electronics

MDPI

Article

CrptAC: Find the Attack Chain with Multiple Encrypted System Logs

Weiguo Lin [1], Jianfeng Ma [1], Teng Li [1,*], Haoyu Ye [1], Jiawei Zhang [1] and Yongcai Xiao [2]

[1] School of Computer Science and Technology, Xidian University, Xi'an 710071, China
[2] State Grid Jiangxi Electric Power Research Institute, Nanchang 330077, China
* Correspondence: tengli@xidian.edu.cn

Abstract: Clandestine assailants infiltrate intelligent systems in smart cities and homes for different purposes. These attacks leave clues behind in multiple logs. Systems usually upload their local syslogs as encrypted files to the cloud for longterm storage and resource saving. Therefore, the identification of pre-attack steps through log investigation is crucial for proactive system protection. Current methodologies involve system diagnosis using logs, often relying on datasets for feature training. Furthermore, the prevalence of mass encrypted logs in the cloud introduces a new layer of complexity to this domain. To tackle these challenges, we introduce CrptAC, a system for Multiple Encrypted Log Correlated Analysis, aimed at reconstructing attack chains to prevent further attacks securely. CrptAC initiates by searching and downloading relevant log files from encrypted logs stored in an untrusted cloud environment. Utilizing the obtained logs, it addresses the challenge of discovering event relationships to establish the attack provenance. The system employs various logs to construct event sequences leading up to an attack. Subsequently, we utilize Weighted Graphs and the Longest Common Subsequences algorithm to identify regular steps preceding an attack without the need for third-party training datasets. This approach enables the proactive identification of pre-attack steps by analyzing related log sequences. We apply our methodology to predict attacks in cloud computing and router breach provenance environments. Finally, we validate the proposed method, demonstrating its effectiveness in constructing attack steps and conclusively identifying corresponding syslogs.

Keywords: attack chain; SSE; provenance; log correlation

Citation: Lin, W.; Ma, J.; Li, T.; Ye, H.; Zhang, J.; Xiao, Y. CrptAC: Find the Attack Chain with Multiple Encrypted System Logs. *Electronics* **2024**, *13*, 1378. https://doi.org/10.3390/electronics13071378

Academic Editor: Zbigniew Kotulski

Received: 13 March 2024
Revised: 1 April 2024
Accepted: 2 April 2024
Published: 5 April 2024

1. Introduction

There are more and more intelligent devices, such as drones, vehicles, and AI cameras, connected to networks. At the same time, malicious attackers increasingly find breaches in these systems and make use of them. For instance, in the real world, an experienced cyberattacker can initiate elusory infiltration steps to prepare for the final attack like reconnaissance and gathering as much information about the target as possible, scanning and sending probes to identify the vulnerabilities, and gaining access and exploiting the breaches [1]. Thus, the malicious adversary has accomplished their goal in advance of network or device paralysis. Such attacks, e.g., Advanced Persistent Threat [2] (APT) can infiltrate target systems progressively, establishing a presence within them for an extended duration while evading detection [3,4]. Intrusions can lead to significant consequences, including the leakage of private personal information, theft of property, and pilferage of website content. The consequences can affect regular city services, social order, and the safety of people's lives and properties. However, detecting or preventing these attacks proves challenging. First, they are significantly complicated, and some of the attacks are government-funded and used as network weapons. Second, they are not hit-and-run attacks. In layman's terms, once a network is infiltrated, the perpetrator remains to gain as much information as possible. Third, they are manually launched against the system

and are hard to predict. Furthermore, disclosing the root cause and attack trace is vital for the system or devices to avoid future harm. Even though these attacks are difficult to detect, they still leave clues after the 'crime', as various logs record the 'crime scene'. The clues to these attacks can be spotted, and the regular attack steps can be diagnosed from the log data. Thus, log analysis provides the ability to identify the malicious adversaries and respond to security threats.

However, referring to the logs does not grant omnipotence, and there are still primary and thorny challenges in this field—starting with the vast volume of data posing a computational challenge. Network devices can create a mass of logs, and leveraging logs to trace back the origin cause or detect the anomalies is like finding a needle in a haystack [4]. Second, encrypted log storage [5] posing a searching challenge. Due to the immense amount of data, logs are transferred to the Syslog Server or cloud for storage. Because logs are sensitive data, they are usually encrypted to prevent information leakage, and this can make data searching difficult. Third, logs present a semantic challenge for analysis due to their siloed lenses and unstructured nature [6]. For instance, a network IDS focuses on streams and packets, while an application log examines sessions, users, and requests. Although both systems log similar events, they articulate activities differently. Moreover, logs record static, fixed points in time, lacking the complete sequence context, which complicates attack prevention. Thus, these challenges add difficult dimensions to attack chain construction.

For the observation of attacks on devices or systems, investigating only one type of log is not enough for a diagnosis [7]. A substantial number of sophisticated attacks against the system or device and their variants persist. The depiction of regular attack steps, as indicated by secure syslogs, cannot be fully captured by a singular source of logs [8]; thus, auditors cannot obtain a comprehensive overview of the conditions necessary to trigger the alarm. Current log analysis techniques [9] are generally applied in the attack diagnosis field by causally analyzing DNS logs, HTTP logs, WFP logs, and system logs and correlating them to trace the origins of attacks. However, most current solutions in logging are coarse-grained (e.g., Cisco routers document unauthorized IP addresses in both malicious and benign events). The coarse granularity of the chain of attacks and origin will incur a dependence explosion problem [10,11] and makes it complicated to recognize the true events relevant to an attack.

As a promising privacy-preserving information retrieval technique, symmetric searchable encryption (SSE) was first proposed by Song et al. [12]. Following this, a large number of studies have emerged on this subject. Recently, Wang et al. [13] proposed MFSE to address the challenge of multi-keyword fuzzy search over encrypted data without a predefined dictionary. However, the above schemes suffer from many shortcomings that constrain their practical application in the log file system. Moreover, many approaches to log correlation analysis rely on parameter-based causality analysis [14,15]. They may yield meaningfully false relations due to their inability to uncover sound logical or semantic connections among logs or events [16,17]. Moreover, some log analysis approaches are invalid in the face of the immense volumes of encrypted data, and all-data decryption is the least efficient way to tackle this problem. Finally, the approaches based on deep learning [18] and machine learning [19] must combine historic attack chain data in their training sets and cannot break the bottleneck of detecting new malicious attack chains. A comparison between current methods and the proposed method is shown in Table 1.

Table 1. Method Comparison.

Method	[9]	[10]	[18]	[16]	[20]	CrptAC
Ciphertext Search						✓
Correlation	✓	✓			✓	✓
Provenance		✓			✓	✓
Training set	✓		✓	✓		

Note: ✓ indicates that the feature is supported by the method.

Given the aforementioned constraints, we are motivated to identify the attack chain by leveraging the information contained in multiple encrypted logs. As an extension of our earlier work [7,21], we put forward a secure and potential mechanism for logically related attack path discovery. Because of the immense amount of encrypted logs, deciphering all the data and analyzing the logs is not practical. Thus, we firstly need to search for related log files within encrypted data and decrypt the logs into plain text. With this approach, we can evade deciphering all the data and diminish the time cost. Rather than taking only a single source of log data, as in [22], we adopt multi-source log data to understand the system's state, thereby enhancing the accuracy and relevance of the attack chain reconstruction. It is noteworthy that, to the best of our understanding, this proposal signifies the inaugural application of the Longest Common Subsequence (LCS) in identifying logically connected attack paths. The introduction of LCS allows the detection of subtle and complex behavioral patterns across different log entries, thus aiding in the identification of threats that might elude detection due to their intricate or obfuscated nature—something that may not be readily apparent through traditional analysis methods. These advantages enable CrptAC to efficiently process vast amounts of encrypted log data, thereby allowing the rapid and accurate identification of complex attack behaviors within encrypted log data, all while ensuring the privacy and security of the data.

In summary, this paper contributes in the following ways:

1. We propose a novel SSE scheme that supports fuzzy multi-keyword search and result ranking, offering a solution for handling keyword imprecision. This enhances the system's ability to recognize various attack patterns, particularly when dealing with ambiguous or fluctuating keywords. (Section 3.1);
2. We enhance the accuracy of tracing the cause of attacks by integrating and correlating logs from multiple sources within the system, rather than relying on a singular perspective. This multi-faceted approach allows for a more comprehensive analysis, leading to more precise identification of attack origins. (Section 3.2);
3. Our framework employs the LCS matching algorithm for tracing regular steps and constructing attack chains, enabling the identification of systematic attack patterns even in the absence of explicit alerts from security system logs. This approach thus provides a powerful mechanism for early threat detection and prevention. (Section 3.4);
4. Our approach can identify syslog sequences to proactively defend against attacks to avoid device or network paralysis. (Section 4).

The remaining sections of this paper are structured as follows. In Section 2, we provide a brief overview of related work. Section 3 delineates the system model, presents the comprehensive construction of our system, and offers a detailed theoretical analysis. In Section 4, we showcase the experimental results and conduct a performance evaluation of our system. Finally, we conclude this paper in Section 5.

2. Related Work

Log Correlation Analysis. Log correlation techniques, such as those proposed in [9,21], are widely adopted in the field of attack diagnosis. These methods conduct causality analysis on various logs, including DNS logs, HTTP logs, WFP logs, and system logs, from communication platforms or devices. Through correlating these logs, they establish attack provenance. However, many existing logging techniques exhibit a coarse granularity,

as seen in the example of Cisco routers reporting unauthorized IPs in both malicious and benign events. This coarse granularity, when analyzing the attack chain or root causes, leads to the dependence explosion problem [23], making it challenging to identify the actual events associated with the attack. Additionally, some log correlation analysis approaches rely on parameter-based causality analysis [14], which can result in significant false relations due to a failure to uncover valid logical or semantic relations among logs or events [16,17]. Moreover, the identification of events in log entries poses a challenge [20,24]. Many approaches attempt to address this issue by relying on sliding time windows [18,25]. However, how to set the time window is often ambiguous, and the heavy workload makes these approaches impractical for acceptance in a real system [26].

Anomaly Detection. Using syslogs for anomaly detection is a well-researched topic in systems and networks. Fukuda et al. [27] suppressed less important and usual log messages to uncover hidden anomalies by employing global weights [25,28]. However, since only unique events carry high weight, these methods struggle to distinguish apparent differences in anomaly detection results. FDiag, a diagnostic tool introduced by Chuah et al. [29], identifies significant events leading to compute node soft lockup failure through message template extraction, statistical event correlation, and episode construction. Nonetheless, its diagnostic capabilities are limited to identifying event sequence dates and correlated events for only one time period. Barre et al. [30] utilize a recent APT malware corpus to extract features from logs and train classifiers to detect new anomalies. However, these studies primarily focus on anomaly detection and overlook attack prediction and prevention aspects. Furthermore, they use the provided training labeled dataset, which cannot be may not be representative of some new attacks [31].

Symmetric Searchable Encryption. It is widely acknowledged that Song et al. [12] introduced the first symmetric searchable encryption (SSE) scheme, enabling keyword searches over encrypted data within linear search time. Subsequently, numerous schemes have been developed based on this foundational work. In an effort to address the limitation of static searching, the scheme presented in [32] introduces dynamic SSE. However, it lacks support for forward privacy. The scheme proposed in [33] addresses this latter concern by offering forward privacy through a complex hierarchical data structure, and the work in [34] achieves limited forward privacy. Furthermore, Wang et al. [35] devised an SSE scheme that resolves the problem of result ranking using techniques such as keyword frequency and novel order-preserving encryption. Nevertheless, none of the SSE schemes mentioned above accommodate multi-keyword setting. Seeking to enhance searchability, Cao et al. [36] introduced a multi-keyword ranked search scheme with privacy-preserving features using symmetric encryption. However, all of the above schemes exclusively support exact keyword searches.

3. System Model

Unlike conventional anomaly detection, our proposal focuses on the approach of reconstructing an attack chain by starting from the point that the IDS have found the anomalies. We split the multiple keywords from the final attack logs. In the working environment, the log data are stored encrypted on a cloud or server. Thus, we must search the related log entries in the encrypted data according to multiple keywords. We regard these logs as having logical and textual data related to the attack, even though they may be regular events and not malicious events. If these steps consistently manifest preceding an attack, we consider them as key steps in the attack chain. For instance, when an adversary successfully cracks the system's password, subsequent login activity is deemed a plausible action that lacks spurious indications in log entries. Such semantic relationships are beyond the detection capabilities of anomaly detection approaches. The construction of the entire logical attack chain and the identification of the attacker's actions pose challenges in our log correlation analysis.

The overall system, which is shown in Figure 1, consists of three main components:

(1) Cipher text retrieve. Initially, we query encrypted logs for keywords related to the attack, verifying the integrity of the logs found. This step ensures that we focus only on relevant data;
(2) Events correlation. Using a weighted graph, we assess how different events relate to one another, removing logs that do not contribute to understanding the attack. This analysis helps to streamline the pool of data under consideration;
(3) Attack chain construction. In this part, we aim to establish a logical sequence of events leading up to the attack. After decrypting the relevant logs, we standardize data from varied sources like DNS, CPU usage, and firewalls for preprocessing. Then, employing the LCS algorithm, we convert the identification of attack steps into a search for the longest common sequence of conditions. Finally, by matching syslogs with identical timestamps to these conditions, we pinpoint specific events and analyze syslog templates and parameters to uncover attackers' tactics.

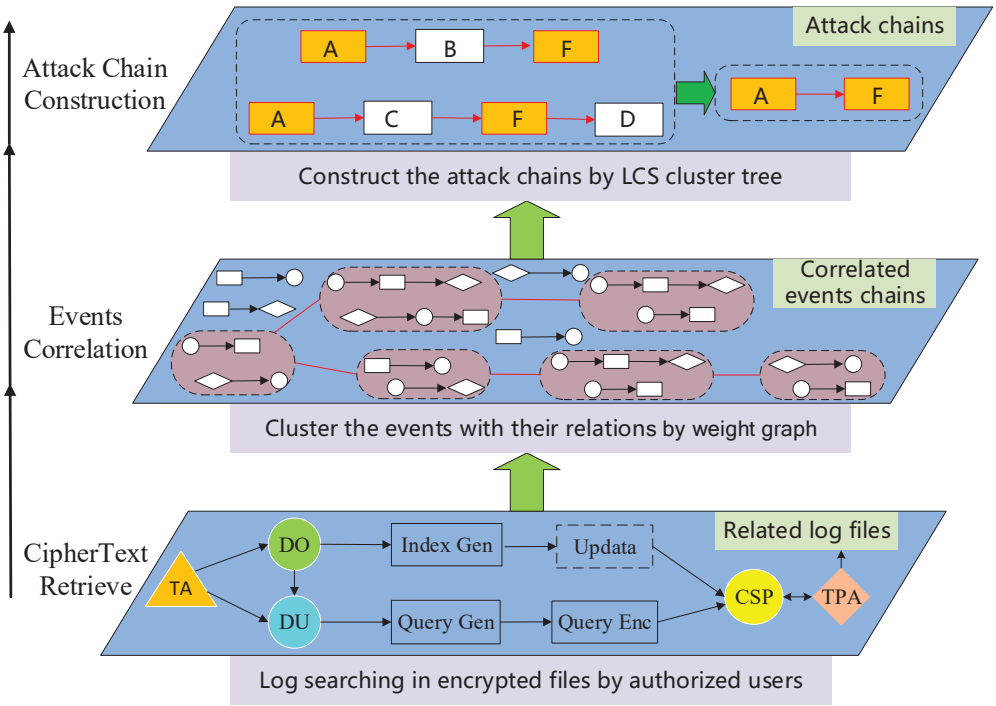


Figure 1. System Model.

Through the collaborative interaction of these components, CrptAC offers a swift and efficient method for dissecting complex attack behaviors within vast volumes of encrypted logs.

3.1. Fuzzy Multi-Keyword Symmetric Searchable Encryption

In this section, we present the Accurate Dynamic Fuzzy Multi-keyword Symmetric Searchable Encryption (Search+) approach. The proposal achieves fuzzy multi-keyword retrieval over encrypted log files outsourced in the cloud with a more accurate result ranking.

3.1.1. Notations and Preliminaries

Here, we will introduce some major notations and preliminaries used in our proposal. In Table 2, we show the some important notations.

Table 2. Notations in our proposal.

Notations	Descriptions
$F = \{f_1, \ldots, f_m\}$	The file set of all log files
$D = \{w_1, \ldots, w_n\}$	The dictionary of all keywords in F
$CF = \{c_1, \ldots, c_m\}$	The encrypted log file stored in CSP
$W(f_i) = \{w_{i_1}, \ldots, w_{i_s}\}$	The keyword set in f_i with number s
$F(w_j) = \{f_{j1}, \ldots, f_{jt}\}$	The file set with keyword w_j
$Q = \{w_j\}$	The query with each keyword w_j

Definition 1 (**Term Frequency and Inverse Document Frequency (TF-IDF)**). *TF-IDF stands out as one of the most widely employed methods for ranking functions, used to assess the relevance scores of retrieval results. This technique encompasses two crucial attributes: term frequency (TF) and inverse document frequency (IDF). TF gauges the significance of a term within a document, calculated as the number of occurrences of a specific keyword in a file, as illustrated in Equation (1). Meanwhile, IDF assesses the significance of a term across the entire document collection. The IDF of a particular keyword in the file set corresponds to the ratio of the total number of documents in the file set, denoted as F, to the total number of files containing the keyword, as outlined in Equation (2). The TF-IDF assigns to each keyword a light weight in the file by computation in Equation (3).*

$$tf_{i,w_j} = n_{i,w_j} \tag{1}$$

where tf_{i,w_j} is the TF value of keyword w_j in file f_i, and n_{i,w_j} is the number of keyword w_j in file f_i:

$$idf_{w_j} = \log \frac{N}{|F_{w_j}|} \tag{2}$$

where N represents the total number of files in file set F and $|F_{w_j}|$ denotes the number of files containing keyword w_j.

$$tf\text{–}idf_{i,w_j} = tf_{i,w_j} \times idf_{w_j} \tag{3}$$

Definition 2 (**Bloom Filter (BF) [37]**). *A Bloom Filter is a highly space-efficient data structure, represented by an m-bit array in which all positions are initially set to 0. It serves the purpose of representing a collection and determining whether an element belongs to the collection. Suppose a collection as $S = \{a_1, \cdots, a_n\}$, a Bloom Filter employs l independent hash functions from $H = \{h_i \mid h_i : S \to [1, m], 1 \leq i \leq l\}$ to insert an element $a_i \in S$ into the Bloom Filter by setting all of the $h(a_i)$-th positions to be 1. To verify whether an element $a' \in S$, it is input into each of the l hash functions to obtain the l array positions. If the bit at any position is 0, then $a' \notin S$; otherwise, either $a' \in S$ or a' yields a false positive. The false positive rate of an m-bit Bloom Filter is approximately $(1 - e^{-\frac{ln}{m}})^l$. The optimal false positive rate is $(\frac{1}{2})^l$ when $l = \frac{m}{n} \cdot ln2$.*

Definition 3 (**Locality-Sensitive Hashing (LSH)**). *Given a distance metric d, e.g., Euclidean distance, a LSH function hashes close items to the same value with higher probability than those far apart. If any two points s, t and $h \in H$ satisfy:*

$$If \quad d(s,t) \leq r_1 : Pr[h(s) = h(t)] \geq p_1 \tag{4}$$

$$If \quad d(s,t) \geq r_2 : Pr[h(s) = h(t)] \leq p_2 \tag{5}$$

A hash function family H is $(r1, r2, p1, p2)$-sensitive, where $d(s,t)$ is the distance between the two points s and t. In our proposal, we use a p-stable LSH family.

Definition 4 (**P-stable LSH**). *A p-stable LSH is a kind of LSH that has the form:*

$$h_{a,b}(v) = \left\lfloor \frac{a \cdot v + b}{w} \right\rfloor \tag{6}$$

where a, v are vectors, $b \in [0, w]$ is a real random, and w is a fixed constant for one family.

3.1.2. Search+ Model

The system model of our proposed verifiable dynamic fuzzy multi-keyword symmetric searching encryption system with accurate ranking consists of Trusted Authority (TA), Third Party Auditor (TPA), Data Owners (DO), Cloud Service Provider (CSP), and Data Users (DU). TA is the essential entity for managing system parameters and generating and distributing the keys of symmetric key encryption. DO generates a sequence of log files periodically and continuously in incremental form. The log files will be encrypted with symmetric key encryption (i.e., AES) and then uploaded to CSP dynamically. DO is responsible for generating a secure index for the log file set and outsource the index to CSP with encrypted log files. DU queries and receives encrypted log files from CSP. When querying for documents, DU generates a token and transmits the secure token to the CSP. Upon receiving the top-k results ranked by the CSP, the DO decrypts the results using a symmetric key. CSP supports unlimited storage for encrypted log files and secure index. Meanwhile, the CPS can also provide query and computing services for DO and DU. TPA is used to check the integrity of the encrypted logs search results by interacting with CSP by running the public auditing protocol.

3.1.3. Threat Model

In our system, TA and DO are assumed to be fully reliable, whereas CSP and TPA are considered to be "honest-but-curious", that is, honestly conducting the designated protocol while attempting to infer and disclose the stored documents and private data. CSP may also observe the queries of DU or the search results to conclude whether the same keyword is being searched from secure indexes and tokens. Moreover, CSP may deduce the linkage of one token to another and the relationship between these queries. Furthermore, searching ability may be granted to unauthorized DU, which may incur data leakage. Finally, with periodically generated log files, CSP may learn some specific keywords contained in newly added log files.

3.1.4. Design Goals

In our work, we devise an efficient dynamic fuzzy multi-keyword top-k log file retrieval system with more ranking accuracy in a multi-user setting. The design goals of the system are as briefly explained below:

(1) Dynamic rank fuzzy multi-keyword search: The proposal should support top-k search result ranking and the dynamic updating feature;
(2) Privacy guarantee: The CSP should be prevented from containing additional information from encrypted logs, secure index, search result, and newly added logs;
(3) Token unlinkability: The CSP should have no ability to infer the relationship between tokens to determine whether they are from the same query, which requires randomized algorithms in tokens and queries;
(4) Multi-user support: The unauthorized DU should not have the same ability as the authorized ones;
(5) Efficiency and accuracy: The efficiency should be at least equivalent to that of the original scheme while achieving improved high ranking accuracy in search results.

3.1.5. Construction

We assume that the keyword dictionary of the log file set has been constructed in advance. In our proposal, we introduce the p-stable LSH and Bloom Filter for fuzzy keyword searching. To satisfy the requirement of LSH, we employ unigram to covert each keyword into a vector. In addition, to improve the accuracy of the ranked results, we leverage the TF-IDF technique as well as the Privacy-preserving Euclidean Distance Comparison (PEDC) scheme [38] to generate auxiliary indexes for accurate result ranking. Moreover, in a log system, a new log file is continuously generated and uploaded to a semi-

trusted CSP; thus, the secure index needs to be updated simultaneously and the integrity of the search results also need to be verified, which brings new challenges. Furthermore, the log system should satisfy the conditions of a multi-user setting, i.e., grant search ability only to authorized data users. To meet these requirements, we propose the scheme Search+, the workflow of which is shown in Figure 2. Specifically, our proposal contains the following algorithms: $KeyGen$, $BuildIndex$, $TokenGen$, $LFSearch$, $Update$.

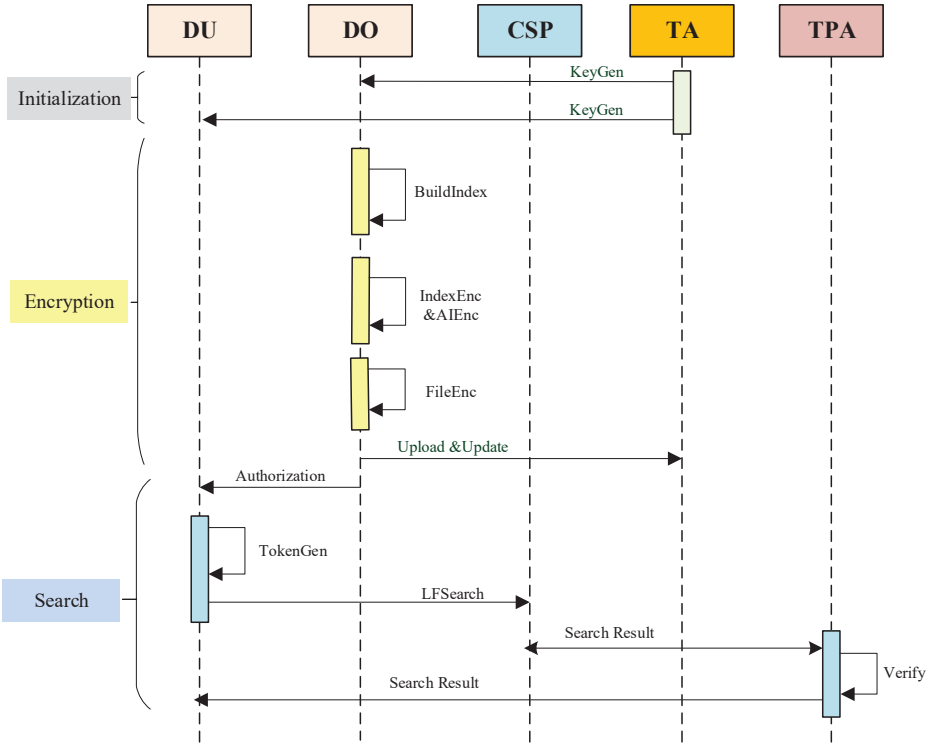


Figure 2. The Workflow of Search+.

KeyGen $(1^\lambda, F)$: The algorithm is executed by TA. Given the security parameter λ, the TA generates the secret key $SK = \{M_1, M_2, S\}$ in which $M_1, M_2 \in R^{\lambda \times \lambda}$ are two invertible random matrices of $\lambda \times \lambda$ dimension and $S \in \{0,1\}^\lambda$ is a random vector of λ dimension in which the number of 0 is approximately equal to that of 1. Meanwhile, the TA will construct another system secret key $SK' = \{M_3, {M_3}^{-1}\}$. Here, $M_3 \in R^{(n+\mu+1)\times(n+\mu+1)}$ is an invertible random matrix of $(n+\mu+1) \times (n+\mu+1)$ dimension, where n is the total number of keywords W in file sets F (μ is a random number). The output of the algorithm is $sk = \{SK, SK'\}$. Finally, the TA generates a secure symmetric key encryption scheme $SE = \{GenKey, SEnc, SDec\}$ and runs $GenKey$ to obtain K_{sym} for file set encryption.

FileEnc (F, K_{sym}): The algorithm encrypts each file $f_i \in F$ with SE to generate its ciphertext $C_i = SE.Enc(f_i, K_{sym})$ and signature $\sigma_i = (H_0(i) \cdot h^{H_1(C_i)})^{K_o}$, where i is the index of each ciphertext C_i in the encrypted log file set and H_0, H_1 are two collision-resistant hash functions generated before by TA.

BuildIndex (D, F, sk): DO generates two indexes, that is, the primary index is for query search and result ranking, the auxiliary index is for accurate ranking. As shown in Algorithm 1, for each log file f_i, the algorithm firstly runs FunGen(f_i) to create hash functions $H_i = \{h_i | i \in [1, l], h_i \in H\}$ and a λ-bit Bloom Filter (BF) I_i for the primary index and keyword set $W(f_i) = \{w_{i_1}, \ldots, w_{i_s}\}$ for the auxiliary index. Then, the auxiliary index is generated by $AIGen(W(f_i), F) = \{Score(w_1, f_i), \ldots, Score(w_n, f_i)\}(w_j \in F)$.

IndexEnc (SK, I_i): The algorithm is executed by the data user, shown in Algorithm 2. After the index I_i, each log file is divided, and the primary index will be encrypted by $SI' = \{{M_1}^T \cdot I_{i1}, {M_2}^T \cdot I_{i2}\}$.

Algorithm 1 Index Construction

Require: D, F, SK
Ensure: SI
1: **for** each f_i in F do **do**
2: $(H_i, BF, W(f_i)) = FunGen(f_i)$
3: **for** each w_{i_j} in $W(f_i)$ do **do**
4: $I_i = BF(H_i(unigram(W(f_i))))$
5: **end for**
6: $AI_i = AIGen(W(f_i), F)$
7: $SI' = IndexEnc(SK, I_i)$
8: $SI'' = AIEnc(SK', AI_i)$
9: **end for**

AIEnc **(**SK'**,** AI_i**)**: The algorithm aims to encrypt the auxiliary index for the log file set F. Given the vector AI_i for each log file, the algorithm will extend it with random numbers to (n+u+1) dimensions and obtain $AI_i{}' = \{AI_i, \alpha_1, \alpha_2, \ldots, \alpha_\mu, 1\}$. It will compute $SI'' = Enc(AI_i{}') = (\beta \cdot AI_i{}' + \gamma_I) \times M_3$, where $\beta \in R$, $r_I \in R^\lambda$.

TokenGen **(**Q**,** sk**)**: The algorithm is executed by the data user. Given the query $Q = \{w_j\}$, it generates a query vector $\overrightarrow{q} = \{q_1, \ldots, q_n\}$, where $q_j = 1$ if $w_j \in D$, otherwise, $q_j = 0$. Selecting two random integers η and ζ, it generate a λ-bit Bloom Filter V_Q and inserts the query Q into the vector V_Q using the same method as $BuildIndex$.

Algorithm 2 Index Encryption

Require: SK, I_i
Ensure: SI'
1: Divide the index I_i into two parts (I_{i1}, I_{i2}) with the random vector S of SK using the following steps.
2: **for** each i_j in I_i do **do**
3: **if** $s_j = 1$ where $s_j \epsilon S$ **then**
4: $i1_j = i2_j = i_j$
5: **else**
6: $i1_j = 0.5i_j + r$ (r is a random number)
7: $i2_j = 0.5i_j - r$
8: **end if**
9: **end for**

LFSearch **(**SI**,** T_Q**)**: The algorithm is executed by CSP. For each log file, the CSP computes the relevance of the query and current log file using the following process:

$$\begin{aligned} Score_{rel} &= < SI', T_Q{}' > \\ &= {M_1}^T I_{i1} \cdot {M_1}^{-1} \cdot V_Q j' + {M_2}^T I_{i2} \cdot {M_2}^{-1} \cdot V_Q j'' \\ &= {I_{i1}}^T \cdot V_Q j' + {I_{i2}}^T \cdot V_Q j'' \end{aligned} \tag{7}$$

The above score is used to rank the result top-k log files that are most relevant to the query. Then, the corresponding top-k file identifiers will be returned to the data user. When there exist equal scores for some log files, it is usually challenging for CSP to obtain a more

accurate ranking result. Thus, in our design, the auxiliary index works for a further ranking in this scenario. The specific procedure is described below:

$$\begin{aligned} Score_{prec} &= \left\lfloor \frac{SI'' \times {T_Q}''}{\beta^2} \right\rfloor = \left\lfloor \frac{Enc(AI') \times {T_Q}''}{\beta^2} \right\rfloor \\ &= \sum_{j=1}^{n} Score(w_j, f_i) \cdot q_j + \sum_{j=1}^{n} \eta\alpha_j + \zeta \end{aligned} \tag{8}$$

The above score is effective for accurate ranking in the scenario mentioned above. Considering multi-user authorization and updating, we have following two algorithms:

TokenGen(*Q*, *sk)*: After generating the token T_Q, DO picks a symmetric key r and shares this with CSP and authorized data users. Thus, an authorized data user can generate a valid token by $SEnc_r(T_Q)$ with key r. The CSP also can recover the token with key r by $SDec_r(SEnc_r(T_Q))$.

Verify (*C_r*, *PK_o)*: The search result C_r with n_r ciphertexts of corresponding log files is sent to TPA for integrity verification. Then, for each log file ciphertext $C_i \in C_r$, TPA selects a random number $\eta_i \in_R Z_p$ and sends the tuples $\{(i, \eta_i)\}$ to CSP. To generate the proof, CSP computes $\eta = \sum_1^{n_r} \eta_i \cdot H_1(C_i), \nu = \prod_{i=1}^{n_r} \sigma_i^{\eta_i}$ and sends the proof $\{\eta, \nu\}$ back to TPA. Next, TPA checks the equation $\hat{e}(\nu, g) \stackrel{?}{=} \hat{e}(\prod_{i=1}^{n_r} H_0(i)^{\eta_i} \cdot h^{\eta}, PK_o)$. If the equation holds, the integrity of the search result C_r is correct and TPA returns it to DU; otherwise, TPA reports an error to DU.

Update (*D*, *F*, *sk)*: The algorithm is executed by DO. When a new log file is added into log file set F, the data owner needs to update secure index SI in addition to uploading the new encrypted log file.

3.1.6. Security Analysis

In this section, we evaluate the security of our system to demonstrate that the proposal achieves the design goals.

(1) Data Confidentiality: The log files stored in the CSP are encrypted with symmetric key encryption algorithm. Meanwhile, the symmetric key is unknown to the CSP and unauthorized users, which ensures the confidentiality of the log files;
(2) Index and token security: In our proposal, the indexes and tokens are all encrypted with secret key sk, which is kept concealed from the attacker as shown in *BuildIndex* and *TokenGen*. Obviously, it is difficult for attackers to infer additional information from secure indexes and tokens;
(3) Token unlinkability: The token in our proposal is encrypted as in *TokenGen* with non-deterministic encryption algorithms. Hence, the CSP cannot obtain γ_Q for ${T_Q}''$ and obtain extra information about ${T_Q}'$ permuted by pseudo-random functions. As a result, it is infeasible for the CSP to establish the relationship between two tokens;
(4) Forward Privacy: The CSP cannot obtain more information from encrypted log files or the secure indexes, which means that the CSP has no ability to link the keyword in the newly added log files to any stored encrypted keyword. Moreover, as the CSP cannot obtain search results and establish the relationship between tokens. It also cannot learn whether the newly log files contains any stored encrypted keywords or not;
(5) Multi-user search ability: In our proposal, we use the group key to protect data users' token and make sure that only authorized data users can generate valid search tokens. Unauthorized data users cannot obtain the group key shared by the data owner and thus have no capacity to generate a satisfactory token. (Assuming that the CSP is not allowed to collude with the data users)

3.2. Plaintext Data Preprocessing

When we spot the relevant log chunks, we decipher the corresponding parts. Then, we collect logs encompassing similar types of attacks to pinpoint the timestamps of these

attacks and subsequently retrieve a diverse range of logs from that point onward. We present the description of the conditions from different data dimensions.

Table 3 gives a sampling log type for a four-dimensional case. The logs are collected from views of available memory, network traffic, available disk space, number of processes, database operations, and database network status, which implies that the condition is described by a 28-dimensional vector. The vector may not act the same for the same attack condition. For instance, attackers may employ various malicious hosts to compromise the target system, resulting in varying packet numbers in the LAN log. Nevertheless, when a picture scales, the original one has equal scaling to the scaled one in each dimension compared while their size is different, which means that the scale rate need to be evaluated.

Definition 1 (Condition Sequence) C_{ij} represents the jth pre-attack sampling sequence of the ith type attack, $C_{ij} = (\vec{c}_{ij\_1}, \vec{c}_{ij\_2}, \vec{c}_{ij\_3}, \dots)$. For instance, we take the DoS attack as the second type, and the system obtains a 28-dimension vector every 2 s. $\vec{c}_{21\_1}$ shows the condition of the DoS attack happening and $\vec{c}_{21\_2}$ shows the condition 2 s before the attack.

Definition 2 (Attack Sampling) The ith attack can be represented as $a_i = (C_{i1}, C_{i2}, \dots)$, with $A = (a_1, a_2, \dots)$ denoting the attack sampling cluster. To assess this, scaling rate ω of the same attack sampling should be obtained first by $\omega C_{i1} = C_{i2}$. Then, the initial condition of the same attack from different samplings is inferred by $\omega C_{i1\_1} = C_{i2\_1}$.

Table 3. The meaning of different dimension data.

D	Meaning	Calculation
d_1	The value of cpu utilization	Get from log directly
d_2	Rate of increase of cpu utilization	$(u_i - u_{i-1})/\Delta t$
d_3	cpu utilization rank rate	$(u_i - u_{min})/(u_{max} - u_{min})$
d_4	cpu utilization deviation	$(u_i - \bar{u})/\bar{u}$

To reduce computational complexity, we refrain from adopting k-means or k-medoid methods as used in Dlog [20]. This decision is driven by the fact that the data source in this paper originates from multiple types of syslogs, resulting in relatively high-dimensional data. To enhance processing efficiency, we employ *Fast Linear SVM* (http://vikas.sindhwani.org/svmlin.html, accessed on 5 March 2023) for condition classification. If two conditions are in the same group, they are labeled with the same condition, i.e., $\sum_{d_{i1} \epsilon C_{i1\_1}} \sum_{i=1}^{k} \alpha_i \cdot d_{i1} = \sum_{d_{i2} \epsilon C_{i2\_1}} \sum_{i=1}^{k} d_{i2}$. In addition, the weight $\omega = \vec{\alpha}$ can be assigned to each tuple of the vector such that the different sampling data of the same attack can be formalized by a decided ω.

3.3. Log Correlation Analysis

As we have converted diverse types of log information into corresponding vectors, the identification of relationships among events requires consideration. However, the initial vectors are not directly available for computation due to their high dimensionality, which consumes substantial computing resources and hinders efficiency. To tackle this issue, we utilize dimensionality reduction techniques, like the Principal Component Analysis (PCA) algorithm. PCA identifies the principal components of the dataset and transforms the data into a lower-dimensional subspace, enhancing clustering performance. The found encrypted data are only the primary data for attack chain reconstruction, and some may not have close relations to the attack. Thus, we need to find the logical relationships among the events using the plain text and further narrow the relevant event cluster.

We use the low-dimensional vector as the feature edge to construct an initial graph $G = (V, E, D)$, where V represents the set of log nodes, E denotes the set of feature edges, and D signifies the dimension of these vectors. For each log entry $v_i (i = 1, 2, \cdots, n)$, there will be a corresponding relation matrix $M_{i,j,k}$ to store the relation vector. $M_{i,j,k} = 1$ denotes that there exists a k-dimensional relationship between node i and node j, otherwise $M_{i,j,k} = 0$. However, there are many weak dependencies in the initial graph. A typical one is that the process of reading a file will be connected with all previous file-writing processes, but

this connection yields almost no significant semantics and instead leads to the dependency explosion problem. Faced with such a situation, we propose a method to transform the problem of assigning weights to different edges into a convex optimization problem.

We define A to represent an attack-related log set, while B denotes a benign log set. In the optimal case, $|e_A| \gg |e_{AB}|$ and $|e_B| \gg |e_{AB}|$, where $|e_B|$ represents the edge number within its own cluster, and $|e_{AB}|$ represents the edge number between these two clusters. However, in real-world scenarios, the case $|e_A| \ll |e_{AB}|$ often occurs, potentially reducing the accuracy of cluster identification. To address this issue, we assign corresponding weights to the edges and achieved the goal $\omega_A \cdot |e_A| > \omega_{AB} \cdot |e_{AB}|$ (ω_A is the weight assigned to $|e_A|$ and ω_{AB} is the weight assigned to $|e_{AB}|$). That implies that the algorithm should assign a global weight vector $\vec{\alpha}$ to each edge. Thus, the equation $\sum_{e \epsilon e_A} \sum_{i=1}^{k} \alpha_i \cdot e_i > \sum_{e \epsilon e_{AB}} \sum_{i=1}^{k} \alpha_i \cdot e_i$ still holds, where k is the dimension number of the edge vector, e_i is the i-th value of the vector $\vec{e}$, and we designate the dot product result as the weight of each edge ($\omega = \sum_{i=1}^{k} \alpha_i \cdot e_i$). However, the weight can take negative values using this method, and ω does not represent the global optimal solution in the graph. Consequently, we transform the aforementioned formula into a convex optimization problem:

$$\begin{aligned} max_{\vec{\alpha}} & \sum_{e \epsilon e_A} \sum_{i=1}^{k} \alpha_i \cdot e_i + \sum_{e \epsilon e_{AB}} \sum_{i=1}^{k} \alpha_i \cdot e_i \\ & -\lambda \sum_{e \epsilon e_A} \sum_{i=1}^{k} \alpha_i \cdot e_i - \frac{1}{2} \vec{\alpha}^T \cdot \vec{\alpha} \\ & s.t. 0 \leq \vec{\alpha}^T e \leq 1 \end{aligned} \tag{9}$$

where λ serves as the trade-off parameter to balance the first two terms against the third term in the objective function. The regularizer $\frac{1}{2} \vec{\alpha}^T \cdot \vec{\alpha}$ is incorporated to mitigate the risk of overfitting. The objective function is convex, and the resultant weight vector $\vec{\alpha}$ represents the global optimum.

After we assign weights to the edges in the initial graph, it is necessary to efficiently extract the cluster structure from the weighted graph. This paper uses the Enhanced Louvain Algorithm (ELA), which is based on the Louvain algorithm [39], but differs in that it takes not only modularity but also JSC similarity as optimization objectives and performs greedy iterations based on multilevel refinement. We will first introduce the optimization objective of ELA, and then describe the whole process of it.

Optimization objective: We define $A_{i,j}$ as the weight between node i and node j. $k_i = \sum_j A_{i,j}$ is the weight sum connected to node w. Here, c_i is the cluster to which node i is assigned. Function δ means that if $i = j$, then $\delta(i, j) = 1$, and otherwise $\delta(i, j) = 0$. Thus, the modularity is defined as $Q = \frac{1}{2m} \sum_{i,j} [A_{i,j} - \frac{k_i k_j}{2m}]$, where $m = \frac{1}{2m} \sum_{i,j} A_{i,j}$.

Jaccard similarity is the ratio of common neighbors of two nodes to all their neighboring nodes and it is calculated as follows:

$$Jac(i, j) = \frac{|\Gamma(i) \cap \Gamma(j)|}{|\Gamma(i) \cup \Gamma(j)|} \tag{10}$$

The cosine similarity is the quotient of the common neighbors of two nodes over the geometric mean of their respective neighboring nodes, and it is calculated as follows:

$$Sim(i, j) = \frac{|\Gamma(i) \cap \Gamma(j)|}{\sqrt{d_i d_j}} \tag{11}$$

where $\Gamma(i)$ and $\Gamma(j)$ are the neighboring node domains of node i and node j, and d_i and d_j are the degrees of node i and node j, respectively.

JCS similarity is a similarity measure calculated by combining Jaccard similarity and cosine similarity, and it is calculated as follows:

$$JCS(i,j) = Jac(i,j) + Sim(i,j) + r \tag{12}$$

where e_{ij} denotes the number of common edges between node i and node j, l denotes the total number of all edges in the graph and

$$r = \left(2 \times e_{ij} - \frac{d_i \times d_j}{l}\right). \tag{13}$$

We take the modularity and *JCS* similarity as the optimization objectives of ELA, and greedily perform bottom-up clustering to identify benign and malicious communities in the weighted graph.

Main Process of ELA: We conclude the two main phases of the Enhanced Louvain algorithm in Figure 3: (1) Local Optimization, and (2) Community Refinement and Aggregation.

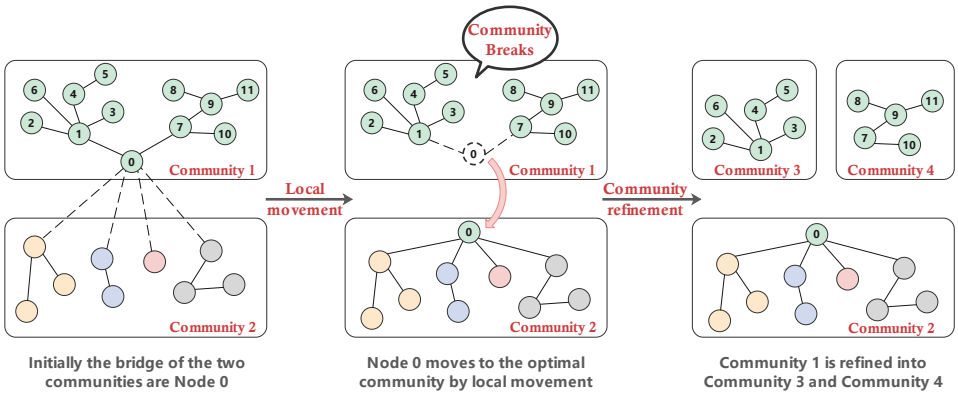


Figure 3. Main Process of the Enhanced Louvain Algorithm.

Phase 1: Local Optimization. In the first stage of local optimization, we introduce the idea of the Louvain Prune Algorithm [40], which prunes some neighboring nodes in the Louvain algorithm that make it difficult to guarantee the growth of modularity, leaving only those neighbors that have the potential to be able to increase the modularity. It is easy to observe that the movement of a node i between communities affects only the nodes of its neighbors, the nodes of its neighboring communities, and the nodes of its new community. More specifically, when we move node i from community A to community B, four types of nodes have the potential to increase community modularity:

- Neighboring nodes of the node i that are not in its new communities B that have the potential to increase the modularity of the original community A;
- Neighboring nodes of the node i that are in in the new community B. Their removal has the potential to decrease the modularity of the new community;
- Neighboring nodes of community A that have no links to the node i;
- Nodes in community B that have no links to node i.

When we consider only the first group of nodes (because they have the greatest impact on the modularity), this hardly affects the final result [40]. Assuming that there are k neighbor nodes of node i, the pruning process can reduce the time complexity of node movement from $O(k)$ to $O(1)$. Node pruning achieves a faster local optimization process while ensuring the quality of community division.

The whole process of local optimization is shown in Algorithm 3. We first initialize each node in G as a single-node community, set up the queue and add log nodes to the queue in random order, calculate $\triangle JCS$ and $\triangle Q$ of all neighbor nodes, and if $max \triangle JCS > 0$

and $max \triangle Q > 0$, divide node i from the original community A to community B where the neighbor node with the largest $\triangle JCS$ and $\triangle Q$ is located, and then add all neighbor nodes of node i that do not belong to the new community B to the queue, and continue this operation until the queue element is empty, at which time the similarity in the whole graph reaches the local optimum.

Phase 2: Community Refinement and Aggregation. Another important difference between ELA and Louvain is the refinement of community . As we can see in Figure 3, if a bridge node is moved to another community, the original community will break and thus cause poor connectivity [41]. Obviously, it is better to separate community 1 so that no disconnected communities are created, which means that nodes 1–6 should be classified as community 3 and nodes 7 should be classified as community 4. To achieve community refinement, we need to extract the existing community structure, and for each community, construct a sub-network, repeat the local optimization steps in the sub-network, and identify the refinement communities within it. Then, we aggregate each of the refinement communities into a super node. At this point, we obtain a new graph consisting of these aggregated super nodes, after which, we repeat *Phase 1* and *Phase 2* until the partition cannot be reduced; then, we can arrive at the globally optimal community.

Algorithm 3 Local Optimization

Require: *Gragh G*
Ensure: *Partition P*
  $P \leftarrow$ **INITIAL**(G);
  $L \leftarrow$ **Queue**(G);
  while isNotEmpty(L) **do**
    $i \leftarrow L$.**pop()**;
    $BestJSC = 0$;
    $BestQ = 0$;
    for each neighboring node j of i **do**
      Cj = community of j;
      if $BestJSC < GainJSC \wedge BestQ < GainQ$ **then**
        $BestJSC = GainJSC$;
        $BestQ = GainQ$;
        $BestC \leftarrow Cj$;
      end if
    end for
    P.**MoveToBestC(j)**;
    for each neighboring nodes j of i **do**
      if j **isNotIn** C **then**
        L.**push(j)**;
      end if
    end for
  end while
  return P

Through community detection on the weighted graph, we can successfully cluster log nodes into benign communities and malicious communities. By further analyzing the malicious communities in the provenance graph, we can identify the activities of the attackers and build an attack chain.

3.4. Attack Chain Construction

The LCS of two strings is usually performed in dynamic programming. Suppose $S_1 = \mathbf{a}qe\mathbf{k}pt\mathbf{p}n\mathbf{f}$ and $S_2 = \mathbf{a}x\mathbf{k}bir\mathbf{p}lc\mathbf{f}$, then $LCS = akpf$. Given a pre-attack sampling condition sequence, each condition can be regarded as a letter in the above string. If the same subsequence condition before the attack is detectable, the regular status sequence before the attack can be identified. Typically, the *divide and conquer* approach can be

employed to find the subsequence of each pair of condition chains. Assuming there are N chains in the attack sampling, the LCS computation will be $N-1$. Theoretically, *divide and conquer* outlines the regular attack steps, but the final result may obscure some diverse attack steps. For example, when an attacker attempts to crack the username and password of a router configuration, various methods can be followed to obtain them. For example, the attacker may inspect the format of the input parameters on the login page and then initiate a brute force attack. Alternatively, he can analyze and discover the vulnerabilities of the router after scanning the router configuration. After this, he initiates the attack and manages to insinuate through the backdoor. The two different attack approaches are shown in Figure 4. The original log records are shown in Figure 5, and they will record all of the system conditions along the timeline. We denote these four chains as C_{11}–C_{14} according to Definition 2.

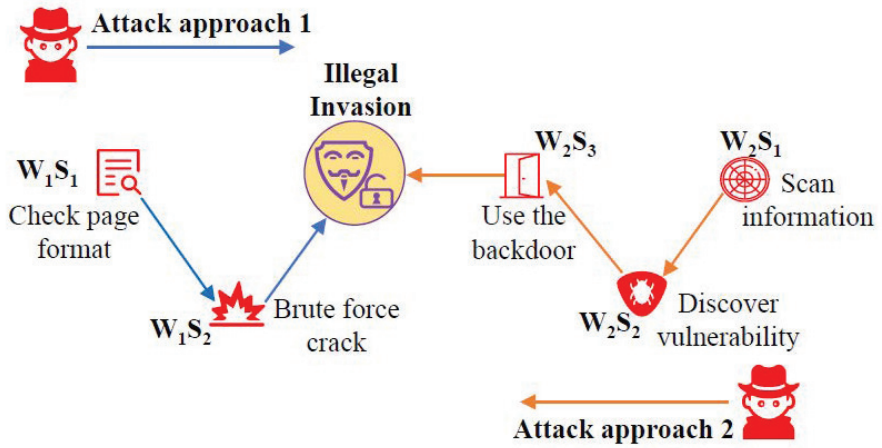


Figure 4. Two different attack routes.

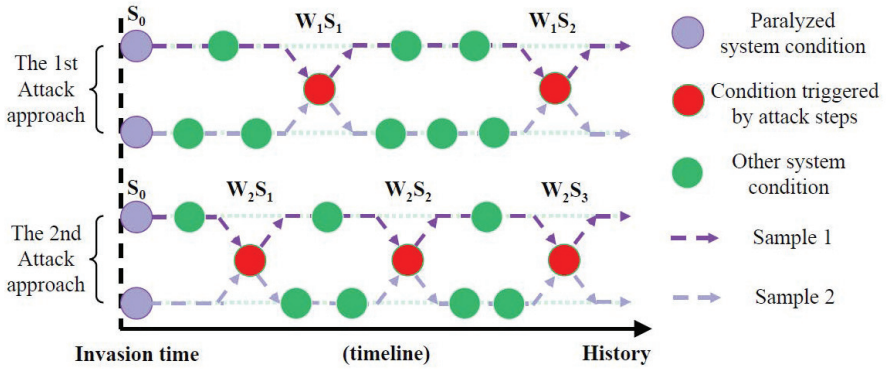


Figure 5. Different history samples for the same attack.

Using the LCS cluster tree-building algorithm, the LCS of each pair of pre-attack history conditions can be determined. As illustrated in Figure 4, since the contents of the same attack may not share common subsequences, we cannot iterate the LCS algorithm. Instead, it is imperative to construct a tree of the contents to represent the attack path and convey the logic of the conditions' relationships. The appearance time of nodes in the tree signifies the frequency of the condition, providing insights into the most commonly adopted attack chains by the attacker.

3.5. Event Analysis

The attack chains constructed from the conditions (Section 3.4) do not directly represent the events. Instead, they are utilized to identify the corresponding syslogs, enabling the

analysis of correlations among the events through log clusters. Consider the example in Figure 4. If an attack chain is identified as S_0–W_1S_1–W_1S_2, then the timestamp of W_1S_1 is extracted to locate the corresponding syslogs. The log at this timestamp serves as the central log for the event. A window φ can be set around the central point, with its length denoted as ϕ. For instance, if $\phi = 20$, this means that there are 20 syslogs before and after the central point. Subsequently, the templates of syslog clusters can be extracted using the approach from a prior work [20]. Following this, the LCS algorithm, in conjunction with *divide and conquer*, is applied to extract fixed log template sequences of the event. This facilitates the retrieval of original log entries and identification of specific attacker information, such as IP address, MAC address, and tool information.

4. Performance Evaluation

Our approach is evaluated with logs collected from the Xidian University education network's cloud and DragonStack Cloud (DragonStack Cloud address: http://222.25.188.1:50161/, accessed on 3 May 2023) from 7 January 2022 to 20 March 2022, with a total of 48.6 TB encrypted data. We take the existing work Dlog [20] and HERCULE [9] as the baseline for comparisons. Dlog discovers the root cause in the routers and HERCULE uses correlated log grapsh to find attack footprints. To be specific, we design the experiment to discover the regular attack steps based on the history syslog and predict the attack before the system is paralyzed. To validate the performance of the prediction, we launch some attacks as shown in Table 4 into the system. Figure 6 shows the multiple logs example of the DragonStack Cloud. We implement the Search+ with JAVA, and we use AES for symmetric key encryption.

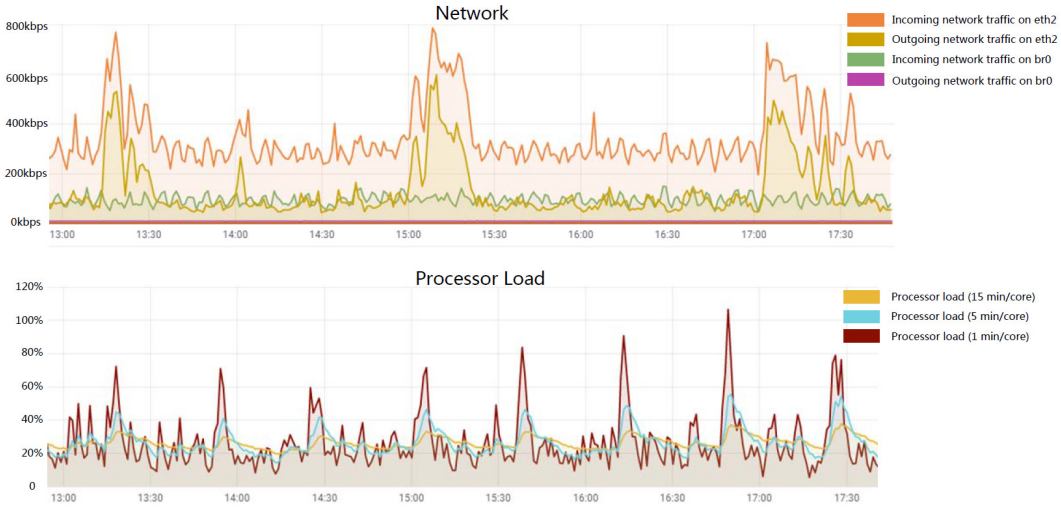


Figure 6. Multiple logs from the DragonStack Cloud.

Table 4. Emulated Attacks.

Attack Name	CVE	Initial Tactics	Post Exploitation
"Cisco IOS Input Validation Error"	2018-0171	Buffer stack overflow	Remotely execute system commands
"Cisco IOS XE Static Credential"	2018-0150	Default account vulnerability	Unauthorized remote access
"Cisco DCNM Authentication Bypass"	2019-1619	Authentication Bypass	Unauthorized access
"Cisco RV320 Access Control"	2019-1653	Unauthorized access control	Retrieve sensitive information
"Cisco RV110W Buffer Error"	2019-1663	Buffer stack overflow	Arbitrary code execution
"Cisco RV110W Authentication Bypass"	2020-3144	Bypass authentication	Unauthorized access control
"Cisco RV110W Remote command execution"	2020-3323	Send a specially crafted HTTP request	Arbitrary code execution
"Cisco ASA Remote Arbitrary File Reading"	2020-3452	Unauthorized directory traversal	Read sensitive files

4.1. Performance of Search+

We develop three processes to simulate the data owner, data user, and the CSP in the log file system. The three processes communicate with each other through RPC. We adopt the 2-stable $(\sqrt{3}, 2, p_1, p_2)$-LSH where $p_1 = 0.559$ and $p_2 = 0.286$, and choose

$l = 50, \lambda = 5000, n = 140, \mu = 10$, where λ, n, μ are the parameters used in *KeyGen* and l is the number of independent hashes in the Bloom Filter used in *BuildIndex*. The accuracies of the search result and ranking and running time of each process are shown in Figure 7.

Figure 7a,b shows the time of index generation with respect to the number of documents and keywords, respectively. Although we introduce an auxiliary index in our scheme, the procedure of index generation can be executed in parallel. Thus, the computational overhead of auxiliary index generation is of little impact. From Figure 7c,d, we can conclude that our scheme is more efficient than that of [42]. The primary reason is that the first layer index that the latter contains introduces the encrypted score for each keyword by Order Preserving Encryption (OPE) to construct an auxiliary ScoreTable, which adds much computational overhead.

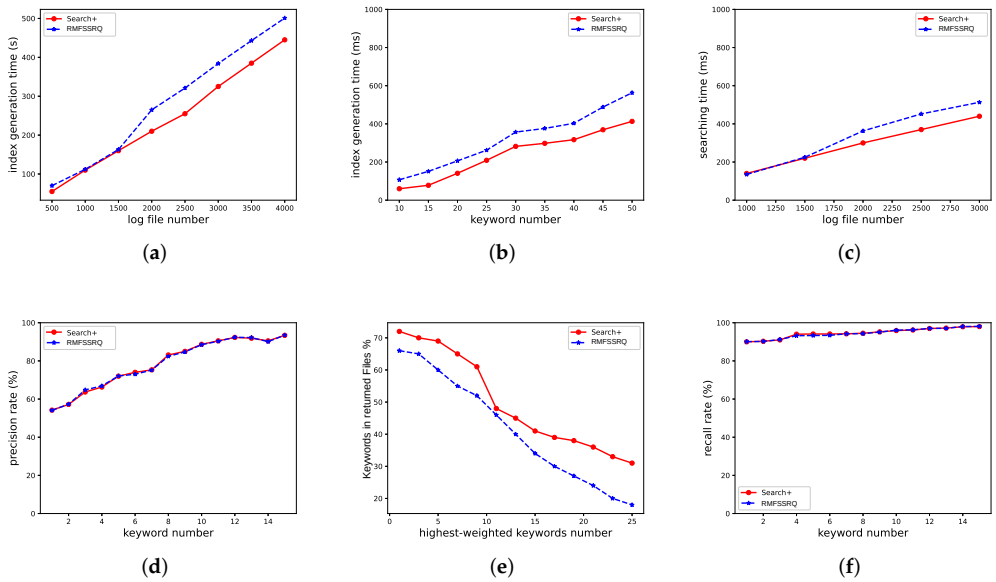


Figure 7. Performance evaluation of Search+ (RMFSSRQ scheme in [42]). (**a**) Time of Index Generation with Documents. (**b**) Time of Index Generation with Keywords. (**c**) Time of Keyword Search. (**d**) Accuracy of Precision Rate. (**e**) Accuracy Percentage of Ranking. (**f**) Accuracy of Recall Rate.

Figure 7c presents the time cost of search for files in the document set. Here, we set the search keyword number to be five as an example. We find that the time cost of multi-keyword fuzzy search is in linear relationship with the size of the document files. Moreover, from the comparison, the time cost of our scheme is less because the search procedure in [42] includes additional operations relevant to ScoreTable.

Figure 7d,e,f presents the accuracy of search results from the aspect of precision rate, ranking, and recall rate. As the accuracies of precision rate and recall rate are dependent on the parameter of hash functions, which is common between our scheme and [42], the precision rate and recall rate of two schemes are similar. Nevertheless, the result ranking of our scheme is more accurate as we introduce the auxiliary index with more accurate score value.

4.2. Performance of Log Correlation Analysis

As is shown in Section 3.2, each condition contains a vector of 28 dimensions. The dimension of the data is so high that it imposes a high computation burden on the computing nodes and decreases system performance. To counter this, Dlog [20] reduces the dimension through the use of the PCA algorithm, that is, it can reduce the 28-dimension vector to a 9-dimension vector, achieving a data compression ratio of 67.8%. However, it maintains data fidelity above 90%. The effectiveness of this dimension reduction is demonstrated in Table 5, which lists the top nine principal components identified by the PCA algorithm.

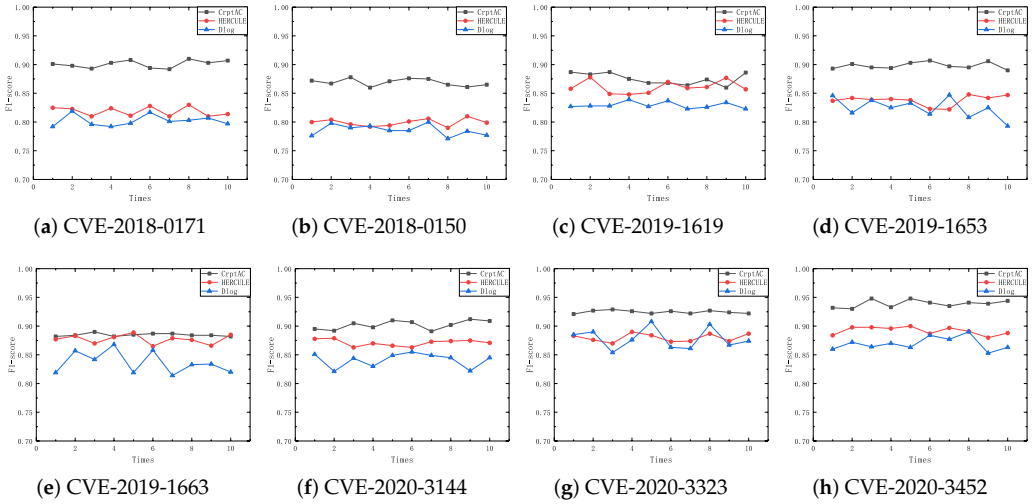
Table 5. Top Nine Principal Components.

Eigenvalues	Contribution Rate	Cumulative
2.73	15.4%	15.4%
2.55	14.8%	30.2%
2.26	13.2%	43.4%
1.94	10.6%	54.0%
1.52	9.8%	63.8%
1.27	8.5%	72.3%
1.01	7.9%	80.2%
0.83	7.4%	87.6%
0.75	5.1%	92.7%

In order to evaluate the performance of attack-related community detection, we use *F*1-*score* to measure the accuracy of classification. The number of nodes correctly classified as attack logs, the number of nodes incorrectly classified as attack logs, the number of nodes correctly classified as irrelevant logs, and the number of nodes incorrectly classified as irrelevant logs are denoted as tp, fp, tn, fn, respectively. We define *precision* to represent the proportion of nodes that are actually related to the attack among all predicted attack log nodes. *Recall* refers to the proportion of the actual log nodes related to the attack that are correctly divided into the attack log community. *F*1-*score* is the harmonic mean of precision and recall. When precision and recall are both close to 1, *F*1-*score* is also closer to 1, that is, the detection effect of the corresponding method is better.

$$\begin{aligned} precision &= \frac{tp}{tp + fp} \\ recall &= \frac{tp}{tp + fn} \\ F1 &= \frac{2 \times precision \times recall}{precision + recall} \end{aligned} \tag{14}$$

We present a performance comparison between our method and the Dlog and HERCULE methods in constructing a provenance graph to detect attack communities in Figure 8. According to the simulation experiment performance comparison of 16 Cisco router vulnerabilities in Figure 8, it can be seen that the accuracy of CrptAC in constructing the provenance graph is generally higher than that of the HERCULE method and the Dlog method in all eight simulation experiments, and the detection results are relatively more stable and reliable. The most striking result is for the detection of CVE-2018-0171, where the *F*1-*score* of CrptAC is 10% higher than that of HERCULE and 12% higher than that of Dlog. This is a significant improvement in the accuracy of attack detection. In addition, the CrptAC method does not show any outliers in any experiment except CVE-2020-3330, which indicates that the CrptAC method has higher stability and can obtain more reliable detection results.



(**a**) CVE-2018-0171 (**b**) CVE-2018-0150 (**c**) CVE-2019-1619 (**d**) CVE-2019-1653

(**e**) CVE-2019-1663 (**f**) CVE-2020-3144 (**g**) CVE-2020-3323 (**h**) CVE-2020-3452

Figure 8. Performance of CrptAC, HERCULE, and Dlog on attack-related community detection.

We show a box plot of *F*1-*score* for CrptAC, HERCULE and Dlog in Figure 9. This visualizes the distribution of *F*1-*score* for the three methods. Based on the box lengths, it can be seen that both CrptAC and HERCULE have smaller *F*1-*score* fluctuations, but CrptAC has a higher median and average line; thus, CrptAC has higher accuracy in the process of malicious community detection. By observing the outliers in the box plot, we can see that CrptAC rarely has outliers beyond the quartiles, while both HERCULE and Dlog have more, which indicates that our method is more stable than HERCULE and Dlog.

We show the provenance graph in Figure 10. The red elements represent the malicious community related to the attack, while the black elements represent the benign community unrelated to the attack. By analyzing the malicious communities, we can identify the attack chain.

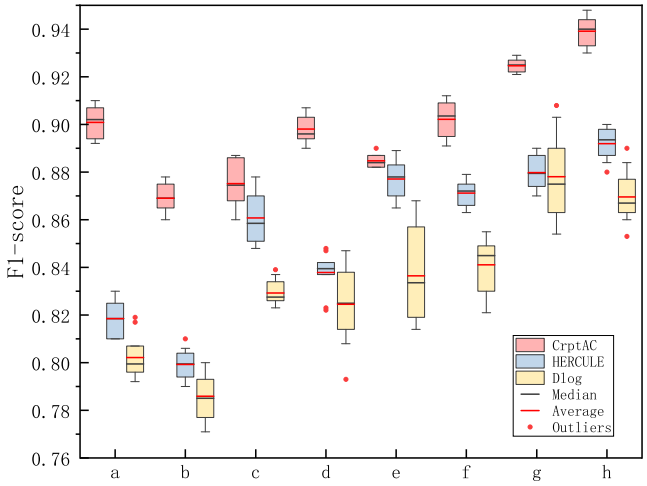
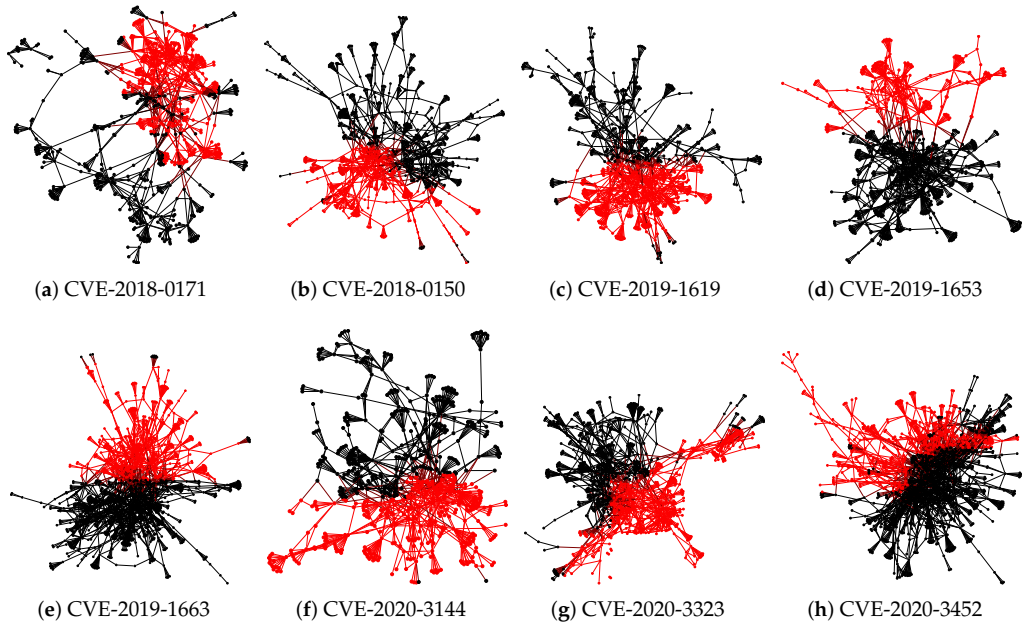


Figure 9. *F*1-*score* distribution for CrptAC, HERCULE, and Dlog in detecting malicious communities. Labels a-h denote the communities evaluated.



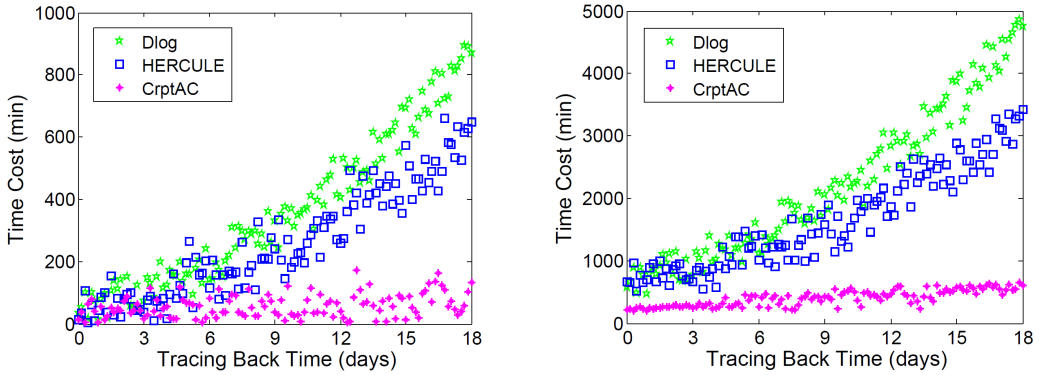
(**a**) CVE-2018-0171 (**b**) CVE-2018-0150 (**c**) CVE-2019-1619 (**d**) CVE-2019-1653

(**e**) CVE-2019-1663 (**f**) CVE-2020-3144 (**g**) CVE-2020-3323 (**h**) CVE-2020-3452

Figure 10. Distribution of malicious communities from log analysis, highlighting red for attack-related and black for benign communities, aiding in attack chain identification.

4.3. Performance with Respect to Finding the Attack Chain

In CrptAC, we employ the Fast Linear SVM algorithm for data processing to avoid computation of all the dimensions and cluster them under the same condition to transform the high-dimensional vector into a label. As a result, LCS can be obtained without directly involving the high-dimensional data. As Figure 11 shows, the approaches can perform well with the plaintext log, but the time cost with the encrypted log is almost five times the former. The results indicate that CrptAC can greatly decrease the time cost in finding the attack chain, whereas the time complexity of LCS is $O(n^2)$. The factor of n^2 is reduced from 0.13 ($0.13x^2 + 3.7x + 137.03$) to 0.00594 ($0.00594x^2 - 0.031x + 101.51$).



(**a**) Time cost with plaintext log (**b**) Time cost with encrypted log

Figure 11. Time Cost Comparison.

The accuracy with respect to finding the attack chain (i.e., Equation (15)) is also compared. In Figure 12, the highest accuracy comes from the approach that utilizes the original high-dimensional data as it preserves the most information about the events. PCA can obtain an accuracy rate of 53% on average while causing data loss and data confusion during the condition comparison. CrptAC can achieve a higher accuracy rate than PCA, approaching that of the original data. The training of the Fast Linear SVM incurs an average loss of 92.4% accuracy. Overall, CrptAC can achieve better results in terms of time cost without compromising too much on accuracy.

$$Accuracy = \frac{Correct\ chain\ finding\ number}{Total\ chain\ finding\ number} \tag{15}$$

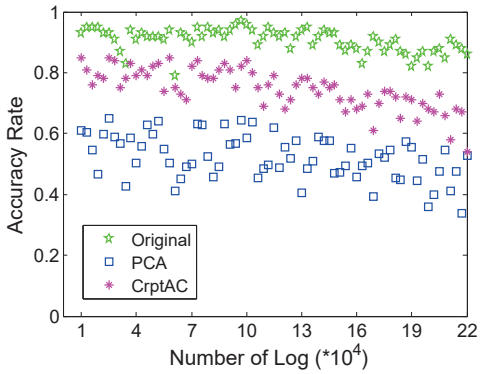
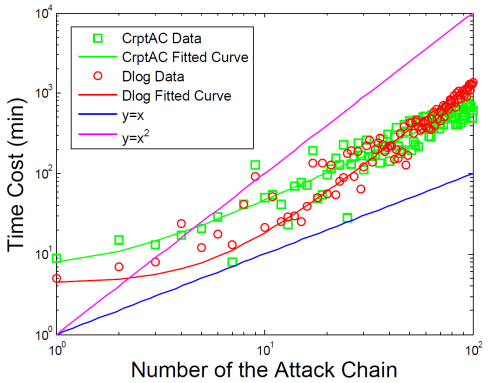


Figure 12. Attack chain finding accuracy comparison

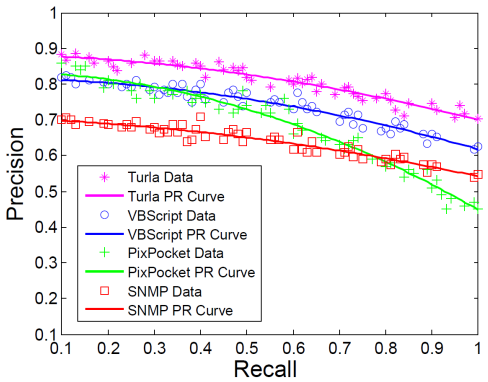
4.4. Performance with Respect to Extracting the Event's Log Sequences

With the determination of condition sequences before an attack, it becomes essential to associate the event with the condition based on knowledge of the log sequences for the specific event. Upon detecting an attack, the syslogs can be referenced directly without computing multiple source logs. For a specific condition of n sampling, the divide and conquer approach is employed to identify the LCS. The recursive tree has $\lg n + 1$ layers. The cost of each layer is denoted by cn, and the total cost of the entire tree is $cn \lg n + cn$.

We compare CrptAC with Dlog, which employs the log tree to extract templates for events, unlike the approach presented in this paper. The time complexity of Dlog that we fit is $0.14n^2 + 4.349$, obtained by traversing the log tree, as depicted by the red line in Figure 13a. The time cost of CrptAC, with a time complexity of $0.69n \lg n + 1.3n + 4.39$, is represented by the green line. As the input attack chain increases, CrptAC consumes less time when the chain exceeds 45. When processing extensive historical data, it yields satisfactory results in seeking log sequences of events.



(**a**) Time cost of finding the event's log sequences



(**b**) Precision and recall of attack prediction

Figure 13. Time Cost Comparison.

4.5. Performance with Respect to Predicting the Attack

There are four types of attacks (shown in Table 4) initiates in a cloud computing platform consisting of Windows hosts, Cisco routers, and Cisco firewalls. The history data are used for attack chain discovery, while the syslogs are directly inspected to detect the previous steps of the attack. Figure 13b illustrates the worst performance in predicting *Turla*. This implies that the attacker employs multiple vulnerability toolkits in the first step, leading to various forms of conditions and syslogs in the system. In the case of different

attack modes, regular steps can be employed for predicting the attack. With regard to the other three attacks, relatively fixed regular steps can be found before the attack and sound results obtained for their predication.

5. Conclusions

In this article, we introduce an approach, CrptAC, designed to automatically correlate multiple source logs and reconstruct attack chains in a secure manner. In our proposal, we search related log entries in encrypted logs according to keywords gained from an attack event even if the CSP is malicious. Then, we use a weighted graph to analyze the relations among events and further delete irrelevant logs. As our experimental results show, CrptAC can detect the attack steps of different injected threats and improve the speed from $0.13n^2$ to $0.00594n^2$ compared to approaches in the literature. CrptAC can also help the system prevent the attack proactively. In our future work, we intend to break the bottleneck of the deciphering process, and we will carry out all steps contained in the approach on the encrypted data and preserve the privacy of the users on the cloud.

Author Contributions: Conceptualization, W.L. and T.L.; methodology, W.L. and T.L.; software, W.L. and T.L.; validation, W.L. and J.Z.; formal analysis, W.L., T.L. and J.Z.; investigation, W.L. and T.L.; resources, J.M. and H.Y.; data curation, J.M. and Y.X.; writing—original draft preparation, W.L. and T.L.; writing—review and editing, W.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research is funded by the National Key Research and Development Program of China (2022YFB3103500), National Natural Science Foundation of China under Grant (No. 62272370, No. U21A20464), the Fundamental Research Funds for the Central Universities (QTZX23071, XJSJ23183), Young Elite Scientists Sponsorship Program by CAST (2022QNRC001), the China 111Project (No. B16037), the Qinchuangyuan Scientist + Engineer Team Program of Shaanxi (No. 2024QCY-KXJ-149), and Science Basic Research Program of Shaanxi (Program No. 2024JC-YBMS-544).

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: Author Yongcai Xiao was employed by the company State Grid Jiangxi Electric Power Research Institute. The author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Zhang, K.; Shi, Y.; Karnouskos, S.; Sauter, T.; Fang, H.; Colombo, A.W. Advancements in industrial cyber-physical systems: An overview and perspectives. *IEEE Trans. Ind. Inform.* **2022**, *19*, 716–729.
2. Rahman, Z.; Yi, X.; Khalil, I. Blockchain based AI-enabled Industry 4.0 CPS Protection against Advanced Persistent Threat. *IEEE Internet Things J.* **2022**, *10*, 6769–6778.
3. Xiong, C.; Zhu, T.; Dong, W.; Ruan, L.; Yang, R.; Chen, Y.; Cheng, Y.; Cheng, S.; Chen, X. CONAN: A Practical Real-time APT Detection System with High Accuracy and Efficiency. *IEEE Trans. Dependable Secur. Comput.* **2020**, *19*, 551–565.
4. Hassan, W.U.; Bates, A.; Marino, D. Tactical Provenance Analysis for Endpoint Detection and Response Systems. In Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 18–21 May 2020.
5. Wang, J.; Yin, X.; Ning, J.; Xu, S.; Xu, G.; Huang, X. Secure Updatable Storage Access Control System for EHRs in the Cloud. *IEEE Trans. Serv. Comput.* **2022**, *16*, 2939–2953.
6. Yu, L.; Ma, S.; Zhang, Z.; Tao, G.; Zhang, X.; Xu, D.; Urias, V.E.; Lin, H.W.; Ciocarlie, G.; Yegneswaran, V.; et al. ALchemist: Fusing Application and Audit Logs for Precise Attack Provenance without Instrumentation. In Proceedings of the 2021 Network and Distributed System Security Symposium, Virtually, 21–25 February 2021.
7. Li, T.; Ma, J.; Shen, Y.; Pei, Q. Anomalies Detection and Proactive Defence of Routers Based on Multiple Information Learning. *Entropy* **2019**, *21*, 734.
8. Hassan, W.U.; Noureddine, M.A.; Datta, P.; Bates, A. *Omega-Log: High-Fidelity Attack Investigation via Transparent Multi-Layer Log Analysis*; NDSS: San Diego, CA, USA, 2020.
9. Pei, K.; Gu, Z.; Saltaformaggio, B.; Ma, S.; Wang, F.; Zhang, Z.; Si, L.; Zhang, X.; Xu, D. Hercule: Attack story reconstruction via community discovery on correlated log graph. In Proceedings of the Computer Security Applications, Los Angeles, CA, USA, 5–8 December 2016; pp. 583–595.

10. Kwon, Y.; Wang, F.; Wang, W.; Lee, K.H.; Lee, W.C.; Ma, S.; Zhang, X.; Xu, D.; Jha, S.; Ciocarlie, G.; et al. Mci: Modeling-based causality inference in audit logging for attack investigation. In Proceedings of the NDSS, San Diego, CA, USA, 18–21 February 2018.
11. Irshad, H.; Ciocarlie, G.; Gehani, A.; Yegneswaran, V.; Lee, K.H.; Patel, J.; Jha, S.; Kwon, Y.; Xu, D.; Zhang, X. TRACE: Enterprise-Wide Provenance Tracking for Real-Time APT Detection. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 4363–4376.
12. Song, D.X.; Wagner, D.; Perrig, A. Practical techniques for searches on encrypted data. In Proceedings of the 2000 IEEE Symposium on Security and Privacy, S&P 2000, Berkeley, CA, USA, 14–17 May 2000; pp. 44–55.
13. Wang, B.; Yu, S.; Lou, W.; Hou, Y.T. Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In Proceedings of the INFOCOM, Toronto, ON, Canada, 27 April–2 May 2014; pp. 2112–2120.
14. Zengy, J.; Wang, X.; Liu, J.; Chen, Y.; Liang, Z.; Chua, T.S.; Chua, Z.L. Shadewatcher: Recommendation-guided cyber threat analysis using system audit records. In Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 22–26 May 2022; pp. 489–506.
15. Li, Z.; Chen, Q.A.; Yang, R.; Chen, Y.; Ruan, W. Threat detection and investigation with system-level provenance graphs: A survey. *Comput. Secur.* **2021**, *106*, 102282.
16. Kimura, T.; Ishibashi, K.; Mori, T.; Sawada, H.; Toyono, T.; Nishimatsu, K.; Watanabe, A.; Shimoda, A.; Shiomoto, K. Spatio-temporal factorization of log data for understanding network events. In Proceedings of the INFOCOM, Toronto, ON, Canada, 27 April–2 May 2014; pp. 610–618.
17. Kavousi, M.; Yang, R.; Ma, S.; Chen, Y. SemFlow: Accurate Semantic Identification from Low-Level System Data. In *Proceedings of the Security and Privacy in Communication Networks*; Garcia-Alfaro, J., Li, S., Poovendran, R., Debar, H., Yung, M., Eds.; Springer International Publishing: Berlin/Heidelberg, Germany, 2021.
18. Du, M.; Li, F.; Zheng, G.; Srikumar, V. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In Proceedings of the CCS, Dallas, TX, USA, 30 October–3 November 2017; pp. 1285–1298.
19. Wang, S.; Tuor, T.; Salonidis, T.; Leung, K.K.; Makaya, C.; He, T.; Chan, K. When edge meets learning: Adaptive control for resource-constrained distributed machine learning. In Proceedings of the INFOCOM, Honolulu, HI, USA, 16–19 April 2018; pp. 63–71.
20. Li, T.; Ma, J.; Sun, C. Dlog: Diagnosing router events with syslogs for anomaly detection. *J. Supercomput.* **2018**, *74*, 845–867.
21. Li, T.; Ma, J.; Pei, Q.; Shen, Y.; Lin, C.; Ma, S.; Obaidat, M.S. AClog: Attack Chain Construction Based on Log Correlation. In Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 9–13 December2019; pp. 1–6.
22. Hassan, W.U.; Lemay, M.; Aguse, N.; Bates, A.; Moyer, T. Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs. In *Proceedings of the NDSS*, Internet Society: San Diego, CA, USA, 2018.
23. Zhu, T.; Wang, J.; Ruan, L.; Xiong, C.; Yu, J.; Li, Y.; Chen, Y.; Lv, M.; Chen, T. General, Efficient, and Real-Time Data Compaction Strategy for APT Forensic Analysis. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 3312–3325.
24. Zou, Q.; Singhal, A.; Sun, X.; Liu, P. Automatic Recognition of Advanced Persistent Threat Tactics for Enterprise Security. In Proceedings of the Sixth International Workshop on Security and Privacy Analytics, New Orleans, LA, USA, 18 March 2020.
25. Yang, J.; Zhang, Q.; Jiang, X.; Chen, S.; Yang, F. Poirot: Causal Correlation Aided Semantic Analysis for Advanced Persistent Threat Detection. *IEEE Trans. Dependable Secur. Comput.* **2021**, *19*, 3546–3563.
26. Gui, J.; Li, D.; Chen, Z.; Rhee, J.; Xiao, X.; Zhang, M.; Jee, K.; Li, Z.; Chen, H. APTrace: A Responsive System for Agile Enterprise Level Causality Analysis. In Proceedings of the 2020 IEEE 36th International Conference on Data Engineering (ICDE), Dallas, TX, USA, 20–24 April 2020.
27. Fukuda, K. On the use of weighted syslog time series for anomaly detection. In Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops, Dublin, Ireland, 23–27 May 2011; pp. 393–398.
28. Hu, E.; Fu, A.; Zhang, Z.; Zhang, L.; Guo, Y.; Liu, Y. ACTracker: A Fast and Efficient Attack Investigation Method Based on Event Causality. In Proceedings of the IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Vancouver, BC, Canada, 10–13 May 2021.
29. Chuah, E.; Kuo, S.h.; Hiew, P.; Tjhi, W.C.; Lee, G.; Hammond, J.; Michalewicz, M.T.; Hung, T.; Browne, J.C. Diagnosing the root-causes of failures from cluster log files. In Proceedings of the 2010 International Conference on High Performance Computing, Goa, India, 19–22 December 2010; pp. 1–10.
30. Barre, M.; Gehani, A.; Yegneswaran, V. Mining data provenance to detect advanced persistent threats. In Proceedings of the 11th International Workshop on Theory and Practice of Provenance (TaPP 2019), Philadelphia, PA, USA, 3 June 2019.
31. Li, Z.; Cheng, X.; Sun, L.; Zhang, J.; Chen, B. A Hierarchical Approach for Advanced Persistent Threat Detection with Attention-Based Graph Neural Networks. *Secur. Commun. Netw.* **2021**, *2021*, 9961342.
32. Kamara, S.; Papamanthou, C.; Roeder, T. Dynamic searchable symmetric encryption. In Proceedings of the Proceedings of the 2012 ACM Conference on Computer and Communications Security, Raleigh North, CA, USA, 16–18 October 2012; pp. 965–976.
33. Kim, K.S.; Kim, M.; Lee, D.; Park, J.H.; Kim, W.H. Forward secure dynamic searchable symmetric encryption with efficient updates. In Proceedings of the Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1449–1463.

34. Bost, R.; Minaud, B.; Ohrimenko, O. Forward and backward private searchable encryption from constrained cryptographic primitives. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1465–1482.
35. Wang, C.; Cao, N.; Ren, K.; Lou, W. Enabling secure and efficient ranked keyword search over outsourced cloud data. *IEEE Trans. Parallel Distrib. Syst.* **2011**, *23*, 1467–1479.
36. Cao, N.; Wang, C.; Li, M.; Ren, K.; Lou, W. Privacy-preserving multi-keyword ranked search over encrypted cloud data. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *25*, 222–233.
37. Bloom, B.H. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* **1970**, *13*, 422–426.
38. Yuan, J.; Tian, Y. Practical privacy-preserving mapreduce based k-means clustering over large-scale dataset. *IEEE Trans. Cloud Comput.* **2017**, *7*, 568–579.
39. Blondel, V.D.; Guillaume, J.L.; Lambiotte, R.; Lefebvre, E. Fast unfolding of communities in large networks. *J. Stat. Mech. Theory Exp. 2008*, 10008.
40. Ozaki, N.; Tezuka, H.; Inaba, M. A simple acceleration method for the Louvain algorithm. *J. Comput. Electr. Eng.* **2016**, *8*, 207.
41. Traag, Vincent A., L.W.; Eck, N.J.V. From Louvain to Leiden: Guaranteeing well-connected communities. *Sci. Rep.* **2019**, *9*, 5233.
42. Wang, J.; Yu, X.; Zhao, M. Privacy-preserving ranked multi-keyword fuzzy search on cloud encrypted data supporting range query. *Arab. J. Sci. Eng.* **2015**, *40*, 2375–2388.