Explaining Attacks using Causal Logs

1st Viet Huy Ha
SAMOVAR, Télécom SudParis
Institut Polytechnique de Paris
Palaiseau, France
viet-huy.ha@telecom-sudparis.eu

2nd Vincent Gauthier
SAMOVAR, Télécom SudParis
Institut Polytechnique de Paris
Palaiseau, France
vincent.gauthier@telecom-sudparis.eu

3rd Eric Totel
SAMOVAR, Télécom SudParis
Institut Polytechnique de Paris
Palaiseau, France
eric.totel@telecom-sudparis.eu

***Abstract*—In order to detect and mitigate cybersecurity threats, analysts usually use Security Information and Event Management (SIEM) systems as an important tool for analyzing large volumes of heterogeneous log data. However, the lack of contextual information and the inherent complexity of multi-step attack scenarios pose significant challenges in correlating events and explaining sophisticated attacks. To address these limitations, this paper proposes a novel approach to generating causal logs using eBPF (extended Berkeley Packet Filter), a powerful and efficient technology for observing kernel-level events. By leveraging eBPF, we produce logs that are highly relevant and explicitly linked to causal chains of events, enabling improved attack explanations. We assess the effectiveness of our approach by applying it to scenarios like SQL injection attacks, showcasing its capability to uncover causal pathways and support forensic investigations. The findings highlight that the proposed method improves the contextual relevance and clarity of logs, offering a stronger foundation for comprehending and addressing intricate cybersecurity challenges.**

***Index Terms*—Causal Logs, Causality Graphs, Intrusion Detection, Attack Explanation**

I. Introduction

In the rapidly evolving landscape of cybersecurity, organizations face increasing challenges in detecting and mitigating complex threats. Security Information and Event Management Systems (SIEMs) play a crucial role in this defense strategy by aggregating and analyzing large volumes of log data from various sources. Nevertheless, the diversity in the format, structure, and semantic content of these logs usually creates barriers to effective analysis. This heterogeneity, coupled with the high volume of data and inherent noise, makes it difficult to establish meaningful correlations between events and identify the root causes of sophisticated, multi-step attack scenarios.

The availability of existing SIEM solutions to provide sufficient contextual information and causal insights limits their effectiveness in handling advanced threats. Current approaches often focus on correlation-based techniques, which struggle to explain the progression of attacks across multiple stages. This lack of interpretability hampers the ability to trace and mitigate attacks effectively.

To address these challenges, we propose a novel framework for generating causal logs, leveraging the extended Berkeley Packet Filter (eBPF) technology. By observing kernel-level events, our method produces logs enriched with contextual relevance and explicit causal relationships, enabling a deeper understanding of attack mechanisms. This approach not only enhances the interpretability of log data but also provides a foundation for more effective forensic analysis and threat mitigation strategies.

II. State of the art

A. Causal Logs

Causal logs are referenced in studies [4] [2], but they are not specifically defined. These studies focus on mining causality within conventional log data, requiring inference of causal relationships. However, they do not propose a method for generating Causal logs.

B. Causal Dependency

Causal dependency is a critical concept in distributed systems, underpinning the understanding of how events influence one another. Causality ensures that the sequence of operations adheres to the logical order in which they occur [3]. Defining causal dependencies accurately is essential for maintaining consistency, debugging, and security in distributed systems. At its core, causality in distributed systems can be categorized into two types [5]:

- *Internal Causality*: Refers to the dependencies between events within a single process
- *External Causality*: Refers to the dependencies between events across different processes, considered as mediated by message passing.

Several models and techniques have been developed to formalize and analyze causal dependencies:

1) Lamport's Happened-Before Relation among Events: Lamport's logical clocks introduce the "happened-before" relation (denoted $\prec$), which is used to establish a partial ordering of events in a distributed system [3]. According to Lamport, event a happened before event b ($a \prec b$) in one of following cases:

- If a and b are events in the same process, a comes before b.
- If a sent a message m, b received exactly that message.
- If $a \prec c$, $c \prec b$, then $a \prec b$

2) Contextual Dependency Models: More advanced models, such as D'Ausbourg's and Xosanavongsa's causal dependency relationships, incorporate additional dimensions, including object states and contextual actions [6] [1]. These models

provide a nuanced understanding of dependencies, particularly useful in heterogeneous and multi-step attack scenarios.

The accurate definition and modeling of causal dependencies are essential in many domains, especially in cybersecurity. In systems where multiple processes interact, causality helps to identify the progression of events leading to incidents, enabling more precise detection, correlation, and response strategies. Despite its importance, challenges remain in defining causality due to the absence of global synchronization, the complexity of distributed interactions, and the inherent philosophical and practical nuances of the concept.

III. Xosanvongsa Model

The Xosanavongsa model provides a formal framework for defining and analyzing causal dependencies among heterogeneous events in distributed systems, known as the contextual action causal dependency relationship [6]. This model combines the strengths of Lamport's "happened-before" relationship and D'Ausbourg's causal dependency framework, extending them with a contextual dimension to deepen the understanding of causality in complex, multi-event systems.

1) Contextual Action: A contextual action is defined as the combination of an action a performed by an object o and the state (o, t) of object o at time t. Objects are categorized as:

- *Active Objects*: Processes or network interfaces that produce actions.
- *Passive Objects*: Information containers, such as files, sockets, memory, and pipes, which do not produce any action.

For active objects, actions link directly to their context. For passive objects, only the context is observed. The set of actions for an object o is denoted as *ObjectActions(o)* and a contextual action is represented as $(a, (o, t))$ where $a \in ObjectActions(o)$.

2) Session Concept: A session is defined as a sequence of contextual actions performed by an object where the actions are causally dependent on each other within the session, but the first action of a session is independent of the last action of the previous session. Formally, given an object o, a session $Session_n(o)$ is defined as a sequence of contextual actions $(a_i, (o, t_i)$, where $a \in ObjectActions$(o) and $Session_n(o) = \{(a_i, (o, t_i)/(o, t_i) \rightarrow (o, t_{i+1})$ and $(o, t_{end_{n-1}}) \nrightarrow (o, t_{start_n})$ and $(o, t_{end_n}) \nrightarrow (o, t_{start_{n+1}})\}$; t_{start_n} is the time of the first contextual action of $Session_n(o)$ and t_{end_n} is the time of the last contextual action in $Session_n(o)$.

3) Contextual Action Causal Dependency Relationship: The contextual action causal dependency relationship, denoted as "$\mapsto$", is defined on the set of all contextual actions produced by all objects in the system. Given two contextual actions, (a_1, (o_1, t_1) and (a_2, (o_2, t_2)), the latter is causally dependent on the former, written as (a_1, (o_1, t_1)) $\mapsto$ (a_2, (o_2, t_2)), is true:

- if o_1 and o_2 are the same object o, $\exists$n so that (a_1, (o_1, t_1)) $\in Session_n(o)$, (a_2, (o_2, t_2)) $\in Session_n(o)$, and $t_1 < t_2$
- or if $o_1 \neq o_2$, $(o_1, t_1) \rightarrow (o_2, t_2)$ (sense of d'Ausbourg, there is an information flow from state 1 to 2)
- or if $o_1 \neq o_2$, action a_1 is sender of message m, a_2 corresponds to the reception of m $\Rightarrow a_1 \prec a_2$
- or $\exists$(c, (o,t)) so that (a_1, (o_1, t_1)) $\rightarrow$ (c, (o,t)) and (c, (o,t)) $\rightarrow$ (a_2, (o_2, t_2))

4) Contextual Events: A contextual event is represented as a triple (e, o, t_e), where e is an event from set of logged events E, o is the observed object, t_e is the event's timestamp. The observation of a contextual action $(a, (o, t_a))$ is defined as:

$$\text{Obs((a, (o, } t_a\text{)))} = \{(e_i, o, t_{e_i})\} \cup \{(\emptyset, o, t_a)\}$$

where $a \in ObjectActions(o)$, $e_i \in E$ corresponds to observation of a at time t_{e_i}, and $\{(\emptyset, o, t_a)\}$ indicates the absence of an observation. The contextual event causal dependency relationship ($\rightharpoonup$) states that two contextual events (e_1, o_1, t_{e_1}) and (e_2, o_2, t_{e_2}) are causally dependent if there exist contextual actions $(a_1, o_1, t1)$ and $(a_2, o_2, t2)$ such that:

- $(a_1, (o_1, t1)) \mapsto (a_2, (o_2, t2))$
- $(e_1, o_1, t_{e_1}) \in$ Obs$((a_1, (o_1, t1)))$
- $(e_2, o_2, t_{e_2}) \in$ Obs$((a_2, (o_2, t2)))$

5) The Event Causal Dependency Relationship: Finally, Xosanavongsa introduces the event causal dependency relationship, denoted as "$\triangleright$". Given two events $e_1 \in \mathbb{E}$ and $e_2 \in \mathbb{E}$, they are causally dependent, $e_1 \triangleright e_2$, if only if $(e_1, o_1, t_{e_1}) \rightharpoonup (e_2, o_2, t_{e_2})$ where:

- o_1 and o_2 are the observed objects
- t_1 and t_2 are the timestamps of events

The Xosanavongsa model demonstrates that if actions in the system are causally dependent, then the logs of these actions are causally dependent too. This demonstration leads to our approach to generate causal logs.

IV. Generating Causal Logs

Causal Logs generation is an advanced logging mechanism designed to capture not only individual events but also the causal relationships between them. Unlike traditional logs, which document events in isolation, causal logs focus on identifying how one event leads to another, forming a network of cause-and-effect relationships. This approach provides a comprehensive understanding of system behavior, making it particularly useful in complex scenarios such as cybersecurity, distributed systems, and forensic analysis.

A. EBPF Implementation

The Extended Berkeley Packet Filter (eBPF) is a revolutionary Linux kernel technology that enables efficient and dynamic monitoring of system calls, making it an ideal tool for capturing causal relationships within system events. By running custom programs in a safe, sandboxed environment, eBPF facilitates real-time tracing without modifying kernel source code.

1) Capturing System Calls with eBPF: System calls, which mediate interactions between user applications and the kernel, are crucial for understanding system behavior. Using eBPF, these calls can be dynamically instrumented to trace their execution and extract details such as process IDs, arguments,

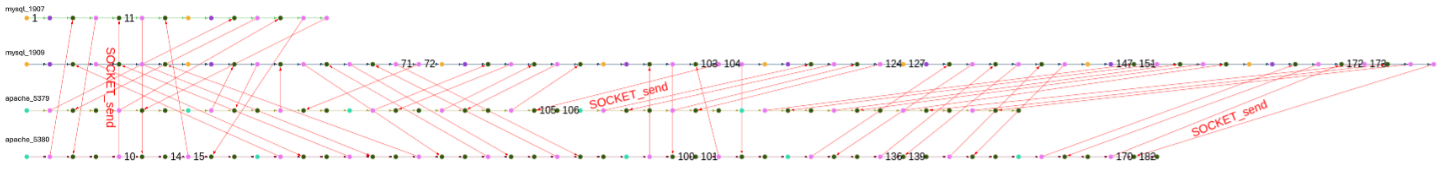

Fig. 1. Causal Dependency Graph

return values,... From those information, we can compute the causal dependencies.

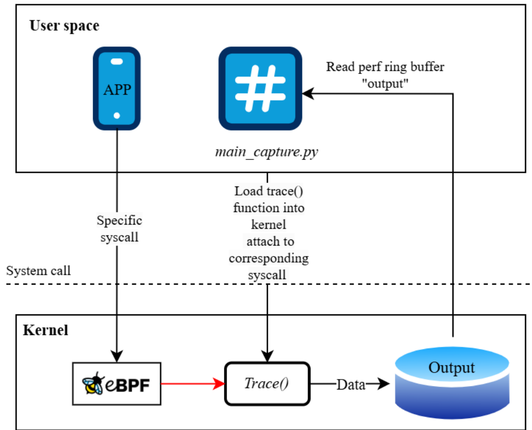

Fig. 2. EBPF Implementation

Figure 2 illustrates the interaction between user-space applications and the Linux kernel using eBPF for system call monitoring. In user space, an application (APP) initiates system calls, which are intercepted by an eBPF program loaded into the kernel via a Python script (e.g., main_capture.py). This script dynamically loads the eBPF program into the kernel and attaches it to specific system call hooks. In the kernel, the eBPF program captures relevant data during system call execution and processes it through a trace() function. The collected data is stored in a ring buffer, which acts as an efficient intermediary for transferring information between the kernel and user space. The Python script retrieves this data from the ring buffer and makes it available for further analysis or logging, enabling real-time insights into system behavior and causal relationships between system calls. This mechanism showcases eBPF's capability to seamlessly bridge user-space monitoring tools and kernel-level data collection.

2) Establishing Causal Dependencies: Based on prior research [3] [6] [1], causal dependencies between system calls can be effectively identified and represented in causal logs. These relationships are established by analyzing the interactions between events. For events occurring within a single process, Lamport's "happened-before" relationship [3] is utilized to determine causality. For external events—those involving distinct processes—D'Ausbourg's method [1] and Xosanavongsa's approach [6] provide robust frameworks for constructing these dependencies. This combination ensures a comprehensive representation of both internal and inter-process causal relationships. Here are some typical examples:

- *Process Creation*: Calls like $fork$ and $clone$ create causal links between parent and child processes.
- *Message Passing*: Communication via $pipe$ or $socket$ generates dependencies between $send$ (sender) and $recv$ (receiver).
- *File Operations*: Writing and subsequent reading from a file establish causal chains among processes.

Figure 3 provides a snippet of Causal logs, where each column represents the following in order: ID, PID, SYSCALL, TIME, PREV_NODE, VECTOR_TIME, UUID. The vector time is the mean to compute causal dependency. This example illustrates how communication between two processes is established and executed.

```
21,1892,accept4,4846861086685,[5],{1892: 3},0d85a6e3-5606-4e12-8883-ddc2c05598ef
22,6233,getpeername,4846861235019,[16],"{6233: 8, 1892: 2}",06cc0d5f-2c41-4636-aac8-c3d3035625fb
23,6233,sendto,4846861314654,[22],"{6233: 9, 1892: 2}",9d300aba-998b-4236-9d57-6430a39b0193
24,1893,recvfrom,4846861399253,"[20, 23]",{1893: 9},2575c682-f0e6-48f3-ade8-aad279ba847a
25,1893,sendto,4846861453227,[24],{1893: 11},9f780c84-fede-4457-a285-3dd9b4223b8e
26,6233,recvfrom,4846861484204,"[23, 25]","{6233: 11, 1892: 2}",bae95833-0dbe-4dc1-a4c1-ed8f06a9e76
```

Fig. 3. Causal logs

B. Result Graphs

The experiments were conducted on a system comprising two web applications with SQL injection vulnerabilities. The resulting graph, shown in Figure 1, demonstrates a significant number of $send$ and $recv$ system calls between the Apache and MySQL processes. This graph effectively illustrates the causal relationships between events, including those spanning across different processes. The red arrows, indicating socket communication (external causality), highlight how data flows between Apache and MySQL, allowing us to trace the dependencies between these events. The visualization provides a clear representation of inter-process interactions, making it easier to analyze complex behaviors and detect potential security vulnerabilities.

V. Attack Explanation

A. SQL Injection attack

In Figure 4, the causal dependencies of a SQL injection attack on a web application are isolated for detailed analysis. This visualization highlights the interaction between Apache ($apache$_1939 and $apache$_1940) and MySQL ($mysql$_1975 and $mysql$_3917) during the attack, capturing both internal events and their relationship with external activities. The explanation of workflow:

- *Initial Query*: When a user accesses the web application, $apache$_1940 is triggered to interact with the database, querying information to handle initial operations.
- *Attack Execution*: During a search or SQL injection attack, $apache$_1939 processes a query for full data retrieval. This interaction is reflected in the graph as a series of $SOCKET_send$ events (red arrows) between Apache and MySQL processes.

The graph not only illustrates these interactions but also provides insights into the flow of activities within and across processes, including the $CLONE$ operation that establishes process relationships. This detailed view enables us to trace the progression of the attack more efficiently.

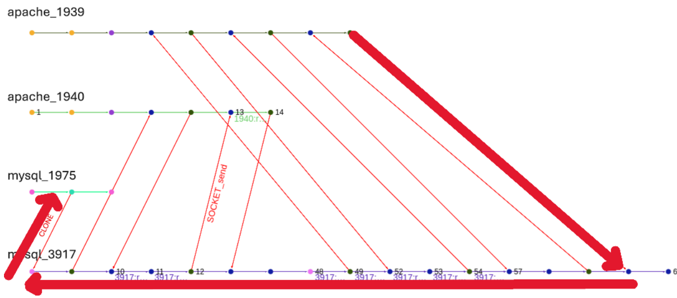

Fig. 4. SQL Injection attack graph

B. Explaining the attack

In this section, we analyze the SQL injection attack by tracing it backward from its endpoint. As shown in Figure 4, the final message is sent from $apache$_1939 to $mysql$_3917, indicating that the data query has been completed. Here, $apache$_1939 serves as the intermediary process, handling user queries from the web application and communicating them to the database. The database process, $mysql$_1975, is responsible for managing the database, while $mysql$_3917 — a clone of $mysql$_1975 — is designated for exchanging data with the $apache$_1939 process. By all of above information, we can infer the path of attack:

- *Attack Initiation*: The attacker performs the SQL injection through the web application, with $apache$_1939 being the primary process handling the malicious query.
- *Database Query*: $apache$_1939 establishes a connection to the database to process the attacker's request.
- *Connection Handling*: The database process, $mysql$_1975, accepts the connection request and clones a new process, $mysql$_3917, to handle the communication with $apache$_1939.
- *Data Exchange*: The attacker retrieves the requested data through the socket communication $send$ and $recv$ between $mysql$_3917 and $apache$_1939.

This backward analysis (as shown on Figure 4) provides a clear understanding of the interaction flow and the roles of each process during the attack. The causal graph highlights critical points in the attack path, making it easier to identify vulnerabilities and the potential impact of SQL injection in the system.

VI. Conclusion and Future Work

This paper presented a novel framework for generating causal logs using eBPF to enhance the detection and analysis of cybersecurity threats. By leveraging kernel-level instrumentation, our approach provides explicit causal relationships between system events, offering improved clarity and relevance in explaining complex multi-step attacks. The experiments conducted on SQL injection scenarios demonstrated the framework's capability to uncover attack paths and support forensic investigations. This methodology enhances the interpretability of logs and strengthens the foundation for addressing intricate cybersecurity challenges.

While the proposed approach shows significant promise, several limitations were identified. The overhead introduced by eBPF instrumentation in capturing detailed causal relationships poses a challenge, particularly in high-performance or resource-constrained systems. Furthermore, scalability remains an issue when addressing large-scale distributed environments with extensive volumes of system events.

Future research will aim to address these limitations and expand the applicability of the framework through the following directions:

- *Performance Optimization*: Reducing the overhead of system instrumentation by exploring lightweight tracing mechanisms or leveraging hardware-assisted monitoring tools.
- *Scalability Enhancements*: Developing distributed architectures to handle causal graph generation across multiple nodes, minimizing computational loads on individual systems while supporting large-scale deployments.
- *Advanced Detection Techniques*: Integrating machine learning approaches, such as Graph Neural Networks, to analyze causal graphs for detecting subtle attack patterns and uncovering complex correlations that may be overlooked by traditional methods.

By addressing these aspects, this research sets the stage for advancing the robustness, scalability, and applicability of causal log generation frameworks. These developments will contribute to strengthening the effectiveness of security monitoring systems and improving defenses against sophisticated cyber threats.

VII. Acknowledgement

This work has been partially supported by the French National Research Agency under the France 2030 label (Superviz ANR-22-PECY-0008). The views reflected herein do not necessarily reflect the opinion of the French government.

References

[1] D'ausbourg, B. (1994). Implementing secure dependencies over a network by designing a distributed security subsystem. In Computer Security—ESORICS 94: Third European Symposium on Research in Computer Security Brighton, United Kingdom, November 7–9, 1994 Proceedings 3 (pp. 247-266). Springer Berlin Heidelberg.

[2] Kobayashi, S., Fukuda, K., & Esaki, H. (2017, May). Mining causes of network events in log data with causal inference. In 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM) (pp. 45-53). IEEE.
[3] Lamport, L. (2019). Time, clocks, and the ordering of events in a distributed system. In Concurrency: the Works of Leslie Lamport (pp. 179-196).
[4] Markakis, M., Youngmann, B., Gao, T., Zhang, Z., Shahout, R., Chen, P. B., ... & Cafarella, M. From Logs to Causal Inference: Diagnosing Large Systems.
[5] Xosanavongsa, C., Totel, E., & Bettan, O. (2019, June). Discovering correlations: A formal definition of causal dependency among heterogeneous events. In 2019 IEEE European Symposium on Security and Privacy (EuroS&P) (pp. 340-355). IEEE.
[6] Xosanavongsa, C. (2020). Heterogeneous Event Causal Dependency Definition for the Detection and Explanation of Multi-Step Attacks (Doctoral dissertation, CentraleSupélec).