Cybersecurity

RESEARCH

Open Access

A flexible approach for cyber threat hunting based on kernel audit records

Fengyu Yang[1,2], Yanni Han[1*], Ying Ding[1], Qian Tan[1] and Zhen Xu[1]

Abstract

Hunting the advanced threats hidden in the enterprise networks has always been a complex and difficult task. Due to the variety of attacking means, it is difficult for traditional security systems to detect threats. Most existing methods analyze log records, but the amount of log records generated every day is very large. How to find the information related to the attack events quickly and effectively from massive data streams is an important problem. Considering that the knowledge graph can be used for automatic relation calculation and complex relation analysis, and can get relatively fast feedback, our work proposes to construct the knowledge graph based on kernel audit records, which fully considers the global correlation among entities observed in audit logs. We design the construction and application process of knowledge graph, which can be applied to actual threat hunting activities. Then we explore different ways to use the constructed knowledge graph for hunting actual threats in detail. Finally, we implement a LAN-wide hunting system which is convenient and flexible for security analysts. Evaluations based on the adversarial engagement designed by DARPA prove that our platform can effectively hunt sophisticated threats, quickly restore the attack path or assess the impact of attack.

Keywords: Advanced persistent threat, Cyber threat hunting, Kernel audit log, Knowledge graph

Introduction

To cope with rampant cyber threats, modern enterprises deploy various defense facilities, such as firewalls, IDS and IPS, Endpoint Detection and Response (EDR), Security Information and Event Management (SIEM) and so on. However, the methods of attack are diverse, and attackers will constantly change their attacking means until the attack successes, making it difficult for automated security systems to defend. In addition, attackers usually penetrate the system at low speed and imitate normal system behavior to avoid being observed, which increase the difficulty of automated threat detection.

Therefore, many enterprises employ professional security teams to detect potential threats in their systems. The main task of these teams is to explore the footprints of attack based on their own experience and external threat intelligence. Since the attack behavior is unknown, security analysts need to make assumptions combined with the internal network architecture of the enterprise and verify the assumptions, then eliminate false positives or restore the complete attack path. This process can be called cyber threat hunting.

In cyber threat hunting activities, security analysts usually choose kernel audit logs as the data source for analysis. The kernel audit logs record the interactions between various processes, files, memory areas, and external hosts at the bottom of the system. They are usually stored in the system as events. By cascading the events in kernel logs, security analysts can get contextual information about an event and understand the causality of the event, which is very useful for finding the sources of threats.

However, raw kernel audit logs have fatal flaws: huge size, analytical difficulty and semantic isolation. So threat hunting inevitably becomes a complicated and difficult task. For efficient analysis of massive low-level audit records, many existing methods convert the kernel audit

*Correspondence: hanyanni@iie.ac.cn
[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
Full list of author information is available at the end of the article

logs into provenance graph (Milajerdi et al. 2019; Hossain et al. 2017; Milajerdi et al. 2019; Han et al. 2020), which is a labeled, typed and directed graph. In provenance graphs, nodes represent system objects such as processes, files, networks, and memory, while edges represent specific system calls. But the provenance graph does not support interactive exploration and analysis. Threat hunting based on provenance graph requires high technical complexity and large memory space. We need to explore a more convenient and flexible method of graph data representation and application. Shu et al. (2018) designs a dedicated graph engine to achieve "threat intelligence computing" but does not study how to design agile hunting process.

To realize a practical and agile threat hunting system, the following three challenges need to be solved:

(1) The original audit events are complex and heterogeneous, and limited by the kernel audit granularity, automated information flow analysis will lead to a large number of false positives. So during the hunting process, the manual threat validation is indispensable. Therefore, an unified, well-defined graph representation is needed which should be friendly to both human and computer.
(2) When analyzing advanced threats, a practical threat hunting system should support the alignment of external IOCs(Indicators of Compromise) and the quick integration of various prior knowledge. Prior knowledge can be divided into log information, open threat intelligence and expert experience knowledge.
(3) In view of the massive characteristics of kernel audit events, as well as the APT's long-term latent and decentralized operations, the practical hunting system needs to consider a variety of factors, including storage cost, query latency and dump time.

Developing concept of Knowledge Graph(KG) provides ideas for solving the above problems. KG is used to describe various entities and concepts existing in the real world, as well as their relationships, and is represented in the form of structured triples. KG provides an ideal storage for the collected various types of graphic data and supports fast alignment of different knowledge. Interactive exploratory analysis based on knowledge graph can simulate human's thinking process to verify and reason knowledge, thus reducing human's workload. Moreover, its presentation is friendly to both human-reading and machine-processing. Compared with the traditional storage method, the data retrieval speed of the graph data storage method is faster, and it can realize the real-time response of human-computer interaction, so that security analysts can make real-time decisions and promote hunting efficiency.

In this work, we consider constructing knowledge graph based on kernel audit logs and realize agile threat hunting by integrating threat intelligence and expert knowledge. Based on knowledge graph, it is convenient for security analysts to perform pruning and limit search in advance. Moreover, through the rapid summary of massive formatted data, statistics-based anomaly detection without pre-training can be achieved, thereby promoting hunting against unknown threats.

Our contributions are summarized as follows:

- We propose to construct knowledge graph based on kernel audit logs. According to the standard life cycle of KG, we design the construction and application process of KG, which can be applied to actual threat hunting activities.
- We optimize the log dump process. By dumping knowledge into graphs, massive data is efficiently organized, which facilitates subsequent knowledge search and reasoning.
- We promote threat hunting in two phases based on constructed knowledge graph. First, hunters can formulate hypotheses from different semantic levels, including observable IOCs and contextual behavior patterns. Then during the verification of hypotheses, based on efficient graph query and visual output, the hunter can quickly complement the attack paths and eliminate false positives.
- We implement a LAN-wide hunting system called THKG(Threat Hunting based on Knowledge Graph). Evaluations based on the dataset of DARPA TC program show the effectiveness of our hunting system.

RoadMap: The rest of this paper is organized as follows. In Related work section, we describe some works in this field. In Threat model and system design section, we explain the threat model of our work and introduce the overall design of our hunting system. Then, we describe the construction process of knowledge graph in detail in Construction of knowledge graph section. Next in Knowledge application to threat hunting section, based on constructed knowledge graph, we present how to apply generated knowledge graph to perform threat hunting and demonstrate the effectiveness of our hunting system under the adversarial engagement designed by DARPA. Performance evaluation section shows the performance evaluating results of our hunting platform, including time cost, disk usage and query delay. Finally, we summary and discuss our work in Conclusion section.

Related work

Cyber threat hunting and APT detection

In threat hunting, Shu et al. (2018) proposes "threat intelligence computing" which models threat hunting as a graph computation problem and designs a domain-specific graph language with interactive visualization support and a distributed graph database. Milajerdi et al. (2019) models threat hunting as an inexact graph pattern matching problem between the query graph constructed out of threat intelligence and the provenance graph constructed out of kernel audit logs. Mavroeidis and Jøsang (2018) presents an automated threat assessment system based on Sysmon logs and a threat intelligence ontology, and augment cyber defensive capabilities through situational awareness, prediction, and automated courses of action. However, it is possible to miss unknown threats by focusing on threat intelligence only. Our work combines a variety of prior knowledge to achieve a more complete threat hunting.

As for APT detection, SLEUTH (Hossain et al. 2017) uses tag-based techniques to detect APT attacks and reconstructs the attack steps using main-memory based provenance graph. Further more, HOLMES (Milajerdi et al. 2019) uses APT kill-chain as the pivotal reference to produce a detection signal that indicates the presence of an APT campaign. NODOZE (Hassan et al. 2019) tries to combat threat alert fatigue by using contextual and historical information of generated threat alert. It returns the compressed minimum dependence graph related to attacks by prioritizing the paths based on anomaly score. Besides, UNICORN (Han et al. 2020) presents an anomaly-based APT detector using graph sketching and further improves detection capability by a novel modeling approach to understand long-term behavior as the system evolves.

However, Poirot, HOLMES and NODOZE all need to set the appropriate threshold or learn the abnormal behavior by training. In the actual application scenario, the user's behavior is complex and has lots of uncertainty, and it is hard to distinguish between the unusual behavior and threat behavior, so the threat validation is indispensable. Our work combines threat detection and threat verification, and does not make assumptions or set thresholds for the machine reasoning process, but limits the search through the preparation of accurate patterns, and completes accurate and fast pruning through expert interaction.

Attack forensics and Kernel audit

A lot of works focus on attack forensics based on kernel audit logs. Backtracking King and Chen (2003) is one of the first works to backtrack the root cause of intrusions. PRIOTRACKER (Liu et al. 2018) proposes a backward and forward causality tracker that automatically prioritizes the investigation of abnormal causal dependencies based on the rareness and topological features of system events. In order to make forensics analysis based on kernel audit more practical, many works (Lee et al. 2013; Xu et al. 2016; Ma et al. 2016; Kwon et al. 2018; Hossain et al. 2018) try to reduce kernel audit logs by compression and other reduction methods. Other works (Lee et al. 2013; Ma et al. 2016, 2017) try to achieve more refined forensics and information flow tracking. In addition, there are some works trying to achieve more precise provenance tracking by record and replay (Ji et al. 2017, 2018). OmegaLog (Hassan et al. 2020) proposes the notion of universal provenance, which bridges the semantic gap between system and application logging contexts. UISCOPE (Yang et al. 2020) combines low-level causality analysis with high-level UI elements and event analysis to achieve an accurate and visible attack investigation system for GUI applications.

As for the application of kernel audit in real-time scenario, SAQL (Gao et al. 2018a) proposes a novel stream-based query engine to identify abnormal behavior among real-time event feed, including rule-based anomalies, time-series anomalies, invariant-based anomalies, and outlier-based anomalies. SAQL is built for stream-based anomaly detection, thus orthogonal to our work. Similarly, AIQL (Gao et al. 2018b) proposes a query system built upon existing monitoring tools and databases, which is optimized for timely attack investigation. But they are based on relational database and ignore that graph query language itself can express rich anomaly detection logic. A graph query needs to be decomposed into several SQL statements. Compared with traditional relational database, graph query language highlights the relationship between data, which is more suitable for inference of context causality in attack traceability.

Security knowledge graph

There is a plethora of ontological approaches related to cyber security focusing on specific sub-domains, such as threat intelligence, malware detection, vulnerability analysis and more (Gao et al. 2018b; Obrst et al. 2012; Oltramari et al. 2014; Grégio et al. 2014, 2016; Mavroeidis and Bromander 2017; Huang et al. 2010). Above-mentioned security knowledge graph takes threat intelligence and external knowledge as the main body. Our work builds the knowledge graph based on the audit data generated by the enterprise network itself, which can realize agile threat hunting together with the above mentioned knowledge graph.

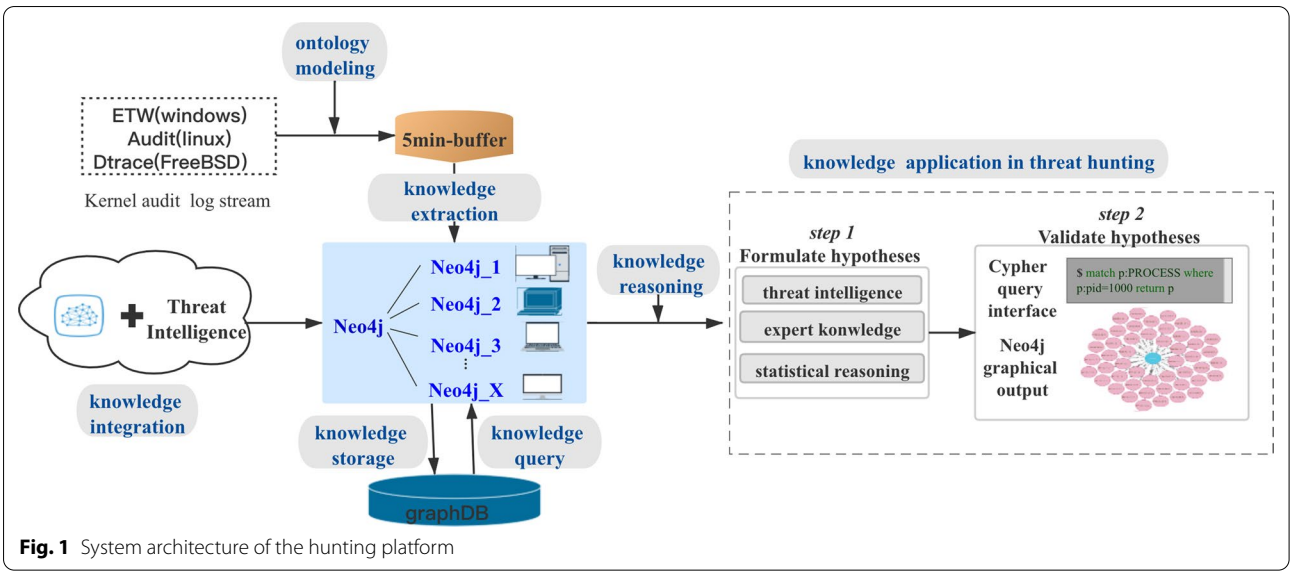

Fig. 1 System architecture of the hunting platform

Threat model and system design

Threat model

We follow the threat model of previous works (Hossain et al. 2017; Liu et al. 2018; Hassan et al. 2019; Pasquier et al. 2018), i.e, the underlying operation system and the back-end graph database are in our trusted computing base (TCB). Kernel-level attacks that deliberately compromise audit frameworks are beyond our scope. Secure and efficient log storage is always ensured by a dedicated and hardened log server.

We do not consider the attacks using side channels that do not go through the syscall interface thus cannot be captured by kernel audit frameworks. We also make the assumption that the adversaries do not have physical access to the host, but need to gain remote access by installing malware on the targeted system, exploiting a running process or injecting a backdoor.

System design

The framework of THKG is shown in Fig. 1. First, THKG collects real-time log streams from different data sources based on kernel audit frameworks. Common kernel audit frameworks include System Audit Framework under Linux, Event Tracing Framework under Windows and Dtrace of FreeBSD. Then according to the standard life cycle of knowledge graph, we design ontology modeling, knowledge extraction, knowledge storage and query, knowledge integration and knowledge reasoning to construct knowledge graph based on kernel audit records. THKG runs an independent instance of Neo4j for each host (from 1 to X) in the LAN, opens the bolt port to receive the query instructions from security analysts and returns the results. Next section introduces the realization of the construction in detail.

Then, the constructed knowledge graph will be applied in actual threat hunting in two stages. In the stage of formulating hypotheses, we can design hypotheses from three different aspects including external threat intelligence, malicious behavior patterns and statistical inference. In the hypotheses verification stage, through Cypher's query interface and Neo4j's graphical output, an agile and interactive hunting process can be achieved.

Construction of knowledge graph

The standard life cycle of knowledge graph includes ontology modeling, knowledge extraction, knowledge storage and query, knowledge integration, knowledge reasoning and knowledge application. We refer to the above procedure to build knowledge graph based on kernel audit records then apply them to subsequent threat hunting.

Ontology modeling

The process of ontology modeling is to construct an ontology to describe the target knowledge according to the applicable domain. Specifically, it is to explicitly express the domain entity category, entity attributes, domain semantic relationships, and relationships between semantic relationships.

We focus on threat hunting based on kernel audit logs. Therefore, the final ontology we select is shown in Fig. 2, in which root nodes represents entity, and leaf nodes represent it's attributes (only entities are listed). This ontology is compatible with the Common Data Model (CDM) of the DARPA TC program. More datailed information

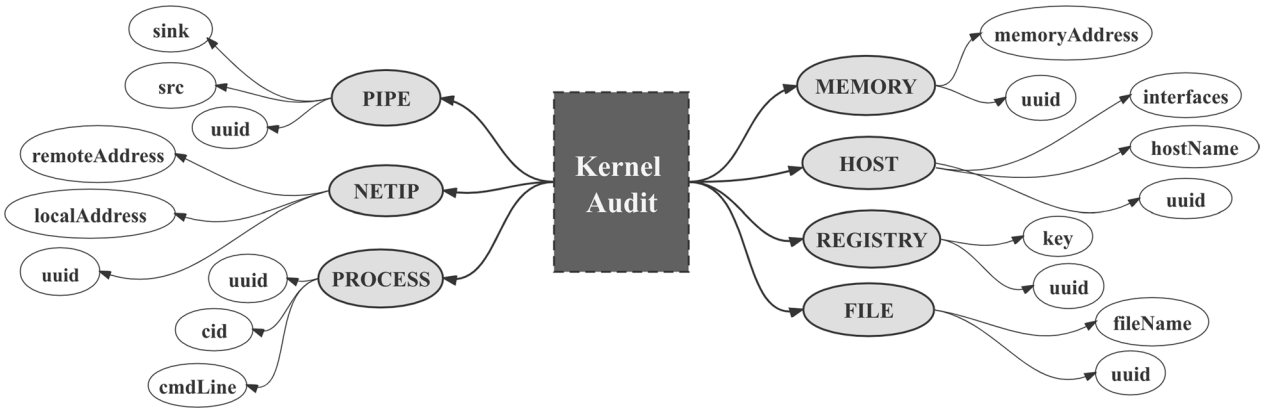

Fig. 2 Ontology of kernel audit for threat hunting

can be refered in the CDMF documentation (Torrey 2020). In addition, another more well-known ontology on provenance data is W3C's PROV-DM (Belhajjame 2013), which is used to form assessments about the quality, reliability or trustworthiness of data. Therefore, it is significantly different from the ontology we build here, which is specifically used for attack investigation based on kernel audit logs.

For the relationship between entities, we specify it as the unified system call name. The attributes include the thread number, startTime and endTime, and the direction is consistent with the actual information flow. For example, the event ⟨ *proc [pid, cmdLine], read [0, Ts, Te], file [fileName]* ⟩ is equivalently converted to the following path in KG:

path=(proc:PROCESS) ⟨-[EVENT_READ{tid=0, start Time=Ts, endTime=Te}]- (file:FILE)

Knowledge extraction

Knowledge extraction is completed by a real-time parsing program (forwarding agent) running in user space. After the operational system throws out audit records in real time, the parser program extracts entities, entity relationships and necessary attributes, and forwards them to remote log server as shown in Fig. 1.

In actual observation, the operational system often invokes a large number of repeated system calls within a short period of time, such as reading a large file or continuous network communication, which result in a large number of repeated events. These events have exactly the same ⟨*sub, op, obj* ⟩ except timestamps, and they have the same effect on system dependency diffusion. If the original audit logs are forwarded directly, it will cause a heavy network load. To reduce the network load, we set up a buffer queue of 5 min to complete the compression before forwarding to the server.

The specific implementation is to create a Map table ⟨ Key *key*, Event *event*⟩ from key = ⟨*src, op, obj*⟩ to *event*. For the newly occurring event *e*, if there is a corresponding Map item matching it, the endTime of the *event* in the map table will be changed to the timestamp of *e*, then *e* will be discarded; if there is no match, a new map entry will be created. Every 5 minutes, platform forwards the buffer to the log sever and clears the buffer and Map table.

Under this compression scheme, according to the results in Performance evaluation section, about 75% of duplicate records can be removed, and eventually each host generates approximately 300–1000M of storage per day. At the same time, the setting of the buffer also bridges the speed gap between the raw audit logs and subsequent processing.

Knowledge storage and knowledge query

Knowledge storage

The emergence of graph databases brings a new option for storing kernel audit logs. Graph database is a database constructed based on graph structure, which uses nodes, edges and attributes to represent and store data. At the same time, graph databases allow the use of semantic queries to find specific paths. Compared with traditional relational database, graph database highlights the relationship between data. In this way, it is very convenient for any data node to retrieve the data node associated with it.

Among them, Neo4j (Platform 2021) graph database is one of the most popular graph databases, which supports rich graph language query syntax to complete complex data retrieval functions. To reduce technical complexity and deployment difficulty, we choose the open-source Neo4j as underlying storage and analysis engine. Each day, log is individually archived for each

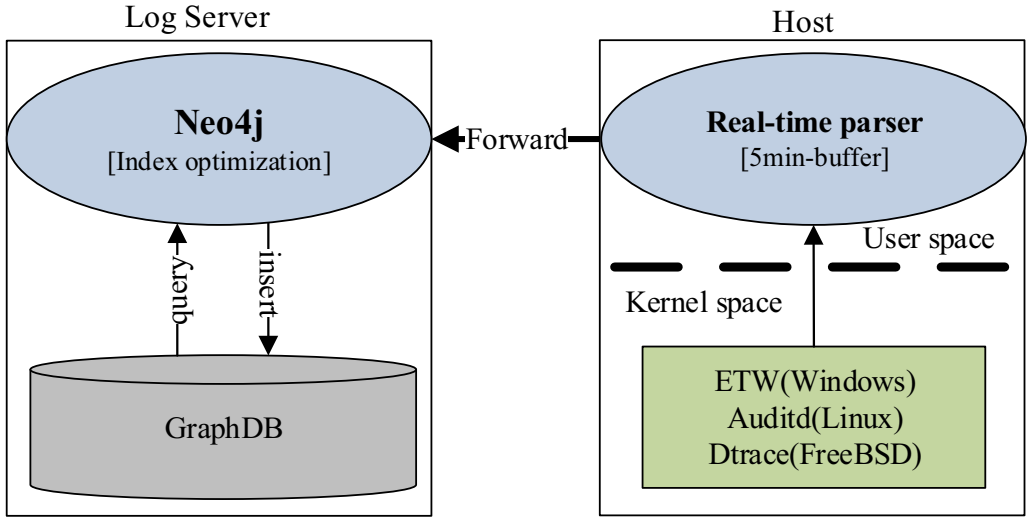

Fig. 3 Real-time kernel audit log forwarding

host and forwarded to the log sever as shown in Fig. 3, which makes full use of the temporal and spatial locality of the kernel audit logs and improves storage and query efficiency.

When the host load is heavy, we can also use Neo4j's index optimization to speed up storage, which can significantly improve the real-time performance of log dump. For related discussion, see Performance evaluation section.

Knowledge query

Cypher is a query language designed specifically for Neo4j, just as SQL for relational databases. So our knowledge query is done by Cypher language, which allows for expressive and efficient query of a property graph. The basic query syntax can be summarized as follows:

```
match variable:LABEL{property_1:value_1} where variable.property_2 = value_2 return variable
```

Among them, *variable* represents goal entities or relationships that meet the conditions; *LABEL* corresponds to the entity category in above ontology; the clause *{property_1 : value_1 }* indicates the restriction on attributes, which can also be written as *property_1* = *value_1*, and it can appear in braces after *LABEL*, or in the *where* clause; the *return* clause is used to return the result set; in addition, there are some built-in aggregation functions such as *distinct, count, mean*, etc.

Next, we will briefly introduce how to write scripts to query entities, relationships or paths.

(1) Query entities

```
Query the process with pid = 1000:
match (p: PROCESS) where p.pid = 1000 return p
Query web server address:
match (n:NETIP) where split(n.remoteAddress,':')[1] = "80" return n
Returns files with "passwd" in fileName:
match (f:FILE) where f.fileName = ~". * passwd. *" return f
```

(2) Query relationships

Query the list of processes that send information to external addresses, and sort by the number of IP addresses.

```
match (p:PROCESS) -[r: EVENT_SEND]-> (n:NETIP)
return p, count(n) as Num order by Num desc
```

(3) Query path

Returns the path that the process interacts with the network address. The asterisk in square brackets indicates unlimited length.

```
match path = (p: PROCESS)-[*]-⟩ (n: NETIP) return path
```

Knowledge integration

Knowledge integration includes two aspects. On the one hand, it is the fusion of different KG instances on various hosts in the enterprise to detect possible lateral penetration. On the other hand, it refers to the fusion with third-party threat intelligence or knowledge graphs of other security sub-domain for rapid detection of IOCs.

The fusion between different KGs in the LAN is mainly based on the IP address[1]. For example, the monitoring record on host A shows that it has accessed NETIP: "localAddress": "10. *. *. 73: 16006", "remoteAddress": "10. *. *. 130: 3389", recorded as host B. In order to correlate this event with host B, we need to perform the following Cypher query in the KG of host B:

```
match path = (n:NETIP {localAddress: "10. *. *.130:3389"})-[*]-() return path
```

Integrating third-party threat intelligence, in short, is to filter the IPs accessed by the host, registry, file name, command line and other information, and match them with threat intelligence libraries such as virusTotal (Virustotal 2020), ThreatMiner, ThreatBook, etc., to quickly find traces of attacks.

Knowledge reasoning

Knowledge reasoning means deriving new knowledge from reasoning based on existing knowledge. The traditional rule-based reasoning method mainly uses simple rules or statistical features to reason on knowledge graph. Fast and customized query of Cypher allows us to get multiple statistical features.

For example, the security analysts can choose to compare with the overall behavior of all process instances on the host, such as TOP-N portscan and TOP-N user documents accessing. Or compare with different instances of the same program, for example, by comparing the syscall distribution of different instances of Firefox to filter potentially compromised processes:

TOP-N processes of portscan:

```
match (proc:PROCESS)-[]-(net:NETIP) with distinct(net.remoteAddress) as netip, proc as proc return distinct(split(netip,':')[0]+':'+split(netip,':')[1]+':'+split(netip,':')[2]) as C_IP, count(*) as num, proc order by num desc limit 5
```

TOP-N processes of accessing user documents:

```
match (proc:PROCESS)-[:EVENT_READ]-(target:FILE) where (toLower(target.fileName)=~'.*\\.doc.*' or toLower(target.fileName)=~'.*\\.xls.*' or toLower(target.fileName)=~'.*\\.rtf.*' or toLower(target.fileName)=~'.*\\.pdf.*') return count(target) as num, proc order by num desc limit 5
```

Compare different process instances of Firefox:

```
match (browser:PROCESS)-[r]-() where browser.cmdLine =~".*firefox.*" return distinct(type(r)), count(*), browser.cid
```

In addition, security analysts can compare the behavior of the same long-term process (services) in different time periods to identify attacks. It is worth noting that our anomaly filtering does not need to explicitly set thresholds, but performs filtering by direct comparison with most situations on the host.

Although TOP-N also needs to specify a threshold, this threshold is a weak one, which has nothing to do with the specific attack scenario, but is related to the hunter's Receiver Operating Characteristic (ROC) and free time. Generally, the range of TOP5-10 works well (Next section gives the hunting case). Based on detailed records of all user behaviors and the efficient summary of massively formatted data, we can establish a baseline of normal behaviors at the same time as the query, avoiding the pre-training phase of traditional anomaly detection methods. And by further combining expert knowledge, security analysts can quickly remove false positives.

Knowledge application to threat hunting

The constructed KGs can be applied in actual threat hunting. We consider three aspects to design hypotheses: designing hypotheses based on IOCs by integrating threat intelligence; implementing rule-based detection by embedding expert knowledge into the Cypher pattern; and filtering suspicious objects based on statistical reasoning to trigger follow-up investigations. Then with the help of the visual output of Neo4j and the friendly human-to-machine interface provided by Cypher, we can quickly perform hypothesis verification.

[1] The native Linux Audit cannot accurately extract the 5-tuples of the network connection and needs auxiliary information from other sources such as lsof (Gehani and Tariq 2012).

Table 1 APT cases of dataset

Log file	APT cases included
Win_1(five)	None
Win_2(five-2)	Attack_1: Firefox Backdoor
	Attack_2: Browser Extension
	Attack_3: Phishing E-mail
Linux_1(trace-1)	Attack_4: Phishing E-mail
	Attack_11: Pine Backdoor
Linux_2(theia-6r)	Attack_5: Firefox Backdoor
	Attack_6: Browser Extension
	Attack_7: Phishing E-mail
	Attack_8: Phishing E-mail
BSD_1(cadets)	Attack_9: Nginx Backdoor
BSD_2(cadets-1)	None
BSD_3(cadets-2)	Attack_10: Nginx Backdoor

To evaluate the effectiveness of our hunting platform, we use a dataset of DARPA TC program red-team vs. blue-team adversarial engagement (Torrey 2020) which contains different OS platforms(Windows, BSD, and Linux) with kernel-audit enabled. During the engagement, benign background traffic was run continuously with the attacks from the red team. This dataset covers APT cases of nation state and several common threats, and contains a ground truth file detailing the attack process for verification. Table 1 lists all the APT cases contained in the dataset. For ease of expression, we have relabeled the files in the original dataset. The original file numbers are shown in the right parenthesis.

Next, we will explicitly introduce three different ways of formulating and validating hypotheses to hunt threats.

Formulating hypotheses based on threat intelligence

With the help of external threat intelligence, or shared hunting scripts from the open source community, we can quickly design our hunting process based on IOCs, such as file name, hash, IP address, command line, etc. For example, Milajerdi et al. (2019) designs hunting processes for known APT cases based on public APT reports. Other resources available include hunting scripts shared in the MITRE Cyber Analytics Repository (CAR) (MITRE 2020), Threat Hunting Project (DavidJBianco 2019), and Sigam Project (Patzke 2017), all of which are ready-to-use threat intelligence.

Next, based on the knowledge graphs generated from the DARPA dataset, we launch our hunting campaign by translating the common shared hunting scripts into Cypher language. Common shared hunting script formats include Pseudocode, EQL for elasticsearch and Splunk native query language.

Converting Pseudocode script into Cypher

CAR-2013-08-001 (Execution with schtasks (MITRE 2013)) describes a hunting script written in Pseudocode to detect the execution of schtasks commands. We first convert it to Cypher format, then perform the corresponding hunting based on the DARPA dataset.

Original Pseudocode script:

process = search Process:Create schtasks = filter process where (exe == " schtasks.exe") output schtasks

Converting to Cypher:

*match (proc:PROCESS) where proc.cmdLine=~".*schtasks.*" return distinct(proc.cmdLine)*

Hunting on the dataset hits the following results in Win_2 as shown in Table 2.

Among them, the results in bold are very suspicious. Further investigation confirms that they are part of the APT case of Phishing E-mail.

Converting EQL script into cypher

CAR-2014-05-002 (Services launching Cmd (MITRE 2014)) is used to detect cmd commands executed by the

Table 2 Hunting results in win_2

	proc.cmdLine
1	"schtasks.exe"
2	"schtasks.exe/change/tn 'Microsoft/Office/Office Automatic Updates'/enable"
3	"schtasks.exe/change/tn 'Microsoft/Office/Office ClickToRun Service Monitor'/enable"
4	**"schtasks/create/tn WindowsUpdate/tr 'powershell -nop -ep bypass -encodedCommand KABOAGUAdwAtAE8AYqYwB0A⋯YAIAAt'"**
5	**"schtasks/create/tn WindowsUpdate-tr 'powershell.exe -nop -ep Bypass -encodedCommand KABOAGUAdwAtAEYUAYwB0A⋯AuADY'"**

service program. Similarly, we translate it into Cypher and perform the hunting on the dataset.

Original EQL hunting script:

```
process where subtype.create and (process_name == "cmd.exe" and parent_process_name == "services.exe")
```

Converting to Cypher:

```
match (parent:PROCESS)-[:EVENT_FORK]-(proc:PROCESS) where parent.cmdLine = ~".*services.*" and proc.cmdLine = ~".*cmd.exe.*" return proc
```

Hunting on the dataset hits no targets.

Converting Splunk script into Cypher

This Splunk script is used to detect malicious PowerShell based on Sysmon, which is an ETW-like auditing tool developed by Microsoft.

Hunting hits no targets. However, our comparison with Ground Truth during the verification of the previous hunt found that the attacker actually ran an encrypted powershell command *{(New-ObjectNet.WebClient) .downloadfile ('http: //*.*.*. * / update.ps1 ')* (see the obfuscated string in Table 2). It shows that threat hunting based on single IOC indicator will cause false negatives when the attackers hide themselves, which requires human experts to perform pruning.

Original Splunk hunting script:

```
index=sysmon SourceName ="Microsoft-Windows-Sysmon" EventCode ="1" (powershell.exe OR cmd.exe) | eval CommandLine2 =replace(CommandLine,"['+\"\^]","")| search (Image ="*\\powershell.exe" OR Image ="*\\cmd.exe") CommandLine2 ="*WebClient*" CommandLine2 ="*DownloadFile*"
```

Converting to Cypher:

```
match (proc:PROCESS) where proc.cmdLine =~".*powershell.*" and proc.cmdLine =~".*WebClient.*" and proc.cmdLine =~".*DownloadFile.*" return proc
```

Defining malicious behavior pattern by embedding expert knowledge

In this section, we try to formulate malicious behavior pattern based on a series of operations that attackers must complete in order to implement specific goals. We call this series of operations as "Behavior", and the semantic level is between the observable system object and the "Techniques" of MITRE ATT & CK (Corporation 2015). For example, in order to implement the technique of "Spearphishing Attachment" indexed as T1193, the host must download the attachment and trigger the program to execute after accessing the receiving port of the mail server. According to this assumption, we start our hunting for fishing e-mail. The following shows the hunting process of Attack_4 as an example.

Formulating hypothesis

(1) Collect common receiving ports of mail server through search engines, such as 109 for POP2, 110/995 for POP3 and 143/993 for IMAP.
(2) Filter processes that access the mail server's receiving port.

```
match (p)-[]-(n:NETIP) where split(n.remoteAddress,':')[1] in ['109','110','995','143','993'] return distinct(p.cmdLine)
```

The returned results are as follows:*/usr/lib/thunderbird/thunderbird, /tmp/ztmp, pine and tcexec*. According to the search engine, thunderbird and pine are common email clients, so the remaining programs are actually suspicious. But this time we hunt by patterns thus we just ignore them.

(3) Further filter mail clients that write files to disk.

```
match (p)-[]-(n:NETIP) where split(n.remoteAddress,':')[1] in ['109','110','995','143','993'] with distinct(p) as mail match PATH = (mail)-[:EVENT_WRITE]->(exe:FILE) return distinct(exe.fileName), mail.cmdLine
```

The filtered results are shown in Table 3. Combined with expert knowledge, only the results in bold need to be investigated.

Validating hypothesis

(1) Investigate the execution caused by above suspicious files.

Table 3 Mail clients writing to disk

exe.fileName	mail.cmdLine
/home/admin/.pine-debug1	pine
/dev/null	bash
/home/admin/.pine-debug1	bash
/home/admin/.bash_history	bash
/home/admin/.pine-debug1	./pine
/tmp/tcexfil	**./pine**
/tmp/tcexec	**./pine**

$$match\ p = (proc)\text{-}[r{:}EVENT\_EXECUTE]\text{-}\rangle(exe)\ where\ exe.cmdLine = \sim\text{'.*tcex.*'}\ return\ proc.cmdLine,\ exe$$

Table 4. shows all the executions caused by the suspicious files. Based on expert knowledge and further pruning, we only need to investigate the process in bold.

(2) Investigate file operations, process fork and execution, and network access about above suspicious process. In the returned results in Fig. 4, the *'sh'* command is usually used to open a remote control channel on the victim's computer, and the *'uname'* command is used to probe information about the victim's computer, which prove that it is a phishing attachment.

Besides, we find that in addition to performing portscan, suspicious attachments continue communicating with the IP of "162.66.239.75:80" during the above operations (Fig. 5), so it can be confirmed as the IP of Command and Control (C&C) server. At this point, the complete hunting campaign is over.

Table 4 Suspicious execution

cmdLine	exe
bash	cmdLine:chmod +x tcexec,uuid:7890EEEB-B6FA-AFBD-D5BA-A2422F30BF99,cid:26541
bash	cmdLine:python3 command-not-found – tcexec,uuid:63901CB6-67CD-9C7D-2476-D0F75FEB22C5,cid:26543
./pine	**./cmdLine:tcexec,uuid:0BF26B23-2DE5-B70A-45F7-64BE377293F3,cid:27201**

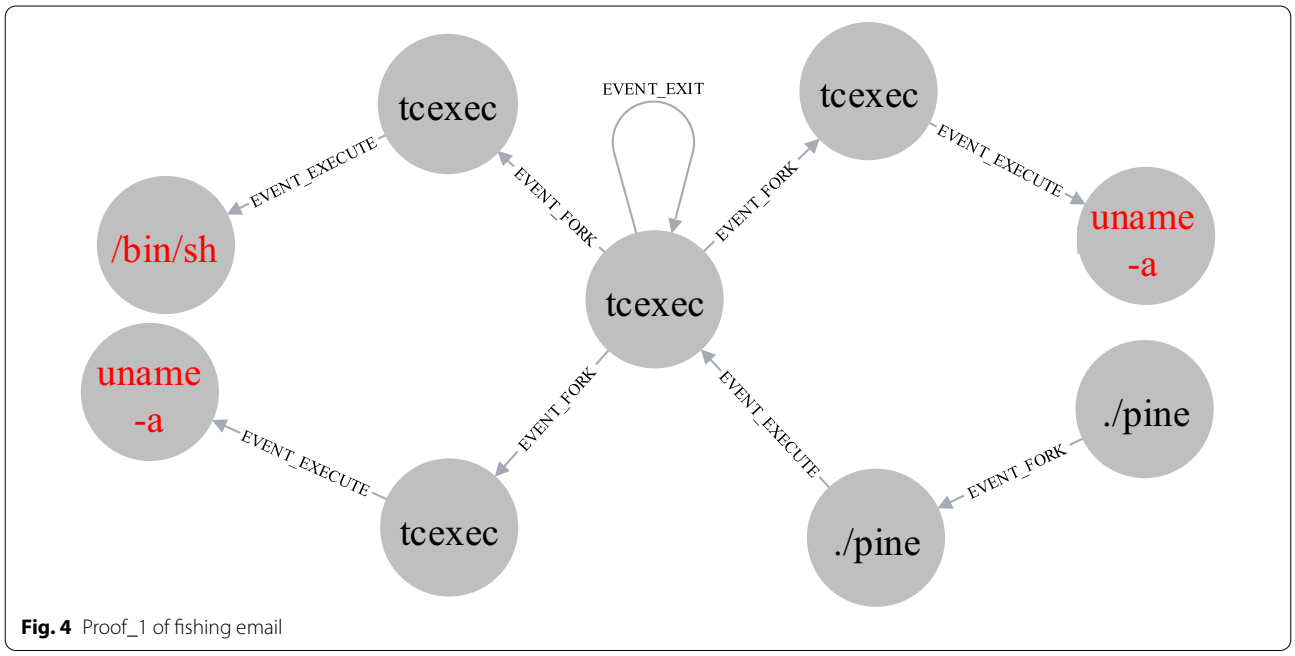

Fig. 4 Proof_1 of fishing email

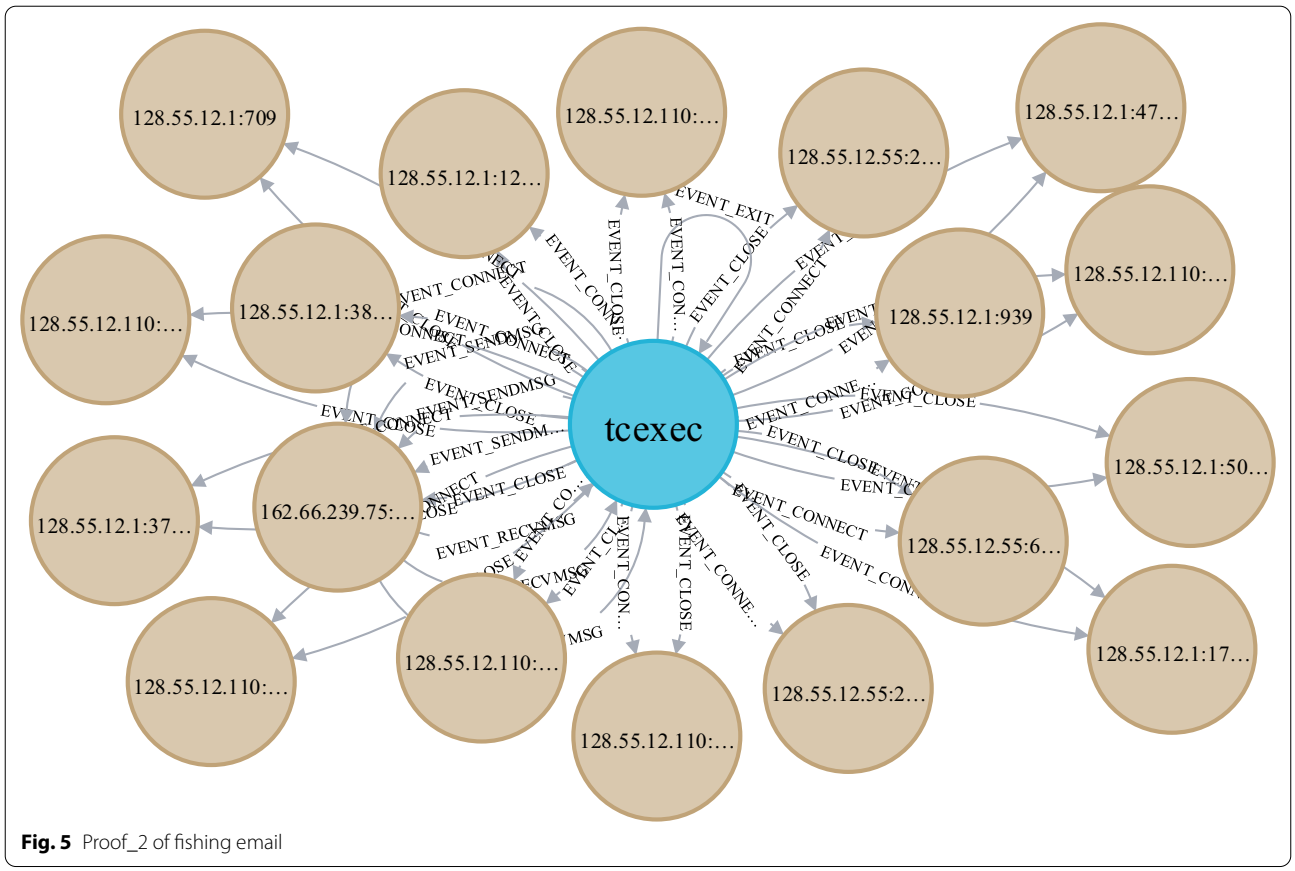

Fig. 5 Proof_2 of fishing email

Filtering anomaly by statistical reasoning

We introduce three examples to show how to find abnormal behaviors by statistical reasoning.

Filtering suspicious Portscan

First, we run following script to query the Top-5 processes of portscan on the Linux_1 dataset.

```
match (proc:PROCESS)-[]-(net:NETIP) with distinct(net.remoteAddress) as netip, proc as proc return distinct(split(netip,':')[0] +':' +split(netip,':')[1] +':' +split(netip,':')[2]) as C_IP, count(*) as num,proc order by num desc limit 5
```

The returned results are shown in Table 5.

The first two processes are very suspicious, and subsequent investigation proves that they are part of the Attack_4 and Attack_11.

Reading user documents

Running the hunting script on the Win_2 dataset returns the following results in Table 6.

```
match (proc:PROCESS)-[:EVENT_READ]-(target:FILE) where (toLower( target.fileName) =~'.*\\.doc.*' or toLower( target.fileName) =~'.*\\.xls.*' or toLower( target.fileName) =~'.*\\.rtf.*' or toLower( target.fileName) =~'.*\\.pdf.*') return count(target) as num, proc order by num desc limit 5
```

Table 5 Suspicious portscan

C_IP	num	proc
128.55.12	**47824**	**{cmdLine:tcexec,cid:27201}**
128.55.12	**721**	**{cmdLine:/tmp/ztmp,cid:19482}**
128.55.12	108	{cmdLine:sshd,cid:1810}
128.55.12	15	{cmdLine:avahi-daemon,cid:1170}
216.66.26	5	{cmdLine:firefox,cid:31814}

Table 6 Counting of reading user documents

num	proc
49	{cmdLine:C:/WINDOWS/Explorer.EXE,cid:5172}
17	**{cmdLine:"C:/Program Files/Mozilla Firefox/firefox.exe", cid:9968}**
14	{cmdLine:"C:/WINDOWS/system32/SearchProtocol-Host.exe", cid:2560}
9	{cmdLine:"C:/ProgramData/Microsoft/Windows Defender/platform/4.12.17007.18022-0/MsMpEng.exe",cid:3160}
8	{cmdLine:"C:/Program Files (x86)/Microsoft Office/Office15/EXCEL.EXE"/dde,cid:4328}

Except for the firefox process marked in bold, the other programs open user files under normal circumstances, so we investigate the firefox process, and subsequent hunting proves that it belongs to Attack_1.

Filtering suspicious programs that access regular ports (such as port 80)

```
match (p)-[]-(n:NETIP) where split(n.remoteAddress,':')[1] in ['80'] return distinct(p.cmdLine)
```

Running the hunting script on the dataset BSD_1 returns the following results: *wget, nginx, links and vUgefal*. The result of "vUgefal" is very suspicious. After follow-up investigation, it hit Attack_9. The same situation also exists in Attack_3 in the Win_2 dataset, which can directly hit the malicious powershell program. Both Attack_5 and Attack_6 of the Linux_2 dataset hit suspicious programs accessing port 80.

Performance evaluation

We evaluate the performance of THKG from three aspects: time cost, space occupation and query delay.

Time-cost

Real-time requirements refer to whether our hunting platform can dump the continuously generated kernel audit logs to the graph database in time under different host loads. Since we set a 5-min buffer before forwarding, it refers to whether the hunting platform can insert the received event buffer into the graph database within 5 minutes. We select a Windows PC and a Linux Server for evaluation, and count the total number of compressed events and the time spent inserting into graph database every 5 minutes.

The hardware configuration of the two monitored hosts is shown in Table 7. Among them, Window PC is for normal user office. In order to conduct stress tests, we open many video websites and open local videos as well between 15:30 pm and 17:00 pm for injecting load on Windows PC. Linux Server is used to provide computational support for team research, which often keeps multiple people online and continuously runs heavy scientific tasks, such as training neural networks and parsing massive logs.

First we test the real-time performance of the two platforms without any storage optimization. The final statistical results are shown in the Figs. 6 and 7.

It can be seen that without any optimization, Windows PC can still meet the real-time requirements even under a heavy load. Even though the insertion time will gradually increase with the accumulation of events. However, the real-time performance of Linux Server deteriorates sharply when the load rises, making it impractical. Therefore, we accelerated the log dump of Linux Server based on Neo4j's index optimization. Results after optimization are shown in Fig. 8.

After optimization, the insertion time is drastically reduced from tens of minutes to seconds, and the time consumption is only related to the number of inserted events, regardless of the size of the database. However, Neo4j's index optimization will cause extra storage space. Therefore, in actual deployment, one can choose only to optimize for servers with high load. Regardless, the above evaluation shows that our hunting platform can meet the real-time requirements under any load condition.

Table 7 Hardware configuration of monitored hosts

Platform	Hardware configuration	
Windows PC	CPU	Intel(R) Core(TM) i7-6700 @ 3.40GHz
	Memory	12.0GB
	OS	64 bit Windows7 ultimate SP1
	Purpose	Personal office
Linux Server	CPU	Intel(R) Xeon(R) E5-2620 v2 @ 2.10GHz
	Memory	64.0GB
	OS	64 bit Ubuntu 16.04.6 LTS
	Purpose	Team research platform

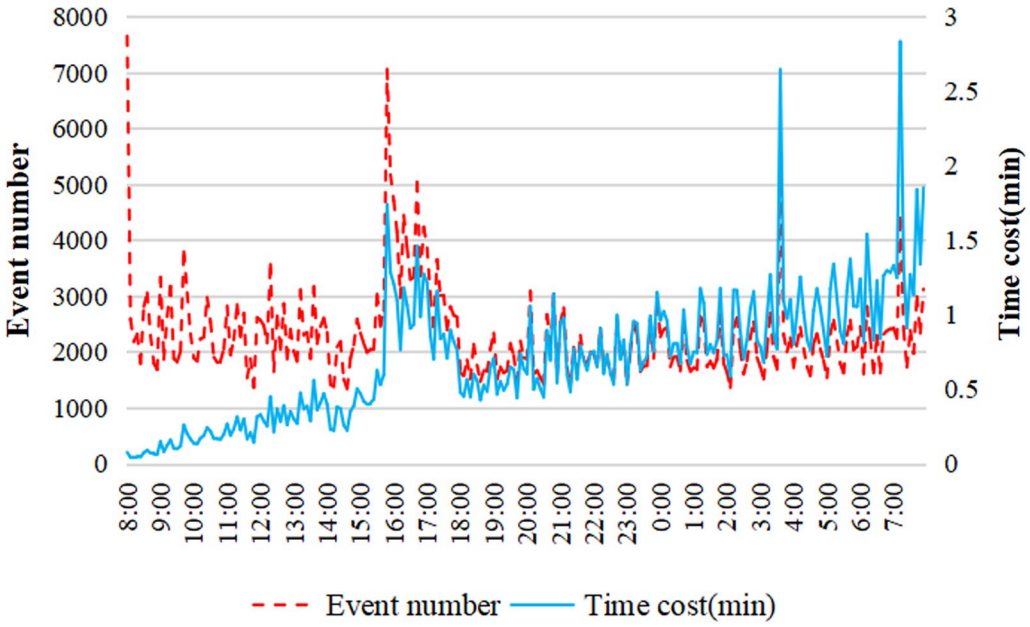

Fig. 6 Time cost (min) for Windows PC without optimization

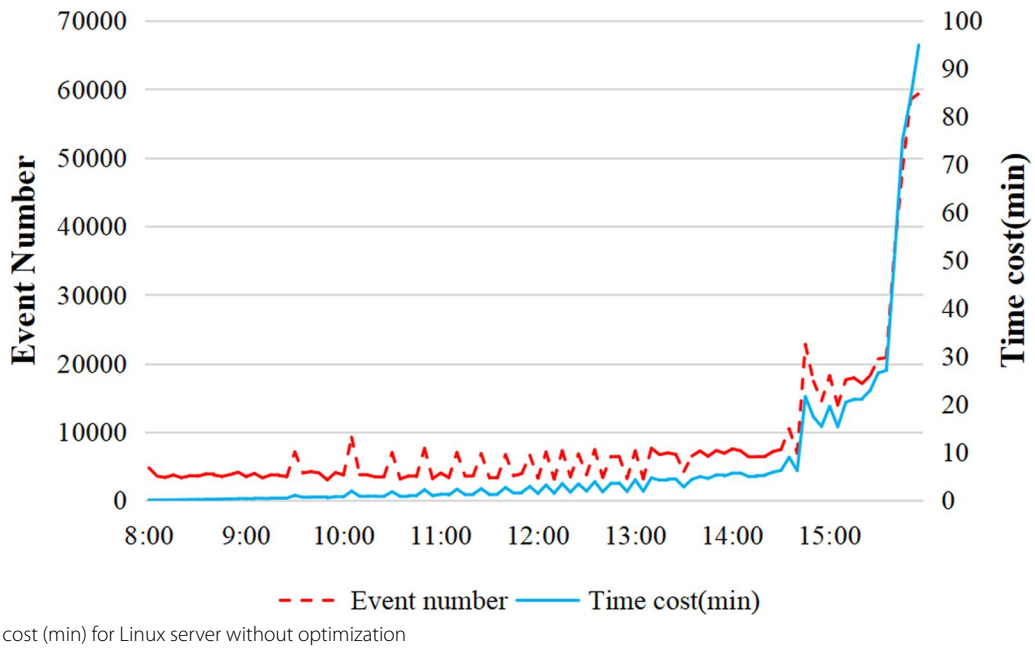

Fig. 7 Time cost (min) for Linux server without optimization

Space occupation

The space occupation includes statistics of the total compression rate of the events and the disk space occupied by the graph database. The compression rate refers to the total number of events after compression divided by the number of original events. The smaller compression rate represents the better compression effect. The statistics in Table 8 are mainly based on the offline DARPA dataset, and also include the results of the 1-day online Windows PC and Linux Server.

In addition, we tested the compression rate and time cost under different buffer time. The results are shown in Fig. 9. According to the statistical results, the compression rate declines as the buffer time increases because

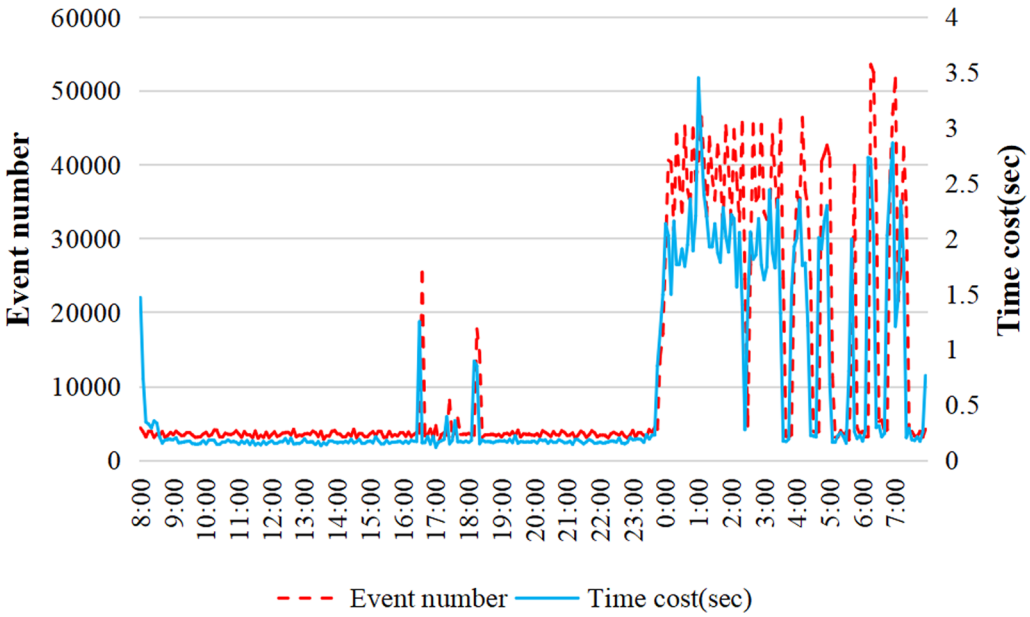

Fig. 8 Time cost (s) for Linux server after optimization

Table 8 Space occupation of different dataset

Platform	Size of log file (MB)	Original event number	Compressed event number	Size of graphDB (MB)	Compression ratio of size (%)	Compression ratio of event number (%)
Win_1	15,114	21,049,902	2,979,782	1798	11.90	13.66
Win_2	203,582	256,621,363	40,246,958	24,610	12.09	15.25
Linux_1	19,688	21,891,709	10,098,882	11,433	58.07	31.43
Linux_2	45,773	59,590,393	9,633,259	6435	14.06	15.09
BSD_1	11,451	12,904,605	5,421,830	2500	21.83	38.97
BSD_2	18,176	20,551,276	8,250,615	3799	20.90	37.40
BSD_3	6936	7,796,898	3,188,001	1477	21.29	37.80
Win_online_1day	–	3,024,428	553,552	284	–	18.30
Linux_online_1day	–	12,462,348	3,353,144	1332	–	26.91
Average	–	–	–	–	22.88	26.09

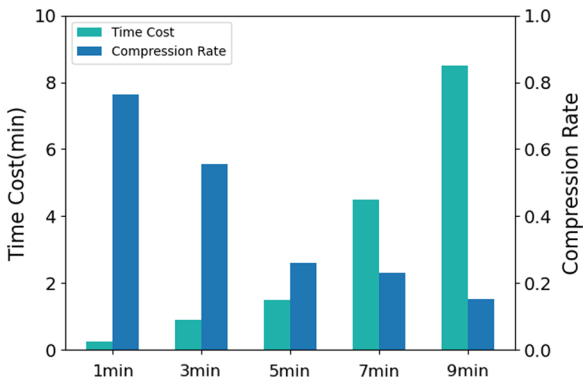

Fig. 9 Compression rate and time cost under different buffer time

more and more events are merged. But when the buffer time exceeds a certain threshold, the rate of decline becomes slow, which is because that the total number of original events is very large. And if the time is set too short, the log compression rate is not enough and the goal of reducing the system load cannot be achieved. For the dump time, with the increase of buffer time, event number grows, and the time cost of log dump continues to increase. Further more, when the buffer time is set too long, it will cause the speed gap between the log storage and the real-time query. Considering real-time performance and compression efficiency comprehensively, the 5-min setting of buffer time is a better choice.

Table 9 Query delay of different complexity by OSQuery and THKG

	Platform	Data size (MB)	Simple node query (ms)	Complex path query (ms)	Statistical summary (ms)
OSQuery	Win_2	24,610	13	12,459	400,993
	Linux_2	6435	237	6257	78,347
	BSD_2	3799	156	2708	39,238
THKG	Win_2	24,610	15	11,031	361,092
	Linux_2	6435	279	6000	73,413
	BSD_2	3799	170	2551	36,378

Query delay

Query delay refers to the latency from the submission of Cypher instructions to the return of results. We select Win_2, Linux_2, and BSD_2 in the DARPA dataset to test simple node queries, complex path queries, and statistical summary of the entire data. In order to verify the query efficiency of our system, we try to compare with the existing attack investigation systems. However, most of the work is not publicly available, so we choose the open source tool OSQuery (Osquery-for-security 2021) as the baseline for our evaluation. OSQuery exposes the operating system as a high-performance relational database that can be queried based on SQL query language. We compare the query efficiency of the two different platforms. The experimental results are shown in Table 9. When we perform simple queries, the query delay does not differ much, but when performing complex queries, the query delay of graph database is more advantageous, which proves the efficiency of graph database for complex association analysis. In fact, it is difficult for traditional relational database to do multi-layer association analysis when the data scale is huge, because the multi-layer aggregation operation of massive data is very complex. In contrast, Cypher query can be more flexible for deep aggregation of data.

Summary

According to the above evaluations, our system meets the real-time requirements under different loads. Based on efficient graph storage, long-term and large-scale KG storage becomes possible. Moreover, the process of real-time log dump organizes massive knowledge in time. With the unified knowledge representation of KGs and the optimization of Neo4j, even hard disk-based graph retrieval can achieve comparable efficiency to memory-based graph retrieval, which lays the foundation for agile threat hunting.

Conclusion

Our work considers the kernel audit records as the knowledge advantage possessed by the defender, and achieves agile threat hunting based on the knowledge graph constructed from kernel audit logs. With the concept alignment of knowledge graph, we successfully integrate all types of knowledge required for hunting, including kernel audit records, expert knowledge and threat intelligence into a single system. The hunting system we designed supports two working modes: real-time monitoring abnormality and historical backtracking causality. It also supports association analysis with different KG instances of LAN hosts and threat intelligence for fast IOCs search. In addition, the security analysts can edit custom query scripts to assist their hunting process or automate routine hunting.

Acknowledgements

We would like to thank the anonymous reviewers for detailed comments and useful feedback.

Author's contributions

All authors contributed to conducting this work and writing this manuscript. All the authors read and approved the final manuscript.

Funding

This work is supported in part by the Industrial Internet Innovation and Development Project "Industrial robot external safety enhancement device" (TC200H030) and the Cooperation project between Chongqing Municipal undergraduate universities and institutes affiliated to CAS (HZ2021015).

Availibility of data and materials

Not applicable.

Declarations

Competing interests

The authors declare that they have no competing interests.

Author details

[1]Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China. [2]School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China.

Received: 27 September 2021 Accepted: 17 January 2022
Published online: 01 June 2022

References

Belhajjame K (2013) PROV-DM: the PROV data model. https://www.w3.org/TR/prov-dm/

Corporation TM (2015) APT&CK. https://attack.mitre.org

DavidJBianco: the threathuting project (2019). https://www.threathunting.net

Gao P, Xiao X, Li D, Li Z, Jee K, Wu Z, Kim CH, Kulkarni SR, Mittal P (2018) {SAQL}: a stream-based query system for real-time abnormal system behavior detection. In: 27th {USENIX} security symposium ({USENIX} security 18), pp 639–656

Gao P, Xiao X, Li Z, Xu F, Kulkarni SR, Mittal P (2018) {AIQL}: enabling efficient attack investigation from system monitoring data. In: {USENIX} annual technical conference ({USENIX}{ATC} 18), pp 113–126

Gehani A, Tariq D (2012) SPADE: support for provenance auditing in distributed environments. In: ACM/IFIP/USENIX international conference on distributed systems platforms and open distributed processing. Springer, pp 101–120

Grégio A, Bonacin R, de Marchi AC, Nabuco OF, de Geus PL (2016) An ontology of suspicious software behavior. Appl Ontol 11(1):29–49

Grégio A, Bonacin R, Nabuco O, Afonso VM, De Geus PL, Jino M (2014) Ontology for malware behavior: a core model proposal. In: IEEE 23rd international WETICE conference. IEEE, pp 453–458

Han X, Pasquier T, Bates A, Mickens J, Seltzer M (2020) Unicorn: runtime provenance-based detector for advanced persistent threats. arXiv preprint. arXiv:2001.01525

Hassan WU, Guo S, Li D, Chen Z, Jee K, Li Z, Bates A (2019) NODOZE: combatting threat alert fatigue with automated provenance triage. In: network and distributed systems security symposium

Hassan WU, Noureddine MA, Datta P, Bates A (2020) OmegaLog: high-fidelity attack investigation via transparent multi-layer log analysis. In: Network and distributed system security symposium

Hossain MN, Milajerdi SM, Wang J, Eshete B, Gjomemo R, Sekar R, Stoller S, Venkatakrishnan V (2017) {SLEUTH}: real-time attack scenario reconstruction from {COTS} audit data. In: 26th {USENIX} security symposium ({USENIX} security 17), pp 487–504

Hossain MN, Wang J, Weisse O, Sekar R, Genkin D, He B, Stoller SD, Fang G, Piessens F, Downing E et al (2018) Dependence-preserving data compaction for scalable forensic analysis. In: 27th {USENIX} security symposium ({USENIX} security 18), pp 1723–1740

Huang H-D, Chuang T-Y, Tsai Y-L, Lee C-S (2010) Ontology-based intelligent system for malware behavioral analysis. In: International conference on fuzzy systems. IEEE, pp 1–6

Ji Y, Lee S, Downing E, Wang W, Fazzini M, Kim T, Orso A, Lee W (2017) RAIN: refinable attack investigation with on-demand inter-process information flow tracking. In: Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, pp 377–390

Ji Y, Lee S, Fazzini M, Allen J, Downing E, Kim T, Orso A, Lee W (2018) Enabling refinable cross-host attack investigation with efficient data flow tagging and tracking. In: 27th {USENIX} security symposium ({USENIX} security 18), pp 1705–1722

King ST, Chen PM (2003) Backtracking intrusions. In: Proceedings of the nineteenth ACM symposium on operating systems principles, pp 223–236

Kwon Y, Wang F, Wang W, Lee KH, Lee W-C, Ma S, Zhang X, Xu D, Jha S, Ciocarlie GF et al (2018) MCI: modeling-based causality inference in audit logging for attack investigation. In: NDSS

Lee KH, Zhang X, Xu D (2013) High accuracy attack provenance via binary-based execution partition. In: NDSS, p 16

Lee KH, Zhang X, Xu D (2013) LogGC: garbage collecting audit log. In: Proceedings of the 2013 ACM SIGSAC conference on computer and communications security, pp 1005–1016

Liu Y, Zhang M, Li D, Jee K, Li Z, Wu Z, Rhee J, Mittal P (2018) Towards a timely causality analysis for enterprise security. In: NDSS

Mavroeidis V, Bromander S (2017) Cyber threat intelligence model: an evaluation of taxonomies, sharing standards, and ontologies within cyber threat intelligence. In: European intelligence and security informatics conference (EISIC). IEEE, pp 91–98

Mavroeidis V, Jøsang A (2018) Data-driven threat hunting using sysmon. In: Proceedings of the 2nd international conference on cryptography, security and privacy, pp 82–88

Ma S, Zhai J, Wang F, Lee KH, Zhang X, Xu D (2017) {MPI}: multiple perspective attack investigation with semantic aware execution partitioning. In: 26th {USENIX} security symposium ({USENIX} security 17), pp 1111–1128

Ma S, Zhang X, Xu D (2016) ProTracer: towards practical provenance tracing by alternating between logging and tainting. In: NDSS

Milajerdi SM, Eshete B, Gjomemo R, Venkatakrishnan V (2019) Poirot: aligning attack behavior with kernel audit records for cyber threat hunting. In: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security, pp 1795–1812

Milajerdi SM, Gjomemo R, Eshete B, Sekar R, Venkatakrishnan V (2019) Holmes: real-time apt detection through correlation of suspicious information flows. In: IEEE symposium on security and privacy (SP). IEEE, pp 1137–1152

MITRE: CAR-2013-08-001: execution with schtasks (2013). https://car.mitre.org/analytics/CAR-2013-08-001/

MITRE: CAR-2014-05-002: services launching Cmd (2014). https://car.mitre.org/analytics/CAR-2014-05-002/

MITRE: MITRE cyber analytics repository (2020). https://car.mitre.org

Obrst L, Chase P, Markeloff R (2012) Developing an ontology of the cyber security domain. In: STIDS, pp 49–56. Citeseer

Oltramari A, Cranor LF, Walls RJ, McDaniel PD (2014) Building an ontology of cyber security. In: STIDS, pp 54–61. Citeseer

Osquery-for-security (2021). https://medium.com/@clong/osquery-for-security-b66fffdf2daf

Pasquier T, Han X, Moyer T, Bates A, Hermant O, Eyers D, Bacon J, Seltzer M (2018) Runtime analysis of whole-system provenance. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security, pp 1601–1616

Patzke T (2017) SigmaHQ. https://github.com/SigmaHQ/sigma

Platform NGD (2021) Neo4j graph platform—the leader in graph databases. https://neo4j.com/

Shu X, Araujo F, Schales DL, Stoecklin MP, Jang J, Huang H, Rao JR (2018) Threat intelligence computing. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security, pp 1883–1898

Threatbook. https://x.threatbook.cn/

ThreatMiner. https://www.threatminer.org/

Torrey J (2020) Transparent-computing. https://github.com/darpa-i2o/Transparent-Computing

Virustotal (2020). https://www.virustotal.com/gui/

Xu Z, Wu Z, Li Z, Jee K, Rhee J, Xiao X, Xu F, Wang H, Jiang G (2016) High fidelity data reduction for big data security dependency analyses. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, pp 504–516

Yang R, Ma S, Xu H, Zhang X, Chen Y (2020) UIScope: accurate, instrumentation-free, and visible attack investigation for GUI applications. In: NDSS