This section provides an example and our threat model.

2.1. Motivating Example

Figure 1 illustrates the challenges faced by existing pruning solutions (left) and our key ideas (right).

Attack Description. For simplicity, this example relies on a simple attack scenario (more complex scenarios are discussed in Section 4.2.2). This attack scenario assumes a `setuid` *bit set* alert is generated by the exploitation of CVE-2022-0847 (i.e., *Dirty Pipe*, a local privilege escalation vulnerability in the Linux kernel [28], [29]). The vulnerability enables writing to memory pages in the page cache that are linked to read-only files, which should normally be immutable. To exploit the vulnerability, the attacker fills a Linux pipe to set the `PIPE_BUF_MERGE` flag, then empties the pipe and replaces it with their intended data. When the *splice* system call merges the page caches, the flag causes the new data to be merged into the original read-only file. Exploiting this vulnerability, an attacker can replace the content of a `setuid` bit-enabled binary (e.g., the *passwd* utility) with his/her own shellcode to obtain the root access [30].

Challenges Faced by Existing Approaches. Even to investigate such a simple attack, the analyst may face a fairly large PG (illustrated on the left side of Figure 1), since the attack-related activities may be hidden inside thousands of system calls generated by system auditing tools such as Sysdig [2] and Strace [1]. Therefore, most provenance analysis solutions employ one or more pruning techniques to reduce the scope of analysis, although with limitations. First, the V-shape gray shade in the figure illustrates that searching backward from the POI can lead to a large number of false dependencies [22], as illustrated by white nodes inside the V-shape. Second, the nodes with numbers inside illustrate anomaly score-based techniques, which inherently relies on the completeness of training data, a condition that is typically difficult to ensure due to diverse system behaviors [9], [20]. Third, as illustrated by a white node connected to many edges, the degree of node and other generic graph-theory features may not capture the attack semantics, and thus can be inaccurate [19], [20]. Fourth, the dashed line shows that searching for connected paths of specific lengths can miss an attack-related node (i.e., the black node above the dashed line) because it is located beyond the specified path length [9], [12].

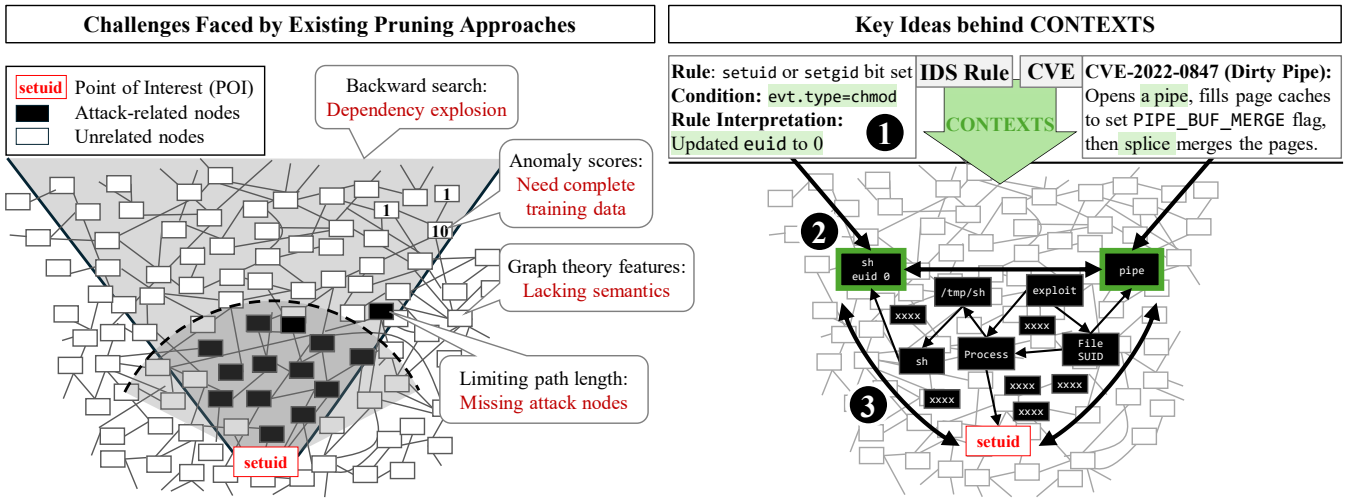

Figure 1: An example of attack investigation using existing pruning approaches (left) and CONTEXTS (right).

Finally, although not shown in the figure, many of those approaches may require numerous graph traversals (e.g., for calculating the occurrence probability of all events or assigning weights to all nodes and edges), which may become computationally expensive for large PGs [25].

Key Ideas Behind CONTEXTS. Instead of directly addressing the aforementioned challenges, CONTEXTS leverages external information about the POI to prune the PG in a way that is complementary to most existing approaches (as they do not employ such information). As a result, the output of CONTEXTS may serve as an (significantly smaller) input to existing approaches to indirectly improve their effectiveness, as demonstrated in our experimental results in Section 4. Specifically, the right side of Figure 1 illustrates the main ideas behind CONTEXTS. First, externally to the PG, we identify additional contextual information (highlighted in green) through analyzing the IDS detection rule used to generate the incident alert (e.g., the event type is *chmod* and *euid* is updated to *0*, i.e., *root*), the CVE description of the corresponding vulnerability (e.g., the use of *splice* and *pipe* system calls), and potentially other data sources (❶). Second, we map such external information to PG node or edge properties, and automatically generate queries to label the corresponding PG elements as waypoints, as illustrated by the nodes with green borders in the figure (❷). Third, we correlate those two waypoints together with the POI to form a subgraph, illustrated by the triangular shape formed by the three double arrows (❸). The subgraph is relatively small, and yet covers all the attack-related activities, i.e., (i) the attacker takes advantage of the *Dirty Pipe* vulnerability to overwrite part of the *SUID* binary with a crafted payload; (ii) upon successful hijacking, s/he executes the hijacked binary, which in turn drops another executable binary (*/tmp/sh*), writes to it, and changes its permission; (iii) the second binary (*/tmp/sh*) executes a shell code containing `setuid(0)`, `setgid(0)` to provide a root shell.

2.2. Threat Model

Following existing provenance-based solutions [10], [21], the underlying operating system, auditing framework, IDS, and provenance graph generator are assumed to be parts of the trusted computing base (TCB). Therefore, attacks that may compromise the auditing, detection, log collection, and provenance generation or analysis subsystems are beyond the scope of this work. Undetected 0-day attacks are out of the scope, unless there is a POI to start with (e.g., they are followed by other attack steps that are detectable using IDS, or they are detected by node/edge-level anomaly detection approaches [14], [15]). In the latter case, the dynamic waypoint approach of CONTEXTS can be applied, as detailed in Section 3.3. Attacks not involving system calls at all, e.g., stack-based buffer overflow and race condition in user space, are also out of scope. Finally, we assume the external information obtained from public sources such as IDS rule-sets and CVE databases is accurate and up-to-date.

3. Methodology and Implementation

This section first provides an overview of CONTEXTS and then details each of its four stages.

Overview. As shown in Figure 2, CONTEXTS consists of four main stages as follows. First, it extracts contextual information from data sources such as IDS rule-sets and CVE databases, and stores such information inside a repository. Second, upon receiving an alert, it retrieves contextual information related to the alert from the repository, and generates queries to identify and label relevant PG nodes and edges as waypoints (WPs). Third, it correlates the WPs and POI to obtain a subgraph containing nodes and edges relevant to the alert. Finally, it utilizes the subgraph to assist the analyst in investigating the attack, deriving its root cause, or enriching the IDS rule-set.

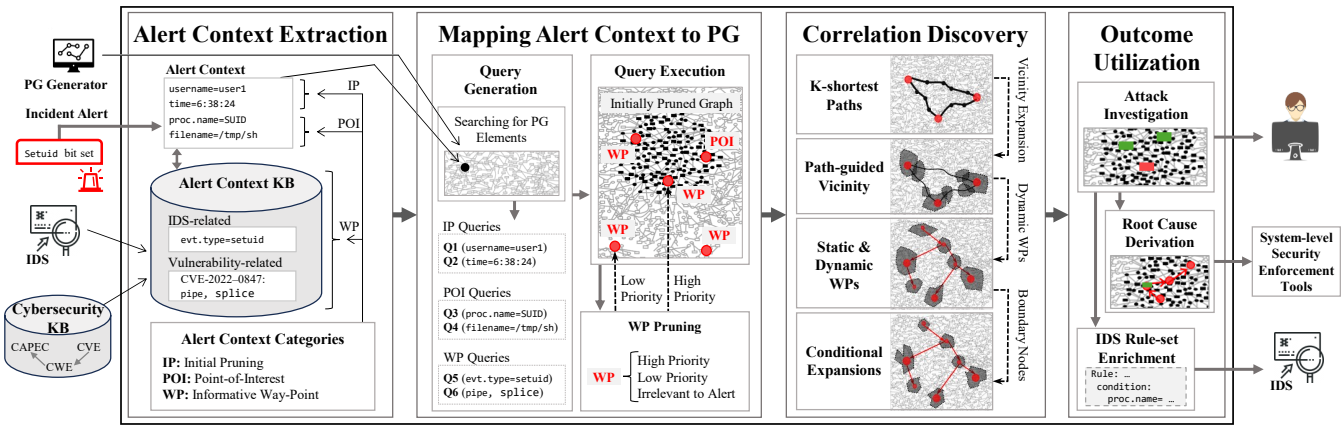

Figure 2: Overview of CONTEXTS.

3.1. Alert Context Extraction

This section details how CONTEXTS collects external information related to alerts, namely, *alert context*.

Alert Context Knowledge Base. CONTEXTS extracts alert contexts from various data sources and stores such information in a local repository, namely, *alert context knowledge base* (ACKB). Depending on the data sources, alert contexts can be divided into two broad categories: (i) IDS-related alert contexts include information that is explicitly represented in the alert (e.g., the detection time), and information that is not part of the alert itself, but can be derived from conditions or keywords used by a rule-based IDS (e.g., [31], [32]) to generate the alert, such as *evt.type=chmod*. (ii) Vulnerability-related alert contexts include information about the vulnerabilities, weaknesses, or exploit patterns related to the alert, and can be obtained from public repositories (e.g., CVE [33], CWE [34], and CAPEC [35]).

To extract such alert contexts from the corresponding data sources, CONTEXTS searches their provided data for information that matches what is captured inside a system-level PG (which is our focus in this paper), e.g., filenames, libraries, and system calls. To achieve this, our methodology is twofold. First, we build a lexicon of Linux kernel-related concepts to be utilized by CONTEXTS as a reference to perform searches on data from various sources. The search results provide alert contexts that can later be mapped to PG elements (detailed in Section 3.2). Additionally, for vulnerability-related alert contexts, CONTEXTS searches for CVEs with similar impacts on the system (e.g., privilege escalation) as indicated in each alert, and regards such CVEs as potential causes of that alert. Once the alert context extracted from a CVE is successfully mapped to the PG, that CVE enables CONTEXTS to further link the alert to other data sources, such as CWEs and CAPECs, by following their references to the same CVE, as outlined in the Cybersecurity Knowledge Graph (CSKG) [36], [37], [38]. This allows CONTEXTS to obtain more alert contexts (and waypoints).

Example 1. Figure 3 shows the simplified alert context related to the *setuid bit set* alert. The IDS-related alert

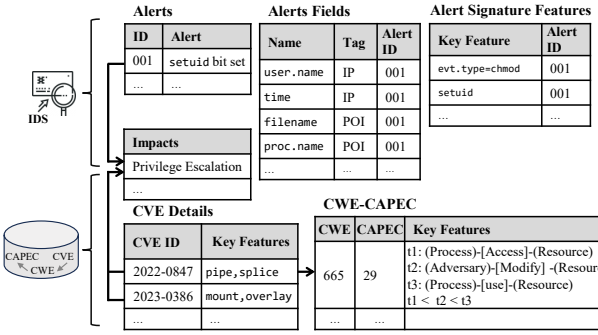

Figure 3: Example alert contexts for a "`setuid` *bit set*" alert.

context is shown at the top. The *Alert Fields* and *Alert Signature Features* tables list information extracted from the alert itself, and from the detection rule features, respectively. Additionally, as the *Impacts* table (middle left) shows, the alert indicates a privilege escalation that links the alert to CVEs with similar impacts (e.g., CVE-2022-0847 and CVE-2023-0386). Correspondingly, the *CVE Details* table shows the vulnerability-related alert context, where the *Key Features* column shows information that can later be mapped to the PG. In particular, CVE-2022-0847 involves the combination of *pipe* and *splice* system calls, and CVE-2023-0386 relates to the *OverlayFS* subsystem. Furthermore, CVE-2022-0847 is linked to the *Improper Initialization* weakness (CWE-665) [29], which arises due to a Race Condition (CAPEC-29) [39]. Those can provide additional alert contexts (and waypoints).

Alert Context Categorization. CONTEXTS classifies the alert context information into three categories, each representing a different purpose in the later stages of the methodology. First, *Initial Pruning* (IP) alert contexts provide an overview about the alert incident, such as username and time, which can be applied for an initial pass of PG pruning (e.g., using a time-based pruning method [9], [12], [40]). Second, *Point-Of-Interest* (POI) alert contexts offer detailed information about the detected incident, such as the process name or the name of the affected file, which

will be used to prune potential waypoints (Section 3.2) and correlate with waypoints (Section 3.3). Third, *Waypoint* (WP) alert contexts provide additional information about the detection or cause of the alert, such as features of the alert signature and CVE, which will be used to correlate with the POI (Section 3.3). For instance, in Figure 3, the IP and POI categories are indicated in the *Tag* column of the *Alerts Fields* table, and the data in the other tables generally belong to the WP category.

Implementation. We implement the alert context knowledge base using PostgreSQL [41] for its advanced features like full-text search. We deploy Falco [42], a popular, open-source, rule-based IDS for detection. We build a lexicon of Linux kernel-related concepts from the *man-pages* project, by processing each page and identifying associated filenames, processes, and system calls using regular expressions. To link alerts to CVEs, we process all Linux kernel vulnerability descriptions obtained from the National Vulnerability Database (NVD) [43], and extract the *impact* of each CVE. This is achieved either based on the cited CWE and CAPEC entries when available, or through matching the keywords using lemmatization with the Spacy Python library [44], e.g., both "escalate their privileges", "escalation of privileges" will match with "privilege escalation" (a similar lemmatization technique is also applied to the descriptions of the alerts).

3.2. Mapping Alert Context to PG

This section details how CONTEXTS maps alert contexts to a PG through generating graph queries and labeling PG elements based on their relevance to the incident.

Query Generation. To map alert contexts to a PG, CONTEXTS automatically generates graph queries to identify the corresponding PG nodes and edges. This is achieved in two steps. First, it searches the properties of all the nodes and edges to identify those that store the alert context information. This is necessary considering the fact that different provenance construction tools (e.g., [45], [46], [47]) may adopt different names in the node and edge properties for the same concept. For instance, SPADE [47] records the file name in *path* property of nodes and PROVDETECTOR [21] records it in *file path* property. Second, CONTEXTS formulates a condition for each node/edge property identified in the previous step, and constructs a query based on the type of that property. For instance, it generates a node query `MATCH (node) WHERE C` for each condition `C` that is mapped to a node property. Conversely, it generates an edge query `MATCH (Node1)-[Edge]-(Node2) WHERE C` for each `C` mapped to an edge property.

Example 2. Figure 4 (top left) shows the table created by mapping the *filename* and *evt.type* alert context information to PG elements. Specifically, CONTEXTS searches the PG based on the key-value pair (*filename*, */tmp/sh*) to find the value of *filename* (i.e., */tmp/sh*) stored in the property *path* of the nodes representing system files (i.e., *Node.subtype* = *file*). Similarly, (*evt.type*, *chmod*) is found in an edge property *Edge.operation*. Based on those mappings, Figure 4 (bottom left) shows how CONTEXTS generates a node query and an edge query, respectively.

Query Execution. CONTEXTS prioritizes the execution of generated queries by dividing them into three categories based on the corresponding types of alert contexts (Section 3.1), i.e., Initial Pruning (IP) queries, Point-Of-Interest (POI) queries, and Waypoint (WP) queries. First, IP queries are given the highest priority for execution. By labeling the intersection of the results of those IP queries, CONTEXTS can identify a smaller subgraph G' of the original PG to make the execution of subsequent queries more efficient. Second, POI queries are executed next to identify and label nodes within G' as the POI. Third, WP queries are executed at last on the original PG instead of G', since their results are not necessarily limited to be within G'.

Example 3. Following Example 2, the middle of Figure 4 illustrates the query execution. First, executing the two IP queries Q_1 and Q_2 labels G' within the PG (nodes highlighted in black) based on the username specified in the alert (*username=user1*), and the time of the alert (*time=6:38:24*), respectively. Second, executing the two POI queries Q_3 and Q_4 labels the POI within G' as nodes satisfying the properties *filename=/tmp/sh* and *proc.name=SUID*, respectively. Finally, executing the WP queries (i.e., $Q_5 - Q_8$) labels two groups of nodes as waypoints WP_1 and WP_2 (to be detailed in the next example).

Waypoint Pruning. Since alert contexts are extracted from external data sources, the corresponding waypoints may not always be relevant to the investigated alert, e.g., WP queries may return multiple CVEs with similar impacts as the alert. Therefore, in waypoint pruning, CONTEXTS examines the nodes returned by WP queries, and prune those that are not connected to the POI. More formally, the i^{th} waypoint can be denoted as $\{n \mid n \in \textit{Nodes}(G), n \in \textit{Result(WP-}Q_i\textit{)}, n \overset{\text{path}}{\leftrightarrow} POI\}$. Moreover, waypoints that fall inside the subgraph G' will be given higher priorities in correlation discovery (Section 3.3), which is helpful in cases where a WP query returns several disconnected nodes.

Example 4. Following Example 3, the right side of Figure 4 shows a flowchart for waypoint pruning and several example results of WP queries. First, since the result of Q_5 is part of the POI, it is labeled as such. Second, since the results of Q_6 and Q_7 are both connected to the POI and inside the subgraph G', those are labeled as WP_1 and WP_2, respectively, and will be given priority. Finally, the result of Q_8 is not labeled since it is not connected to the POI.

Implementation. We utilize the *Auditd* Linux auditing daemon [48] to collect logs, and the latest version of SPADE [47] to generate provenance graphs from the logs. We specify the *syscalls=all* option in order to include additional system calls. We store the captured provenance data in a Neo4j [49] database. We retrieve all properties of nodes and edges from the SPADE data model [50], and

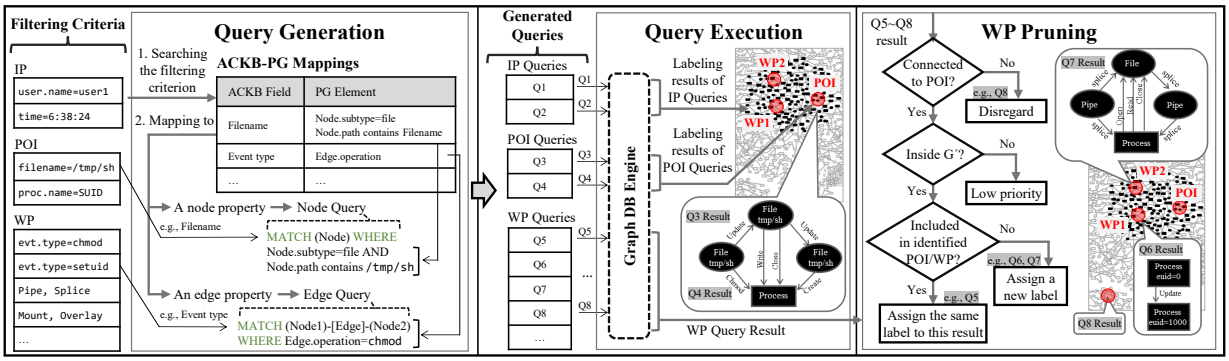

Figure 4: Example of query generation and execution for a "`setuid` bit set" alert.

query those properties to identify the mapping from ACKB to PG for each alert context, with the results stored in our PostgreSQL database. We use Python for generating queries in the Cypher [51] query language, and we execute those queries through the Neo4j Python driver API. For waypoint pruning, we utilize the *apoc.path.expandConfig* procedure [52] from Neo4j's *APOC* library.

3.3. Correlation Discovery

This section presents a series of correlation discovery approaches for linking the POI and WPs to form a subgraph relevant to the investigated alert. Those approaches are *incremental* in the sense that each subsequent approach incorporates additional features to improve the result.

K-Shortest Paths (KPath). Since the POI and WPs will mostly be treated equally during the correlation discovery stage, we will refer to them collectively as *informative points* from now on. Given that those informative points may represent different perspectives of the incident, a straightforward way to correlate them is by examining the paths connecting the PG nodes involved in those informative points. Based on such an intuition, our first approach, the K-Shortest Paths (KPath), finds the top K paths with the smallest number of edges between each pair of informative points. K is a parameter that can be adjusted to strike a balance between better coverage of relevant nodes (larger K) and lower computational overhead with less irrelevant nodes (smaller K). Nonetheless, a key challenge is that, even with a relatively small K, a standard K-shortest path algorithm [53] may still lead to many irrelevant nodes since it ignores the semantics of PGs.

To address this, we propose a semantics-aware K-shortest path approach as follows: (i) We only discover the K shortest paths between nodes representing processes from different informative points, while paths between other nodes in those informative points (e.g., files) will be kept implicit. The rationale is that those process nodes typically represent the activities to which other nodes in the same informative point are connected. Therefore, a path between the process nodes may be sufficient to correlate all other nodes in the informative points. (ii) When an informative point contains multiple connected process nodes, we only

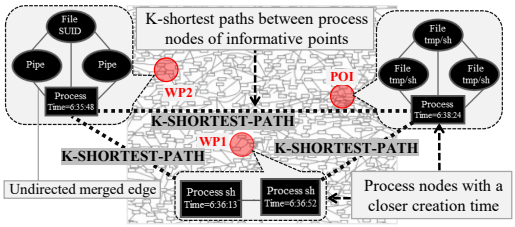

Figure 5: Example of the KPath approach.

discover the shortest paths between the pair of process nodes with the closest creation times. The rationale is similar, i.e., the shortest paths between two process nodes with the closest creation times are likely shared by the paths between other process nodes in the same informative points. More formally, the shortest paths between two informative points IPt_i and IPt_j are obtained based on the process nodes $P_i \in IPt_i$ and $P_j \in IPt_j$ where $P_i.time - P_j.time = \min_{\forall P'_i \in IPt_i, \forall P'_j \in IPt_j}(|P'_i.time - P'_j.time|)$. Algorithm 1 (in Appendix A) provides further details of the KPath approach, and we evaluate its performance in Section 4.4.

Example 5. Figure 5 illustrates how the KPath approach finds the shortest paths between the three informative points. By identifying paths between the process nodes, all the other connected nodes (e.g., *File* nodes) can also be correlated without the need for explicitly identifying the paths between them (which would incur significantly more overhead as there are totally 32 pairs of nodes).

Path-Guided Vicinity (PthVic). Our second approach, the path-guided vicinity (PthVic), adds vicinity expansion to the KPath approach. The intuition is that the PG nodes close to the informative points and shortest paths are likely also related to the incident. Therefore, the PthVic approach expands from the result of KPath in two steps. First, it expands to the N-hop vicinity of each node inside the informative points. Specifically, starting from each such node, it traverses connected edges up to N hops, and labels the nodes and edges included in such a traversal. N is a parameter that can be adjusted to achieve more coverage of relevant nodes, without causing dependency explosion [22], which will be further evaluated in Section 4.4. More formally, the N-hop vicinity of an informative point *IPt* can be represented as:

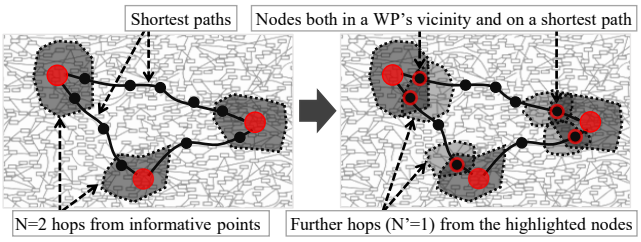

Figure 6: Example of the PthVic approach.

$$\text{N-hop}(IPt) = \bigcup_{\forall x \in \text{Nodes}(IPt)} \big(\text{Vic}(x,N)\big), \text{ where}$$

$$\text{Vic}(x,N) = \begin{cases} x & \text{if } N = 0 \\ \text{Vic}(x,N\text{-}1) \cup \left\{ \begin{matrix} n \in \text{Nodes}(G),\ e \in \text{Edges}(G) \mid \\ \exists m \in \text{Nodes}\big(\text{Vic}(x,N\text{-}1)\big), n \leftrightarrows m \end{matrix} \right\} & \text{if } N > 1 \end{cases} \quad (1)$$

Second, the PthVic approach also expands along the K shortest paths. Specifically, starting from each node that is both (i) inside the N-hop vicinity of an informative point (as defined above), and (ii) on one of the K shortest paths (identified by the KPath approach), it expands N' further hops, where N' is a parameter satisfying $N' < N$. Algorithm 2 (in Appendix A) outlines the steps of this approach, and we will evaluate its performance under different combinations of parameter values through experiments in Section 4.4.

Example 6. The left side of Figure 6 depicts the 2-hop vicinity of the informative points (N=2), as well as the shortest path (K=1) between them. The right side highlights the nodes that are both inside the vicinity and on the paths, and depicts how the 2-hop vicinity is selectively expanded to 3-hop (N'=1) from only those nodes.

Vicinity of Static and Dynamic WPs (StatDyn). The previous two approaches only leverage WPs derived from alert contexts (i.e., information extracted from external sources), which will be referred to as *static WPs* from now on. Our next approach, the vicinity of static and dynamic WPs (StatDyn), additionally captures *dynamic WPs*, which correspond to the ancestor and descendant processes of the process nodes inside the informative points (POI and static WPs). The intuition is that those ancestor and descendant processes may be indirectly related to the incident, and capturing their vicinity may identify additional relevant nodes.

This approach has two important implications. First, those are called dynamic WPs since they may vary depending on the attackers' (order of) activities, which enables CONTEXTS to capture different attack instances exploiting the same vulnerability. Second, since dynamic WPs can be obtained using only POI, the StatDyn approach allows CONTEXTS to be applied to cases where no static WPs are available (e.g., no alert context is found from external sources). Algorithm 3 (in Appendix A) details the steps of this approach, and Section 4.4 examines the effectiveness of the StatDyn approach in scenarios both with and without static WPs.

Example 7. Figure 7 (left) shows the process nodes within the three informative points (red circles) and their ancestor (blue) and descendant (green) processes, where the parent-child relationships are indicated by the *Pid* (process ID) and *PPid* (Parent Process ID) of process nodes. Figure 7 (right) depicts the captured vicinity of the static and dynamic WPs.

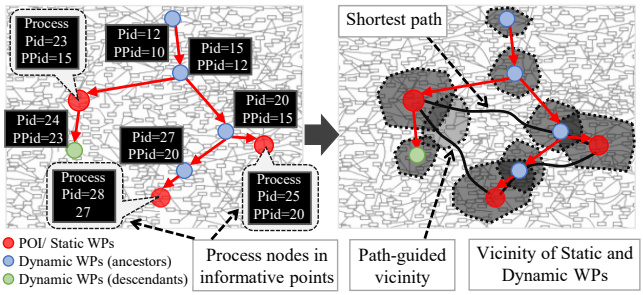

Figure 7: Example of the StatDyn approach.

Conditional Expansion (CondExp). Our final approach, the conditional expansion (CondExp), incorporates all the aforementioned features, and additionally mitigates the potential dependency explosion caused by nodes representing frequent system activities. Specifically, such nodes are typically connected to a large number of activities that may be irrelevant to the investigated incident [24], e.g., shared object files used in Unix-like operating systems, or DLL files in Windows systems. The CondExp approach identifies such *boundary nodes* (denoted as BN) as having a direct edge to at least τ process nodes not inside any dynamic WPs, where τ is a parameter, as detailed below:

$$BN = \big\{ n \in \text{Nodes}(G) \mid \tau \leq |\{proc \in P \mid (n, proc) \in \text{Edge}(G)\}| \big\},$$
$$\text{where } P = \{p \in \text{Nodes}(G) \mid p.type = \text{Process} \wedge p \notin \text{Dynamic WPs}\}$$

To avoid potential dependency explosion caused by the boundary nodes, the CondExp approach limits the extent of vicinity expansion from such nodes (while expanding from other nodes as usual). Specifically, it expands N'' hops from each boundary nodes, where N'' is a parameter satisfying $N'' < N$. Section 4.4 evaluates this approach under different parameter values.

Example 8. Figure 8 (left) illustrates an example of two boundary nodes, i.e., the *root bash* process node, which is extracted as the ultimate ancestor of other processes, and a node corresponding to shared library (*.so*) files [54], which are commonly used by processes in Unix-like operating systems. Figure 8 (right) shows the CondExp approach with N=2 and N'=0, where the expansion of static and dynamic WPs' vicinity is terminated at the boundary nodes.

Implementation. To implement those correlation discovery approaches, we have developed 19 query patterns to generate the corresponding Cypher queries in Python. For the K-Shortest Paths approach, we aggregate the parallel edges between each pair of nodes using *gds.graph.project*, and utilize Yen's algorithm [55] from the Neo4j Graph Data Science (GDS) library [56] to compute a number of shortest paths between two nodes.

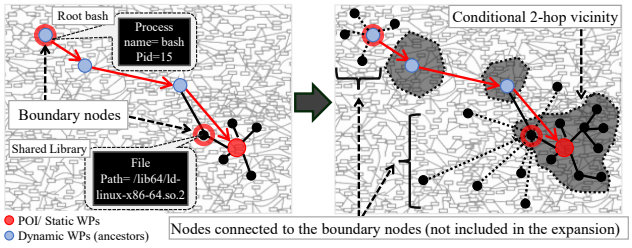

Figure 8: Example of the CondExp approach.

3.4. Outcome Utilization

Since the main outcome of CONTEXTS is a subgraph that contains PG nodes and edges relevant to the given incident, it can be directly utilized by analysts to reduce the scope of attack investigations. Moreover, this subgraph (instead of the original PG) can be used as the input for existing provenance-based attack investigation solutions, and the significantly reduced size of such an input can potentially improve their effectiveness, as demonstrated in Section 4.3. Additionally, the parameterized approach of CONTEXTS allows analysts to control the scope and size of its outcome (e.g., through adjusting K, N, etc.) based on their needs, as demonstrated through our parameter tuning experiments (Section 4.4). Finally, the outcome of CONTEXTS can be further utilized for performing root cause analysis and enriching IDS rule-sets, as detailed in the following.

Root Cause Analysis. As one of the common goals of attack investigation, identifying the root cause of an incident is usually crucial for attack prevention or mitigation [27], [57], [58]. To assist analysts in such a challenging task, the outcome of CONTEXTS is designed to include additional node labels that can be used by analysts to prioritize their analysis of the root cause. Those node labels can be divided into two main categories, i.e., (i) labels that reflect graph theory information, such as the rank of paths including the nodes and the number of hops away from an informative point, (ii) labels that reflect semantics of the nodes, such as the alert contexts associated with informative points, the ancestor processes of dynamic waypoints, and the system activities represented by boundary nodes. Furthermore, the outcome of CONTEXTS includes auxiliary information that allows it to (i) order the PG nodes based on the time of their occurrence, and (ii) map those nodes back to corresponding data items in the system traces (e.g., Auditd logs adopted by SPADE [47]). Those features can benefit root cause analysis applications that rely on system calls [59], [60], [61], their arguments [27], [58], [62], and their timestamps [63].

Example 9. In Figure 9, CONTEXTS helps analysts derive a sequence of system calls as the root cause. Specifically, the node labels allow analysts to focus on the vicinity of informative points and order the edges (i.e., system calls) chronologically to form a sequence (from `fork` to `close`).

IDS Rule-set Enrichment. The rule-set of a rule-based IDS is typically maintained based on historical data about known attacks and human knowledge (e.g., derived from

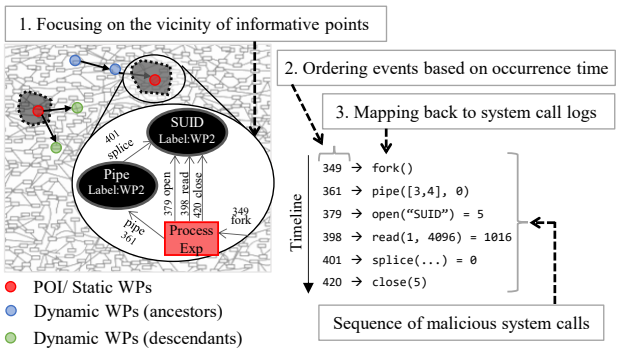

Figure 9: Example of root cause analysis utilizing CONTEXTS outcome.

reverse engineering or honeypot analysis [64], [65], [66]). Although not designed for this purpose, the dynamic waypoint approach of CONTEXTS (detailed in Section 3.3) may provide an opportunity to construct new IDS rules to detect malicious activities that are conducted before (e.g., initial access or execution) or after (e.g., persistence or exfiltration) a detected incident. Specifically, such activities can be identified through investigating the ancestor and descendant nodes of static waypoints and POI. New IDS rules would be created to detect those activities, if they are not yet detected by the IDS, but are either (i) executed prior to the incident, or (ii) can be part of other attacks. The former case can enable an early detection, while the latter may potentially allow detecting unknown attacks.

Example 10. Figure 10 shows how CONTEXTS helps forming a new rule for Falco IDS [42] to enable the early detection of CVE-2022-0492, which is a container escape attack leveraging the Cgroup *notify_on_release* feature [67]. The incident alert mentions a shell opened by a root user, indicating a privilege escalation. The outcome of CONTEXTS enables the analyst to trace the static WPs (in red) back to their ancestors (in blue). This reveals an unusual relationship (*mount* with the *cgroups* filesystem type) between an ancestor process node (the *sh* process) and the WP (the */tmp/escape* file). Accordingly, the analyst can create a new rule to detect the use of the *mount* system calls (`evt.type="mount"`) with *cgroups* as filesystem type (`evt.args.flags contains "cgroups"`), and thus enable the early detection of a similar future attacks.

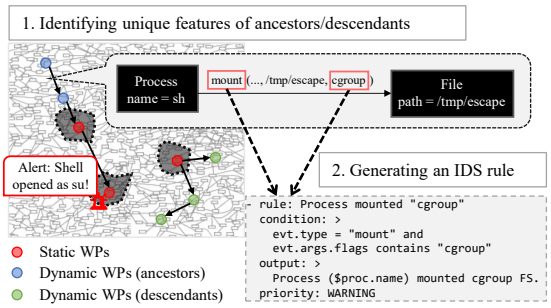

Figure 10: Example of updating IDS rules utilizing CONTEXTS outcome.

4. Evaluation

This section evaluates CONTEXTS by answering the following research questions:
RQ1: How effective is CONTEXTS in identifying the exploit of real-world vulnerabilities, while working independently?
RQ2: How much can CONTEXTS improve the results of existing pruning solutions, while working as a pre-processor?
RQ3: How accurate are the different correlation approaches of CONTEXTS, and what are their optimal parameters?
RQ4: How scalable is CONTEXTS in terms of handling PGs with an increased amount of irrelevant nodes?
RQ5: How do real users from different backgrounds think about CONTEXTS while using it for attack investigation?

4.1. Experimental Settings

All the experiments are performed on a testbed consisting of an IDS (Falco [42]) and a PG generation tool (SPADE [47]) running on a virtual machine (VM) with 16 vCPUs and 32GB of RAM with various Linux distributions and kernel versions depending on the investigated attacks.

Dataset[1] Preparation. To evaluate CONTEXTS, the experimental dataset needs to satisfy the following two requirements. (i) Since both the IDS and PG generation tools (Falco and SPADE) work at the kernel level, the dataset naturally focuses on kernel-level and system call-related CVEs. (ii) The experimental results need to be validated against the actual system calls involved in executing the exploit code, and therefore the dataset should consist of CVEs with available exploit codes[2]. To the best of our knowledge, none of the publicly available datasets (e.g., the DARPA TC dataset [69] and the Unicorn dataset [70]) can meet both of those requirements. To address this challenge, we generated our own datasets by implementing real attacks that target different aspects of the Linux kernel (detailed in Table 5 in Appendix B) using proof-of-concept exploit codes (e.g., [71], [72], [73]). We set up a separate virtual machine to implement each CVE by running a Linux version that was unpatched against the corresponding vulnerability (also detailed in Table 5). We used Vagrant for VM deployment with VirtualBox as the provider, and we utilized "bento", "generic" and "ubuntu" as base boxes [74]. The kernel versions required adjustments (either downgrading or upgrading) in some cases since the default kernel in the Vagrant boxes was typically patched.

Result Evaluation. To evaluate the effectiveness of CONTEXTS in identifying the exploit of a vulnerability, we compare its results to the subgraph of the PG that is related to the execution of the corresponding exploit code, namely, the *reference subgraph* (Appendix C details the extraction of a reference subgraph). Specifically, we identify the edges in the CONTEXTS result that correspond to system calls executed by the exploit code. To ensure exact matches, we map each edge in the reference graph to our result through the unique edge ID assigned by SPADE during PG generation, which also ensures the uniqueness of the corresponding nodes. As the edge IDs change over each run, we only compare the results with the reference subgraph obtained for the same run. We report True Positive (TP) as the number of edges that appear in both the reference subgraph and CONTEXTS results, and False Positive (FP) as the number of edges that appear in CONTEXTS results, but not in the reference subgraph. Additionally, we report the True Positive Rate (TPR) and False Positive Rate (FPR).

1. Our dataset is available online [68].
2. Note CONTEXTS does not require any exploit code to function.

4.2. Effectiveness

This section answers RQ1 by evaluating the effectiveness of CONTEXTS as a standalone tool for attack investigation. Section 4.2.1 presents the overall results for all attacks, and Section 4.2.2 provides more detailed insights on two attacks (due to page limits).

4.2.1. General Results. The first column of Table 1 shows the list of CVEs involved in the implemented attacks. The second column reports the size of the reference subgraph. The third column shows whether the IDS can directly detect the attack, or can only detect its post-exploitation consequences (e.g., unauthorized access to a sensitive file). The next two columns show the availability of static WPs based on two sources of external information, i.e., IDS rules and CVEs. The CVEs in Table 1 are chosen to challenge CONTEXTS with increasingly difficult cases. Specifically, the first six CVEs have static WPs from both the matching IDS rules and the corresponding CVE descriptions. In particular, the sixth row represents a chain of two CVEs and the result of CONTEXTS in correlating the POI and the corresponding WPs. The next three CVEs only have static WPs from the IDS rule, whereas the next four only have static WPs from CVE descriptions (the IDS rules can only generate post-exploitation alerts, which cannot be used for static WPs). Finally, the last seven CVEs have no static WPs at all. Dynamic WPs are always available since they are based on ancestor or descendant processes, which extends the applicability of CONTEXTS. We will provide more details of such WPs in Section 4.2.2.

As the results show, CONTEXTS achieves 100% TPR for 19 out of 20 CVEs, with an FPR below 0.6% for 16 out of 20. The first 13 CVEs have both static and dynamic WPs available, which helps CONTEXTS to achieve a perfect TPR and relatively low FPRs. Among these, the chained CVEs (the sixth row) shows a relatively higher FPR due to the inclusion of some irrelevant nodes along the paths connecting the informative points of the two CVEs. The last seven CVEs lack static WPs, but CONTEXTS can still provide perfect TPRs through its use of dynamic WPs (and allowing a one-hop expansion from the boundary nodes for the last four CVEs, i.e., $N''=1$ in the Conditional Expansion approach, as detailed in Section 3.3). The higher FPRs are due to the combined effect of lacking static WPs and complex exploit codes (as evidenced by larger reference subgraphs). In particular, the lower TPR in the last CVE is due to the limitation in obtaining dynamic WPs from the POI, which appears in a new root shell due to the nature of the attack.

TABLE 1: Overview of the implemented attacks, reference subgraph size, nature of detection, availability of WPs, and overall results of CONTEXTS (using the Conditional Expansion correlation approach, with $N''=1$ for the results indicated with * and $N''=0$ for the rest). Note (1)/(2) indicate different exploitations of the same CVE.

CVE	Reference Subgraph		IDS Detection	Static WPs		Dynamic WPs	CONTEXTS Results	
	# of Nodes	# of Edges		IDS Rules	CVE Info.		TPR	FPR
2021-3156	133	291	Direct	✓	✓	✓	100%	0.31%
2022-0847(1)	56	98	Direct	✓	✓	✓	100%	0.02%
2022-0847(2)	99	297	Direct	✓	✓	✓	100%	0.03%
2021-4034	454	977	Direct	✓	✓	✓	100%	0.3%
2024-48990	1,675	5,529	Direct	✓	✓	✓	100%	0.2%
2021-44228 & 2022-0847	174	2,048	Direct	✓	✓	✓	100%	3.04%
2023-32233	1,361	1,493	Direct	✓	–	✓	100%	0.29%
2016-6516	52	77	Direct	✓	–	✓	100%	0.04%
2021-22555(1)	37	94	Direct	✓	–	✓	100%	0.04%
2023-0386	362	690	Indirect	–	✓	✓	100%	0.39%
2016-5195	62	102	Indirect	–	✓	✓	100%	0.58%
2023-32629	304	576	Indirect	–	✓	✓	100%	0.24%
2023-22809	121	596	Indirect	–	✓	✓	100%	0.44%
2023-31248	93	186	Indirect	–	–	✓	100%	0.11%
2021-3490	66	111	Indirect	–	–	✓	100%	0.36%
2021-22555(2)	41	78	Indirect	–	–	✓	100%	0.17%
2022-34918	93	165	Indirect	–	–	✓	100%*	4.19%*
2022-2639	307	590	Indirect	–	–	✓	100%*	27.84%*
2021-4154	1,797	2,406	Indirect	–	–	✓	100%*	18.21%*
2022-2588	121	2,254	Indirect	–	–	✓	96.76%*	13.23%*

4.2.2. Case Study. This section demonstrates the effectiveness of CONTEXTS by providing more details on how it works for the 10th and 11th CVEs in Table 1. Those CVEs are more representative as they correspond to sophisticated attack scenarios involving different types of WPs.

CVE-2023-0386. For this attack, the IDS detects a post-exploitation step and generates a *"Sensitive file opened for reading"* alert (i.e., the POI), indicating that the `/etc/shadow` file is opened at *time=3:18:21*. CONTEXTS investigates this incident as follows. First, it searches for `/etc/shadow` file node(s) with an edge representing the `open` operation, and labels them as the POI. Second, it generates queries to perform initial pruning based on the temporal information in the POI (i.e., *time=3:18:21*) (Section3.2). Third, it searches for CVE entries with the impact of privilege escalation to generate WP queries, and prunes the obtained WPs based on their relevance to the POI (Section 3.2). In this specific case, the WP queries search for `mount`, `overlay`, or `overlayfs` in the *command* property of process nodes to locate the WPs relevant to the POI. Figure 11 illustrates the simplified result of CONTEXTS (which provides 100% TPR, as mentioned in Table 1). In this result, the POI is shown in step (5). The static WPs are shown in shaded boxes in steps (1) and (2), and all other boxes of type *Process* represent dynamic WPs. From this result, the analyst can easily observe that the attacker (i) creates a filesystem in user space; (ii) creates an isolated namespace, and mounts the overlay filesystem with specified *lower*, *upper*, and *work* directories; (iii) leverages the overlay filesystem to combine directories in a new

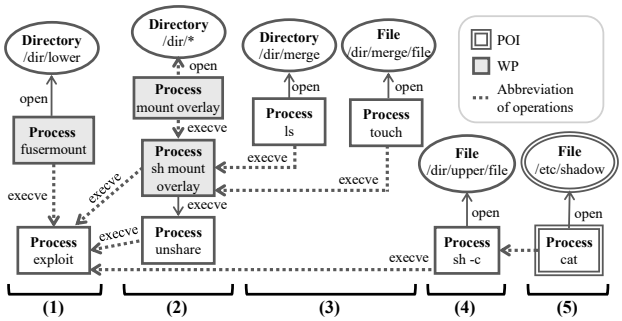

Figure 11: The result of CONTEXTS (simplified) showing the main steps in exploiting CVE-2023-0386.

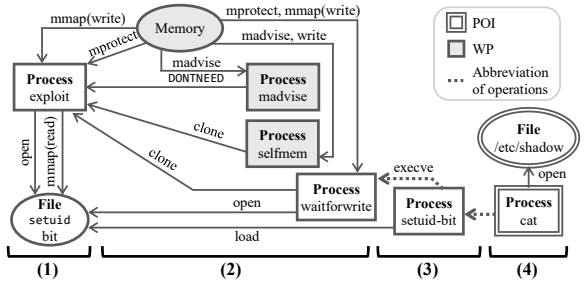

Figure 12: The result of CONTEXTS (simplified) showing the main steps in exploiting CVE-2016-5195 (all memory interactions are shown through a single node for readability).

mount namespace, by creating a new file in the merged directory; (iv) executes a file located in the upper directory of the overlay filesystem, which allows the attacker to run with elevated privileges due to the manipulation of the mount namespaces and overlay filesystem; (v) accesses the `/etc/shadow` file, which triggers the IDS alert.

CVE-2016-5195. Although this is a different CVE, the alert generated by the IDS is similar, since it can only detect the post exploitation. Therefore, CONTEXTS follows similar steps to identify and annotate the POI and WPs, until it identifies the `madvise` operation as a WP. Figure 12 shows the simplified result of CONTEXTS, from which the analyst can observe that the attacker's script (i) takes a `setuid` bit binary and maps it onto memory using `mmap`; (ii) creates two threads to implement a race condition, i.e., the *madvise* process continuously calls `madvise` on the mapped memory with the `DONTNEED` flag, which indicates to the kernel that the memory is not needed, but in reality leads to a race condition in the *copy-on-write* mechanism. As a result, the *selfmem* process repeatedly tries to write a crafted payload into the same memory space; (iii) executes the payload, i.e., a shellcode whose execution grants escalated privileges; (iv) opens the `/etc/shadow` file, which triggers the alert.

4.3. Enhancing Existing Pruning Methods

This section answers RQ2 by integrating CONTEXTS with several popular PG pruning strategies, and comparing the TPR and FPR results with, and without CONTEXTS working as a pre-processor. As illustrated in Table 4

TABLE 2: CONTEXTS as a pre-processor. (B): Baseline, (C): With CONTEXTS. DS1: 10k, DS2: 160k, DS3: 1M.

			Backward/Forward Threshold				**Path Anomaly** Path Length				**Path Time** Time Interval				**Path Anomaly Time** (Time Interval, Path Length) (*10s*, _)				(*120s*, _)			
			0.8	*0.9*	*1*	*none*	*3*	*5*	*10*	*20*	*10s*	*30s*	*60s*	*120s*	*3*	*5*	*10*	*20*	*3*	*5*	*10*	*20*
DS1	*B*	*TP*	18	149	161	654	277	300	308	308	269	310	310	310	258	263	265	265	277	300	308	308
		TPR	2.61%	21.6%	23.3%	94.8%	40.1%	43.5%	44.6%	44.6%	39.0%	44.9%	44.9%	44.9%	37.4%	38.1%	38.4%	38.4%	40.1%	43.5%	44.6%	44.6%
	C	*TP*	18	145	157	651	275	298	306	306	267	308	308	308	257	262	264	264	275	298	306	306
		TPR	2.61%	21.0%	22.8%	94.3%	39.9%	43.2%	44.3%	44.3%	38.7%	44.6%	44.6%	44.6%	37.2%	38.0%	38.3%	38.3%	39.9%	43.2%	44.3%	44.3%
	B	*FP*	2	2.7k	2.7k	5.9k	2.9k	3.2k	3.2k	3.2k	194	3.2k	3.2k	3.2k	144	184	184	184	2.9k	3.2k	3.2k	3.2k
		FPR	0.02%	29.8%	29.8%	64.8%	31.2%	34.5%	34.6%	34.6%	2.11%	34.8%	34.8%	34.8%	1.6%	2.00%	2.00%	2.00%	31.2%	34.5%	34.6%	34.6%
	C	*FP*	0	12	12	34	1	4	6	6	0	7	7	7	0	0	0	0	1	4	6	6
		FPR	0%	0.13%	0.13%	0.37%	0.01%	0.04%	0.07%	0.07%	0%	0.08%	0.08%	0.08%	0%	0%	0%	0%	0.01%	0.04%	0.07%	0.07%
DS2	*B*	*TP*	6	152	164	565	263	284	291	291	42	293	293	293	38	38	38	38	263	284	291	291
		TPR	0.92%	23.3%	25.2%	86.8%	40.4%	43.6%	44.7%	44.7%	6.45%	45.0%	45.0%	45.0%	5.84%	5.84%	5.84%	5.84%	40.4%	43.6%	44.7%	44.7%
	C	*TP*	6	152	164	564	262	283	290	290	40	292	292	292	38	38	38	38	262	283	290	290
		TPR	0.92%	23.3%	25.2%	86.6%	40.2%	43.5%	44.5%	44.5%	6.14%	44.9%	44.9%	44.9%	5.84%	5.84%	5.84%	5.84%	40.2%	43.5%	44.5%	44.5%
	B	*FP*	0	7.8k	7.8k	9.8k	25.3k	28.3k	28.3k	28.3k	10	181	181	396	0	0	0	0	382	382	382	382
		FPR	0%	4.98%	4.98%	6.26%	16.1%	18.1%	18.1%	18.1%	≈0.0%	0.12%	0.12%	0.25%	0%	0%	0%	0%	0.24%	0.24%	0.24%	0.24%
	C	*FP*	0	16	16	31	8	9	9	9	0	0	0	1	0	0	0	0	1	1	1	1
		FPR	0%	0.01%	0.01%	0.02%	≈0.0%	≈0.0%	≈0.0%	≈0.0%	0%	0%	0%	≈0.0%	0%	0%	0%	0%	≈0.0%	≈0.0%	≈0.0%	≈0.0%
DS3	*B*	*TP*	157	157	169	580	273	298	305	305	42	306	306	308	38	38	38	38	273	298	305	305
		TPR	23.6%	23.6%	25.4%	87.2%	41.1%	44.8%	45.9%	45.9%	6.32%	46.0%	46.0%	46.3%	5.71%	5.71%	5.71%	5.71%	41.1%	44.8%	45.9%	45.9%
	C	*TP*	157	157	169	578	268	293	300	300	41	302	302	303	38	38	38	38	268	293	300	300
		TPR	23.6%	23.6%	25.4%	86.9%	40.3%	44.1%	45.1%	45.1%	6.17%	45.4%	45.4%	45.6%	5.71%	5.71%	5.71%	5.71%	40.3%	44.1%	45.1%	45.1%
	B	*FP*	16.8k	16.8k	16.8k	26.1k	148k	156k	159k	159k	10	191	191	750	0	0	0	0	710	730	730	730
		FPR	1.82%	1.82%	1.82%	2.84%	16.1%	16.9%	17.3%	17.3%	≈0.0%	0.02%	0.02%	0.08%	0%	0%	0%	0%	0.08%	0.08%	0.08%	0.08%
	C	*FP*	24	24	24	48	7	7	8	8	0	0	0	5	0	0	0	0	4	4	4	4
		FPR	≈0.0%	≈0.0%	≈0.0%	≈0.0%	≈0.0%	≈0.0%	≈0.0%	≈0.0%	0%	0%	0%	≈0.0%	0%	0%	0%	0%	≈0.0%	≈0.0%	≈0.0%	≈0.0%

(Section 5), existing PG solutions usually combine a basic pruning strategy (either a backward/forward search, or a path-based search [10]) with other complementary (e.g., anomaly [10] or time-based [3]) strategies. Therefore, we re-implement those strategies and combine them to form five different pruning approaches. Specifically, the first and second approaches perform a backward/forward search and check the event timings, while the second approach further utilizes anomaly scores to filter edges. The last three approaches search for paths connected to the POI and further utilize anomaly scores, time intervals, and both, respectively.

Table 2 shows the baseline result (i.e., applying each existing pruning approach on the original PG), and the result with CONTEXTS (i.e., using CONTEXTS's output as the input of each existing pruning approach). The top, middle, and bottom rows show the results on DS1 (size 10k), DS2 (size 160k), and DS3 (size 1M), respectively. The *Backward/Forward* column shows both the first (under threshold of "none") and second approach (with the anomaly threshold [10] ranging from 0.8 to 1 following [9]), and the next columns show the other three approaches. Overall, the results show that using CONTEXTS as a pre-processor does not significantly affect the TP/TPR values, but can significantly reduce the FP/FPR values, with FPRs close to zero for all approaches and their parameter settings. More importantly, while the number of FPs of existing approaches can reach 2k–159k, CONTEXTS can reduce these to less than 50 in all cases. This is important considering that human analysts typically cannot comprehend a PG with thousands of nodes and edges.

More specifically, (i) the baseline backward/forward approach (threshold "none") provides the highest TPRs for all dataset sizes, but also lead to high FPRs. With the help of CONTEXTS, the FPR is reduced to almost zero, while preserving the high TPR. (ii) The backward/forward with anomaly approach under threshold 0.8 can reduce the FPR values to similar levels as CONTEXTS for DS1 and DS2, but not for DS3. However, this also comes with an unacceptably lower TPR. (iii) The path-anomaly approach has low TPRs (less than 50%), and very high FPRs (with FPs ranging from 2.9k to 160k), which are again reduced to almost zero by CONTEXTS (FPs reduced to less than 10). (iv) The path-time approach (the third column) has similar TPRs and lower FPRs (under smaller time intervals), which are again reduced to almost zero (FPs less than 10) by CONTEXTS. (v) The last approach (combining path-based pruning with both time and anomaly) can reduce FPRs, but also has low TPRs (less than 50%), and CONTEXTS achieves similar reduction in FPRs.

4.4. Effectiveness and Optimal Parameters of the Correlation Approaches

This section answers RQ3 by evaluating the correlation approaches (Section 3.3) and identifying their optimal parameter(s) (i.e., K, N, N', and N'') for 10 selected CVEs. Among these, seven CVEs have static WPs (selected from the first 13 CVEs listed in Table 1) and three do not (selected from the last seven CVEs). The former seven CVEs are used to evaluate all correlation approaches, whereas the latter three are additionally used for StatDyn and CondExp (as they can handle CVEs with no static WPs).

K-Shortest Paths (KPath). Figure 13 (top) shows the results of applying the KPath approach. Overall, the KPath approach has a relatively low TPR for all the attacks, since it only focuses on connecting the informative points but does not consider relevant nodes close to (but not on) the paths. This is also reflected in the fact that, although increasing the value of K improves the TPR, the improvement eventually flattens, which shows the limitation of only considering nodes on the paths. On the other hand, the FPR stays relatively low for most attacks, which shows the KPath approach is indeed a promising starting point for identifying relevant PG nodes, as also confirmed in our user study (Section 4.6).

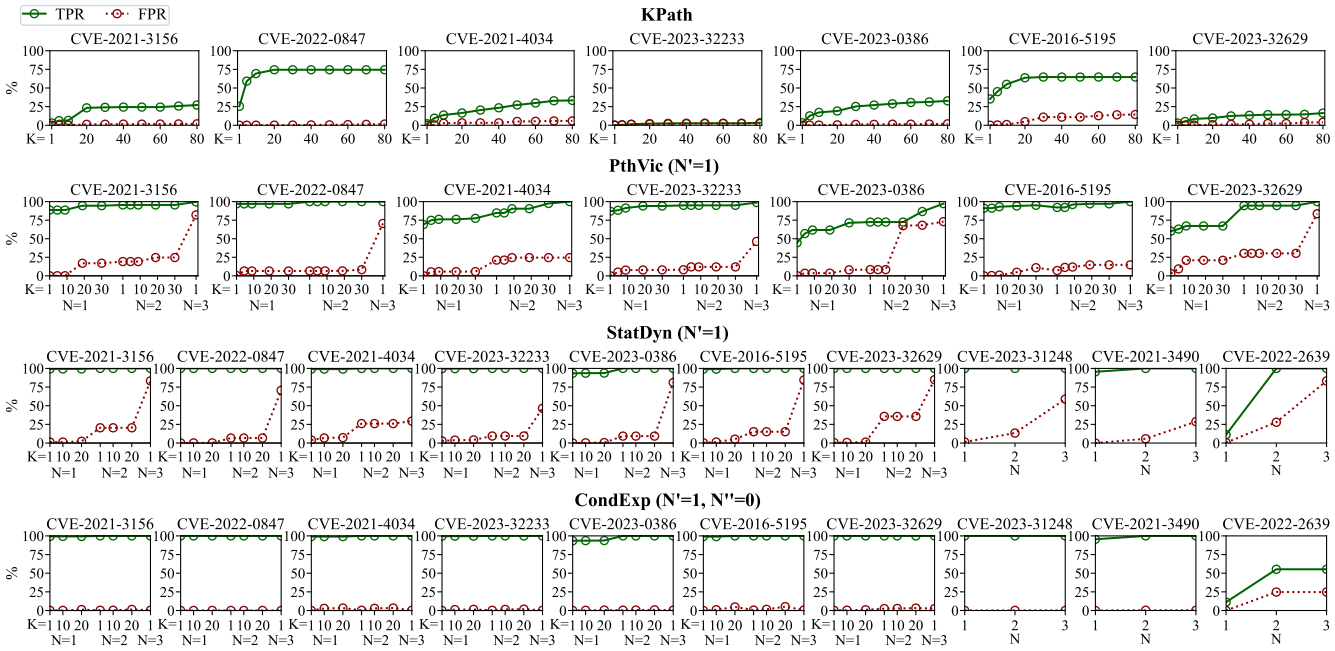

Figure 13: The effectiveness of KPath (top), PthVic (upper-middle), StatDyn (lower-middle), and CondExp (bottom).

Path-guided Vicinity (PthVic). Figure 13 (upper-middle) shows the results of PthVic for different combinations of N and K, with $N'=1$. Compared to KPath, the significantly higher TPR values of PthVic indicate the benefit of its additional feature, i.e., expanding from the shortest paths. On the other hand, the TPRs generally flatten after a certain combination of K and N, and the FPRs also start to increase at a certain point. Unlike KPath, PthVic can provide a high TPR and acceptable FPR for CVE-2023-32233 and CVE-2021-4034, although its results are still not satisfactory for CVE-2023-0386 (all three with complex exploit codes), which demonstrates the inherent limit of such graph-theoretic approaches.

Vicinity of Static and Dynamic WPs (StatDyn). Figure 13 (lower-middle) shows the results of the StatDyn approach for different values of N and K, with $N'=1$. The results are reported for all of the 10 attacks, since StatDyn can leverage dynamic WPs for the last three attacks (with no static WPs). Compared to PthVic, StatDyn achieves consistently higher TPR results (more than 93%) and lower FPRs (less than 5%) under $N=1$ (and $K=1$, when applicable) for nine out of ten attacks, which even outperforms PthVic under larger parameters ($N=2$, $K=1$). This clearly shows the benefit of additionally considering dynamic WPs in StatDyn. On the other hand, the FPRs generally grow fast for higher values of K and N, which indicates a dependency explosion issue. Overall, those results show that StatDyn can be effective even in the lack of static WPs, although the high FPRs introduced by dependency explosion remains a challenge.

Conditional Expansions (CondExp). Figure 13 (bottom) presents the results of the CondExp approach. The results show, for nine attacks, CondExp ($N''=0$) achieves similar TPRs as StatDyn, and significantly lower FPRs even under larger N values. This flat FPR curve clearly shows the benefit of limiting the expansion at boundary nodes in CondExp and allows us to choose a larger N to further boost TPR. For instance, CondExp under $N=2$ and $N=3$ can achieve 100% TPR with less than 0.6% FPR for six of the seven attacks with static WPs, and two attacks without static WPs (CVE-2023-31248 and CVE-2021-3490). For the more sophisticated attacks, such as (CVE-2022-2639), the best results of CondExp are obtained under $N''=1$ (under which CondExp becomes equivalent to StatDyn). Overall, those results show that CondExp is very effective in most cases, and only faces challenges when the lack of static WPs is paired with a complex attack scenario (e.g., CVE-2022-2639). This limitation is expected since the key idea behind CONTEXTS is to leverage external information (i.e., static WPs).

4.5. Scalability

To answer RQ4, this section evaluates the scalability of CONTEXTS on larger PGs with more irrelevant nodes. Specifically, we apply CONTEXTS (with Conditional Expansion) on 15 datasets, each of which contains PGs with increasing sizes, and evaluate both the time taken by CONTEXTS and the sizes of its outputs. All the PGs are generated based on the data collected from a virtual machine on which the same CVE (CVE-2022-0847, as shown in the second row of Table 1) is exploited under increasing workloads of normal applications (concurrent threads of Nginx, Flask, SQLite, and the Linux `stress` utility) and over time intervals of increasing lengths. The results are averaged over 100 iterations under similar conditions.

Figure 14 (left) shows the sizes of the CONTEXTS results (in terms of the number of edges) for various durations of data collection (reported in the colored boxes). While

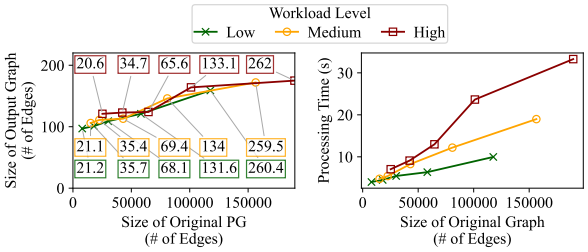

Figure 14: The sizes of the outputs (left) and the processing time (right) of CONTEXTS for PGs collected under different workloads and durations.

the size of the original PG grows from 6,000 to more than 180,000 edges, the size of the subgraph produced by CONTEXTS remains under 200 edges. This demonstrates the capability of CONTEXTS in pruning irrelevant nodes and edges from the PG, regardless of its size. The slight variation in the output sizes is due to a small number of FPs connected to shared files (mostly, Linux *locales*). Figure 14 (right) reports the time taken by CONTEXTS for processing those PGs, measured on a virtual machine with 64GB of memory and 16 vCPUs. The results show that CONTEXTS can process graphs with more than 180,000 edges in around 30s, and its execution time grows almost linearly with the size of the PG. This result largely depends on the delay of Neo4j, and can be improved using more efficient graph databases.

4.6. User Study

To answer RQ5, we conducted a user-based study[3] following standard practices [75]. During the study, 18 participants from both academia and industry were asked to identify the subgraph of a PG and the root cause related to a highlighted POI, with and without the help of CONTEXTS. All participants had a general background in security, with varying levels of knowledge and experience in provenance analysis and Linux kernel. Table 3 shows the distribution of participants, their levels of familiarity with those concepts, and the average agreement scores for each question.

Specifically, at the beginning of the study, the participants were given a PG generated for an exploit of CVE-2023-0386 (explained in Section 4.2.2). The complete PG has 3,425 nodes and 9,869 edges, with a highlighted alert as the POI (displayed in Neo4j Browser [49]). The participants were asked to perform the investigation tasks without, and with, CONTEXTS, using its different correlation approaches. To limit the duration of the study, we imposed time constraints for each task (10 minutes for Q1 and Q2, respectively, and five minutes for each remaining statement). For each task, we asked the participants to express their level of agreement (*Strongly Agree*, *Agree*, *Neutral*, *Disagree*, or *Strongly Disagree*) with our provided statements (the list of complete statements can be found in Appendix D). To quantify the results, we assign an integer between one and five to each level (five represents *Strongly Agree*).

3. This study has been approved by our university's Institution Review Board (IRB). We designed the questions and study materials following standard practices for usability questionnaires [75].

TABLE 3: Distribution of participants and their levels of of agreement. *VL*, *L*, *M*, *H*, and *VH* mean Very Low, Low, Medium, High, and Very High, respectively. The results in each cell are the average answers of participants with the same level of familiarity with the Linux kernel ($*/$) and with provenance analysis ($/*$), respectively.

	Industry (7)					**Academia (11)**				
Famil.	*VL*	*L*	*M*	*H*	*VH*	*VL*	*L*	*M*	*H*	*VH*
# of Partic.	1/0	1/2	2/2	1/1	2/2	0/2	1/3	3/3	6/2	1/1
Q1	5./-	4./4.5	4.5/3.5	3./5.	5./5.	-/2.5	5./4.33	4.67/4.67	3.67/4.5	4./4.
Q2	4./-	5./4.5	4.5/4.5	4./4.	5./5.	-/3.5	4./4.33	5./4.67	4./4.5	2./2.
Q3	5./-	5./5.	5./5.	5./5.	5./5.	-/3.5	5./5.	3.33/3.33	4./3.5	4./4.
Q4	5./-	5./5.	5./5.	5./5.	5./5.	-/4.	4./3.67	4.33/4.67	4.33/5.	5./5.
Q5	4./-	4./5.	5./4.5	5./4.	5./5.	-/4.5	5./5.	4.67/4.67	4.83/5.	5./5.
Q6	4./-	5./4.5	5./4.5	4./4.	4.5/5.	-/5.	5./5.	4./4.67	4.67/4.5	5./5.
Q7	5./-	5./5.	5./5.	5./5.	5./5.	-/5.	5./4.67	5./5.	4.83/5.	5./5.
Q8	3./-	4./4.5	4.5/4.5	5./3.	4.5/4.5	-/5.	5./4.33	4.33/4.67	4.83/5.	5./5.
Q9	5./-	4./4.	4.5/4.5	5./5.	4.5/5.	-/4.5	5./4.67	4./4.33	4.67/4.5	3./3.
Q10	5./-	5./4.5	4.5/5.	5./5.	5./5.	-/4.5	5./4.33	3.67/3.67	4.33/4.5	4./4.
Avg.	4.5/-	4.6/4.65	4.75/4.6	4.6/4.5	4.85/4.95	-/4.2	4.8/4.43	4.3/4.43	4.42/4.6	4.2/4.2

Results. Figure 15 presents the results of our user study. Overall, the large majority of participants agreed or strongly agreed with all the statements. In particular, when presented with the results of "Conditional Expansion of WPs" (Q7), all participants except one (who agreed) strongly agreed with the statement (that the approach) "Greatly reduces irrelevant nodes". Most participants (from both academia and industry) reported that they could not find the root cause of the attack without our solution (Q1), even when they were already given the attack scenario (Q2). Among the statements regarding different correlation approaches, K-Shortest Paths (Q3) showed the largest amount of disagreement (three disagreed), showcasing the limitations of this building block approach compared to more advanced ones (e.g., PthVic, StatDyn, and CondExp). Q1 and Q2 also showed some levels of disagreement, which we attribute to the time constraints imposed, since some participants believed they could have found the root cause if they were given more time. Among the statements asking about the amount of false positives (FPs) and false negatives (FNs), interestingly, the participants tended to agree more when asked about the FPs (Q5: 4.78, Q7: 4.95) than when asked about the FNs (Q4: 4.61, Q6: 4.55). This suggests that irrelevant nodes may have had more negative impact on the analysis than missed relevant nodes.

The last row of Table 3 shows the average level of agreement depending on the background (*industry* or *academia*) and level of familiarity with the two related concepts. The results show that participants from the industry generally answered more positively to all the statements than those from academia. We believe that industry professionals who were acquainted with those concepts (especially PGs) tended to be more sensitive to the tediousness of unassisted investigation tasks (Q1 and Q2), especially if these must happen very frequently (e.g., due to a large amount of false alerts). Among the specific results

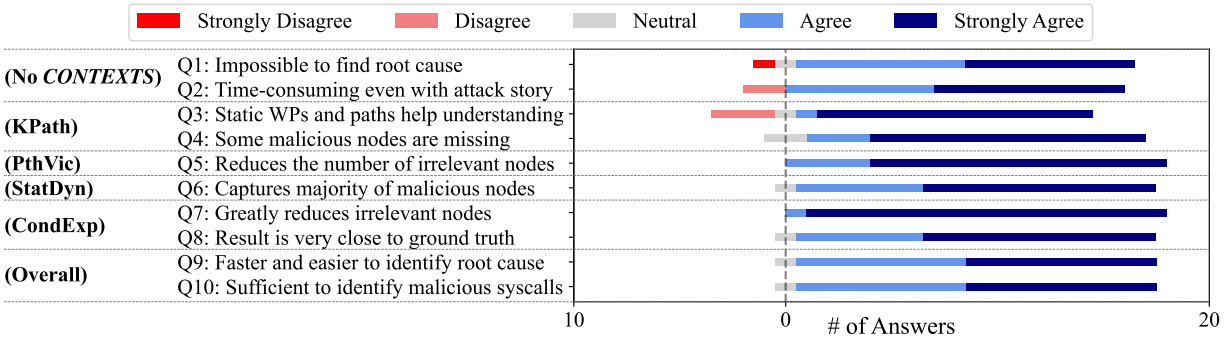

Figure 15: The (shortened) questions of the user study and the result distribution.

of each question, one outlier is that one participant from academia with *very high* familiarity answered 2 (*Disagree*) to Q2. This is due to the fact that our study did not allow participants to change their answers once submitted (this participant in fact changed his/her mind afterwards and believed the right answers should have been 5 instead of 2).

The participants also provided us with additional feedback during and after the study. First, the great majority of users who were not familiar with PG analysis reported feeling "overwhelmed" without using our solution (Q1 and Q2). Second, only one academic user (who was very familiar with Linux kernel vulnerabilities) was able to identify a major part of the subgraph within the time limit based on just the attack story, and still reported that s/he agreed with Q2. Third, overall, many users (in particular those who were familiar with Linux) showed interest in using our solution. For instance, one academic user reported that they "never thought about representing process and files this way" and was "amazed by how complex CVEs can be presented with such level of detail and clarity". Another participant showed interest in using our tool on a vulnerability they were leveraging for their own project.

5. Related Work

Provenance-based attack investigation has received extensive attention in the literature, as surveyed in [25]. To reduce the scope of analysis, there exist many PG pruning approaches. Specifically, Table 4 lists some popular approaches adopted by several state-of-the-art solutions. Nonetheless, most pruning approaches also come with their limitations. For instance, backward searching from the POI can prune some irrelevant nodes but cannot prevent dependency explosion [22]. Anomaly-based solutions can avoid dependency explosion (e.g., [10], [12], [13], [14], [15], [21]), while ensuring the completeness of training datasets can be difficult in practice [9]. Some pruning approaches (e.g., [4], [9]) rely on generic features such as node degrees, which may not capture the semantics of attacks. Some solutions [9], [10], [12] extract and merge paths while limiting their lengths, which may miss attack-related events that are farther away. Many solutions require multiple traversals over large graphs, which may imply high overhead [25]. While most of the forensic analysis solutions only consider

TABLE 4: Popular pruning strategies of existing solutions.

	CPR [22]	PrioTracker [12]	DEPIMPACT [9]	NODOZE [10]	ATLAS [20]	SPARSE [19]	Zebra [23]
Backward (BW) / Forward (FW)	✓/✓	✓/✓	✓/_				✓/_
Paths to Alert				✓	✓	✓	
Anomaly Score		✓	✓	✓			
Time	✓	✓	✓			✓	
Data flow		✓	✓			✓	✓
Node Degree		✓	✓	✓			
Merging Edges	✓		✓	✓	✓	✓	
Node Aggregation					✓		

the POI as a starting point, CONTEXTS leverages external information about the POI to further enhance the pruning. Therefore, it can work in tandem with existing solutions and enhance their results, as demonstrated in our experiments.

CONTEXTS can also work with some fine-grained provenance-based anomaly detection approaches [13], [14], [15] as well as threat hunting solutions [8], [16], [17], [18], [76]. The former category of solutions prune the PGs based on calculated anomaly scores, whose accuracy inherently depends on the completeness of benign activities (for training), and those approaches typically require post-pruning to further reduce the size of the results (e.g., KAIROS [14] applies the PG reduction techniques of [22]). Such approaches can potentially leverage CONTEXTS as either a pre- or post-processing step (by regarding the identified anomalous nodes as POIs). The latter category of solutions can also leverage CONTEXTS for further investigation of the POI subgraphs which they produce by matching general attack patterns with PGs.

AIQL [77] and SAQL [78] propose domain-specific query languages to detect known attack steps in PGs, which is similar to our query generation mechanism but with a different goal. Some provenance-based solutions also leverage IDS rules like CONTEXTS does, but for different reasons, e.g., continuously monitoring the PG for matching attacks or scoring alerts [4], [79]. In summary, as those solutions mostly focus on detection, their results may serve as an input (POI) for CONTEXTS to perform further investigation. Moreover, unlike those works, CONTEXTS integrates multiple sources (e.g., IDS rules and CVEs) and may cover out-of-order attack steps.

6. Conclusion

We have presented CONTEXTS, a solution that leveraged external information about the POI to identify relevant PG nodes/edges and reduce the scope of attack investigation. We described the methodology and implementation to extract alert context information from external data sources, map them to the PG to generate waypoints, and correlate those waypoints and the POI to obtain the output subgraph. We evaluated CONTEXTS both as a standalone solution and as a pre-processor of other pruning solutions, and our results demonstrated its effectiveness in both cases. Finally, our user study results confirmed its real-world applicability.

Limitations and Extensions. First, we have focused on using Falco [42] for detection, SPADE [47] for PG generation, and Falco rules and CVEs as the main sources of external information in our implementation. Our approach can potentially be extended through integration with (i) other IDSes with the initial effort of adapting to their reported alerts and detection rules, (ii) other PG generation tools by rewriting queries based on their PG formats, (iii) other sources of external information (e.g., [32], [34], [35]) through developing new extraction scripts. Such extensions can also expand the scope of attacks which CONTEXTS can handle, e.g., in-memory object attacks (which are not supported by the current PG capturing tool used by CONTEXTS). Second, although we have focused on kernel-level CVEs, our approach can potentially be extended to handle APTs. For instance, CONTEXTS can generate waypoints for APTs based on external information extracted from data sources like the MITRE ATT&CK framework [80] and the CSKG [36], [37], [38] (which allows linking each MITRE technique to related CVEs and their details). Third, the extraction of alert context information is currently based on customized scripts, and applying natural language processing (NLP) techniques for this purpose is an interesting future direction. Fourth, while CONTEXTS is designed for assisting human analysts in forensics investigations, addressing more dynamic applications such as real-time attack response is a potential future direction, which would require the capability of incremental knowledge base construction and maintenance, and improving the efficiency of provenance graph collection and processing techniques. Fifth, while CONTEXTS shows superior performance when static WPs are available, improving its performance without static WPs is another future direction. Sixth, while employing external information is the key strength of CONTEXTS, it also means a unique reliance on the quality and timeliness of such information. Finally, while the waypoint-pruning step of CONTEXTS (Section 3.2) can eliminate most unrelated-to-POI waypoints, and we have not encountered a case where multiple CVE matching cause FPs, real deployments may produce significantly larger PGs to increase that chance.

Acknowledgment

We thank the anonymous shepherd and reviewers for their valuable comments. This work was supported by the Natural Sciences and Engineering Research Council of Canada and Ericsson Canada under the Industrial Research Chair in SDN/NFV Security and the Discovery Grant N01035, and by the Canada Foundation for Innovation under JELF Project 3859.

References

[1] "Strace." https://linux.die.net/man/1/strace, 2022.

[2] "Sysdig." https://sysdig.com/, 2023.

[3] S. T. King and P. M. Chen, "Backtracking Intrusions." in *ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[4] W. U. Hassan, A. Bates, and D. Marino, "Tactical Provenance Analysis for Endpoint Detection and Response Systems." in *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[5] H. Irshad, G. Ciocarlie, A. Gehani, V. Yegneswaran, K. H. Lee, J. Patel, S. Jha, Y. Kwon, D. Xu, and X. Zhang, "Trace: Enterprise-wide Provenance Tracking for Real-time APT Detection." *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 16, pp. 4363–4376, 2021.

[6] F. Yang, J. Xu, C. Xiong, Z. Li, and K. Zhang, "PROGRAPHER: An Anomaly Detection System based on Provenance Graph Embedding." in *USENIX Security Symposium*, 2023.

[7] Z. Li, Q. A. Chen, R. Yang, Y. Chen, and W. Ruan, "Threat Detection and Investigation with System-level Provenance Graphs: A Survey." *Computers & Security*, vol. 106, p. 102282, 2021.

[8] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Poirot: Aligning Attack Behavior with Kernel Audit Records for Cyber Threat Hunting." in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019.

[9] P. Fang, P. Gao, C. Liu, E. Ayday, K. Jee, T. Wang, Y. F. Ye, Z. Liu, and X. Xiao, "Back-Propagating System Dependency Impact for Attack Investigation." in *USENIX Security Symposium*, 2022.

[10] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage." in *Network and Distributed Systems Security Symposium (NDSS)*, 2019.

[11] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. Ciocarlie *et al.*, "MCI: Modeling-based Causality Inference in Audit Logging for Attack Investigation." in *Network and Distributed Systems Security Symposium (NDSS)*, 2018.

[12] Y. Liu, M. Zhang, D. Li, K. Jee, Z. Li, Z. Wu, J. Rhee, and P. Mittal, "Towards a Timely Causality Analysis for Enterprise Security." in *Network and Distributed Systems Security Symposium (NDSS)*, 2018.

[13] A. Goyal, G. Wang, and A. Bates, "R-CAID: Embedding Root Cause Analysis within Provenance-based Intrusion Detection." in *IEEE Symposium on Security and Privacy (S&P)*, 2024.

[14] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, "KAIROS: Practical Intrusion Detection and Investigation Using Whole-system Provenance," in *IEEE Symposium on Security and Privacy (S&P)*, 2024, pp. 3533–3551.

[15] M. U. Rehman, H. Ahmadi, and W. U. Hassan, "FLASH: A Comprehensive Approach to Intrusion Detection via Provenance Graph Representation Learning." in *IEEE Symposium on Security and Privacy (S&P)*, 2024, pp. 139–139.

[16] P. Gao, F. Shao, X. Liu, X. Xiao, Z. Qin, F. Xu, P. Mittal, S. R. Kulkarni, and D. Song, "Enabling Efficient Cyber Threat Hunting with Cyber Threat Intelligence." in *IEEE International Conference on Data Engineering (ICDE)*, 2021, pp. 193–204.

[17] E. Altinisik, F. Deniz, and H. T. Sencar, "ProvG-Searcher: A Graph Representation Learning Approach for Efficient Provenance Graph Search." in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2023.

[18] K. Satvat, R. Gjomemo, and V. Venkatakrishnan, "Extractor: Extracting Attack Behavior from Threat Reports." in *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2021.

[19] J. Ying, T. Zhu, W. Cheng, Q. Yuan, M. Ma, C. Xiong, T. Chen, M. Lv, and Y. Chen, "SPARSE: Semantic Tracking and Path Analysis for Attack Investigation in Real-time." *arXiv preprint arXiv:2405.02629*, 2024.

[20] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu, "ATLAS: A Sequence-based Learning Approach for Attack Investigation." in *USENIX Security Symposium*, 2021.

[21] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. A. Gunter *et al.*, "You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis." in *Network and Distributed Systems Security Symposium (NDSS)*, 2020.

[22] Z. Xu, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang, "High Fidelity Data Reduction for Big Data Security Dependency Analyses." in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016, pp. 504–516.

[23] X. Yang, H. Liu, Z. Wang, and P. Gao, "Zebra: Deeply Integrating System-level Provenance Search and Tracking for Efficient Attack Investigation." *arXiv preprint arXiv:2211.05403*, 2022.

[24] Y. Tang, D. Li, Z. Li, M. Zhang, K. Jee, X. Xiao, Z. Wu, J. Rhee, F. Xu, and Q. Li, "NodeMerge: Template Based Efficient Data Reduction for Big-data Causality Analysis." in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018, pp. 1324–1337.

[25] M. A. Inam, Y. Chen, A. Goyal, J. Liu, J. Mink, N. Michael, S. Gaur, A. Bates, and W. U. Hassan, "SoK: History is a Vast Early Warning System: Auditing the Provenance of System Intrusions." in *IEEE Symposium on Security and Privacy (S&P)*, 2023, pp. 2620–2638.

[26] M. Abubakar, A. Ahmad, P. Fonseca, and D. Xu, "SHARD: Fine-Grained Kernel Specialization with Context-Aware Hardening." in *USENIX Security Symposium*, 2021, pp. 2435–2452.

[27] H. Kermabon-Bobinnec, Y. Jarraya, L. Wang, S. Majumdar, and M. Pourzandi, "Phoenix: Surviving Unpatched Vulnerabilities via Accurate and Efficient Filtering of Syscall Sequences." in *Network and Distributed Systems Security Symposium (NDSS)*, 2024.

[28] NIST, "CVE-2022-0847 Detail." https://nvd.nist.gov/vuln/detail/CVE-2022-0847.

[29] CVEDetails, "CVE-2022-0847 detail." https://www.cvedetails.com/cve/CVE-2022-0847/.

[30] J. Avery, "CVE-2022-0847 (Dirty Pipe): Linux Local Privilege Escalation," https://sysdig.com/blog/cve-2022-0847-dirty-pipe-sysdig.

[31] Sysdig, "Falco rules supported fields." https://falco.org/docs/reference/rules/supported-fields/.

[32] SigmaHQ, "Sigma Rules." https://sigmahq.io/docs/basics/rules.html.

[33] MITRE, "Common Vulnerabilities and Exposures (CVE)." https://cve.mitre.org.

[34] ——, "Common Weakness Enumeration (CWE)." [Online]. Available: http://cwe.mitre.org

[35] ——, "Common Attack Pattern Enumeration and Classification (CAPEC)." http://capec.mitre.org.

[36] L. F. Sikos, "Cybersecurity Knowledge Graphs." *Knowledge and Information Systems*, vol. 65, no. 9, pp. 3511–3531, 2023.

[37] K. Kurniawan, A. Ekelhart, and E. Kiesling, "An ATT&CK-KG for Linking Cybersecurity Attacks to Adversary Tactics and Techniques." in *International Semantic Web Conference (ISWC)*, 2021.

[38] E. Kiesling, A. Ekelhart, K. Kurniawan, and F. Ekaputra, "The SEPSES Knowledge Graph: an Integrated Resource for Cybersecurity." in *International Semantic Web Conference*, 2019.

[39] CVEDetails, "CWE-665 - Improper Initialization." [Online]. Available: https://www.cvedetails.com/cwe-details/665/Improper-Initialization.html

[40] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime Provenance-Based Detector for Advanced Persistent Threats." in *Network and Distributed Systems Security Symposium (NDSS)*, 2020.

[41] T. P. G. D. Group, "PostgreSQL: The World's Most Advanced Open Source Relational Database." https://www.postgresql.org/.

[42] Sysdig, "Falco: Detect Security Threats in Real Time." https://falco.org/.

[43] National Institute of Standards and Technology (NIST), "National Vulnerability Database (NVD)," https://nvd.nist.gov/, 2024.

[44] "spaCy: Industrial-strength Natural Language Processing in Python." 2020.

[45] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. Eyers, M. Seltzer, and J. Bacon, "Practical Whole-system Provenance Capture." in *Symposium on Cloud Computing (SoCC)*, 2017.

[46] D. J. Pohly, S. McLaughlin, P. McDaniel, and K. Butler, "Hi-Fi: Collecting High-fidelity Whole-system Provenance." in *Annual Computer Security Applications Conference (ACSAC)*, 2012.

[47] A. Gehani and D. Tariq, "SPADE: Support for Provenance Auditing in Distributed Environments." in *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, 2012.

[48] Grubb, Steve, "Linux Auditd." https://linux.die.net/man/8/auditd.

[49] Neo4j, "Neo4j Graph Platform." https://neo4j.com/.

[50] Gehani, Ashish, "Audit provenance," https://github.com/ashish-gehani/SPADE/wiki/Audit-provenance.

[51] Neo4j, "Cypher Query Language." https://neo4j.com/docs/cypher-manual/current/introduction/.

[52] ——, "Awesome Procedures On Cypher (APOC)," https://neo4j.com/labs/apoc/4.1/overview/apoc.path/apoc.path.expandConfig/.

[53] D. Eppstein, "Finding the K Shortest Paths." *Journal on Computing*, vol. 28, no. 2, pp. 652–673, 1998.

[54] The Linux Documentation Project (TLDP), "Program Library "How To": Shared Libraries," 2024. [Online]. Available: https://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html

[55] Neo4j, "Yen's Shortest Path Algorithm." https://neo4j.com/docs/graph-data-science/current/algorithms/yens/.

[56] ——, "The Neo4j Graph Data Science Library." [Online]. Available: https://neo4j.com/docs/graph-data-science/current/

[57] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion Detection using Sequences of System Calls." *Journal of Computer Security*, vol. 6, no. 3, pp. 151–180, 1998.

[58] D. Mutz, W. Robertson, G. Vigna, and R. Kemmerer, "Exploiting Execution Context for the Detection of Anomalous System Calls." in *Recent Advances in Intrusion Detection (RAID)*, 2007.

[59] The Kernel Development Community, "Seccomp BPF." 2022. [Online]. Available: https://www.kernel.org/doc/html/v5.10/userspace-api/seccomp_filter.html

[60] W. Blair, F. Araujo, T. Taylor, and J. Jang, "Automated Synthesis of Effect Graph Policies for Microservice-Aware Stateful System Call Specialization." in *IEEE Symposium on Security and Privacy (S&P)*, 2023, pp. 64–64.

[61] J. Byrnes, T. Hoang, N. N. Mehta, and Y. Cheng, "A Modern Implementation of System Call Sequence-based Host-based Intrusion Detection Systems." in *IEEE TPS-ISA*, 2020, pp. 218–225.

[62] Y. Jeon, J. Rhee, C. H. Kim, Z. Li, M. Payer, B. Lee, and Z. Wu, "PoLPer: Process-aware Restriction of Over-privileged Setuid Calls in Legacy Applications." in *ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2019.

[63] A. Jones and S. Li, "Temporal Signatures for Intrusion Detection." in *Annual Computer Security Applications Conference (ACSAC)*, 2001, pp. 252–261.

[64] W. Lee, S. J. Stolfo, and K. W. Mok, "A Data Mining Framework for Building Intrusion Detection Models." in *IEEE Symposium on Security and Privacy (S&P)*, 1999, pp. 120–132.

[65] Y. Yamamoto and S. Yamaguchi, "Defense Mechanism to Generate IPS Rules from Honeypot Logs and Its Application to Log4Shell Attack and Its Variants." *Electronics*, vol. 12, no. 14, p. 3177, 2023.

[66] C. Kreibich and J. Crowcroft, "Honeycomb: Creating Intrusion Detection Signatures using Honeypots." *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 1, p. 51–56, 2004.

[67] S. Chierici, "CVE-2022-0492: Privilege Escalation Vulnerability Causing Container Escape." https://sysdig.com/blog/detecting-mitigating-cve-2022-0492-sysdig.

[68] S. Mohammadi, H. Kermabon-Bobinnec, A. Tabiban, L. Wang, T. Navarro Múnera, and Y. Jarraya, "A Dataset of Kernel Exploits Represented as Provenance Graphs," https://doi.org/10.5281/zenodo.15200285.

[69] Torrey Jacob, "Transparent Computing Engagement 5 Data Release," https://github.com/darpa-i2o/Transparent-Computing, 2020.

[70] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Crimson Unicorn's Github Repo," https://github.com/crimson-unicorn, 2024.

[71] Verton Robin, "CVE-2016-5195 (DirtyCoW): Local Root PoC." https://gist.github.com/rverton/e9d4ff65d703a9084e85fa9df083c679.

[72] Ahmed Alexis, "CVE-2022-0847 (Dirty Pipe): Exploits." https://github.com/AlexisAhmed/CVE-2022-0847-DirtyPipe-Exploits.

[73] Datadog, "CVE-2023-0386: OverlayFS." https://github.com/DataDog/security-labs-pocs/tree/main/proof-of-concept-exploits/overlayfs-cve-2023-0386.

[74] HashiCorp, "Vagrant Boxes." https://app.vagrantup.com/boxes.

[75] A. Assila, H. Ezzedine *et al.*, "Standardized Usability Questionnaires: Features and Quality Focus." *Electronic Journal of Computer Science and Information Technology*, vol. 6, no. 1, 2016.

[76] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "DeepHunter: A Coverage-guided Fuzz Testing Framework for Deep Neural Networks." in *ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 146–157.

[77] P. Gao, X. Xiao, Z. Li, F. Xu, S. R. Kulkarni, and P. Mittal, "AIQL: Enabling Efficient Attack Investigation from System Monitoring Data." in *USENIX Annual Technical Conference (USENIX ATC)*, 2018.

[78] P. Gao, X. Xiao, D. Li, Z. Li, K. Jee, Z. Wu, C. H. Kim, S. R. Kulkarni, and P. Mittal, "SAQL: A Stream-based Query System for Real-Time Abnormal System Behavior Detection." in *USENIX Security Symposium*, 2018.

[79] K. Kurniawan, A. Ekelhart, E. Kiesling, G. Quirchmayr, and A. M. Tjoa, "KRYSTAL: Knowledge Graph-based Framework for Tactical ttack Discovery in Audit Data." *Computers & Security*, vol. 121, p. 102828, 2022.

[80] MITRE, "MITRE ATT&CK Framework," https://attack.mitre.org/.

[81] NIST, "NIST CVE Database." https://nvd.nist.gov/vuln.

[82] MITRE, "MITRE CVE Database," https://cve.mitre.org.

Appendix A. Algorithms

K-Shortest Paths. Algorithm 1 details the K-Shortest Paths (KPath) approach employed to connect different informative points, as described in Section 3.3. Similar to some existing solutions [9], [22], this approach merges all parallel edges between each pair of nodes (line 2), as their presence can lead to repetitive node sets even when the value of K increases. After identifying the shortest paths, the original parallel edges are revisited for further investigation (line 11), thereby avoiding any loss of information.

Algorithm 1 K-Shortest Paths Approach

Inputs:
$G \leftarrow$ Provenance graph
IP-Set $\leftarrow \{POI, WP_1, \ldots, WP_n\}$ // The set of informative points
Output:
G'': Output Subgraph // G'' = G''-Nodes $\cup$ G''-Edges

1: **function** K-SHORTEST PATHS(*IP-Set*,K)
2: $g' \leftarrow$ Merge-Parallel-Edges(G)
3: G''-Nodes $\leftarrow \{\}$
4: G''-Edges $\leftarrow \{\}$
5: **for all** (IPt_i, IPt_j) **in** (*IP-Set* $\times$ *IP-Set*), $i \neq j$ **do**
6: $P_i \leftarrow$ ProcessNodes(IPt_i)
7: $P_j \leftarrow$ ProcessNodes(IPt_j) // If an IPt does not include a process node, this function returns the process that is directly connected to it.
8: **if** $(P_i.time - P_j.time = \min(|P'_i.time - P'_j.time|), \forall P'_i \in IPt_i, \forall P'_j \in IPt_j)$ **then**
9: Paths $\leftarrow$ KShortest(g', P_i, P_j, K)
10: G''-Nodes $\leftarrow G''$-Nodes $\bigcup$ Nodes(Paths)
11: G''-Edges $\leftarrow G''$-Edges $\bigcup$ Unmerge(Edges(Paths)) // Revisiting the parallel edges corresponding to each edge of Paths.
12: **end if**
13: **end for**
14: **end function**

Path-Guided Vicinity. Algorithm 2 details the Path-Guided Vicinity (PthVic) approach discussed in Section 3.3. The approach initializes the output graph with N-hop vicinity of informative points and the K-Shortest Paths between them (lines 2–4). Then, it finds the nodes that are both within an N-hop vicinity and on shortest paths (lines 5–7). These are expanded with N' additional hops (using Vic(node,N') defined in Equation 1), and added to the output graph (line 8).

Algorithm 2 Path-Guided Vicinity Approach

Inputs and **Output** are as defined in Algorithm 1.

1: **function** PATH-GUIDED VICINITY(*IP-Set* K, N, N')
2: $G_1 \leftarrow \bigcup N$-hop(IPt), $\forall IPt \in$ *IP-Set* // Equation 1
3: $G_2 \leftarrow$ K-SHORTEST PATHS(*IP-Set*, K) // Algorithm 1
4: $G'' \leftarrow G_1 \bigcup G_2$
5: NVic-nodes $\leftarrow$ Nodes(G_1)
6: Path-nodes $\leftarrow$ Nodes(G_2)
7: **for all** node **in** NVic-nodes $\bigcap$ Path-nodes **do**
8: $G'' \leftarrow G'' \bigcup$ Vic(node,N') // Equation 1
9: **end for**
10: **end function**

Vicinity of Static and Dynamic WPs. Algorithm 3 details the Vicinity of Static and Dynamic WPs (StatDyn) approach, as described in Section 3.3. The approach begins by retrieving the process nodes in informative points and their immediate parent and child processes, i.e., using the *Pid* and *PPid* properties in each process node (lines 3–5). It continues by iteratively acquiring all parents of parent nodes (lines 6–9) and all children of child nodes (lines 10–13) until no further parent or child can be found. The output is the combination of the PthVic result (line 15) and the vicinity of the obtained dynamic WPs (lines 16, 17).

Appendix B. Overview of the Exploited CVEs

Table 5 shows an overview of the real attacks implemented for testing CONTEXTS.

Algorithm 3 Vicinity of Static and Dynamic WPs Approach

Inputs and **Output** are as defined in Algorithm 1.

function STAT-DYN-VICINITY(*IP-Set*, K, N, N')
  Parent-Set $\leftarrow \{\}$, Child-Set $\leftarrow \{\}$
  IPt-Processes $\leftarrow \bigcup$ ProcessNodes(*IPt*), $\forall$*IPt* $\in$ *IP-Set*
  Temp-Parent-Set $\leftarrow$ GET-PARENTS(IPt-Processes)
  Temp-Child-Set $\leftarrow$ GET-CHILDREN(IPt-Processes) // *P* is *C*'s parent iff *C*.PPid= *P*.Pid.
  while Temp-Parent-Set $\neq \{\}$ **do** // Retrieving Ancestors
    Parent-Set $\leftarrow$ Parent-Set $\bigcup$ Temp-Parent-Set
    Temp-Parent-Set $\leftarrow$ GET-PARENTS(Temp-Parent-Set)
  end while
  while Temp-Child-Set $\neq \{\}$ **do** // Retrieving Descendants
    Child-Set $\leftarrow$ Child-Set $\bigcup$ Temp-Child-Set
    Temp-Child-Set $\leftarrow$ GET-CHILDREN(Temp-Child-Set)
  end while
  Dynamic-WPs $\leftarrow$ Child-Set $\bigcup$ Parent-Set
  $G'' \leftarrow$ PATH-GUIDED VICINITY(*IP-Set*, K, N, N') // Algorithm 2
  for all node **in** Dynamic-WPs **do**
    $G'' \leftarrow G'' \bigcup$ Vic(node,N') // Equation 1
  end for
end function

Appendix C. Exploit Code Reference Subgraph Extraction

To extract the reference subgraphs mentioned in Section 4.1, these steps are followed: (1) We apply our prior knowledge of the directory where the exploit code is stored to generate queries for identifying the process nodes related to the exploit code by obtaining the ones that have the same working directory (*cwd*) as the exploit code's and iteratively expanding them with all their progeny processes. (2) We manually validate the obtained processes by comparing them with the exploit code. (3) We execute queries to obtain the interactions between exploit-related processes, where we also consider the specified time window constraint (i.e., the creation time of the edges must be after the initialization of the exploit). (4) To validate the extracted reference subgraph, we compare its system calls with those recorded by tracing tools, such as Strace [1] or Sysdig [2], during the exploit code execution. The comparison shows a near-perfect match between the reference subgraph and the trace of the exploit code.

Appendix D. User Study Statements

Table 6 lists the full statements used in our user study, as well as the average agreement level of participants.

TABLE 5: Overview of the exploited CVEs in our dataset preparation, and the corresponding experiment setup. ↑ and ↓ denote that the kernel must be upgraded or downgraded from the default version, respectively. Note (1)/(2) indicate different exploitations of the same CVE.

CVE	Description [81], [82]	OS version *Box version*	Kernel version
2021-3156	Heap-based buffer overflow in the *sudo* utility	bento/ubuntu-20.04 *v202206.03.0*	v5.4.0-110-generic
2022-0847(1) *setuid bit*	Incorrect handling of Unix pipes	bento/ubuntu-20.04 *v202309.09.0*	v5.8.0-23-generic ↑
2022-0847(2) */etc/passwd*	Variant of DirtyPipe directly modifying `/etc/passwd`	bento/ubuntu-20.04 *v202309.09.0*	v5.8.0-23-generic ↑
2021-4034	Local privilege escalation in the *pkexec* utility	bento/ubuntu-20.04 *v202105.25.0*	v5.4.0-73-generic
2024-48990	Privilege escalation in the *needrestart* utility	generic/ubuntu-22.04 *v4.0.0*	5.15.0-30-generic
2021-44228 & 2022-0847	Chaining of Log4Shell (RCE) followed by DirtyPipe (privilege escalation)	bento/ubuntu-20.04 *v202309.09.0*	v5.8.0-23-generic ↑
2023-32233	Anonymous sets mishandle in *nftables*	generic/ubuntu-23.04 *v4.3.12*	v6.2.0-20-generic ↓
2016-6516	Double fetch vulnerability in the `ioctl` system call	bento/ubuntu-16.04 *v201801.02.0*	v4.6.1-generic ↑
2021-22555(1) *regular*	Heap memory corruption in *netfilter/x_tables*	ubuntu/focal64 *v20221003.0.0*	5.8.0-48-generic ↑
2023-0386	Unauthorized execution of `setuid` bit files in OverlayFS	generic/ubuntu-22.04 *v4.0.0*	v5.15.0-30-generic
2016-5195	Race condition during Copy-on-Write	bento/ubuntu-16.04 *v201801.02.0*	v4.4.0-21-generic ↓
2023-32629	Local privilege escalation in Ubuntu's OverlayFS	bento/ubuntu-22.10 *202303.13.0*	v5.19.0-35-generic
2023-22809	Privilege escalation in the *sudoedit* utility	bento/ubuntu-22.04 *v4.2.0*	5.15.0-92-generic
2023-31248	Use-after-free local privilege escalation in *nftables*	generic/ubuntu-23.04 *v4.3.0*	v6.20.0-20-generic ↓
2021-3490	Out-of-bounds reads and writes in eBPF ALU32	generic/ubuntu-20.04 *v3.5.0*	v5.8.0-48-generic ↑
2021-22555(2) *pipe*	Similar to above but exploit using pipe primitives	ubuntu/focal64 *v20221003.0.0*	5.8.0-48-generic ↑
2022-34918	Privilege escalation due to buffer overflow in *netfilter*	ubuntu/jammy64 *v20211020.0.0*	v5.15.0-39-generic ↓
2022-2639	Out-of-bounds write access in Open vSwitch	generic/ubuntu-20.04 *v4.3.0*	v5.13.0-37-generic ↑
2021-4154	Use-after-free in *cgroups* using the `fsconfig` system call	generic/ubuntu-20.04 *v4.3.0*	v5.8.0-48-generic ↑
2022-2588	Use-after-free in *cls_route* filter, exploited using DirtyCred	ubuntu/focal64 *v20221202.0.1*	v5.0.10-generic ↓

TABLE 6: User Study Statements. To quantify the results, we convert participants' agreement level to scores between one and five (score five represents *Strongly agree*).

Category	Statement	Code	Score
No CONTEXTS	Given the Provenance Graph and the POI, it is infeasible to find the root cause events related to the attack.	Q1	4.22
	Knowing the attack story, it is very time-consuming to find the root cause in the provenance graph.	Q2	4.28
K-Shortest Paths	Having the annotated suspicious activities (static WPs) and the paths connecting them made understanding the attack story easier.	Q3	4.33
	Some of the malicious nodes are missing in the provided result.	Q4	4.61
Path-guided Vicinity	This approach reduces the number of irrelevant nodes (w.r.t. K-Shortest Paths).	Q5	4.78
Vicinity of Static & Dynamic WPs	The graph generated through the ancestors of suspicious activities captures the great majority of malicious nodes w.r.t. the previous approaches.	Q6	4.55
Conditional Expansion of WPs	Using conditional expansion drastically reduces the irrelevant nodes.	Q7	4.95
	The result is very close to the ground truth.	Q8	4.56
RCA with CONTEXTS	Investigating the CONTEXTS result for identifying the sequence of root cause activities is much faster and easier.	Q9	4.44
	The provided result is sufficient for identifying the sequence of malicious system calls, eliminating the need to investigate other parts of the provenance graph.	Q10	4.44

Appendix E.
Meta-Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

E.1. Summary

The paper introduces CONTEXTS, a novel approach to cyber attack investigation that leverages a rich set of auxiliary information to pre-process provenance graphs and enhance investigation efforts. These external contexts (e.g., context given in the alerting IDS rules, *man-pages* around relevant system calls, etc.) provide more relevant information on identified attacks, allowing CONTEXTS to significantly reduce the size of provenance graph.

E.2. Scientific Contributions

- Creates a New Tool to Enable Future Science
- Provides a Valuable Step Forward in an Established Field

E.3. Reasons for Acceptance

1) This paper creates a new tool to enable future science, by introducing a new methodology that allows existing approaches to be augmented with external information to assist with pruning efforts. This provides an opportunity for future work to integrate even richer sources of information to help with pruning.
2) The paper provides a valuable step forward in an established field, by showing that accurate attack investigation requires more than just localized information collected from standard audit tools. The results show that by doing so it is possible to significantly reduce false positives, which has been a longstanding problem.

E.4. Noteworthy Concerns

1) The idea of leveraging CVE descriptions is promising, but its execution could be improved by analyzing the actual patched code for each CVE and extracting the key features from the patch.
2) CONTEXTS depends on accurate, up-to-date external information sources, which makes it vulnerable to issues with data quality or coverage. The paper does not explain if this is a limitation shared by other pruning approaches, or provide discussion that would help users pick the right tool per their needs.