universität wien

DISSERTATION / DOCTORAL THESIS

Titel der Disseratation / Title of the Doctoral Thesis

"Improving Cybersecurity through Semantic Log Monitoring, Analysis and Attack Reconstruction"

verfasst von / submitted by

Kabul Kurniawan

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Doktor der Technischen Wissenschaften (Dr. techn.)

Wien, 2023 / Vienna, 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA A 786 880
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Doktoratsstudium Informatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. Dr. Gerald Quirchmayr o. Univ.-Prof. Dipl.-Ing. Dr. A Min Tjoa

Acknowledgements

First and foremost, I would like to express my sincerest gratitude to Allah (the Most Gracious, the Most Merciful) and His messenger, Rasulullah Muhammad (Peace be upon Him), who have been the source of strength and guidance throughout my life. Without their blessings, I would not have been able to embark on this academic journey and complete this thesis. I pray that Allah *Subhanahu Wa Ta'ala* accepts this work as a humble contribution to knowledge and continues to guide me in all my future endeavors.

I would like to express my deepest gratitude to my supervisors Prof. Gerald Quirchmayr and Prof. A Min Tjoa, for their guidance and support throughout my Ph.D. journey. They have provided invaluable insights and feedback that have greatly improved the quality of my research. Their expertise and knowledge have been instrumental in the successful completion of my thesis. I am truly grateful for their mentorship and support.

I am deeply thankful to my inspiring mentor, Dr. Elmar Kiesling, for his guidance and support in navigating the challenges of conducting research and completing my thesis. Dr. Kiesling is my role model who has shown me the true meaning of dedication, hard work, and passion for research. I am honored to have had the opportunity to work with him and to have his support and friendship. I also express my gratitude to Dr. Andreas Ekelhart for inspiring me with critical thinking, precision, and expertise. Their experience has been invaluable in guiding me through the process and providing valuable feedback and insights.

This Ph.D. research is generously supported by the *Beasiswa Unggulan Scholarship* from the Ministry of Education, Culture, Research, and Technology, Indonesia, as well as financed by netidee SCIENCE and the Austrian Science Fund (FWF) under grant P30437-N31. Therefore, I would like to express my sincere gratitude to both institutions that have supported me throughout my Ph.D. journey. Without their financial assistance, I would not have been able to pursue my passion for research and contribute to the field of Computer Science.

During my research, I am grateful for the opportunity to not only work as a Ph.D. student at the University of Vienna, but also to work as a researcher at the Technical University of Vienna (TU Wien), the Vienna University of Economics and Business (WU Wien), and the Austrian Center for Digital Production (ACDP). I would like to express my gratitude to all of my colleagues from these organizations for their support throughout this process. Their friendship and collaborative spirit have greatly enriched my experience at these organizations and made my time there truly valuable.

Special thanks to all my friends here in Austria, especially to Mas Pandu who picked me up and help me upon my arrival in Vienna, Mas Fajar, Mas Yusfan, Mas Adhi, and Mas Peb. The "Travellers" and "Culinary" Team (Mas Adryan, Mas Sisko, Mas Helmi, Mas Agis, Mas Arie). Those from Wapena Team (Pak Acha, Pak Arya, Bu Dewi, Bu Nining, Pak Said, Pak Sugeng, Pak Dindin, Pak Suhermanto, Pak Agus, Mba Rasmi, Bu Rini, Mas Arfi, Mas Henri, Mas Anam, Mad Addin, Mas Adit, Mas Guntur, Mas Ardhi), Those from KBRI (Pak Endang, Mas Yudho, Mas Sohib, Pak Ricky). The PPLN Team (Pak Victor, Mba Anies, Mba Fritska, Mba Vicky, Mba Palupi, Bang Gonti). Those who were in

my *Liqo'* team (Pak Ismail, Mas Gunar, Mas Gungun, Mas Wisnu, Mas Adi, Mas Fajar). I would also like to acknowledge all of my friends who cannot be named individually here. Their support and friendship have made a significant impact on my experience in Austria.

Finally, I would like to express my heartfelt gratitude to my parents (Bapak Wachir and Ibu Dayati), brothers and sisters for their support and encouragement throughout my Ph.D. journey. Their guidance, love, and prayers have been instrumental in my academic and personal growth. I cannot thank them enough for their care, sacrifices, and dedication to my success. I would also like to express my deepest gratitude to my family: my beloved wife, Riesa Yosita and my "twin" children, Thalita Ilmi and Rijalul Ilmi, for their support, care, and love. I could not have completed this journey without them. I am truly grateful to have such a loving family. Thank you for always being there for me.

This thesis would not have been possible without the support of all of these individuals, and I am deeply grateful for their contributions.

Abstract

The widespread adoption and reliance on IT systems nowadays has led to a significant increase in the prevalence and sophistication of cyber-threats. These increasing difficulties often make organizations unaware of attempted attacks and their impacts. Many organizations are typically unable to quickly identify security incidents and understand their causes. As a result, the confidentiality, availability, and integrity of their sensitive information become more and more threatened. A sensitive data breach can have serious consequences, including financial losses, decreased trustworthiness, and reputational damages.

Log data and cybersecurity resources are highly valuable information since they can provide security analysts with clear visibility and understanding of system activities and allow them to investigate security incidents, identify attacks, and monitor the system's health. However, they remain to pose several challenges in terms of data heterogeneity and inconsistent structure due to the complexity and variety of systems. Furthermore, the vast amount of generated log data distributed across multiple hosts and networks makes it more complicated and difficult to comprehend.

Several existing cybersecurity tools such as Security Information and Event Management (SIEM) and Intrusion Detection Systems (IDS) are widely used by security analysts to analyze log data and help them to contain cybersecurity attacks. Moreover, several methods have recently been proposed by researchers to tackle these challenges. Nevertheless, they typically lack grounding in a formal conceptualization and standard data model, do not adequately address data heterogeneity (integration), do not support automatic reasoning and linking between log events (inference), and do not consider reuse and linking to cybersecurity information (contextualization).

In this thesis, we propose novel approaches that leverage Semantic Web technologies and Knowledge Graphs for semantic log monitoring & analysis, threat detection, and attack reconstruction. To this end, we develop: *(i)* vocabularies and ontologies that provide a uniform representation to integrate heterogeneous, dispersed log data and cybersecurity information; *(ii)* continuously updated cybersecurity knowledge graphs constructed from various highly valuable cybersecurity resources that provide log event enrichment and contextualization; *(iii)* frameworks and tools based on RDF Stream Processing engine (RSP) that support (near) real-time semantic log monitoring and analysis; *(iv)* a virtual knowledge graph framework that provides a scalable approach for log analysis and querying over multiple distributed hosts/networks; *(v)* a knowledge graph-based framework for threat detection and attack reconstruction.

We assess the applicability and usability of our approaches on a variety of use cases and application scenarios using synthetic data as well as existing, well-established datasets. Furthermore, we perform an empirical evaluation to validate the feasibility and effectiveness of our approaches in terms of performance and scalability. Based on these evaluations, we found that our approaches facilitate security log monitoring, analysis and attack reconstruction in an efficient and scalable manner and facilitate effective linking and contextualization, therefore reducing alert fatigue and improving situational awareness.

Kurzfassung

Die weitverbreitete Einführung und Abhängigkeit von IT-Systemen hat zu einer signifikanten Zunahme von Cyber-Bedrohungen in Bezug auf Häufigkeit und Raffinesse geführt. Diese zunehmenden Schwierigkeiten machen Organisationen oft unbewusst von versuchten Angriffen und deren Auswirkungen. Viele Organisationen sind typischerweise nicht in der Lage, Sicherheitsvorfälle schnell zu identifizieren und deren Ursachen zu verstehen. Infolgedessen werden die Vertraulichkeit, Verfügbarkeit und Integrität ihrer sensiblen Informationen immer mehr bedroht. Ein Datenleck von sensiblen Daten kann schwerwiegende Folgen haben, einschließlich finanzieller Verluste, verringerter Vertrauenswürdigkeit und Reputationsschäden.

Logdaten und Cybersicherheitsressourcen sind hochwertige Informationen, da sie Sicherheitsanalysten klare Sichtbarkeit und Verständnis ihrer Systemaktivitäten bieten, wie die Untersuchung von Sicherheitsvorfällen, die Identifizierung von Angriffen und die Überwachung der Systemgesundheit. Sie stellen jedoch weiterhin mehrere Herausforderungen in Bezug auf die Heterogenität der Daten und die inkonsistente Struktur aufgrund der Komplexität und Vielfalt von Systemen dar. Darüber hinaus macht die enorme Menge an generierten Logdaten, die über mehrere Hosts und Netzwerke verteilt sind, es komplizierter und schwieriger, diese zu verstehen.

Es gibt mehrere bestehende Cybersecurity-Tools wie das Security Information and Event Management (SIEM) und Intrusion Detection-Systeme (IDS), die von Sicherheitsanalysten verwendet werden, um Logs zu analysieren und ihnen bei der Bekämpfung von Cyberangriffen zu helfen. Zudem wurden in jüngster Zeit von Forschern mehrere Methoden vorgeschlagen, um diese Herausforderungen anzugehen. Sie haben jedoch typischerweise keine formale Konzeptualisierung und kein Standarddatenmodell als Grundlage und nehmen die Heterogenität der Daten (Integration) nicht ausreichend in Betracht, unterstützen keine automatische Schlussfolgerung und Verknüpfung von Logereignissen (Schlussfolgerung) und berücksichtigen die Wiederverwendung und Verknüpfung von Cybersecurity-Informationen (Kontextualisierung) nicht.

In dieser Dissertation schlagen wir neuartige Ansätze vor, die Semantic Web-Technologien und Knowledge Graphs für die semantische Log-Analyse und -Überwachung, Bedrohungserkennung und Angriffsrekonstruktion nutzen. Hierzu entwickeln wir: *(i)* Vokabulare und Ontologien, die eine einheitliche Darstellung bereitstellen, um heterogene, zerstreute Logdaten und Cybersecurity-Informationen zu integrieren; *(ii)* ständig aktualisierte Cybersecurity-Wissensgraphen, die aus verschiedenen hochwertigen Cybersecurity-Ressourcen erstellt werden und Log-Ereigniserweiterung und -Kontextualisierung bieten; *(iii)* Frameworks und Tools auf der Basis von RDF Stream Processing-Engines (RSP), die (nahezu) echtzeitige semantische Log-Überwachung und -Analyse unterstützen; *(iv)* ein Framework für virtuelle Wissensgraphen, das einen skalierbaren Ansatz für die Log-Analyse und -Abfrage über mehrere verteilte Hosts/Netzwerke bereitstellt; *(v)* ein Framework für die Bedrohungserkennung und Angriffsrekonstruktion auf der Basis von Wissensgraphen.

Wir beurteilen die Anwendbarkeit und Benutzerfreundlichkeit unserer Ansätze anhand

einer Vielzahl von Anwendungsfällen und Anwendungsszenarien mithilfe von synthetischen Daten sowie bestehenden, gut etablierten Datensätzen. Darüber hinaus führen wir eine empirische Evaluation durch, um die Machbarkeit und Wirksamkeit unserer Ansätze hinsichtlich Leistung und Skalierbarkeit zu validieren. Aufgrund dieser Bewertungen haben wir festgestellt, dass unsere Ansätze die Sicherheitslog-Analyse, -Überwachung und -Angriffsrekonstruktion auf effiziente und skalierbare Weise erleichtern, aber auch Verknüpfung und Kontextualisierung bieten und somit die Alarmmüdigkeit verringern und das Situationsbewusstsein verbessern.

Contents

List of Tables

List of Figures

List of Algorithms

Listings

List of Abbreviations

APT Advanced Persistent Threat.

ATT&CK Adversarial Tactics, Techniques, and Common Knowledge.

BGP Basic Graph Pattern.

CAPEC Common Attack Pattern Enumeration and Classification.

CCE Common Configuration Enumeration.

CEE Common Event Expression.

CEP Complex Event Processing.

CERT Computer Emergency Response Team.

CPE Common Platform Enumeration.

CSKG Cybersecurity Knowledge Graph.

CTI Cybersecurity Threat Intelligence.

CVE Common Vulnerabilities and Exposures.

CVSS Common Vulnerability Scoring System.

CWE Common Weakness Enumeration.

CybOX Cyber Observable eXpression.

DDoS distributed Denial-of-Sevice.

EDR Endpoint Detection and Response.

ETL Extraction, Transformation, Loading.

FINE Format for Incident Information Exchange.

HDT Header, Dictionary, Triples.

HIDS Host-based Intrusion Detection Systems.

HTTP Hypertext Transform Protocol.

ICT Information and Communication Technologies.

IDMEF Intrusion Detection Message Exchange Format.

IDS Intrusion Detection System.

IEEE Institute of Electrical and Electronics Engineers.

IETF Internet Engineering Task Force.

IoC Indicator of Compromise.

IODEF Incident Object Description Exchange Format.

IoT Internet of Things.

IRI International Resource Identifier.

ISO International Organization for Standardization.

ITU International Telecommunication Union.

JSON JavaScript Object Notation.

KB Knowledge Base.

KG Knowledge Graph.

LD Linked Data.

MAEC Malware Attribute Enumeration and Characterization.

ML Machine Learning.

NIDS Network-based Intrusion Detection Systems.

NIST National Institute of Standards and Technology.

NLP Natural Language Processing.

OBDA Ontology-based Data Access.

OVAL Open Vulnerability and Assessment Language.

OWL Web Ontology Language.

RDF Resource Description Framework.

RDFS Resource Description Framework Schema.

RML RDF Mapping Language.

RSP RDF Stream Processing.

SEM Security Event Management.

SIEM Security Information and Event Management.

SIM Security Information Management.

SPARQL SPARQL Protocol and RDF Query Language.

STIX Structured Threat Information eXpression.

STUCCO Situation and Threat Understanding by Correlating Contextual Observations.

TAXII Trusted Automated eXchange fo Indicator Information.

TPF Triple Pattern Fragments.

TTP techniques, tactics, procedures.

UCO Unified Cybersecurity Ontology.

URI Uniform Resource Identifiers.

VA Vulnerability Assessment.

VKG Virtual Knowledge Graph.

W3C World Wide Web Consortium.

WWW World Wide Web.

XML extensible markup language.

1. Introduction

1.1. Motivation

Organizations' reliance on Information and Communication Technology (ICT) systems has grown significantly in recent years. Although these systems provide numerous benefits, they also leave organizations vulnerable to increasing cyber-security threats. According to industry analyses [1, 2], today's ICT systems are under unprecedented threats from increasingly non-trivial and targeted attacks. These sophisticated threats typically combine multiple attack vectors, including client-side attacks, to bypass traditional perimeter security [3]. Furthermore, they quickly update their tactics in order to maintain an advantage against improvements in security measures put in place by organizations and governmental bodies. This increasing difficulty frequently makes organizations unaware of attempted attacks and their impacts [4]. Most of them fail to or slow to identify security incidents and understand their causes. Consequently, the confidentiality, availability, and integrity of their sensitive information become threatened. Inadequate system security may have a serious impact, including the disruption of business operations and the exposure of sensitive data, which could result in not only financial losses but also decreased trustworthiness and reputational damages [5].

The effort to keep systems safe from cyber-security threats requires comprehensive security analyses. Logs produced by systems can be used to support security analysts to precisely understand malicious events and activities that occurred within them [6]. They are composed of log entries; each entry contains information related to a specific event that has occurred within a system or network [7]. For instance, security log data describes users' activities when they attempt to gain access to a certain system via a network. These traces of events will then be stored in a log that contains a set of information such as *date, time-stamp, username, type of access, message etc.* By extracting and mining those data, security analysts can reveal events that occurred on a given system [6].

In addition to using log data as one of the primary sources for addressing cybersecurity threats, security analysts often rely on publicly available cybersecurity information to protect their ICT systems. They contain information related to cybersecurity such as *vulnerabilities, weaknesses, threats& attack patterns, cybertheat intelligences, etc.* and are typically represented in various formats, either structured (e.g., CVE[1], CWE[2], CAPEC[3], etc.) or in a raw text (e.g., documentation, reports, blogposts, etc.). Cybersecurity information plays a vital role as it provides the necessary knowledge for security analysts, e.g., to contextualize log events, link events to high-level attack patterns, etc.

Despite the plethora of log data and cybersecurity information, it remains challenging for security analysts to utilize these resources effectively in detecting attacks and mitigating their effects. This is due to the heterogeneous and dispersed nature of the data, which

[1] `https://cve.mitre.org/`
[2] `https://cwe.mitre.org/`
[3] `https://capec.mitre.org/`

is often scattered across multiple hosts and networks. To address this issue, security analysts typically utilize a set of security tools such as Security Information and Event Management (SIEM) and Intrusion Detection System (IDS). SIEMs are typically used to aggregate relevant information from the extracted log data from multiple sources in order to provide system administrators convenient access to logging information. Using a statistical correlation method, for example, SIEMs can generate relationships between event log entries. IDSs, on the other hand, are often used to detect suspicious activities on hosts and networks. They typically use two approaches, i.e., *(i) signature-based-detection* that compare events against the database of suspicious activity to identify known attack and *(ii) anomaly-based detection* that uses statistical techniques to identify patterns that are not only unknown but also have not been observed before.

Current approaches in security log analysis mainly focus on system anomaly detection through log parsing and log mining [8]. These approaches typically encounter challenges with incomplete information of logs, which limits the extent of analysis [9]. These challenges emerge due to the difficulty of getting the relevant information from incomplete log sources, as an event occurred in IT systems are typically distributed into different log sources in different locations. Another challenge is the extraction of semantic information from security logs, which is not yet addressed in the current approaches. Without a correct understanding of information contained in logs, the causally related events from security logs are difficult to be inferred. The aim of this thesis is to address research challenges described in Section 1.2 and to answer research questions stated in Section 1.4 by exploring the potential of semantic web technologies and knowledge graphs within the context of cybersecurity and information security (cf. Section 1.3).

1.2. Research Challenges

In the following, we identified several research challenges that serve as the primary focus of this study.

Challenge 1 (C1): *Log data heterogeneity* Security log data is typically generated by a variety of devices and systems within a network, such as servers, routers, firewalls, etc. It has a wide range of structural schemas and is typically poorly structured, highly verbose, and uses a variety of terminologies. Figure 1.1 shows several examples of log messages from different log sources including *Firewall Log, Windows Event Log, Linux Authlog and Syslog*. These log sources expose multiple heterogeneity including: *(i) Syntactic heterogeneity*, i.e., similar data represented in different format/syntax, e.g., varying timestamp format in different logs. *(ii) Semantic heterogeneity*, i.e., different meaning and interpretation, e.g., varying severity scales from different log standards; *(iii)* Inconsistent identifier, e.g., hostname vs. IPAddress. This example shows how different systems produce log messages with various structures. Other discrepancies in log messages can be found, including those in the names of the attributes and the structure's hierarchy. This inconsistent data representation makes it difficult for analyst to connect and identify indicators of compromise hence attack detection becomes infeasible specially in an emergency situation.

Challenge 2 (C2): *Evolving heterogeneous cybersecurity information* Real-world cybersecurity information on vulnerabilities, weaknesses, attack patterns, cyberthreat

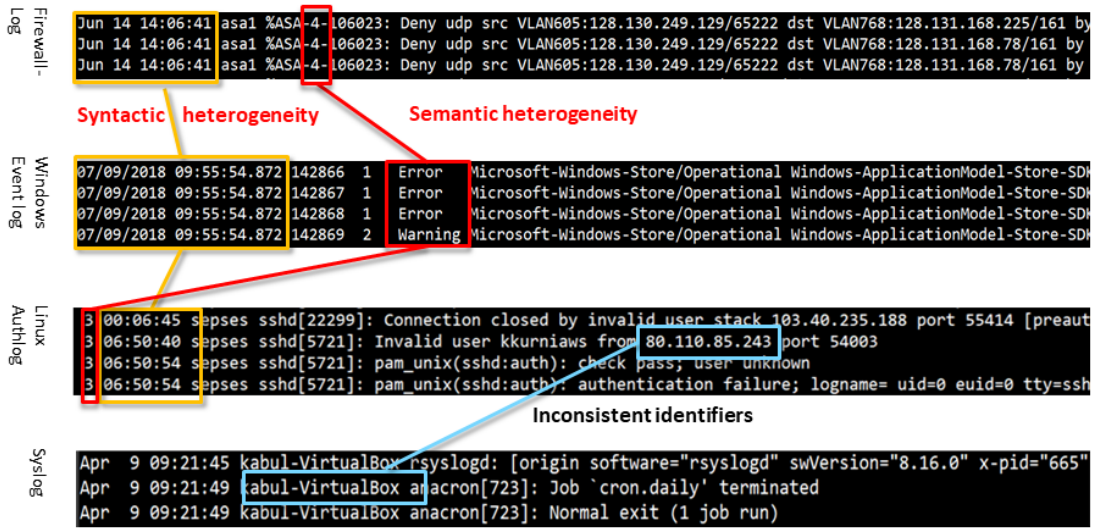

Figure 1.1.: Security log messages generated from heterogeneous applications and systems

intelligences is publicly available and can support security professionals in protecting their ICT systems. Security incidents are frequently published in the media and typically communicated informally as text. Furthermore, efforts to make security information accessible in well-defined structured formats, notably led by MITRE[4] and NIST[5], have made significant progress and produced a wide range of standards. These standards describe high-level schemas for cybersecurity information that are accessible for browsing on the web and downloading in heterogeneous structured formats, such as CVE, CWE, CAPEC, CVSS[6], MITRE ATT&CK[7], etc. However, while these resources are provided in well-defined structured formats, the individual entities and datasets remain isolated and lack semantics. This limits the automated machine interpretation potentials, hence making it difficult for security analysts to keep track of all available sources and identify relevant information.

Challenge 3 (C3): *Dispersed raw log sources* Log sources are typically scattered across multiple systems, hosts and networks within an organization. This is a common practice since organizations may use different hosts and networks for different purposes. Another typical reason is to ensure the availability and reliability of the system, e.g., by providing backup systems and networks in case of an outage. As a result, the generated log data can be physically and geographically distributed across different locations. Figure 1.2 shows an example of a cybersecurity threat (e.g, data exfiltration) and illustrates how such incidents may leave traces and generate log data in numerous log sources in multiple distributed hosts.

[4] https://www.mitre.org
[5] https://nist.gov
[6] https://www.first.org/cvss/
[7] https://attack.mitre.org/matrices/enterprise/

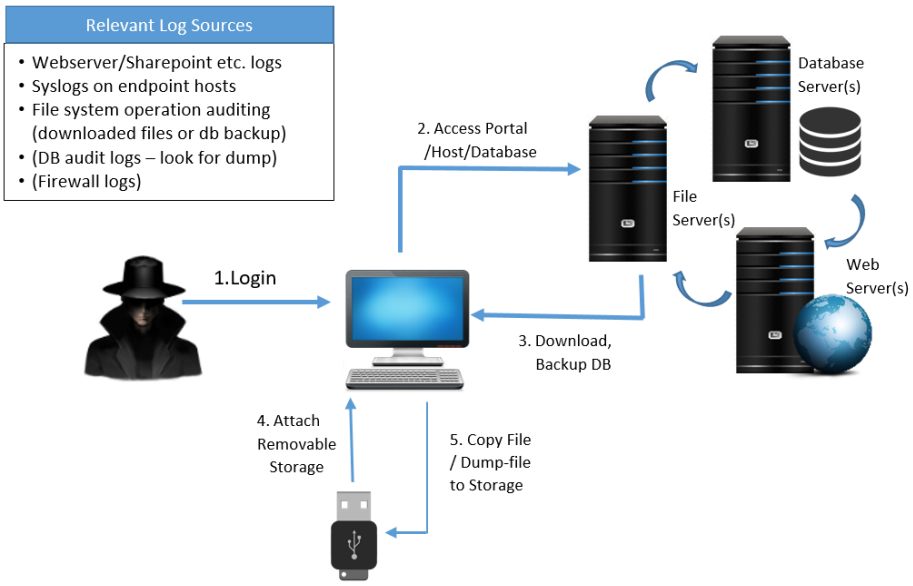

Figure 1.2.: An example case of data exfiltration

On a local computer, there is a user who has access to a specific system using either authorized or illegitimate credentials. This incident will be recorded in a relevant log (e.g., Windows Event Log). The user may download a credential file or dump a database from the database server after gaining access to a particular server, such as a file server, web server, database server, etc. Then, a pertinent log will be updated with these events (e.g., Webserver log, Share-point logs, Syslogs on endpoint hosts, file system operation auditing for downloaded files or database backups, database audit logs, firewall logs, etc.). In order to transfer or move the downloaded contents to her or his own storage, the user can then attach a portable storage medium (such as a USB flash drive). A system log will thereafter have a record of these actions (attach and copy activities) (e.g., Win Event log).

As described above, a single intrusion could result in numerous low-level events and leave traces over multiple log sources, which can be scattered across different systems, devices, and hosts. Identifying such isolated indicators of compromise over dispersed log sources is necessary in order to detect such attacks. However, analyzing those disparate log sources remains challenging as security analyst may easily be over-burdened and lose of keeping track of log data, particularly when the intrusion happens suddenly or in an emergency situation.

Challenge 4 (C4): *Complex event connection within and across log sources* As described in *C2*, the generation of multiple traces of (low-level) events - as a result of an incident - creates complex event connections within log sources. Furthermore, attacks often combine multiple vectors in a sequence that on the one hand produce even more traces which highly increases event connection complexity. On the other hand, timing and sequence of events are crucial to link indicator and analyze the attack progression. Therefore, manually searching log files and comparing timestamp to reconstruct complex

chains of attack becomes more and more challenging. This make it even more complicated due to the fact that the existing suspicious events are frequently concealed behind the abundance of benign events in log sources.

Challenge 5 (C5): *Attack analysis and interpretation are highly context-specific* Isolated indicators are often inconspicuous in their local context, but they may actually be indicators of a sophisticated attack when considered in the larger context. Without the ability to link events to corresponding contextual knowledge, it can be challenging to fully understand the individual indicators of an attack and their connection to the overall attack chain. Therefore, it is important to examine all available data, including both isolated indicators and their broader context. This can help security analysts gain a more comprehensive understanding of the attack (e.g., understanding attack tactics and techniques used by attackers).

1.3. Background

This section gives a brief background on this thesis, which is based on two key topics. First, we provide a general overview of cybersecurity and security log management. Second, we discuss knowledge graphs and semantic web technologies, as well as their potential application to the cybersecurity domain.

1.3.1. Information Security, Cybersecurity and Security Log Management

Information Security and Cybersecurity *Information Security* is the protection of information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide confidentiality, integrity, and availability [10]. Confidentiality refers to the preservation of authorized restrictions on the access to and disclosure of information (e.g., proprietary information and individual privacy), while integrity ensures that information is not improperly modified or destroyed and that it is authentic and non-repudiated. Availability, on the other hand, refers to the prompt and reliable access to information [10]. *Cybersecurity* is a subset of information security that specifically focuses on protecting data from unauthorized electronic access, such as cyberattacks, by deploying various defense processes, policies, procedures, and technology [11]. Cyberattacks come in many different forms such as *Malware, Ransomware, Phishing, Social Engineering, Insider Threats, distributed Denial-of-Sevice (DDoS), Advanced Persistent Threat (APT)s, etc.* Cybersecurity can be categorized into several distinct types, including *Critical infrastructure security, Application security, Network security, Cloud security, Internet of Things (IoT)*, etc [12].

Security Log Data and Standard Format Today, a variety of logging standards are in use, frequently concentrating on a particular application domain, such as operating system logs (e.g., syslogd and Windows Event Logs), web server logs (e.g., W3C[8] Extended log file format, NGINX[9] logging), database logs, firewall logs, etc. Log entries are frequently stored as semi-structured lines of text with unstructured fields such as a *message* and

[8] `https://www.w3.org/`
[9] `https://www.nginx.com/`

structured sections such as a *timestamp* and *severity level*. The content of the unstructured fields comprises context-specific information and typically lacks uniformity, whereas the structured elements are normally standardized. Raw log data must first be split into their relevant parts, such as a *key-value* based representation, in order to be (automatically) analyzed. Predefined regular expressions (regex) are typically used in this preprocessing step. Other standards, such as the Windows Event Log (EVTX), are already represented as XML[10] or JSON[11] and are highly structured. Despite efforts at standardization, different log formats can provide a barrier to efficient analysis. Common Event Expression (CEE)[12], an early project led by MITRE, aims to unify heterogeneous vocabularies to express events in electronic systems in a uniform, device-independent manner. In order to facilitate log interchange, CEE separates the taxonomy (semantic event type), the log syntax (instance data), and the log transport component. However, due to the U.S. Government's suspension of funding in 2014, MITRE ceased all work on CEE.

Security Log Management and Analytics The increasing number volume and variety of logs has prompted the need for systematic computer security log management. These initiatives are described in a number of standards and best practices for industry, including the NIST Cybersecurity Framework [13] and the NIST SP 800-92 Guide to Computer Security Log Management [7]. In order to extract knowledge from these logs, a variety of log-analytic techniques have been developed, including anomaly detection, clustering, and rule-based intrusion detection. Security Information and Event Management (SIEM) is a combination of Security Information Management (SIM) and Security Event Management (SEM) [14]. SIEMs are widely used to provide a centralized perspective on security-relevant events inside an organization with a focus on data collection, correlation, and typically rule-based alerting. They also frequently have monitoring and alerting systems. SIEMs have been implemented in various security service and application, e.g., *Splunk*[13], *Papertrail*[14], *Librato*[15], *ArcSight*[16], *ElasticSearch*[17], *QRadar*[18], *LogRhythm*[19], *McAfee ESM*[20], *etc.* SIEM systems have made a first step toward gathering log information in a centralized repository and enabling security specialists to run queries. However, they lack a foundation in a formal conceptualization and semantics of their data. Consequently, they do not facilitate contextualization, linking, and automatic reasoning. Therefore, it is difficult to infer semantic correlations between events.

Cybersecurity Information and Exchange Standards Common standards are necessary for effective information exchange, especially in fields with a wide range of domains and a fast-paced environment such as cybersecurity. In light of this, a set of standards have been developed that specify the syntax of description languages for structured cybersecurity

[10] https://www.w3.org/XML/
[11] https://www.json.org/
[12] https://cee.mitre.org/
[13] https://www.splunk.com/
[14] https://www.papertrail.com/
[15] https://www.librato.com/
[16] https://www.microfocus.com/en-us/cyberres/secops/arcsight-esm
[17] https://www.elastic.co/
[18] https://www.ibm.com/qradar
[19] https://logrhythm.com/
[20] https://www.mcafee.com/

information as well as the semantics of those descriptions in natural language. Some of these standards are being developed by well-known standardization organizations such as ISO[21], ITU[22], IEEE[23], or IETF[24]. A standardized schema for describing, communicating, and specifying increasingly complex events is represented by Cyber Observable eXpression (CybOX)[25]. Among other things, CybOX makes it possible to define attack pattern, describe malware, and log events. The Malware Attribute Enumeration and Characterization (MAEC)[26] project, the Structured Threat Information eXpression (STIX)[27] project, and the Trusted Automated eXchange fo Indicator Information (TAXII)[28] project are similar initiatives. the Intrusion Detection Message Exchange Format (IDMEF)[29], the Incident Object Description Exchange Format (IODEF)[30], and the Format for Incident Information Exchange (FINE)[31] covers alert. Another important example of cybersecurity information sharing standards is Common Vulnerabilities and Exposures (CVE) for publicly known vulnerabilities, Common Attack Pattern Enumeration and Classification (CAPEC) for known attack patterns used by adversaries, Common Weakness Enumeration (CWE) for software security weaknesses, and Common Platform Enumeration (CPE) for encoding the names of IT products and platforms. These standards are frequently used by security professionals and incorporated into security goods and services, but they also provide an essential framework for study. These standards share the ability to establish schematic models and exchange formats, but their semantic expressiveness is typically limited. They therefore do not allow for full semantic compatibility of sharing threat intelligence across organizations.

Intrusion Detection Systems Intrusion Detection System (IDS)s are a mechanism to monitor networks or systems for malicious activities. There are two primary methods of detection [15], i.e., *(i)* signature-based detection, and *(ii)* anomaly-based detection. By comparing an event against a database of malicious activity signatures, signature-based detection is used to identify known attacks, whereas anomaly-based detection uses statistical methods to identify patterns that have never before been observed in addition to unknown patterns. Recently, successful applications of machine learning and complex event processing algorithms have been made in this context[16]. IDSs can be classified into two main categories, i.e., *(i)* Network-based intrusion detection systems (NIDS), and *(ii)* Host-based intrusion detection systems (HIDS). NIDS check all of a subnet's traffic and verify it using the packet content and metadata. A warning alert is provided to the network administrator if the NIDS discovers any intrusion in the network. HIDS analyses the incoming and outgoing traffic of the device only. It warns the administrator when it discovers unusual activity on the device such as suspicious logins, connections, operating system calls, and different command parameters.

[21] https://iso.org/
[22] https://www.itu.int/
[23] https://www.ieee.org/
[24] https://www.ietf.org/
[25] https://cybox.mitre.org/about/
[26] https://maecproject.github.io/
[27] https://oasis-open.github.io/cti-documentation/stix/intro.html
[28] https://taxiiproject.github.io/
[29] https://www.rfc-editor.org/rfc/rfc4765.html
[30] https://datatracker.ietf.org/doc/rfc7203/
[31] https://datatracker.ietf.org/doc/rfc5070/

1.3.2. Semantic Web, Linked Data and Knowledge Graph

Semantic Web The term "Semantic Web" was coined by Tim Berners-Lee, the inventor of the World Wide Web (WWW) and director of the World Wide Web Consortium (W3C)[32]. The Semantic Web can be conceived as an extended version of WWW or *Web 3.0* as it enables data to be linked from one source to another source and to be processed and understood by computers [17]. From a technical point of view, the Semantic Web consists primarily of three technical standards, i.e., Resource Description Framework (RDF), Web Ontology Language (OWL), and SPARQL Protocol and RDF Query Language (SPARQL). These standards provide a common framework and exchange protocols that allow data to be shared and reused across application, enterprise, and community boundaries.

RDF, RDFS, OWL Resource Description Framework (RDF)[33] is a standard framework and language for representing data and their interconnection on the Web. RDF statements express relationships between resources and are commonly known as "triple", a basic concept consisting of interconnected nodes such as *subject*, *predicate*, *object*. There are three different types of nodes in RDF: *(i) Resource Node* is anything that can have things said about it. It is typically represented as Uniform Resource Identifiers (URI) or International Resource Identifier (IRI) that identify a resource whether abstract or physical; *(ii) Literal Node*, a particular data value that can be a string, a date, or a numeric; *(iii) Blank Node*, known as an *anonymous* resource or a *bnode*, about which only the relationship is known.

RDF's initial syntax was based on the XML. Nevertheless, other syntaxes, such as N-Triple [34], JSON-LD[35] , or Turtle[36] are now more widely used.

The Resource Description Framework Schema (RDFS)[37] is a W3C standard data model for knowledge representation. It extends the basic RDF vocabulary with a set of classes and RDFS entailment (inference patterns)[38] and provides mechanism for describing class of connected resources and the relation among them. These resources are used to determine characteristics of other resources, such as the *domains* and *ranges* of properties [18]. The Web Ontology Language (OWL)[39] is a Semantic Web language to describe detailed and complex knowledge about various objects, groups of objects, and relationships between them. OWL is a computational logic-based language that allows computer programs to use the knowledge expressed in it, for example, to make implicit knowledge explicit [19].

SPARQL SPARQL Protocol and RDF Query Language (SPARQL) is a W3C recommended query language that can be used to retrieve and manipulate RDF data. It provides extensive expressivity for sophisticated queries like *negation, filtering, aggregation, and subqueries*. Through *query federation*[40], SPARQL also offers the ability to express queries across various dispersed data sources. Thus, that makes it possible to retrieve and manipulate distributed data sources over different SPARQL endpoints.

[32] https://www.w3.org/
[33] https://www.w3.org/RDF/
[34] https://www.w3.org/TR/n-triples/
[35] https://json-ld.org/
[36] https://www.w3.org/TR/turtle/
[37] https://www.w3.org/TR/rdf-schema/
[38] https://www.w3.org/TR/rdf11-mt/
[39] https://www.w3.org/OWL/
[40] https://www.w3.org/TR/sparql11-federated-query/

Semantic Reasoning One of the main concepts behind both OWL and RDFS is the ability to reason over a knowledge base. This is crucial because it can be used to automatically infer new facts from existing data based on inference rules and ontologies. Reasoning is possible using OWL entailment[41] (*owl:inverseOf* and *owl:SymmetricProperty*) as well as RDFS entailment[42] (*rdfs:domain, rdfs:range, rdfs:subPropertyOf, and rdfs:subClassOf*).

Linked Data The idea of "Linked Data" was proposed in 2009 to link various datasets together via the Semantic Web. It provides a method of publishing structured data so that it can be interlinked and become more useful through semantic queries. Linked data became part of the Semantic Web technology stack and has four design principles [20] coined by Tim by Tim Berners-Lee: *(i)* Use Uniform Resource Identifiers (URIs) as names for things; *(ii)* Use HTTP URIs so that people can look up those names; *(iii)* When someone looks up a URI, provide useful information, using the standards (i.e., RDF, OWL, SPARQL); *(iv)* Include links to other URIs so that they can discover more things.

Knowledge Graphs Over the years, the adoption of linked data has increased significantly, and it has played a crucial role in the emergence and development of "Knowledge Graphs". The term "Knowledge Graphs" itself has been used in the literature at least since 1972, but its modern usage dates to the Google Knowledge Graph announcement in 2012 that utilize semantic knowledge in the application of web search. The knowledge graph idea gained great popularity which was followed by announcements of knowledge graphs from *Amazon, eBay, Facebook, IBM, LinkedIn, Microsoft, Uber, Airbnb, etc* [21]. Knowledge graphs are frequently connected to *Linked Open Data* initiatives since the advent of the Semantic Web such as *DBPedia*[43] and *Wikidata*[44].

A knowledge graph is a directed, labeled graph with the G = (V, E), where V denotes a set of nodes (vertices), and E is a set of edges (properties). A single graph is represented as a set of triples T =< s p o >, where s denotes the subject, p is the predicate, and o denotes the object.

Figure 1.3 shows an excerpt of a log knowledge graph that expresses a single *Apache* log event in RDF. It shows a graphical representation of this log event in a knowledge graph. The subject :logEntry-24e is characterized by a number of properties that specify its type (cl:ApacheLog), the timestamp of creation, the originating host of the log event, the client that made the request, and the request string. Furthermore, the highlighted IP-Address indicates that the objects link to other entities in the graph which in turn have another object property (rdfs:type) and data property (cl:hostName).

Virtual Knowledge Graphs The Virtual Knowledge Graph (VKG) paradigm for data integration is typically used to provide integrated access to heterogeneous relational data. The approach, also known in the literature as Ontology-based Data Access (OBDA), aims to replace the rigid structure of tables in traditional data integration layers with the flexibility of graphs. This makes it possible to connect data silos by means of conceptual graph representations that provide an integrated view on the data. To this end, VKGs integrate three main ideas [22]:

[41] `http://www.w3.org/ns/entailment/OWL-Direct`
[42] `https://www.w3.org/ns/entailment/RDFS`
[43] `https://www.dbpedia.org/`
[44] `https://www.wikidata.org/`

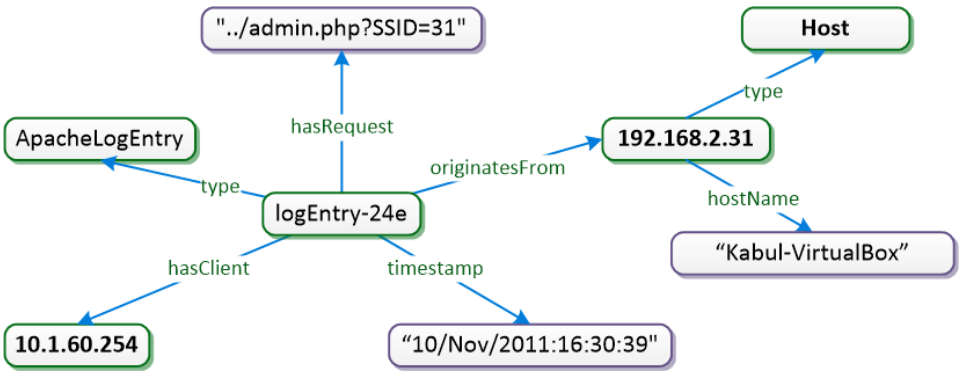

Figure 1.3.: RDF example of log data

- *Data Virtualization (V)*, i.e., they provide a conceptual view that avoids exposing end users to the actual data sources. This conceptual view is typically not materialized in order to make it possible to query the data without paying a price in terms of storage and time for the data to be made accessible.
- *Domain Knowledge (K)*, i.e., the graphs can be enriched and contextualized with domain knowledge that makes it possible to derive new implicit knowledge from the asserted facts at the time a query is executed.
- *Graph Representation (G)*, i.e., the data is represented as graph where object and data values are represented as nodes, and properties of those object are represented as edges. Compared to traditional relational integration tables, the graph representation provides flexibility and through mapping and merging makes it easier to link and integrate data.

1.4. Research Questions

In this section, we define our research questions based on the motivation and research challenges we have described before. The **main research question** in this thesis is:

To what extent can semantic web technologies improve cybersecurity monitoring and analysis?

In order to provide a more focused and comprehensive examination of the topic at hand, we break down our main research question into three sub-questions together and formulate a set of hypotheses.

Our first research question is mainly derived from the two challenges described above, i.e., *log data heterogeneity (C1)* and *the evolving heterogeneous cybersecurity information (C2)*.

> **Research Question 1 (RQ1):** *How to uniformly represent heterogeneous log data and cybersecurity information?*

Conceptualization of domain knowledge can be used to support unambiguous and useful interlinking between log data and cybersecurity information so that it can be understood

by a machine. A uniform conceptual model can semantically lift heterogeneous security log data and cybersecurity information from diverse sources. Therefore, *RQ1* is associated with the following hypothesis:

> **Hypothesis 1 (H1):** *Security log data and cybersecurity information can be structured and enhanced by semantics, to support unambiguous and useful interlinking between logs and cybersecurity information in a knowledge graph. A uniform conceptual model can semantically lift heterogeneous security log data and cybersecurity information from diverse sources.*

As described in *C3*, security log data generated by various systems and applications are commonly dispersed across multiple hosts and networks. Traditional security log analyses typically tackle this problem through centralized data integration (e.g., log server, SIEM systems, etc.).Extraction, Transformation, Loading (ETL)[23] is a typical method for traditional data processing. However, it has limitations in dealing with highly verbose, redundant, incoherent and poorly structured information. As systems typically generate vast amounts of high-frequency and fine-grained log data, analyzing data by means of a centralized method is not ideal for real-time processes or on-demand access, where scalability is required. Therefore, this leads to the following research question:

> **Research Question 2 (RQ2):** *How can dispersed log data and cybersecurity information be integrated and interlinked?*

We expect that decentralized semantic log processing can effectively handle large volumes of dispersed log data and facilitate more efficient integration and analysis, potentially addressing scalability issues in traditional centralized approaches. In addition, using semantic query federation, we expect that both dispersed log sources and cybersecurity information can be integrated and linked automatically. Thus, we ground our explanation of *RQ2* on the following hypothesis:

> **Hypothesis 2 (H2):** *Through decentralized semantic log processing, multiple dispersed security log data can be integrated in a scalable manner. Furthermore, distributed security log events together with cybersecurity information can be interlinked via semantic query federation to obtain meaningful results.*

The growing amounts of log data and their sparsity inhibits security analyses from timely detect and response to attacks. Current security analysis processes typically rely on human intelligence, rather than on automated systems to perform better task inference. This is because human intelligence has the ability to take into account contextual information that may not be easily captured by automated systems. However, human cognitive capacity is limited and they are easily over-burdened by high complexity of the event connections (*C4*) and context-sensitive interpretation of events (*C5*). Therefore, it makes manual security analysis would become infeasible. This leads to our next research question as follows:

> **Research Question 3 (RQ3):** *How can we discover and reconstruct attacks from system event log information?*

Knowledge graphs can be used to model complex causal relations between events and represent dependency graphs. They facilitate automated inference that once log data

have been lifted into uniform graph representation, a causal dependency link can be generated, e.g., by exploiting property chains, transitive and reflexive properties of graphs. Furthermore, existing threat detection rules can be defined and combined in flexible and declarative manner. This make it possible to not only detect potential threats but also link individual indicators of compromise to background knowledge, identify their root cause, and reconstruct attack scenarios. Therefore, our *RQ3* leads to the following hypothesis:

> **Hypothesis 3 (H3):** *Semantic reasoning can be used to infer sequences of events from security log data and generate causal dependency graphs. This makes it is possible to flexibly and declaratively specify existing attack patterns and rules to detect potential threats, link individual indicators of compromises, identify their root cause and reconstruct attack scenarios.*

1.5. Research Methodology

This section describes the research methodology we have chosen and explains its implementation in this thesis.

1.5.1. Design Science Research

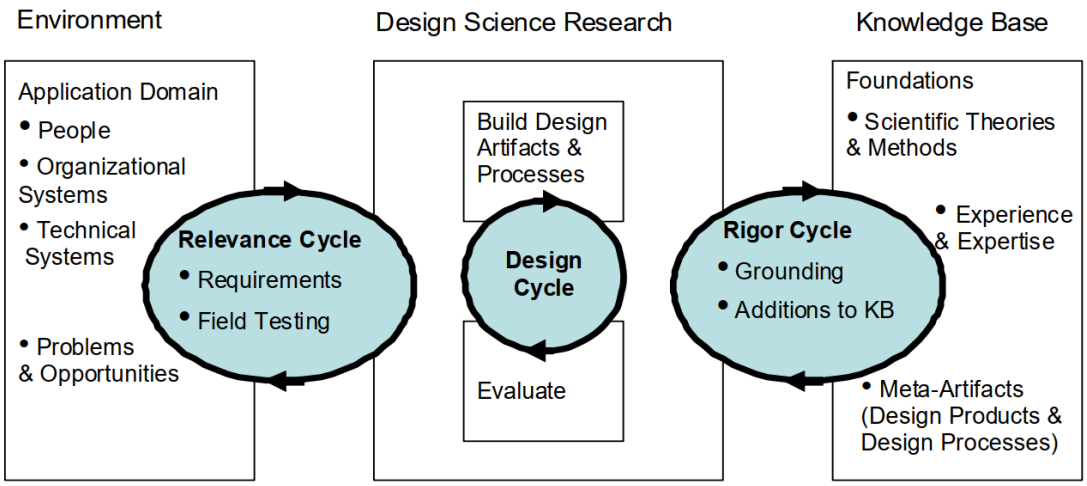

Figure 1.4.: Three Cycles of Design Science by Hevner [24]

We follow the design science principles proposed by Havner [25] as our research methodology. Design science is one of the prominent research methodologies in the area of Information Systems (IS) and is used as a guideline to develop and evaluate new innovative artifacts. The design science methodology consists of three building blocks [25], i.e., *(i) Knowledge base*, scientific theories and engineering methods that provide a foundation for rigorous design science; *(ii) Design science research*, where new innovative artifacts are built to address a certain problem. The developed artifacts are also evaluated to ensure the proposed solution meets the requirements of the environment; *(iii) Environment*, defines the problem or contextual domain in which the phenomenon of interest exists. It includes people, organization, problems, tasks, opportunities, etc.

In addition to the previous method, Hevner introduced three cycle views of design science research [24]. It emphasizes the embodiment of three closely related cycles of activities in design research methodology, as shown in Figure 1.4. We summarize it as follows: *(i) Rigor Cycle* connects the design science activities with the knowledge base of scientific foundations. It provides grounding for the design science to support building artifacts. It also provides input or addition to the Knowledge Base (KB) as a result of design science research activities; *(ii) Relevance Cycle* bridges the contextual environment of the research project with the design science activities; *(iii) Design Cycle* iterates between the core activities of building and evaluating the design artifacts and the processes of research.

1.5.2. Implementation of Design Science Research Methodology

Figure 1.5 shows an instance of design science research cycles in this PhD thesis. We describe the implementation of the methodology in our research as follows:

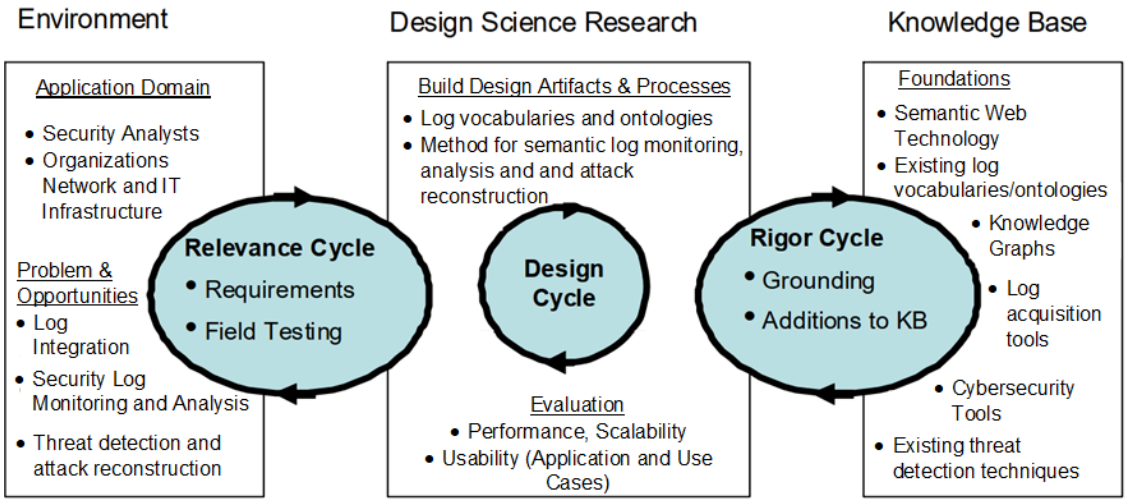

Figure 1.5.: An implementation of Three Cycle of Design Science on our thesis

Knowledge base We start our design science research with a literature review from two different research fields, i.e., Cybersecurity/Information Security and Semantic Web/Knowledge Graphs. This helps us to understand the existing related work and state-of-the-art concepts, methods, and technologies from those domains. We continue with the observation of the existing research gaps and their limitations and find options for research directions and further improvement. This observation supports us during research activities in design science (i.e., building and evaluating new artifacts). For example, the theoretical concepts in the Semantic Web domain help us in developing standard log vocabularies and construct the cybersecurity knowledge graph. Likewise, concepts in the cybersecurity domain help us to understand attack patterns and threat detection techniques that we also use as a basis of our artifact development.

This shows that the ***Rigor Cycle*** is fully implemented as we *ground* our research activities to the rigorous theories from both domain as well as contribute the *addition to KB* from our research findings.

Environment The environment of our design science research involves cybersecurity as our application domain. This includes organizations that have cybersecurity issues, persons in the organization (such as security analysts) who take the responsibility of dealing with cybersecurity, and the organization's infrastructures, e.g., IT assets, log data, etc. The environment provides research problems such as log integration, log monitoring and analysis, threat detection and attack reconstruction. Other supporting resources provide *opportunities* including log data, IT infrastructure, cybersecurity resources, etc.

The ***Relevance Cycle*** here is shown as we defined research *requirements* we need to fulfill during the design cycle with regards to the aforementioned environment. Furthermore, the environment provides *field testing* such as IT infrastructures, networks, etc. to where our developed artifacts are deployed, simulated and evaluated.

Design Science Research We develop several methods as a result of our observations to both *Knowledge Base* and *Environment* to address the research questions. This includes the development of several artifacts ranging from the conceptualization of methods, solution architecture / framework and their components, and the prototype implementation of the approach. The ***Design Cycle*** is applied as the developed methods and artifacts are iteratively evaluated through several aspects as follows: *(i) Use case applications*, to evaluate the usability and the feasibility of the developed methods/artifacts against different use-cases. *(ii) Performance and Scalability*, to measure the performance and scalability of the developed artifact (i.e., the prototype application) during the execution of the artifact in various testing scenarios.

1.6. Contributions Overview

In this section, we summarize the contributions of this PhD thesis that address the identified challenges and research questions outlined in the previous sections.

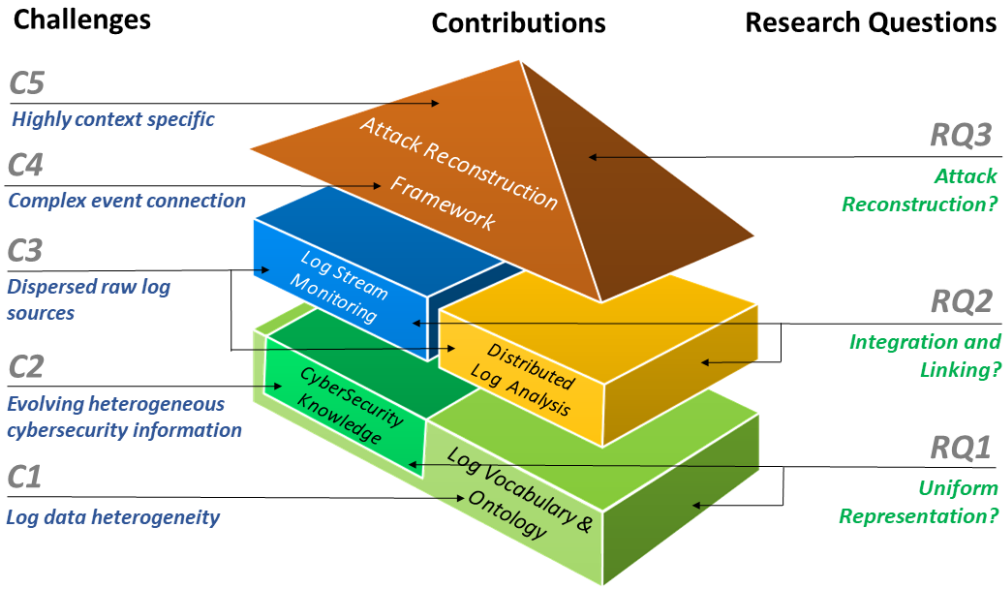

Figure 1.6.: Contribution Overview

Figure 5.1 shows the contribution overview of the research and the corresponding research challenges and research questions. The contributions consist of five building blocks (BBs) as follows:

BB1: This first block introduces modular log vocabularies and ontologies based on the OWL/RDFS for standardized representation of log data and cybersecurity information. These modular log vocabularies and ontologies serve as the foundation for the other building blocks and help to address the issue of log data heterogeneity (C1) by providing uniform representation (RQ1).

BB2: This block involves the construction of integrated cybersecurity knowledge graphs from publicly available cybersecurity information and attack patterns. These knowledge graphs provide a "rich" cybersecurity knowledge base that is crucial for providing context in log analysis and identifying potential attacks.

BB3: This third block involves the development of a method for integration and linking (RQ2) dispersed stream log data (C2). To this end, we developed a prototype framework for log monitoring based on the RDF Stream Processing (RSP) engine that supports real-time detection of suspicious file activities and contextualizes them with background knowledge.

BB4: It introduces a method for decentralized security log analysis using Virtual Knowledge Graph (VKG) approach that allow for the virtual and on-demand processing of log data (i.e, without a priori parsing, aggregation and processing log data). Furthermore, using semantic query federation, multiple dispersed log data (C3) can be integrated and queried simultaneously without the necessity for a centralized processing and repository (RQ2).

BB5: Finally, this block involves the developement of a modular framework for threat detection and attack reconstruction based on knowledge graphs. The semantic log representation makes it possible to infer complex event connections (C4) and combine the state-of-the art techniques for threat detection. Furthermore, the uniform and flexible graph models also enable enrichement and contextualization through linking to background knowledge (C5), providing a comprehensive mechanism for attack reconstruction (RQ3).

1.7. Publications

This PhD Thesis is based on the cumulative work compiled from numerous peer-reviewed publications. We describe the publications and highlight their roles in this thesis in chronological order as follows:

- Kurniawan K. Semantic Query Federation for Scalable Security Log Analysis. In: The Semantic Web: ESWC 2018 Satellite Events. ESWC 2018. Lecture Notes in Computer Science, vol 11155. Springer, Cham. p. 294–303. 2018 [26].

*This PhD Symposium paper serves as an initial proposal of this thesis and contributes to the development of **Chapter 1** (i.e., motivation and research questions).*

- Ekelhart A, Kiesling E, Kurniawan K.Taming the logs - Vocabularies for semantic security analysis. In: Proceeding of 14th SEMANTiCS vol. 137; p. 109–119. 2018 [27].

  *This conference paper focuses on C1 and investigates RQ1. It is included in this thesis in **Chapter 2**.*

- Kiesling E, Ekelhart A, Kurniawan K, Ekaputra F.The SEPSES Knowledge Graph: An Integrated Resource for Cybersecurity. In: The Semantic Web – ISWC 2019. Lecture Notes in Computer Science, vol 11779. Cham:Springer International Publishing; p. 198–214. 2019 [28].

  *This conference paper focuses on addressing C2 and provides an answer to RQ1. It appears in **Chapter 2** of this thesis.*

- Kurniawan, K., Ekelhart, A., Kiesling, E., Froschl, A., Ekaputra, F.: Semantic integration and monitoring of file system activity. In: Proceeding of 15th SEMANTiCS (2019). [29]

  This poster paper provides preliminary results that we extend to a conference paper [30].

- Kurniawan K, Kiesling E, Ekelhart A, Ekaputra F. Cross-Platform File System Activity Monitoring and Forensics – A Semantic Approach. In: ICT Systems Security and Privacy Protection. SEC 2020. IFIP Advances in Information and Communication Technology. Springer, Cham; p 384–397. 2020. [30].

  *This conference paper focuses on addressing C3 & C5 and provides an answer to RQ2. It appears in **Chapter 3** of this thesis.*

- Kurniawan K., Ekelhart, A., Kiesling, E. An ATT&CK-KG for Linking Cybersecurity Attacks to Adversary Tactics and Techniques. In: The Semantic Web: ISWC 2021. Cham:Springer International Publishing; 2021 [31].

  This poster paper extends our prior work [28] by introducing knowledge graph for cyberthreat intelligence (i.e. MITRE ATT&CK).

- Kurniawan K, Ekelhart A, Kiesling E, Winkler D, Quirchmayr G, and Tjoa A. 2021. Virtual Knowledge Graphs for Federated Log Analysis. In The 16th International Conference on Availability, Reliability and Security (ARES 2021). ACM, New York, NY, USA p. 1-11. 2021 [32].

  This conference paper focuses on addressing C3 & C5 and provides an answer to RQ2. It has been extended to a journal article [33].

- Kurniawan K, Ekelhart A, Kiesling E, Winkler D, Quirchmayr G, Tjoa AM. VloGraph: A Virtual Knowledge Graph Framework for Distributed Security Log Analysis. Machine Learning and Knowledge Extraction. 2022;4(2):371-96 [33].

*This journal article is an extended version of our prior work [32] with more extensive evaluation and discussion. It appears in **Chapter 3** of this thesis.*

- Kurniawan K, Ekelhart A, Kiesling E, Quirchmayr G, Tjoa AM. KRYSTAL: Knowledge graph-based framework for tactical attack discovery in audit data. Computers Security. 2022;121:102828 [34].

  *This journal article focuses on addressing C4 & C5 and provides an answer to RQ3. It is included in this thesis in **Chapter 4**.*

1.8. Outline

The remainder of this thesis is organized as follows:

Chapter 2 introduces modular vocabularies and ontologies for semantic representation of log events. We also describe how these representation models can be used as a foundation for log integration, monitoring and analysis. Furthermore, we propose a knowledge graph construction method that provides an integrated cybersecurity knowledge base derived from publicly available cybersecurity information.

Chapter 3 discusses methods for semantic log monitoring as well as federated log analysis through Virtual Knowledge Graphs. We start by describing their concepts, explain their architecture and components as well as show their prototype implementation. Furthermore, we describe the application of these approaches over several real-world use cases, perform an evaluation and discuss our findings.

Chapter 4 discusses a modular knowledge graph based framework for threat detection and attack graph construction. In this chapter, we focus on introducing a "hybrid" approach that combines different state-of-the-art threat detection and attack reconstruction techniques. We describe concepts, architecture as well as prototype implementation of this approach. Furthermore, we evaluate our approach over well-established, large scale datasets and discuss the results.

Chapter 5 summarizes our work, highlights our answer to the stated research questions and challenges and highlights our contributions. We also discuss the research impact as well as the remaining challenges and directions for future work.

Chapters 2-4 contain sections on related work that provide an overview of the current state-of-the-art and highlight how our contributions differ from other approaches.

2. Uniform Log Representation and Contextualization

In this chapter, we present our approaches that address our first research question (RQ1) as discussed in Chapter 1, i.e., *"How to uniformly represent heterogeneous log data and cybersecurity information?"*. We structured our discussion into two sections as follows:

- ***Vocabulary for log data and semantic security analysis***. In this section, we discuss the advantage of semantic web technology to bring meaning to vast volumes of raw log data. We focus on introducing vocabularies and ontologies for security log data and propose an initial framework for log data acquisition, extraction, enrichment with background knowledge and semantic analysis. The vocabulary serves as a foundational model and uniform representation that can be used to harmonize heterogeneous log data and integrate them. To this end, we used RDF/OWL, a W3C standard data modelling language to build the vocabulary/ontology. The initial framework together with log vocabularies and ontologies provides foundation for semantic security log processing and analysis.
- ***An integrated knowledge graph for cybersecurity information***. This section discusses our approach in developing automated knowledge graph construction from the existing cybersecurity resources, namely SEPSES CSKG. To this end, we develop CSKG vocabularies and integrate multiple heterogeneous cybersecurity resources, e.g., CVE, CWE, CPE, CVSS and CAPEC in a single, integrated knowledge graph. The constructed knowledge graph provides comprehensive and continuously updated cybersecurity knowledge that can be used to link and provide contextualization for semantic security log analysis and monitoring in an automated way.

2.1. Vocabulary for log data and semantic security analysis

Andreas Ekelhart, Elmar Kiesling, Kabul Kurniawan
Published as "Taming the logs – Vocabularies for semantic security analysis" in *The 14th International Conference on Semantic Systems (SEMANTiCS)*, 2018 [27].

Abstract

Due to the growing complexity of information systems and the increasing prevalence and sophistication of threats, security management has become an enormously challenging task. To identify suspicious activities, security analysts need to monitor their systems constantly, which involves coping with high volumes of heterogeneous log data from various sources. Processes to aggregate these disparate logs and trigger alerts when particular events occur are often automated today. However, these methods are typically based on regular expressions and statistical correlations and do not involve any interpretation of the context in which an event occurred and do not allow for inference or sophisticated rules. Inspection and in-depth analysis of log information to link events from various sources (e.g., firewall, syslog, web server log, database log) and establish causal chains has therefore largely remained a tedious manual search process that scales poorly with a growing number of heterogeneous log sources, log volumes, and the increasing complexity of attacks.

In this paper, we make the case for a semantic approach to tackle these challenges. By lifting raw log data and modeling their context, events can be linked to rich background knowledge, integrated based on causal relations, and interpreted in a context-specific manner. This builds a foundation for more comprehensive extraction of the meaning of events from unstructured log messages. Based on the results, we envision a platform to partly automate security monitoring and support analysts in coping with fast evolving threat landscapes, alleviate alert fatigue, improve situational awareness, and expedite incidence response.

2.1.1. Introduction

According to industry analyses, today's ICT systems are threatened on an unprecedented scale by increasingly sophisticated and targeted attacks [2, 1]. This has resulted in a series of widely publicized data breach cases [35] associated with enormous economic costs [36]. A key issue in this context is the lack of awareness about sophisticated multi-vector attacks, which are typically difficult to detect automatically using standard intrusion detection methods. This slow detection and limited understanding of the scope of security incidents severely inhibits organizations' ability to react adequately and take timely measures to mitigate and contain their impact. According to an industry survey, 59% of companies do not have adequate intelligence or are unsure about attempted attacks and their impact. Furthermore, 51% say their security solutions do not inform them or they are unsure if their solution can inform them about the root causes of an attack. [37]

These difficulties are not necessarily caused by a lack of data, as most hard- and software components provide comprehensive logging facilities that produce fine-grained and high-frequency information about their state and about observed events. Consequently, organizations' failure to detect and respond to security incidents is not caused by a lack of

clues that point to attacks, but by the rapid growth of log data and the manual analysis, which is becoming less and less feasible.

Apart from sheer volume, this is further complicated by the fact that these logs are typically weakly structured, use a variety of inconsistent formats and terminologies, and are spread across different files and disparate systems. To detect sophisticated multi-vector attacks, it is necessary to identify related events and connect them to create a complete picture for the identification of malicious activity. Isolated indicators are often inconspicuous in their local context and it is therefore necessary to harmonize and integrate disparate log information to obtain a complete picture. Furthermore, the interpretation of log information is highly context-specific, which makes it difficult for monitoring applications to identify relevant information without a deeper understanding of their context. Manual log analysis by human experts as a last resort does not scale in this context and there is a lack of automated mechanisms to integrate and interpret security information. Consequently, security analysts struggle to cope with the enormous volume of raw log data, to extract insights from these heterogeneous sources, and to identify and respond to increasingly complex attacks.

In this paper, we make the case for a semantic approach towards security analysis. This approach has the potential to reconcile today's highly fragmented log information landscape by extracting and interlinking relevant security information, facilitating automated inference, and providing security analysts with an integrated perspective that promotes understanding and improves situational awareness.

To this end, we develop a set of vocabularies for the uniform representation of log events and discuss how a solid conceptual foundation facilitates linking of disparate log information, extraction of relevant events, contextualization, and enrichment with background knowledge. Furthermore, we describe an architecture for semantic log processing that provides integrated access to log information via various interfaces. This creates a foundation for semantic security monitoring, incidence response, and forensic applications. In particular, the proposed approach will enable context-aware decision support and thereby overcome the limited scope and lack of interpretation capabilities of current security monitoring and response technologies such as virus scanners, intrusion detection systems (IDSs), and security incident and event management (SIEM) systems. Furthermore, the developed vocabularies provide a foundation for sharing threat intelligence across organizations.

The remainder of this paper is structured as follows: Section 2.1.2 introduces the wider context of the SEPSES semantic log analysis architecture (Section 2.1.2.1) and then specifically focuses on log extraction (Section 2.1.2.2) and vocabularies (Section 2.1.2.3); Section 2.1.3 discusses results from our prototypical implementation and illustrates how the approach addresses current challenges in security monitoring; Section 2.1.4 provides an overview of related work within industry and academia; Section 2.1.5 closes with conclusions and an outlook on future work.

2.1.2. Semantic security log analysis

The log extraction approach introduced in this paper constitutes the core of a larger semantic log analysis framework called SEPSES[1], which will provide a platform for semantic security monitoring, auditing, and forensics.

[1]Semantic Processing of Security Event Streams

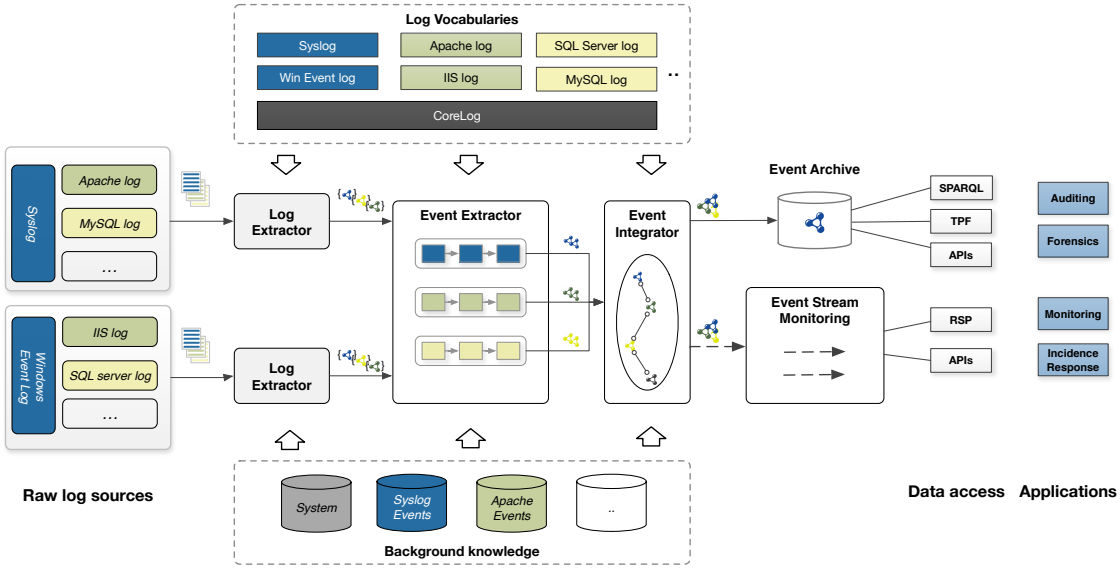

Figure 2.1.: Semantic log extraction architecture

2.1.2.1. SEPSES Architecture

The SEPSES architecture is designed to facilitate scalable semantic processing, integration, aggregation, and interpretation of heterogeneous logs in highly diverse environments. Its main components are illustrated in Figure 2.1.

In order to provide an integrated perspective on all relevant security information and enable semantic processing, it is first necessary to collect log data from a multitude of *log sources* (e.g., system logs, network logs, application logs, etc.) in different formats. A wide range of tools to acquire and store log information locally exists, but to provide an integrated perspective it is necessary to transfer local log information to a central repository and harmonize the heterogeneous log data. This typically involves local log agents (installed on local machines) which harvest log information from various sources and ship them to extractor components. This approach is flexible and allows for scalable log extraction at an appropriate level of centralization. The options range from fully decentralized extraction on the local machines, to fully centralized and load-balanced extraction on servers.

In our architecture, the *log extractor* components then transform the raw log data, received from multiple sources in various formats, into JSON-LD [38], a lightweight Linked Data format. Because most modern log management systems use JSON to encode and transport log messages, a key advantage of this approach is that JSON-LD annotations can be added easily in an efficient and scalable manner in order to make the data interoperable at web scale. This transformation from raw log data into an integrated JSON-LD log stream applies a set of well-specified log vocabularies and thereby solves syntactic interoperability challenges and – through alignment of formats and scales – creates a minimum level of semantic interoperability (cf. Section 2.1.2.2). To this end, the log extractors make use of a modular *log vocabulary* stack – described in Section 2.1.2.3 – that consists of `slog:core`, a foundation that provides basic terms to describe log messages irrespective of their source, and specialized vocabularies that extend the core vocabulary with log source-specific terms.

The harmonized stream of RDFized log messages in JSON-LD format is then forwarded to *event extraction pipelines*, which apply a sequence of enrichment, alignment, reconciliation, and integration steps to extract an interpretation of the meaning contained within the log messages[2]. These pipelines can combine a variety of information extraction techniques (e.g., named entity recognition, classifiers, and hand-written regular expressions) with inference, linking, and rule-based approaches. In summary, extraction pipelines link incoming log messages to rich *background knowledge* and obtain context-sensitive, machine-readable interpretations of security-related events.

Subsequently, extracted log events are sent to the *event integration* component, which establishes links between related events, irrespective of their original log source (e.g., combining the traces of a remote login event from the client, the server, and the firewall) and thereby establishes causal relations between isolated low-level events and integrates them into high-level events. For instance, a single high-level *login* event can be associated with multiple low-level events, i.e., a series of syslog events, authentication events, firewall events, etc. This step can be supported by statistical, learning-based, and (stream) reasoning-based event integration approaches.

Finally, the integrated event stream can be monitored with *RDF stream processing* techniques and forwarded to an *RDF log archive*, i.e., a triple store, for later analysis. The prepared log data can be made available for log auditing, analysis, and forensic applications via various interfaces. These interfaces, including APIs, SPARQL, and Linked Data Fragments (LDF) interfaces, constitute the data access layer of the architecture. One option for (near) real-time log monitoring by applications is to register continuous SPARQL queries.

In the following section, we focus specifically on the log extraction of the proposed architecture.

2.1.2.2. Log extraction

In our prototypical implementation, we use Logstash [39], an open source tool for collecting and parsing logs, as *log extraction* component. Furthermore, we use Filebeat to ship raw log data from remote agents installed on local systems to the log extractor.

To inject JSON-LD annotations into the JSON log stream, we configured Logstash with custom filters that restructure the data to conform to the SEPSES log vocabularies (Section 2.1.2.3) and add appropriate `@context` and `@id` annotations to yield valid JSON-LD output. A key benefit of this approach over plugin based configuration, hardcoded extraction or the use of generic RDF mapping tools (e.g., RML [40]) is that it can perform initial lifting from many sources in a flexible and scalable manner using tools that are optimized for large-scale, high-throughput log processing. Furthermore, it permits (near) real-time extraction and usage of the lifted log streams in stream processing scenarios.

The log vocabularies and JSON-LD format provide a well-defined interface that makes it possible to exchange the log extraction component (i.e., Logstash in our implementation) or combine it with other extraction tools, provided they can be adapted to produce JSON-LD output.

Whereas the interpretation of the log messages will be handled in the *event extraction* phase, the log extraction process does perform preliminary harmonization steps and results

[2]Out of the scope of the present paper

in an integrated, uniform representation of log streams. For instance, log parsing and transformation already harmonizes heterogeneous time stamp formats and represents them uniformly as `xsd:dataTime` properties.

Furthermore, the model of the RDF log streams is structured in a way that provides "attachment points" for subsequent enrichment, alignment, entity reconciliation, and linking of the log information to background knowledge in the event extraction and integration phases. These attachment points are modeled as auxiliary nodes that have a number of literal properties and a randomly generated identifier (UUID). These nodes are subsequently used for alignment (e.g., harmonizing severity levels in logs from different operating systems and hence different severity scales), entity reconciliation, and linking to background knowledge (e.g., link hosts that appear in a log stream to system background knowledge, irrespective of whether they are identified by IP, hostname, MAC address, etc.). In the event extraction and integration phase, the literal properties of these nodes will be used to identify the associated concepts and create `owl:sameAs` links between them.

2.1.2.3. Log vocabularies

The event extraction approach introduced in this paper transforms raw log events into a uniform RDF representation. This necessitates appropriate vocabularies to express the heterogeneous log information. As a foundation of all log vocabularies, the `core`[3] vocabulary provides basic concepts common for all log messages produced by a log extractor, most importantly the *LogEntry* class that represents a single generic log entry, as well as the *Host* and *Logtype* classes. The `core` log vocabulary also defines several properties, i.e., *hostName*, *ipAddress* with sub-properties *ip4Address* and *ip6Address*, *timestamp*, *logMessage*, and *logFilePath*.

A key principle in the design of our conceptual models is modularity, i.e., we organize the concepts into specialized vocabularies that can be mixed and matched, which is key to tackle log heterogeneity. This is achieved with log source-specific vocabularies that extend the *LogEntry* class with more specific subclasses. In our example illustrated in Figure 2.2, we use *ApacheLogEntry* provided by the *apacheLog*[4] vocabulary for Apache web server logs and *sysLogEntry* provided by *syslog*[5] vocabulary for Unix system logs.

In the design of these vocabularies, we aim to reuse existing vocabularies wherever possible. For instance, we reuse the existing `StatusCode` and `Method` concepts from the *http*[6] vocabulary for `ResponseType` and `RequestVerb`, respectively.

2.1.3. Implementation and preliminary results

To validate our approach w.r.t. current security monitoring challenges, we implemented the acquisition and extraction components (cf. Section 2.1.2.1) using Logstash as a platform for event collection and processing as well as our log vocabularies and system ontologies. We then set up a test system in a virtual machine that ran an Apache web server. From this machine, we acquired syslog and Apache log data using Filebeat and set up a custom Logstash configuration that transforms the log stream into semantically

[3] `https://purl.org/sepses/vocab/log/coreLog`, recommended prefix: `scl`
[4] `https://purl.org/sepses/vocab/log/apacheLog`, recommended prefix: `apacheLog`
[5] `https://purl.org/sepses/vocab/log/sysLog`, recommended prefix: `syslog`
[6] `http://www.w3.org/2011/http`

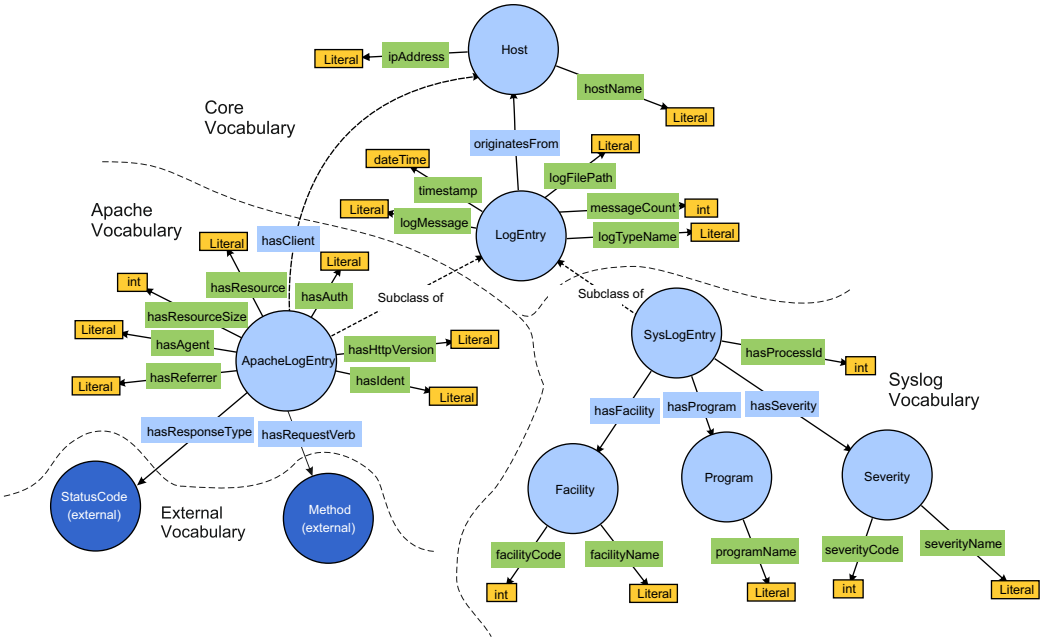

Figure 2.2.: Log vocabularies

explicit JSON-LD (cf. Section 2.1.2.2). In the following, we briefly discuss current major challenges in security monitoring and illustrate with minimalist examples how our semantic log processing approach can help to tackle them.

Volume The ever-growing volume of data relevant for security analyses poses a significant challenge – hard- and software components produce fine-grained and high-frequency log data. Whereas our approach does not reduce the amount of generated log data itself, it has the potential to reduce the data a security analyst must consider and inspect (e.g., by harmonizing logs, filtering noise, and aggregating events). Furthermore, the semantic integration provides a foundation for powerful semantic querying that can, for instance, use inference for generalizations or apply context in the automatic interpretation of events. This makes it easier to cope with large amounts of data and we expect that this approach will be vastly more efficient than manual auditing and more effective than simpler log correlation approaches that tend to produce a lot of false positives and thereby sometimes contribute rather than solve the problem of data volume.

Heterogeneity Logging data is typically highly verbose, redundant, incoherent, and poorly structured. Our current solution approach tackles this challenge on multiple levels. In particular, it 2.1.(i) resolves syntactic heterogeneity (e.g., varying time stamp formats in different logs) by lifting raw log data and describing it with a rich semantic log vocabulary; 2.1.(ii) performs entity resolution to uniquely identify entities even in heterogeneous log sources with varying identifiers (e.g., IP addresses vs host names, different user names in different applications, etc.); 2.1.(iii) tackles semantic heterogeneity (e.g., varying severity scales of different log standards) and offer mappings for an aligned

```
Apr 9 09:37:47 kabul-VirtualBox systemd[1]:  Mounted Huge Pages File System.
```

Figure 2.3.: Original raw log message

representation; 2.1.(iv) enables generalization (e.g., Apache and IIS are both web servers).

Figure 2.3 demonstrates an excerpt of the collected log data in original raw format and Listing 2.1 shows the JSON-LD output produced by the implemented *log extractor*.

```
{
   "@context":"http://sepses.ifs.tuwien.ac.at/contexts/syslog.jsonld",
   "logMessage":"Apr  9 09:37:47 kabul-VirtualBox systemd[1]: Mounted Huge
    Pages File System.",
   "timestamp":"2018-04-09T07:37:47.000Z",
   "hasProcessId":"1",
   "hasSeverity":{
      "severityName":"notice",
      "severityCode":"5"
   },
   "@type":"http://purl.org/sepses/vocab/log/sysLog#SysLogEntry",
   "hasLogType":"http://example.org/system#syslog",
   "@id":"http://example.org/logEntry#logEntry-ca1c3894-114f-432d-befd-
    abca46258e85",
   "hasProgram":{
      "programName":"systemd"
   },
   "logFilePath":"/var/log/syslog",
   "hasFacility":{
      "facilityCode":"1",
      "facilityName":"user-level"
   },
   "input":{
      "type":"log"
   },
   "originatesFrom":{
      "hostName":"kabul-VirtualBox"
   }
}
```

Listing 2.1: Example log message transformed into JSON-LD

The SPARQL query in Listing 2.2 demonstrates entity resolution by querying for log events from any host within the defined timeframe.

As the result excerpt in Table 2.1 shows, all events have a host name and IP address. This information comes from the background knowledge, by linking the host node from the log stream (`owl:sameAs`) to the host entity defined in the background knowledge as shown in Figure 2.4. Silk [41], an open source framework for integrating heterogeneous data sources, is an option to reach this goal. Listing 2.3 shows an `owl:sameAs` rule (`LinkageRule`) with a `levenshteinDistance` comparison to introduce a link if the host names match.

The host type is also defined in the background knowledge and could be automatically determined based on the software running on the host.

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>
prefix scl: <http://purl.org/sepses/vocab/log/coreLog#>
prefix sys_bg: <http://purl.org/sepses/bg/system#>

SELECT  ?time ?logType ?hostName ?ipAddress ?hostType ?message
WHERE { ?logEntry a scl:LogEntry;
            scl:originatesFrom ?host;
            scl:hasLogType ?logType;
            scl:logMessage ?message;
            scl:timestamp ?time.
        ?host sys_bg:hostType ?hostType;
            scl:hostName ?hostName;
            scl:ipAddress ?ipAddress.

 FILTER (?time > "2018-04-09T07:29:00"^^xsd:dateTime  &&
 ?time <"2018-04-09T07:34:00"^^xsd:dateTime)}
```

Listing 2.2: SPARQL Query that demonstrates entity resolution

```
<LinkageRule linkType="owl:sameAs">
     <Compare id="levenshteinDistance1" required="false" weight="1" metric="
    levenshteinDistance" threshold="0.0" indexing="true">
          <TransformInput id="lowerCase1" function="lowerCase">
            <Input id="sourcePath1" path="/syslog:hostName"/>
          </TransformInput>
          <TransformInput id="lowerCase2" function="lowerCase">
            <Input id="targetPath1" path="/bgk:hostName"/>
          </TransformInput>
          <Param name="minChar" value="0"/><Param name="maxChar" value="z"/>
       </Compare>
</LinkageRule>
```

Listing 2.3: Silk `owl:sameAs` LinkageRule

Table 2.1.: An excerpt of query result showing events have a host name and IP Address

time (xsd:dateTime)	**logType**	**hostName**	**ipAddress**	**hostType**	**message**
2018-04-09T07:29:15	`syslog`	kabul-VirtualBox	192.168.0.164	DatabaseServer	"org.debian.apt[683]: ..."
2018-04-09T07:31:45	`apache`	linux-Machine	192.145.0.124	WebServer	*"GET /presentations/ "*
2018-04-09T07:31:45	`apache`	linux-Machine	192.145.0.124	WebServer	*"GET /presentations/ ..."*
2018-04-09T07:31:45	`syslog`	kabul-VirtualBox	192.168.0.164	DatabaseServer	*"systemd-tmpfiles[3572]: ..."*
2018-04-09T07:31:45	`syslog`	kabul-VirtualBox	192.168.0.164	DatabaseServer	*"systemd[1]: Started ..."*

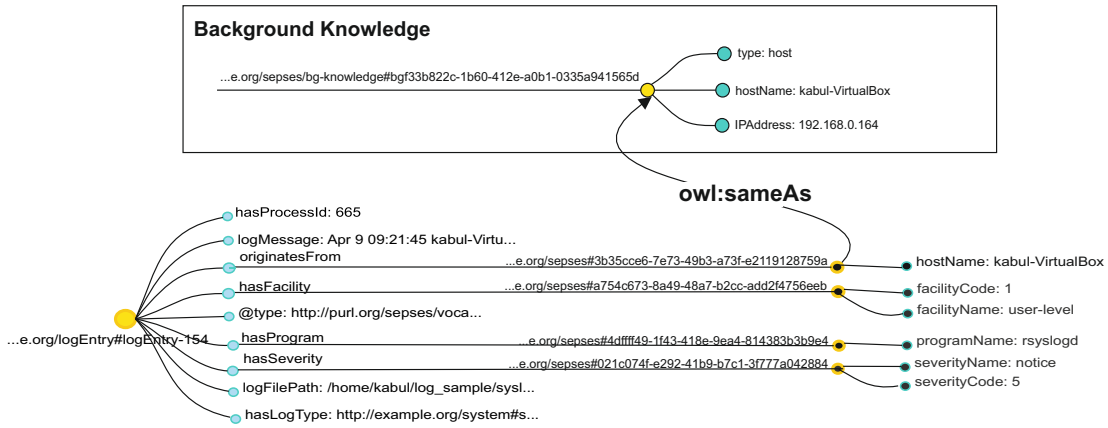

Figure 2.4.: SameAs linking between an auxiliary node from the log stream and a host entity from background knowledge

Integration In the course of monitoring events to identify activities relevant from a security perspective, it is often necessary to correlate isolated indicators from various log sources, hence, an integrated view is required. Our solution combines different log sources into a single log stream with a uniform core and using appropriate vocabularies for each log source. Whereas standard SIEM systems also provide some level of integrated access, our approach makes log data machine-readable and semantically integrate-able from the beginning rather than collecting raw log data and integrating it at query time. The query results in Table 2.1 already illustrate this by combining two disparate log sources (*syslog* and *apache*) and providing integrated results.

Context Security-related events are typically highly context-specific and hence, their interpretation requires extensive background knowledge. In our approach, we dynamically link rich background knowledge to the extracted log events and their associated entities, respectively.

Listing 2.4 demonstrates how vulnerability information from Common Vulnerabilities and Exposures (CVE) [42] can be linked to affected servers. Based on the background knowledge on installed software (organization-specific system knowledge) and CVE definitions mapped to software versions, an analyst could ask for all vulnerabilities on a specific host.[7] Naturally, any other background knowledge can be linked and queried in a similar fashion.

Automation Manual inspection of each log event is generally infeasible due to the vast amount of log data. Therefore, security analysts typically rely on reactive measures, such as alarms triggered by simple rules and subsequent analysis of potential causes. Our approach aims to reduce the burden on analysts by harmonizing various log sources and facilitating highly expressive semantic queries. Based on the log vocabularies introduced in this paper, we plan to develop methods for automated log interpretation and causal linking of events in the future.

[7]The integration of formally modeled vulnerability information will be an interesting area for future work

```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix scl: <http://purl.org/sepses/vocab/coreLog#>
prefix syslog: <http://purl.org/sepses/vocab/log/sysLog#>
prefix cve:  <http://purl.org/sepses/vocab/cve#>

SELECT  ?programName ?CVE ?CVE_Desc
WHERE { ?logEntry a scl:LogEntry;
            scl:originatesFrom ?host;
            syslog:hasProgram ?program.
            ?program cve:hasVulnerability ?CVE;
                     syslog:programName ?programName.
            ?CVE cve:cveDesc ?CVE_Desc.
            ?host scl:hostName "kabul-VirtualBox"^^rdfs:Literal}
```

Listing 2.4: CVE linking to affected servers

2.1.4. Related Work

Despite various initiatives to define common logging standards, a variety of log sources, inconsistent log timestamps and content, and inconsistent log formats [7] remain a challenge to this day. In the following, we partition related work into industrial practice, log vocabularies, and approaches focused on semantic log analysis.

Industry A variety of logging systems are in production use today, such as operating system logs (e.g., syslogd [43] and Windows Event Logs [44]), web server logs (e.g., Extended Log File Format [45], NGINX logging [46], W3C Extended log file format [47]), database logs, firewall logs, etc. To collect and manage log messages from multiple sources, a variety of services have emerged, including Splunk [48], Papertrail [49], Librato [50], and Logstash [39], but they do not strive for semantic integration. Key-value based log formats, such as LOGFMT [51] have become a common approach to format log data. To extract information from raw textual log data, regular expressions are a widely used option.

Log vocabularies An early initiative to standardize heterogeneous vocabularies to express electronic systems' events in a uniform, device-independent manner is Common Event Expression (CEE), which was driven by MITRE [52]. CEE distinguishes the taxonomy (semantic event type), the log syntax (instance data), and the log transport component for exchange. In 2014, however, MITRE stopped all work on CEE due to discontinued funding of the U.S. Government.

Heterogeneous log file formats have been identified as a serious impediment to the effectiveness of security controls and intrusion detection systems [53]. In the same paper, the authors discuss disadvantages of existing common log formats for normalization of security events and propose a new, extensible log format that is specifically designed for intrusion detection purposes.

In another work that aims to apply semantics to log information [54], the authors focus on web application firewalls and introduce ontologies to avoid ambiguities and vagueness. They take a first step and propose *OntoSeg*, an ontology to model web application firewall logs. The ontology was built in Protégé by inspecting actual log files and identifying major classes and their relationships manually. Compared to our approach, their scope is rather

narrow and focused on ontology engineering rather than large-scale RDF processing.

Another ontology-driven approach based on the well-known java logging utility log4j [55] is RLOG [56]. It covers the basic classes for application logging, such as *Log Entry*, *Log Level*, *Status Code* and the fundamental properties, like *Logging message* and *Logging date*. In summary, this ontology comprises some basic concepts, but provides far less than the expressiveness we require. Another ontology, with more depth is the OpenLink Logging Ontology [57]. It uses FOAF, Dublin Core and OWL 2. The SPECIAL Policy Log Vocabulary (Splog) [58] offers a vocabulary for log data that models processing and sharing events that should comply with a given consent provided by a data subject. Due to its specific purpose for consent checking, it does not cover the necessary elements to model common log sources. The log provenance concepts from this vocabulary could be a useful future extension for our approach.

Semantic log analysis An early approach towards semantic modeling of the logging domain was developed in the context of computer forensics [59, 60]. The authors argue that context information, such as environmental data and configuration information, is vital in forensic analyses. Their overall goal is to develop a general solution (*FORE* - "Forensics of Rich Events") to facilitate human understanding and automated inference. To this end, they define an OWL ontology that captures event information and allows for generalization and composition (e.g., *DownloadExecutionEvent*, which is composed of a *FileReceiveEvent* and an *ExecutionEvent*). The authors develop a custom rule language, based on F-Logic, to express correlation rules. An event browser provides the interface to investigate events and explore detected causalities.

Although FORE aims for a similar goal as our approach (i.e., semantic log analysis), a key difference is its strong focus on forensics. This is reflected in more heavyweight ontology engineering approaches and their choice of technologies. Our approach is grounded in more recently developed Linked Data and Semantic Web technologies such as JSON-LD and state-of-the-art technologies for scalable log harvesting. This will allow us to implement fast extraction pipelines and potentially allow processing of semantic log streams in (near) real-time, based on recent advances in RDF stream processing technologies [61]. Finally, whereas FORE's ad-hoc ontologies were designed to illustrate particular use cases, we aim to develop a modular vocabulary stack that can cover a variety of different log sources.

Another early approach that tackles the problem that manual review of syslog messages is tedious and error prone is discussed in [62]. The authors apply machine-learning algorithms to detect anomalies in the syslog message stream through classification and investigate cause-effect hypotheses in large multiple-source event log sets. Automatically generated word-granular regular expressions for system event logs are used. They consider their approach as an alternative to expert systems, which require a set of rules that are time-intensive to create and maintain.

A set of publications focuses on log files in connection with web usage mining. Properties of web log data, such as their format, location, and access rules are discussed in [63]. Another Log Ontology (LO) for web usage mining is introduced in [64]. In this ontology, *ComplexEvents* are composed out of *AtomEvents*, which is similar to the idea of extracting high-level events from low-level events discussed in this paper.

2.1.5. Conclusion

The SEPSES architecture introduced in this paper aims to create a versatile platform for semantic security log analysis to facilitate effective and efficient security monitoring, auditing, forensics, and incidence response applications.

This platform will support security analysts and complement their human intuition and expertise with machine interpretation of security-related information. Thereby, we aim to reduce the manual work necessary to connect individual pieces of information in disparate log sources and support security analyses by contextualizing, integrating, and linking disparate log information. This will create a semantically explicit, context-rich environment that has the potential to complement existing intrusion detection techniques and increase the precision of alerts and reduce security analysts' "alert fatigue" due to a large number of false positives they tend to generate.

In this paper, we focused on the foundation of the proposed approach to bring meaning to vast volumes of raw textual log data, i.e., log vocabularies, log acquisition, and initial extraction. We outlined how we combine state-of-the-art semantic web and log acquisition technologies and illustrated how the vocabularies provide a foundation for alignment, integration, and linking to background knowledge. Based on a prototypical implementation and illustrative examples, we also highlighted how the proposed approach can tackle current challenges in security analysis.

In future work, we will focus on the interpretation of individual events as well as semantic techniques to establish relations between events and link them to security background knowledge (e.g., vulnerabilities and threat intelligence). Next, we will investigate scalable approaches for the storage and semantic querying of large log archives. To this end, we will evaluate big data approaches and scalable linked data querying interfaces and explore architectural options for scalable semantic log processing infrastructures. Finally, another promising venue for future work on semantic security monitoring is RDF stream processing.

2.2. An integrated knowledge graph for cybersecurity information

Elmar Kiesling, Andreas Ekelhart, Kabul Kurniawan, Fajar Juang Ekaputra Published as *"The SEPSES knowledge graph:An integrated resource for cybersecurity"*, In: The Semantic Web – ISWC 2019. vol.11779 p. 198-214.[28].

Abstract

This paper introduces an evolving cybersecurity knowledge graph that integrates and links critical information on real-world vulnerabilities, weaknesses and attack patterns from various publicly available sources. Cybersecurity constitutes a particularly interesting domain for the development of a domain-specific public knowledge graph, particularly due to its highly dynamic landscape characterized by time-critical, dispersed, and heterogeneous information. To build and continually maintain a knowledge graph, we provide and describe an integrated set of resources, including vocabularies derived from well-established standards in the cybersecurity domain, an ETL workflow that updates the knowledge graph as new information becomes available, and a set of services that provide integrated access through multiple interfaces. The resulting semantic resource offers comprehensive and integrated up-to-date instance information to security researchers and professionals alike. Furthermore, it can be easily linked to locally available information, as we demonstrate by means of two use cases in the context of vulnerability assessment and intrusion detection.

2.2.1. Introduction

Security and privacy have become key issues in today's modern societies characterized by a strong dependence on Information and Communication Technologies (ICT). Security incidents, such as ransomware and data theft, are widely reported in the media and illustrate the ongoing struggle to protect ICT systems. In their mission to secure systems, security professionals rely on a wealth of information such as known and newly identified vulnerabilities, weaknesses, threats, and attack patterns. Such information is collected and published by, e.g., Computer Emergency Response Teams (CERTs), research institutions, government agencies, and industry experts. Whereas a lot of relevant information is still shared informally as text, initiatives to make security information available in well-defined structured formats, largely driven by MITRE[8] and NIST[9], have made significant progress and resulted in a wide range of standards [65]. These standards define high-level schemas for cybersecurity information and have resulted in various structured lists that are available for browsing on the web and for download in heterogeneous structured formats. This wealth of cybersecurity data is highly useful, but the current approach for sharing it is associated with several limitations: First, individual entities and data sets remain isolated and cannot easily be referenced and linked from other data sets. Second, whereas the governed schemas provide a well-defined structure, the semantics are not as well-defined. This limits the potential for integration and automated machine interpretation. Consequently, the resulting abundance of data raises challenges for security analysts and professionals who

[8] `https://www.mitre.org`
[9] `https://nist.gov`

have to keep track of all the available sources and identify relevant information within them.

In this paper, we propose that integrating cybersecurity information into a regularly updated, public knowledge graph can overcome these limitations and open up exciting opportunities for cybersecurity research and practice. Thereby, it is possible not only to query public cybersecurity information, but also to use it to contextualize local information. As we illustrate with two example use cases in this paper, this facilitates applications such as *(i)* improved vulnerability assessment by automatically determining which new vulnerabilities affect a given infrastructure, and *(ii)* improved incident response through better contextualization of intrusion detection alerts.

Our main contributions can be summarized as follows: For cybersecurity research and practice, we advance the state of the art by providing an integrated up-to-date view on cybersecurity knowledge in a semantically explicit representation. Furthermore, we provide tools and services to query and make use of this interlinked knowledge graph. From a semantic web research perspective, we illustrate how Linked Data principles can be applied to combine local and public knowledge in a highly dynamic environment characterized by fast-changing, dispersed, and heterogeneous information. To this end, we develop an ETL pipeline that integrates newly available structured data from public sources into the knowledge graph, which involves acquisition, extraction, lifting, linking, and validation steps. We provide the following resources[10]: *(i)* vocabularies for the rich representation and interlinking of security-related information based on five well-established standards in the cybersecurity domain. *(ii)* a comprehensive SEPSES Cybersecurity Knowledge Graph (KG)[11] with detailed instance data[12] accessible through multiple interfaces. *(iii)* an ETL workflow published as open source that updates the knowledge graph as new information becomes available. *(iv)* a website[13] that provides documentation, status information, and pointers to the various access mechanisms provided. *(v)* a set of services to access the data, i.e., a SPARQL endpoint, a triple pattern fragments interface, a Linked Data interface, and download options for the whole data set as well as various subsets.

This semantic approach can provide a foundation for tools and services that support security analysts in applying external security knowledge and efficiently navigating dynamic security information. Ultimately, this should contribute towards improved cybersecurity knowledge sharing and increased situational awareness, both in large organizations that have dedicated security experts who are often overwhelmed by the large amount of information, and in smaller organizations that do not have the resources to invest in specialized tools and experts.

The remainder of this paper is organized as follows: Section 2.2.2 provides an overview of related work; Section 2.2.3 covers construction and maintenance of the KG, including vocabularies, data acquisition mechanisms, and updating pipelines; Section 2.2.4 provides an overview of the provided mechanisms to access the data in the KG and discusses its sustainability, maintenance and extensibility; Section 2.2.5 illustrates the usefulness of the resource by means of two example use cases; Section 2.2.6 concludes the paper with an outlook on future work.

[10] Available at `https://w3id.org/sepses/cyber-kg`
[11] *Semantic Processing of Security Event Streams* is an ongoing research project
[12] 36,594,388 triples as of July 2, 2019
[13] `https://sepses.ifs.tuwien.ac.at`

2.2.2. Related Work

Various information security standards, taxonomies, vocabularies, and ontologies have been developed in academia, industry, and government agencies. In this subsection, we review these lines of related work, which fall into two broad categories: *(i)* standard data schemas for information sharing in the cybersecurity domain (covered in Section 2.2.2.1) and *(ii)* higher-level conceptualizations of security knowledge (covered in Section 2.2.2.2). We conclude the subsection by identifying the gap between those strands of work.

2.2.2.1. Standard Data Schemas

Efficient information exchange requires common standards, particularly in highly diverse and dynamic domains such as cybersecurity. Hence, a set of standards has emerged that define the syntax of description languages for structured cybersecurity information and the semantics associated with those descriptions in natural language. Some of these standards are driven by traditional standardization bodies such as ISO, ITU, IEEE or IETF. The majority, however, are contributed by open source communities or other entities such as MITRE[14], a not-for-profit research and development cooperation.[15]

Salient examples for information sharing standards, all of which are integrated in the knowledge graph presented in this paper, include Common Vulnerabilities and Exposures (CVE)[16] for publicly known vulnerabilities, Common Attack Pattern Enumeration and Classification (CAPEC)[17] for known attack patterns used by adversaries, Common Weakness Enumeration (CWE)[18] for software security weaknesses, Common Platform Enumeration (CPE)[19] for encoding names of IT products and platforms, and Common Vulnerability Scoring System (CVSS)[20] for vulnerability scoring. These standards are widely used by security practitioners and integrated into security products and services, but they also serve as an important point of reference for research.

2.2.2.2. Security Ontologies

A related line of academic research aims at a high-level conceptualization of information security knowledge, which has resulted in numerous ontologies (e.g., [66, 67, 68, 69, 70, 71, 72]) that typically revolve around core concepts such as *asset*, *threat*, *vulnerability*, and *countermeasure*. The resulting security ontologies are typically scoped for particular application domains (e.g., risk management, incident management). The high-level ontology developed in [73], for instance, mainly focuses on malware and aspects such as *actors*, *victims*, *infrastructure*, and *capabilities*. The authors argue that expressive semantic models are crucial for complex security applications and name Open Vulnerability and Assessment Language (OVAL), CPE, Common Configuration Enumeration (CCE), and CVE as the most promising starting points for the development of a cybersecurity ontology. Inspired by that work, Oltramari et al. [74] introduce an ontological cyber security framework

[14] `https://www.mitre.org`
[15] For a review of standards for the exchange of security information, cf. [65].
[16] `https://cve.mitre.org`
[17] `https://capec.mitre.org`
[18] `https://cwe.mitre.org`
[19] `https://cpe.mitre.org`
[20] `https://www.first.org/cvss/`

that comprises a top-level ontology based on DOLCE, a mid-level ontology with security concepts (e.g., *threat*, *attacker*, *vulnerability*, *countermeasure*), and a domain ontology of cyber operations including defensive and offensive actions. A comprehensive survey and classification of similar security ontologies can be found in [75].

More recently, various initiatives aimed at developing security ontologies that cover the standard schemas outlined in Section 2.2.2.1, including an ontology for CVE vulnerabilities [76, 77, 78] that can be used to identify vulnerable IT products. Ulicny et al. [79] take advantage of existing standards and markup languages such as Structured Threat Information eXpression (STIX), CAPEC, CVE and CybOX and transform their respective XML schemas through XSLT translators and custom code into a Web Ontology Language (OWL) ontology. Furthermore, they integrate external information, e.g., on persons, groups and organizations, IP addresses (WhoIs records), geographic entities (GeoNames), and "killchain" phases. In an application example, the authors illustrate how this can help to inspect intrusion detection events, e.g., by mapping events to kill chain stages and obtaining more information about threat actors based on IP addresses.

As part of a research project (STUCCO), Iannacone et al. [80] outline an approach for a cybersecurity knowledge graph and note that they aim to integrate information from both structured and unstructured data sources. Some extraction code and JSON schema data is available on the project website[21] but no integrated knowledge graph has been published. In a similar effort, Syed et al. [81] integrate heterogeneous knowledge schemas from various cybersecurity systems and standards and create a Unified Cybersecurity Ontology (UCO) that aligns CAPEC, CVE, CWE, STIX, Trusted Automated eXchange fo Indicator Information (TAXII)[22] and Att&ck[23]. Whereas most ontologies proposed in the literature are not publicly available, UCO is offered for download[24], including some example instances from industry standard repositories. However, the instance data in the dump is neither complete nor updated, and there is no public endpoint available. Finally, the Cyber Intelligence Ontology[25] is another example of an ontology that is available for download in RDF and offers classes, properties and restrictions on many industry standards, but no instance data.

Overall, a review of related work shows that although basic concepts in the cybersecurity domain have been formalized repeatedly, no model has so far emerged as a standard. Furthermore, the proposed high-level conceptualizations typically lack concrete instance information.

On the other hand, there are many standards for cybersecurity information sharing and the information is published in various structured formats[26], navigable on the web and/or available for download; however, there is no integrated view on this scattered, heterogeneous information. Hence, each application that makes use of the published data has to parse and interpret each source individually, which makes reuse, machine interpretation, and integration with local data difficult. In the following subsection, we describe how an evolving cybersecurity knowledge graph that provides an integrated perspective on the cybersecurity landscape can fill this gap.

[21] `https://github.com/stucco`
[22] `https://oasis-open.github.io/cti-documentation/`
[23] `https://attack.mitre.org`
[24] `https://github.com/Ebiquity/Unified-Cybersecurity-Ontology`
[25] `https://github.com/daedafusion/cyber-ontology`
[26] Most commonly as XML or JSON files

2.2.3. Knowledge Graph Construction and Evolution

To construct and regularly update the SEPSES Cybersecurity KG, we define a set of vocabularies, described in Section 2.2.3.1, and an architecture for initial ingestion and incremental updating of the graph, covered in Section 2.2.3.2. Publication via Linked Data (LD), Triple Pattern Fragments (TPF), a SPARQL endpoint, and RDF dumps are covered in Section 2.2.4.

2.2.3.1. Conceptualization and Vocabularies

To model the domain of interest, we started with a survey and found that the vast majority of conceptualizations described in the literature are not available online. Those that were available did not provide sufficiently detailed classes and properties to represent all the information available in the cybersecurity repositories we target.

Hence, we opted for a bottom-up approach starting from a set of well-established industry data sources. We structured our vocabularies based on the schemas used to publish existing instance data and chose appropriate terms based on the survey of existing conceptualizations. In choosing this approach, our main design goal was to include the complete information from the original data sources and make the resulting knowledge graph self-contained. To facilitate mapping to other existing conceptualizations, we kept the Resource Description Framework (RDF) model structurally similar to the data models of the original sources. This should make it easy for users already familiar with the original data sources to navigate and integrate our semantic resource. Furthermore, we can easily refer to the original documentation and examples in the vocabularies. We then created

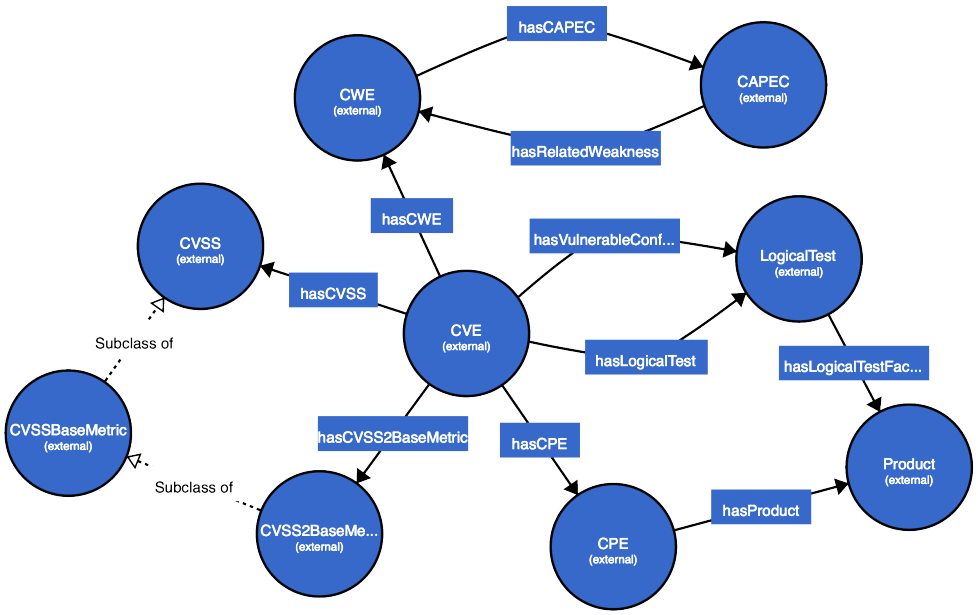

Figure 2.5.: SEPSES Knowledge Graph Vocabulary High-level Overview

a schema that covers the following security information repositories (cf. Figure 2.5 for a

high-level overview).[27]

CVE is a well-established industry standard that provides a list of identifiers for publicly known cybersecurity vulnerabilities. In addition to CVE, we integrate the National U.S. Vulnerability Database (NVD), which enriches CVEs with additional information, such as security checklist references, security-related software flaws, misconfigurations, product names, and impact metrics. We represent this information in the CVE class, which includes data type properties such as `cve:cveId`, `cve:description`, `cve:issued` and `cve:modified` timestamps. Based on the NVD information, we can link CVE to affected products (`cve:hasCPE`), vulnerable configurations (`cve:hasVulnerableConfiguration`), impact scores (`cve:hasCVSS`), related weaknesses (`cve:hasCWE`), and external references (`cve:hasReference`).

CVSS provides a quantitative model to describe characteristics and impacts of IT vulnerabilities. It is well-established as a standard measurement system for organizations worldwide. We integrate the CVSS scores provided by NVD, and model the CVSS metrics by means of the CVSSBaseMetric, CVSSTemporalMetric, and CVSSEnvironmentalMetric classes to comply with the CVSS specification[28].

CPE provides a structured naming scheme for IT systems, software, and packages based on URIs. NIST hosts and maintains the CPE Dictionary, which currently is based on the CPE 2.3 specification. We represent CPEs with the CPE class and reference product information with `cpe:hasProduct`. Furthermore, we define a set of properties that describe a product, such as product name, version, update, edition, language, etc. The vendor of each product is modeled as a Vendor and referenced by `cpe:hasVendor`.

CWE is a community-developed list of common software security weaknesses that contains information on identification, mitigation, and prevention. NVD vulnerabilities are mapped to CWEs to offer general vulnerability information. This information is modeled using the CWE class and a set of datatype properties such as `cwe:id`, `cwe:name`, `cwe:description`, and `cwe:status`, as well as object properties, to, e.g., link applicable platforms (`cwe:hasApplicablePlatform`), attack patterns (`cwe:hasCAPEC`), consequences (`cwe:hasCommonConsequence`), related weaknesses to model the CWE hierarchy (`cwe:hasRelatedWeakness`) and potential mitigations (`cwe:hasPotentialMitigation`).

CAPEC is a dictionary of known attack patterns used by adversaries to exploit known vulnerabilities, and can be used by analysts, developers, testers, and educators to advance community understanding and enhance defenses. We model CAPEC patterns in the CAPEC class with datatype properties such as `capec:id`, `capec:name`, `capec:likelihoodOfAttack`, and `capec:description`. Additional information is linked via object properties such as consequences `capec:hasConsequences`, required skills `capec:hasSkillRequired`, attack prerequisites `capec:prerequisites`, and attack consequences `capec:hasConsequence`.

Most of these data sets define identifiers for key entities such as vulnerabilities, weaknesses, and attack patterns and reuse some concepts from other standards (e.g., CPE names and CVSS scores are used within CVE). In the next subsection, we will describe how we leverage these references to link the data.

[27] The figure omits detailed concepts for the sake of clarity. The complete vocabularies can be found at `https://github.com/sepses/vocab`

[28] `https://www.first.org/cvss/specification-document`

2.2.3.2. ETL Process

Figure 2.6 illustrates the overall architecture and the data acquisition, resource extraction, entity linking and validation, storage and publication steps necessary to provide a continuously updated cybersecurity knowledge graph. In the following, we describe the steps in the core Extraction, Transformation, Loading (ETL) process that periodically checks and digest data from the various sources.

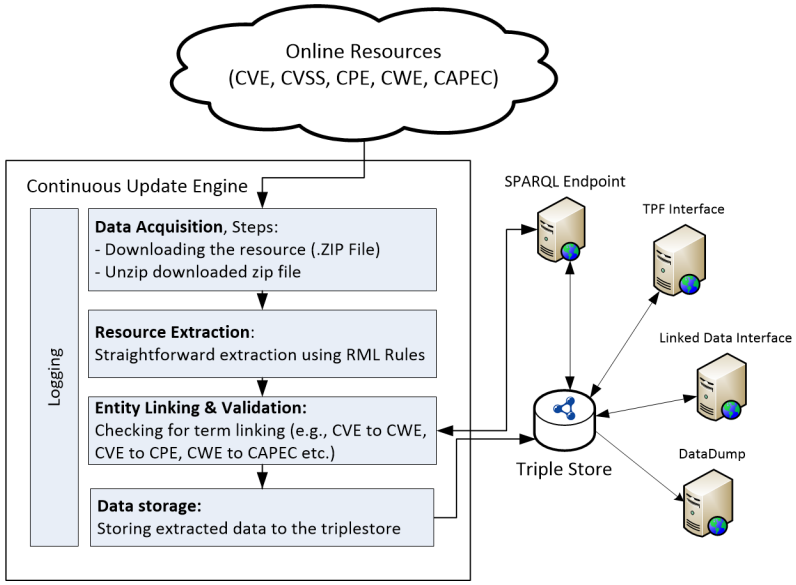

Figure 2.6.: Architecture: ETL process and publishing

Data Acquisition We populate our KG using data from various sources that provide data on their respective web sites for download in heterogeneous formats such as CSV, XML, and JSON. These cybersecurity data sources are updated regularly to reflect changes in the real-world. CVE data, for instance, is typically updated once every two hours.[29] In order to capture changes and reflect them in the knowledge graph, our ETL engine will regularly poll for updates and ingest the latest version of the sources.

Resource Extraction We use the caRML engine[30] to transform the original source files from their various formats. Furthermore, we use Apache Jena[31] to transform the raw RDF data obtained from the RML mappings into the structure of the final ontology. Initially, we developed RDF Mapping Language (RML) transformation mappings that utilized specific features from caRML, such as *carml:multiJoinCondition*. Due address performance issues, however, we decided to restructure the initial mapping into generic RML mappings that do not involve specific constructs from caRML, which improved performance considerably.[32] Because the original data sources have an established ID system, instance ID generation was straightforward for most sources (i.e., CWE, CVE CAPEC, and CVSS). For CPE,

[29] cf. `https://nvd.nist.gov/vuln/data-feeds`
[30] `https://github.com/carml/carml`
[31] `https://jena.apache.org`
[32] In some cases, this reduced processing time from appr. an hour to less than a minute.

however, the instance name is a composite of several naming elements (e.g., product name, part, vendor, version, etc.), separated by special characters. To solve the issue, we use `XPath` functions to clean and produce a unique name for each CPE instance.

Entity linking & Validation In this part of the ETL process, we link data from different sources based on common identifiers in the data. Each CWE weakness, for example, typically references several CAPEC attack patterns. Based on these identifiers, we create direct links between associated resources. Specifically for CPE, we deploy the exact same `XPath` functions for its identifier in the two sources (CPE and CVE) where CPE instances are generated, to make sure that these data can be linked correctly. To ensure data quality, we validate the generated RDF with SHACL to make sure that the necessary properties are included for each generated individual. Furthermore, we validate whether the resulting resources are linked correctly, as references to identifiers that are not or no longer available in other data sets are unfortunately a common issue. As an example, a CVE instance may have a relation to another resource such as a CPE identifier. In this case, the validation mechanism will check whether the referenced CPE instance exists in the extracted CPE data, log missing instances and create temporary resources for them.

Data storage We store the extracted data in a triple store and generate statistics such as parsing time, parsing status (success or fail), counts of instances, links, and generation time. To make sure that the data is continuously up to date, we wrote a set of bash scripts that are set to be executed in regular intervals to trigger the knowledge generation process and store the result in the triple store. To date, this resulted in more than half a million instances and 36 million triples; Section 2.2.3.2 provides a breakdown of the generated data.

	CVE	**CVSS**	**CPE**	**CWE**	**CAPEC**	**SnortRules**
Axioms	68	248	111	256	149	486
Class Count	7	9	5	10	8	10
Object Property Count	6	8	4	9	6	10
Data Property Count	8	37	18	40	22	103
Individual count	123,005	123,220	393,695	808	516	3,488

Table 2.2.: SEPSES Knowledge Graph Statistics[33]

2.2.4. Knowledge Graph Access

The SEPSES web site[34] provides pointers to the various resources covered in this paper, i.e., the LD resources[35], the SPARQL[36] and TPF query interfaces[37], a download link for the complete RDF snapshots[38], and the ETL engine source code[39]. This allows users to choose the most appropriate access mechanism for their application context.

[33] As per July 2, 2019.
[34] `https://w3id.org/sepses`
[35] e.g., `https://w3id.org/sepses/resource/cve/CVE-2014-0160`
[36] `https://w3id.org/sepses/sparql`
[37] `https://ldf-server.sepses.ifs.tuwien.ac.at`
[38] `https://w3id.org/sepses/dumps/`
[39] `https://github.com/sepses/cyber-kg-converter`

2.2.4.1. Sustainability, Maintenance and Extensibility

The SEPSES KG is being developed jointly by TU Wien and SBA Research, a well-established research center for information security that is embedded within a network of more than 70 companies as well as 15 Universities and research institutions. Endpoints and data sets are hosted at TU Wien and maintained as part of the research project SEPSES, which aims to leverage semantic web technologies for security log interpretation. During this project, we will extend the KG and leverage it as background knowledge in research on semantic monitoring and forensic analysis.

To keep the KG in sync with the evolving cybersecurity landscape, we will continue to automatically poll and process updates of the original raw data sources. We choose our polling strategy according to the varying update intervals of the data sources: CVEs are typically updated once every two hours, CPEs are typically updated daily. CWE and CAPEC are less dynamic and are updated approximately on a yearly schedule.

Furthermore, SBA Research has an active interest in developing and diffusing the KG internally and within its partner network, which will secure long-term maintenance beyond the current research project. We also expect the KG to grow and establish an active external user community during that time. To this end, we publish our vocabularies and the source code under an open source MIT license[40] and encourage community contributions.[41] Adoption success will be measured *(i)* based on access statistics (web page access, SPARQL queries, downloads, etc.), and *(ii)* the emergence of a community around the knowledge graph (code contributions, citations, attractiveness as a linked data target, number of research and community projects that make use of it, etc.).

2.2.5. Use Cases

In this subsection, we illustrate the applicability of the cybersecurity knowledge graph by means of two example scenarios.

2.2.5.1. Vulnerability Assessment

In security management, identifying, quantifying, and prioritizing vulnerabilities in a system is a key activity and a necessary precondition for threat mitigation and elimination and hence for the successful protection of valuable resources. This Vulnerability Assessment (VA) process can involve both active techniques such as scanning and penetration testing and passive techniques such as monitoring the wealth of public data sources for relevant vulnerabilities and threats. For the latter, keeping track of all the relevant information and determining relevance and implications for the assets in a system is a challenging task for security professionals. In this scenario, we illustrate how the developed knowledge graph can support security analysts by linking organization-specific asset information to a continuously updated stream of known vulnerabilities.

Setting: To illustrate the approach, we modeled a simplified example network comprising of three Hosts – two workstations, a server – and NetworkDevices. All hardware components are sub classes of ITAssets. Furthermore, we model the software installed

[40] `https://opensource.org/licenses/MIT`

[41] The original raw data are published by MITRE with a no-charge copyright license and by NVD without copyright.

on each HOST by means of the `hasInstalledProduct` property that links the host to a CPE specification. To determine the potential severity of an impact, we also include DATAASSETS, their `classification` (*public*, *private*, *restricted*), and their storage location (`storedOn` HOST) in the model. In practice, the modeling of a system can be supported by existing IT asset/software discovery and inventory tools.

Query 1: Once a model of the local system has been created, the vulnerability information published in the cybersecurity knowledge graph can be applied and contextualized by means of a federated SPARQL query. Note that we also provide a TPF interface for efficient querying. In particular, a security analyst may be interested in all known vulnerabilities that potentially apply to each host, based on the software that is installed on it (cf. Listing 2.5). Table 2.3 shows an example query result. Each resource in the table points to its Linked Data representation, which can serve as a starting point for further exploration. Note that as new vulnerability information becomes available and is automatically integrated into the knowledge graph through the process described in Section 2.2.3, the query results will automatically reflect newly identified vulnerabilities.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX asset: <http://w3id.org/sepses/vocab/bgk/assetKnowledge#>
PREFIX cve: <http://w3id.org/sepses/vocab/ref/cve#>
PREFIX cpe: <http://w3id.org/sepses/vocab/ref/cpe#>
PREFIX cvss: <http://w3id.org/sepses/vocab/ref/cvss#>
PREFIX cwe: <http://w3id.org/sepses/vocab/ref/cwe#>

SELECT  distinct ?hostName str(?ip) as ?IP ?product
(group_concat(?cveId) as ?cveIds)  from
<http://localhost:8890/localdata2>
WHERE {
      ?s a asset:Host.
      ?s rdfs:label ?hostName.
      ?s asset:ipAddress ?ip.
      ?s asset:hasProduct ?p.
SERVICE <http://sepses.ifs.tuwien.ac.at/sparql> {
      ?cve cve:hasCPE ?p .
      ?cve cve:id ?cveId.
    ?p cpe:title ?product .
  }
}
group by ?hostName ?ip ?product
```

Listing 2.5: Vulnerability Asessment Query 1 – Vulnerable Assets

hostName	**IP**	**product**	**cveIds**
DBServer1	192.168.1.3	Windows Server 2016	CVE-2016-3332, ..., CVE-2017-8746
Workstation1	192.168.1.1	Windows 10	CVE-2016-3302, ..., CVE-2015-2554

Table 2.3.: Vulnerability Assessment Query 1 – Results

Query 2: In order to assess the potential impact that a newly identified vulnerability may have, it is critical to asses which data assets might be exposed if an attacker can successfully exploit it. In the next step, we hence take advantage of the modeled data

assets and formulate a query (cf. Listing 2.6) to retrieve the most severe vulnerabilities, i.e., those that affect hosts that store *sensitive private* data (classification value = 1) and have a *complete* confidentiality impact (as specified in CVSS). Table 2.4 shows the query result and illustrates how such immediate analysis can save time by avoiding manual investigation steps.

```
SELECT DISTINCT ?hostName ?cveId
?confidentiality as ?conf ?cvssScore AS ?score ?dataAsset ?classification AS
    ?class ?consequence
FROM <http://localhost:8890/localdata>
WHERE {
    ?s a asset:Host.
    ?s rdfs:label ?hostName.
    ?s asset:hasProduct ?product.
    ?s asset:hasDataAsset ?dt.
    ?dt rdfs:label ?dataAsset.
    ?dt asset:hasClassification ?c.
    ?c rdfs:label ?classification.
    ?c asset:dataClassificationValue ?cv
FILTER (?confidentiality = "COMPLETE")
FILTER (?cv = 1)

SERVICE <http://sepses.ifs.tuwien.ac.at/sparql> {
    ?cve cve:hasCPE ?product .
    ?cve cve:id ?cveId.
    ?cve cve:hasCVSS2BaseMetric ?cvss2.
    ?cvss2 cvss:confidentialityImpact ?confidentiality.
    ?cvss2 cvss:baseScore ?cvssScore.
    ?cve cwe:hasCWE ?cwe.
    ?cwe cwe:hasCommonConsequence ?cc.
    ?cc cwe:consequenceImpact ?consequence
   }
}
```

Listing 2.6: Vulnerability Assessment Query 2 – Critical Vulnerabilities[42]

Exploration: The query results can serve as a starting point for further exploration of the Linked Data in the knowledge graph[43]. By navigating it, a security analyst can access information from various sources such as, e.g., attack prerequisites and potential mitigations from CAPEC, weakness classifications and potential mitigations from CWE, and scorings from CVSS.

hostName	cveId	conf	score	dataAsset	class	consequence
Workstation2	2016-1646	COMPLETE	9.3	EmpData	Private	Read Memory
Workstation2	2016-1653	COMPLETE	9.3	EmpData	Private	DoS: Crash, Exit...
Workstation2	2016-1583	COMPLETE	7.2	EmpData	Private	DoS: Resource Cons...
Workstation2	2016-1583	COMPLETE	9.3	EmpData	Private	Execute Unauthorized ...

Table 2.4.: Vulnerability Assessment Query 2 – Results

[42] Prefixes identical to Listing 2.5

[43] e.g., `https://w3id.org/sepses/resource/cve/CVE-2016-1646`

2.2.5.2. Intrusion Detection

In this scenario, we illustrate how alerts from the Network Intrusion Detection System (NIDS) Snort[44] can be connected to the SEPSES Cybersecurity KGin order to obtain a deeper understanding of potential threats and ongoing attacks. As a first step, we acquired the Snort community rule set[45] and integrated it into our cybersecurity repository using a defined vocabulary[46]. Snort can monitor these rules and trigger alerts once it finds matches to these patterns in the network traffic. We represent SnortRules as a class with two linked concepts SnortRuleHeader and SnortRuleOption. For SnortRuleOption we include properties such as `sr:hasClassType` and `sr:hasCVEReference`, which will be used to link incoming alerts to CVEs.

```
[**] [1:1807:12] WEB-MISC Chunked-Encoding transfer attempt [**]
[Classification: Web Application Attack] [Priority: 1]
11/10-11:10:12.321349 10.2.189.248:54208 -> 154.241.88.201:80
TCP TTL:61 TOS:0x0 ID:36462 IpLen:20 DgmLen:1200 DF
***A**** Seq: 0xCFAD1EE0 Ack: 0xB27D1032 Win: 0xB7 TcpLen: 32
TCP Options (3) => NOP NOP TS: 2592976 143157138
```

Listing 2.7: IDS Alert Example from MACCDC

Setting: We use a large data set collected during the MACCDC 2012[47] cybersecurity competition as a realistic set of real-world intrusion detection alerts (cf. Listing 2.7 for an example). We provide and use a Snort alert log vocabulary[48] to map those alerts into RDF.

Query: When a Snort alert is triggered, a security expert typically has to analyze its relevance and decide about potential mitigations. False positives are common in this context. For instance, a particular attack pattern may be detected frequently in a network, but it may not be relevant if the targeted host configuration is not vulnerable. To support security analysts in this time-critical and information-intensive analysis task, we identify the corresponding Snort rule that triggered each particular alert. These rules often include a reference to a CVE, which we can use to query our knowledge graph for detailed CVE information related to an alert. Furthermore, by matching the installed software on the host to the vulnerable product configuration defined in CVE (cf. Scenario 1), we can automatically provide security decision makers a better foundation to estimate the relevance of a Snort alert wrt. to their protected assets. To illustrate this process, Listing 2.8 shows an example query to obtain CVE Ids and vulnerable products from Snort alerts. Based on the result Table 2.5, a security analyst can query if the attacked host has the vulnerable software installed (similar to Listing 2.5).

[44] `https://www.snort.org`
[45] `https://www.snort.org/downloads`
[46] `https://w3id.org/sepses/vocab/rule/snort`
[47] `https://maccdc.org/2012-agenda/`, source: `https://www.secrepo.com`
[48] `https://w3id.org/sepses/vocab/log/snort-alert`
[49] Prefixes from Listing 2.5 are reused.

```
PREFIX cve:          <http://w3id.org/sepses/vocab/ref/cve#>
PREFIX cpe:          <http://w3id.org/sepses/vocab/ref/cpe#>
PREFIX snort:        <http://w3id.org/sepses/vocab/ref/snort#>
PREFIX snort-rule:   <http://w3id.org/sepses/vocab/rule/snort#>
PREFIX snort-alert:  <http://w3id.org/sepses/vocab/log/snort-alert#>

SELECT DISTINCT ?alert ?message ?sid ?sourceIp ?destinationIp ?cveId ?cpeId
FROM <http://localhost:8890/snortalert>
WHERE {
    ?alert a snort-alert:IDSSnortAlertLogEntry ;
           snort:signatureId ?sid ;
           snort:message ?message ;
           snort:sourceIp ?sourceIp ;
           snort:destinationIp ?destinationIp .

    SERVICE <http://w3id.org/sepses/sparql> {
        ?rule a snort-rule:SnortRule ;
              snort-rule:hasRuleOption ?ruleOption .
        ?ruleOption snort:signatureId ?sid ;
                    snort-rule:hasCveReference ?cve .
        ?cve cve:id ?cveId ;
             cve:hasCPE/cpe:id ?cpeId
    }
}
```

Listing 2.8: Intrusion Detection query[49]

alert	message	sid	sourceIP	targetIP	cveId	cpeId
Alert001	WEB-MISC Chunked...	1807	10.2.190.254	154.241.88.201	2002-0392	cpe:/a:apa...
Alert002	WEB-MISC WebDAV...	1070	10.2.190.254	154.241.88.201	2000-0951	cpe:/a:micr...
Alert003	WEB-MISC TRACE...	2056	10.2.197.241	154.241.88.201	2004-2320	cpe:/a:bea:w...
Alert004	WEB-FRONTPAGE...	1248	10.2.190.254	154.241.88.201	2001-0341	cpe:/o:micr...
Alert005	WEB-MISC Netscape...	1048	10.2.197.241	154.241.88.201	2001-0250	cpe:/a:netsc...

Table 2.5.: Intrusion Detection Query Results

2.2.6. Conclusion

In this resource paper, we highlight the need for semantically explicit representations of security knowledge and the current lack of interlinked instance data. To tackle this challenge, we present a cybersecurity knowledge graph that integrates a set of widely adopted, heterogeneous cybersecurity data sources.

To maintain the knowledge graph and integrate newly available information, we developed an ETL process that updates it as new security information becomes available. In order to make this resource publicly available and easy to use, we offer multiple services to access the data, including a SPARQL endpoint, a triple pattern fragments interface, a Linked Data interface, and download options for the complete data set.

We demonstrated the usefulness of the graph by means of two example use cases in vulnerability assessment and semantic interpretation of alerts generated by intrusion detection systems. Given the compelling need for efficient exchange of machine-interpretable cybersecurity knowledge, we expect the KG to be useful for practitioners and researchers, and hope that the resource will ultimately facilitate novel and innovative semantic security tools and services. Future work will focus on disseminating the resource in the security domain, building a community of users and contributors around it, and growing the knowledge graph by integrating additional security standards and information extracted from structured and unstructured sources.

3. Semantic Log Integration, Monitoring and Analysis

In this chapter, we present our contributions that address our second research question (RQ2), i.e. *"How can dispersed log data and cybersecurity information be integrated and interlinked?"*. We answer this research question by two approaches structured in two sections as follows:

- ***Cross-Platform Semantic Log Monitoring and Forensics***. This section discusses an approach for semantic log monitoring and forensics across heterogeneous platforms. We design and implement an architecture that includes semantic log acquisition, extraction, linking & storage. In particular, we extend our previously developed log vocabularies to not only model low-level log events but also represent high-level events (i.e., File Access Event). We leverage RDF Stream Processing (RSP) engines to execute semantic continuous queries to track file system activity across hosts. Furthermore, using semantic query federation, we can not only integrate logs, but can also link and contextualize them to the external background knowledge, e.g., vulnerability information, attack patterns, etc.

- ***Virtual Knowledge Graph for Distributed Security Log Analysis***. This section discusses a distributed approach for log analysis based on Virtual Knowledge Graph (VKG). Specifically, we introduce a novel concept and architecture to dynamically extract heterogeneous raw log sources into log knowledge graphs directly from the distributed monitoring hosts at query time, i.e., without a priori log parsing, aggregation, processing, and materialization. We leverage semantic query federation that can not only integrate log graphs from dispersed sources but also link and contextualize them to internal/external background knowledge (e.g., asset information, cybersecurity information, etc.) on demand.

3.1. Cross-Platform Semantic Log Monitoring and Forensics

Kabul Kurniawan, Elmar Kiesling, Andreas Ekelhart, Fajar Juang Ekaputra.
Published as *"Cross-Platform File System Activity Monitoring and Forensics – A Semantic Approach"* in ICT Systems Security and Privacy Protection. SEC 2020 [30].

Abstract

Ensuring data confidentiality and integrity are key concerns for information security professionals, who typically have to obtain and integrate information from multiple sources to detect unauthorized data modifications and transmissions. The instrumentation that operating systems provide for the monitoring of file system level activity can yield important clues on possible data tampering and exfiltration activity but the raw data that these tools provide is difficult to interpret, contextualize and query. In this paper, we propose and implement an architecture for file system activity log acquisition, extraction, linking and storage that leverages semantic techniques to tackle limitations of existing monitoring approaches in terms of integration, contextualization, and cross-platform interoperability. We illustrate the applicability of the proposed approach in both forensic and monitoring scenarios and conduct a performance evaluation in a virtual setting.

3.1.1. Introduction

In our increasingly digitized world, Information and Communication Technologies pervade all areas of modern life. Consequently, organizations face difficult challenges in protecting the confidentiality and integrity of the data they control, and theft of corporate information – i.e., data breaches or data leakage – have become a critical concern [82].

In the face of increasingly comprehensive collection of sensitive data, such incidents can become an existential threat that severely impacts the affected organization, e.g., in terms of reputation loss, decreased trustworthiness, and direct consequence that affect their bottom line. Fines and legal fees, either due to contractual obligations or laws and regulations (e.g., the General Data Protection Regulation in the EU), have become another critical risk. Overall, the number and size of data breaches have been on the rise in recent years years[1].

On a technical level, exfiltration of sensitive data is often difficult to detect. In this context, we distinguish two main types of adversaries and associated threat models: *(i)* an insider with legitimate access to data, who either purposely or accidentally exfiltrates data, and *(ii)* an external attacker who obtains access illegitimately. Insiders typically have multiple channels for exfiltration at their disposal, including conventional protocols (e.g., ftp, sftp, ssh, scp), cloud storage services (e.g., dropbox, onedrive, google drive, WeTransfer), physical media (e.g., USB, laptop, mobile phone), messaging and email applications, and dns tunneling [83]. Whereas an insider may leverage legitimate access permissions directly or at least internal resources as a starting point, an external attacker must first infiltrate the organization network and obtain access to the data (e.g., by spreading malware or spyware, stealing credentials, eavesdropping, brute forcing employee passwords, etc.).

[1] `https://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/`

State-of-the-art perimeter security solutions such as intrusion detection and prevention systems (IDS/IPS), firewalls, and network traffic anomaly detection are per se generally not capable of detecting insider attacks [84]. However, such activities typically leave traces in the network and on the involved systems, which can be used to spot potential misuse in real time or to reconstruct and document the sequence of events associated with an exfiltration and its scope ex-post. This examination, interpretation, and reconstruction of trace evidence in the computing environment is part of digital forensics. Upon detection of security violations, forensic analysts attempt to investigate the relevant causes and effects, frequently following the hypothesis-based approach to digital forensics [85]. Although there are a variety of tools and techniques available that are employed during a digital investigation, the lack of integration and interoperability between them, as well as the formats of their sources and resulting data hinder the analysis process [86].

In this paper, we introduce a novel approach that leverages semantic web technologies to address these challenges in the context of file system activity analysis. This approach can harmonize heterogeneous file and process information across operating systems and log sources. Furthermore, it provides contextualization through interlinking with relevant information and background knowledge.

The research question we address in this article is: *How can semantic technologies support digital file activity investigations?* Addressing this question resulted in the following main contributions: *(i)* a set of log and file event vocabularies (Section 3.1.3); *(ii)* an architecture and prototypical implementation for file system log acquisition, event extraction, and interlinking across heterogeneous systems and with background knowledge (Section 3.1.4); *(iii)* a set of demonstration scenarios for continuous monitoring and forensic investigations (Section 3.1.5); and *(iv)* a performance evaluation in a virtual setting (Section 3.1.6).

3.1.2. Related Work

Our approach builds upon and integrates multiple strands of work, which we will review in the following: *(i)* approaches for file activity monitoring, both in the academic literature and commercial tools; *(ii)* file system ontologies; and *(iii)* semantic file monitoring & forensics.

File Activity Monitoring In contrast to the approach presented in this paper, prior work in this category does not involve semantic or graph-based modeling, which facilitates interoperability and integration, contextualization through interlinking with background knowledge, and reasoning.

The authors in [87] focus on data exfiltration by insiders. They first apply statistical analyses to characterize legitimate file access patterns and compare those to file access patterns of recent activities to identify anomalies. The authors mention that the approach can result in a high number of suspicious activities, which can be impractical for individual investigation. [88] aims to predict insider threats by monitoring various parameters such as file access activity, USB storage activity, application usage, and sessions. In their evaluation, they train a deep learning model on legitimate user activity and then use the model to assign threat scores to unseen activities. In [89], the authors introduce a policy-based system for data leakage detection that utilizes operating system call provenance. They facilitate real-time detection of data leakage by tracking operations performed on sensitive files. This approach is similar to the one presented in this paper in its objectives,

i.e., it also aims to monitor file activities (copy, rename, move), but it does not cover contextualization and linking to background knowledge. [90] proposes an approach that leverages data provenance information from OS kernel messages to detect exfiltration of data returned to users from a database. The proposed system builds profiles of users' actions to determine whether actions are consistent with the tasks of the users. While it has similar goals, the focus is limited on data exfiltration from databases via files.

Apart from the academic research on various techniques for file activity monitoring, a wide range of tools is available commercially, such as Solarwind Server and Application Monitor, ManageEngine DataSecurity Plus, PA File Insight, STEALTHbits File Activity Monitor, and Decision File Audit. These tools cover varying scopes of leakage detection and typically provide a simple alerting mechanism upon suspicious activity. Another category of existing tools are Security Information and Event Management systems (e.g., LogDNA, Splunk, ElasticSearch). Their purpose is to manage and analyze logs and they do not specifically tackle the problem of tracking file activity life-cycles.

File System Ontologies Ontological representation of file system information has been explored, e.g., in [91], in which the authors propose TripFS, a lightweight framework that applies Linked Data principles for file systems in order to expose their content via dereferenceable HTTP URIs. The authors model file systems with their published vocabulary that is aligned with the NEPOMUK File Ontology (NFO)[2]. Similar to TripFS, [92] proposes VDB-FilePub to expose file systems as Linked Data and to publish user-defined content metadata. With focus on end-user access, [93] provide an extension to TripFS which enables users to navigate the published files, and to annotate and download them via common web browsers without the need to install special software packages.

In recent work, the authors of [94] proposed a Semantic File System (SFS) Ontology[3] which extends terms from the NEPOMUK ontology. They further provide technical definitions of terms and a class hierarchy with persistent URIs and content negotiation capabilities. In our approach, we use the basic concepts for files, such as file names and file properties as proposed in the related work, but our approach integrates additional concepts, such as, e.g., file activities, source and target locations, and file classification.

Semantic Approaches to File Access Monitoring & Forensics The application of semantics for digital forensics has been the topic of multiple research publications. While they are motivated by similar challenges, such as heterogeneity, variety and volume of data, they do not focus on file activity monitoring and life-cycle construction in particular, but on the digital evidence process in general.

Early work on using semantic web technology in the context of forensics includes [95], which introduces an evidence management methodology to semantically encode why evidence is considered important. An ontology is used to describe the metadata file contents and events in a uniform and application-independent manner. In [96], the authors propose a similar ontology-based framework to assist investigators in analyzing digital evidence. They motivate the use of semantic technologies in general and discuss the advantage of ontological linking, annotations, and entity extraction. A broader architecture to lift the phases of a digital forensic investigations to a knowledge-driven setting is proposed in [86].

[2] `http://oscaf.sourceforge.net/nfo.html`
[3] `https://w3id.org/sfs-ontology#`

This results in an integrated platform for forensic investigation that deals with a variety of unstructured information (e.g., network traffic, firewall logs, and files) and builds a knowledge base that can be consulted to gain insights from previous cases via SPARQL queries.

Finally, in a recent contribution [97], the authors propose a framework that supports forensic investigators during the analysis process. This framework extracts and models individual pieces of evidence, integrates and correlates them using a SWRL rule engine, and persists them in a triplestore. Compared to our approach, their focus is on text processing while file activity analysis is not considered.

The approach presented in this paper extends preliminary work published in [29] by introducing cross-platform interoperability, scenarios that demonstrate the approach, linking to background knowledge and a performance evaluation.

3.1.3. Conceptualization

Operating systems typically provide mechanisms and instrumentation to obtain information on system-level file system operations, typically on the level of kernel calls. Reconstructing the corresponding user activities, such as editing, moving, copying or deleting a file from these low-level signals can be challenging. In particular, the sequence of micro-operations triggered by a file system operation varies across operating systems and applications, which complicates the analysis. On Windows systems, for instance, file operations such as `Create` generate a number of access operations including `ReadAttributes`, `WriteData`, `ObjectClosed`, etc.

To construct our vocabularies, we analyzed the structure, format, and access patterns of the different file activity log sources on both Windows and Linux. Furthermore, as contextualization is a key requirement for the interpretation of file activity in forensic analyses, we also include sources of *(i)* process activity information, and *(ii)* authentication events (login, logout, etc.) . The scenarios in Section 3.1.5 illustrate how we make use of process information and authentication information. Due to space restrictions, we will not cover the process and authentication vocabulary in full detail and refer the interested reader to the source[4].

3.1.3.1. Vocabulary

As existing ontologies (reviewed in Section 3.1.2) do not fully cover the requirements of our approach, we developed a custom ontology. We followed a bottom-up approach starting from low-level information from log sources with the goal to choose and collect appropriate terms directly from the sources of evidence (e.g., users, hosts, files). We organize our semantic model into two levels, i.e., *log entry* level and *file operation* level. On the *log entry* level, we define a vocabulary to represent information on micro-level operations for both Windows and Linux OS log sources which is based on a previously developed vocabulary [27] for generic log data. On the *file operation* level, we model a generic vocabulary to express higher-level events such as actual file event activity (e.g., created, modified, copied, rename, delete) derived from micro-level operations.

[4] `https://w3id.org/sepses/vocab/event/process-event`

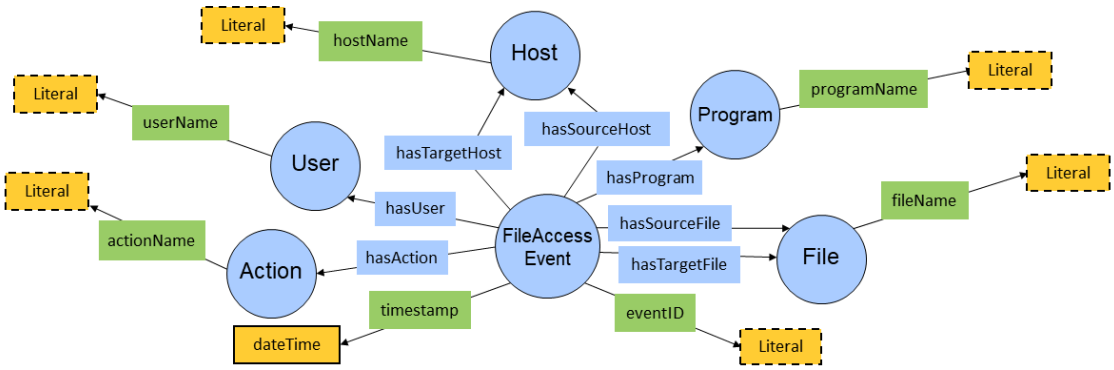

Figure 3.1.: High-Level event vocabularies (File Access Event)

Log Entry vocabularies The Windows Log Event (wle) vocabulary[5] represents Windows file access events using `wle:WindowsEventLogEntry`, a subclass of `cl:LogEntry` from the SEPSES core log[6]. The `wle:Subject` class represents account information such as `wle:accountName` and `wle:logonID`; the `wle:AccessRequest` class represents file access information such as `wle:accessMask` and `wle:accesses`; the `wle:Process` class represents running processes and the `wle:Object` class represents object file information such as `wle:objectName`, `wle:objectType`, and `wle:handleID`. To cover Linux file access events, we developed the Linux Log Event (lle)[7] vocabulary that comprises five main classes: `lle:LinuxEventLogEntry`, a subclass of `cl:LogEntry` from the SEPSES core vocabulary, `lle:Event` class, which covers information on file access events such as `lle:eventType`, `lle:eventId`, `lle:eventCategory`, and `lle:eventAction`; the `lle:File` class represents information about file objects such as `lle:fileName` and `lle:filePath`; the `lle:User` class covers information on users who perform the file event activities such as `lle:userName` and `lle:userGroup`; the `lle:Host` class represents `lle:hostArchitecture`, `lle:hostOS`, `lle:hostName`, `lle:hostId`, etc.

The **File Operation vocabulary**[8] describes `fae:FileAccessEvents` by means of the following properties: `fae:hasAction` reflects the type of access (e.g., created, modified, copied, renamed, deleted); `fae:hasUser` links the file event to the user accessing the file; `fae:hasProgram` represents the executable used to access the file, and `fae:timestamp` captures the time of access. The properties `fae:hasSourceFile` and `fae:hasTargetFile` model the relation between an original and copied instance of a file. Finally, property `fae:hasSourceHost` and `fae:hasTargetHost` represent the hosts where the source and target files are located.

3.1.3.2. Background Knowledge

To support contextualization and enrichment, we leverage several existing sources of internal and external background knowledge.

Internal background knowledge can be developed by manually or automatically collecting an organization's persistent information (e.g., IT Assets, Network Infrastructure, Users).

[5] `https://w3id.org/sepses/vocab/log/win-event`
[6] `https://w3id.org/sepses/vocab/log/core`
[7] `https://w3id.org/sepses/vocab/log/linux-event`
[8] `https://w3id.org/sepses/vocab/event/file-access`

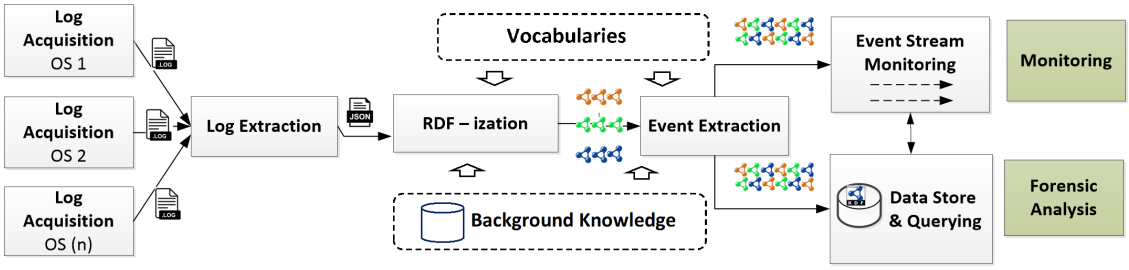

Figure 3.2.: Solution architecture

In our scenarios, we use predefined internal background knowledge to contextualize and create linking with file access events during event extraction.

Furthermore, it is possible to leverage existing external knowledge, such as the SEPSES cybersecurity knowledge graph (CSKG)[9], to link external information with system events.

3.1.4. Architecture & Prototype Implementation

In this subsection, we describe our architecture and prototypical implementation for semantic integration, monitoring, and analysis of file system activity as depicted in Figure 3.2.

The ***Log Acquisition*** component deals with the acquisition of log information and is installed as an agent on clients or servers. We implement our Log Acquisition component on *Filebeat*[10], an open-source log data acquisition tool that ships log data from a host for further processing. Using Filebeat, we can easily select and configure and add log sources from both Windows and Linux machines. Furthermore, we use the Filebeat Audit module to ship process and authentication information from the log sources.

The ***Log Extraction*** component handles the parsing of various log data provided by the Log Acquisition component and can act as a filter that keeps only relevant parts. We use *Logstash*[11], an open source log processing tool that provides options for developing processing pipelines to distinguish and handle different types of log sources. Furthermore, it provides different output options such as a web socket protocol that supports data streaming.

The ***RDF-ization*** component transforms data into RDF by mapping structured log data produced by the Log Extraction component to a set of predefined ontologies (cf. Section 3.1.3). This produces an RDF graph as the basis of file operation events extraction. We use TripleWave[12] to publish RDF streaming data through specified mappings (e.g., RML[13]). Furthermore, TripleWave supports the web socket protocol to publish the output.

The ***Event Extraction*** component generates file operation events by identifying a sequence of low level (e.g., kernel-level) file system events. Furthermore, it enriches the events by creating links between file operation events and existing internal (hosts, users, etc.) and external (e.g., the SEPSES cybersecurity knowledge graph [28]) background

[9] `http://sepses.ifs.tuwien.ac.at`
[10] `https://www.elastic.co/products/beats/filebeat`
[11] `https://www.elastic.co/products/logstash`
[12] `https://streamreasoning.github.io/TripleWave/`
[13] `http://rml.io`

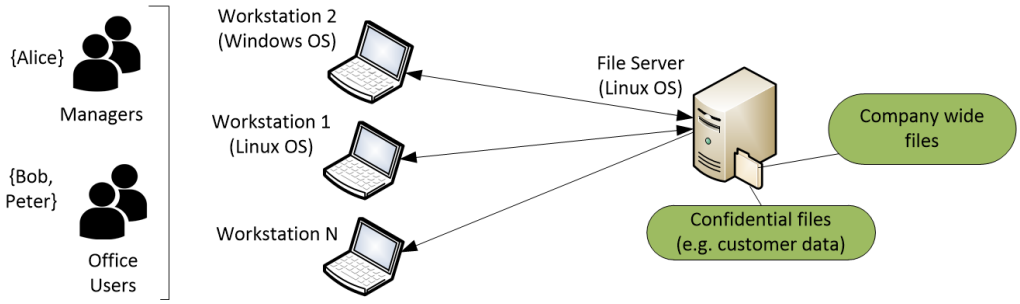

Figure 3.3.: Scenario 1 network excerpt

knowledge. We developed a Java-based event extractor[14] and use the C-Sprite[98] engine to implement the event extraction process. C-Sprite is an RDF stream processing engine that allows us to register a set of continuous SPARQL-Construct queries against the low level RDF graph of file system events to generate a graph of file operation events.

Finally, the ***Data Storage, Querying, and Visualization*** component stores the extracted RDF graph of file operation events in a persistent storage (e.g., a triplestore) and facilitates querying and further analysis. We choose the widely-used Virtuoso[15] triple store, which provides a SPARQL endpoint, for our prototypical implementation. Furthermore, we developed a simple web-based graph visualization interface[16] that helps analysts to interpret file access life-cycles (cf. Section 3.1.5 for an example).

3.1.5. Application Scenarios

In this subsection, we demonstrate the feasibility of our approach by means of two application scenarios. For both scenarios, we set up a virtual lab with several Windows and Linux machines, users, groups, and shared folders.

3.1.5.1. Scenario 1: Data Exfiltration

In the first scenario, we assume that an organization has learned that confidential information was leaked. The task in this scenario is to investigate how and by whom this information has been transferred out of the organizational network.

Figure 3.3 depicts an excerpt of the company network, including Linux and Windows workstations and a Linux file server that stores company-wide shared data as well as confidential data with restricted access permissions (e.g., customer and financial data). The organization's access model distinguishes two groups: *manager* and *office users*. Both groups are authorized to log in to the company workstations and access the internal file shares. Access to the confidential data is restricted to the *manager* group.

As a starting point, the analyst has the name of a file that contains the leaked sensitive information and starts to investigate its history. Listing 3.1 depicts the SPARQL query to obtain lifecycle information for this file. The result is given in Table 3.1 and shows that the file *cstcp001.xls* was accessed and modified multiple times. Inspecting the timeline, we can see that a file *customer.xls* was modified on *FileServer1* with the IP *193.168.1.2*. It

[14] `https://github.com/kabulkurniawan/fileAccessExtractor`
[15] `https://virtuoso.openlinksw.com/`
[16] `https://w3id.org/sepses/sparqlplus`

```
SELECT distinct  ?time ?accessType ?sourceFile ?targetFile  ?hostIP ?hostType
    WHERE {
            ?y fae:timestamp ?timestamp.
            ?y   fae:hasAction/fae:actionName ?accessType.
            ?y fae:hasSourceFile/fae:pathName ?sourceFile.
            ?y fae:hasTargetFile/fae:pathName ?targetFile.
            ?y fae:hasTargetHost ?h.
            ?h cl:IpAddress ?hostIp.?h fae:hasSourceHost ?hostName.
            ?y fae:hasSourceFile/fae:fileName "cstcp001.xls".
            ?x fae:relatedTo* ?y .
        } ORDER BY ASC(?time)
```

Listing 3.1: SPARQL query to retrieve the history of a file

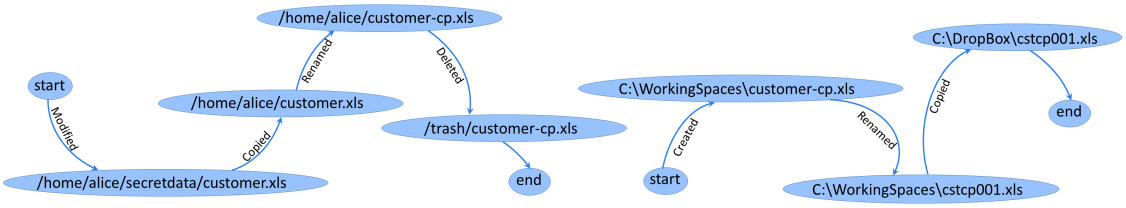

Figure 3.4.: Graph visualization of the file history

thereafter was copied, renamed and modified on the file server. Then, the file appeared on *Workstation2* and got deleted from the file server. Finally, the file was renamed to *cstcp001.xls* and copied to another folder on *Workstation2* with the name *Dropbox* in its file path. Figure 3.4 visualizes the file history.

timestamp	accessType	sourceFile	targetFile	hostIP	hostName
11:06:55	Modified	/home/alc/secdt/customer.xls	/home/alc/secdt/customer.xls	193.168.1.2	FileServer1
13:39:01	Copied	/home/alc/secdt/customer.xls	/home/alc/customer.xls	193.168.1.2	FileServer1
13:39:35	Renamed	/home/alc/customer.xls	/home/alc/customer-cp.xls	193.168.1.2	FileServer1
13:40:23	Modified	/home/alc/customer-cp.xls	/home/alc/customer-cp.xls	193.168.1.2	FileServer1
13:43:17	Created	C:\Work\customer-cp.xls	C:\Work\customer-cp.xls	193.168.2.2	Workstation2
13:43:52	Deleted	/home/alc/customer-cp.xls	/.trash/customer-cp.xls	193.168.1.2	FileServer1
15:50:57	Renamed	C:\Work\customer-cp.xls	C:\Work\cstcp001.xls	193.168.2.2	Workstation2
15:53:52	Copied	C:\Work\cstcp001.xls	C:\DropBox\cstcp001.xls	193.168.2.2	Workstation2

Table 3.1.: File History Results

Next the analyst wants to know how the file was transferred from *FileServer1* to *Workstation2*. A SPARQL query[17] lists the running processes and user names in the time period of the suspicious activities. Potential exfiltration processes are modeled in the background knowledge with the concept *sys:potentialExfitrationProcesses*, which includes channels such as FTP, SCP, SSH, etc. This illustrates how queries can automatically make use of modeled background knowledge. Table 3.2 shows the results of the query. From this, the analyst learns that a secure copy event */usr/bin/scp* was started on *FileServer1* prior to the file copy and also on the Windows host *Workstation2*. The processes on the file server were performed by user *Alice* from the manager group. The analyst concludes that the *customer-cp.xls* file was successfully transferred via SCP (SSH service) by the user *Alice*.

Next, the analyst wants to collect more information about this file transfer and the

[17] `https://w3id.org/sepses/IFIP2020/queries/potentialExfitrationProcesses.sparql`

timestamp	eventType	hostIP	hostName	programName	pid	userName	groupName
13:43:17	ProcessStopped	193.168.1.2	FileServer1	/usr/bin/scp	223	Alice	Manager
13:43:17	ProcessStopped	193.168.1.2	FileServer1	/usr/bin/ssh	224	Alice	Manager
13:43:17	ProcessStarted	193.168.2.2	Workstation2	C:\...\sshd.exe	1988	-	-
13:43:18	ProcessStopped	193.168.2.2	Workstation2	C:\...\sshd.exe	1988	-	-

Table 3.2.: Potential Exfiltration Process – Results

```
SELECT * WHERE {
    ?s rdf:type fae:FileAccessEvent;
        fae:hasFileAccessType sys:Created;
        fae:hasSourceFile/fae:fileName ?filename;
        asset:hasDataClassification  sys:Private;
        fae:hasSourceHost/fae:hostName ?hostName;
        {SELECT ?hostName ?OSName ?hostIP ?cveId ?conf ?score WHERE {
                ?t rdf:type sys:Host. ?t sys:hostName ?hostName.
                ?t sys:OSName ?OSName. ?t sys:IPAddress ?hostIP.
                ?t sys:hasProduct ?p.
            SERVICE <http://sepses.ifs.tuwien.ac.at/sparql> {
                ?cve cve:hasCPE ?p. ?cve cve:id ?cveId.
                ?cve cve:hasCVSS2BaseMetric ?cvss2. ?cvss2 cvss:
    confidentialityImpact ?conf.
                ?cvss2 cvss:baseScore ?cvssScore. }}}}
```

Listing 3.2: Query to check vulnerable host

users involved in those steps. Therefore, a LoginProcess[18] query is executed to retrieve a list of users logged in to these hosts in the time period of interest, including `userName`, `sourceIp`, `targetIp`, `hostName`, and the `timestamp`.

timestamp	eventType	sourceHost	sourceIp	targetHost	targetIp	userName
13:30:23	Login	-	172.24.66.19	Workstation1	192.168.2.1	Bob
13:33:31	Login	-	172.24.66.19	Workstation1	192.168.2.1	Bob
13:38:16	Login	Workstation1	192.168.2.1	FileServer1	192.168.1.2	Alice
14:53:06	Login	-	172.24.66.19	Workstation2	192.168.2.2	Bob

Table 3.3.: Login process results

The query result depicted in Table 3.3 shows that *Alice* was not logged in to *Workstation1* during this time. Instead, *Bob* shows up several times in the login list of Workstation1. From *Workstation1*, a login event was performed on *FileServer1* with Alice's credentials. At the time the file copy to the Dropbox folder happened on *Workstation2*, only Bob was logged in on this computer. Concluding from this evidence, the analysts suspects that Bob logged in to *Workstation1*, then accessed the confidential file on *FileServer1* with the credentials of Alice. Finally, he copied the file to *Workstation2* and exfiltrated the data via Dropbox.

3.1.5.2. Scenario 2: Sensitive data on vulnerable hosts

In the second scenario, we illustrate how the semantic monitoring approach can be used to protect confidential information by combining public vulnerability information with file activity information from inside the company network. We assume a policy that restricts handling of confidential files on hosts with known vulnerabilities. The objective in this scenario is to automatically detect violations of this policy. More precisely, the goal is to spot whenever files flagged as *confidential*[19] are copied or created on an internal host with

[18] `https://w3id.org/sepses/IFIP2020/queries/loginProcess.sparql`
[19] using a classification schema of confidential, private, protected, public

fileName	hostName	OSName	hostIP	cveId	conf	score
C:\Documents\Customer.xls	Workstation2	Windows	192.168.2.1	2016-1653	COMPLETE	9.3
/home/docs/employee.xls	Workstation3	Linux	192.168.2.1	2016-1583	COMPLETE	7.2

Table 3.4.: Vulnerability assessment results excerpt

a known vulnerability.

As background knowledge, we import information on installed software on each host. This information is represented in the Common Platform Enumeration (CPE) format and can be collected automatically by means of software inventory tools. To link this information to known vulnerabilities, we rely on Common Vulnerabilities and Exposures (CVE), a well-established enumeration of publicly known cybersecurity vulnerabilities. We take advantage of our recent work on transforming this structured knowledge into a knowledge graph [28] available via various semantic endpoints. This allows us to directly integrate this information and use it in our scenario.

To implement the monitoring in this scenario, we set up a federated continuous SPARQL query at listing 3.2 to identify whether a sensitive file shows up on a vulnerable workstation. To restrict the query to confidential files, we use the property `asset:hasDataClassification` and restrict our query to `sys:Private` files. Table 3.4 shows the query results and reveals that *Workstation2* and *Workstation3* have critical vulnerabilities, but store confidential files. The results include the `fileName`, `hostName`, `hostIP`, `cveId`, etc. As a next step, an analyst can inspect the life-cycle of the files to understand where they came from, who accessed them and explore information on the vulnerabilities and potential mitigations. Taking automated actions based on the results, such as blocking the access or alerting the user, is a further option.

3.1.6. Evaluation

In this subsection, we present our empirical evaluation setup and discuss the results.

3.1.6.1. Experimental Setup

We ran the experiments on an Intel Core i7 processor with 2,70GHz, 16GB RAM, and 64-bit Microsoft Windows 10 Professional and emulate hosts as docker containers. We used C-Sprite as event extraction engine with a 3 seconds time window that slides every second. In order to simulate user activity, we developed a java-based event generator[20] to generate scripts for random file activities and use weighted random choices to select activities.

3.1.6.2. Experiments and Results

To measure the correctness and the completeness of the event extraction and detection using RDF stream processing with C-Sprite, we define a set of metrics, including *(i)* Actual Events (AE) – number of the events executed in the simulation (ground truth), and *(ii)* Returned Events (RE) – number of events correctly detected by the RDF-Stream processing (C-Sprite). We get detection (%D) by dividing RE by AE.

$$Detection(\%D) = \frac{ReturnedEvents(RE)}{ActualEventsGenerated(AE)} * 100\%$$

[20] `https://github.com/sepses/fileAccessExtractor/tree/master/eventGenerator`

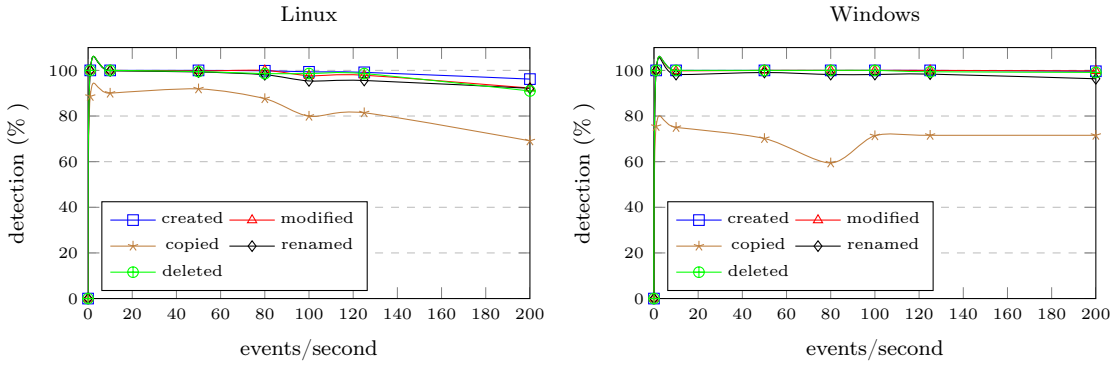

Figure 3.5.: Detection rate on Linux (l) and Windows (r)

On each target OS (Linux and Windows), we test a varying number of events per second, i.e. 1, 10, 20, 50, 80, 100, 125 and 200 events/sec. In the results, we report the mean of detected events over 5 runs with 480 simulated events each.

As shown for Linux in Figure 3.5, all events can be detected close to 100% for all frequencies (1 event/sec up to 200 events/sec) except the copy event, which reached a maximum of 91,89%. At 200 events/sec, we observe that the detection of copy events decreases to approx. 70%, which is mainly caused by incorrect pairings of *readAttribute* and *create* events when these micro operations generated by two or more sequential copy events appear together in the same window. Furthermore, we noticed that low-level events sometimes do not arrive in sequence and hence, are not detected by our queries.

For Windows, the event detection performance for *created*, *modified*, *renamed* and *deleted* events is higher with almost 100% of detected events for all frequencies. However, the copy event detection in Windows achieves a lower detection with a maximum of 75,46%.

Finally, considering scalability we can make an estimation based on [98], which shows that C-Sprite achieves a throughput of more than 300000 triples/s. Consequently, it should be able to handle up to 23000 events/s (an individual event consists of at least 13 triples). For forensic scenarios, the Virtuoso triple store can load more than 500 million triples per 16GB RAM[21], which means that it should be possible to handle more than 38 million events per 16GB RAM.

3.1.7. Conclusion

In this paper, we tackled current challenges in file activity monitoring and analysis, such as the lack of interoperability, contextualization and uniform querying capability, by means of an architecture based on Semantic Web technologies. We introduced a set of vocabularies to model and harmonize heterogeneous file activity log sources and implemented a prototype. We illustrate how this prototype can monitor file system activities, trace file life cycles, and enrich them with information to understand their context (e.g., internal and external background knowledge). The integrated data can then be queried, visualized, and dynamically explored by security analysts, as well as be used to facilitate detection and alerting by utilizing stream processing engines.

Finally, we demonstrate the applicability of the approach in two scenarios in virtual

[21] `http://docs.openlinksw.com/virtuoso/virtuosofaq11/`

environments – one focused on data exfiltration forensics, and another on monitoring policy violations integrating public vulnerability information. The results of our evaluation indicate that the approach can effectively extract and link micro-level operations of multiple operating systems and consolidate them in an integrated stream of semantically explicit file activities.

Overall, the results are promising and demonstrate how semantic technologies can enrich digital investigations and security monitoring processes. In future work, we aim to address the accuracy and scalability limitations of the current approach identified in the streaming evaluation, e.g., by evaluating alternative streaming engines and alternative approaches (e.g., complex event processing) based on big data technologies. Furthermore, we will investigate the integration of our approach into existing standards (e.g., STIX and CASE) to increase interoperability for forensic investigation.

3.2. Virtual Knowledge Graph for Distributed Security Log Analysis

Kabul Kurniawan, Andreas Ekelhart, Elmar Kiesling, Gerald Quirchmayr, A Min Tjoa. Published as *"VloGraph: A Virtual Knowledge Graph Framework for Distributed Security Log Analysis". In Machine Learning and Knowledge Extraction*, 2022 [33].

Abstract

The integration of heterogeneous and weakly linked log data poses a major challenge in many log-analytic applications. Knowledge graphs (KGs) can facilitate such integration by providing a versatile representation that can interlink objects of interest and enrich log events with background knowledge. Furthermore, graph-pattern based query languages such as SPARQL can support rich log analyses by leveraging semantic relationships between objects in heterogeneous log streams. Constructing, materializing, and maintaining centralized log knowledge graphs, however, poses significant challenges. To tackle this issue, we propose VloGraph – a distributed and virtualized alternative to centralized log knowledge graph construction. The proposed approach does not involve any a priori parsing, aggregation, and processing of log data, but dynamically constructs a virtual log KG from heterogeneous raw log sources across multiple hosts. To explore the feasibility of this approach, we developed a prototype and demonstrate its applicability in three scenarios. Furthermore, we evaluate the approach in various experimental settings with multiple heterogeneous log sources and machines; the encouraging results from this evaluation suggest that the approach can enable efficient graph-based ad-hoc log analyses in federated settings.

3.2.1. Introduction

This paper is an extension of [32], in which we initially introduced the concept of Virtual Knowledge Graphs for log analysis.

Log data analysis is a crucial task in cybersecurity, e.g., when monitoring and auditing systems, collecting threat intelligence, conducting forensic investigations of incidents, and pro-actively hunting threats [99]. Currently available log analysis solutions, such as *Security Information and Event Management (SIEM)* systems, support the process by aggregating log data as well as storing and indexing log messages in a central relational database [100]. With their strict schemas, however, such databases are limited in their ability to represent links between entities [101]. This results in a lack of explicit links between heterogeneous log entries from dispersed log sources in turn makes it difficult to integrate the partial and isolated views on system states and activities reflected in the various logs. Furthermore, the central log aggregation model is also bandwidth-intensive and computationally demanding [102, 100, 103], which limits its applicability in large-scale infrastructures. Without a dedicated centralized log infrastructure, however, the process necessary to acquire, integrate and query log data is tedious and inefficient, which poses a key challenge for security analysts who often face time critical tasks.

To illustrate the issue, consider the example in Figure 3.6. It is based on log data produced by multi-step attacks[22] as described in [104]. The various steps of the attack are reflected in

[22]These log data sets will also be used in a scenario in Section 3.2.6.

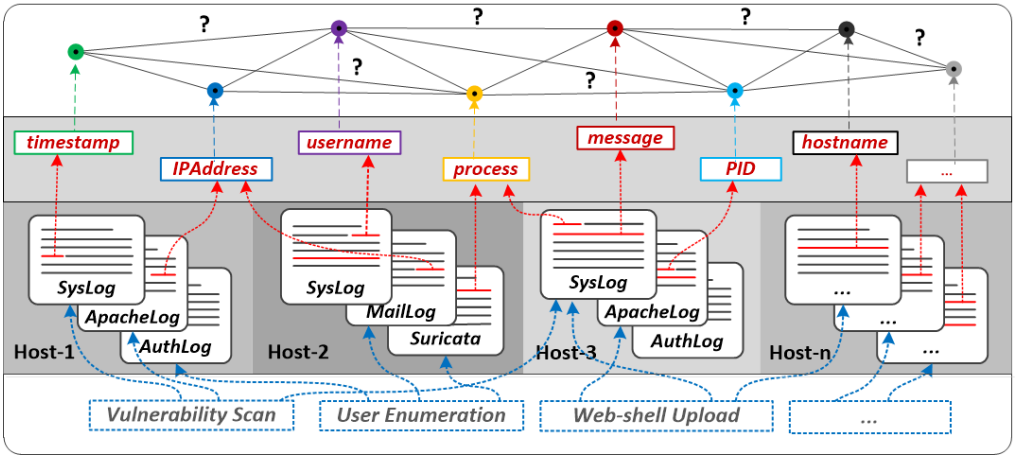

Figure 3.6.: Motivating example illustrating that attack steps leave traces in various log sources across multiple hosts, making it difficult to reconstruct what happened.

a large number of log messages in a diverse set of log sources dispersed across multiple hosts and files (e.g., *Syslog, ApacheLog, AuthLog, MailLog* etc.). *Vulnerability Scan*, for instance – which scans a system for known vulnerabilities – leaves some traces in multiple log sources such as *Syslog* and *ApacheLog* on *Host1* and *Host3*, respectively. *User Enumeration* – an activity that aims to guess or confirm valid users in a system – also leaves some traces in (*AuthLog, MailLog etc.*) stored on *Host1* and *Host2*. As this example shows, a single attack step typically results in a large number of log events that capture comprehensive information. This information can be used for log analysis and attack investigation, but correlating, tracing, and connecting the individual indicators of compromise – e.g., through timestamps, IP addresses, user names, processes etc. – is typically a challenging and often time-consuming task. This is partly due to the weak structure of log sources and their inconsistent format and terminologies. Consequently, it is difficult to get a complete picture of suspicious activities and understand what happened in a given attack – particularly in the face of fast evolving, large volume, and highly scattered log data.

To tackle these challenges, we propose , a decentralized framework to contextualize, link, and query log data. We originally introduced this framework in [32]; in this paper, we extend this prior work with a detailed requirements specification, evaluation with two additional application scenarios, and a section reflecting upon graph-based log integration and analysis, decentralization and virtualization, and discussing applications and limitations.

More specifically, we introduce a method to execute federated, graph pattern-based queries over dispersed, heterogeneous raw log data by dynamically constructing virtual knowledge graphs [105, 22]. This knowledge-based approach is designed to be decentralized, flexible and scalable. To this end, it *(i)* federates graph-pattern based queries across endpoints, *(ii)* extracts only potentially relevant log messages, *(iii)* integrates the dispersed log events into a common graph, and *(iv)* links them to background knowledge. All of these steps are executed at query time without any up-front ingestion and conversion of log messages.

Figure 3.7 illustrates the proposed approach; the virtual log knowledge graph at the center of the figure is constructed dynamically from dispersed log sources based on analysts' queries and linked to external and internal knowledge sources.

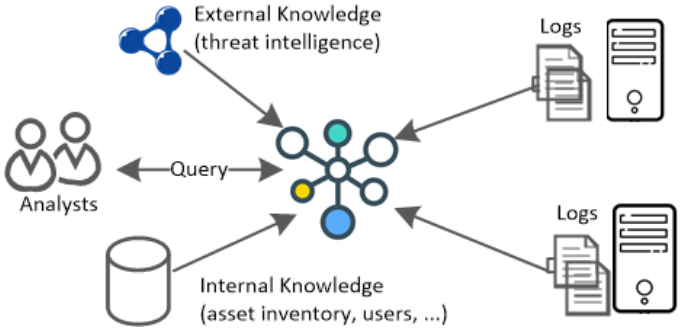

Figure 3.7.: Concept overview

A key advantage of the graph-based model of this virtual knowledge graph is that it provides a concise, flexible, and intuitive abstraction for the representation of various relations such as, e.g., connections in networked systems, hierarchies of processes on endpoints, associations between users and services, and chains of indicators of compromise. These connections automatically link log messages that are related through common entities (such as users, hosts, and IP addresses); such links are crucial in cybersecurity investigations, as threat agent activities typically leave traces in various log files that are often spread across multiple endpoints in a network, particularly in discovery, lateral movement, and exfiltration stages of an attack[23].

In contrast to traditional workflows that store log messages in a centralized repository, shifts the log parsing workload from ingestion to analysis time. This makes it possible to directly access and dynamically integrate the most granular raw log data without any loss of information that would occur if the logs were pre-filtered and aggregated – typical steps performed before transferring them to a central archive.

tackles a number of pressing challenges in security log analysis (discussed in Section 3.2.3) and facilitates *(i)* ad-hoc integration and semantic analyses on raw log data without prior centralized materialization, *(ii)* the collection of evidence-based knowledge from heterogeneous log sources, *(iii)* automated linking of fragmented knowledge about system states and activities, and *(iv)* automated linking to external security knowledge (such as, e.g., attack patterns, threat implications, actionable advice).

The remainder of this paper is organized as follows: Section 3.2.2 provides an overview of related work in this area, and in Section 3.2.3, we discuss challenges in log analysis and derive requirements for our approach. Section 3.2.4 introduces the proposed architecture and describes the components for virtual log knowledge graph construction in detail. In Section 3.2.5 we present a prototypical implementation of the architecture and illustrate its use in three application scenarios. We evaluate our approach on a systematically generated log dataset in Section 3.2.6 and discuss benefits and limitations of the presented approach in Section 3.2.7. Finally, we conclude with an outlook on future work in Section 3.2.8.

3.2.2. Related Work

In this section, we organize the related literature – from general to specific – into three categories:

[23]ATT&CK Matrix for Enterprise `https://attack.mitre.org/`

Log Management and Analytics The rising number, volume and variety of logs has created the need for systematic computer security log management [106] and motivated the development of a wide range of log-analytic techniques to derive knowledge from these logs [107], including anomaly detection [108, 109], clustering [110], and rule-base intrusion detection [111].

In the context of our work, approaches that aim to integrate and analyze log data across multiple sources are particularly relevant. Security Information and Event Management (SIEM) are widely used to provide a centralized view on security-relevant events inside an organization and focus on data aggregation, correlation, and typically rule-based alerting. These ideas are outlined in numerous guidelines and industrial best practices such as the NIST CSF[24] and NIST SP 800-92[25]. In this current state of practice, various commercial offerings provide centralized solutions[26].

Whereas SIEMs facilitate centralized log aggregation and management, however, they lack a semantic foundation for the managed log data and consequently typically do not make it easy to link, contextualize, and interpret events against the background of domain knowledge. To tackle these challenges, [112] create a foundation for semantic SIEMs that introduces a Security Strategy Meta-Model to enable interrelating information from different domains and abstraction levels. In a similar vein, [100] proposes a hybrid relational-ontological architecture to overcome cross-domain modeling, schema complexity, and scalability limitations in SIEMs. This approach combines existing relational SIEM data repositories with external vulnerability information, i.e., CVE.

Graph-based Log Integration and Analysis More closely related to the approach proposed in this paper, a stream of literature has emerged that recognizes the interrelated nature of log data and conceives log events and their connections as graphs – i.e., labeled property graphs (LPGs) or semantically explicit RDF knowledge graphs.

In the former category, LPGs are stored in graph databases and queried through specialized graph query languages. For network log files, for instance, [113] proposes an approach that materializes the log in a Neo4J graph database and makes it available for querying and visualization. The approach is limited to a single log source and focuses exclusively on network log analysis. Similar to this, CyGraph [114] is a framework that integrates isolated data and events in a unified graph-based cybersecurity model to assist decision making and improve situational awareness. It is based on a domain-specific language CyQL to express graph patterns and uses a third-party tool for visualization.

Another stream of literature transforms logs into RDF knowledge graphs that can be queried with SPARQL, a standardized query language. Early work such as [115] has illustrated that the use of explicit semantics can help to avoid ambiguity, impose meaning on raw log data, and facilitate correlation in order to lower the barrier for log interpretation and analysis. In this case, however, the log source considered is limited to a firewall log. Approach like this do not directly transform log data into a graph, but impose semantics to existing raw log data or log data stored in a relational database. More recently, approaches have been developed that aim to transform log data from multiple sources into an integrated log knowledge graph.

[24]NIST Cybersecurity framework
[25]NIST SP 800-92 Guide to Computer Security Log Management
[26]cf., e.g., Gartner Magic Quadrant for SIEM 2021

For structured log files, [116] discusses an approach that analyzes their schema to generate a semantic representation of their contents in RDF. Similar to our work, the approach links log entities to external background knowledge (e.g., DBPedia), but the log source processed is limited to a single log type. Ref. [117] leverages an ontology to correlate alerts from multiple Intrusion Detection Systems (IDSs) with the goal to reduce the number of false-positive and false-negative alerts. It relies on a shared vocabulary to facilitate security information exchange (e.g., IDMEF, STIX, TAXII), but does not facilitate linking to other log sources that may contain indicators of attacks.

LEKG [118] provides a log extraction approach to construct knowledge graphs using inference rules and validates them from a background knowledge graph. It uses local inference rules to create graph elements (triples) which can later be used to identify and generate causal relations between events. Compared to , the approach does not aim to provide integration and interlinking over multiple heterogeneous log sources.

To facilitate log integration, contextualization and linking to background knowledge, [27] proposes a modular log vocabulary that enables log harmonization and integration between heterogeneous log sources. A recent approach proposed in [30] introduces a vocabulary and architecture to collect, extract, and correlate heterogeneous low-level file access events from Linux and Windows event logs.

Compared to the approach in this paper, the approaches discussed so far rely on a centralized repository. A methodologically similar approach for log analysis outside of the security domain has also been introduced in [119], which leverages ontology-based data access to support log extraction and data preparation on legacy information systems for process mining. In contrast to this paper, the focus is on log data from legacy systems in existing relational schemas and on ontology-based query translation.

Decentralized Security Log Analysis Decentralized event correlation for intrusion detection was introduced in early work such as [120], where the authors propose a specification language to describe intrusions in a distributed pattern and use a peer-to-peer system to detect attacks. In this decentralized approach, the focus is on individual IDS events only. To address scalability limitations of centralized log processing, [102] distributes correlation workloads across networks to the event-producing hosts. Similar to this approach, we aim to tackle challenges of centralized log analysis. However, we leverage semantic web technologies to also provide contextualization and linking to external background knowledge. In the cloud environment, [121] proposes a distributed and parallel security log analysis framework that provides analyses of a massive number of systems, networks, and transaction logs in a scalable manner. It utilizes the two-level master-slave model to distribute, execute, and harvest tasks for log analysis. The framework is specific to cloud-based infrastructures and lacks the graph-oriented data model and contextualization and querying capabilities of our approach.

3.2.3. Requirements

Existing log management systems typically ingest log sources from multiple log-producing endpoints and store them in a central repository for further processing. Before they can be analyzed, such systems typically parse and index these logs, which typically requires considerable amounts of disk space to store the data as well as computational power for

log analysis. The concentrated network bandwidth, CPU, memory, and disk space needs limit the scalability of such centralized approaches.

Decentralized log analysis, by contrast, (partly) shifts the computational workloads involved in log pre-processing (e.g., acquisition, extraction, and parsing) and analysis to the log-producing hosts [102]. This model has the potential for higher scalability and applicability in large-scale settings where the scope of the infrastructure prohibits effective centralization of all potentially relevant log sources in a single repository.

Existing approaches for decentralized log processing, however, primarily aim to provide correlation and alerting capabilities, rather than the ability to query dispersed log data in a decentralized manner. Furthermore, they lack effective means for semantic integration, contextualization, and linking, i.e., dynamically creating connections between entities and potentially involving externally available security information. They also typically have to ingest all log data continuously on the local endpoints, which increases continuous resource consumption across the infrastructure.

In this paper, we tackle these challenges and propose a distributed approach for security log integration and analysis. Thereby, we facilitate ad-hoc querying of dispersed raw log sources without prior ingestion and aggregation in order to address the following requirements (R):

- ***R.1 - Resource-efficiency*** Traditional log management systems such as SIEMs perform continuous log ingestion and preprocessing, typically from multiple monitoring endpoints, before analyzing the log data. This requires considerable resources as all data needs to be extracted and parsed in advance. A key requirement for distributed security log analysis is to avoid unnecessary ex-ante log preprocessing (acquisition, extraction, and parsing), thus minimizing resource requirements in terms of centralized storage space and network bandwidth. This should make log analysis both more efficient and more scalable.

- ***R.2 - Aggregation and integration over multiple endpoints*** As discussed in the context of the motivating example in Section 3.2.1, a single attack may leave traces in multiple log sources, which can be scattered across different systems and hosts. To detect sophisticated attacks, it is therefore necessary to identify and connect such isolated indicators of compromise [27]. The proposed solution should therefore provide the ability to execute federated queries across multiple monitoring endpoints concurrently and deliver integrated results. This makes it possible to detect not only potential attack actions, but also to obtain an integrated picture of the overall attack (e.g., through linking of log entries).

- ***R.3 - Integration, Contextualization & Background-Linking*** the interpretation of log information for attack investigation depends highly on the context; isolated indicators on their own are, however, often inconspicuous in their local context. Therefore, the proposed approach should provide the ability to contextualize disparate log information, integrate it, and link it to internal background knowledge and external security information.

- ***R.4 - Standards-based query language*** The proposed approach should provide an expressive, standards-based query language for log analysis. This should make it easier for analysts to formulate queries (e.g., define rules) during attack investigation in an intuitive and declarative manner.

3.2.4. VloGraph Framework Architecture

Based on the requirements set out in Section 3.2.3, we propose , an approach and architecture for security log analytics based on the concept of Virtual Knowledge Graphs (VKGs). The proposed approach leverages Semantic Web Technologies that provide *(i)* a standardized graph-based representation to describe data and their relationships flexibly using RDF[27], *(ii)* semantic linking and alignment to integrate multiple heterogeneous log data and other resources (e.g., internal/external background knowledge), and *(iii)* a standardized semantic query language (i.e. SPARQL[28]) to retrieve and manipulate RDF data.

To address **R.1**, our approach does not rely on centralized log processing, i.e., we only extract relevant log events based on the temporal scope and structure of a given query and its query parameters. Specifically, we only extract lines in a log file that *(i)* are within the temporal scope of the query, and *(ii)* may contain relevant information based on the specified query parameters and filters. The identified log lines are extracted, parsed, lifted to RDF, compressed, and temporarily stored in a local cache on the respective endpoint. This approach implements the concept of data virtualization and facilitates on-demand log processing. By shifting computational loads to individual monitoring agents and only extracting log entries that are relevant for a given query, this approach can significantly reduce unnecessary log data processing. Furthermore, due to the use of RDF compression techniques, the transferred data is rather small; we discuss this further in Section 3.2.6.

To address **R.2**, we distribute queries over multiple log sources across distributed endpoints and combine the results in a single integrated output via query federation[29].

To address **R.3**, we interlink and contextualize our extracted log data with internal and external background knowledge – such as, e.g., IT asset information and cybersecurity knowledge – via semantic linking and alignment. Finally, we use SPARQL to formulate queries and perform log analyses, which addresses **R.4**. We will illustrate SPARQL query federation and contextualization in multiple application scenarios for in Section 3.2.5.

Figure 3.8 illustrates the virtual log graph and query federation architecture for log analysis; it consists of two main components: *(i)* a ***Log Parser*** on each host, which receives and translates queries, extracts raw log data from hosts, parses the extracted log data into an RDF representation, compresses the resulting RDF data into a binary format, and sends the results back to a *(ii)* ***Query Processor***, which provides an interface to formulate SPARQL queries and distributes the queries among individual endpoints; furthermore, it retrieves the individual log graphs from the endpoints, integrates them, and presents the resulting integrated graph. In the following, we explain the individual components in detail.

SPARQL Query Editor This sub-component is part of the *Query Processor* and allows analysts to define settings for query execution, including: *(i) Target Hosts*: a set of endpoints to be included in the log analysis, *(ii) Knowledge bases*: a collection of internal and/or external sources of background knowledge that should be included in the query

[27] `https://www.w3.org/RDF/`
[28] `https://www.w3.org/TR/rdf-sparql-query/`
[29] `https://www.w3.org/TR/sparql11-service-description/`

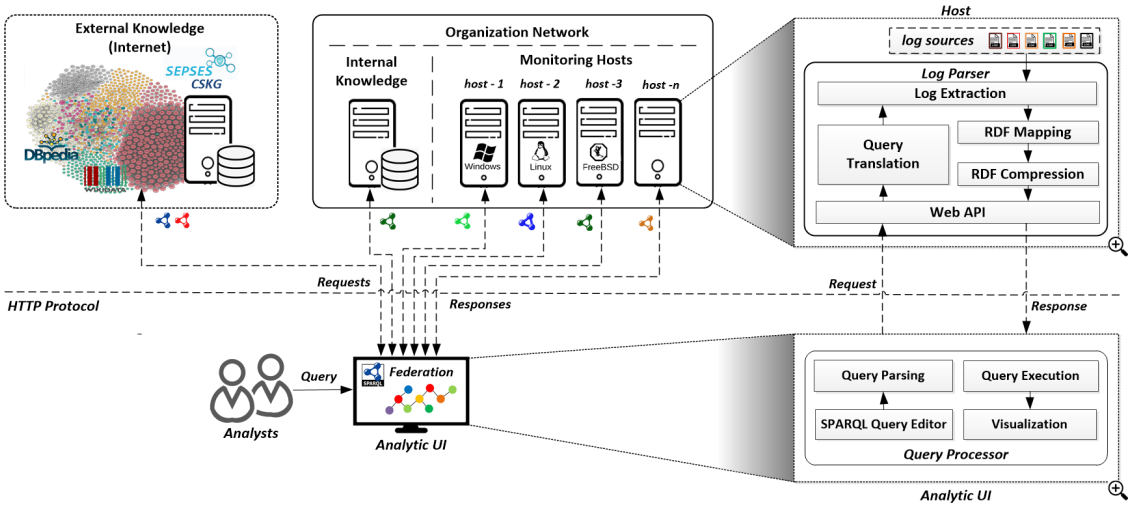

Figure 3.8.: Virtual log graph and query federation architecture

execution (e.g., IT infrastructure, cyber threat intelligence knowledge bases, etc.), *(iii) Time Interval*: the time range of interest for the log analysis (i.e., start time and end time).

Query Parsing The SPARQL query specification [122] provides a number of alternative syntaxes to formulate queries. For uniform access to the properties and variables inside the query, we therefore parse the raw SPARQL syntax into a structured format prior to transferring the query to the monitoring hosts. The prepared SPARQL query is then sent as a parameter to the *Query Translator* via the *Web API* in the *Log Parser* Component.

Query Translation This subcomponent decomposes the SPARQL query to identify relevant properties for log source selection and log line matching. Algorithm 1 outlines the general query translation procedure, which identifies relevant log sources and log lines based on three criteria, i.e., *(i)* prefixes used in the query, *(ii)* triples, and *(iii)* filters. $Prefixes(P)$ is a set of log vocabulary prefixes that appear in a given query Q. In each query, the contained prefixes will be used by the query translator to identify relevant log sources. Available prefixes can be configured to the respected log sources in the *Log Parser* configuration on each client, including, e.g., the path to the local log file location. As an example, `PREFIX auth: <http://w3id.org/authLog>` is the prefix for *AuthLog*; it's presence in a query indicates that the *AuthLog* on the selected hosts will be included in the log processing.

Triples (T) is a set of triples that appear in a query, each represented as *Triple Pattern* or a *Basic Graph Pattern (BGP)* (i.e. `<Subject> <Predicate> <Object>`).

We match these triples to log lines (e.g., hosts and users) as follows: Function $getTriplePattern(Q)$ collects the triple patterns T contained within the query Q. For each triple statement in a query, we identify the type of `Object` $T_{i_{Object}}$. If the type is *Literal*, we identify the $T_{i_{Predicate}}$ as well. For example, for the triple `{?Subject cl:originatesFrom "Host1"}`, the function $getLogProperty()$ identifies $T_{i_{Object}}$ "Host1", and additionally, looks up the property range provided in *regexPatterns* (RP). *regexPatterns* (RP)[30] is the back-

[30]https://github.com/sepses/VloGParser/blob/hdt-version/experiment/pattern/regexPattern.ttl

Algorithm 1: Query translation

Input: SPARQL Query (Q), Vocabulary (V), regexPatterns (RP)
Output: QueryElements (Qe)

Prefixes $P = \{P_1,...,P_n\} \in Q$;
Triples $T = \{Subject, Predicate, Object\} \in Q$;
Filters $F = \{Variable, Value\} \in Q$;
Function `translateQuery(`Q,V,RP`)`**:**
    $P \leftarrow getPrefix(Q)$;
    $T \leftarrow getTriplePattern(Q)$;
    foreach *Triple* $T_i \in T$ **do**
        if *type(*$T_{i_{Object}}$*)=Literal* **then**
            $logProperty \leftarrow getLogProperty(T_{i_{Predicate}},V,RP)$;
            $keyValue \leftarrow \{logProperty, T_{i_{Object}}\}$;
        end
        $triplesKV \mathrel{+}= keyValue$;
    end
    $F \leftarrow getFilterStatement(Q)$;
    foreach *Filter* $F_i \in F$ **do**
        if *type(*$F_{i_{Value}}$*)=Literal* **then**
            $predicate \leftarrow getPredicate(Q,F_{i_{Variable}})$;
            $logProperty \leftarrow getLogProperty(predicate,V,RP)$;
            $keyValue \leftarrow \{logProperty, F_{i_{Value}}\}$;
        end
        $filtersKV \mathrel{+}= keyValue$;
    end
    $Qe \leftarrow \{P,triplesKV,filtersKV\}$;
    return Qe;
End Function

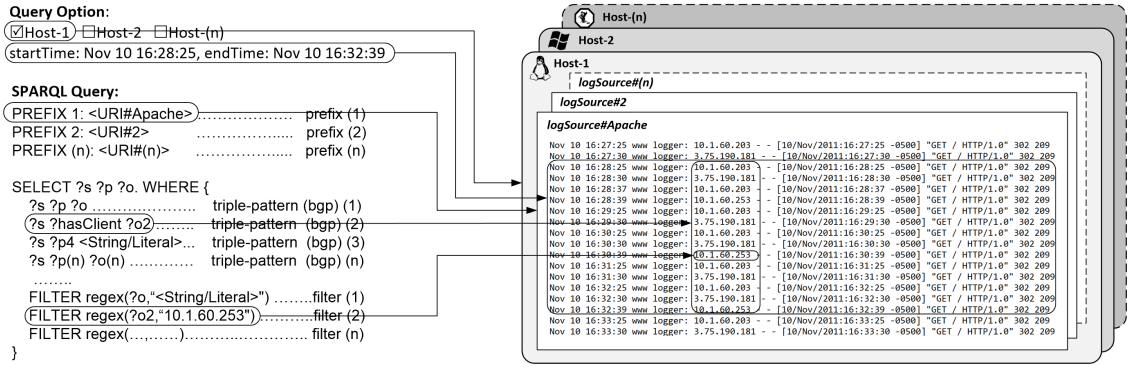

Figure 3.9.: SPARQL Query translation example

ground knowledge that links property terms in a vocabulary to the terms in a log entry and the respected regular expression pattern. For example, the property $cl : originatesFrom$ is linked to the concept `"hostname"` in $regexPattern$ (RP), which has a connected regex pattern for the extraction of host names. The output of the $getLogProperty()$ function is a set of $<logProperty, T_{i_{Object}}>$ key-value pairs.

Similar to triples, we also include $Filters$ (F) that appear in a query Q for log-line matching. Filter statements contain the term `FILTER` and a set of pairs (i.e., $Variable$ and $Value$), therefore each $Filter$ statement F_i has the members $Variable$ $F_{i_{Variable}}$ and $Value$ $F_{i_{Value}}$. Currently, we cover FILTER with simple pattern matching and regular expressions such as $FILTER$ ($?variable = "StringValue"$), $FILTER$ $regex(str(?variable), "StringValue"))$.

The function $getFilterStatement(Q)$ is used to retrieve these filter statements from the query and to identify the type of $Value$ $F_{i_{Value}}$. If it is a Literal, the $getPredicate(Q)$ function will look for the connected $predicate$. Similar to the technique used in triples, we use $getLogProperty()$ to retrieve the regular expression defined in $regexPattern$ (RP). Finally, the collected prefixes and retrieved key-value pairs, both from triples and filters, will be stored in $QueryElements$ (Qe) for further processing. Figure 3.9 depicts a SPARQL query translation example.

Log Extraction This component is part of the *Log Parser* that extracts the selected raw log lines and splits them into a key-value pair representation by means of predefined regular expression patterns. As outlined in Algorithm 2, Log sources (Ls) are included based on the prefixes that appear in the query.

For each log line (Ln_j) in a log source, we check whether the log timestamp ($LnO_{logTime}$) is within the defined TimeFrame (Tf).[31]

If this condition is satisfied, the $matchLog()$ function checks the logline property ($LnO_{logProperties}$) against the set of queried triples ($Qe_{triplesKV}$) and filters ($Qe_{filtersKV}$). If the log line matches the requirements, the selected log line will be parsed using $parseLine()$ based on predefined regular expression patterns. The resulting parsed queries will be accumulated and cached in a temporary file for subsequent processing.

[31] In this version of the algorithm, we leverage the monotonicity assumption in the log context by stopping the log parsing once the end of the temporal window of interest is reached in a log file (i.e., we assume that log lines do not appear out of order). This can be adapted, if required for a specific log source.

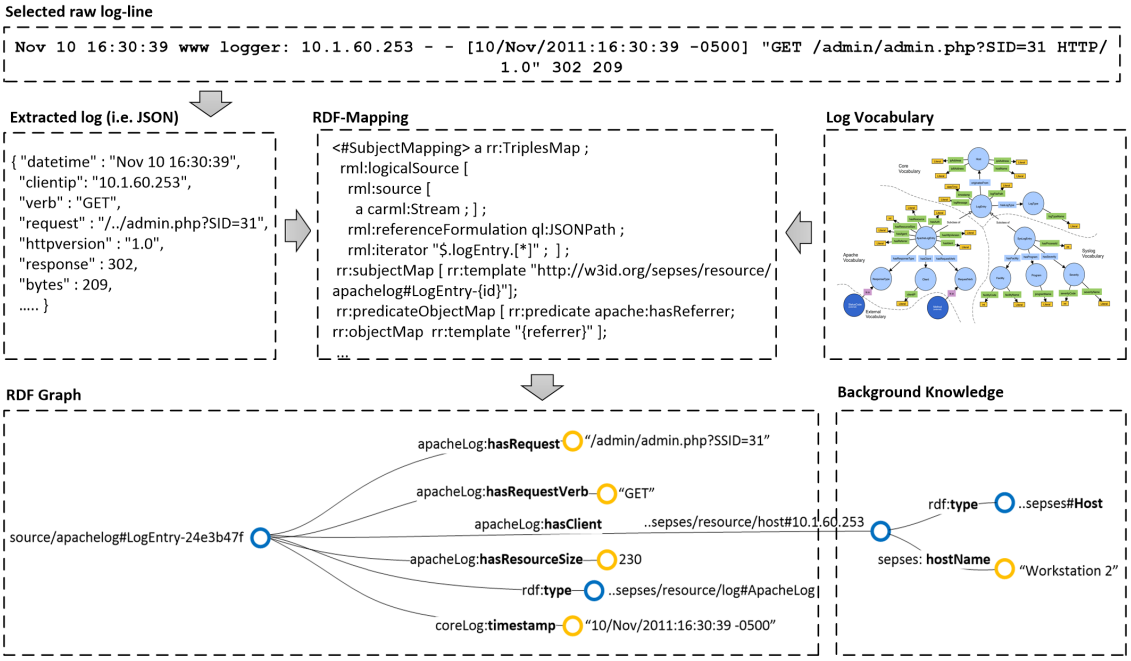

Figure 3.10.: Log graph generation overview

RDF Mapping This sub-component of the *Log Parser* maps and parses the extracted log data into RDF. It uses the standard RDF mapping language to map between the log data and the vocabulary. Different log sources use a common core log vocabulary (e.g., SEPSES coreLog[32]) for common terms (e.g., host, user, message) and can define extensions for specific terms (e.g., the `request` term in ApacheLog). The RDF Mapping also maps terms from a log entry to specific background knowledge (e.g., hosts in a log entry are linked to their host `type` according to the background knowledge). Figure 3.10 provides an overview of the log graph generation process.

RDF Compression This sub-component is part of the *Log Parser*, which transforms the resulting RDF output produced by the *RDF Mapper* into a compact version of RDF. This compression results in a size reduction by an order of magnitude, which has significant advantages in our framework: *(i)* it enables fast data transfer to the *Query Processor* component and thereby reduces latency; *(ii)* it makes the query execution itself more efficient as the compressed RDF version enables query operations without prior decompression directly on the binary representation [123]. We discuss the implementation of this component based on existing libraries in Section 3.2.5 and evaluate the effect of compression on the query execution performance on virtual log graphs in Section 3.2.6.

Query Execution Once the pre-processing on each target host has been completed and the compressed RDF data results have been successfully sent back to the *Query Processor*, a query engine executes the given queries against the compressed RDF data. If multiple hosts were defined in the query, the query engine will perform query federation over

[32] `https://w3id.org/sepses/vocab/log/core/`

Algorithm 2: Log Extraction and RDF Mapping

Input: SPARQL Query (Q), TimeFrame (Tf), LogSources (Ls)
Output: Response (R)

TimeFrame $Tf = \{startT, endT\}$;
LogSources $Ls = \{Ls_1, ..., Ls_n\}$;
LogLines $Ln = \{Ln_1, ..., Ln_n\} \epsilon\ Ls$;
LogSourceOptions $LsO = \{vocabulary, regexPatterns\} \epsilon\ Ls$;
LogLineOptions $LnO = \{logTime, logProperties\} \epsilon\ Ln$;
QueryElements $Qe = \{prefixes, triplesKV, filtersKV\}$;
$Qe \leftarrow translateQuery(Q, LsO_{vocabulary}, LsO_{regexPatterns})$;
foreach *LogSource* $Ls_i \epsilon\ Ls$ **do**
    if $Qe_{prefixes}$ *contains* $LsO_{i_{vocabulary}}$ **then**
        foreach *LogLines* $Ln_j \epsilon\ Ln$ **do**
            $lt \leftarrow LnO_{j_{LogTime}}$;
            if $lt < Tf_{endT} = False$ **then**
                break;
            end
            if $lt > Tf_{startT}$ && $lt < Tf_{endT}$ **then**
                $ml \leftarrow matchLog(LnO_{j_{logProperties}}, Qe_{triplesKV}, Qe_{filtersKV})$;
                if $ml = True$ **then**
                    $parsedLine \leftarrow parseLine(Ln_j)$;
                end
            end
            $parsedData \mathrel{+}= parsedLine$;
        end
        $RDFData \leftarrow RDFMapping(parsedData)$;
        $result \leftarrow compressData(RDFData)$;
        if $result = True$ **then**
            $response \leftarrow$ "$Success$";
        end
    end
    return $response$;
end

multiple compressed RDF data from those individual hosts and combine the query results into an integrated output.

Furthermore, due to semantic query federation, external data sources are automatically linked in the query results in case they were referenced in the query (cf. Section 3.2.5 for an example that links IDS messages to the SEPSES-CSKG[33]).

Visualization Finally, this component presents the query results to the user; depending on the SPARQL query form[34], e.g.,: *(i)* SELECT - returns the variables bound in the query pattern, *(ii)* CONSTRUCT - returns an RDF graph specified by a graph template, and *(iii)* ASK - returns a boolean indicating whether a query pattern matches. The returned result can be either in JSON or RDF format, and the resulting data can be presented to the user as an HTML table, chart, graph visualization, or it can be downloaded as a file.

3.2.5. Implementation & Application Scenarios

In this section, we discuss the implementation of framework[35] and demonstrate its feasibility by means of three application scenarios.

3.2.5.1. Implementation

The prototype relies on a number of existing open source tools and libraries. Specifically, we implement the *Log Parser*[36] component as a Java-based tool that is installed and run on each monitoring host. It supports log parsing from multiple different OSs (e.g., Windows, Linux, etc.) and heterogeneous log files (e.g., authlog, apachelog, IISlog, IDSlog). For the *Log Extraction* component, we integrate *Grok Patterns*[37], a collection of composeable regular expression patterns that can be reused across log sources. Furthermore, we use *CARML*[38] as an *RDF Mapping* component based on RML mappings[39] to map the extracted log data into RDF. For the *RDF Compression* component, we leverage the *HDT* [123] library to efficiently compress the resulting RDF data into a compact, binary format that allows query operations without prior decompression.

For the analysis interface, we implemented a *Query Processor*[40] component as a web-application that receives SPARQL queries, sends them to multiple target hosts, and presents the resulting graph to the analyst. Figure 3.11 shows the user interface of the application, which consist of *(i)* Query Options, including e.g., target hosts, background knowledge, analysis timeframe, as well as predefined queries to select. *(ii)* SPARQL Query Input to formulate and execute SPARQL queries, and *(iii)* Query Results to present the output of an the executed query. The query execution is implemented on top of the *Comunica* [124] query engine that supports query federation over multiple linked data interfaces including HDT files and SPARQL endpoints.

[33] `http://w3id.org/sepses/sparql`
[34] `https://www.w3.org/TR/sparql11-query/#QueryForms`
[35] Source code available at `https://github.com/sepses`
[36] `https://github.com/sepses/VloGParser`
[37] `https://github.com/elastic/logstash/blob/v1.4.2/patterns/grok-patterns`
[38] `https://github.com/carml/carml`
[39] cf. `http://rml.io/`
[40] `https://github.com/sepses/VloGraphQueryProcessor`

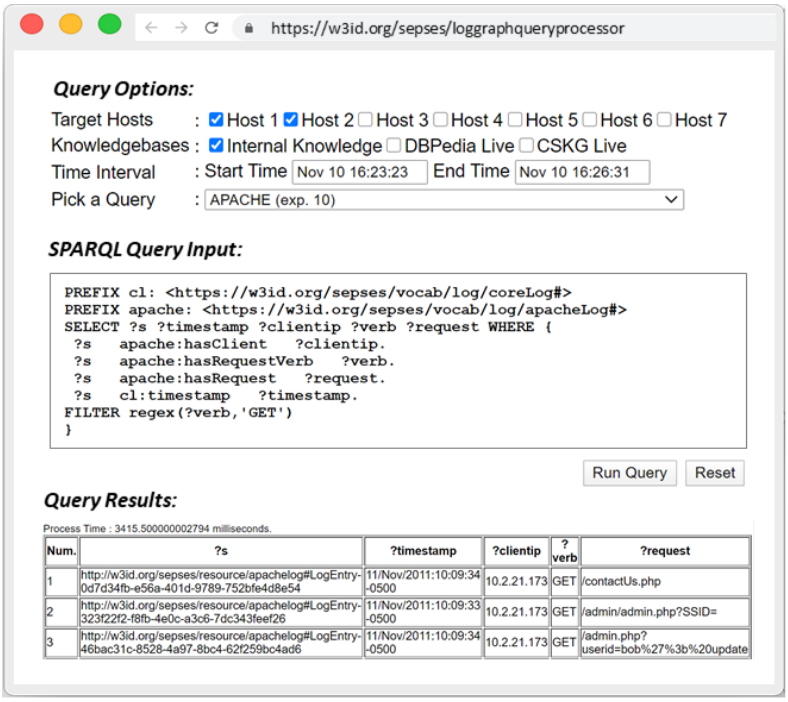

Figure 3.11.: SPARQL query editor interface

3.2.5.2. Application Scenarios

Scenario I - Web access log analysis In this scenario, we simulated two hosts (Windows10 and Ubuntu) with different web servers (Apache and IIS) and analyze their access logs together. In order to identify access from a specific IP address (e.g., 192.168.2.1), we formulate the SPARQL query depicted in Listing 3.3. We specify the client's IP address with `access:hasClient res:ip-192.168.2.1` and filter for "GET" requests via `accs:hasRequestVerb res:GET`. In the query options, we selected the timeframe (from Nov 11 10:00:04 to Nov 11 10:10:04) as well as the two target hosts.

The query results in Table 3.5 show the access information with their log sources and

```
PREFIX cl: <https://w3id.org/sepses/vocab/log/coreLog#>
PREFIX accs: <https://w3id.org/sepses/vocab/log/accessLog#>
PREFIX res: <https://w3id.org/resource/access#>

SELECT ?logType ?hostOS ?hostIp  ?verb ?request
    WHERE {
        ?logEntry cl:originatesFrom ?host.
        ?host cl:hostOS ?hostOS.
        ?logEntry cl:hasLogType ?logType.
        ?host cl:ipAddress ?hostIp.
        ?logEntry accs:hasRequestVerb res:GET.
        ?logEntry accs:hasRequest ?request.
        ?logEntry accs:hasClient res:ip-192.168.2.1.
    } LIMIT 4
```

Listing 3.3: Web access query

Table 3.5.: Web access query result (excerpt)

logType	hostOS	hostIp	verb	request
IIS	Win10	192.168.0.113	*GET*	*/employee.asp&id=12345 ...*
apache	Ubuntu	192.168.0.111	*GET*	*/admin.php?userid=bob...*
apache	Ubuntu	192.168.0.111	*GET*	*/salary.php*
IIS	Win10	192.168.0.113	*GET*	*/global/lwb.min.js ...*

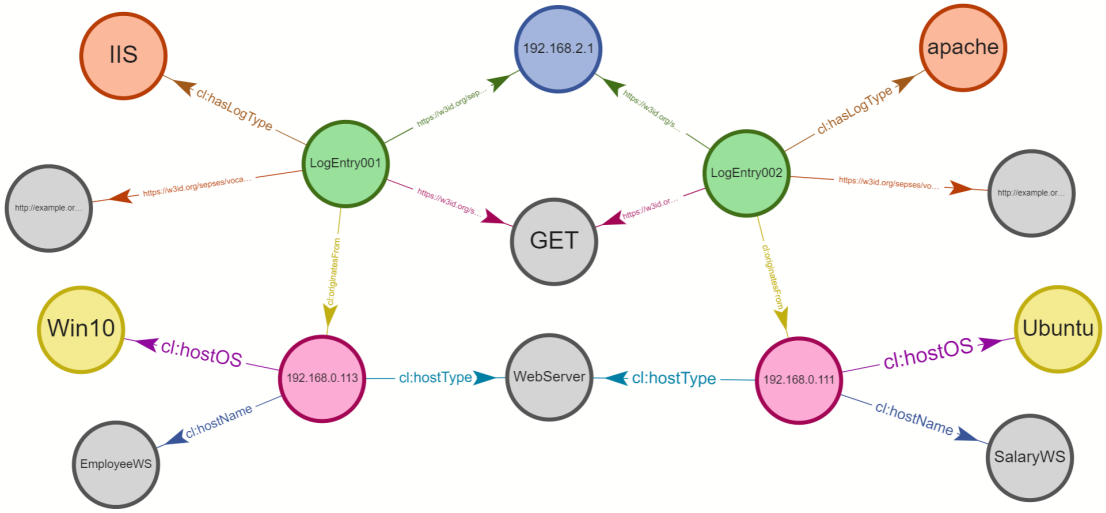

Figure 3.12.: Web access query result visualization (excerpt)

types (`cl:IIS` and `cl:apache`), the host OS (`Win10` and `ubuntu`) with their IPs, the request method, and request paths. Figure 3.12 depicts the graph visualization of the result.

Scenario II - Network monitoring In this scenario, we illustrate how our prototype provides semantic integration, generalization, and entity resolution. We simulated SSH login activities[41] across different servers (e.g., *DatabaseServer, WebServer, FileServer*) with multiple demo users (e.g., *Bob and Alice*) and then queried the authlog files with our federated approach.

Typically, atomic information on the log entry level is not explicitly linked to semantic concepts. Hence, we added extractors to, e.g., detect specific log events from log messages and map them to event types from our internal background knowledge[42] (e.g., `event:Login`, `event:Logout`). Furthermore, we added concept mappings for program names, IP addresses etc. (cf. Section 3.2.4).

Now, an analyst can formulate a SPARQL query as shown in Listing 3.4 to extract successful login events from SSH connections. The query results in Table 3.6 and Figure 3.13 show successful logins via SSH over multiple hosts in the specified time range (from Dec 10 13:30:23 to Dec 10 14:53:06). The host type and target IP address come from internal background knowledge, as the host name is connected to a specific host type.

[41] `http://bit.ly/scenario2dataset`
[42] `https://w3id.org/sepses/knowledge/eventKnowledge.ttl`

```
PREFIX cl: <https://w3id.org/sepses/vocab/log/core#>
PREFIX auth: <https://w3id.org/sepses/vocab/log/authLog#>
PREFIX sys: <https://w3id.org/sepses/resource/system#>
PREFIX ev: <https://w3id.org/sepses/resource/event#>

SELECT ?timestamp ?user ?sourceIp ?targetHostType ?targetIp
    WHERE {
        ?logEntry cl:timestamp ?timestamp.
        ?logEntry auth:hasUser ?user.
        ?logEntry auth:hasSourceIp ?sourceIp.
        ?logEntry auth:hasTargetHost ?th.
        ?logEntry auth:hasAuthEvent ?ae.
        ?ae sys:partOfEvent ev:Login.
        ?th sys:hostType ?targetHostType.
        ?th cl:IpAddress ?targetIp.
    } LIMIT 4
```

Listing 3.4: SSH connections query

This information can be a starting point for security analysts to explore the rich context of the events in the virtual knowledge graph.

Table 3.6.: SSH connections query result (excerpt)

timestamp	user	sourceIp	targetHostType	targetIp
Dec 10 13:30:23	Bob	172.24.66.19	DatabaseServer	192.168.2.1
Dec 10 13:33:31	Alice	172.24.2.1	WebServer	192.168.2.2
Dec 10 13:38:16	Alice	172.24.2.1	DatabaseServer	192.168.1.3
Dec 10 14:53:06	Bob	172.24.66.19	FileServer	192.168.2.4

Scenario III - Threat detection and ATT&CK linking In this scenario, we demonstrate how the framework leverages existing threat detection rules to identify Indicators of Compromise (IoCs) from log sources and link them to the respective attack techniques and tactics. For this scenario, we used an existing log dataset [104] as described in the motivation example in Section 3.2.1. To define our rule-based threat detection queries, we relied on existing community-based threat detection rules such as *Sigma*[43] and transformed them into RDF/SPARQL. Furthermore, we used the ATT&CK-KG [31], a continuously updated cybersecurity knowledge graph generated from the MITRE ATT&CK Matrix[44] in order to link cyber attacks to adversary techniques and tactics.

Listing 3.5 shows an example query for this scenario. Using the transformed *Sigma* rule as internal knowledge, we can list *suspicious* keywords defined in the rules (i.e., via `?sigma sigma:keywords ?keywords`) and use them to filter messages from the targeted log sources. In this case, we target request messages in Apache log (see `?logEntry apache:hasRequest ?req`) and filter them against the keywords (`FILTER regex(str(?req), ?keywords)`). Next, we link the detected log entries to the respective attack techniques (note that *Sigma* typically provides *tags* that associate its rules with ATT&CK techniques).

This can be done via `?sigma rule:hasAttackTechnique ?techn`. The query leverages linked data principles to include external background knowledge from the ATT&CK-KG,

[43] `https://github.com/SigmaHQ/sigma`
[44] `https://attack.mitre.org/matrices/enterprise/`

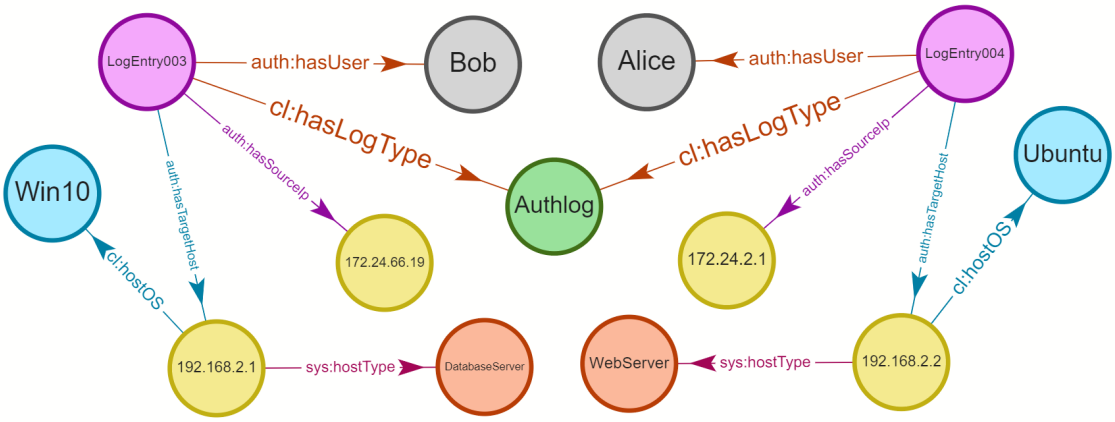

Figure 3.13.: SSH connections query result visualization (excerpt)

```
PREFIX cl: <https://w3id.org/sepses/vocab/log/core#>
PREFIX apache: <https://w3id.org/sepses/vocab/log/apache#>
PREFIX sigma: <http://w3id.org/sepses/vocab/rule/sigma#>
PREFIX rule: <http://w3id.org/sepses/vocab/rule#>
PREFIX attack: <http://w3id.org/sepses/vocab/ref/attack#>
PREFIX dcterm: <http://purl.org/dc/terms/>

SELECT ?logEntry  ?timestamp ?host ?keywords ?techn ?desc ?tactic ?capec
  WHERE {
    ?logEntry apache:hasRequest ?req ;
              cl:originatesFrom ?host;
              cl:timestamp ?timestamp.
    FILTER regex(str(?req),?keywords)
        { SELECT ?keywords ?techn ?tactic {
            ?sigma sigma:keywords ?keywords.
            OPTIONAL {
                ?sigma rule:hasAttackTechnique ?techn.
                ?techn dcterm:description ?desc.
                ?techn attack:accomplishesTactic ?tactic.
                ?techn attack:hasCAPEC ?capec.
            }
        }}
    } LIMIT 4
```

Listing 3.5: Rule-based threat detection and ATT&CK linking query

Table 3.7.: Scenario 4 Query Results (Excerpt)

logEntry	timestamp	host	keywords	techn	desc	tactic	capec
5f4a32...	Mar 04 19:18:43	cup	"whoami"	T1505.003	*"Web Shell"*	persistence	CAPEC-650
468226...	Mar 04 14:05:41	insect	"whoami"	T1505.003	*"Web Shell"*	persistence	CAPEC-650
7cff1d1...	Mar 04 19:18:46	cup	"curl"	T1190	*"Exploit Pub.."*	initial-access	-
600a59...	Mar 04 19:18:43	insect	"wget"	T1190	*"Exploit Pub.."*	initial-access	-

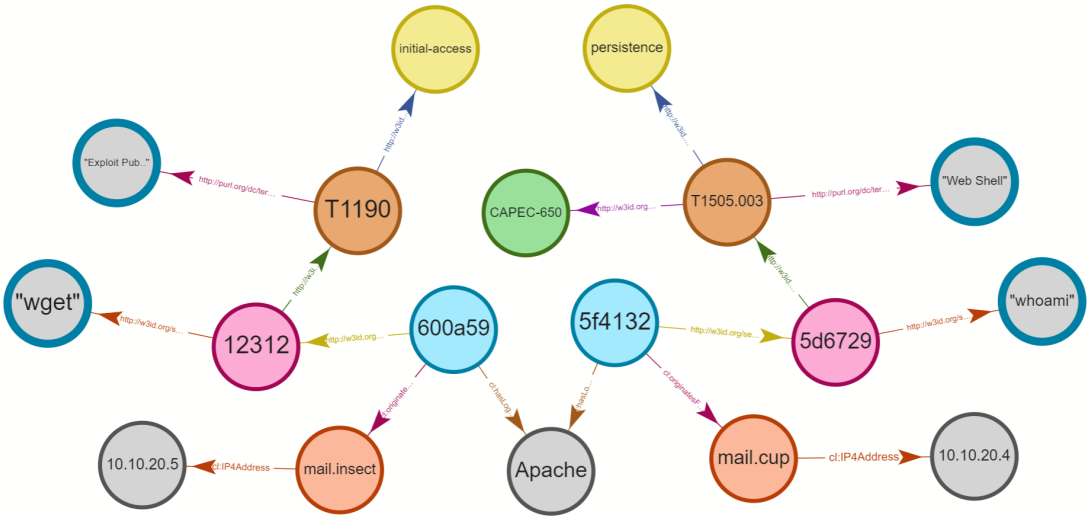

Figure 3.14.: Threat detection and ATT&CK linking visualization (excerpt)

which makes it possible to further link the identified attack technique detailed knowledge such as technique description (via `?techn dcterm:description ?desc`), attack tactic (via `?techn attack:accomplishesTactic ?tactic`), CAPEC[45] attack patterns (`?techn attack:hasCAPEC ?capec`), etc.

Table 3.7 and Figure 3.14 show the query results and visualization from this scenario. Several log entries from a particular host (mail.cup) are associated with suspicious keywords. For example, according to a Sigma rule [46] included as background knowledge, the "whoami" keyword is considered indicative of a *Web Shell* attack technique (T1505.003). This technique in turn is an instance of the tactic Persistence and of attack pattern CAPEC-650.

3.2.6. Evaluation

We evaluated the scalability of our approach by means of a set of experiments in non-federated and federated settings.

3.2.6.1. Evaluation Setup

The experiments were carried out on Microsoft Azure virtual machines with seven hosts (4 Windows and 3 Linux) with 2.59 GHz vCPU and 16 GB RAM each. We reused the log vocabularies from [27] and mapped them to the log data.

Dataset Overview We selected the systematically generated AIT log dataset (V1.1) that simulates six days of user access across multiple web servers including two attacks on the fifth day [104]. As summarized in Table 3.8, the dataset contains several log sources from four servers (*cup*, *insect*, *onion*, *spiral*). To reduce reading overhead and improve log

[45] `https://capec.mitre.org/`

[46] `https://github.com/SigmaHQ/sigma/blob/eb382c4a59b6d87e186ee269805fe2db2acf250e/rules/web/web_webshell_keyword.yml`

processing performance, we split large log files from the data set into smaller files – this can easily be replicated in a running system using log rotation mechanisms. Specifically, we split the files into chunks of 10k–100k log lines each and annotated them with original filename and time-range information.

Table 3.8.: Dataset description

LogType	#prop	*mail.cup.com*		*mail.insect.com*		*mail.onion.com*		*mail.spiral.com*	
		size	#lines	size	#lines	size	#lines	size	#lines
Audit	36	25 GB	123.6 M	22.7 GB	99.9 M	14.6 GB	68.8 M	12.4 GB	59.5 M
Apache	12	36.9 MB	148 K	44.4 MB	169.3 K	22.7 MB	81.9 K	24 .8 MB	100.4 K
Syslog	6	28.5 MB	158.6 K	26.9 MB	150.7 K	15 MB	86.6 K	15.1 MB	85.5 K
Exim	11	649 KB	7.3 K	567 KB	6.2 K	341 KB	3.9 K	355 KB	4 K
Authlog	11	128 KB	1.2 K	115 KB	1.1 K	102 KB	1 K	127 KB	1.2 K

3.2.6.2. Single-host evaluation

We measured the overall time for virtual log graph processing including *(i)* log reading (i.e., searching individual log lines), *(ii)* log extraction (i.e., extracting the raw log line into structured data), *(iii)* RDF Mapping (i.e., transforming json data into RDF), and *(iv)* RDF compression (i.e., compressing RDF into Header, Dictionary, Triples (HDT) format).

In our scenarios, we included several log sources; for each log source, we formulated a SPARQL query[47] to extract 1k, 3k, 5k, and 7k log lines filtering by timestamp in the query option. We report the average times over five runs for experiments with several log sources – i.e., Auditlog (AD), Apache for web logs (AP), Exim for mail transfer agent logs (EX), Syslog for Linux system logs (SY), and Authlog for authentication logs (AT) – for a single host in Figure 3.15. We used the data set from the first web server (i.e., *mail.cup.com*) in this evaluation. Note that we only extracted 1000k log lines from Authlog due to the small original file size (less than 1.2 k log lines).

We found that the performance for log graph extraction differs across the log sources. Constructing a log graph from Auditlog (AD) data resulted in the longest processing times followed by Apache, Exim, Syslog and Authlog. The overall log processing time scales linearly with the number of extracted log lines. Typically, the log extraction phase accounts for the largest proportion ($> 80\%$) of the overall log processing time. Furthermore, we found that the increase in log processing time with a growing number of extracted log lines is moderate, which suggests that the approach scales well to a large number of log lines.

Dynamic Log Graph Generation As discussed in the first part of the evaluation, execution times are mainly a function of the length of text in the log source and the granularity of the extraction patterns (i.e., log properties). As can be seen in Table 3.8, the log sources are heterogeneous and exhibit different levels of complexity. In our setup, Auditlog, for instance, has the largest number of log properties (36), followed by Apache (12), Exim (11), Authlog (11), and Syslog (6).

[47] `https://github.com/sepses/VloGraphQueryProcessor/tree/hdt-client-version/public/queries`

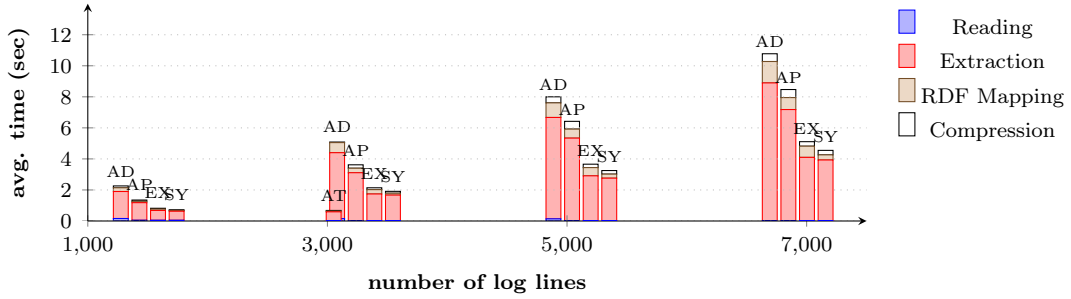

Figure 3.15.: Average log graph generation time for n log lines with a single host (36 extracted properties)

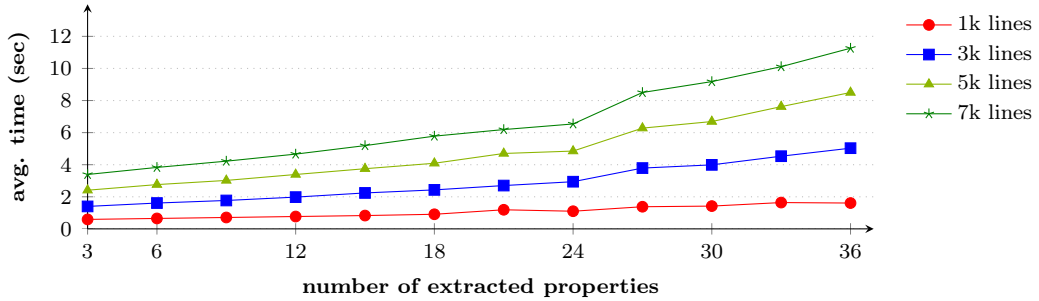

Figure 3.16.: Dynamic log graph generation time

Figure 3.16 shows an evaluation of log graph generation performance with respect to the complexity of the log source. We use the *Auditlog* for this evaluation as it has the highest number of log properties. Overall, the log graph generation performance grows linearly with the number of extracted log properties. Hence, queries that involve a smaller subset of properties (e.g., only *user* or *IP address* rather than all information that could potentially be extracted) will typically have smaller generation times.

Graph Compression Figure 3.17 shows the performance for log graph compression on the *Auditlog* dataset. We performed full property extraction (i.e., all 36 identified properties) against 5k, 10k, 15k, and 20k log-lines, respectively, and compare the original size of raw log data, the generated RDF graph in TURTLE[48] format (.ttl), and the compressed graph output in HDT format.

For 5k log lines (1 MB raw log) compression results in approximately 0.4 MB compared to 5.4 MB for the uncompressed RDF graph. 20k log lines (4 MB raw log) compresses to about 1.87 MB from 21.4 MB uncompressed generated RDF graph. Overall, the compressed version is typically less than half the size of the original raw log and 10x smaller than the generated RDF graph. The resulting graph output would be even smaller for fewer extracted properties, minimizing resource requirements (i.e. storage/disk space).

[48] `https://www.w3.org/TR/turtle/`

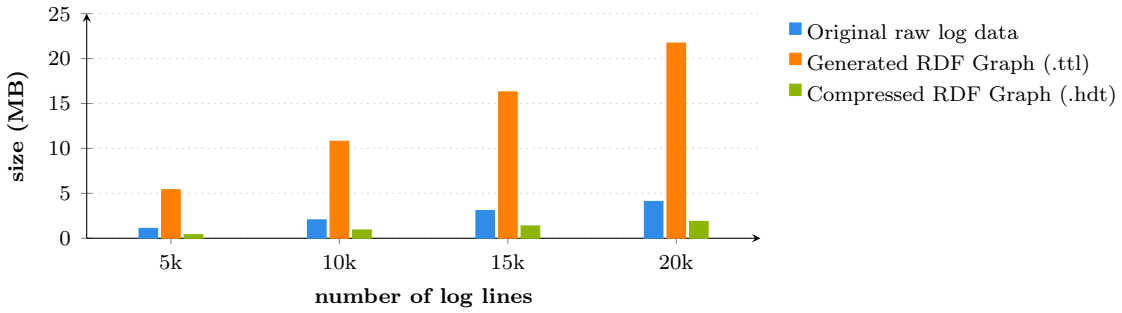

Figure 3.17.: Graph compression

3.2.6.3. Multi-host evaluation

To evaluate the scalability of our approach, we measure the log processing time for multiple hosts on the same network. This evaluation includes not only the log processing but also the query federation performance. Federation means that the queries are not only executed concurrently, but that they involve evaluating and combining individual query results to achieve integrated results.

Table 3.9 summarizes the evaluation setup that consists of six experiments ranging from 30 minutes up to 5 hours. The timeframe describes the starting time and the end time of analysis; log lines per host summarizes the range of log lines per host within the timeframe. For this evaluation, we used the Apache log dataset described in Table 3.8 and conducted the analysis within the log timeframe of March 2nd, 2020, starting from 8pm. Host 1 to host 4 store the data from the original 4 servers in the dataset (host 1 *mail.cup.com*, host 2 *mail.insect.com*, and so on); for the 3 additional hosts in the evaluation, we replicated the log files from *mail.cup.com*, *mail.insect.com*, and *mail.spiral.com*. Similar to the single-host evaluation, for each experiment, we reported the average times over five runs.

Table 3.9.: Multihost Experiment Timeframe

Experiment	Duration	Log lines per host	Experiment	Duration	Log lines per host
E1	30min	0.7k - 1k	E4	3h	3k - 5k
E2	1h	1k - 1.7k	E5	4h	6k - 8k
E3	2h	2.8k - 4k	E6	5h	8k - 10k

Figure 3.18 shows the average log processing times for each experiment. The 1 hour experiment shows that log processing for two hosts takes approx. 4.7 seconds on average. In the same experiment, the time slightly increases with an increasing number of hosts and reaches a max. of 7.5 seconds. The log processing time for the 5 hours experiment with two hosts takes approx. 19.01 seconds on average and reaches the max. average time of 26.10 seconds with 7 hosts. Based on these results, we conclude that the growth of the log processing time as a function of the number of hosts is moderate. Therefore, this approach scales well with a growing number of hosts to monitor, as the log processing on each host is parallelized and the query federation overhead is low.

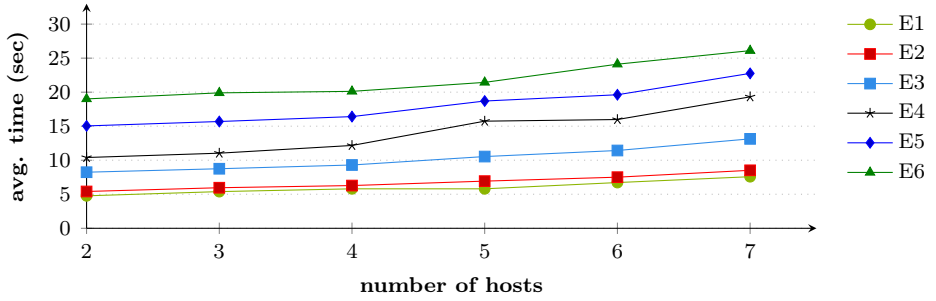

Figure 3.18.: Query execution time in a federated setting for different time frames

3.2.7. Discussion

In this section, we reflect upon benefits, limitations, and possible applications of the proposed virtual log knowledge graph framework.

Graph-based Log Integration and Analysis Representing log data in semantic graph structures opens up new possibilities, such as handling log data in a uniform representation, exploring connections between disparate entities, and applying graph-based queries to search for abstract or concrete patterns of log events. Compared to text-based search, graph-pattern based queries are more expressive and make it possible to link entities that appear in log lines to background knowledge. Furthermore, the ability to provide query results as a graph enables new workflows for analysts and may help them to be more efficient in exploring log data and ultimately improving their situational awareness faster.

In our evaluation, we find that SPARQL as a standardized RDF query language provides powerful means for graph pattern-based ad-hoc log analyses. A challenge, however, is that analysts are typically not familiar with the language and require some training. This may improve in the future, as SPARQL is often already part of computer science curricula and is increasingly being adopted in many industries[49]. Furthermore, intuitive general-purpose visual query building and exploration tools such as [125, 126] could be used and possibly adapted for security applications to abstract the complexity of writing queries directly in SPARQL.

Decentralization and Virtualization Decentralized ad-hoc extraction on the endpoints at query execution time is a particularly useful approach in scenarios where log acquisition, aggregation, and storage are difficult or impractical. This includes scenarios with a large number of distributed hosts and log sources. Pushing log analysis towards the endpoints is also particularly interesting in settings where bandwidth constraints do not permit continuous transmission of log streams to a central log archive.

Whereas these considerations apply generally, the decentralized approach also has benefits that are specific to our knowledge-graph based approach for log integration and analysis. Specifically, the federated execution distributes the computational load of extraction, transformation, and (partly) query execution towards the endpoints. This will be useful in many practical settings where the scale of the log data that is constantly generated in a distributed environment is prohibitively large and it is not feasible to

[49] cf. `http://sparql.club`

transform the complete log data into a KG presentation. In such settings, the decentralized approach facilitates ad-hoc graph-based analyses without the need to set up, configure and maintain sophisticated log aggregation systems.

Our evaluation showed that this ad-hoc extraction, transformation, and federated query execution works efficiently for temporally restricted queries over dispersed log data without prior aggregation and centralized storage. Consequently, the approach is particularly useful for iterative investigations over smaller subsets of distributed log data that start from initial indicators of interest. It supports diagnostics, root cause analyses etc. and can leverage semantic connections in the graph that would otherwise make manual exploration tedious. An inherent limitation, however, is that the computational costs become exceedingly large for queries without any temporal restrictions or property-based filters – i.e., the approach is less useful for large-scale exploratory queries over long time intervals without any initial starting point.

Log Parsing and Extraction The identification and mapping of relevant concepts in log messages is currently based on regular expression patterns. Extracted log lines are filtered and only lines that potentially match the query are transferred from the local endpoint, which minimizes bandwidth usage and processing load at the querying client. A limitation of this approach is that for complex queries, the execution of a large set of regular expression patterns on each log line raises scalability issues.

An approach based on templates, similar to [127], could be applied to learn the structure and content of common log messages and then only extract the expected elements from those log messages. Furthermore, repeated application of regular expression patterns on each log line could also be avoided by building a local index on each endpoint. Such techniques should improve query performance, but these improvements have to be traded off against the additional complexity and storage requirements they introduce.

Applications and Limitations The illustrative scenarios in Section 3.2.5 highlighted the applicability of the approach in web access log analysis, intrusion detection, network monitoring, and threat detection & ATT&CK linking.

In these settings, ad-hoc integration of dispersed heterogeneous log data and graph-based integration can be highly beneficial to connect isolated indicators. Moreover, we found that the virtual log knowledge graph is highly useful in diagnostic applications such as troubleshooting or service management more generally and we are currently working on a framework for instrumenting containers with virtual knowledge graph interfaces to support such scenarios.

In the security domain – the focus in this paper – we found that virtual knowledge graphs can complement existing log analytic tools in order to quickly gain visibility in response to security alerts or to support security analysts in threat hunting based on an initial set of indicators or hypotheses.

Key limitations, however, include that the virtual integration approach is not directly applicable for *(i)* repeated routine analyses over large amounts of log data, i.e., in scenarios where up-front materialization into a KG is feasible and amortizes due to repeated queries over the same large data set or *(ii)* continuous monitoring applications, i.e., scenarios where log data has to be processed in a streaming manner, particularly in the context of low latency requirements. The latter would require the extension of the approach to

streaming settings, which we plan to address in future work.

Evasion and Log Retention A typical motivation for shipping log data to dedicated central servers is to reduce the risk of undetected log tampering when hosts in the network are compromised. This reduces the attack surface, but makes securing the central log archive against tampering all the more critical. Relying on data extracted at the endpoints, by contrast, comes with the risk of local log tampering. File integrity features could help to spot manipulations of log files, but for auditing purposes, the proposed approach has to be complemented with secure log retention policies and mechanisms. Finally, the communication channel between the query processor in the analytic user interface and the local log parsers also represents an attack vector that has to be secured.

3.2.8. Conclusions

In this article, we presented , a novel approach for distributed ad-hoc log analysis. It extends the Virtual Knowledge Graph (VKG) concept and provides integrated access to (partly) unstructured log data. In particular, we proposed a federated method to dynamically extract, semantically lift and link named entities directly from raw log files. In contrast to traditional approaches, this method only transforms the information that is relevant for a given query, instead of processing all log data centrally in advance. Thereby, it avoids scalability issues associated with the central processing of large amounts of rarely accessed log data.

To explore the feasibility of this approach, we developed a prototype and demonstrated its application in three log analysis tasks in security analytics. These scenarios demonstrate federated queries over multiple log sources across different systems. Furthermore, they highlight the use of semantic concepts inside queries and the possibility of linking contextual information from background knowledge. We also conducted a performance evaluation which indicates that the total log processing time is primarily a function of the number of extracted (relevant) log lines and queried hosts, rather than the size of the raw log files. Our prototypical implementation of the approach provides scalability when facing larger log files and an increasing number of monitoring hosts.

Although this distributed ad-hoc querying has multiple advantages, we also discussed a number of limitations. First, log files are always parsed on demand in our prototype. By introducing a template-based approach to learn the structure of common log messages and by building an index on each endpoint to store the results of already parsed messages, query performance could be improved. Second, the knowledge-based ad-hoc analysis approach presented in this article is intended to complement, but does not replace traditional log processing techniques. Finally, while out of scope for the proof of concept implementation, the deployment of the concept in real environments requires traditional software security measures such as vulnerability testing, authentication, secure communication channels, etc.

In future work, we plan to improve the query analysis, e.g., to automatically select relevant target hosts based on the query and asset background knowledge. Furthermore, we will explore the ability to incrementally build larger knowledge graphs based on a series of consecutive queries in a step-by-step process. Finally, an interesting direction for research that would significantly extend the scope of potential use cases is a streaming mode that could execute continuous queries, e.g., for monitoring and alerting purposes.

We plan to investigate this aspect and integrate and evaluate stream processing engines in this context.

4. Knowledge Graph-based Threat Detection and Attack Reconstruction

In this chapter, we present our contributions that address RQ3, i.e., *"How to reconstruct sequences of attack actions from system log event information?"*. We introduce a novel approach for semantic threat detection and attack reconstruction based on standard data representations for log data. We introduce a modular framework that integrates a variety of threat detection and attack reconstruction techniques.

Kabul Kurniawan, Andreas Ekelhart, Elmar Kiesling, Gerald Quirchmayr, A Min Tjoa. Published as "KRYSTAL: Knowledge Graph-based Framework for Tactical Attack Discovery in Audit Data" in *Computers & Security Journal*, Vol 121. 2022 [34].

Abstract

Attack graph-based methods are a promising approach towards discovering attacks and various techniques have been proposed recently. A key limitation, however, is that approaches developed so far are monolithic in their architecture and heterogeneous in their internal models. The inflexible custom data models of existing prototypes and the implementation of rules in code rather than declarative languages on the one hand make it difficult to combine, extend, and reuse techniques, and on the other hand hinder reuse of security knowledge – including detection rules and threat intelligence. *KRYSTAL* tackles these challenges by providing a knowledge graph-based, modular framework for threat detection, attack graph and scenario reconstruction, and analysis based on RDF as a standard model for knowledge representation. This approach provides query options that facilitate contextualization over internal and external background knowledge, as well as the integration of multiple detection techniques, including tag propagation, attack signatures, and graph queries. We implemented our framework in an openly available prototype and demonstrate its applicability on multiple scenarios of the DARPA Transparent Computing dataset. Our evaluation shows that the combination of different threat detection techniques within our framework improved detection capabilities. Furthermore, we find that RDF provenance graphs are scalable and can efficiently support a variety of threat detection techniques.

4.1. Introduction

In the face of complex cyber attacks, it is crucial to not only detect attacks as early as possible, but also to understand their context and implications in order to choose an appropriate response strategy. To protect themselves, organizations typically rely on defenses such as IDSs. These systems are useful in that they can point to indicators of compromise and issues, but they also typically generate a large number of false positive

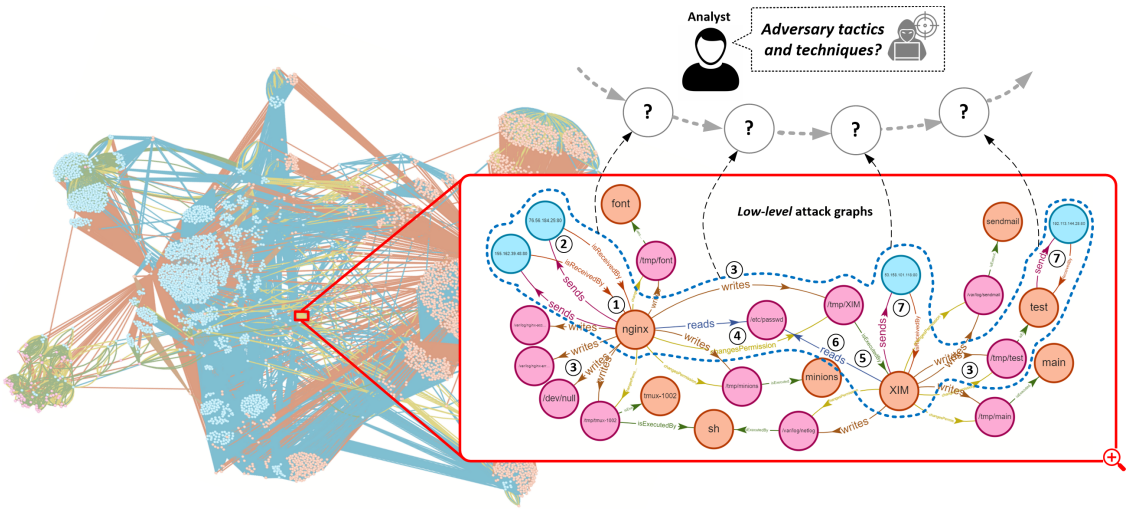

Figure 4.1.: **Motivating Example** - the enlarged section of the graph (the sub-graph inside the red line) depicts an attack graph. Specifically, the graph inside the blue dotted-line shows *low-level* event interaction as part of the attack pattern. While existing approaches can construct such an attack graph, it remains challenging for an analyst to recognize and understand the real attack steps without linking to *high-level* context (i.e., adversary tactics and techniques).

alerts. To identify relevant alerts, it is necessary to investigate their context, which typically requires substantial manual effort and expertise [128, 129]. As the sheer volume of low-level log data grows, manual analyses become increasingly infeasible.

In this context, system-level provenance graphs, which rely on audit data represented in graph structures, have recently attracted considerable attention in the security research community as a promising tool with strong abstract expression ability and relatively high efficiency [130]. Such provenance graphs represent relationships between the control flow and data flow between subjects (e.g., processes, threads) and objects (e.g., files, network sockets) in the system through a timestamped directed graph. Based on that, a large variety of techniques have been developed to automatically detect and connect attack steps in such graphs[1].

Motivating Example Figure 4.1 depicts the provenance graph of an exfiltration attack carried out in the context of the DARPA Transparent Computing program [131]. The depicted graph, which was generated by our prototype, captures running processes, file reads and writes, and sent packets on a host over a given period of time[2]. The enlarged section of the graph highlights an attack fragment in which an attacker exploits a vulnerable `nginx` web server through a malformed HTTP request. The attacker opens a shell connection to the victim's host via the vulnerable web server (①②); manages to write an executable file */tmp/XIM* (③④⑤); starts a process that reads sensitive information from */etc/passwd* (⑥), and finally exfiltrates data via HTTP to an external network (⑦). The automatically constructed provenance graph summarizes the attack scenario as a sequence of connected

[1] cf. [130] for a recent survey
[2] In this case 140 hours.

low level events, which provides a good starting point for security analyses.

Challenges Despite its potential, provenance graph-based analysis faces a number of challenges that currently make it difficult to apply them in practical settings.

Context and interpretability. The closed nature of existing attack graph-based approaches makes it difficult to relate provenance data to internal and external knowledge. This is crucial because context information is important when interpreting security alerts in order to identify rare occurrences of malicious patterns within an overwhelmingly large amount of activities that are benign.

The graphs typically generated by state of the art techniques connect *low-level* events that are not easily interpretable without additional context. In our motivating example, for instance, the sub-graph inside the blue dotted line shows the low-level events that constitute the sequence of attack (steps ① to ⑦). Without relating such granular low-level events to a *high-level* context, it remains difficult to identify the underlying attack tactics and techniques, interpret the events in a broader context, and *understand the attack*. This lack of abstraction from low-level event graphs to higher level techniques and tactics results in tedious attack investigations necessary to link low-level evidence to phases of complex multi-stage attacks.

Robust detection. Provenance graph-based attack discovery approaches have so far been developed as monolithic prototypes implementing a particular combination of techniques. Various prototypes have individually been shown to be effective in identifying and analyzing attacks, but robust detection still remains a key challenge [130].

Interoperability. Existing approaches have so far been developed in the context of tightly coupled research prototypes with proprietary internal data structures that are not interoperable. The respective implementations are typically not openly available, which on the one hand impedes the reproducibility of the findings, and on the other hand hinders integration and reuse.

Solution approach To tackle these challenges, we propose a knowledge graph-driven framework (*KRYSTAL*) that leverages Semantic Web technologies for audit log analysis and tactical attack discovery.

We hypothesize that more robust detection can be achieved in an integrated platform that makes it possible to combine heterogeneous approaches and techniques – which is currently difficult due to the lack of a uniform data model and a common technical foundation. Furthermore, we propose to disentangle data collection, data management, and threat detection – which are currently to a varying degree part of and specific to each approach – by means of a uniform, ontology-based target representation for provenance graphs. This abstracts from storage layer implementation details, facilitates separation of concerns, and provides interoperability between components on these layers. It also makes it possible to reuse and recombine collection, management, and threat detection modules.

To this end, our **contributions** in this paper are as follows: *(i)* We develop a standardized[3], shared and reusable conceptualization of log events, detection rules, and threat intelligence. Thereby, we unify and integrate log events from heterogeneous sources and enrich it with background knowledge; *(ii)* We introduce a modular threat detection and attack graph reconstruction framework that leverages this model and integrates multiple

[3] `https://www.w3.org/standards/`

state of the art techniques such as tag propagation, attenuation & decay, signature-based, and graph querying; *(iii)* We generate attack graphs that can be enriched and contextualized through linking to background knowledge (e.g., assets, vulnerabilities, cyberthreat intelligence, etc.); such contextualization can help analysts to identify and assess high-level attack scenarios in order to understand their significance, cause, and impact. *(iv)* We provide an open source prototype of the system[4] and evaluate it on a well-established large-scale dataset [131].

The remainder of this paper is structured as follows: Section 4.2 discusses related work in threat detection and attack graph construction, Section 4.3 puts forth a set of requirements as a basis of our proposed solution; Section 4.4 discusses the conceptualization of our approach; Section 4.5 introduces our knowledge graph-based attack detection framework, and Section 4.6 discusses the implementation and introduces application scenarios; we evaluate our approach in Section 4.7, discuss the results in Section 4.8 and conclude with an outlook on future research in Section 4.9.

4.2. Related Work

In this article we focus on misuse-based intrusion detection techniques, which search for well-defined patterns of attack [132]. Consequently, we organize related work on threat detection and attack graph construction into the following categories: *(i)* provenance-based tracking, *(ii)* tactical attack construction, *(iii)* graph queries, and *(iv)* ontology-based threat detection. Table 4.1 summarizes and compares existing attack discovery approaches, which will be discussed in the following sections.

Provenance-based Tracking Seminal work on provenance-based attack detection [133] introduced the idea to investigate attacks through backward tracking. A major limitation of initial "naive" backtracking approaches is that they are based on *coarse-grained* provenance data. This typically introduces a large number of false dependencies in the graph, a problem known as "dependence explosion" [134]. Several researchers introduced approaches to mitigate this problem through *fined-grained taint-tracking* or *information flow tracking* [135, 136, 137]. Although these approaches can accurately distinguish suspicious and benign nodes, scaling them remains a challenge [134]. Another approach tried to solve this problem by introducing *tag-propagation*. SLEUTH [128], for instance, introduced *trustworthiness tags (t-tags)* and *confidentiality tags (c-tags)* that are used to assign suspicion levels to nodes and propagate them through the provenance graph. In combination with a policy framework, these tags can trigger alarms and successfully identify unseen attacks in real-time and with low overhead. However, SLEUTH suffers from numerous false-positives for attacks with long-running processes [138]. MORSE [138] extends

this approach by introducing *tag-attenuation* and *tag-decay* to cut down false alarms by more than an order of magnitude. To reduce the provenance graph, it indexes all subjects and objects using a numeric index. However, none of these approaches leverage standard representations, but rely on custom graph models and hard-coded rules and policies. Consequently, they are difficult to expand and it is difficult to investigate the resulting attack graphs further, e.g., by linking them to background knowledge. Furthermore, both SLEUTH and MORSE do not explicitly represent the high-level context of attacks. As a

[4] `https://github.com/sepses/Krystal`

Table 4.1.: **Comparison of attack discovery approaches**. (*DS*) refers to domain-specific, (*N/A*) refers to not available, (✓) refers to provided, (✗) refers to not provided.

Approaches	Data Models	Detection Techniques	Data Reduction Techniques	Attack Reconstruction	High-Level Attack Summarization	Background Knowledge Linking	Published Prototype
Backtracking [133]	Custom Graph	Naive Backtracking	✗	✓	✗	✗	N/A
AIQL [139]	Relational DB	DS Query Language	✗	✗	✗	✗	N/A
SAQL [140]	Relational DB	DS Stream Query	✗	✗	✗	✗	N/A
POIROT [141]	Custom Graph	Graph Alignment	✗	✓	✗	✗	N/A
HOLMES [129]	Custom Graph	HSG/Severity Score	✗	✓	✓	✗	N/A
RapSheet [142]	Custom Graph	EDR/Threat Score	CPR[143]	✓	✓	✗	N/A
Zou, et al. [144]	Custom Graph	Pre/Post Condition	✗	✗	✓	✗	N/A
SLEUTH [128]	Custom Graph	Tag Propagation	✗	✓	✗	✗	N/A
MORSE [138]	Custom Graph	Tag Propagation	CSR[134]	✓	✗	✗	N/A
CyGraph [114]	NoSQL Graph	DS Query	✗	✓	✗	✓	✓
UCO [81]	RDF Graph	SPARQL Query	✗	✗	✗	✓	✓
Ekelhart, et al. [27]	RDF Graph	SPARQL Query	✗	✗	✗	✓	✓
Kurniawan, et al. [30]	RDF Graph	SPARQL Query	✗	✓	✗	✓	✓
SLOGERT [145]	RDF Graph	SPARQL Query	HDT[123]	✗	✗	✓	✓
KRYSTAL	RDF Graph	Hybrid	HDT[123]	✓	✓	✓	✓

result, it becomes exceedingly difficult to identify attacks as attack graphs become more complex.

Our approach introduces a standard and flexible graph model based on an ontology. Hence, with a standard graph model and declarative rules and policies, our approach can produce compact attack graphs and also flexibly integrate and link them to background knowledge. For graph reduction, we use (Header, Dictionary, Triples) HDT[123], a compact RDF-based structure that keeps large provenance graph compressed. We discuss it in more detail in Section 4.5.1.

Tactical Attack Construction Provenance-based tracking typically relies on a bottom-up approach, i.e., identifying attacks based on the causality relationship of system objects, such as processes, files, and sockets. By contrast, several approaches follow a top-down strategy, i.e., they aim to identify attacks based on high-level models of APT campaigns (e.g., techniques, tactics) and/or kill-chain phases. HOLMES [129] introduced threat scores and 16 techniques, tactics, procedures (TTP) proposed by the authors to construct a *high-level scenario graph (HSG)* from a provenance graph. This approach can mitigate dependence explosion and offers a map to TTP. However, its ability to identify attacks with only a single APT stage is limited [138]. RapSheet [142] is an attack graph construction approach based on alert correlation from a commercial Endpoint Detection and Response (EDR) tool. By incorporating 67 EDR rules, it can relate alerts into a *tactical provenance graph* based on MITRE's ATT&CK TTPs. For graph reduction, it implements *causality-preserved reduction* (CPR) [143] that merges edges with identical operations and keeps only the edge with the latest timestamp. Our approach also facilitates TTP mappings, but we rather incorporate rules from open, standard and community-driven detection rules (i.e., SIGMA [146]), thus avoiding a dependence on a commercial platform.

Another recently proposed approach [144] uses tactic-centric Advanced Persistent Threat (APT) recognition to detect APT tactics based on APT techniques' prerequisites (i.e.,

requirements for techniques to be matched to a tactic) and post-conditions (i.e., the result of a technique, e.g., malicious process being created). The identified attack techniques are mapped to specific tactics and ranked based on their tactic matching. This approach shows the detected tactics and techniques, but it does not represent the complete attack scenario, i.e., attack sequences, causality and connections.

In our work, we propose a hybrid approach that facilitates both bottom-up and top-down techniques for threat detection and attack construction in a single, modular framework. Our approach generates detailed attack graphs from low-level events but also facilitates linking and contextualization to existing high-level, tactical attack patterns such as MITRE ATT&CK TTPs.

Graph Queries A number of research efforts resulted in query-based and graph-matching approaches to detect and construct attack scenario graphs from historical audit log data stored in databases. AIQL [139] introduced a domain-specific model and query language to analyze and investigate attacks. SAQL [140] extends AIQL for stream-based querying over system monitoring data. POIROT [141] proposed an attack graph detection approach based on manually extracted graph patterns from previously seen attacks, e.g., in threat intelligence reports. In our approach, we rely on a semantic model, which facilitates graph matching through semantic graph queries (formulated in SPARQL). Furthermore, our RDF-based provenance graph can be easily queried and linked to internal and external background knowledge through SPARQL query federation.

Ontology-based Threat Detection A number of research efforts investigated ontologies to support cybersecurity and threat detection. Early work [147] developed an ontology for the intrusion detection domain based on DAML+OIL [148] to extend simple IDS taxonomies with machine-interpretable definitions. Ref. [149] extended this IDS Ontology to incorporate cybersecurity-related information from heterogeneous resource (e.g., web texts, reports).

UCO [81] introduced a more instance-data driven approach to construct a rich cybersecurity ontology by integrating cybersecurity standards such as STIX[150], CyBox[151], CVE[42], CAPEC[152], CCE[52], and CVSS[153]. UCO provides integration from heterogeneous sources and supports reasoning (e.g. using predefined rules to infer attacks). However, UCO and other previously proposed ontologies were designed to support intrusion detection rather than attack graph construction. More recent works such as [154] proposed an ontology that supports IDS alert correlation. Through the use of reasoning rules, it can infer and classify IDS alerts (i.e., false alert, unclassified, plausible attack) based on existing vulnerability information. As the focus is solely on attack detection, however, this proposed approach does not result in attack graphs.

In prior work, we proposed a modular ontology [27] that can harmonize and integrate heterogeneous log sources (e.g., Syslog, Auth log, Apache log). Subsequent work [28] facilitates forensic analyses of arbitrary logs and contextualizes them with external background knowledge (e.g., CPE, CVE, CVSS, CWE, and CAPEC) in a federated settings [155]. SLOGERT [145] extends this previous work with the ability to construct knowledge graphs automatically from arbitrary raw log messages. It identifies, links, and enriches entities in log sources with background knowledge. In this work, we extend and enhance our previous work on security ontologies to integrate events from heterogeneous log sources, represent

them in attack scenario graphs, and link them to background knowledge.

4.3. Requirements

Based on our analysis of the limitations of the state of the art (cf. Section 4.2), we identify the following set of requirements for the construction of a modular framework for knowledge graph-driven tactical attack discovery.

R1. Contextualization Investigating and prioritizing security alerts, which typically include a large number of false positives, requires extensive security domain knowledge to understand the context of an alert [156]. This makes it necessary to contextualize information in the provenance graph with appropriate background knowledge, which necessitates a flexible data model. As an example, important contextual information on the system under evaluation includes installed software and patch versions of the host where an alert has been raised. Existing approaches (e.g., [141, 138]) typically hard-code some context information (e.g., external networks and software), whereas the majority currently ignores them altogether. To overcome these limitations, the ability to contextualize provenance graphs with background knowledge on the system under investigation is a key requirement for our framework.

R2. Reusability and Extensibility Reusability – e.g., of attack patterns and resulting graphs – and extensibility – e.g., of detection, reconstruction, and summarization techniques – have not been design priorities in existing monolithic solutions for provenance based log analysis. Consequently, approaches each rely on their own data structures developed specifically for each approach, While this allows for some optimization, it makes it difficult to reuse and extend methods, techniques, information, and results. The lack of unified data formats as a (more specific) aspect of this requirement has also been highlighted as a major limitation of the state of the art in a recent survey [130]. To tackle this limitation, the developed framework should – while offering adequate performance – make it possible to: *(i)* formulate and exchange rules for detection, alerting, and attack graph reconstruction in a declarative language, *(ii)* exchange instances of provenance graphs in a standard representation, *(iii)* query the provenance graphs in a standard language, *(iv)* incorporate and exchange threat information, and *(v)* reuse and combine analytic techniques.

R3. Threat intelligence linking Connecting isolated events to reconstruct a complete attack scenario is an important step to understand the relevance and impact of alerts. In this context, an ability to link low-level threat evidence to phases of complex multi-stage attacks would be highly beneficial. While some existing approaches [129, 142, 144] aim to identify steps and associate them with high-level APT phases, they do not take advantage of the abundance of available Cybersecurity Threat Intelligence (CTI) information available in external sources such as MITRE ATT&CK [157], which offers links to CVE [42], CWE [158], and others. We therefore define the ability to leverage such connections and provide integrated querying capabilities as a key requirement for the developed approach. This will facilitates abstraction from low-level event graphs to higher level techniques and tactics, provide additional information for prioritization, impact assessment and mitigation, and make tedious attack investigations more efficient and effective.

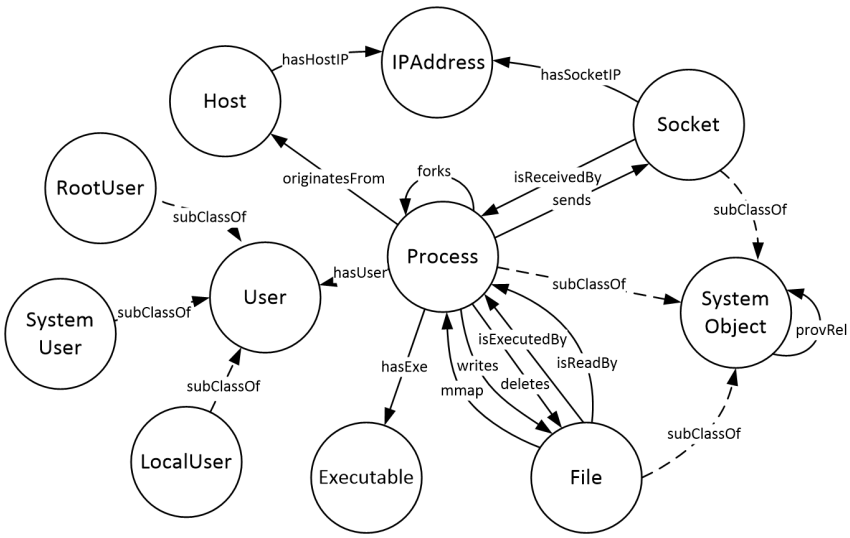

Figure 4.2.: **Krystal Ontology** - nodes represent concepts (classes) and edges represent the class relationship (properties).

R4. Cross-platform Interoperability Integrating provenance data from heterogeneous sources and constructing integrated attack graphs spanning multiple systems would be highly beneficial in the face of complex multi-host/multi-platform attacks [130]. This necessitates a unified provenance representation that can abstract from specifics of individual platforms.

4.4. Conceptualization

In this section, we describe the conceptualization of our approach and the use of inference for provenance graph construction.

4.4.1. System-Call Provenance Representation

Provenance graphs are a highly effective representation to keep track of information flows [130]. They represent interactions between objects as events. Each event involves a *subject*, i.e., a system object (e.g., a process, thread) that performs a particular *operation* (e.g., write, read, send) on an *object* (e.g., file, socket, registry). As objects typically appear in multiple events, a graph emerges. Although system-call provenance is represented similarly across different existing approaches [128, 138, 142, 129], no formal and standardized representation (ontology) exists. Instead, existing approaches typically use ad-hoc and hard-coded data models which limits their interoperability, reusability and extensibility.

4.4.2. *KRYSTAL* Provenance Ontology

We propose the *KRYSTAL* ontology[5] as a standard representation for system-call provenance graphs. Key benefits of using ontologies is the ability to share a common understanding – including concepts and structure, making assumptions explicit, facilitating semantic

[5] `https://w3id.org/sepses/vocab/event/log/`

reasoning, and promoting concept reuse [159]. Furthermore, the use of an ontology for provenance graph representation makes it easy to integrate background knowledge and unify the conceptual model across different threat detection approaches.

In developing the *KRYSTAL* ontology, we followed a *hybrid* approach, i.e., using *bottom-up* and *top-down* approaches concurrently [159]. Bottom-up, we started by analysing low-level data structures from applications (e.g., auditlog) and identifying their entities and relations. Top-down, we considered attack patterns from existing cyberthreat intelligence sources (e.g., MITRE ATT&CK [157]) and compared our ontology to existing non-ontological provenance models/schemas. Our developed ontology is able to represent low-level system-call provenance data and link it to high-level attack patterns (cf. Section 4.5.4).

Figure 4.2 depicts an excerpt of the *KRYSTAL* Ontology centered around core concepts such as `User`, `Host`, `System Object`. The latter represent system entities such as `Process`es, `File`s, and `Socket`s. We assign each system object *so* to a class (identified by `rdfs:Class`) and potential sub-classes (identified by `rdfs:subClassOf`). Each system object *so* is a vertex v and the relations between system objects (e.g., `writes`, `isReadBy`, `sends`, etc.) are represented via edges e (identified by `owl:ObjectProperty`).

The SystemObject class represent system entities in general. It has three sub-classes such as Process, File and Socket. The Process class represents a running process in a system (e.g., firefox, ssh, etc.) while the File class represents a file (e.g., system file, application file, etc.). We introduce `kry:writes`, `kry:isReadBy` and `kry:isExecutedBy`, `kry:deletes`, `kry:mmap` properties to represent links between Process class and File class. The Socket class represents a network connection (combination of an `IPAddress` and `Port`). We defined `kry:sends` and `kry:isReceivedBy` properties that link the Process class to the Socket class.

We also defined User, a class that describes user of a system. It has three sub-classes, i.e., *(i)* RootUser, that represents the root-level user, *(ii)* SystemUser, represents the system-level user, *(iii)* LocalUser, represents the local-level user. We also defined Host class that represent a host machine. The `kry:hasUser` property connects Process class to the User class, while the `kry:originatesFrom` property links it to Host class. Finally, we also defined several other classes such as Executable class that identifies an executable file and IPAddress that represents an IP Address. The `kry:hasExe` property connects Process to Executable and `kry:hasHostIP` that links Process to IPAddress.

Furthermore, we introduce a *provenance relation* (identified by `kry:provRel`) that self-links to SystemObject. It is a generic, upper-level relation property that is used to represent provenance relationships between system objects (e.g., socket to process, process to file, file to process). This provenance relation is automatically inferred from more specific relation properties, as explained in the following.

4.4.3. Inference Features in *KRYSTAL* Ontology

The *KRYSTAL* ontology uses the RDF-S entailment and OWL reasoning rules summarized in Table 4.2 for provenance graph construction.

rdfs2 & rdfs3 These rules are used to automatically infer the specific type of a given system object based on its relations. For example, as depicted in Section 4.4.3, system object

Table 4.2.: RDFS & OWL Reasoning Rules

Rule	Premise	Conclusion
rdfs2	$e_1(v_1, v_2)$, `rdfs:domain`(e_1, X)	$v_1 \in X$
rdfs3	$e_1(v_1, v_2)$, `rdfs:range`(e_1, Y)	$v_2 \in Y$
rdfs7	$e_1(v_1, v_2)$, `rdfs:subPropertyOf`(e_1, e_2)	$e_2(v_1, v_2)$
owl: InverseOf	$e_1(v_1, v_2)$, `owl:inverseOf`(e_1, e_2)	$e_2(v_2, v_1)$

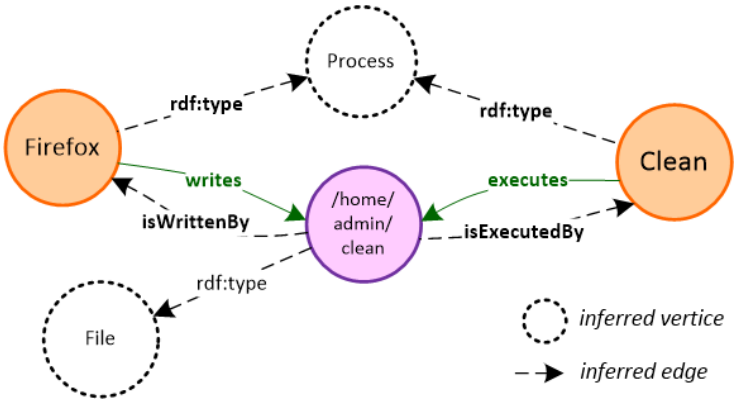

Figure 4.3.: RDFS & OWL reasoning for vertice/edge inference.

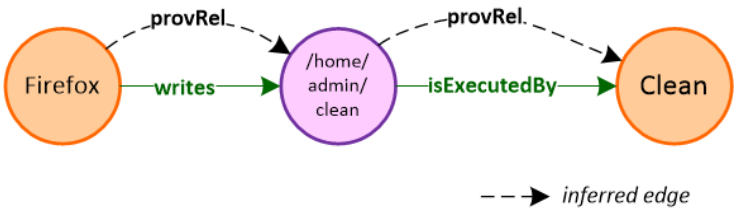

Figure 4.4.: **Rdfs7** inference example.

`:Firefox`[6] has the relation `kry:writes` to another system object `:/home/admin/clean`, since in the ontology we define `kry:writes` as `owl:ObjectProperty` with an `rdfs:domain` of `kry:Process`. Consequently, based on **rdfs2** rule definition, it will be automatically inferred that `:Firefox` $\in$ `kry:Process`. The system object `:/home/admin/clean` is another example. Since `kry:writes` has `rdfs:range` of `kry:File`, based on the **rdfs3** rule definition, it will be deduced that `:/home/admin/clean` $\in$ `kry:File`. This class type inference is useful for, e.g., formulating semantic graph queries to detect attack patterns based on the system object type and their relations. We discuss the query processes in more detail in Section 4.5.

rdfs7 This rule is used to generalize relations based on property hierarchies identified by `rdfs:subPropertyOf`. In the example in Figure 4.4, for instance, `:Firefox` has the relation `kry:writes` to `:/home/admin/clean`, and in the ontology, `kry:writes` is a sub-property of the more generic property `kry:provRel`, which represents provenance relationships. Based on the **rdfs7** rule, `kry:provRel` will be automatically inferred between objects that have a more specific relation; this creates provenance links for all system object relationships (e.g., from the `:Firefox` process to the `:/home/admin/clean` file to the `:Clean` process etc.). This is helpful to identify information flows, track causality of events, and reconstruct attack graph. We discuss the advantage of having explicit provenance relations (e.g., during backward-forward analysis for attack graph reconstruction) further in Section 4.5.3.

owl:InverseOf Finally, a reasoning technique that we use in our ontology is property inversion, identified by **owl:InverseOf**, which creates relations between system objects in both directions. For instance, as depicted in Section 4.4.3, `:Firefox` has a relation `kry:writes` to `:/home/admin/clean` and in the ontology, `kry:writes` is defined as `owl:inverseOf` to `kry:isWrittenBy`. Based on the **owl:inverseOf** rule, it will be automatically inferred that `:/home/admin/clean` also has a relation `kry:isWrittenBy` to `:Firefox`. The same holds for property `kry:executes`, since it has the relation `owl:inverseOf` to `kry:isExecutedBy` in the ontology.

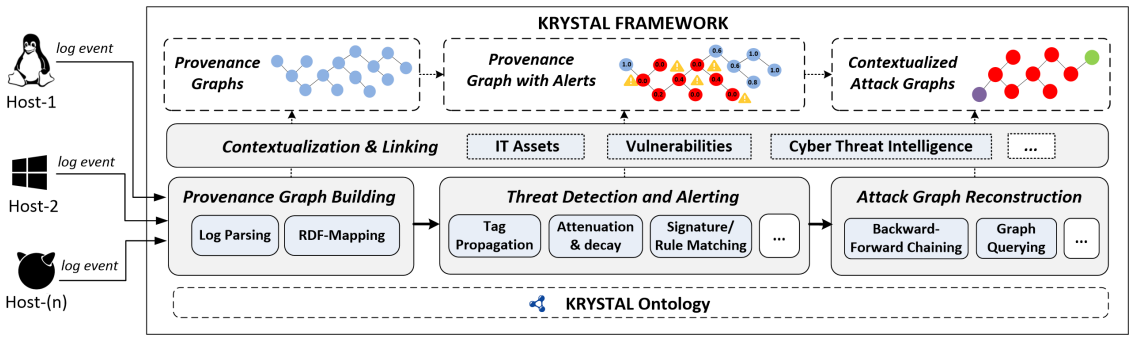

Figure 4.5.: Krystal Framework Architecture

4.5. Solution Architecture

In this section we present *KRYSTAL*, a modular framework for tactical attack discovery in audit data. The proposed framework integrates a variety of attack discovery mechanisms and takes advantage of its semantic model to include internal and external knowledge in the analysis. Figure 4.5 gives an overview of the KRYSTAL attack discovery framework which consists of three main parts, i.e., *(i)* provenance graph building, *(ii)* threat detection and alerting, and *(iii)* attack graph and scenario reconstruction.

Security analysis is typically conducted either *online* or *offline*. Online refers to an analysis performed in a running system in (near) real-time, whereas offline refers to analysis over collected data. These two modes have a different purpose (monitoring vs forensics), but they can complement each other in hybrid settings [160].

KRYSTAL works in an online mode in that it imports each log event in sequence from potentially heterogeneous hosts (e.g., Linux, Windows, FreeBSD). The *Provenance Graph Building module* then generates an RDF-based provenance graph, taking advantage of the well-defined ontology and enabling enrichment with background knowledge. Subsequently, the *Threat Detection & Alerting* module allows for the combination of various approaches on the uniform KG. We illustrate the generality of the approach by implementing a set of common mechanisms as SPARQL queries, combining *(i)* tag propagation, *(ii)* attenuation & decay, and *(iii)* signature-based detection based on Indicators of Compromise (IoCs). The *Attack Graph Reconstruction* module then facilitates (offline) attack graph generation via *Backward-forward* chaining and attack pattern matching via *Graph Querying* over the provenance graph. We explain each component in more detail in the following subsections.

4.5.1. Provenance Graph Building

This component consists of two sub-components, *Log Parsing* and *RDF Mapping*. *Log Parsing* transforms raw log events from heterogeneous hosts and operating systems (e.g., `auditd` from Linux, `ETW` from Windows, and `dtrace` from FreeBSD) into a structured format (i.e., JSON). This component selects and parses important information from the log event such as system entities (e.g., processes, files, sockets, etc.) and their relations (e.g.,

[5] We represent objects with different colors: orange for processes, purple for files, blue for sockets, and transparent-dotted circles for inferred system object types.

[6] *Note that we define prefix `":"` for an instance and `"kry:"` for Krystal ontology. We omit it in figures for simplicity.*

read, write, execute, etc). *RDF-Mapping* maps the structured log data into an RDF graph representation. Specifically, we used the *KRYSTAL* ontology described in Section 4.4.2 and RML[7], a declarative RDF-mapping language, to map the parsed audit data and transform them into well-defined, RDF-based provenance graphs.

```
@prefix : <http://w3id.org/sepses/resource#> .
@prefix kry: <http://w3id.org/sepses/vocab/kystal#> .

:Firefox kry:writes :/home/admin/clean ;
    kry:cmdLine "/usr/bin/firefox" ;
    kry:hasUser :user-2 ;
...
```

Listing 4.1: Excerpt of RDF-based provenance graph generated from a log event

Section 4.5.1 shows an excerpt of a log event in RDF representation. It captures the fact that a system object (Firefox process) created a file ("/home/admin/clean"). In RDF, the Firefox process is modeled as *subject* `:Firefox`, the "write" operation is represented as *property* `kry:writes` and the file "/home/admin/clean" is connected as *object* `:/home/admin/clean`.

Taking advantage of reasoning rules (cf. Section 4.4.3), Section 4.5.1 shows the inferred RDF triples from the provenance graph. `:Firefox` has been identified as a `kry:Process` and `kry:SystemObject`. In addition, the provenance relation *property* `:provRel` with the *object* `:/home/admin/clean` has been added automatically, and the type of `:/home/admin/clean` has been inferred as `kry:File`.

```
@prefix : <http://w3id.org/sepses/resource#> .
@prefix kry: <http://w3id.org/sepses/vocab/kystal#> .
...
:Firefox a kry:Process, kry:SystemObject;
    kry:provRel :/home/admin/clean.

:/home/admin/clean a kry:File, kry:SystemObject.
...
```

Listing 4.2: Excerpt of inferred RDF triples

Graph Reduction & Compression We apply two strategies to reduce the graph size. First, we automatically merge the duplicated RDF output identified by the same URI [161], thus eliminating redundant events (events with the same *subject*, *property* and *object*). Furthermore, we skipped irrelevant events – i.e., events that are not considered in our ontology – from being processed. Second, we use HDT [123] as a graph compression technique that provides a compact data structure and binary serialization format for RDF. HDT keeps large graph data compressed and manageable while enabling query operations without prior decompression. We discuss our graph reduction and compression results in Section 4.7.

4.5.2. Threat Detection & Alerting

Due to the uniform KG representation, the *KRYSTAL* framework allows to combine and integrate a variety of techniques for threat detection and alerting. We illustrate this

[7] `https://rml.io/`

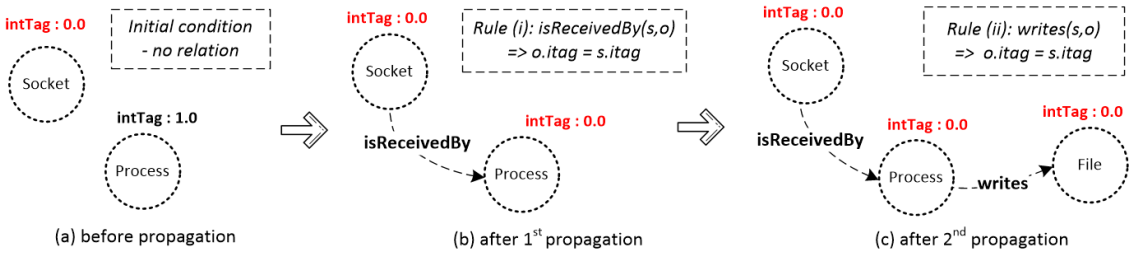

Figure 4.6.: Tag Propagation Example

by *(i)* transforming and integrating established threat detection and alerting techniques *(ii)* incorporating a signature-based threat detection approach through the transformation of curated public rules (e.g., Sigma Rule), and *(iii)* link the identified attack pattern into high-level attack technique and tactic (TTPs Mapping) . Rather than hard-coding these mechanisms and developing them for a custom data structure, we show in the following how they can be implemented as standard declarative SPARQL queries.

Tag Propagation is a prominent method to establish event causality in the context of provenance graphs. It is based on a set of rules to assign tag values onto system object nodes (i.e., process, file, socket, etc.) upon interaction between them [128]. In order to trace the impact of malicious events, tags and values will be propagated sequentially through a provenance graph to other system objects if a given *tag propagation* rule is satisfied.

Figure 4.6 illustrates *tag propagation* with an example. It introduces *integrity* and *confidentiality* tags to identify the suspicion level of a node. We express propagation rules [138] as SPARQL queries processed by our *tag propagation* component. Section 4.5.2 shows the query matching the illustration in Figure 4.6. It checks if a socket connection (`<socket>`) is received by (`:isReceivedBy`) a process (`<process>`) and then compares the minimum (`afn:min`) integrity tag value (`:intTag`) of the `<socket>` (identified by `?oit` variable) and `<process>` (identified by `?sit` variable). [8] If `?oit` is not the same as `?sit`, it updates[9] `?sit` with the new value (identified by the `?nit` variable).

```
PREFIX : <http://w3id.org/sepses/vocab/krystal#>
PREFIX afn: <http://jena.apache.org/ARQ/function#>

DELETE { <process> :intTag ?sit }
INSERT { <process> :intTag ?nit }
 WHERE { <socket> :intTag ?oit;
                  :isReceivedBy <process>.
         <process> :intTag  ?sit.
         FILTER (?oit != ?sit).
         BIND (afn:min(?oit,?sit) AS ?nit). }
```

Listing 4.3: Propagation rule for incoming socket connection

[8] System objects with integrity scores in the interval $[0.0 - 0.5|$ are considered suspicious and scores in the interval $[0.5 - 1.0]$ are considered benign.

[9] SPARQL uses `DELETE` and `INSERT` to perform update operations on triple(s); cf. `https://www.w3.org/TR/sparql11-update/`

Attenuation & Decay are techniques to tackle the "dependence explosion" problem [128], which occurs when a node in the provenance graph interacts with a large number of system objects, causing a large number of benign events to be flagged as being part of an attack [138]. This leads to a large number of false-positive alerts, making it difficult to identify relevant alerts. *Tag attenuation* [138] aims to alleviate this issue by considering objects imperfect intermediaries for propagating malicious behavior through a benign process. In the previous example (cf. Figure 4.6), for instance, the low integrity process p_1 writes a file f_1, lowering its integrity. To avoid excessive propagation to other objects from there, the integrity and confidentiality tags of a benign subject get attenuated before they propagate to another benign object. This can be achieved by applying an additive factor *af* to the original tag value.

Tag decay is based on the assumption that in case a benign subject gets compromised and becomes suspicious, it will do so soon after consuming a suspicious input (e.g., read low integrity file) [138]. Consequently, this technique gradually lifts the score of the low integrity subject and limits benign objects from being propagated and flagged as suspicious, particularly for long-running processes.

```
PREFIX : <http://w3id.org/sepses/vocab/krystal#>
PREFIX afn: <http://jena.apache.org/ARQ/function#>

DELETE {<file> :confTag ?oct}
INSERT {<file> :confTag ?noct}
WHERE  {<file> :confTag  ?oct.
        <process> :writes <file>; :confTag  ?sct.
        FILTER (?oct != ?sct).
        FILTER (?oct != ?noct).
        BIND (?sct + 0.2 AS ?nsct).
        BIND (afn:min(?oct,?nsct) AS ?noct). }
```

Listing 4.4: Attenuation rule for a benign write propagation

Section 4.5.2 illustrates how the *KRYSTAL* framework enables declarative definitions of such rules using SPARQL queries. It shows an example of an attenuation rule for a benign write propagation that checks for a `<process>` that `:writes` a confidential `<file>`. The rule gradually increments the confidentiality tag (`:confTag`) value (`?sct`) of processes.

Provenance-Based Alerting Provenance-based alerting policies detect attacks based on the simultaneous fulfillment of several conditions in the provenance graph. These conditions (cf. [138]) take data integrity tags as well as information associated with nodes (e.g., permissions, users) into account. For example, a suspicious file execution can be detected under the following alert policy:

- a file f is executed by a process p,
- f has a low integrity tag value ($<$0.5) and p is benign (integrity $>$=0.5).

```
PREFIX rule: <http://w3id.org/sepses/vocab/rule#>
PREFIX : <http://w3id.org/sepses/vocab/krystal#>
CONSTRUCT {
       << ?file :isExecutedBy ?process >>
          rule:hasDetectedRule rule:execRule }
 WHERE { ?file :isExecutedBy ?process.
```

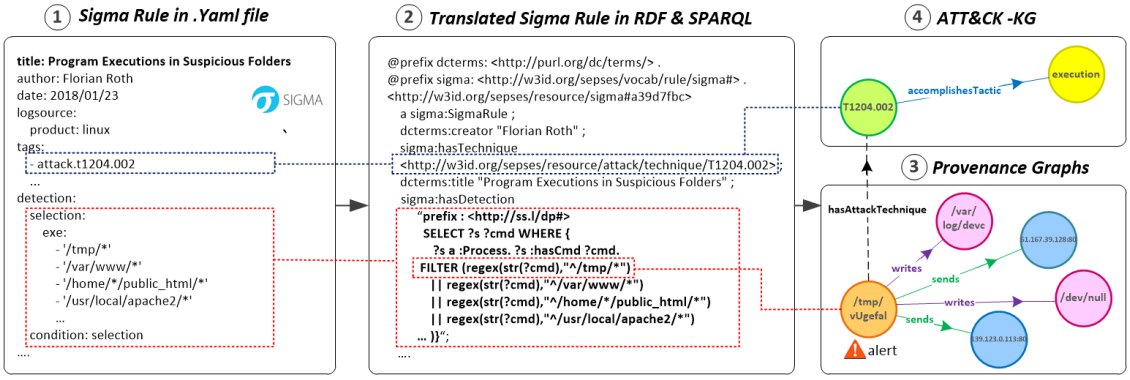

Figure 4.7.: Signature/rule-based threat detection example using the translated *Sigma*-rule query.

```
7           ?file rule:intTag  ?oit.
8           ?process rule:subjTag ?sst.
9           FILTER (?oit < 0.5)
10          FILTER (?sst >= 0.5)}
```

Listing 4.5: Alerting policy represented as SPARQL Query

The *KRYSTAL* framework facilitates provenance-based alerting by expressing alert policies as SPARQL queries and matching them against the provenance log graph. Section 4.5.2 provides an example of an alerting policy for suspicious file execution. The query matches a file (`?file`) with a low integrity value that has been executed by a benign process. Specifically, files with a tag value below 0.5 are considered low integrity and the respective triple `?file :isExecutedBy ?process`) will be marked with the detected rule[10].

Signature/Rule-based Threat Detection We complement the provenance-based mechanisms with a signature/rule-based detection approach that utilizes IoC definitions to identify known attacks in log events. This illustrates that the uniform KG representation facilitates the combination of a variety of approaches by using a common declarative query language.

Signatures are an established and effective approach in detecting known attack patterns, but maintaining the set of rules and signatures can be labour-intensive [130]. To tackle this problem, *KRYSTAL* leverages *Sigma*[11] – an open, shareable, community-driven and generic rule format for threat detection in logs.

Figure 4.7 part (1) shows an example of signature/rule-based threat detection defined in *Sigma*[12]. Each *Sigma*-rule is written in YAML[13] and defines the detection rule and its metadata (*title, id, date, author, log-source* etc.). A key benefit of these rules is that the *tags* metadata links rules to specific TTPs from MITRE ATT&CK. For example, *attack.t1204.002* corresponds to the technique *T1204.002*[14] (*User Execution: Malicious*

[10] We used the SPARQL `CONSTRUCT` syntax to generate new triples and link the detected alert to the respective rule.
[11] `https://github.com/SigmaHQ/sigma`
[12] `https://github.com/SigmaHQ/sigma`
[13] `https://en.wikipedia.org/wiki/YAML`
[14] `https://attack.mitre.org/techniques/T1204/003/`

File). As per November 2021[15], *Sigma* contains more than 1497 rules/signatures for different log sources (e.g., application, network, web logs) and platforms (e.g., Linux & Windows).

Our framework translates these *Sigma* rule specifications[16] automatically and transforms them into SPARQL query expressions. Specifically, we identified two search-identifiers under the *detection* attribute: *(i) lists* that contain strings applied to the full log message and joined with a logical 'OR', and *(ii) maps* that consist of key/value pairs, where key is a field in the log data and value is a string or integer. Lists of maps are joined with a logical "OR" while all elements of a map are joined with a logical "AND". We express *lists* as SPARQL filters with regex matching, while *maps* are represented as triple patterns, i.e. *Subject (S), Predicate (P), and Object (O). S* is a log object, *P* is a log property, i.e., key/field in the log data and *O* is the value. Similar to *lists*, we express the key/value pair matching using regex filters in SPARQL. Furthermore, we also map and transform the rule metadata into RDF. Figure 4.7 part (2) shows an example translation of a *Sigma* rule into SPARQL and RDF.

The translated *Sigma* rules are executed against the provenance graph to detect potential attacks/threats in the *Threat Detection and Alerting* module of our framework. As we can see in Figure 4.7 part (3), we detect an alert called *"Program Executions in Suspicious Folder"* in the provenance graph since the execution of the */tmp/vUgefal* process is located in the */tmp/* folder as defined in the *Sigma* rule. Subsequently, the generated alert will be linked automatically to the T1204.002 (*User Execution: Malicious File*) technique from the ATT&CK knowledge graph [31] in the background knowledge (cf. Figure 4.7 part (4)). We explain this linking mechanism further in Section 4.5.3.

4.5.3. Attack Graph Reconstruction

The next important step once the provenance graph has been constructed and alerts have been raised is to understand how the alerts are connected and to reconstruct potential attack steps. To this end, we construct an attack graph and the attack scenario out of the provenance graph through *(i) Backward-Forward Chaining* and *(ii) Graph Querying*.

Backward-Forward Chaining Backward-Forward Chaining is used to first identify the potential root cause of an attack and then reconstruct the overall attack steps. As provenance-based alerting may produce a lot of alerts, we need to prioritize them and identify potential root cause alerts of an attack. This can be done by assigning an alert score during *backward searching*, i.e., incrementing alert scores of each predecessor alert on a path.

For this search, we leverage *Property Paths*[17], a SPARQL feature that allows us to find routes between nodes in the RDF provenance graph. Recall that we used **rdfs7** inference to automatically generate an upper-level relation `:provRel` between system objects.

As shown in Section 4.5.3, the backward search query consists of a triple pattern [`?currentAlert ^:provRel* ?s`], in which `^..*` represents the *property path* that finds all possible backward connections from a node `?currentAlert` to a node `?s` via the relation property `:provRel`, where `?s` is classified as part of another alert. We represent this

[15]Last access: 11/26/2021
[16]`https://github.com/SigmaHQ/sigma/wiki/Specification`
[17]`https://www.w3.org/TR/sparql11-query/#propertypaths`

condition as an RDF-star[18] statement, i.e., [`«?s ?p ?o» rule:hasDetectedRule ?rule`].

```
PREFIX : <http://w3id.org/sepses/vocab/krystal#>
PREFIX rule: <http://w3id.org/sepses/vocab/rule#>
SELECT  ?s ?p ?o
  WHERE { ?currentAlert ^:provRel* ?s
          <<?s ?p ?o>> rule:hasDetectedRule ?rule.
  }
```

Listing 4.6: Backward searching expressed as SPARQL Query

Next, we iteratively update the alert scores for each predecessor alert (+1). After scoring all alerts, we can construct attack sequences, starting with the alerts with the highest values and performing *forward chaining* to construct attack scenario graphs. We use the same technique as we did for backward chaining, i.e., property paths to find forward routes connected to the defined starting nodes. Section 4.5.3 shows the generic SPARQL query to construct an attack scenario from the provenance graph. From the `?startingNode`, it finds possible routes and visits connected nodes via the `:provRel` relation. A threshold defines which nodes will be connected, e.g., only low integrity nodes with an integrity tag lower than 0.5.

```
PREFIX : <http://w3id.org/sepses/vocab/krystal#>
PREFIX rule: <http://w3id.org/sepses/vocab/rule#>
CONSTRUCT { ?s ?p ?o.}
   WHERE { ?startingNode :provRel* ?s . ?s ?p ?o.
       ?s rule:intTag ?spt. ?o rule:intTag ?spo.
  FILTER ( ?spt < 0.5 && ?spo < 0.5 )
}
```

Listing 4.7: Forward chaining mechanism expressed as SPARQL query

Graph Querying Graph querying can detect attack behavior in a provenance graph based on attack patterns. The graph query patterns can be constructed from observed behavior or existing information in published CTI, incident reports, public malware documentation, etc. Based on that, the patterns can be constructed manually or – potentially – also through automated extraction methods such as AttacKG [162].

Figure 4.8 visualizes a graph query example. A subset of the graph (red box) represents an observed attack pattern inside the provenance graph. Unlike previous work [141, 139, 140] that use custom domain-specific languages to perform graph querying, *KRYSTAL* provides a uniform graph querying mechanism with a high expressivity through SPARQL. Furthermore, it also supports linking to other datasets.

```
PREFIX rule: <http://w3id.org/sepses/vocab/rule#>
PREFIX : <http://w3id.org/sepses/vocab/krystal#>

CONSTRUCT { ?socket :isReceivedBy ?browser.
      ... #similar to where clause
 }WHERE {
     ?socket :isReceivedBy ?browser.
    SERVICE <http://w3id.org/sepses/repositories/knowledge>
     {  ?browser a :Browser. }
```

[18] `https://w3c.github.io/rdf-star/cg-spec/editors_draft.html`

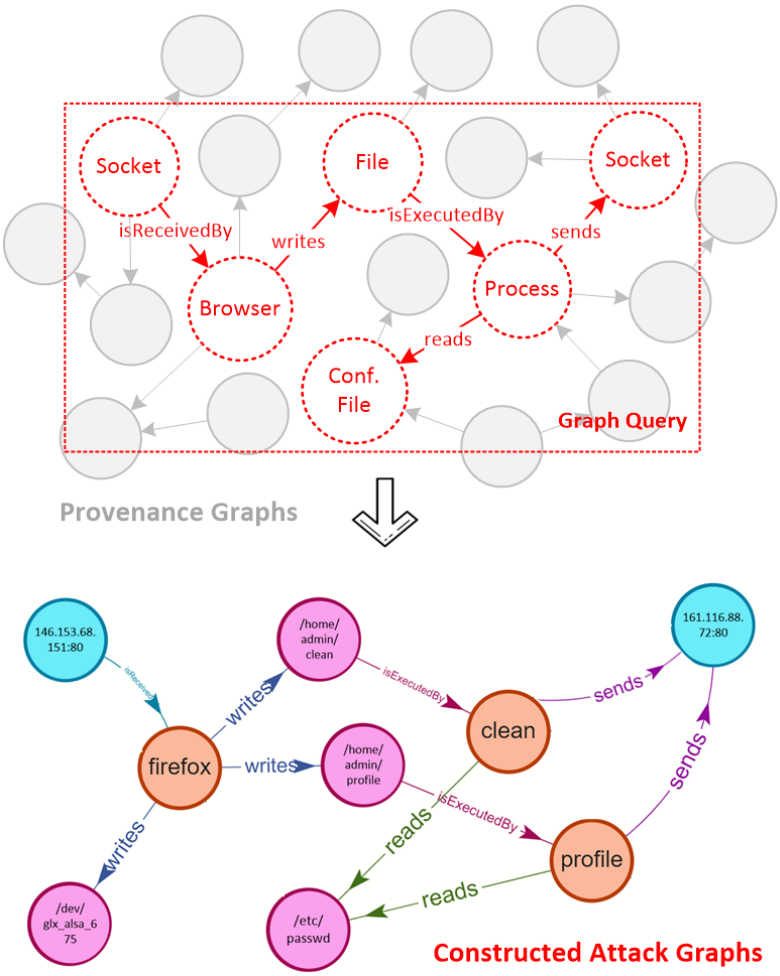

Figure 4.8.: Graph query alignment example: Graph query (red box) is aligned over the provenance graph to detect potential attack patterns. The constructed attack graph below represents a detected pattern.

```
10   ?browser :writes  ?file1. ?file1 :isExecutedBy ?process.
11   ?process :reads ?file2.?process :sends ?socket2.
12   ?file2 rule:confTag ?fit. FILTER (?fit < 0.5)
13 }
```

Listing 4.8: SPARQL graph query[19]for Figure 4.8

To formulate graph patterns in SPARQL[20], we can define system object types as nodes (i.e., via *Classes*) and their interactions as relation properties (i.e., via *Object Properties*). The formulated graph queries can be executed against the provenance graphs to match potential attack patterns. Section 4.5.3 depicts an example graph query visualized in Figure 4.8. It detects a potential attack where a browser receives a connection from a socket (`?socket :isReceivedBy ?browser`). The fact that `?browser` is of type *:Browser* is established in the background knowledge (i.e. `<http://w3id.org/sepses/..knowledge>`). The SPARQL query federation mechanism is used to include the background knowledge in the query (`SERVICE` syntax).

Subsequently, the browser creates a file in the local system (defined by `?browser :writes ?file1`). This file is then executed as a new process (`:file1 :isExecutedBy :process2`) and reads another *confidential* file (`?file1 :reads ?file2`). Next, the process sends the *confidential* file to another socket (`process :sends ?socket2`). We can use the `FILTER` syntax (i.e. `FILTER (?fit < 0.5)`) to focus on confidential files (i.e., with a tag value (`?file2 rule:confTag ?fit`) lower than 0.5).

To reconstruct the resulting attack graph, we used SPARQL `CONSTRUCT` queries to generate new triples in RDF. The generated attack graph can be shared and reused – e.g., for further threat hunting activities and analysis.

4.5.4. Contextualization & Linking

Security-related events are typically highly context-specific and hence, their interpretation requires extensive background knowledge [27]. Such knowledge plays an important role in our approach and can enrich and provide additional information – e.g., to identify high-level attack steps in the generated attack graph.

In particular, we link results to our previously developed SEPSES CSKG[28], a continuously updated cybersecurity knowledge graph that integrates data from various publicly available sources, including CAPEC, CPE, CVE, CVSS, and CWE. Furthermore, we extend the SEPSES CSKG with attack patterns from MITRE ATT&CK that consist of 665 attack techniques and 14 attack tactics (ATT&CK-KG [31]).

Figure 4.9 illustrates this background knowledge linking. The example relation `:isExecutedBy` between a file `:/home/admin/clean` and a process `:Clean` can be linked to attack techniques and tactics in the background knowledge: in this case, the technique *"exploitation for client execution"*, identified by node `:T1203`) from the MITRE ATT&CK matrix[21] [31]. Since node `:T1203` also provides links to additional information (tactics, mitigations, etc.),

[19]Graph pattern that identifies a specific known attack pattern; defined based on the DARPA Transparent Computing TA5.1 Ground Truth Report [131].

[20]Note that the automatic construction of graph patterns from existing resources is out of scope for this paper.

[20]Recall that semantic reasoning infers the type (i.e. *Class*) of system objects automatically based on their relations (i.e., **rdfs2 & rdfs3**), hence, we can define attack patterns based on their relations only.

[21]`https://attack.mitre.org/techniques/T1203/`

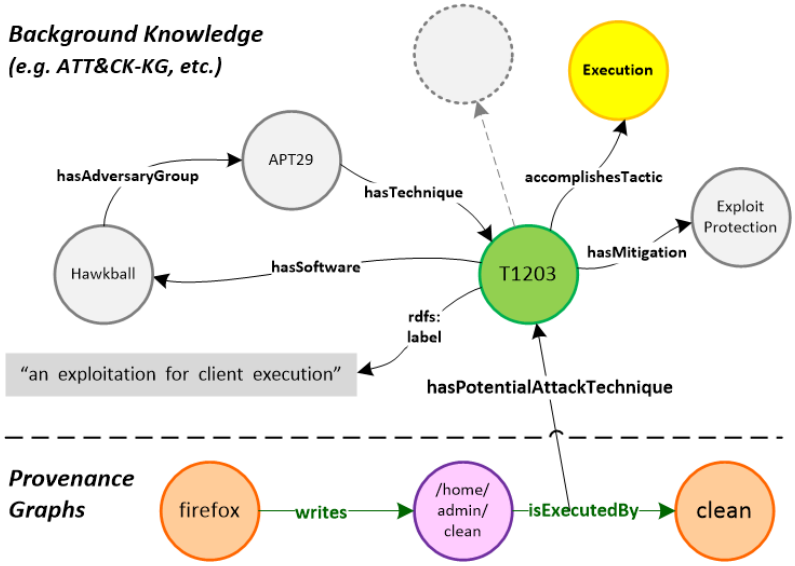

Figure 4.9.: Background linking example, to automatically link alerts detected by the threat detection module to external background knowledge (e.g., ATT&CK-KG).

we can use this node to abstract from concrete indicators to a higher-level conceptualization of attacks and their TTPs.

```
PREFIX : <http://w3id.org/sepses/vocab/krystal#>
PREFIX rule: <http://w3id.org/sepses/vocab/rule#>
PREFIX at: <http://w3id.org/sepses/vocab/ref/attack#>
CONSTRUCT {
  <<?s ?p ?o>> :hasPotentialAttackTechnique ?tech.
  ?s ?p ?o. ?tech at:accomplishesTactic ?tt.}
WHERE {
  OPTIONAL{
   <<?s ?p ?o>> rule:hasDetectedRule ?r.
   SERVICE <http://w3id.org/sepses/repositories/knowledge>
    {?r rule:hasAttackTechnique ?tech.
     ?tech at:accomplishesTactic ?tt.}
  }
   .. #same as forward-chaining query where clause
}
```

Listing 4.9: Contextualization and linking through semantic query federation (SPARQL Service)

Section 4.5.4 shows an excerpt of background linking during *forward chaining* via SPARQL query federation. We modified the query from Section 4.5.3 and introduced additional query patterns such as `SERVICE` followed by an endpoint e.g., *http://w3id.org/sepses/repositories/knowledg* which references external background knowledge. This links a detected rule with a corresponding attack technique in the background knowledge (identified by the triple pattern `?r rule:hasAttackTechnique ?tech`). Finally, we also link the identified technique to a tactic (identified by `?tech at:accomplishesTactic ?tt`). This illustrates how the SPARQL federation mechanism makes it possible to query and link an entity to additional

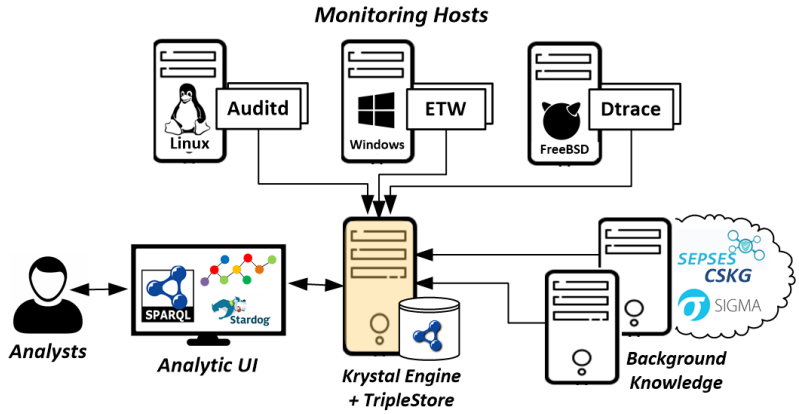

Figure 4.10.: Implementation Setup

resources, such as mitigation techniques, impacts, etc.

4.6. Implementation & Application Scenarios

In this section, we describe the implementation of our framework and demonstrate its feasibility in several scenarios.

4.6.1. Implementation

Figure 4.10 visualizes the implementation architecture of our approach. We developed a Java-based log processing tool called the *KRYSTAL* engine [22] that consumes log data from three different log sources, i.e., Linux *auditd*, FreeBSD *DTrace*[23], and Windows *ETW*. We used the *KRYSTAL* Ontology to parse and map log data into an RDF-based provenance graph. By means of the Jena[24] reasoning engine, we can infer new knowledge during provenance graph building.

Furthermore, we extended the existing SEPSES Cybersecurity Knowledge Graph[28] by including attack techniques and tactics from MITRE ATT&CK and incorporate it as external background knowledge. Furthermore, we translate existing IoC *Sigma*[146] rules into SPARQL queries.

To construct attack graphs (through *backward-forward chaining* and *graph querying*), link internal and external background knowledge (via SPARQL query federation), as well as to visualize the resulting attack graphs, we use the in-memory Jena TDB[25] and the *Stardog Enterprise Knowledge Graph platform*[26].

4.6.2. Application Scenarios

In the following application scenarios, we demonstrate how the *KRYSTAL* framework automatically constructs compact attack graphs, links and contextualizes them with

[22] https://github.com/sepses/SimpleLogProvenance
[23] https://wiki.freebsd.org/DTrace/
[24] https://jena.apache.org/documentation/inference/
[25] https://jena.apache.org/documentation/tdb/
[26] https://www.stardog.com/platform/

Table 4.3.: Attack Scenarios

Scenario ID	Dataset ID	OS Platform	Scenario Name	Scenario description
1	Cadets I	FreeBSD	Nginx backdoor	*Nginx backdoor w/ Drakon in-memory.* An attacker sent a malformed HTTP request to a vulnerable Nginx web server that leads to several malicious file creations and process executions in the local system (Figure 4.11).
2	Cadets I	FreeBSD	Nginx backdoor	*Nginx backdoor w/ Drakon in-memory.* A vulnerable Nginx webserver downloads several malicious files after being exploited by a malformed HTTP request (Figure A.1).
3	Cadets II	FreeBSD	Nginx backdoor	*Nginx backdoor w/ Drakon in-memory.* Similar to Scenario 3, the vulnerable Nginx webserver was successfully exploited by an attacker. It downloads a payload file which leads to sensitive information leaking (Figure A.2).
4	Theia	Ubuntu 12.04	Firefox backdoor	*Firefox backdoor w/ Drakon in-memory.* Firefox process gets exploited by a malicious website to download and execute files to steal sensitive information from users (Figure 4.11).
5	Five Direction	Windows 10	Firefox backdoor	*Firefox backdoor w/ Drakon in-memory.* Firefox process gets exploited via a drakor memory payload after browsing a malicious website (Figure A.4).

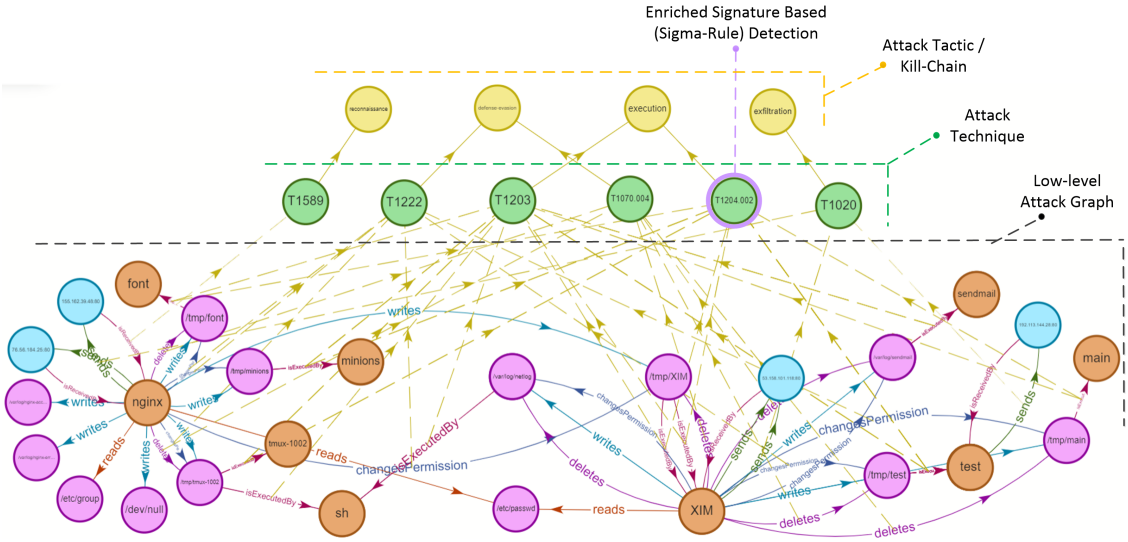

Figure 4.11.: **Scenario 1** (*Nginx backdoor w/ Drakon in-memory*). This attack begins with a vulnerable Nginx web server hosted on a FreeBSD server that gets exploited by a malformed HTTP request. The exploit leads to multiple file creations on the local system. The attacker successfully creates an executable file ("/tmp/XIM"), changes the permissions and runs it as an elevated process. This process reads a sensitive file ("/etc/passwd") and forwards data to an external network (*53.158.101.118:80*).

background knowledge, and finally maps them to high-level attack steps via TTPs defined in MITRE ATT&CK.

We use a DARPA dataset [131] that contains attack scenarios carried out by a red team as part of the DARPA Transparent Computing (TC) program. An overview of the attack scenarios can be found in Table 4.3. Due to space limitation, we only explain one scenarios in this section (Scenario 1), and include the remaining scenarios in the appendix.

Scenario 1 - Nginx backdoor w/ Drakon in-memory This attack scenario has been used as motivation example in Section 4.1. As shown in Figure 4.11, our system detected a number of alerts including *file creation, change permission, file execution, and data-leak*. After performing *backward and forward* chaining, we successfully constructed the attack graph of this scenario. Furthermore, we performed query federation during *forward chaining* to include external background knowledge. Our query yields an attack graph that automatically links the detected alerts to the MITRE ATT&CK techniques and tactics. Finally, we can see the reconstructed attack graph together with its kill-chain phases such as *Reconnaissance, Defense Evasion, Execution* and *Exfiltration*.

4.7. Evaluation

In this section, we evaluate the *KRYSTAL* framework through a set of experiments and discuss the results.

4.7.1. Experimental Setup

We performed the experiments on two machines: *(i)* an Ubuntu 18.04 Server (Intel 2.59 GHz vCPU, 32 GB RAM) was used for provenance graph building and evaluation of threat detection and alerting techniques. *(ii)* a Windows 10 (Intel 2.90 GHz vCPU, 16 GB RAM) machine was used for scenario reconstruction, graph querying and attack graph visualization using *Stardog Studio*[27].

Dataset Overview For the evaluation, we used well-established datasets from red vs. blue team adversarial engagements produced as part of the third Transparent Computing (TC) program organized by DARPA [131]. The datasets are organized into five categories, namely *Cadets, Trace, Theia, FiveDirections and ClearScope*. Each dataset includes log events generated during the engagements on a specific targeted host and platform. For example, *Cadets* represents *dtrace*[28] log data from FreeBSD OS, *Trace* and *Theia* have been generated from *auditd*[29] Ubuntu log data, and *FiveDirection* contains *ETW*[30] log data from Microsoft Windows and *ClearScope* collected from Android logs. In addition, a description of attack steps is available as ground truth. Table 4.4 summarizes the five attack scenarios from the *Theia* (TH), *Cadets* (CD), and *FiveDirection* (FD) datasets that we evaluated. In total, the scenarios covers more than 7 days of log data from three datasets, with more than 53 GB of logs in JSON format[31].

4.7.2. Experiment Results

We collect the following evaluations metrics in our experiments: *(i)* provenance graph size reduction and compression performance, *(ii)* run-time performance, *(iii)* provenance-based alert detection, *(iv)* rule-based alert detection.

Graph Size Reduction & Compression Performance Table 4.4 shows the evaluation results for provenance graph generation and compression in the five scenarios. Our

Table 4.4.: Graph size reduction & compression.

Scenario ID	Dataset ID	Duration (hh:mm)	Log data in JSON (GB)	Prov. in RDF (MB)	Prov. in HDT (MB)
1,2	CD I	48:59	7	150	7
3	CD II	90:01	12	250	10
4	TH	25:33	18.7	280	16
5	FD	19:27	16	25	0.9

system generates RDF-based provenance graphs in RDF TURTLE format[32]. Out of the 18.7 GB Theia log dataset, for instance, we generated a 280 MB provenance graph (i.e.,

[27] `https://www.stardog.com/studio/`
[28] `https://en.wikipedia.org/wiki/DTrace`
[29] `https://linux.die.net/man/8/auditd`
[30] `https://docs.microsoft.com/en-us/windows/win32/etw/event-tracing-portal`
[31] Note that DARPA published the datasets in both binary and JSON formats, we used the JSON data as input in our evaluation
[32] `https://www.w3.org/TR/turtle/`

66x smaller), the set of Cadets datasets (19 GB) resulted in a 400 MB provenance graph (i.e., 48x smaller), and out of the Five direction dataset (16 GB), we generated a 25 MB provenance graph (i.e., 640x smaller).

The last column of Table 4.4 shows the compression results of the generated provenance graphs for each dataset in HDT [123] format. On average, the resulting compressed provenance graphs are approximately smaller by a factor of 22 than the generated provenance graphs in TURTLE format.

Table 4.5.: Scenario graph construction run-time.

Scenario ID	Total Events (M)	Prov. Graph Building & Alerting (events/sec)	Forward Chaining (sec)	Graph Querying (sec)
1,2	7.8	16.8 K	0.79	0.31
3	12.9	17.5 K	0.99	0.26
4	23.4	15.6 K	1.6	0.41
5	19.3	37.8 K	1.99	0.42

Run-time Performance We evaluate three aspects to measure the run-time performance of our approach: *(i)* Time for generating the provenance graphs from the log data, including time for *tag-propagation, attenuation & decay* and *provenance-based alerting*; *(ii)* Time for constructing the scenario graphs via *forward-chaining*; and *(iii)* Time for generating scenario graphs through *graph querying*. We repeated each experiment five times and present average results.

As shown in Table 4.5, our approach can generate RDF-based provenance graphs from log data with up to 20k events/sec on average. The highest performance for provenance generation is achieved in Scenario 5 (*FiveDirection* dataset on Windows), with 37.8k events/sec. Compared to the other scenarios, *FiveDirection* has a larger number of events that are not considered in our model , resulting in a large number of events that can be excluded in the provenance graph generation process.

Compared to MORSE, which achieved 100K events/sec [138], provenance construction is somewhat slower.[33] This performance penalty with respect to approaches based on optimized custom data structures is expected and mainly attributable to the parsing, RDF lifting and the in-memory SPARQL query execution times necessary for tag propagation. The manageable reduction in run time performance in our evaluation demonstrates that the approach is viable. Given the benefits, including improved reusability, interoperability, and enrichment with background knowledge the tradeoff seems favorable. Overall, our framework achieved a performance of 1.2 seconds per attack on average in the scenario graph construction via *forward-chaining*. The highest performance has been achieved in scenario 2 and 3 (*Cadets* dataset with FreeBSD) with 0.79 seconds. Note that we excluded time for *backward-chaining* for root cause identification as it is basically a select query over the provenance graph (without constructing a new graph) and therefore the run times are relatively fast. The run time for scenario graph generation through *graph querying* is even faster, i.e., less than 0.5 seconds for all scenarios. This indicates that our RDF-based

[33] Specifically, by a factor of 5 for Linux/FreeBSD and 3 for Windows, respectively.

provenance graph data model scales well with respect to graph size and query complexity when it comes to graph-query based attack construction.

Table 4.6.: Provenance-based alert detection.

Scenario ID	Total Events (M)	Reconn	Change Perm	File Exec	File Corrupt	Data Leak
1,2	7.8	65	1152	15	115	3
3	12.9	4	1274	1	1040	3
4	23.4	3	9	2	618	3
5	19.3	22	448	0	288	10

Propagation-Based Alert Detection Performance In the following, we measure the effectiveness of our approach in detecting alerts based on alerting policies over the tagged provenance graph. We leverage alert policies such as *Change Permission, File Execution, File Corruption*, or *Data Leak* from [138]. In addition, we created a custom alert *Reconn* that detects connections from external IPs to processes that access sensitive files. Table 4.6 summarizes the detected alerts for all scenarios. Overall, our approach detected all high-level attack activities in the ground truth and achieved similar detection performance as the evaluation in [138] (cf. Table 4.3).

Signature-Based Alert Detection Performance In this evaluation, we used the *Sigma* rules incorporated into our *KRYSTAL* framework to detect attacks based on IoCs from logs. As discussed in Section 4.5.2, we automatically translated *Sigma* rules into executable SPARQL queries and run them against the provenance graph. To this end, we translated most of the existing *Sigma* rules for Linux logs (33 rules) and Windows logs (160 rules). At the time of writing[34], there are no specific *Sigma* rules for *dtrace* FreeBSD, however, *dtrace* FreeBSD logs have a similar structure to *auditd* Linux logs in the evaluated DARPA dataset, hence, we could also use them to detect attacks in FreeBSD logs.

Table 4.7.: Detected alerts by *Sigma* rules.

Scenario ID	Dataset ID	Total Events (M)	Correctly Identified Alert	Incorrectly Identified Alert
1,2	CD I	7.8	34	0
3	CD II	12.9	41	0
4	TH	23.4	19	0
5	FD	19.3	1162	261

Table 4.7 shows the number of triggered alerts based on *Sigma* rules from all five scenarios within our *KRYSTAL* framework. For scenarios with Linux and FreeBSD as log sources, the translated *Sigma* rules detected similar alerts as *propagation-based alert detection* without any incorrectly identified alerts. It includes alerts for *system owner or*

[34]last access 04/05/2021

user discovery[35]*, file or folder permissions change*[36]*, privilege escalation preparation*[37]*, program executions in suspicious folders*[38]*, etc.* Furthermore, *Sigma* rules detected more specific alerts which have been missed by *propagation-based approaches*, such as *bash_profile modification*[39] (as part of the *persistence* phase), and *data compressed*[40] (as part of the *data-exfiltration* phase).

For Windows, we identified more alerts on Scenario 5 (Five Direction) with a total of 1162 alerts (with 261 incorrect alerts that could not be linked to actual attack activities in the scenario). Relevant alerts include, e.g., *Suspicious Service Path Modification*[41], *Suspicious XOR Encoded PowerShell Command Line*[42], *Rar with Password or Compression Level*[43], *Change Default File Association*[44], *LSASS Memory Dumping*[45], and *Capture a Network Trace with netsh.exe*[46].

Integrated and Enriched Cross-Technique Graphs Our experiments showed that the knowledge graph foundation enables the integration of results from various detection techniques and their linking to additional knowledge and gives the analyst a rich view for comprehensive, multi-paradigmatic threat analysis.

The scenario attack graph in Figure 4.11, for instance, shows an enriched attack graph constructed through a combination of techniques summarized in Table 4.8, which compares *KRYSTAL* to other state of the art approaches and highlights the integrative aspect of the framework. The identified attack steps are linked to rich background knowledge on TTPs (SEPSES ATT&CK-KG[31]) in (cf. Figure 4.9, detailed information on tactics and techniques not shown due to space constraints).

Table 4.8.: Detection techniques supported by state of the art approaches.

Detection Technique	Holmes [129]	Morse [138]	Poirot [141]	Rapsheet [142]	Krystal
Tag-Propagation	-	✓	-	-	✓
Rule/Policy-Based	✓	✓	-	-	✓
Signature-Based	-	-	-	✓	✓
Graph Query	-	-	✓	-	✓
Tactical Analysis (TTP Mapping)	✓	-	-	✓	✓

[35] http://bit.ly/sigmaDiscovery
[36] http://bit.ly/sigmaFolderPermission
[37] http://bit.ly/sigmaPrivilegeEscalation
[38] http://bit.ly/sigmaProgramExecution
[39] http://bit.ly/sigmaBashProfileModification
[40] http://bit.ly/sigmaDataCompressed
[41] http://bit.ly/SigmaWinSuspService
[42] http://bit.ly/SigmaWinPowerShellXOR
[43] http://bit.ly/SigmaWinRarFlags
[44] http://bit.ly/SigmaWinChangeFileAssoc
[45] http://bit.ly/SigmaWinLSASSDump
[46] http://bit.ly/SigmaWinNetSHPacket

4.8. Discussion

In this section, we discuss how *KRYSTAL* can support threat detection and attack reconstruction processes and cover its limitations.

Uniform Data Model and Representation A uniform representation is a key foundation to be able to fulfill the requirements put forth in Section 4.3. Existing provenance-graph based detection and investigation approaches lack a unified data format, which hinders their reuse and integration. *KRYSTAL* fills this gap (also cf. [130]) with a knowledge graph framework based on the W3C standards RDF and OWL[47]. The unified model allowed us to combine various state of the art threat detection techniques and apply them on a common provenance graph – fulfilling R2. Furthermore, it also also makes it easier to construct datasets and share provenance graph data.

Our evaluation showed that the *KRYSTAL* ontology can be used to model audit data across platforms – i.e., Linux (*auditd*), FreeBSD (*dtrace*), and Windows (*ETW*) – thereby fulfilling R4. Overall, this should contribute towards lowering the barrier for further research, reproduction, and quantitative comparison.

Finally, we also find that the uniform representation makes it possible to contextualize provenance graphs with knowledge from internal and external sources (R1). This is particularly useful for *KRYSTAL*'s ability to not only reconstruct *low-level* attack graphs, but also link them to *high-level* attack tactics and techniques from MITRE ATT&CK (R3).

Future approaches building on our model can take advantage of the semantic flexibility and richness offered by the existing RDF ecosystem. RDF can, for instance, support multiple paradigms for the implementation of data management architectures in provenance graph-based detection systems, including *(i)* materialized graphs in triple stores, *(ii)* cached graphs implemented as in-memory triple stores [163], *(iii)* distributed graphs [164, 165, 166], *(iv)* virtualized graphs [33, 167], and *(v)* stream reasoning techniques [168].[48]

Standard Query Language *KRYSTAL* leverages SPARQL [171], a graph-based query language for RDF that offers high expressivity and supports complex querying (e.g., aggregation, subqueries, negation) in a declarative manner [172]. We find that this standardized query language provides powerful means to define reusable rules, policies and graph patterns (cf. R2).

In particular, we observe that SPARQL *property path* queries can perform analyses that are critical to provenance-based analyses – such as *backward-forward* chaining for attack graph reconstruction – efficiently. This is despite the fact that theoretical studies [173] on the computational complexity of property paths found that the implementation of a naïve (unfiltered) query can result in double exponential runtime complexity, which becomes critical, e.g., for nodes with more than 15k sequence paths. The typical property path lengths in our scenario attack graphs, however, tend to be rather short, with a maximum path length of 37. This is attributable to the absence of long chains in the provenance data to begin with on the one hand, but also due to tagging and filtering mechanisms such

[47] cf. `https://www.w3.org/standards/`

[48] For a survey of RDF data storage and query processing schemes, cf. [169]. For a survey of approaches to scale to massive data, cf. [170].

as attenuation and decay, which effectively limit path lengths by focusing only on relevant suspicious events and nodes with low integrity. As a consequence, we observe that the attack graph reconstruction through property paths performs efficiently.

Although SPARQL is part of computer science curricula and is increasingly being adopted in many industries[49], it is typically new to security analysts and thus requires some training. This could partly be addressed with general-purpose visual query building and exploration tools such as [174, 126], but we also see a potential for future work in the development of intuitive specialized interfaces for threat detection and attack reconstruction.

Integrated & Modular Framework Current research has resulted in numerous prototypes that each provide solutions for specific detection techniques. *KRYSTAL*, by contrast, enables the combination of multiple different threat detection techniques and attack reconstruction approaches in a single integrated framework (cf. R2). Instead of applying different techniques – each with their own preprocessing pipelines – in isolation, the framework allows us to compare and combine different techniques in a single model. In Sections 4.5 and 4.6, we specifically showed how a variety of detection and attack reconstruction techniques can be formulated in SPARQL and executed within the *KRYSTAL* framework. The modularity of the framework also makes it extensible for future techniques.

Distributed Log Analysis In production settings, security analysis will often require and involve data from disparate sources. *KRYSTAL* is built upon Semantic Web technologies which are explicitly designed for decentralization. Consequently, *KRYSTAL* inherently supports distributed analysis (i.e., querying data across different machines). In Sections 4.5 and 4.6, we demonstrated how *KRYSTAL* can integrate external information sources and facilitate distributed analysis through SPARQL query federation, a technique that enables multiple data sources to be queried in an integrated manner.

To support large-scale provenance graph based attack discovery in production environments, it is necessary to distribute and parallelize computational loads to multiple (local) log processing nodes. This will be part of our future research, where we plan to allow for independent local analysis modules for threat detection and alerting and to integrate the local results into a (global) module, e.g., via SPARQL query federation in order to construct complete attack graphs to address cross-machine attack scenarios. If necessary, several layers of hierarchies can be introduced to better scale the coordination effort.

Online Attack Graph Reconstruction *KRYSTAL* has been evaluated in an offline setting, which can facilitate, e.g., forensic analyses. A (near real-time) online deployment mode would require consideration of issues for attack reconstruction over streaming data. In particular, it would require (i) strategies to dynamically construct attack graphs, (ii) mechanisms to manage continuous updates on a multitude of parallel attack graph reconstruction processes, (iii) policies for prioritizing and discarding attack graph reconstruction processes. Furthermore, the complexity of the approach grows if analyses should be performed in large distributed scenarios and over multiple data streams, which raises issues around time synchronization, latency, throughput, etc. We plan to investigate online scenarios with (semantic) streaming technologies and extend the framework accordingly in future work.

[49] cf. `http://sparql.club`

Unknown Attack Behavior To increase robustness against new ways to evade detection, *KRYSTAL* provides a more general framework that makes it possible to combine a variety of attack reconstruction techniques on top of a common knowledge graph foundation. For instance, we demonstrated how *KRYSTAL* can incorporate existing community-based threat detection rules such as Sigma and integrate them with state of the art detection techniques that have been demonstrated to be effective in the context of APTs.

We argue that although evasion techniques to circumvent detection will always be a concern, the possibility to apply multiple techniques (combining, e.g., rule-based, graph queries, and tag propagation) in the KRYSTAL framework can provide more robust detection of unknown attack behavior compared to the isolated application of individual approaches. Furthermore the flexible framework allows for faster adaptation, experimentation, and parameter tuning. For instance, the rules and policies in the Listings in this paper can be adapted quickly to address new evasion techniques.

4.9. Conclusions

In this paper, we proposed *KRYSTAL*, a modular knowledge graph-based framework for threat detection, scenario reconstruction, and tactical attack analysis. We provide an open, standards-based provenance graph representation based on Semantic Web technologies that can flexibly combine multiple threat detection techniques and contextualize provenance data over both internal system knowledge and external cyber-security knowledge. Based on the *KRYSTAL* ontology, we provide a foundation for provenance and attack graph modeling in a unified framework. The ontology provides semantic interoperability and allows users to leverage community knowledge for tactical attack analysis. Furthermore, our framework introduces a declarative and modular architecture to overcome the inflexibility of monolithic prototypes; hence, it supports rapid development and integration of new approaches, lowering the barriers for rule development, reproducibility, and further research.

To evaluate the ability of SPARQL to effectively express threat detection rules, we implemented several state of the art techniques including *tag propagation*, *attenuation* and *decay*, *signature/rule-based* detection and *graph queries*. We evaluated the feasibility of our approach for threat detection and attack construction through multiple attack scenarios from the well-established DARPA-TC dataset. The evaluation shows that our ontology-based RDF provenance graphs are scalable with respect to graph size and query complexity without sacrificing graph reduction, compression, and attack reconstruction performance. This makes it possible to combine a variety of threat detection techniques, which improved the detection capabilities in our evaluation. For instance, we found that complementary rule-based threat detection identified threats which were missed by tag-propagation techniques.

Finally, our framework facilitates linking to high-level attack patterns to establish a "kill-chain" of high-level attacker tactics, which results in a navigable and queryable provenance graph enriched with security knowledge. This can help improve attack understanding and situational awareness.

As next steps in our research, we plan to integrate with multiple large-scale heterogeneous log sources beyond audit log data and aim for performance optimization (e.g., for better exploration and visualization). Based on the possibility to combine detection techniques, we want to explore how alerts raised by one technique could automatically trigger analysis

by other techniques for confirmation and additional information. Further research will also focus on exploring other threat detection approaches, including *anomaly detection* techniques, and integrating them into our framework. Finally, we plan to adapt and evaluate the attack graph construction approach in different implementation settings such as in *decentralized* and *distributed* scenarios.

5. Conclusions

In this chapter, we review our research questions and hypotheses and discuss how our contributions address them and provide impacts and practical implications. We also discuss the remaining challenges and further research directions.

5.1. Contributions & Impact Summary

As discussed in Chapter 1, the **main research question** in this thesis is:

> *To what extent can semantic web technologies improve cybersecurity monitoring and analysis?*

We decomposed this research question into three sub-research questions (cf. Section 1.4) and focused on five research challenges (cf. Section 1.2). To address those challenges and answer those research questions, we introduced and developed several concepts, methods and frameworks and evaluated them.

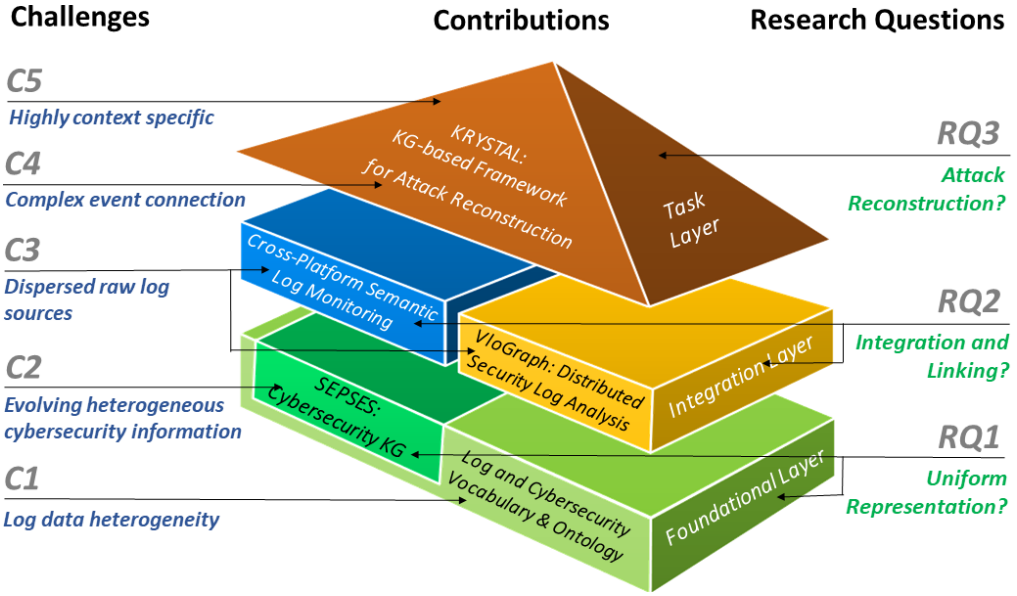

Figure 5.1.: Contribution Overview

Figure 5.1 summarizes the overall contributions we have made and their alignment to the challenges (left) and the research questions (right). The contributions are visualized as a "House" that consists of *three layers of building blocks*, i.e., *(i) Foundational layer*, a basic layer of building block that is required to accommodate the other layers; *(ii) Integration layer*, this layer connects different components in building blocks and enables them to communicate and work together; *(iii) Task layer*, the highest level of the block that is

responsible for providing specific functionality. Each layer comprises one or more building blocks; the foundational layer consists of two building blocks, the integration layer has two building blocks and the task layer has one building block. Each building block represents a contribution, which provides one or more solutions to challenges and answers to research questions. We describe those layers and their building blocks in more detail as follows:

Foundational layer This layer focuses on addressing the foundational challenges of this thesis, i.e., *(i) Log data heterogeneity (C1)* and *(ii) Evolving heterogeneous cybersecurity information (C2)*, which leads to the following research question:

> *RQ1: How to uniformly represent heterogeneous log data and cybersecurity information?*

with the following hypothesis:

> *H1: Security log data and cybersecurity information can be structured and enhanced by semantics to support unambiguous and useful interlinking between logs and cybersecurity information in a knowledge graph. A uniform conceptual model can semantically lift heterogeneous security log data and cybersecurity information from diverse sources.*

On this layer, we provide two building blocks of contributions that address both *C1* & *C2* and answer *RQ1*. First, we introduced modular log vocabularies and ontologies to model and represent log data in RDF. Based on the results, we found that these representations provide semantic integration that not only lifts raw log data but also enriches them with semantic log vocabulary, which makes it easy to connect individual pieces of log events from heterogeneous log sources. Furthermore, they provide a foundation for semantic querying that can be used to not only integrate log events but also to link and contextualize them to rich background knowledge. Therefore they address *C1*.

The second contribution of this layer is knowledge graphs for cybersecurity information (namely SEPSES-CSKG). To this end, we introduced an ontology for cybersecurity information and developed a method to automatically construct knowledge graphs from openly available cybersecurity information and standards such as CVE, CPE, CWE, CVSS, CAPEC, ATT&CK, etc. Based on the results, we found that the generated knowledge graph serves as a rich "external" background knowledge that is useful to link and contextualize local knowledge (i.e., log events,) to detect and identify potential threats as well as to reconstruct attack patterns. Therefore, this contribution addresses *C2*.

In summary, the aforementioned contributions provide evidence to accept *H1* and answer *RQ1*.

Research Impact: *(i)* the log vocabulary introduces a novel concept for semantic log representation and the applicability of *Linked Data Principles* to harmonize and integrate heterogeneous log sources; *(ii)* the SEPSES CSKG advances the state of the art by providing integrated and continuously updated cybersecurity resources in semantically explicit representation, with a tool and service to query and make use of the interlinked knowledge graph.

Practical Implications: *(i)* the proposed approach complements human expertise and intuition (e.g., of a security analyst) with machine interpretation and context-aware decision support for log monitoring and analysis; *(ii)* the CSKG provides integrated & continuously updated security-related knowledge that can be used by security analysts to support automated contextualization and linking during log monitoring and analysis.

Integration layer In this layer, we focus on addressing the challenge of *dispersed raw log sources (C3)*. This leads to the following research question:

> *RQ2: How can dispersed log data and cybersecurity information be integrated and interlinked?*

with the following hypothesis:

> *H2: Through decentralized semantic log processing, multiple dispersed security log data can be integrated in a scalable manner. Furthermore, distributed security log events together with cybersecurity information can be interlinked via semantic query federation to obtain meaningful results.*

To address the challenge and answer the research question, we provide two building blocks of contributions in this layer. First, we developed a framework for real-time semantic log monitoring across heterogeneous platforms. Based on our previously developed log vocabularies and ontologies, the framework lifts and harmonizes incoming heterogeneous raw log data streams into RDF. Next, an existing RDF Stream Processing (RSP) engine, i.e., *C-Sprite* is used to continuously query the parsed log data stream to extract *high-level* file-system events (e.g., *write, read, update, delete*, etc.). Based on our evaluation, we found that the approach can effectively detect file system activities and enrich them with background knowledge to reconstruct the file-access life-cycle across heterogeneous hosts.

The second contribution of this layer introduces a Virtual Knowledge Graph (VKG) approach that provides distributed log analysis across multiple heterogeneous hosts. In contrast to the previous approach that performs log processing on a centralized machine, it dynamically extracts and parses raw-log data into RDF directly from the log-producing hosts on demand (i.e., at query time). The distributed RDF log data are then queried together via semantic query federation to integrate as well as link them to background knowledge (e.g., SEPSES-CSKG). Based on our evaluation, we found that the approach can be used to efficiently integrate & analyze log data from heterogeneous sources. Furthermore, it shifts computational load from a centralized processor to the decentralized hosts, hence addressing *C3*.

In summary, the evaluation of both approaches provides evidence to accept *H2*, therefore answering *RQ2*.

Research Impact: The first contribution shows the applicability of *Stream Reasoning* concepts to tackle the problem of heterogeneous data stream processing in the cybersecurity domain to facilitate, i.e., log monitoring and forensics over heterogeneous platforms.
The second contribution shows the applicability of data processing via A Virtual Knowledge Graph (VKG) approach to tackle the problem of distributed heterogeneous data processing, in particular for the cybersecurity domain, i.e., federated log analysis. The evaluation shows that it is not only able to integrate and aggregate distributed log data in a scalable manner but also to dynamically contextualize and link them to background knowledge.

Practical Implications: the first contribution provides security analysts with automated event detection across heterogeneous log streams. This can be used to automatically detect a potential attack in real-time. The second contribution complements security analysts with an optional distributed log analysis mechanism.

Task Layer This layer focuses on addressing the following challenges: *(i) Complex events connection within and across log source (C4)* and *(ii) Attack analysis and interpretation are very context-specific processes (C5).* They lead to the following research question:

> *R3: How can we discover and reconstruct attacks from system event log information?*

with the following hypothesis:

> *H3: Semantic reasoning can be used to infer sequences of events from security log data and generate causal dependency graphs. This makes it is possible to flexibly and declaratively specify existing attack patterns and rules to detect potential threats, link individual indicators of compromises, identify their root cause and reconstruct attack scenarios.*

To address the challenge and research question, we introduced semantic knowledge graph representation based on RDF to model provenance graphs of log data (i.e., auditlog) from heterogeneous platforms. It automatically infers the sequence of log events based on the identified system objects (e.g., *files, processes, sockets*) and their relations (e.g., *writes, reads, executes, sends, etc*) and generates dependency graphs. Furthermore, we proposed a modular knowledge-graph-based framework (called KRYSTAL) that provides integration of multiple threat detection techniques (e.g., *tag propagation, attenuation and decay, rule-based detection*) and attack reconstruction (e.g., *backward-forward chaining and graph query*). Based on our evaluation, we found that the integration improves threat detection capabilities. This is because of the flexible data model and declarative architecture provided by the KRYSTAL framework. Therefore, it addresses *C4*.

Next, *C5* is addressed by the flexibility and the interoperability of the provenance graph model (i.e., RDF-based graph) that can link and contextualize the aforementioned threat detection techniques to internal and external background knowledge. (i.e., cyber threat intelligence from the CSKG) via semantic query federation.

In summary, the evaluation of the proposed KRYSTAL framework provides evidence to accept *H3*, therefore it gives answers for *RQ3*.

Research Impact: The aforementioned contributions advance the current state-of-the-art threat detection and attack construction approaches by introducing standard and reusable graph representations based on RDF to model provenance graphs from audit log data. The approach can not only integrate a variety of threat detection and attack reconstruction techniques but also contextualize them to the existing internal system knowledge and external cybersecurity information. Furthermore, the extensible and flexible graph model makes it easy for approach customization (e.g., expanding the model, integrating new threat detection techniques, links to other knowledge resources, etc.)

Practical Implications: The proposed approach helps security analysts to apply a variety of threat detection techniques and reconstruct potential attack scenarios in a single modular framework based on a standard data model (i.e., RDF) and querying mechanism (i.e., SPARQL). The flexibility of threat detection integration makes it effective and efficient to better detect potential attacks and therefore improves cybersecurity awareness.

Finally, Table 5.1 shows the summary of the contributions with respect to the research questions and challenges.

Table 5.1.: Summary of Contributions and their alignment to the Research Questions, & Challenges, Artifacts and Publications

Contributions	Challenges	Research Questions	Artifacts & Publications
Vocabulary and ontology for log data and cybersecurity information	C1	RQ1	Log & Cybersecurity Ontology [27, 28]
Knowledge graph for cybersecurity infomation and attack patterns	C2	RQ1	*SEPSES-CSKG* [28, 31]
Semantic approach for security log stream monitoring	C3, C5	RQ2	Semantic Log Stream Monitoring [30, 29]
VKG-based approach for distributed security log analysis	C3, C5	RQ2	*VloGraph* Framework [32, 33]
Knowledge graph-based framework for attack reconstruction	C4, C5	RQ3	*KRYSTAL* Framework [34]

5.2. Limitations and Future Work

This section reviews the current limitation of the proposed approaches and their future work.

Vocabulary and ontology for log data and cybersecurity information Log vocabulary and ontology provide uniform representation for log data integration. However, log data produced by multiple logging and software systems have a large variety of types and structures as well as different levels of granularity. Therefore, it is important to understand these differences as a basis of ontology design and development and how to align them. Furthermore, dynamic updates of systems and logging mechanisms can lead to schema or structural changes in log data, particularly when new systems are introduced. As a result, continuous maintenance and evaluation of the developed vocabulary and ontology are required to ensure their validity, consistency, and robustness. Future work can further explore methods such as ontology evolution [175] in order to address the dynamic evolving schema and structural changes of log data and cybersecurity information.

Knowledge graph for cybersecurity information and attack patterns The dynamic nature of evolving cybersecurity information resources produced by different communities, vendors and companies requires considerable effort to ensure the constructed knowledge graph remains up-to-date and reflects the current state. Especially, for resources that have periodic minor or major schema and structure changes such as MITRE ATT&CK [1], the maintainability of CSKG is essential to ensure the validity of the constructed KG and avoid errors during the updates. Furthermore, the current CSKG focuses mostly on extracting cybersecurity information from structured formats. Therefore, future work could explore methods that incorporate unstructured types of cybersecurity information (e.g., *blog-posts, reports, documentation, tweets, etc*) into the CSKG, e.g., by leveraging Natural Language Processing or Machine Learning techniques. Another limitation of the current CSKG includes performance issues when processing large amounts of resources. This is due to the inherent issue that exists on the current RML engine on processing large files. Therefore, future research can investigate an efficient RDF mapping algorithm that can handle large-scale knowledge graph generation.

Semantic approach for security log stream monitoring Current approach for semantic stream log monitoring has shown promising results in detecting complex file access events in near real-time with high accuracy. However, scalability limitations have been identified when processing log streams with highly concentrated events, such as those produced by *copying a folder containing a large number of files*, which could occur at rates of more than a thousand events per second. The accuracy of the approach decreases as the number of events per second increases.

Therefore, future research could further investigate the accuracy and scalability limitations of the current approach (e.g., by evaluating alternative RSP engines [176] and Complex Event Processing approaches) as well as implement the approach in different settings, i.e., distributed or decentralized scenarios.

[1] MITRE updates their ATT&CK schema bi-annually, cf. `https://attack.mitre.org/resources/versions/`

VKG-based approach for distributed security log analysis A current limitation of VKG approach is the restriction of the extracted log lines (currently based on time frames) when executing queries for analysis to keep the processing time manageable. This is because the log files are parsed on demand and not indexed. Therefore, future research could investigate the possibility to improve the processing time efficiency, e.g., to extend the current approach to support analysis in a streaming mode, e.g, via distributed semantic continuous queries. Furthermore, the current VKG approach requires an analyst to select the target hosts manually before analysis. Therefore, further research could improve the current approach to be able to automatically select the relevant targeted hosts based on the query and background knowledge.

Knowledge graph-based framework for attack reconstruction The performance of Krystal's provenance graph generation (specifically the parsing raw log data to RDF) has been found to be slightly lower compared to the previous work, i.e., MORSE [138]. This is mainly due to the RDF representation that requires unique identification (i.e., URI/URI) of both entities and properties, resulting in *less compact* underlying graph data. Therefore, future work can further investigate alternative RDF representations that strike a balance between unique identification and data compactness.

Additionally, the use of SPARQL for declarative rule language must consider efficient query formulation, as more complex queries can result in longer execution times. Future work could explore efficient querying mechanisms, specifically for a process that involves routine query execution (e.g., tag propagation, threat detection, etc). Apart from this, future research could explore the possibility of involving different log types (beyond the audit log) as well as other threat detection techniques, e.g., *anomaly detection*.

5.3. Remaining Challenges and Further Research Direction

Semantic web technologies and knowledge graphs have the potential to improve cybersecurity monitoring, analysis and attack reconstruction. However, there are several challenges that must be addressed in order to fully unlock their potential. We outline several remaining challenges and provide options for research direction based on the reflection of our findings and limitations.

Intuitive KG-based platform interface Semantic web technologies and knowledge graphs may not be familiar to security professionals who do not have a background in this area. For example, the use of SPARQL-queries for security log analysis may not be intuitive for them. Therefore, future research should focus on developing methods that support them with an intuitive interface, e.g., by providing cybersecurity domain-specific language for querying and searching mechanisms with knowledge graphs. This will enable security professionals to easily and quickly analyze log data using semantic web technologies and knowledge graphs, without the need for extensive training or expertise in these technologies.

Privacy-aware log analysis Log data typically contains sensitive information of individuals, such as *usernames, hostnames, IP addresses* and other relevant details. The analysis of raw log data without considering data privacy can potentially harm individuals' privacy. Future research should explore approaches for knowledge graph-based privacy-aware log

monitoring and analysis. One potential solution is to incorporate anonymization and annotation techniques that preserve individual privacy data in log sources during analysis. However, this can be challenging as analyzing anonymized log data is more difficult. For instance, it may be difficult to identify attack actors and other malicious entities in anonymized log data. Additionally, future research could also explore the use of ML such as federated graph learning [177], which allows for the analysis of log data while preserving individual's privacy data and keeping it decentralized.

Hybrid AI The combination of *symbolic AI* such as knowledge representation and *sub-symbolic AI* such as ML-based approaches has strong potential to address dynamic and highly complex cybersecurity threats. ML-based approaches can support humans in coping with complex, tedious, and repetitive analytic tasks while knowledge representation provides a contextualization and enrichment with formalized knowledge from domain experts, which is typically limited in purely ML-based approach. The current CSKG and log knowledge graphs represented in this thesis can serve as a foundation for more advanced analytics via *Hybrid AI* approach [178]. For example, graph-based learning and relational machine learning [179] - which are largely unexplored filed in cybersecurity - can be used for several tasks, such as: *(i)* node classification and clustering for anomaly detection; *(ii)* link prediction for attack sequence reasoning. *(iii)* sub-graph classification for attack pattern behavior matching, etc.

KG-based cybersecurity research While this thesis primarily focuses on log monitoring, analysis and attack reconstruction, future research can further explore the feasibility of KG-based approach to address cybersecurity challenges in a broader context [180]. In *vulnerability assessment and management* for example, knowledge graphs can be used to model IT assets information (e.g., *software version, types, dependencies, etc.*) and link them to vulnerability knowledge such as CVE. The constructed KG can be used to map and identify potential vulnerabilities of systems based on CVE information, and prioritize them via CVSS scoring mechanism. Furthermore, the enriched KG provides information on how the identified vulnerabilities can affect other connected systems but also how to mitigate them in case of no patches are available yet.

In another context such as *network security*, knowledge-graph can be used to represent network infrastructures such as network topology and their interconnected devices (e.g., *servers, firewalls, hosts, etc.*). The model can then be linked to other contextual information (e.g., *users, policies, control, etc.*) and serve as integrated knowledge graphs for network security analysis, contextualize network threat detections and so on. Other relevant topics such as *risk assessment, incident response, social engineering*, etc. can also be explored to validate the feasibility of KG-approach in addressing cybersecurity challenges.

KG-based cybersecurity research in other domains Other directions in cybersecurity research include the application of knowledge-graph based approaches in various domains, especially in a domain where interconnected components exist. For example, in a *manufacturing industry* specifically in production systems, several components such as processes, products, resources, and technologies are highly interconnected. A knowledge graph can be used to represent them and model their relations. This representation can then be used as a basis for cybersecurity-related tasks e.g., to trace back the root cause of attacks

and propagate their impacts through the production systems network. Apart from that, knowledge graphs are also well-suited for representing other domains as well such as *energy sector* where several energy utilities and networks are interconnected; *transportation industry* where railways network, routes, traffic control, etc. are typically linked; *financial services* that involve transactions, payments, banks, customers, and so on.

Finally, there may be no single approach to security log monitoring and analysis that is suitable for all types of log data and that can satisfy all scenarios. This is because different types of log data can only be used for different purposes, e.g., stream log data is necessary for log monitoring, historical log data for forensic analysis, etc. However, it is necessary to rely on standardized and uniform representations of log data, e.g., via RDF graph model, as it provides interoperability and facilitates integration that makes it easier to develop approaches upon them.

Bibliography

[1] Mandiant. M-Trends 2018; 2018. Available from: https://www.fireeye.com/content/dam/collateral/en/mtrends-2018.pdf.

[2] Symantec. 2018 Internet Security Threat Report; 2018. Available from: https://www.symantec.com/security-center/threat-report.

[3] FireEye, Mandiant. Cybersecurity's maginot line: A real-world assessment of the defensein-depth model; 2014. Available from: https://www.techrepublic.com/resource-library/whitepapers/cybersecurity-s-maginot-line-a-real-world-assessment-of-the-defense-in-depth-model/.

[4] Institute P. Exposing the cybersecurity cracks: A Global perspective; 2014. Available from: https://www.cybersecurityintelligence.com/blog/exposing-cybersecurity-cracks-a-global-perspective--12.html.

[5] KMGP. Cybercrime survey report insight and perspective. KPMG India; 2017. Available from: https://assets.kpmg/content/dam/kpmg/in/pdf/2017/12/Cyber-Crime-Survey.pdf.

[6] Calvanese D, Montali M, Syamsiyah A, van der Aalst WMP. Ontology-Driven Extraction of Event Logs from Relational Databases. In: Reichert M, Reijers HA, editors. Business Process Management Workshops. Cham: Springer International Publishing; 2016. p. 140-53. Available from: https://doi.org/10.1007/978-3-319-42887-1_12.

[7] Kent K, Souppaya MP. SP 800-92. Guide to Computer Security Log Management. Gaithersburg, MD, United States: National Institute of Standards & Technology; 2006. Available from: https://csrc.nist.gov/publications/detail/sp/800-92/final.

[8] He P, Zhu J, He S, Li J, Lyu MR. An Evaluation Study on Log Parsing and Its Use in Log Mining. In: The 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN); 2016. p. 654-61. Available from: http://doi.org/10.1109/DSN.2016.66.

[9] Oliner A, Ganapathi A, Xu W. Advances and Challenges in Log Analysis. Commun ACM. 2012 Feb;55(2):55–61. Available from: https://doi.org/10.1145/2076450.2076466.

[10] Nieles M, Dempsey K, Pillitteri VY, et al. An introduction to information security. NIST special publication. 2017. Available from: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-12r1.pdf.

[11] SECUREWORKS. Information Security vs. Cybersecurity vs. Network Security; 2021. Accessed: 2022-12-36. https://www.secureworks.com/blog/cybersecurity-vs-network-security-vs-information-security.

[12] Comptia. What Is Cybersecurity?; 2022. Available from: https://www.comptia.org/content/articles/what-is-cybersecurity.

[13] Keller N. NIST Cybersecurity Framework. National Institute of Standards & Technology; 2022. Available from: https://www.nist.gov/cyberframework.

[14] Miller DR. Security Information and Event Management (SIEM) Implementation. McGraw-Hill Higher Education; 2011.

[15] Axelsson S. Intrusion detection systems: A Survey and Taxonomy. Citeseer. 2000. Available from: http://www.cse.msu.edu/~cse960/Papers/security/axelsson00intrusion.pdf.

[16] Gander M, Felderer M, Katt B, Tolbaru A, Breu R, Moschitti A. Anomaly Detection in the Cloud: Detecting Security Incidents via Machine Learning. In: Moschitti A, Plank B, editors. Trustworthy Eternal Systems via Evolving Software, Data and Knowledge. Berlin, Heidelberg: Springer Berlin Heidelberg; 2013. p. 103-16. Available from: https://doi.org/10.1007/978-3-642-45260-4_8.

[17] Berners-Lee T, Hendler J, Lassila O. The semantic web. Scientific American. 2001;284(5):34-43. Available from: https://www.academia.edu/download/30225519/p01_thesemanticweb.pdf.

[18] RDFS - Resource Description Framework Schema; 2021. Accessed: 30.03.2021. Available from: https://www.w3.org/TR/rdf-schema/.

[19] OWL - Ontology Web Language; 2021. Accessed: 30.03.2021. Available from: https://www.w3.org/TR/owl2-overview/.

[20] Bizer C, Heath T, Berners-Lee T. Linked data: The story so far. In: Semantic services, interoperability and web applications: emerging concepts. IGI global; 2011. p. 205-27. Available from: http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf.

[21] Hogan A, Blomqvist E, Cochez M, d'Amato C, de Melo G, Gutiérrez C, et al. Knowledge Graphs. No. 22 in Synthesis Lectures on Data, Semantics, and Knowledge. Morgan & Claypool; 2021. Available from: https://kgbook.org/.

[22] Xiao G, Ding L, Cogrel B, Calvanese D. Virtual Knowledge Graphs: An Overview of Systems and Use Cases. Data Intelligence. 2019;1(3):201-23. Available from: https://doi.org/10.1162/dint_a_00011.

[23] Kimball R, Ross M. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling. 2nd ed. USA: John Wiley Sons, Inc.; 2002.

[24] Hevner AR. A three cycle view of design science research. Scandinavian journal of information systems. 2007;19(2):4. Available from: https://aisel.aisnet.org/sjis/vol19/iss2/4.

[25] Hevner AR, March ST, Park J, Ram S. Design science in information systems research. MIS quarterly. 2004:75-105. Available from: https://doi.org/10.2307/25148625.

[26] Kurniawan K. Semantic Query Federation for Scalable Security Log Analysis. In: Gangemi A, Gentile AL, Nuzzolese AG, Rudolph S, Maleshkova M, Paulheim H, et al., editors. The Semantic Web: ESWC 2018 Satellite Events. Cham: Springer International Publishing; 2018. p. 294-303. Available from: https://doi.org/10.1007/978-3-319-98192-5_48.

[27] Ekelhart A, Kiesling E, Kurniawan K. Taming the logs – Vocabularies for semantic security analysis. In: 14th SEMANTiCS Conference; 2018. Available from: https://doi.org/10.1016/j.procs.2018.09.011.

[28] Kiesling E, Ekelhart A, Kurniawan K, Ekaputra F. The SEPSES Knowledge Graph: An Integrated Resource for Cybersecurity. In: The Semantic Web – ISWC 2019. vol. 11779. Cham: Springer International Publishing; 2019. p. 198-214. Available from: http://link.springer.com/10.1007/978-3-030-30796-7_13.

[29] Kurniawan K, Ekelhart A, Kiesling E, Froschl A, Ekaputra F. Semantic Integration and Monitoring of File System Activity. In: 15th SEMANTiCS; 2019. Available from: https://ceur-ws.org/Vol-2451/paper-17.pdf.

[30] Kurniawan K, Kiesling E, Ekelhart A, Ekaputra F. Cross-Platform File System Activity Monitoring and Forensics – A Semantic Approach. In: Hölbl M., Rannenberg K., Welzer T. (eds) ICT Systems Security and Privacy Protection. SEC 2020. IFIP Advances in Information and Communication Technology. Springer, Cham; 2020. Available from: https://doi.org/10.1007/978-3-030-58201-2_26.

[31] Kurniawan K, Ekelhart A, Kiesling E. An ATT&CK-KG for Linking Cybersecurity Attacks to Adversary Tactics and Techniques. In: The Semantic Web – ISWC 2021; 2021. p. 5. Available from: https://ceur-ws.org/Vol-2980/paper363.pdf.

[32] Kurniawan K, Ekelhart A, Kiesling E, Winkler D, Quirchmayr G, Tjoa AM. Virtual Knowledge Graphs for Federated Log Analysis. In: The 16th International Conference on Availability, Reliability and Security; 2021. p. 1-11. Available from: https://doi.org/10.1145/3465481.3465767.

[33] Kurniawan K, Ekelhart A, Kiesling E, Winkler D, Quirchmayr G, Tjoa AM. VLog-Graph: A Virtual Knowledge Graph Framework for Distributed Security Log Analysis. Machine Learning and Knowledge Extraction. 2022;4(2):371-96. Available from: https://www.mdpi.com/2504-4990/4/2/16.

[34] Kurniawan K, Ekelhart A, Kiesling E, Quirchmayr G, Tjoa AM. KRYSTAL: Knowledge graph-based framework for tactical attack discovery in audit data. Computers Security. 2022;121:102828. Available from: https://www.sciencedirect.com/science/article/pii/S016740482200222X.

[35] McCandless D, Evans T. World's Biggest Data Breaches; 2018. http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/ <accessed May 6, 2018>.

[36] IBM Corporation, Ponemon Institute. 2015 Cost of Data Breach Study: Global Analysis; 2015. Available from: https://www.ponemon.org/news-updates/blog/security/2015-cost-of-data-breach-global.html.

[37] Ponemon Institute. Exposing the Cybersecurity Cracks: A Global Perspective; 2014. Available from: https://www.cybersecurityintelligence.com/blog/exposing-cybersecurity-cracks-a-global-perspective--12.html.

[38] Lanthaler M. Creating 3rd generation web APIs with hydra. In: Proceedings of the 22nd International Conference on World Wide Web. ACM; 2013. p. 35-8. Available from: http://www.markus-lanthaler.com/research/creating-3rd-generation-web-apis-with-hydra.pdf.

[39] Turnbull J. The Logstash Book. James Turnbull; 2013. Available at https://logstashbook.com.

[40] Ghent University. Enabling Linked Open Data to accomplish the envisaged Semantic Web; 2018. Accessed: 2018-07-26. Available from: http://rml.io/.

[41] Volz J, Bizer C, Gaedke M, Kobilarov G. Silk – A Link Discovery Framework for the Web of Data. Madrid, Spain; 2009. Available from: http://events.linkeddata.org/ldow2009/papers/ldow2009_paper13.pdf.

[42] MITRE. Common Vulnerabilities and Exposures (CVE); 2018. Accessed: 2018-07-26. Available from: http://cve.mitre.org/about/.

[43] die net. syslogd; 2018. Accessed: 2018-07-26. Available from: https://linux.die.net/man/8/syslogd.

[44] Microsoft. Windows Event Log; 2018. Accessed: 2018-07-26. Available from: https://msdn.microsoft.com/en-us/library/windows/desktop/aa385780(v=vs.85).aspx.

[45] Hallam-Baker PM, Behlendorf B. Extended Log File Format; 2018. Accessed: 2018-07-26. Available from: https://www.w3.org/TR/WD-logfile-960221.html.

[46] NGINX Inc . Configuring Logging; 2018. Accessed: 2018-07-26. Available from: https://docs.nginx.com/nginx/admin-guide/monitoring/logging/.

[47] Apache Software Foundation. Log Files; 2018. Accessed: 2018-07-26. Available from: https://httpd.apache.org/docs/1.3/logs.html#common.

[48] Splunk Inc . Splunk; 2018. Accessed: 2018-07-26. Available from: https://www.splunk.com/.

[49] Papertrail Inc . Solarwinds Papertrail; 2018. Accessed: 2018-07-26. Available from: https://papertrailapp.com/.

[50] Librato Inc . librato; 2018. Accessed: 2018-07-26. Available from: https://www.librato.com/.

[51] Setter M. Logfmt: A Log Format That's Easy To Read and Write; 2018. Accessed: 2018-07-26. Available from: https://blog.codeship.com/logfmt-a-log-format-thats-easy-to-read-and-write/.

[52] MITRE. Common Event Expression (CEE); 2018. Accessed: 2018-07-26. Available from: https://cee.mitre.org.

[53] Sapegin A, Jaeger D, Azodi A, Gawron M, Cheng F, Meinel C. Hierarchical object log format for normalisation of security events. In: 2013 9th International Conference on Information Assurance and Security (IAS); 2013. p. 25-30. Available from: https://doi.org/10.1109/ISIAS.2013.6947748.

[54] do Nascimento CH, Ferraz FS, Assad RE, e Silva DL, da Rocha VH. OntoLog: Using Web Semantic and Ontology for Security Log Analysis. In: Proceedings of the Sixth International Conference on Software Engineering Advances (ICSEA 2011); 2011. p. 177-82. Available from: https://www.thinkmind.org/download.php?articleid=icsea_2011_7_40_10158.

[55] Apache Software Foundation. Apache Log4j 2; 2018. Accessed: 2018-07-26. Available from: https://logging.apache.org/log4j/2.x/.

[56] Hellmann S. RLOG - an RDF Logging Ontology; 2018. Accessed: 2018-07-26. Available from: http://persistence.uni-leipzig.org/nlp2rdf/ontologies/rlog/rlog.html.

[57] OpenLink Software. OpenLink Logging Ontology; 2018. Accessed: 2018-07-26. Available from: http://www.openlinksw.com/ontology/logging.

[58] Bonatti P, Dullaert W, Fernández JD, Kirrane S, Milosevic U, Polleres A. The SPECIAL Policy Log Vocabulary; 2018. Accessed: 2018-07-26. Available from: https://aic.ai.wu.ac.at/qadlod/policyLog/.

[59] Clark A, Mohay G, Schatz B. Rich Event Representation for Computer Forensics. In: Kozan E, editor. Proceedings of the Fifth Asia-Pacific Industrial Engineering and Management Systems Conference (APIEMS 2004). Gold Coast, Queensland: Queensland University of Technology Publications; 2004. p. 2.12.1-2.12.16. Available from: https://schatzforensic.com/publications/APIEMS2004.pdf.

[60] Schatz B, Mohay GM, Clark A. Generalising Event Forensics Across Multiple Domains. In: Valli C, editor. 2nd Australian Computer Networks Information and Forensics Conference. Perth, Australia: School of Computer Networks Information and Forensics Conference, Edith Cowan University; 2004. p. 136-44. Available from: https://www.jstor.org/stable/26504018.

[61] Dell'Aglio D, Della Valle E, van Harmelen F, Bernstein A. Stream reasoning: A survey and outlook. Data Science. 2017;(Preprint):1-25. Available from: http://doi.org/10.3233/DS-170006.

[62] Stearley J. Towards informatic analysis of syslogs. In: 2004 IEEE International Conference on Cluster Computing (IEEE Cat. No.04EX935); 2004. p. 309-18. Available from: https://doi.org/10.1109/CLUSTR.2004.1392628.

[63] Grace LKJ, Maheswari V, Nagamalai D. Web Log Data Analysis and Mining. In: Meghanathan N, Kaushik BK, Nagamalai D, editors. Advanced Computing. Berlin, Heidelberg: Springer Berlin Heidelberg; 2011. p. 459-69. Available from: https://doi.org/10.1007/978-3-642-17881-8_44.

[64] Ming S, Shang Q, Bo C. A Taxonomic relationship Learning Approach for Log Ontology Content Event. JDIM. 2012;10:109-13. Available from: https://www.dline.info/fpaper/jdim/v10i2/9.pdf.

[65] Dandurand et al . Standards and tools for exchange and processing of actionable information. Luxembourg: European Union Agency for Network and Information Security; 2015. Available from: https://www.enisa.europa.eu/publications/standards-and-tools-for-exchange-and-processing-of-actionable-information/at_download/fullReport.

[66] Raskin V, Hempelmann C, Triezenberg K, Nirenburg S. Ontology in Information Security: A Useful Theoretical Foundation and Methodological Tool. In: Proceedings of the 2001 Workshop on New Security Paradigms; 2001. Available from: http://dx.doi.org/10.1145/508171.508180.

[67] Martimiano A, Moreira EdS. An owl-based security incident ontology. In: Proceedings of the Eighth International Protege Conference;. Available from: https://protege.stanford.edu/conference/2005/submissions/posters/poster-martimiano.pdf,.

[68] Undercoffer J, Joshi A, Pinkston J. Modeling Computer Attacks: An Ontology for Intrusion Detection. In: Recent Advances in Intrusion Detection; 2003. Available from: https://doi.org/10.1007/978-3-540-45248-5_7.

[69] Kim A, Luo J, Kang M. Security Ontology for Annotating Resources. In: On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE; 2005. Available from: https://doi.org/10.1007/11575801_34.

[70] Ekelhart A, Fenz S, Neubauer T. AURUM: A Framework for Information Security Risk Management. In: Proceedings of the 42nd Hawaii International Conference on System Sciences; 2009. Available from: https://doi.org/10.1109/HICSS.2009.82.

[71] Fenz S, Ekelhart A. Formalizing Information Security Knowledge. In: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security; 2009. Available from: https://doi.org/10.1145/1533057.1533084.

[72] Schumacher M. In: Toward a Security Core Ontology; 2003. Available from: https://doi.org/10.1007/978-3-540-45180-8_6.

[73] Obrst L, Chase P, Markeloff R. Developing an Ontology of the Cyber Security Domain. In: Proceedings of the 7th International Conference on Semantic Technologies for Intelligence, Defense, and Security; 2012. Available from: https://ceur-ws.org/Vol-966/STIDS2012_T06_ObrstEtAl_CyberOntology.pdf.

[74] Oltramari A, Cranor L, Walls R, McDaniel P. Building an Ontology of Cyber Security. In: Proceedings of the 9th Conference on Semantic Technology for Intelligence,

Defense, and Security; 2014. Available from: https://robert.walls.ninja/papers/oltramari14_stids.pdf.

[75] Souag A, Salinesi C, Comyn-Wattiau I. Ontologies for Security Requirements: A Literature Survey and Classification. In: Advanced Information Systems Engineering Workshops; 2012. Available from: https://doi.org/10.1007/978-3-642-31069-0_5.

[76] Guo M, Wang J. An Ontology-based Approach to Model Common Vulnerabilities and Exposures in Information Security. In: ASEE Southeastern Section Annual Conference; 2009. Available from: http://se.asee.org/proceedings/ASEE2009/papers/PR2009034GUO.PDF.

[77] Wang J, Guo M. Security Data Mining in an Ontology for Vulnerability Management. In: 2009 International Joint Conference on Bioinformatics, Systems Biology and Intelligent Computing; 2009. Available from: https://doi.org/10.1109/IJCBS.2009.13.

[78] Wang J, Guo M, Camargo J. An Ontological Approach to Computer System Security. Information Security Journal: A Global Perspective. 2010;19(2). Available from: https://doi.org/10.1080/19393550903404902.

[79] Ulicny B, Moskal J, Kokar M, Abe K, Smith J. In: Inference and Ontologies; 2014. Available from: http://doi.org/10.1007/978-3-319-11391-3_9.

[80] Iannacone M, Bohn S, Nakamura G, Gerth J, Huffer K, Bridges R, et al. Developing an Ontology for Cyber Security Knowledge Graphs; 2015. Available from: http://doi.org/10.1145/2746266.2746278.

[81] Syed Z, Padia A, Finin T, Mathews L, Joshi A. UCO: A Unified Cybersecurity Ontology. In: Thirtieth AAAI Conference on Artificial Intelligence; 2016. Available from: https://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/download/12574/12365.

[82] Cheng L, Liu F, Yao DD. Enterprise data breach: causes, challenges, prevention, and future directions. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 2017;7(5). Available from: https://doi.org/10.1002/widm.1211.

[83] Gordon P. Data Leakage - Threats and Mitigation. SANS Institute; 2007. Available from: https://www.sans.org/white-papers/1931/.

[84] Suresh NR, Malhotra N, Kumar R, Thanudas B. An integrated data exfiltration monitoring tool for a large organization with highly confidential data source. In: 4th CEEC; 2012. Available from: https://doi.org/10.1109/CEEC.2012.6375395.

[85] Carrier BD. A Hypothesis-based Approach to Digital Forensic Investigations. Purdue University; 2006. Available from: https://docs.lib.purdue.edu/dissertations/AAI3232156/.

[86] Cuzzocrea A, Pirró G. A semantic-web-technology-based framework for supporting knowledge-driven digital forensics. In: 8th MEDES Conference; 2016. Available from: https://doi.org/10.1145/3012071.3012099.

[87] Hu Y, Frank C, Walden J, Crawford E, Kasturiratna D. Profiling file repository access patterns for identifying data exfiltration activities. In: IEEE Symposium on CICS;. Available from: http://doi.org/10.1109/CICYBS.2011.5949404.

[88] Bhavsar K, Trivedi B. Predicting Insider Threats by Behavioural Analysis using Deep Learning. In: International Conference on SAM; 2018. Available from: https://www.proquest.com/openview/6f06225130e29da55fc58b0365ce3772/1?pq-origsite=gscholar&cbl=1976342.

[89] Awad A, Kadry S, Maddodi G, Gill S, Lee B. Data Leakage Detection using System Call Provenance. In: International Conference on INCoS; 2016. Available from: http://doi.rog/10.1109/INCoS.2016.95.

[90] Daren Fadolalkarim EB. PANDDE: Provenance-based ANomaly Detection of Data Exfiltration. Journal of Computer & Security. 2019;84:276-8. Available from: https://doi.org/10.1016/j.cose.2019.03.021.

[91] Schand B, Popitsch N. Lifting File Systems into the Linked Data Cloud with TripFS. In: WWW2010 Workshop on Linked Data on the Web; 2010. Available from: https://eprints.cs.univie.ac.at/69/1/schandl.pdf.

[92] Shen Z, Hou Y, Li J. Publishing distributed files as Linked Data. In: 8th International Conference on FSKD; 2011. Available from: http://doi.org/10.1109/FSKD.2011.6019871.

[93] Popitsch N, Schandl B. Ad-hoc File Sharing Using Linked Data Technologies. In: International Workshop on PSD 2010; 2010. Available from: https://eprints.cs.univie.ac.at/89/1/submission_20100824.pdf.

[94] Mashwani SR, Khusro S. The Design and Development of a Semantic File System Ontology. Journal of Engineering, Technology & Applied Science Research. 2018;8. Available from: https://doi.org/10.48084/etasr.1898.

[95] Kahvedžić D, Kechadi T. Semantic Modelling of Digital Forensic Evidence. In: 2nd ICDF2C; 2010. Available from: https://doi.org/10.1007/978-3-642-19513-6_13.

[96] Alzaabi M, Jones A. An Ontology-Based Forensic Analysis Tool. In: Annual ADFSL Conference on Digital Forensics, Security and Law; 2013. Available from: https://commons.erau.edu/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1245&context=adfsl.

[97] Amato F, Cozzolino G, Mazzeo A, Moscato F. An application of semantic techniques for forensic analysis. In: 32nd WAINA; 2018. Available from: http://doi.org/10.1109/WAINA.2018.00115.

[98] Bonte P, Tommasini R, De Turck F, Ongenae F, Valle ED. C-Sprite: Efficient Hierarchical Reasoning for Rapid RDF Stream Processing. In: 13th ACM International Conference on DEBS. ACM; 2019. p. 103-14. Available from: https://doi.org/10.1145/3328905.3329502.

[99] Chuvakin A, Schmidt K, Phillips C. Logging and log management: the authoritative guide to understanding the concepts surrounding logging and log management. Newnes; 2012.

[100] Kotenko I, Polubelova O, Chechulin A, Saenko I. Design and Implementation of a Hybrid Ontological-Relational Data Repository for SIEM Systems. Future Internet. 2013 Jul;5(3):355-75. Available from: http://www.mdpi.com/1999-5903/5/3/355.

[101] Oliner A, Ganapathi A, Xu W. Advances and Challenges in Log Analysis. Commun ACM. 2012 Feb;55(2):55–61. Available from: https://doi.org/10.1145/2076450.2076466.

[102] Grimaila MR, Myers J, Mills RF, Peterson G. Design and Analysis of a Dynamically Configured Log-based Distributed Security Event Detection Methodology. The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology. 2012 Jul;9(3):219-41. Available from: http://journals.sagepub.com/doi/10.1177/1548512911399303.

[103] Guillermo Suárez de Tangil EP. Advances in Security Information Management: Perceptions and Outcomes. COMPUTER NETWORKS SERIES. Commack, NY, USA: Nova Science Publishers, Incorporated; 2013.

[104] Landauer M, Skopik F, Wurzenberger M, Hotwagner W, Rauber A. Have it Your Way: Generating Customized Log Datasets With a Model-Driven Simulation Testbed. IEEE Transactions on Reliability. 2021 Mar;70(1):402-15. Available from: https://ieeexplore.ieee.org/document/9262078/.

[105] Xiao G, Calvanese D, Kontchakov R, Lembo D, Poggi A, Rosati R, et al. Ontology-Based Data Access: A Survey. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence. Stockholm, Sweden: International Joint Conferences on Artificial Intelligence Organization; 2018. p. 5511-9. Available from: https://www.ijcai.org/proceedings/2018/777.

[106] Kent KA, Souppaya M. Guide to Computer Security Log Management. National Institute of Standards and Technology; 2006. SP 800-92. Available from: https://csrc.nist.gov/publications/detail/sp/800-92/final.

[107] Svacina J, Raffety J, Woodahl C, Stone B, Cerny T, Bures M, et al. On Vulnerability and Security Log Analysis: A Systematic Literature Review on Recent Trends. In: Proceedings of the International Conference on Research in Adaptive and Convergent Systems. RACS '20. New York, NY, USA: Association for Computing Machinery; 2020. p. 175–180. Available from: https://doi.org/10.1145/3400286.3418261.

[108] Jose S, Malathi D, Reddy B, Jayaseeli D. A survey on anomaly based host intrusion detection system. In: Journal of Physics: Conference Series. vol. 1000. IOP Publishing; 2018. p. 012049. Available from: http://doi.org/10.1088/1742-6596/1000/1/012049.

[109] Yadav RB, Kumar PS, Dhavale SV. A survey on log anomaly detection using deep learning. In: 2020 8th International Conference on Reliability, Infocom Technologies

and Optimization (Trends and Future Directions)(ICRITO). IEEE; 2020. p. 1215-20. Available from: http://doi.org/10.1109/ICRITO48877.2020.9197818.

[110] Landauer M, Skopik F, Wurzenberger M, Rauber A. System log clustering approaches for cyber security applications: A survey. Computers & Security. 2020;92:101739. Available from: https://doi.org/10.1016/j.cose.2020.101739.

[111] Sabahi F, Movaghar A. Intrusion Detection: A Survey. In: 2008 Third International Conference on Systems and Networks Communications; 2008. p. 23-6. Available from: http://dx.doi.org/10.1109/ICSNC.2008.44.

[112] Schütte J, Rieke R, Winkelvos T. Model-Based Security Event Management. In: Computer Network Security. vol. 7531. Berlin, Heidelberg: Springer Berlin Heidelberg; 2012. p. 181-90. Available from: http://link.springer.com/10.1007/978-3-642-33704-8_16.

[113] Diederichsen L, Choo KKR, Le-Khac NA. A graph database-based approach to analyze network log files. In: International Conference on Network and System Security. Springer; 2019. p. 53-73. Available from: https://doi.org/10.1007/978-3-030-36938-5_4.

[114] Noel S, Harley E, Tam KH, Limiero M, Share M. Chapter 4 - CyGraph: Graph-Based Analytics and Visualization for Cybersecurity. In: Gudivada VN, Raghavan VV, Govindaraju V, Rao CR, editors. Cognitive Computing: Theory and Applications. vol. 35 of Handbook of Statistics. Elsevier; 2016. p. 117-67. Available from: https://www.sciencedirect.com/science/article/pii/S0169716116300426.

[115] do Nascimento CH, Assad RE, Lóscio BF, Meira SRL. Ontolog: A security log analyses tool using web semantic and ontology. In: 2nd OWASP Ibero-American Web Applications Security Conference; 2010. p. 1-12. Available from: https://www.thinkmind.org/download.php?articleid=icsea_2011_7_40_10158.

[116] Nimbalkar P, Mulwad V, Puranik N, Joshi A, Finin T. Semantic Interpretation of Structured Log Files. In: 2016 IEEE 17th International Conference on Information Reuse and Integration (IRI); 2016. p. 549-55. Available from: http://doi.org/10.1109/IRI.2016.81.

[117] Kenaza T, Aiash M. Toward an Efficient Ontology-Based Event Correlation in SIEM. vol. 83; 2016. p. 139-46. Available from: https://linkinghub.elsevier.com/retrieve/pii/S1877050916301326.

[118] Wang F, Bundy A, Li X, Zhu R, Nuamah K, Xu L, et al. LEKG: A System for Constructing Knowledge Graphs from Log Extraction. In: The 10th International Joint Conference on Knowledge Graphs. IJCKG'21. New York, NY, USA: Association for Computing Machinery; 2021. p. 181–185. Available from: https://doi.org/10.1145/3502223.3502250.

[119] Calvanese D, Kalayci TE, Montali M, Santoso A. OBDA for Log Extraction in Process Mining. In: Reasoning Web. Semantic Interoperability on the Web: 13th International Summer School 2017, London, UK, July 7-11, 2017, Tutorial

Lectures. Cham: Springer International Publishing; 2017. p. 292-345. Available from: https://doi.org/10.1007/978-3-319-61033-7_9.

[120] Krügel C, Toth T, Kerer C. Decentralized Event Correlation for Intrusion Detection. In: Goos G, Hartmanis J, van Leeuwen J, Kim K, editors. Information Security and Cryptology — ICISC 2001. vol. 2288. Berlin, Heidelberg: Springer Berlin Heidelberg; 2002. p. 114-31. Available from: http://link.springer.com/10.1007/3-540-45861-1_10.

[121] Xiaokui Shu, Smiy J, Danfeng Yao, Heshan Lin. Massive distributed and parallel log analysis for organizational security. IEEE; 2013. p. 194-9. Available from: http://ieeexplore.ieee.org/document/6824985/.

[122] Harris S, Seaborne A, Prud'hommeaux E. SPARQL 1.1 query language. W3C recommendation. 2013;21(10):778.

[123] Fernández JD, Martínez-Prieto MA, Gutiérrez C, Polleres A, Arias M. Binary RDF Representation for Publication and Exchange (HDT). Web Semantics: Science, Services and Agents on the World Wide Web. 2013;19:22–41. Available from: http://www.websemanticsjournal.org/index.php/ps/article/view/328.

[124] Taelman R, Van Herwegen J, Vander Sande M, Verborgh R. Comunica: A Modular SPARQL Query Engine for the Web. In: The Semantic Web – ISWC 2018. vol. 11137. Cham: Springer International Publishing; 2018. p. 239-55. Available from: http://link.springer.com/10.1007/978-3-030-00668-6_15.

[125] Haag F, Lohmann S, Bold S, Ertl T. Visual SPARQL querying based on extended filter/flow graphs. In: Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces - AVI '14. Como, Italy: ACM Press; 2014. p. 305-12. Available from: http://dl.acm.org/citation.cfm?doid=2598153.2598185.

[126] Vargas H, Buil-Aranda C, Hogan A, Lopez C. RDF Explorer: A Visual Query Builder for Semantic Web Knowledge Graphs. 2019:4. Available from: https://ceur-ws.org/Vol-2456/paper60.pdf.

[127] Ekelhart A, Ekaputra FJ, Kiesling E. The SLOGERT Framework for Automated Log Knowledge Graph Construction. In: European Semantic Web Conference. Springer; 2021. p. 631-46. Available from: https://doi.org/10.1007/978-3-030-77385-4_38.

[128] Hossain MN, Milajerdi SM, Wang J, Eshete B, Gjomemo R, Sekar R, et al. SLEUTH: Real-Time Attack Scenario Reconstruction from COTS Audit Data. In: Proceedings of the 26th USENIX Conference on Security Symposium. SEC'17. USA: USENIX Association; 2017. p. 487–504. Available from: https://doi.org/10.5555/3241189.324122.

[129] Milajerdi SM, Gjomemo R, Eshete B, Sekar R, Venkatakrishnan VN. HOLMES: Real-Time APT Detection through Correlation of Suspicious Information Flows. In: 2019 IEEE Symposium on Security and Privacy (SP). San Francisco, CA, USA: IEEE; 2019. p. 1137-52. Available from: https://ieeexplore.ieee.org/document/8835390/.

[130] Li Z, Chen QA, Yang R, Chen Y, Ruan W. Threat detection and investigation with system-level provenance graphs: A survey. Computers Security. 2021;106:102282. Available from: https://www.sciencedirect.com/science/article/pii/S0167404821001061.

[131] DARPA transparent computing engagement 3 data release; 2018. Accessed: 02.02.2021. Available from: https://drive.google.com/drive/folders/1QlbUFWAGq3Hpl8wVdzOdIoZLFxkII4EK.

[132] Kumar S. Classification and detection of computer intrusions. Purdue University; 1995. Available from: https://www.cerias.purdue.edu/assets/pdf/bibtex_archive/95-08.pdf.

[133] King ST, Chen PM. Backtracking Intrusions. SIGOPS Oper Syst Rev. 2003 Oct;37(5):223–236. Available from: https://doi.org/10.1145/1165389.945467.

[134] Hossain MN, Wang J, Sekar R, Stoller SD. Dependence-Preserving Data Compaction for Scalable Forensic Analysis. In: Proceedings of the 27th USENIX Conference on Security Symposium. SEC'18. USA: USENIX Association; 2018. p. 1723–1740. Available from: https://doi.org/10.5555/3277203.3277331.

[135] Ji Y, Lee S, Downing E, Wang W, Fazzini M, Kim T, et al. 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. Dallas Texas USA: ACM; 2017. p. 377-90. Available from: https://dl.acm.org/doi/10.1145/3133956.3134045.

[136] Ji Y, Lee S, Fazzini M, Allen J, Downing E, Kim T, et al. Enabling Refinable Cross-Host Attack Investigation with Efficient Data Flow Tagging and Tracking. In: Proceedings of the 27th USENIX Conference on Security Symposium. SEC'18. USA: USENIX Association; 2018. p. 1705–1722. Available from: https://dl.acm.org/doi/10.5555/3277203.3277330.

[137] Kemerlis VP, Portokalidis G, Jee K, Keromytis AD. Libdft: Practical Dynamic Data Flow Tracking for Commodity Systems. SIGPLAN Not. 2012 mar;47(7):121–132. Available from: https://doi.org/10.1145/2365864.2151042.

[138] Hossain MN, Sheikhi S, Sekar R. Combating Dependence Explosion in Forensic Analysis Using Alternative Tag Propagation Semantics. In: 2020 IEEE Symposium on Security and Privacy (SP). San Francisco, CA, USA: IEEE; 2020. p. 1139-55. Available from: https://ieeexplore.ieee.org/document/9152772/.

[139] Gao P, Xiao X, Li Z, Jee K, Xu F, Kulkarni SR, et al. AIQL: Enabling Efficient Attack Investigation from System Monitoring Data. In: Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference. USENIX ATC '18. USA: USENIX Association; 2018. p. 113–125. Available from: https://doi.org/10.5555/3277355.3277367.

[140] Gao P, Xiao X, Li D, Li Z, Jee K, Wu Z, et al. SAQL: A Stream-based Query System for Real-Time Abnormal System Behavior Detection. In: 27th USENIX Security

Symposium (USENIX Security 18). Baltimore, MD: USENIX Association; 2018. p. 639-56. Available from: https://www.usenix.org/conference/usenixsecurity18/presentation/gao-peng.

[141] Milajerdi SM, Eshete B, Gjomemo R, Venkatakrishnan VN. 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. CCS '19. New York, NY, USA: Association for Computing Machinery; 2019. p. 1795–1812. Available from: https://doi.org/10.1145/3319535.3363217.

[142] Hassan WU, Bates A, Marino D. Tactical Provenance Analysis for Endpoint Detection and Response Systems. In: 2020 IEEE Symposium on Security and Privacy (SP); 2020. p. 1172-89. Available from: https://doi.org/10.1109/SP40000.2020.00096.

[143] Xu Z, Wu Z, Li Z, Jee K, Rhee J, Xiao X, et al. High Fidelity Data Reduction for Big Data Security Dependency Analyses. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. Vienna Austria: ACM; 2016. p. 504-16. Available from: https://dl.acm.org/doi/10.1145/2976749.2978378.

[144] 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. IWSPA '20. New York, NY, USA: Association for Computing Machinery; 2020. p. 43–52. Available from: https://doi.org/10.1145/3375708.3380314.

[145] Ekelhart A, Ekaputra FJ, Kiesling E. The SLOGERT Framework for Automated Log Knowledge Graph Construction. In: The Semantic Web: ESWC 2021. Springer International Publishing; 2021. p. 16. Available from: https://doi.org/10.1007/978-3-030-77385-4_38.

[146] Roth F, Patzke T. Sigma :Generic Signature Format for SIEM Systems; 2021. Accessed: 30.03.2021. Available from: https://github.com/SigmaHQ/sigma.

[147] Pinkston J, Undercoffer J, Joshi A, Finin T. A Target-Centric Ontology for Intrusion Detection. In: Workshop on Ontologies in Distributed Systems, held at The 18th International Joint Conference on Artificial Intelligence; 2003. p. 9. Available from: https://www.csee.umbc.edu/~finin/papers/ijcai03OntologiesIDS.pdf.

[148] Mcguinness DL, Fikes R, Hendler J, Stein LA. DAML+OIL: an ontology language for the Semantic Web. IEEE Intelligent Systems. 2002;17(5):72-80. Available from: https://doi.org/10.1109/MIS.2002.1039835.

[149] More S, Matthews M, Joshi A, Finin T. A Knowledge-Based Approach to Intrusion Detection Modeling. In: 2012 IEEE Symposium on Security and Privacy Workshops. San Francisco, CA, USA: IEEE; 2012. p. 75-81. Available from: http://ieeexplore.ieee.org/document/6227687/.

[150] STIX - Structured Threat Information eXpression; 2021. Accessed: 30.03.2021. Available from: https://stixproject.github.io/.

[151] CyBox - Cyber Observable eXpression; 2021. Accessed: 30.03.2021. Available from: https://cyboxproject.github.io/.

[152] CAPEC - Common Attack Pattern Enumeration and Classification; 2021. Accessed: 30.03.2021. Available from: https://capec.mitre.org/.

[153] CVSS - Common Vulnerability Scoring System; 2021. Accessed: 30.03.2021. Available from: https://www.first.org/cvss/.

[154] Kenaza T, Aiash M. Toward an Efficient Ontology-Based Event Correlation in SIEM. Procedia Computer Science. 2016;83:139-46. Available from: https://linkinghub.elsevier.com/retrieve/pii/S1877050916301326.

[155] SPARQL 1.1 Federated Query; 2021. Accessed: 30.03.2021. Available from: https://www.w3.org/TR/sparql11-federated-query/.

[156] Sarker IH, Kayes ASM, Badsha S, Alqahtani H, Watters P, Ng A. Cybersecurity data science: an overview from machine learning perspective. Journal of Big Data. 2020 Dec;7(1):41. Available from: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00318-5.

[157] Mitre ATT&CK Matrix; 2021. Accessed: 30.03.2021. Available from: https://attack.mitre.org/matrices/enterprise/.

[158] CWE - Common Weakness Enumeration; 2021. Accessed: 30.03.2021. Available from: https://cwe.mitre.org/about/index.html.

[159] Noy NF, McGuinness DL. Ontology Development 101: A Guide to Creating Your First Ontology. Stanford knowledge systems laboratory technical report KSL-01-05; 2001. Available from: http://www.ksl.stanford.edu/KSL_Abstracts/KSL-01-05.html.

[160] Marz N, Warren J. Big data: principles and best practices of scalable real-time data systems. Shelter Island, NY: Manning; 2015. OCLC: ocn909039685.

[161] URI - Uniform Resource Identifier; 2021. Accessed: 30.03.2021. Available from: https://en.wikipedia.org/wiki/Uniform_Resource_Identifier.

[162] King ST, Chen PM. Backtracking Intrusions. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles. SOSP '03. New York, NY, USA: Association for Computing Machinery; 2003. p. 223–236. Available from: https://doi.org/10.1145/945445.945467.

[163] Hu C, Wang X, Yang R, Wo T. ScalaRDF: a distributed, elastic and scalable in-memory RDF triple store. In: 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS). IEEE; 2016. p. 593-601. Available from: https://doi.org/10.1109/ICPADS.2016.0084.

[164] Gu R, Hu W, Huang Y. Rainbow: a distributed and hierarchical RDF triple store with dynamic scalability. In: 2014 IEEE International Conference on Big Data (Big Data). IEEE; 2014. p. 561-6. Available from: https://doi.org/10.1109/BigData.2014.7004274.

[165] Lehmann J, Sejdiu G, Bühmann L, Westphal P, Stadler C, Ermilov I, et al. Distributed semantic analytics using the SANSA stack. In: International Semantic Web Conference. Springer; 2017. p. 147-55. Available from: https://doi.org/10.1007/978-3-319-68204-4_15.

[166] Abdelaziz I, Harbi R, Khayyat Z, Kalnis P. A survey and experimental comparison of distributed SPARQL engines for very large RDF data. Proceedings of the VLDB Endowment. 2017;10(13):2049-60. Available from: https://doi.org/10.14778/3151106.3151109.

[167] Xiao G, Ding L, Cogrel B, Calvanese D. Virtual knowledge graphs: An overview of systems and use cases. Data Intelligence. 2019;1(3):201-23. Available from: https://doi.org/10.1162/dint_a_00011.

[168] Dell'Aglio D, Della Valle E, van Harmelen F, Bernstein A. Stream reasoning: A survey and outlook. Data Science. 2017;1(1-2):59-83. Available from: https://doi.org/10.3233/DS-170006.

[169] Wylot M, Hauswirth M, Cudré-Mauroux P, Sakr S. RDF data storage and query processing schemes: A survey. ACM Computing Surveys (CSUR). 2018;51(4):1-36. Available from: https://doi.org/10.1145/3177850.

[170] Ma Z, Capretz MA, Yan L. Storing massive Resource Description Framework (RDF) data: a survey. The Knowledge Engineering Review. 2016;31(4):391-413. Available from: https://doi.org/10.1017/S0269888916000217.

[171] Consortium WWW, et al.. SPARQL 1.1 overview. World Wide Web Consortium; 2013. Available from: https://www.w3.org/TR/sparql11-overview/.

[172] Kaminski M, Kostylev EV, Cuenca Grau B. Semantics and expressive power of subqueries and aggregates in SPARQL 1.1. In: Proceedings of the 25th International Conference on World Wide Web; 2016. p. 227-38. Available from: https://doi.org/10.1145/2872427.2883022.

[173] Arenas M, Conca S, Pérez J. Counting beyond a Yottabyte, or How SPARQL 1.1 Property Paths Will Prevent Adoption of the Standard. In: Proceedings of the 21st International Conference on World Wide Web. WWW '12. New York, NY, USA: Association for Computing Machinery; 2012. p. 629–638. Available from: https://doi.org/10.1145/2187836.2187922.

[174] Haag F, Lohmann S, Bold S, Ertl T. Visual SPARQL Querying Based on Extended Filter/Flow Graphs. AVI '14. New York, NY, USA: Association for Computing Machinery; 2014. p. 305–312. Available from: https://doi.org/10.1145/2598153.2598185.

[175] Noy NF, Klein M. Ontology evolution: Not the same as schema evolution. Knowledge and information systems. 2004;6:428-40.

[176] Tommasini R, Bonte P, Ongenae F, Della Valle E. Rsp4j: An api for rdf stream processing. In: The Semantic Web: 18th International Conference, ESWC 2021, Virtual Event, June 6–10, 2021, Proceedings 18. Springer; 2021. p. 565-81.

[177] Zhang H, Shen T, Wu F, Yin M, Yang H, Wu C. Federated Graph Learning - A Position Paper. CoRR. 2021;abs/2105.11099. Available from: `https://arxiv.org/abs/2105.11099`.

[178] Oltramari A, Francis J, Henson CA, Ma K, Wickramarachchi R. Neuro-symbolic Architectures for Context Understanding. CoRR. 2020;abs/2003.04707. Available from: `https://arxiv.org/abs/2003.04707`.

[179] Nickel M, Murphy K, Tresp V, Gabrilovich E. A Review of Relational Machine Learning for Knowledge Graphs. Proceedings of the IEEE. 2016;104(1):11-33. Available from: `http://doi.org/10.1109/JPROC.2015.2483592`.

[180] Liu K, Wang F, Ding Z, Liang S, Yu Z, Zhou Y. A review of knowledge graph application scenarios in cyber security. arXiv; 2022. Available from: `https://arxiv.org/abs/2204.04769`.

A. Appendix

Scenario 2 - Nginx backdoor w/ Drakon in-memory In this scenario our system detects several alarms on the provenance graph as shown on Figure A.1. We performed *backward-forward* chaining with query federation to construct the attack graph together with its kill-chain phases. We successfully linked the generated attack graph to MITRE ATT&CK techniques such as *Gather Victim Identity Information* (T1589), *File and Directory Permissions Modification* (T1222), *Exploitationfor Client Execution* (T1203) and *Automated Exfiltration* (T1020). Finally, following the connection from techniques to tactics, we can see kill-chain phases including *Reconnaissance, Defense-evasion, Execution* and *Exfiltration*.

Scenario 3 - Nginx backdoor w/ Drakon in-memory As shown in Figure A.2, from the detected alerts, we performed *backward-forward* chaining over the provenance graphs together with query federation to external background knowledge. The system successfully constructed this scenario attack graph together with connections to MITRE ATT&CK techniques and tactics. In that figure, we see identified attack techniques such as *Gather Victim Identity Information* (T1589), *File and Directory Permissions Modification* (T1222), *Exploitation for Client Execution* (T1203) and *Automated Exfiltration* (T1020). Finally, these detected techniques lead to the attack phases including *Reconnaissance, Defense-evasion, Execution* and *Exfiltration*.

Scenario 4 - Firefox backdoor w/ Drakon in-memory As shown in Figure A.3, our system detects several alarms in this scenario, including file execution (*"Clean"* and *"Profile"* process) and data leaks (both *"Clean"* and *"Profile"* processes read a sensitive file *"/etc/passwd"* and send data to *"161.116.88.72:80"*). Furthermore, due to the query federation those detected alarms are automatically linked and mapped to our CTI background knowledge, yielding two MITRE ATT&CK techniques namely *Exploitation for Client Execution* (T1203) and *Automated Exfiltration* (T1020). Those techniques are further linked to high-level MITRE ATT&CK tactics, i.e. *Execution* and *Exfiltration*.

Scenario 5 - Firefox backdoor w/ Drakon in-memory As shown in Figure A.4, our system detects several alerts within this scenario. After performing *backward-forward* chaining and queries with federation to background knowledge, our system successfully identified two MITRE ATT&CK attack techniques within the attack graph, namely *Gather Victim Identity Information* (T1589) and *Automated Exfiltration* (T1020). Finally, we identify kill-chain phases such us *Reconnaissance* and *Exfiltration*.

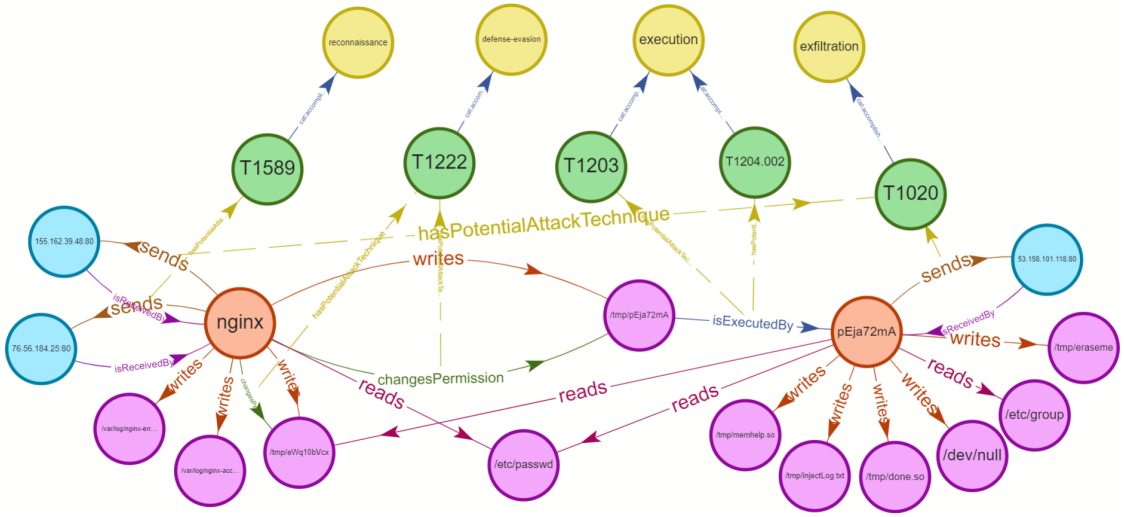

Figure A.1.: **Scenario 2** (*Nginx backdoor w/ Drakon in-memory*). The attack begins with a vulnerable Nginx installed on a FreeBSD host that gets exploited by an attacker. The attacker sends a malformed HTTP request that results in downloading several malicious files on the local system. One of the files, i.e., */tmp/pEja72mA* then gets executed, which spawns a process *pEja72mA*. This process reads sensitive information*/etc/passwd)* and connects remotely via C&C to the attacker console.

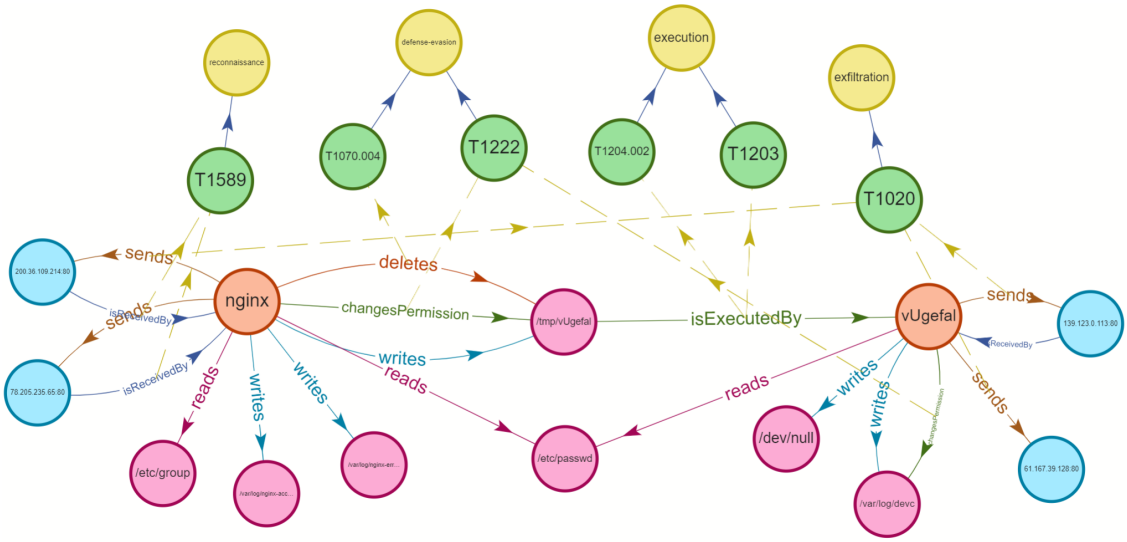

Figure A.2.: **Scenario 3** (*Nginx backdoor w/ Drakon in-memory).* The same case as before, this time the attacker successfully exploits a vulnerable Nginx by downloading another payload file ("/tmp/vUgefal"). The attacker spawns an elevated process. This process later reads sensitive information ("/etc/passwd") from the local system and sends data the the external network.

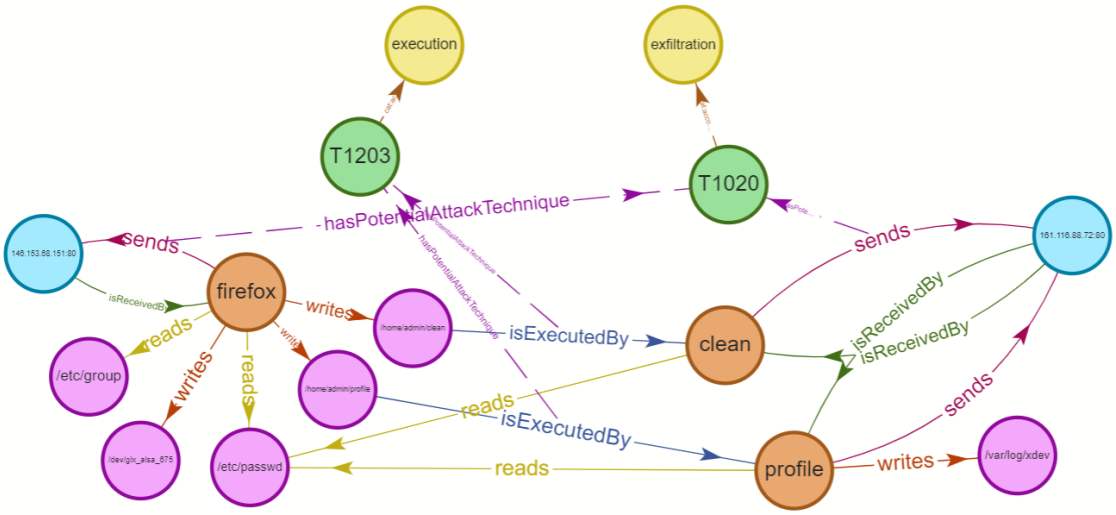

Figure A.3.: **Scenario 4** (*Firefox backdoor w/ Drakon in-memory*) The attack starts with the exploitation of Firefox 54.01 on Ubuntu 12.04 by a malicious ad server. The *Firefox* process gets compromised after visiting a malicious website, and subsequently downloads a malicious file to the user directory *"/home/admin/clean"*. This file is then executed spawning a new process *Clean*. Another file *Profile* is also created and spawned. Both processes access a sensitive file *"/etc/passwd"* and send the data to an external network (*161.116.88.72:80*).

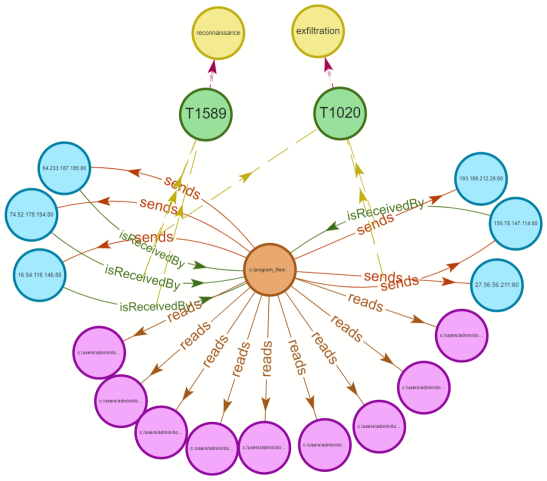

Figure A.4.: **Scenario 5** (*Firefox backdoor w/ Drakon in-memory*). This attack begins with the exploitation of Firefox 54.01, installed on a Windows 10 host. The exploit loads drakon into Firefox memory, which opens a C&C connection to the attack network. Via the Firefox process, multiple user documents are successfully exfiltrated to the attacker's network.