Leveraging Data Science & Engineering for Advanced Security Operations

Pejman Najafi

Dissertation
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
(Dr.-Ing.)

in der Wissenschaftsdisziplin
Cybersecurity

eingereicht an der
Digital-Engineering-Fakultät
der Universität Potsdam

Datum der Disputation: 12. September 2023

Betreuer

Prof. Dr. Christoph Meinel
Hasso Plattner Institute, University of Potsdam

Gutachter

Prof. Dr. Günther Pernul
Universität Regensburg

Prof. Dr. Jingyue Li
Norwegian University of Science and Technology

Published online on the
Publication Server of the University of Potsdam:
https://doi.org/10.25932/publishup-61225
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-612257

I dedicate this thesis to my beloved mother and the memory of my father.

Acknowledgments

I am incredibly grateful for the support and guidance of my supervisor, Prof. Dr. Christoph Meinel, throughout my Ph.D. studies. I also extend my heartfelt thanks to Dr. Feng Cheng for serving as my mentor and providing invaluable guidance throughout my journey. His thoughtful advice and encouragement have been instrumental in helping me navigate the challenges of my research.

To my colleagues and friends, Wenzel Pünter, Alexander Mühle, Dr. David Jaeger, and Dr. Andrey Sapegin, I owe a debt of gratitude for our many fruitful discussions, inspirational brainstorming sessions, and amazing teamwork. I would also like to express my sincere appreciation to all members of the Security Engineering team at HPI who have been with me throughout this journey.

Finally, I wish to express my deepest appreciation to my parents for their loving support, encouragement, and unwavering belief in me. Though my beloved father is no longer with me, his memory continues to inspire me to always strive for excellence and to take one step further. This achievement would not have been possible without their constant love and support.

Abstract

The Security Operations Center (SOC) represents a specialized unit responsible for managing security within enterprises. To aid in its responsibilities, the SOC relies heavily on a Security Information and Event Management (SIEM) system that functions as a centralized repository for all security-related data, providing a comprehensive view of the organization's security posture. Due to the ability to offer such insights, SIEMS are considered indispensable tools facilitating SOC functions, such as monitoring, threat detection, and incident response.

Despite advancements in big data architectures and analytics, most SIEMs fall short of keeping pace. Architecturally, they function merely as log search engines, lacking the support for distributed large-scale analytics. Analytically, they rely on rule-based correlation, neglecting the adoption of more advanced data science and machine learning techniques.

This thesis first proposes a blueprint for next-generation SIEM systems that emphasize distributed processing and multi-layered storage to enable data mining at a big data scale. Next, with the architectural support, it introduces two data mining approaches for advanced threat detection as part of SOC operations.

First, a novel graph mining technique that formulates threat detection within the SIEM system as a large-scale graph mining and inference problem, built on the principles of guilt-by-association and exempt-by-reputation. The approach entails the construction of a Heterogeneous Information Network (HIN) that models shared characteristics and associations among entities extracted from SIEM-related events/logs. Thereon, a novel graph-based inference algorithm is used to infer a node's maliciousness score based on its associations with other entities in the HIN. Second, an innovative outlier detection technique that imitates a SOC analyst's reasoning process to find anomalies/outliers. The approach emphasizes explainability and simplicity, achieved by combining the output of simple context-aware univariate submodels that calculate an outlier score for each entry.

Both approaches were tested in academic and real-world settings, demonstrating high performance when compared to other algorithms as well as practicality alongside a large enterprise's SIEM system.

This thesis establishes the foundation for next-generation SIEM systems that can enhance today's SOCs and facilitate the transition from human-centric to data-driven security operations.

Zusammenfassung

In einem Security Operations Center (SOC) werden alle sicherheitsrelevanten Prozesse, Daten und Personen einer Organisation zusammengefasst. Das Herzstück des SOCs ist ein Security Information and Event Management (SIEM)-System, welches als zentraler Speicher aller sicherheitsrelevanten Daten fungiert und einen Überblick über die Sicherheitslage einer Organisation geben kann. SIEM-Systeme sind unverzichtbare Werkzeuge für viele SOC-Funktionen wie Monitoring, Threat Detection und Incident Response.

Trotz der Fortschritte bei Big-Data-Architekturen und -Analysen können die meisten SIEMs nicht mithalten. Sie fungieren nur als Protokollsuchmaschine und unterstützen keine verteilte Data Mining und Machine Learning.

In dieser Arbeit wird zunächst eine Blaupause für die nächste Generation von SIEM-Systemen vorgestellt, welche Daten verteilt, verarbeitet und in mehreren Schichten speichert, damit auch Data Mining im großen Stil zu ermöglichen. Zudem werden zwei Data Mining-Ansätze vorgeschlagen, mit denen auch anspruchsvolle Bedrohungen erkannt werden können.

Der erste Ansatz ist eine neue Graph-Mining-Technik, bei der SIEM-Daten als Graph strukturiert werden und Reputationsinferenz mithilfe der Prinzipien guilt-by-association (Kontaktschuld) und exempt-by-reputation (Reputationsbefreiung) implementiert wird. Der Ansatz nutzt ein heterogenes Informationsnetzwerk (HIN), welches gemeinsame Eigenschaften und Assoziationen zwischen Entitäten aus Event Logs verknüpft. Des Weiteren ermöglicht ein neuer Inferenzalgorithmus die Bestimmung der Schädlichkeit eines Kontos anhand seiner Verbindungen zu anderen Entitäten im HIN. Der zweite Ansatz ist eine innovative Methode zur Erkennung von Ausreißern, die den Entscheidungsprozess eines SOC-Analysten imitiert. Diese Methode ist besonders einfach und interpretierbar, da sie einzelne univariate Teilmodelle kombiniert, die sich jeweils auf eine kontextualisierte Eigenschaft einer Entität beziehen.

Beide Ansätze wurden sowohl akademisch als auch in der Praxis getestet und haben im Vergleich mit anderen Methoden auch in großen Unternehmen eine hohe Qualität bewiesen.

Diese Arbeit bildet die Grundlage für die nächste Generation von SIEM-Systemen, welche den Übergang von einer personalzentrischen zu einer datenzentrischen Perspektive auf SOCs ermöglichen.

Contents

List of Figures

List of Tables

1 Introduction

The Security Operations Center (SOC) is a centralized department in enterprises responsible for handling security issues at both organizational and technical levels. The SOC integrates *people*, *processes*, and *technologies* to accomplish its objectives. Common SOC tasks comprise identifying risks, managing vulnerabilities, meeting compliance requirements, responding to incidents, and others[183, 266].

Although every organization can establish a SOC, its level of maturity can vary based on the company's size and commitment to IT security. Nonetheless, it's evident that more and more organizations recognize the significance of SOC activities. According to the IBM Security Cost of a Data Breach Report 2022, 83% of organizations have experienced more than one breach[232]. A successful attack on an organization not only incurs costs but also affects reputation and stock prices. As such, an effective SOC is critical in today's threat landscape.

The efficacy of SOC functions is contingent on its personnel, technologies, and processes. However, the fundamental cornerstone of SOC is its data, data that facilitate its operations, e.g., asset inventory, vulnerability management, compliance, monitoring, threat detection/hunting, and incident response.

In this regard, Security Information and Event Management Systems (SIEMs) function as centralized repositories of all security-relevant data, indispensable tools that facilitate a multitude of SOC operations [10, 209].

The primary source of the data in SIEMs is event logs produced by IT systems across the enterprise's landscape, such as security devices, network infrastructure, host and endpoint systems, applications, and cloud services. Other data sources include network telemetry and information about inventories, users, assets, and vulnerabilities.

While SIEM solutions were initially developed to meet regulatory and compliance requirements (e.g., PCI DSS, HIPAA, and SOX), their ability to correlate heterogeneous data and provide a comprehensive view of enterprise security posture has made them highly effective for threat detection, hunting, and monitoring [10, 126].

Threat detection is among the most crucial operations executed by SOCs. Although threat detection methods can be deployed across several platforms, including firewalls, IPS/IDS, and EDR systems, an increasing number of companies are dedicating resources to threat detection within SIEMS [10, 80].

The hypothesis here is that if there is a threat that has managed to successfully bypass the perimeters of defense such as firewall, intrusion detection system, anti-virus, etc., it is quite likely that there are traces of its activities somewhere in these log data shipped to the centralized repository (SIEM or XDR). In this regard, one can apply the detection approaches to these events/logs to identify missed threats.

Consider the SolarWinds hack [81], also referred to as Solorigate, Sunburst, or UNC2452, a prime example of one of the most notable and intricate cyber attacks ever to occur. This attack impacted major Fortune 500 corporations and various governmental organizations, and it brought to light the susceptibility of the software supply chain.

The attackers were able to insert malicious code into a SolarWinds Orion software update which is used for network and system monitoring. This allowed them to gain access to the networks of numerous SolarWinds customers who had installed the compromised software update. The attackers were also able to remain undetected for an extended period of time, reportedly lasting several months, during which they were able to conduct reconnaissance, move laterally within compromised networks, and steal sensitive data.

Sophisticated attacks like the SolarWinds hack can be hard to predict and hard to defend against; however, there are chances to detect them earlier. In the case of SolarWinds, the security firm FireEye uncovered the hack simply because it deployed two-factor authentication for all its employees and found unusual activity. Nevertheless, the attack had left detectable traces within the standard logs collected by most of the mature organizations, such as Windows Events, EDR, Proxy, DNS, and so on [81].

Perhaps the SolarWinds hack could have been detected if there were smarter ways of analyzing terabytes of event logs to uncover the traces of such a sophisticated attack. The lessons learned from the SolarWinds hack are reinforced by the 2022 Data Breach Investigations Report by Verizon [265], which shows that the majority of cyberattacks are detected a month or more after the initial breach event, despite the fact that the traces were available at the time.

Attacks like SolarWinds draw attention to one of the limitations of current cyber threat detection capabilities, the struggle with detecting unknowns, even when we have traces that could assist us in uncovering them.

Another noteworthy case is the 2013 Target hack, during which the attackers infiltrated Target's network and deployed malware that was specifically designed to steal credit card information from the retailer's customers. Several months prior to the breach, FireEye's security solution detected signs of the malware and raised an alert that was subsequently received by Target's SOC. However, the SOC team ignored the alert, and no action was taken to address the issue [221].

The malware used in Target's breach was far from sophisticated. Nonetheless, it was able to bypass the security controls and protocols of a large and well-resourced organization. Despite the presence of traces and alerts related to the malware, the attackers were still able to successfully carry out their attack.

Data breaches such as Target's 2013 hack raise awareness of the fact that security monitoring is typically reactive and heavily relies on human-centered and manual decision-making processes, with security tools used to support the work of analysts.

1.1 Problem Statement

"Knowledge is power" is the adage that has perhaps not lived up to its potential in cybersecurity.

Threat detection in today's enterprises remains human-centric and heuristic-based [10], which have their limitations [171]. The more specific a signature is, the easier it becomes for attackers to evade detection. Conversely, the more general a signature is, the greater the likelihood of generating false positives, leading to alert fatigue, and the possibility of ignorance [10], e.g., in the case of target hack[221] Furthermore, these methods are not effective in detecting advanced threats like the SolarWinds hack or zero-days such as log4j vulnerability exploitation [102].

To address these challenges, academia suggests taking a data-centric approach utilizing data mining techniques such as anomaly/outlier detection[63, 77, 119, 123, 228, 229, 262]. However, industrial applications of these techniques are still limited [10, 228].

Architectural limitations— One of the main reasons for the lack of a data-driven mindset to threat detection can be attributed to the architectural limitations that hinder our ability to analyze data at scale. In a medium-sized enterprise, an established Security Operations Center (SOC) assimilates data from thousands of systems, resulting in a data scale that reaches billions, potentially resulting in terabytes of data per day[284]. Consequently, this places the company's data within the domain of truly big data, characterized by its variety, velocity, and volume. However, the majority of today's SIEM systems are based on the early 2000s technologies originally designed as search engines that are now being used for heuristic-based threat detection, i.e., searching for events that may indicate a known threat. While these systems are quite advanced in their ability to provide storage and accessibility at scale, despite their promises, they do not satisfy the distributed processing requirements for data mining at scale, .e.g, statistical analysis, machine learning [140, 270], and graph mining [40, 98, 99].

Analytical limitations— The other challenge to adapting data mining and machine learning techniques in cybersecurity is the legitimate skepticism among industry professionals, who have been exposed to false, unrealistic promises both from industry and academia. This has led to a reluctance to embrace these techniques, resulting in the growing gap between academia (research) and industry (development) [18]. In a qualitative study conducted by Alahmadi et al. [10], which surveyed SOCs of varying sizes, it was discovered that there is a lack of adoption of machine learning (ML). The authors highlighted that SOC analysts are skeptical of ML in commercial tools, there is a lack of accreditation bodies to attest to the correctness of the ML model design, and high false positive rates are also a concern. This finding is consistent with a 2019 study by SANS [49], which found that there is a general dissatisfaction with AI/ML tools.

1.2 Thesis Contributions

This thesis advocates the shift from a human-centric SOC to data-driven SOC, specifically focusing on the data-driven threat section. In particular, this thesis provides three main contributions:

1. **Reference Architecture Next-generation SIEMs**: Architectural blueprint for next-generation SIEM systems that emphasize distributed processing and multi-layered storage to enable data-driven approaches to tackle SOC problems. The proposed architecture has been implemented as a Proof-of-Concept (PoC) in an academic setting using open-source technologies (e.g., Apache Spark, Flink, Arrow, HDFS, Yarn) as well as in an industrial setting with commercial alternatives (e.g., Databricks, and Azure Data Lake Storage). The platform aims to empower a data-driven Security Operations Center (SOC). This platform can be considered the next-generation SIEM or XDR.

2. **Threat detection via Graph-inference**: A novel graph mining technique that formulates threat detection within the SIEM system as a large-scale graph mining and inference problem, using host-level system events (endpoint logs) and network events (proxy and DNS server logs) enriched with. The approach is built on the principles of guilt-by-association and exempt-by-reputation, with the intuition that an adversary's resources are limited, thereby making the reuse of infrastructure and techniques inevitable. The approach entails the construction of a Heterogeneous Information Network (HIN) that models shared characteristics of entities extracted from underlying event logs enriched with Cyber Threat Intelligence (CTI) and Open-Source Intelligence (OSINT). These characteristics include parent/sub-processes, written/read

files, loaded libraries, registry entries, network connections, and other relevant information. Thereon, a novel graph-based inference algorithm is used to infer a node's maliciousness score based on its associations with other entities in the HIN. The algorithm combines the learnings of PageRank and Belief Propagation to ensure a more accurate inference process.

3. **Threat detection via Outlier Detection**: An innovative outlier detection technique that is designed to imitate a SOC analyst's reasoning process in finding anomalies/outliers and deciding maliciousness. The approach emphasizes explainability and simplicity, achieved by combining the output of simple context-aware univariate submodels that calculate an outlier score for each entry. The process involves fitting a kernel density estimate-like function and examining the density and distance of the observation compared to the underlying distribution.

1.3 Thesis Structure

The thesis is structured as follows:

Chapter: 2 Security Operation Center— provides the necessary background information to comprehend the context, specifically the terminologies used within the Security Operation Centers.

Chapter: 3 Next-Generation SIEM Systems with Big Data Architectures— This chapter starts by presenting an extended background required for understanding big data architectures, leading to the requirements for next-generation SIEM systems that incorporate big data architectures to enable advanced analytics. It then presents the architectural blueprint of next-gen SIEM systems. This will be followed by a case study illustrating the effectiveness of the next-gen SIEM system for beacon detection at scale. This chapter will conclude with a summary and a transition to the next chapter.

Chapter: 4 Graph-based Inference for Threat Detection— This chapter focuses on the application of graph mining, specifically graph inference, as a method for detecting advanced threats. It begins by providing the necessary background knowledge on graph theory and inference. Next, it proposes schemas for a graph/HIN based on SIEM-related logs. Subsequently, the chapter introduces a novel graph inference algorithm named MalRank, along with its unique characteristics and requirements. This will be followed by an experimental setup and two case studies evaluating MalRank in a real-world setting. Finally, this chapter will conclude with a discussion, limitations, future work, a literature review, and a summary.

Chapter: 5 Outlier Detection for Threat Detection— This chapter focuses on the application of outlier detection and anomaly detection in cybersecurity, starting with background knowledge on outlier detection, followed by a section on the requirements and challenges. This leads to the presentation of a unique outlier detection algorithm. The chapter will continue with details of experimental setups and evaluations highlighting the superiority of the proposed algorithm, leading to a discussion section and lessons learned. Finally, the chapter will end with a literature review, limitations, future work, and a summary.

Chapter: 6 Conclusion— The final chapter provides an overview and summary of this thesis's main contributions and findings, followed by limitations and areas for further direction. Finally, ends with final thoughts and remarks.

2 Background Context: SOC

Cybersecurity Framework (CSF) developed by the National Institute of Standards and Technology (NIST) as of the most popular frameworks for standards, guidelines, and best practices for cybersecurity-related risk [52], categorizes cybersecurity functions of Security Operation Centers (2.1) under:

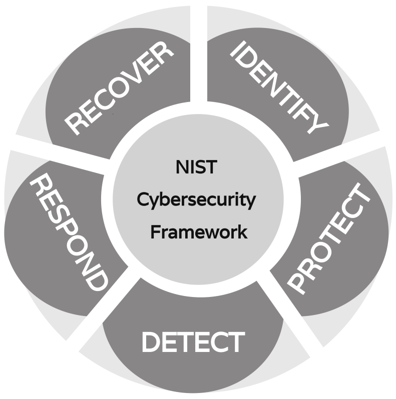


Figure 2.1: NIST Cybersecurity Framework.

1. *Identify*: This stage involves identifying and understanding the systems, assets, data, and capabilities that require protection. It also involves assessing and managing risk based on business objectives and identifying regulatory and compliance requirements such as HIPAA, PCI, GDPR, etc.
2. *Protect*: This stage focuses on the security of infrastructure, assets, and data. It includes implementing access controls, protective technology, managing vulnerabilities (scanning and patching), and security awareness training for employees.
3. *Detect*: This stage involves monitoring and detecting cybersecurity risks and threats. It includes implementing continuous monitoring systems and procedures to identify threats, vulnerabilities, and incidents. As part of detection, organizations may also perform penetration tests and Purple/Red teaming exercises to identify gaps and test their detection capabilities.

4. *Respond*: This stage focuses on responding to cybersecurity incidents in a timely and effective manner, including isolation of the targeted systems, alerting members, implementing playbooks, and remediation steps.

5. *Recover*: This stage involves restoring the organization's systems and data to their pre-incident state. It also includes incorporating lessons learned from the incident into the organization's cybersecurity policies, procedures, and practices.

Many of today's mature SOCs review NIST recommendations when designing and implementing their SOC. The standard also formulates a set of concrete tasks for each category as guidelines for operating SOCs within today's enterprises.

There are also other standards, frameworks, and guidelines allowing organizations to better prioritize their cyber defense efforts, e.g., CSI CSC [43], COBIT [45], and ISO 31000:2018 [116]. The details of these frameworks and guidelines are beyond the scope of this thesis. However, one could refer to the provided references to learn more.

While discussing all SOC operations, tools, and procedures that are beyond the scope of this thesis, in this section, we expand on some key terminologies that allow us to better understand the rest of this thesis.

2.1 SOC Maturity Model

There are models and frameworks that allow one to assess the maturity of an organization's SOC department, e.g., CMMI, SOC-CMM, and ISACA COBIT 5 Process Assessment Model.

If we take a broader perspective on SOC, which encompasses people, processes, and technology, we can define its maturity levels as follows:

- First-generation SOC: These SOCs have limited data coverage and rely on a few security tools. They focus mainly on risk management and have limited incident response practices.

- Second-generation SOC: These SOCs use data correlation and consolidation to turn data into security-relevant events, enabling automated and simplified security monitoring and formalized incident response (via the usage of playbooks). They can provide more advanced risk management and mature incident response services. However, they are still largely reactive.

- Third-generation SOC: These SOCs introduce additional capabilities, such as vulnerability management and compliance, and are expected to be more proactive in threat preparation.
- Fourth-generation SOC: These SOCs leverage the latest SOC technologies and trends, invest significant resources in improving processes, technologies, and people, and continuously measure, evaluate, and improve their functionalities. They are not only proactive but also highly adept at monitoring and responding to security incidents.

In this thesis, we expand on the future of SOC, specifically as a fourth-generation SOC that leverages data-driven approaches and techniques in order to improve various SOC activities, including advanced threat detection, vulnerability analysis, alert prioritization, and more.

2.2 SOC Tools and Technologies

As mentioned, SOC is defined as the combination of people, technology, and processes. While people are beyond the scope of this thesis, technologies and processes are in the scope.

The tools and technologies utilized by the SOC depend on its maturity level. Figure 2.2 provides an example of the technologies maintained by the SOC for an organization. Furthermore, while discussing all technologies used within today's SOCs is also beyond the scope, here we expand on some of the most relevant ones.

2.2.1 IDS/IPS

An Intrusion Detection System (IDS) is a monitoring system that is typically installed on a single system attempting to detect suspicious activities and generates alerts that will be consumed by SOC analysts for further investigation. IDS systems are usually categorized into Host-Based IDS (HIDS) and Network-Based IDS (NIDS) [156]. While HIDs are typically installed on endpoints (hosts) to monitor operating system resources, NIDs are typically deployed on intermediary network nodes to monitor network traffic. There are two categories of IDS detections: signature-based techniques and anomaly-based techniques. Signature-based techniques involve evaluating activities using a set of well-known signatures or patterns of attack stored in the IDS database. When an attempt matches a signature, the IDS triggers an alert. On the other hand, anomaly-based techniques rely on creating a baseline profile that represents normal/expected network behavior. Any observed

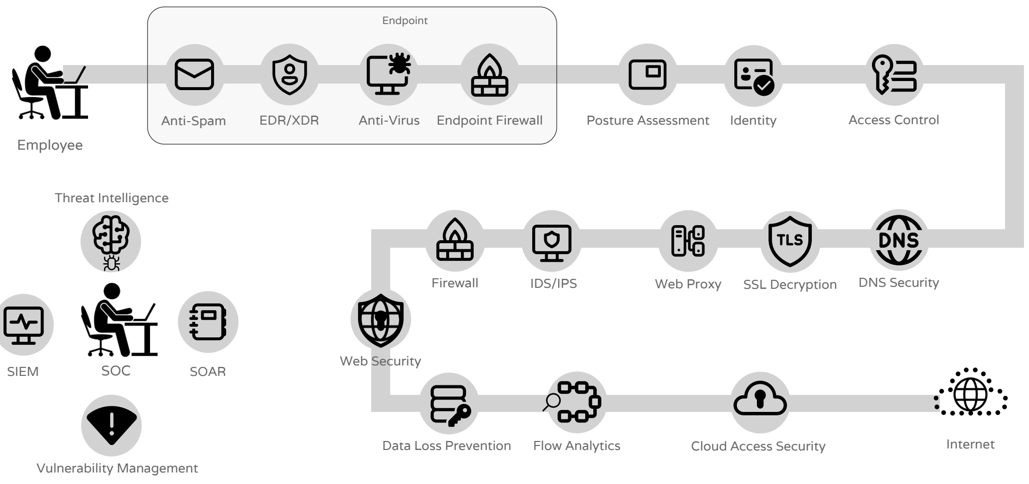


Figure 2.2: Solutions and technologies utilized by typical SOC.

deviation from this profile is considered anomalous. This profile is mostly generated using statistical and historical network traffic data.

An Intrusion Prevention System (IPS) is a network monitoring tool that operates inline, which means that it can block detected threats to prevent them from reaching their targets. However, it can also be configured to operate passively, where it only inspects and copies traffic, acting more like an Intrusion Detection System (IDS).

2.2.2 SIEM

A SIEM system integrates two formerly heterogeneous systems, a Security Information Management (SIM) system, and a Security Event Management (SEM) system. SEM systems were originally designed as a tool to provide real-time monitoring for security events and alerts oriented to identify and manage threats. In comparison, SIM systems were designed as a log management tool for recordkeeping and reporting of security-related events supporting compliance, forensic investigation, and analysis of security threats [2]. SIEM systems were raised as the result of integrating SIM and SEM to simplify the IT landscape. Since then, these systems have evolved to support a wide variety of needs.

One of the limitations of IDS systems is their limited ability to have a holistic view of the IT landscape to support better decision-making, i.e., event correlation. For instance, while a failed login event is nothing to concern with, multiple failed logins to a different host by a single user is concerning. This can only be recognized while correlating events from various endpoints. That is why over time, SIEM systems have evolved to also act as an IDS system supporting threat detection as the last perimeter of defense.

The typical workflow of a SIEM system comprises several stages, starting with data pre-processing, which involves identifying the data source, parsing data fields, and converting them into a structured data format comprehensible by the tool. Additional pre-processing steps, such as normalization and deduplication, may also be performed. Subsequently, the system moves to data correlation and enrichment, where additional information is added to the incoming data. Finally, the data is stored and indexed, with many SIEM solutions built on top of technologies designed for string search, such as index stores. This is because the majority of data are semi-structured, with various fields that are challenging to unify. Therefore, allowing the search for terms in raw data to collect relevant data is a critical capability of the SIEM system.

Gartner research group [225] characterized the main requirements for Security Information and Event Management systems as follows:

- Information and Event Management: The main requirement for SIEM systems remains as the collection and storage of events and logs from heterogeneous devices in the organization, allowing SOC analysts to monitor the landscape, providing visualization, reporting, and alerting mechanisms. These trends can be created based on real-time and/or historical data to identify patterns that can aid in gaining insight into high-risk behavior. The report can also be used to measure the status against compliance regulations and standards such as PCI DSS, GDPR, HIPAA, and SOX.

- Threat Hunting and Investigation: SIEM systems are expected to be the centralized repository holding all security-relevant information and event. These systems are used by SOC analysts to freely explore and analyze data, hunt for threats, or investigate known security incidents. Thus the search features and functionality is fundamental in a SIEM tool. This requires the platform to run efficient ad-hoc queries against massive amounts of data and combine heterogeneous structured and unstructured data. Searching capability can vary from a descriptive taxonomy to free-flowing search queries (e.g., regex) and vendor-specific language.

2.2.3 SOAR

While SIEMs provide the capability to have a holistic view of enterprise security, they lack the functionality to execute actions in case of an attack. For instance, blocking the source of an attack or quarantining the affected assets. In the absence of this feature, analysts would depend on the SIEM product to stay aware of events while using alternative means to take action. Organizations may develop documented procedures for carrying out actions and transform them into playbooks, which can eventually be automated. This process, known as orchestration, is part of the incident response.

This is the main functionality of a Security Orchestration, Automation, and Response (SOAR) system. A system that integrates incident response processes and automated workflows. SOAR platforms aim to streamline incident response by automating repetitive tasks such as alert contextualization and prioritization, playbooks run, and reporting.

SOARs are mostly considered as an extension of the SIEM systems filling the gap for automated response and orchestration.

2.2.4 TIP

Many organizations use dedicated threat intelligence exchange platforms that connect to other parties in open and closed circles to collect and share Cyber Threat Intelligence (CTI), providing security teams and solutions with the latest threat insights for better signature-based detection of known attacks and threats. Although often integrated with SIEM, these platforms allow for various forms of intelligence exchange, including Indicators of Compromise (IoCs), such as lists of known malware hashes or domains, as well as information on techniques and tactics employed by threat actors.

2.2.5 Vulnerability Management

Vulnerabilities are weaknesses of systems that can be exploited by an attacker. Vulnerabilities can be an extremely easy way for attackers to achieve their objectives, as some could pose significant impacts, e.g., remote code execution.

Vulnerability management and scanner tools are essential components of a robust SOC. Vulnerability management refers to the process of identifying, assessing, and mitigating vulnerabilities in an organization’s IT infrastructure. This involves regular scans of networks, systems, and applications to detect and prioritize vulnerabilities based on their risk to the organization. Vulnerability scanner tools are software applications that automate the scanning process, allowing security teams to quickly identify potential vulnerabilities and take appropriate remedial action. These tools use a variety of techniques, such as port scanning, network mapping, and vulnerability testing, to identify weaknesses in an organization’s security posture. There are also various OSINT repositories that today's SOC utilizes to be updated with the most recent vulnerabilities, e.g., CVE[50], CWE[51].

2.2.6 EDR/XDR

Endpoint detection and response (EDR) are the ewer iterations of Anti-Virus (AV) solutions that incorporate a lightweight system agent that monitors system events (e.g., network connections, file modifications, processes, etc.) and sends events to a cloud or SIEM system to detect suspicious events. Furthermore, offering remote control to initiate actions on the system. In addition to standard AV functionalities such as IOC-based detection, blocking, and quarantining, these solutions often include behavioral analysis components.

EDRs also collect system events, including the executable's hash, file name, file path, command line, parent process, sub-process creation, file modifications, libraries loaded, registry modifications, and network connections, making EDR logs one of the most valuable data sources commonly forwarded to the enterprise's SIEM systems. 61% of today's organizations deploy EDR tools [259].

While traditional EDR tools focus only on endpoint data, XDR solutions seek to unify disparate security tools extending the detection scope beyond the endpoints to networks, servers, cloud workloads, etc.

Some consider the XDR as the future of SIEM systems, combining the capabilities associated with separate SIEM, UEBA, NDR, and EDR tools while leveraging advanced analytics and machine learning, and automation.

2.3 Threat Detection in SOC

One of the most vital security operations carried out by today's SOCs is threat detection. While the majority of security appliances attempt the detection (hence possible prevention) at the place in which the appliance is located (e.g., Firewall where the traffic flow or EDR at the endpoint where processes are executed), there is a trend of moving the detection as more of an aftermath with the advantage of getting a holistic view across all solutions (i.e., within the SIEM or XDR solutions).

While most of the literature categorizes detection as signature-based and anomaly-based, we would like to utilize the Rumsfeld-Matrix [223], which classifies cyber threats into four categories of *Known-Knowns*, *Known-Unknowns*, *Unknown-Knowns*, *Unkown-Unknowns*.

Following this categorization of threats, we argue that the detection technique can fall under *known detection* and *unknown detection.*

2.3.1 Detection of Knowns

Known detection techniques are very similar in nature to signature-based detection or heuristic-based detection, which focuses on known knowns as well as unknown knowns. In other words, looking for previously identified threats.

Although signature-based detection remains one of the most commonly employed techniques in the cybersecurity industry, it is not without its limitations. The more specific a signature is, the easier it becomes for attackers to evade detection, for instance, through minor modifications in malware that generate new hashes. In addition, evading techniques such as encryption, packing, obfuscation, polymorphism, metamorphism, fileless [148], or stealthy malware [79, 148, 153] challenges the signature-based detection [281]. Conversely, the more general a signature is, the greater the likelihood of generating false positives, leading to alert fatigue.

Heuristics can be extended to include behavior, which means looking for specific Tactics, Techniques, and Procedures (TTPs) rather than a particular signature. The advantage of this approach is that while an attacker may be able to bypass a specific static signature, it is much more difficult to modify the underlying behavior. For example, in the case of ransomware, one would expect the encryption of the hard drive of the infected system as part of its operation. While attempting to detect the ransomware via its file hash (signature) can be easily circumvented, the encryption behavior cannot, as it is a core part of the malware's behavior.

The majority of today's known detection is carried out within the enterprise SIEM system with a set of correlation rules (heuristics) that describes the potential attack.

Known detection or heuristic-based detection relies on the knowledge of previously identified threats and attacks, i.e., Cyber Threat Intelligence. This intelligence can take various forms, including simple IoCs such as file hashes, domain names, or IP addresses, as well as more complex attack patterns expressed in the form of heuristics or signatures (TTP) [25].

While for simpler IoCs, there are various formats and standards to exchange intel, e.g., STIX [248], OPENIOC[117], MISP[267], IODEF [56].

For more complex TTPs, there are frameworks such as MITRE ATT&CK [177] and CAPEC [35] that allow analysts to better understand complex adversary TTPs (attacks). These are good sources of CTI both on the tactical level (threat modeling) and operational level creation of behavioral rules for the detection, as well as adversary emulation (red teaming) and defensive gap analysis.

2.3.2 Detection of Unknowns

Although heuristic-based detection is widely accepted and employed in the real world, it has limitations in detecting unknown threats such as zero-days and APTs. To address this gap, the industry and academia suggest anomaly detection [63, 77, 119, 123, 228, 229, 262]. The hypothesis is that deviation from normal is an anomaly, and an anomaly translates to maliciousness (a threat). However, as we will discuss in later sections, this assumption may not necessarily be true.

While anomaly detection holds the potential for identifying unknown threats, implementing it correctly can be challenging due to hypothetical and technical challenges. This thesis aims to explore data-driven solutions to tackle the detection of unknowns.

2.4 Data in SOC

There are various data one can collect that could benefit the IT security of an organization; however, one has to consider collection, storage, processing, and the values possible to derive.

One cannot simply collect all security-relevant data and dump it to a SOC tool such as SIEM and expect the analyst to find the needle in the haystack. It is important to define the expectations and the way.

That is why it is crucial to understand the *data* within today's SOCs before proceeding with architecture to handle the big security-related data and the analytical techniques to derive the desired values.

2.4.1 Event Logs

Logs represent events generated by a computing system ranging from operating systems to applications and containing information about the actions.

Different types of systems within an organization can generate logs, as shown in Figure 2.2, which depicts possible security solutions installed in an enterprise. Each system can have its corresponding logs.

Logs can be found in various forms throughout an enterprise, ranging from device logs such as IoT devices and physical access controls to application-specific logs on servers. Although the list of logs can be endless, the following are general categories of the most relevant logs of interest for SOCs.

- *Endpoint (host-based) Logs*: Devices such as laptops, servers, mobile devices, and workstations generate a variety of logs, ranging from the operating system to the applications installed. These logs can contain information about the status of the device as well as the activities performed on the device. They are essential for threat detection and incident response activities as they provide visibility into the behavior of devices within an organization. The information captured via these logs can include details of user activities, network connections, file access, system events, etc. Examples of endpoint logs include Windows Event Logs and Endpoint Detection and Response (EDR) logs.

- *Networking (Network-based) Logs*: In a typical enterprise, there are a number of networking devices, such as switches, routers, and load balancers, that generate logs detailing the network traffic flow. These logs are referred to as network-based logs. Some of the information observed in these logs includes the sources and destinations of IP addresses, users, protocols, and traffic volume. Proxy and DNS logs are considered to be among the most valuable sources of networking logs.

- *Security Tools*: As shown in Figure 2.2, one can see that there are many solutions and tools across the enterprise landscape that could generate event logs. These tools include but are not limited to firewalls, IDS/IPS (Intrusion Detection/Prevention System), EDR (Endpoint Detection and Response), and Access Controls. Due to their focus on possible intrusions, these logs are considered to be one of the most critical sources of data in the enterprise security landscape.

2.4.2 Inventory

Effective inventory management is vital to the success of any enterprise, as it enables organizations to efficiently monitor and manage their digital assets and resources. However, maintaining up-to-date inventory lists is increasingly challenging in today's fast-paced business environment. Nevertheless, a robust inventory management system facilitates enhancing security operations within an enterprise. For example, with accurate knowledge of all critical assets, such as the most vital servers or users known as *crown jowls*, organizations can prioritize detection efforts and associate higher risk factors with corresponding alerts.

Inventory data that is valuable to the SOC operations include:

- *Identity*: User's information and their link to assets. Common sources of identity data include Identity and Access Management (IAM) Systems, directories, Enterprise Resource Planning (ERP) systems, and Microsoft Active Directory (AD).

- *Asset*: Information about the assets and their details, such as device information, the owner, location, software running, and their versions. Asset details can be obtained from tools such as Configuration Management Database (CMDB) and Network Access Control (NAC).
- *Vulnerability Scans*: results of the vulnerability scanners and management tools.
- *Network Diagrams*: Information regarding asset connectivity and separation across various networks, VLANs, subnets, and permitted connections.

2.4.3 OSINT/CTI

Open-Source Intelligence, commonly abbreviated as OSINT, refers to any information that is gathered from publicly available sources, providing context to a given situation. This type of information can range from the basic context surrounding a domain name or IP address (e.g., ASN, registrar) to more complex scrapes of darknet forums in search of the latest identity leaks or zero-day exploits.

The potential of OSINT to enhance the decision-making processes of Security Operations Centers (SOCs) is immense. For instance, by monitoring social media platforms such as Twitter, organizations can stay ahead of the curve in identifying the latest vulnerabilities [224]. This information can then be correlated with the organization's asset inventory to determine whether they are at risk of exploitation through these vulnerabilities. Furthermore, collecting intelligence on threat actors of interest and their typical tactics can help organizations prepare and prevent attacks from occurring.

Furthermore would like to define Cyber Threat Intelligence (CTI) as a subset of OSINT that can aid particularly with the threat detection tasks (e.g., list of Indicators of Compromise such as malicious domains or threat TTPs) [42].

There are several curated OSINT/CTI resources maintained by the open-source community that provides a comprehensive list of valuable sources to gain intelligence for cybersecurity [121, 124, 241].

2.5 Data Collection within SOC

As previously mentioned, there is an abundance of cybersecurity-related data that can be collected and utilized to derive value. However, it's not feasible to collect all available data. Therefore, SOC teams must carefully evaluate and select which data sources to use, consider how they can overlap, and identify the insights that can be extracted from them. Addressing these questions beforehand is crucial to ensure optimal results, particularly for tasks like detection.

In identifying the need and prioritizing data collection strategies, there are generally two approaches that can be taken: a *problem-mapping mindset* and a *capability-assessment* mindset.

When approaching the problem-mapping mindset, it's essential to define the problem, perform threat modeling, and gain an understanding of the context surrounding the data collection and its purpose. To illustrate this approach, let's consider a hypothetical scenario. Through our threat intelligence investigation, we have determined that the most significant threat actors targeting our organization use targeted phishing emails to gain an initial foothold, specifically targeting IT employees with developer rights. Through gap analysis and simulations such as Red Teaming exercises, we have identified a lack of detection and visibility, particularly regarding email security solutions and insight into the emails received by our employees. Now that we have defined the problem understood its limitations, and the need for a solution, we can deploy tools to gain visibility into email headers sent and received by employees and develop data-driven solutions to detect targeted phishing emails, such as analyzing the reputation of the sender and attached URLs.

In the capability assessment mindset, the focus is on first understanding the strengths and capabilities of our current setup, such as identifying all available security tools and the data they can generate. Then, we can explore ways to supplement this data, reduce blind spots, filter unnecessary data, and consolidate overlapping data sources. Let's consider a scenario to illustrate this approach. After conducting our analysis, we discovered that while we have visibility into endpoint activities through EDR and network connections via proxy logs, we lack visibility into DNS requests and responses. We also know that DNS is a commonly misused protocol in many adversarial techniques, and although we may not be addressing a specific problem at this point, we recognize that DNS data could be valuable for further analysis. Therefore, we prioritize collecting DNS data to complement our visibility and allow better future data-driven approaches to tackle specific problems.

While they are both very similar in practice, they lead to different conclusions.

2.6 Data-Driven SOC

Having established a process to prioritize and collect the most valuable data to develop impactful data-driven solutions for critical security operations, we now require a process for identifying and proposing use cases to tackle specific SOC problems.

The term *use cases*in this context is a vernacular for the approach to meaningful ways to leverage the underlying data to derive a particular value for a particular problem.

In this regard, one can take different angles when attempting to develop use cases. More specifically, problem-mapping, data-mapping, and algorithm-mapping mindsets.

Although it is beyond the scope of this thesis to cover all potential problems, data, and algorithms that can be addressed through a data-driven approach, we present a few examples to provide a basic understanding. For more information, refer to [63, 119, 228, 229, 262].

2.6.1 Problem Mapping

This is a similar concept to problem mapping in data prioritization, here where we aim to comprehend the critical business needs and prioritize the most impactful problems.

Example of high-level SOC-related problems that can be tackled using data-driven approaches includes advanced threat detection, which involves identifying unknown threats, insider threats, and data exfiltration; prioritizing vulnerability management; improving alert functionality through correlation and prediction; detecting policy violations and misconfiguration; and vulnerable employee detection.

2.6.2 Data Mapping

The aim here is to understand the data and identify all potential security-relevant values that can be derived. Then, prioritize the use cases based on the derived values. For instance, a thorough understanding of all the fields and features within proxy logs can enable the development of use cases such as command-and-control beaconing detection or user behavior analytics to identify policy violations.

2.6.3 Algorithm Mapping

The purpose of algorithm mapping is to incorporate learnings and algorithms from various domains that have proven successful in addressing a wide range of challenges and applying them to SOC-related challenges. For example, draw from social networks and graph theory and apply the hypothesis of guilt-by-association to detect new variations of previously known malware (Chapter 4). Another example is utilizing natural language processing (NLP) to summarize alerts. Data mining techniques that can be adopted for cybersecurity include statistical analysis, graph inference, time-series analysis, clustering techniques, outlier detection, supervised and unsupervised machine learning, and NLP, as outlined in [63].

3 NextGen SIEM

Upon examining the variety, velocity, and volume of the data available in today's SOC, we quickly realize that we are in the realm of big data. In practice, in a medium-sized company, millions of systems produce data at the scale of trillions reaching easily terabytes of data per day.

The advent of Big Data presents significant challenges, particularly regarding storage, processing, and accessibility. Therefore, it is expected that tools such as SIEMs will efficiently handle this Big Data to provide necessary capabilities like detection, correlation, and investigation.

Many of today's organizations leverage big data pipelines and analytics to make data-driven decisions, e.g., LinkedIn [250], Facebook [261]. Taking advantage of the emergence of distributed storage systems such as HDFS and Kafka and distributed processing frameworks like Map Reduce, Hadoop, Hive, and Spark.

Despite the evolving needs of the cybersecurity industry, many organizations still depend on SIEM solutions that were originally created to fulfill compliance obligations and offer security visibility as required through event management. Although these solutions are currently being employed for more extensive purposes, including threat detection, hunting, and analytics, their technology has not progressed to keep up with these requirements, and they continue to operate as basic search engines without the capacity for performing large-scale processing to support data mining at scale.

This limitation prevents today's SIEM from embracing data-driven approaches. Some consider XDRs [90] as the evolution of EDRs to integrate many sources, fulfill threat-centric use cases, and replace SIEM in that regard. However, while XDR looks promising at first, the trend needs more time to prove itself as truly the future of today's SIEM rather than yet another buzzword [141].

Regardless, SIEMs are indispensable tools within organizations[3] due to their significant role in facilitating SOC activities [10]. This makes them a prime candidate for next-generation big data platforms that enable data mining at scale.

This chapter establishes the background knowledge, requirements, and reference architectures to build next-generation SIEM systems that embrace big data architectural patterns to enable data-driven security operations at scale.

Chapter Contribution

The main contributions of this chapter are summarized below:

- **SIEMA RA**: Establishing a Reference Architecture (RA) for next-generation SIEM platforms that incorporate advanced analytical capabilities, henceforth referred to as Security Information, Event Management, and Analytics (SIEMA).
- **SIEMA Implementation**: Providing valuable insights gained during the implementation and deployment of next-generation SIEMA both in academic and real-world settings.
- **AA Beaconing Detection**: Presenting beaconing detection as a case study to underscore the necessity and worth of SIEMA systems with Advanced Analytical (AA) capabilities.

Chapter Structure

This chapter begins by providing the necessary background knowledge to understand big data architectures and design patterns (Section 3.1). Subsequently, it explores the requirements and design patterns required to bring big data platforms to the cybersecurity domain and build next-generation SIEM systems capable of advanced analytics while also supporting traditional SIEM capabilities (Section 3.2). With the established requirements, the reference architecture required to build such systems is described in Section 3.3. Section 3.4 illustrates the experimental setups used in two scenarios, an in-house research workbench and a real-world enterprise setup. Followed by Section 3.5 providing a case study, highlighting the need for next-gen SIEM with advanced analytical capabilities. Finally, the chapter concludes with lessons learned and future directions for research (Section 3.6).

3.1 Background: Big Data Architectures

In this section, we will explore some fundamental principles of Big Data architectures and best practices before introducing our proposed reference architecture, taking leanings from [71, 139, 168, 272].

The data available to capture come in different structures and formats. Data engineers tend to categorize the data structures into:

- *Structured data*: This is data that has fixed fields with a fixed schema for every record. The biggest advantage of such data is that it is clearly defined, which enables better storage, processing, and accessibility procedure. However, the downside of structured data is the need for a fixed schema, which can be challenging to evolve if there are changes in format or new fields added over time. SQL databases are an example of structured data.
- *Unstructured data*: Unstructured data refers to data that does not have a specific format or structure, e.g., text, images, audio, video, etc. Unlike structured data, unstructured data is not stored in a traditional database, and it can be challenging to analyze and interpret using traditional data analysis tools.
- *Semi-structured data*: Semi-structured data is a type of data that has some organizational structure but does not fit into a traditional relational database model. Unlike structured data, semi-structured data does not have a rigid schema, but it contains some tags, metadata, or markers that provide a certain level of organization. Examples of semi-structured data include XML and JSON.

The majority of data available in the realm of Security Operations is semi-structured, with numerous formats falling under this category. Examples of such formats include Comma-Separated Values (CSV), Syslog, JSON, Parquet, Avro, Protobuf, and Pickle. Additionally, vendors have created specific data formats to add structure to the semi-structured nature of event logs, with examples including Windows Event Logs, Common Event Format (CEF), Common Log Format (CLF), and Extended Log Format (ELF).

3.1.1 Big Data Storage

In order to enable data-driven solutions to security operations, one has to consider data storage, particularly considering the different access patterns and utilization of the big underlying data.

OLTP vs. OLAP

OLAP Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP) are two different types of database systems mindsets. OLTP is a type of database system used to process short, simple transactions in real time. The focus of OLTP systems is on fast and accurate data entry, updates, and retrieval, with a high level of concurrency and transaction processing. OLAP is a type of database system that is designed for data analysis. OLAP databases store large amounts of historical and aggregated data and provide the ability to query and analyze the data flexibly and dynamically. OLAP databases are optimized for read-intensive operations.

Although OLTP and OLAP have been in use for over four decades and have been replaced by newer terms such as data lake house, the concepts, and ideas behind them are still relevant today. There are distinct needs for transactional queries and analytical queries, each with its own specific requirements for insert, update, and access.

While these terms are more than four decades, outdated, and replaced with tending topics such as data lake house, the concepts and ideas behind them are still relevant today. More specifically, even in the most recent data pipelines and storage systems, there are distinct needs for transactional and analytical queries, each with its own specific requirements for insert, update, and access.

Scaling with Big Data

When it comes to big data, we need a way to scale as the data grows. There are two common approaches to scaling a system: scaling up and scaling out.

Scaling up, also known as vertical scaling, involves increasing the resources, such as CPU, memory, and storage of a single machine. This can be achieved by upgrading the hardware components of the machine or by adding more resources to it. Scaling out, also known as horizontal scaling, on the other hand, involves adding more machines. This approach involves distributing the workload across multiple machines or nodes.

Both scaling up and scaling out have their advantages and disadvantages. Scaling up is simpler and requires less coordination; it can be limited as there is a cap on how much we can upgrade the resources of a single machine. Additionally, the cost of upgrading a single machine can grow exponentially. In contrast, scaling out can be more complex and requires coordination between multiple machines, introducing challenges associated with distributed systems. However, scaling out can provide virtually unlimited scalability and high availability.

CAP & BASE Theorem

The Consistency, Availability, and Partition tolerance (CAP) theorem, also known as Brewer’s theorem, expresses a triple constraint related to distributed storage systems. It states that a distributed storage system running on a cluster can only provide two of the following three properties:

- *Consistency*: stating that a read from any node in the system must result in the same data.
- *Availability*: stating that a read/write will always be acknowledged in the form of a success or a failure, e.g., not a time-out.
- *Partition tolerance*: stating that the distributed storage system can tolerate communication outages that split the cluster into multiple silos and can still service read/write requests

If consistency (C) and availability (A) are required, available nodes need to communicate to ensure consistency (C). Therefore, partition tolerance (P) is not possible. If consistency (C) and partition tolerance (P) are required, nodes cannot remain available (A) as the nodes will become unavailable while achieving a state of consistency (C). If availability (A) and partition tolerance (P) are required, then consistency (C) is not possible because of communication requirements between the nodes, i.e., the database can remain available (A) but with inconsistent results.

On the other hand, partition tolerance (P) must always be supported by a distributed database by definition; therefore, CAP is generally a choice between choosing either C+P or A+P. The requirements of the system will dictate which is chosen.

In practice, CAP is usually implemented as BASE - Basically Available, Soft state, and Eventual consistency. BASE describes the storage system that is *basically available* by responding to a client's request, either with the requested data or a success/failure notification. *Soft state*, which indicates that data may change due to eventual updates for consistency. *Eventual consistency* means that a write may not immediately propagate to all nodes; hence no guarantee of immediate consistency after a write. In other words, the system will be available but maybe remain in a soft state until achieving eventual consistency.

The constraint of the BASE model makes it highly desirable for data lakes and OLAP systems. However, it is not suitable for transactional systems where consistency is critical.

Distributed Storage

Understanding the CAP and BASE theorems enables individuals to comprehend the fundamental decisions and design choices necessary for creating distributed storage systems.

Additionally, there are several concepts worth noting during the development process.

Append-only— A type of distributed storage system that realizes the BASE theorem for a highly scalable distributed storage system that emphasizes performance by enforcing an append-only mindset. One of the biggest challenges with distributed storage systems is to maintain consistency across multiple nodes, particularly with updates, as they can introduce conflicts or locking (hence reducing performance). Append-only storage systems overcome this challenge by only allowing data to be written, pushing the complexity of consistency to a later stage. This way, multiple nodes can write data to the database simultaneously without the risk of conflicts or locking. To achieve consistency, the expectation is that the reader will calculate the most appropriate read from a set of entries. For example, instead of updating a variable x from 2 to 3 in a regular database and having it propagated (hence sacrificing performance), in an append-only database, two entries are added (x=2 and x=3). Upon reading, the reader finds the most recent value of x (which is 3) according to logic, such as the write timestamp.

Sharding—The process of horizontally partitioning a large dataset into a collection of smaller, more manageable subsets called shards. These shards are then distributed across multiple nodes. Each shard is stored on a separate node, and each node is responsible for only the data stored on it. All shards collectively represent the complete dataset. By partitioning data into shards, processing loads can be distributed across multiple nodes, thereby improving read/write times.

Replication— The process of storing multiple copies of a dataset, known as replicas, on multiple nodes. Replication provides scalability and availability due to the fact that the same data is replicated on various nodes. Fault tolerance is also achieved since data redundancy ensures that data is not lost when an individual node fails. There are different methods of implementing replications: master-slave and peer-to-peer.

Hadoop, an open-source framework designed for large-scale data storage and processing on commodity hardware, is an excellent real-world example of a distributed storage system that incorporates all the principles mentioned above. It has established itself as a de facto industry platform for contemporary Big Data solutions.

ETL

Extract Transform Load (ETL) is a process of loading data from a source system into a target system. The source system can be a database, a flat file, or an application. Similarly, the target system can be a database or another storage system. ETL pipelines are useful in practice to load the data from a variety of data sources to a centralized data lake.

3.1.2 Big Data Processing

Having an understanding of big data storage systems and practices. It is also essential to grasp some fundamentals of big data processing fundamentals.

Processing can be scaled both vertically and horizontally, similar to storage. However, when it comes to big data, it is crucial to enable horizontal scaling for better processing power. This introduces challenges of how to distribute processing (task) across multiple nodes.

Parallel & Distributed Data Processing

Parallel data processing involves the simultaneous execution of multiple sub-tasks that collectively comprise a larger task. The goal is to reduce the execution time by dividing a single larger task into multiple smaller tasks that run concurrently. Parallel processing is usually associated with a single machine (i.e., better utilization of multiple processors or cores). Distributed data processing is similar in mindest, but rather, focuses on the distribution of tasks to multiple nodes forming a cluster.

Batch & Stream Processing

Big data processing usually divides into two types: batch and stream processing.

Batch processing refers to the execution of a set of tasks or jobs in a batch or batch jobs, typically performed in offline mode, which in turn results in high-latency responses. Batch workloads typically involve large quantities of data with sequential read/writes and comprise groups of read or write queries. OLAP systems commonly process workloads in batches. MapReduce is a widely used implementation of a distributed batch-oriented processing framework. It is based on the principle of divide-and-conquer, dividing a big problem into a collection of smaller problems that can each be solved quickly and separately. In contrast, stream processing is a real-time data processing technique that continuously processes a stream of data as it arrives. Stream processing is commonly used for data processing tasks that require near-real-time processing. Additionally, there are variations between the two extremes, such as micro-batch processing, which aims to bridge the gap between batch processing and stream processing.

Lambda Architecture

Lambda architecture[137] is a data processing architecture designed to combine batch processing and stream processing and building Big Data systems as a series of layers; each layer satisfies a subset of the properties and builds upon the functionality provided by the layers beneath it. More specifically, in lambda architecture, data is processed through two parallel pipelines: batch processing and stream processing. The batch processing pipeline performs time-insensitive computations on large batches of historical data, generating immutable batch views of the data. In contrast, the stream processing pipeline processes real-time data as it arrives, generating real-time speed views of the data. The batch and stream processing pipelines' outputs are then combined in a serving layer, creating a unified view of the data. This enables users to query and analyze the data using the same tools and APIs, regardless of whether the data is historical or real-time. The system is designed to tackle the challenges of distributed storage in terms of eventual consistency while considering the processing aspect.

There are several other architectures that are widely recognized for big data processing, including the kappa architecture[137], data lake architecture[114], event-driven architecture[170], and lakehouse architecture[17].

3.2 Requirements

Before proceeding to the reference architecture, it is crucial to outline the added requirements for the next-gen SIEM system. We categorized the main requirements into two groups: Business requirements (BR) and Architectural Requirements (AR).

BR1. Data Science (Advanced Analytics)

One of the main limitations of traditional SIEM systems is their reliance on signature or heuristic-based threat detection, limiting the detection only to previously known threats. Finding truly unknowns requires the utilization of state-of-the-art data science (statistics, machine learning, and data mining algorithms). In this regard, the platform should acknowledge state-of-the-art data science tools and techniques [113].

Data science can also be used to eliminate static rules which pose high false-positive rates. For example, instead of looking at 10 authentication failures within 1 minute (i.e., attempt to find brute force attacks), one could learn the threshold per user and endpoint, thus reducing false positives.

BR2. Data Engineering (Complex Data Processing)

As different use cases may require different shapes of data, the platform should be able to handle data engineering pipelines. This can include data aggregation, correlation, enrichment, normalization, parsing, validation, tagging, duplication, and transformation. In this regard, a next-gen SIEM should allow data scientists, data engineers, and SOC analysts to seamlessly develop, combine, manage, and maintain different data processing pipelines.

AR1. Distributed, Scalable, And Fault-tolerant

A next-gen SIEM is expected to handle big data with 3Vs: large *volume*, high rate of generation (*velocity*), and heterogeneity of the types of structured and unstructured data (*variety*). Hence, to cope with the volume, velocity, and variety of data produced by today's enterprises, the platform should be scalable and elastic. To achieve this, the best practices in distributed systems (e.g., distributed storage and processing) should be followed and adhered across all architectural levels of the platform.

In a distributed setting, availability and resilience to failure become challenging yet crucial aspects of the system. In this regard, the platform is also expected to be fault-tolerant and available during a failure/outage, such as network outages or hardware failures.

AR2. Extensible

Today's technology landscape is evolving faster than ever. The most relevant technologies or solutions of today may be irrelevant in a few years. A next-gen SIEM should be able to undergo numerous modifications and extensions to stay relevant in an ever-changing technological world, e.g., able to adopt a new distributed processing framework.

AR3. Open

Today's open-source community is very active and often ahead of its commercial competitors. Therefore, the platform needs to respect open-source solutions and technologies by allowing the adoption of open-source. This will ensure the system's relevance with state-of-the-art technologies.

Furthermore, one of the main criticisms of today's legacy SIEMs is their locked-in data model. A next-gen SIEM should have an open data model respecting the users and data portability.

AR4. Integration

The platform should have standard methods to interface and integrate with other external tools or systems via APIs. This allows other tools to appreciate better the values provided by the next-gen SIEM.

AR5. Data Lake for All Storage Requirements

Storage is a core aspect of a next-gen SIEM. Different use cases require different storage systems, from a relational database to a distributed file store. Particularly, to enable advanced analytics, new data lake architectures are needed.

AR6. Modular Data Ingestion

A next-gen SIEM is expected to ingest a variety of data. These data can be from external sources, such as vulnerability data, indicators of compromise, and related OSINT. It can also be from internal sources, such as event logs from network and security systems (e.g., Intrusion detection systems, endpoint security, firewalls, VPN, proxy, DNS), applications, endpoints, assets, network topologies, security configuration, and policies. Thus, a next-gen SIEM is expected to ingest data from both external and internal sources. The ingestion should mainly expect authenticated incoming data and the possibility of crawling or collecting, e.g., to crawl related OSINT.

AR7. Security and Privacy

Big data is becoming a critical resource for many organizations, bringing it into the spotlight as a high-value target for cyberattacks. A next-gen SIEM is expected to guarantee security (security by design).

More specifically, *data management*: addressing how big data is collected and identified to ensure the scope of what to protect is properly assessed and understood. *Identity and access management*: enforcing access control, not only for users, i.e., who can access the data but also ingestors, i.e., validating where the data is coming from an authorized source to avoid including modified or unwanted data. Furthermore, applying the design patterns such as privilege separation, the least privilege principle, access, and audit logging. *Data protection and privacy*: the effort in protecting the big data, such as encryption at rest, as well as addressing any privacy concerns, e.g., enforcing anonymization or pseudonymization. *Infrastructure security and integrity*: securing data as it is collected and moved between systems, i.e., encryption at transit. Furthermore, securing the platform's components."

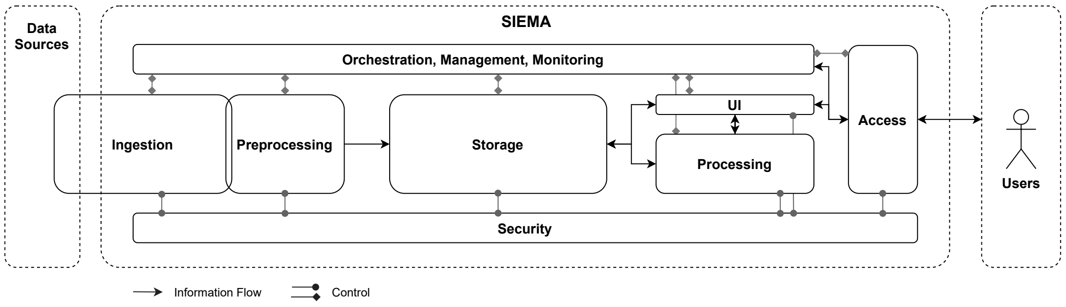


Figure 3.1: Module Decomposition of SIEMA Reference Architecture

The Cloud Security Alliance (CSA) Big Data Security Working Group identifies the top ten security and privacy challenges that need to be addressed for big data[179], and Ajit [85] has proposed recommendations to tackle the security challenges in a big data environment.

3.3 Reference Architecture

We design our RA based on decade-long experience and knowledge revolving around the best practices in designing big data architectures and pipelines, e.g., LinkedIn [250], Facebook [261], and other reference architectures [138]. Figure 3.1 shows the high-level reference architecture for the proposed SIEM. Figure 3.2 illustrates the system workflow consisting of five main stages.

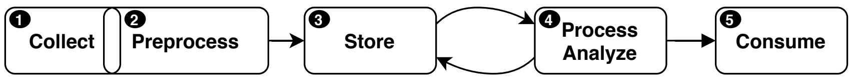


Figure 3.2: SIEMA proposed workflow

Data Ingestion and Collection

This layer is expected to consist of a collection of extendable ingestors and collectors, each designed to ingest a particular data source. Data sources can be *internal* data, such as event logs, telemetries, and inventories, or *external* data, such as OSINT and vulnerability data.

These modules can either accept authenticated data being forwarded (push model) or pull particular data points. Data being forwarded can either come directly from the data sources or be forwarded by a remote ingestor node. The pull mechanism is expected to be limited due to the lack of a dedicated agent.

Pre-Processing

The next stage in the system is pre-processing, i.e., data validation, cleansing, optimization (e.g., de-duplication), parsing, and basic transformation (e.g., standardizing the timestamps), and basic enrichment/tagging (e.g., tagging the events according to their type and source).

All pre-processing steps are expected to be simple, efficient, and scalable. Further enrichment and correlation are expected to be carried out by the analytic/processing modules.

After pre-processing, the data shall be dumped on messaging queues or directly to file storage, where the primary data processing pipelines could take the lead for more advanced data processing (e.g., normalization), storage (e.g., ETL, indexing, etc.), or analytics.

Storage

Similar to the data lake architectural pattern, storage is a key and fundamental component of RA. The main objective of the data storage layer is to provide reliable and efficient access to persisted data. Part of this is to offer multiple representations for single data records to accommodate different use cases, e.g., OLAP-style data analytics, OLTP queries, or string searches.

Note that the storage system of such next-gen SIEM is expected to be highly scalable and agile. This is integral, as organizational and business needs change over time, calling for adjustments in technologies and environment setups.

One can categorize storage needs based on latency, throughput, access patterns, and data type. Some examples of storage needs in the context of a next-gen SIEM are: NoSQL transactional database for results. Index store and search engine to allow string search on event logs for efficient ad-hoc threat hunting and investigation. A Distributed file/object-store to satisfy data lake requirements for advanced and distributed analytics and machine learning models. A queuing mechanism to enable the reliable transmission of data across different processing layers. For instance, a messaging queue that enables an enrichment module to enhance the result of an anomaly detection algorithm, i.e., outliers are pushed to a queue, which the enrichment module is subscribing to, enriching the outliers with related contexts. An in-memory caching mechanism allows multiple executors or workers within a processing module to exchange data efficiently. A graph database to store the relationships between various internal and external entities.

Processing

The Processing layer is responsible for efficient, scalable, distributed, and reliable processing. At a high level, it can serve two main purposes: *data engineering* and *data science*. The data engineering sub-layer is responsible for data processing and transformation, e.g., event correlation and enrichment, normalization, ETL pipelines, storage optimization (compression, partitioning, bucketing), pattern matching, etc. On the other hand, the data science-based modules are concerned with knowledge extraction from the data, e.g., machine learning, data mining, statistical analysis, graph analytics, etc. It is worth noting that analytical processing can be interactive, batch, and stream.

Access

The access layer is the interaction point of the system with external actors. These actors can be SOC analysts, data scientists, data engineers, administrators, managers, or external APIs. Each actor is expected to require interaction with a specific part of the platform for a particular reason. The access layer is responsible for managing these interactions while ensuring security and load balancing. For example, a SOC analyst requires an interactive interaction with the platform's search capabilities to investigate threats, define rules for the correlation engine and dashboard to view the alert and visualize the trends.

UI

The user interface layer is responsible for abstracting actors' interactions with storage or processing modules. For instance, a UI that enables SOC analysts to run their query against the storage system or the data science notebooks designed to allow the data scientist to interactively analyze data loaded from the storage layer in the processing layer.

Orchestration, Management, and Monitoring

This layer has three main responsibilities: orchestration, management (administration), and monitoring.

The orchestration module is responsible for providing configuration, management, and coordination between the various platform layers and their modules. This includes job submission, collectors' configurations, data engineering pipelines, etc. In addition, this module is also expected to provide monitoring capabilities for every module within each layer, e.g., monitoring the analytical jobs and their status.

The administration/management module is responsible for the configuration, provisioning, and control of the underlying infrastructure and the platform itself, e.g., managing the underlying storage system.

Lastly, the platform auditing module supports the health monitoring of the system and its underlying heterogeneous systems. For instance, it is expected that the storage layer will consist of multiple systems, e.g., distributed files system, NoSQL database, and a messaging queue. In this regard, this module should allow administrators to monitor the health and performance of these systems.

Security

Given the nature of our next-gen SIEM system, there are concerns about the security and privacy of such big data platforms. In this regard, this layer is responsible for the security of the platform and its underlying data. This includes enforcement of access rules, restricting access based on classification or need-to-know, and securing data at rest or in transit.

3.4 Implementation and Deployment

We have endeavored two implementations of the proposed RA, an in-house academic research workbench and a real-world experimental setup in an international enterprise's infrastructure.

3.4.1 In-house Research Workbench

Our first attempt to develop and deploy a SIEMA system according to the proposed RA was made on a cluster consisting of two Dell PowerEdge (R730, R820) and five Fujitsu Primergy RX600 with a total of 1,864 GB RAM, 24 CPUs (200 total cores), and 4 TB storage interconnected via 10 Gb optical fiber. In addition, an external Network Attached Storage (NAS) connected to the cluster via 3x 10Gb optical fiber. Table 3.1 presents the leading underlying technologies used for this setup.

The platform was utilized during multiple successful research to apply different data mining and machine learning approaches to the problem of malicious Domain/IP detection using proxy and DNS logs, resulting in multiple publications [186, 187].

Table 3.1: In-house SIEMA as a research workbench.

Technology	Reference	Usage
Kubernetes	Orchestration/Management	Backbone system and orchestrator.
Ansible	Orchestration/Management	Platform operation and administration.
Zookeeper	Orchestration	Configuration maintenance and synchronization.
Apache Spark	Processing	Distributed processing and analytics engine.
Presto	Processing	Distributed processing (SQL query engine).
Apache Livy	Management	Multi-tenancy and job management.
Apache Kafka	Storage	Distributed messaging and queueing.
Hadoop HDFS	Storage	Distributed file system and object-store.
Elasticsearch	Storage	Search engine, index store.
Prometheus	Storage	Time series database for metrics regarding the platform health.
HBase	Storage	Distributed NOSQL data store on top of HDFS.
Apache Nifi	Processing, UI	Orchestration and data preprocessing.
Kibana	UI	UI for interaction with the search engine.
Grafana	UI	UI for platform health monitoring.
Zeppelin	UI	Notebook for ad-hoc data science.
CMAK	UI, Orchestration	Cluster manager for Apache Kafka
Hue	UI	Hadoop interface.

3.4.2 Real-world Enterprise Setup

We also had the opportunity to explore a SIEMA system in a real-world setting with a large international company with quite a mature cyber defense. This company had a cloud-based legacy SIEM continuously utilized for threat hunting, monitoring, and rule-based threat detection. We attempted to build around it with analytical capabilities to explore the potential values. We utilized available cloud-based services compliant with the company's policies, such as Azure Data Factory, Azure Data Lake Storage, and Databricks, to enable advanced data engineering and science.

The first significant value of this platform was the ability to run basic aggregation, correlation, and statistical queries over larger time frames (over 100 terabytes of data) which would not be possible with traditional SIEM systems as they were designed for only interactive investigation and searches.

The next value was the ability to create successful advanced use cases using the underlying data (EDR, proxy, DNS). Examples of such use cases are beaconing detection, malicious processes detection, suspicious DNS and proxy requests, windows logon anomalies, and user behavior analytics over weeks of data. The result of these use cases led to the rise of multiple incidents missed by traditional rule-based detections.

Lastly, the ability to train a model to rate and prioritize traditional SIEM alerts according to past experiences. Most of today's organizations receive 17, 000 alerts per week; more than 51% of the alerts are false positives, and only 4% of the alerts get adequately investigated [108]. Therefore, prioritization of alerts can help SOC analysts to focus their efforts better [10]. This was the last use case, designed to read past investigated alerts, their artifacts, and the associated responses (thus labels) to train a model that attempts to prioritize the new alerts according to their potential to be true positives. This prioritization was achieved by adding a confidence score to the alert passed to the traditional SIEM dashboard. The initial impression and qualitative evaluation of this use case seemed promising.

3.5 Case Study: Beaconing Detection

To better understand the need for advanced analytical capabilities within today's SIEM, we decided to prepare a simplified experiment performed on our real-world setup. Particularly a heavy yet straightforward use case of beaconing detection.

One of the characteristics of sophisticated cyber threats, such as Advanced Persistent Threats (APTs), is periodic attempts to reach out to the command and control (C&C) infrastructure controlled by the adversary to receive further instructions. Such heartbeat and callback behavior is known as beaconing.

Malware beaconing is typically characterized by two main configurations: sleep time and jitter (variations from central value). The beaconing frequency can vary from slow and stealthy to fast and aggressive (from a few seconds to hours or even days of sleep time). Nevertheless, generally, the adversaries are expected to maintain regular beacons for better visibility and control of the infected machines [109].

While at first glance, beaconing detection seems simple, it is quite challenging:

Temporal Analysis in Big Data: To detect beaconing, one has to analyze the traffic behavior of all source and destination pairs over an extended period of time. This makes beaconing detection a big data problem.

Intentional Randomness and Jitter: One of the other challenges with beaconing detection is the adversarial strategies to hide the beaconing behavior. One of the common ways the adversaries attempt to prevent detection is by varying the sleep time to make it appear as normal traffic. Other methods can include omitting certain beacons or injecting additional random beacons.

False Positives (Benign Applications): While we discussed the maliciousness of beaconing behavior, there are several scenarios in which beaconing is an integral part of communication and does not indicate maliciousness. For instance, Network Time Protocol (NTP), automated software patching, mailing clients, updates, or keep-alive traffic in long-lived sessions may also appear as beacons.

External Factors: There can be unanticipated external factors that can introduce errors while looking at the periodicity, such as the host (endpoint) going offline or network interruptions.

3.5.1 Detection Approach

Beaconing detection has been studied widely in the literature [87, 109, 234, 264], and while there are many ways to develop a beaconing detection approach, here we present one of the simplest ones using statistical methods.

(i) **Data Preparation**: We start by pre-processing the network connection events keeping only the source (host unique identifier), the destination (e.g., IP address and port), and the timestamp fields. One could also validate to ensure the destinations are valid and timestamps are in Unix-Timestamp format.

(ii) **Delta Time Calculations**: Next, we group connections by source and destination and sort them by their timestamp. This will allow us to calculate the time deltas between connections of each source and destination pair. For instance, if host *H1* connects to destination *D1* at the time *t1* and *t2*, the time delta between these two connections is $t2 - t1$.

(iii) **Clean Time Deltas**: To ensure the detection quality, we need to filter out bad entries, e.g., border time delta (nulls) - indicating no previous connections, or time deltas equal to zero - indicating network issues resulting in multiple connections in a very short time.

As mentioned, one of the challenges with beaconing detection can also be external factors such as network interruptions or the host going offline (host shutting down). We tackle this challenge by filtering out outliers in time deltas for each source and destination pair using Interquartile Range [278]. This ensures that when the host has gone offline, the time delta showing the big gap is treated as an outlier, hence eliminated from further calculations.

(iv) **Periodicity via Average and Standard Deviations**: While there are more sophisticated ways to detect periodicity [67, 68, 217], here we take the simplest approach. We calculate the average and the standard deviation of time deltas for each source and destination pair to estimate the periodicity of connections.

(v) **Destination's Reputation**: To tackle the false positives (i.e., benign applications), we also calculate the prevalence of each destination, i.e., how many hosts (sources) have connected to this destination during the analysis period. This can be achieved by simple grouping and counting.

(vi) **Scoring**: At this point, for each source and destination, we have the average and standard deviation of time deltas, number of connections (beacons), and the prevalence of the destination. One can now define certain heuristics to score the beaconing behavior to prioritize the alerts. For this case study, we simplify our scoring to three main functions:

- *Low Coefficient of Variation*: The coefficient of variation is defined as the ratio of the standard deviation to the mean.

  While a low standard deviation of time deltas means perfect periodicity (almost all time deltas are the same), it does not consider how big the average is. That is why the relative standard deviation (coefficient of variation) can help by looking at the ratio.

  We utilize an exponential function (Equation 3.1) to transform the coefficient of variation into a score. The reasoning here is that all low ratios (an indication of better periodicity and beaconing behavior) should be scored closer to 1, and as the ratio gets bigger, it should have a decaying effect (logarithmic) in the scores, getting closer to 0.

  $$S_{cv} = e^{-\sqrt{\frac{\sigma}{\mu}}} \tag{3.1}$$

  where μ and σ are the average and standard deviation of time deltas respectively, and S_{cv} is the score derived from the coefficient of variation.

- *Low Destination Reputation*: Destinations with high reputations (largely accessed by the majority of the endpoint) typically indicate benignness. That is why we would prioritize those beaconing alerts whose destination is rarely observed. More specifically:

  $$S_{rep} = e^{-\frac{p(dest)}{k}} \tag{3.2}$$

  where $p(dest)$ indicates the prevalence of the destination (i.e., how many hosts have been observed connecting to this destination). k is a numerical constant internally determined based on domain knowledge as the threshold to smooth the curve (i.e., after the value for k, the scores should smoothen as we don't care anymore). For example, if k is set to 100 that means as $p(dest)$ gets closer and passes the threshold of 100 hosts, the score should be low, and there is no significant difference between 300 to 600 to 1000, as we only care for low numbers (e.g., 1 or 10 hosts).

- *Beaconing Consistency*: Sometimes, just the destination prevalence is insufficient to filter out benign applications, particularly updates. In this regard, one could take yet another attempt to eliminate those beaconing-like behaviors. A malicious beaconing could be characterized by its consistency, whereas some benign behavior, such as an update, will only appear for a certain time. Thus, one could analyze the consistency of the beaconing behavior by looking at the ratio of the number of the beacons and their average delta time to the analysis range.

$$S_{con} = \frac{b \cdot \mu}{t_e - t_s} \tag{3.3}$$

where b is the number of beacons, μ is the average, $t_e - t_s$ is the analysis range (e.g., 86400 seconds).

Note that while here we discussed only three simple scoring functions on top of the information available with our case study, one could design multiple other heuristics to reduce the false-positive rate. Lastly, we aggregate the scores via an aggregator function, such as a weighted average, to derive a single score.

(vii) **Alerting**: Having a final score for each source and destination pair, we could sort the alerts descending and take the first k items (where k is set by the rate the analyst can handle). One could also do further analysis for the distributions of the scores to dynamically set k.

3.5.2 Experiment Setup

We carried out our experiment for this case study within the premise of a large international organization. Particularly, we implemented the described methodology for beaconing detection as a use case within the enterprise SIEM system as well as our analytical platform (discussed in Section 3.4.2). Therefore, the implementations were identical in terms of their logic.

While we cannot discuss the details of the traditional SIEM system used by the enterprise due to NDA, we can confirm that the SIEM system is among the top SIEM leaders identified by the Gartner Research Group [126]. Furthermore, the setup is among one of the largest enterprise SIEM setups, designed to handle the ingestion of more than 10 TB per day.

Our analytical platform for this experiment was configured on Databricks with 5 "Standard_D32s_v3" workers. Thus, having a big data platform backed up by Apache Spark with a total of: 640-GB Memory, 160 vCPU Cores, 1280 GB temp SSD storage.

We ran our main experiment on one day of network connections collected from an EDR tool (172.5 million events) which spanned approximately 102GB.

3.5.3 Results

Running the described beaconing detection on the traditional SIEM took 45 minutes to go over 89 million events before reaching the disk usage limit (set by the enterprise) and returning 246 events. This is because traditional SIEMs are not designed for large-scale analysis (i.e., distributed processing). Instead, they tend to aggregate the events of interest into a single server where the calculations occur. In contrast, our analytical environment, supported by Databricks and Apache Spark, was able to go through all 172.5 million events and finish the use case within only 26 seconds.

Although there are many variables in place that make this comparison unfair (e.g., the cluster sizes not being the same, the implementations of a simple calculation such as mean, etc.), one can observe the enormous gap between the capabilities.

One of the other expectations of such analytical platforms is their ability to scale out. Figure 3.3 shows the runtime of beaconing detection as we add more workers to run the use case.

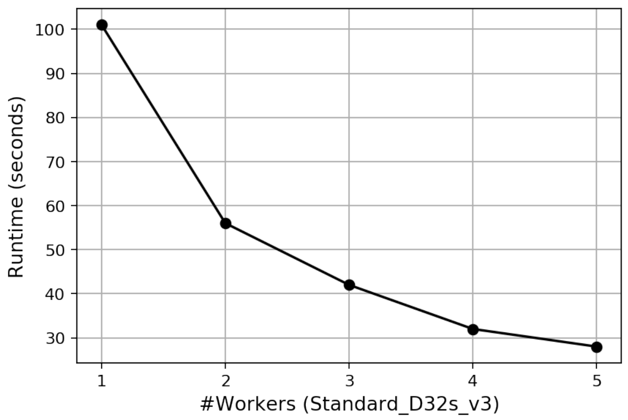


Figure 3.3: Beaconing Detection use case runtime on Databricks based on the number of workers.

Lastly, while we could not run the use case for 5 weekdays on the traditional SIEM, we could run it in the analytical environment. In this regard, the described setup (with 5 "Standard_D32s_v3" workers) could analyze approximately 960 million events (over 550 GB) in 126 seconds.

3.5.4 Discussion and Lessons Learnt

With a simple statistical-based use case, we highlighted that traditional SIEM systems are not designed for advanced analytics. One could imagine how more sophisticated analytics, e.g., machine learning, data mining, and graph analytics, will further challenge traditional SIEMs.

As highlighted by most related work, the ability to run complex data analytics is one of the most critical capabilities required for the next-gen SIEM systems to give us a fighting chance against previously unknown threats.

Nevertheless, we cannot underestimate the need for legacy SIEMs, particularly when investigating incidents and alerts. While the analytical platform will take a long time to search, as it requires touching almost all files, the legacy SIEMs are designed for optimized searching, allowing a SOC analyst to run ad-hoc queries investigating incidents and correlating data on demand. For instance, in our example, while the legacy SIEM took 23 seconds to search the context of one of the alerts, the analytical platform took more than 1 minute. Note that the searches are on one day of data; as the time frame gets bigger, the analytical platform will take even longer (for random searches). Although one could argue that there are ways to speed up the search, e.g., partitioning, bucketing, indexing, and adding meta-data, it still will not be comparable to traditional SIEMs (i.e., index stores) that are designed for optimized searches.

This highlights the need for next-gen SIEM systems that integrate the capabilities of both big data platforms and legacy SIEMs to provide ultimate value to today's SOCs.

3.6 Chapter Summary

In this chapter, we have identified the limitations of current SIEM systems, specifically their inability to perform advanced analytics and incorporate cutting-edge data mining, machine learning, and graph mining approaches. To address this gap, we have introduced the concept of SIEMA (Security Information/Event Management and Analytics), a next-generation SIEM that enables advanced analytical capabilities.

We have provided a reference architecture for SIEMA, drawing on best practices and design patterns from big data architectures and pipelines. Moreover, we have presented our implementation of SIEMA in two different settings: a research workbench using open-source technologies and a version of the proposed architecture deployed in a real-world environment alongside an international organization's traditional SIEM system.

Our work highlights the value of SIEMA's analytical capabilities not only in advancing research in data mining for threat detection but also in developing successful use cases in an industrial setting, leading to the detection of genuine threats and incidents. Finally, we have presented a case study on beaconing detection, which clearly demonstrates the limitations of traditional SIEM systems when compared to those with advanced analytical capabilities.

With SIEMA, it is now possible to address SOC challenges using data-driven approaches shifting from reactive measures to proactive measures.

4 Graph-based Inference for Threat Detection

In a comprehensive empirical analysis, Arp et al. [18] identified common pitfalls that can lead to unrealistic performance and interpretations when applying machine learning in security. These pitfalls obstruct understanding the security problem and highlight the gap between the research community and industry. Although many factors contribute to the skepticism in the adoption of data mining and machine learning techniques in the cybersecurity industry [18], this chapter focuses on two specific challenges: *feature engineering* and the *adversarial domain*.

Feature engineering plays a critical role in data mining and machine learning, particularly in the cybersecurity domain, where the majority of features can be unreliable [10]. The dynamic nature of this fast-growing field can cause older learning to become obsolete quickly. Moreover, in an adversarial domain, an adversary can easily exploit this, for example, by changing a few lines to create a new set of values for a particular feature [247]. To illustrate this challenge, consider an ML algorithm trained to distinguish between legitimate and malicious URLs based on the number of subdomains and URL entropy as distinct features. Although this approach may work initially, it cannot be assumed to perform well under different circumstances. A slight alteration to the URL string can quickly defeat the ML classifier, and broadening the pattern recognition approach can drastically increase the false positive rate. This is one of the main challenges in applying generic data mining and machine learning algorithms in security. In other words, the existence of an adversary that can adapt to defeat detection algorithms cannot be ignored, i.e., adversarial machine learning [271].

Contextualization is another challenge in feature engineering [10]. For example, if an event count is used as a feature for user behavior analytics (UBA), it is crucial to consider the time of day, week, and user when analyzing the feature. Event counts for different users and times should not be compared together.

We refer to such features as local features, which are context-specific, unreliable, and can be easily forged by the adversary. In contrast, global features are harder to change, such as behaviors or meaningful associations among entities [25, 130, 187]. For example, in malicious domain detection, the URL structure is a local feature specific to a single entity. However, IP address resolution or the mapping of ASNs/IP ranges are global features as they examine the association among different entities.

The hypothesis here is that an adversary's resources are limited due to economic constraints. Therefore reuse of infrastructure, Tactics, Techniques, and Procedures (TTPs) are inevitable [186, 187], e.g., usage of the same X.509 certificate, autonomous system, registrar for domains, or even. The other key intuition is that an external entity is less likely to be malicious when it is associated with a large number of benign entities, e.g., if a domain is visited by the majority of the workstations in a company, it is less likely to be a malicious domain [40, 257].

The underlying principles in this chapter can be summarised into *guilt-by-association* [144] and exempt-by-reputation. Unsurprisingly, this set of reasoning is also adopted in SOC or forensic investigation. For example, investigating a potentially malicious domain involves the investigation of open source intelligence and threat intelligence related to that domain, e.g., its registrar, subdomains, connected domains, TI feed observation, etc. In this regard, while an association with malicious entities does not necessarily imply maliciousness, it could indicate higher risk.

This chapter formulates threat detection as a large-scale graph mining/inference problem using data captured within typical SIEM systems focusing on the detection of unknown knowns, i.e., new variants of previously known malicious entities. More specifically, it proposes the construction of a Heterogeneous Information Network (HIN) from SIEM-related logs (EDR, Proxy, DNS) enriched with related Cyber Threat Intelligence (CTI) and Open-Source Intelligence (OSINT). The HIN emphasizes shared attributes across processes/files that accommodate maliciousness inference, such as shared libraries, registry entries, and network connections. Maliciousness is then inferred from a set of previously known malicious entities, i.e., Indicators Of Compromise (IOCs), by aggregating messages in the network, continuously calculating a maliciousness score for each entity considering its prior, neighbors' priors, in-degree, and the influence weights.

Chapter Contribution

The main contributions of this chapter are summarized below:

- **CyberHIN**: Modeling SIEM-related events as a Heterogeneous Information Network emphasizing the most important entities and relationships observed in endpoint logs like EDR and network logs like DNS and proxy logs. It also incorporates relevant Open-Source Intelligence (OSINT) and Cyber Threat Intelligence (CTI) to enrich the HINs. Additionally, the significance of each entity and relationship contained in the HIN or Knowledge Graph is discussed in detail.

- **MalRank**: Proposing a large-scale graph inference algorithm adopted from belief propagation and PageRank algorithm. MalRank is tailored to infer and emphasizes maliciousness using the associations presented in the HIN.

- **MalLink**: Designing and implementing a system to connect to an enterprise's SIEM, ingesting the underlying event logs, generating the proposed HIN, running the MalRank, and outputting a set of previously unseen malware samples with their corresponding maliciousness score.

- **Evaluation and Discussion**: Present the findings and leanings when evaluating MalLink in a large international enterprise landscape, demonstrating the feasibility, realism, and effectiveness of detecting previously unseen malware in a real-world setting.

Chapter Structure

This chapter begins by providing the necessary background knowledge to graph theory and mining (Section 4.1). Section (4.2), introduces the proposed CyberHIN, focusing on two specific scenarios, capturing entities and relationships captured from endpoint-related logs (i.e., EDR) and network events (i.e., proxy and DNS logs). Section 4.3 presents our novel graph-based inference algorithm, MalRank, followed by Section 4.4 and 4.5 detailing our implementation of MalRank and the proposed HIN to build a large-scale system designed for the detection of new threats in a real-world setting (named MalLink). Two case studies are then presented: one which tailors the proposed system for malware detection (Section 4.6) and one which tailors it for detecting malicious domains and IPs (Section 4.7). The limitations of MalRank and our proposed approach are discussed in Section 4.9, while Section 4.10 provides a comprehensive literature review of graph mining applications, with a focus on inference for threat detection in the cybersecurity domain. Finally, the chapter concludes with a summary(Section 4.11).

4.1 Background: Graph-based Inference

In this section, we will explore some fundamental principles of graph theories taking leanings from [143].

4.1.1 Heterogeneous Information Network

▶ **Definition 4.1.** (Information network) [252]: An *information network* is defined as a directed weighted graph $G = (\mathcal{V}, \mathcal{E})$ with an object/entity type mapping function $\tau : \mathcal{V} \rightarrow \mathcal{A}$ and link/relationship type mapping function $\varphi : \mathcal{E} \rightarrow \mathcal{R}$, where each object $v \in \mathcal{V}$ belongs to one particular object type $\tau(v) \in \mathcal{A}$, and each relationship $e \in \mathcal{E}$ belongs to a particular relation $\varphi(e) \in \mathcal{R}$. Note that we explicitly distinguish object types and relationship types in the network. If there exists a relationship from object type A to object type B, denoted as $A \xrightarrow{\text{R}} B$ or simply *A–(R)–B*, the inverse relationship R^{-1} holds naturally for $A \xleftarrow{R^{-1}} B$. Note that R and its inverse R^{-1} are usually not equal unless the two types are the same, and R is symmetric. When the types of objects $|\mathcal{A}| > 1$ or the type of relationships $|\mathcal{R}| > 1$, the network is called a *heterogeneous information network*; otherwise, it is a *homogeneous information network.* ◀

▶ **Definition 4.2.** (Network schema) Sun and Han [252]: The *network schema* or *graph schema*, denoted as $T_G = (\mathcal{A}, \mathcal{R})$, is a meta template for a heterogeneous network $G = (\mathcal{V}, \mathcal{E})$ with the object type mapping $\tau : \mathcal{V} \rightarrow \mathcal{A}$ and the relationship type mapping function $\varphi : \mathcal{E} \rightarrow \mathcal{R}$, which is a directed weighted graph defined over object types $\mathcal{A}$, with edges as relationships from $\mathcal{R}$. ◀

In this work, we interchangeably move from HIN to Knowledge Graph, another similar concept.

4.1.2 Graph-based Inference Algorithms

While investigating all related graph-based inference algorithms in detail is beyond the scope of this paper, it is still essential to recognize and briefly review the most influential related work and their shortcomings, hence leading to the introduction of our MalRank algorithm and how it is designed to fit our requirements the best. We would like to refer the reader to the references provided to learn more about the details of each algorithm.

Belief Propagation

(BP) originally proposed by Judea Pearl [202], also known as sum-product, is one of the most popular and successful applications of label propagation in probabilistic graphical models such as Bayesian Networks and Markov Random Field. BP views inference as marginal probability estimation in graphs. At a high level, BP infers a node's label from some prior knowledge about that node and other neighboring nodes by iteratively passing messages between all pairs of nodes in the graph. The message-passing phase terminates when messages do not change significantly between iterations. In the end, each node will calculate its belief which is an estimated marginal probability [283].

Although BP is the most widely adopted graph-based inference algorithm for threat detection, this algorithm has various limitations. First, BP is designed to work best with probabilistic graphical models that do not generally consider the type of nodes/edges nor directions. Some works have attempted to address those issues, such as zooBP [74], which proposes a closed form with a guaranteed convergence version of BP that is intended to work on heterogeneous graphs with multiple types of nodes and edges. However, there are still limitations that exist within the core concept of BP. More specifically, BP is used to approximate the marginal distribution of a random variable being in a particular state (e.g., a malicious or benign node). Thus, by definition, this probability will be biased towards the majority class (benignness), and with numerical instability of multiplication, maliciousness will end up disappearing from the graph. For example, consider a node having 3 connections to neutral nodes with $P(x_{unknown}) = 0.5$ and 1 connection to a malicious node with $P(x_{mal}) = 1$; running BP until convergence will change the score of the node from originally 0.5 ($P(x_{mal}) = P(x_{ben}) = 0.5$) to $P(x_{mal}) = 0.508$, which is clearly, a low score for such a structure.

Random Walk with Restart (RWR)

RW-based algorithms emulate random walkers taking steps within a graph while having a small probability of teleporting to a random node rather than following an out-edge; hence, Random Walk with Restart. Note that the small probability of teleportation to a random node, also known as the taxation parameter is to avoid spider traps, i.e., the surfer reaching a dead end.

RWR has been successfully utilized in numerous settings, perhaps the most famous algorithm being Google's classic PageRank (PR) developed by Brin and Page initially introduced in [32, 194]. PageRank algorithm uses link information to assign global importance scores to all pages on the web. PR achieves this by simulating the behavior of many random surfers, each starting at a random page and at any step, moving at random to one of the pages to which their current page links. The limiting probability of a surfer being at a given page is the PageRank of that page. The intuition behind PR is that a web page can be considered important if other important pages point to it.

The original RWR-based algorithms, such as PageRank, suffer from various limitations, such as the ability to define different types of nodes and edges or the ability to introduce weights on the edges. In this regard, there have been a number of works tackling those specific issues, e.g., Personalized PageRank [21] and Topic-Sensitive PageRank [100] to incorporate the node's context (types), Biased Random Walks, Weighted PageRank [276] introducing the concept of edge weights. Furthermore, there has been several adoptions, e.g., TrustRank [146] to detect web spams by propagating trust label rather than importance using topic-sensitive PageRank, where the "topic" is a set of pages believed to be trustworthy, similarly Distrust Rank [58] and SybilRank [34].

Similar to BP, RWR-based algorithms do not fit our requirements since they were designed to measure the importance and not beliefs. The problem is that importance is a relative measure which means, in most of the RWR-based algorithms, the values are never created nor destroyed but rather passed from one node to another. These works are great in measuring importance but not maliciousness. Maliciousness needs to be treated like a disease. When someone infects another person with a particular disease, he does not, in turn, get healthy himself, but rather they will be both infected (i.e., maliciousness is created). Lastly, RWR-based algorithms assume a connected graph, whereas our knowledge graph is extremely sparse. Koutra et al. [144], provide a comparison of BP and RWR-based algorithms.

Diffusion in Social Network

Next is the study of *Influence* and *Diffusion* in social networks, e.g., how ideas are spread among peers. Although the definitions might initially seem irrelevant to our problem, one can view the algorithms in this field as an inference or label propagation problem. In this regard, there are various algorithms that model this influence and diffusion, most notably, Linear Threshold (LT) and Independent Cascade (IC) [128]. These models, despite being simple, have the closest intuition to our problem, yet, they require significant adjustments to support our other requirements, e.g., edge weights, influence maximization, etc.

Others

Other notable algorithms include *SimRank* [120], a graph-based structural context similarity measure with the intuition that two objects are similar if they are related to similar things. *Graph-based Semi-Supervised Learning*, also known as label propagation, tackles the problem of unlabeled data with the principle idea that it can be utilized to decide the “metric” between data points and improve models' performance [249]. *Graph Embedding*, which attempts to represent a graph as low dimensional vectors while preserving the graph structures, e.g., DeepWalk [206], Line [258], GraphSAGE [95]. Although these algorithms have proven successful in other domains, we still feel that they are not sufficiently mature for practical usage at applications of our scale when raw data could easily reach petabytes. That being said, we would like to explore some of these algorithms, particularly those based on graph embedding, in our future research.

4.2 CyberHIN: SIEM-based Knowledge Graph

This section presents a detailed description of the proposed heterogeneous information network or knowledge graph (CyberHIN), specifically in two scenarios: one designed based on endpoints (EDR) logs and another from network logs (DNS, Proxy, DHCP).

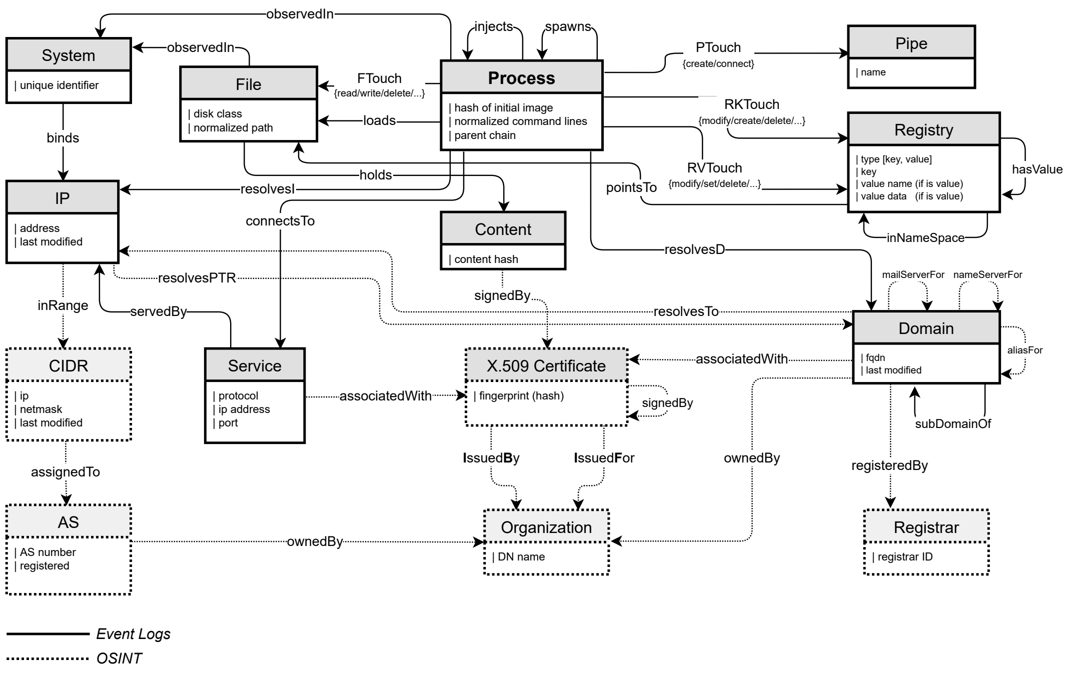


Figure 4.1: HIN network schema based on EDR logs enriched with related OSINT and CTI.

The nodes in the HIN reflect globally unique and identifiable entities that are valid across different entities, e.g., Files, Domains, Autonomous Systems, etc. These entities have mandatory and auxiliary properties. The mandatory properties allow global re-identification, e.g., file SHA256, Autonomous System Number, whereas

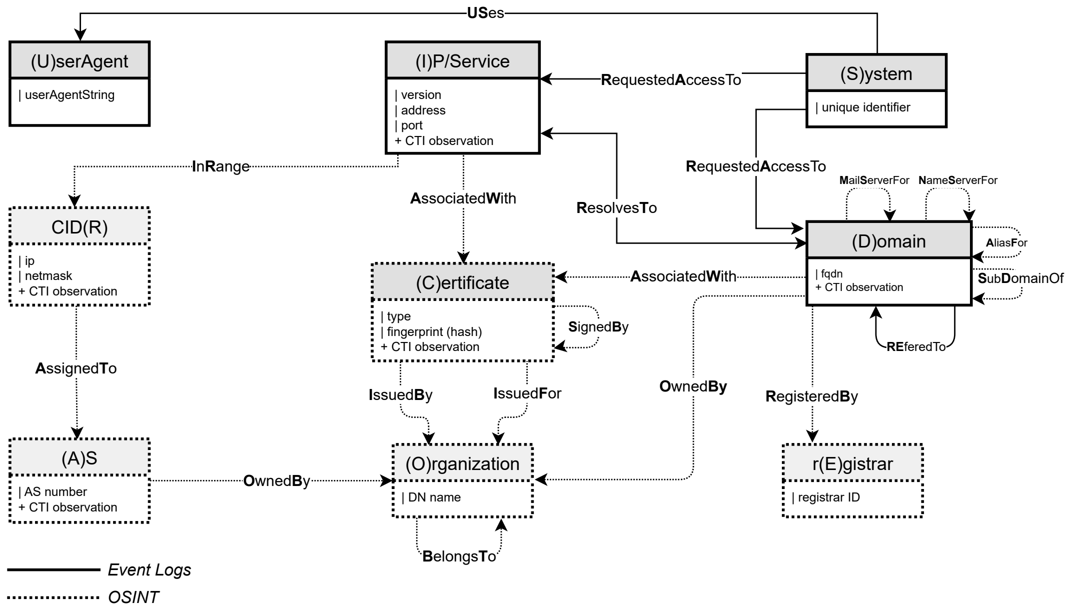


Figure 4.2: HIN network schema based on DNS, Proxy, and DHCP logs enriched with related OSINT and CTI.

the auxiliary properties are informative, e.g., file name. The relationships capture object-related actions and resource relations, including actions such as process creation, loaded libraries, file modification, network connections, and relations such as IP address to CIDR and domain to a registrar. This modeling allows us to re-identify (infer) shared behavior and associations across applications.

Figure 4.1 presents CyberHIN solely based on EDR logs, while Figure 4.2 is based on Proxy, DNS, and DHCP logs,

Below, we expand on the intuition behind the main entities and relationships, their importance, and their role in maliciousness inference. To ensure consistency among the EDR data models, we substituted the key underlying events with those observed in Sysmon logs (See table 4.1). Additionally, we narrowed our focus to the most frequently occurring fields in proxy and DNS logs. We would like to acknowledge that one can expand the entities and relationships extracted given the support from the underlying data source (See the Section 4.9 on future work).

4.2.1 Entities & Relationships Extracted from EDR

As previously mentioned, endpoint events are considered one of the most valuable sources of information, which can be captured by tools like EDR. Several studies have focused on utilizing endpoint and EDR logs for threat detection [12, 134, 192]. In this context, we aim to expand entities and relationships one can capture by analyzing such logs.

Data provenance, originally introduced by King et al. [135, 136], describes system execution and facilitates causal analysis of system activities. Provenance graphs typically encode causal dependence relations between system subjects (e.g., processes) and system objects (e.g., files, network sockets), usually represented as a directed acyclic graph, i.e., provenance graphs.

Provenance graph analysis is a promising approach to investigating cyber attacks. In recent years, there has been significant research in the application of provenance graphs [292], for instance, alert triage [99], threat detection [270] and investigation [23, 97, 107, 173, 174, 203].

While provenance graphs were originally expected to be generated by observing os-level objects and API calls, one can generate such graphs using events generated by EDR or similar tools. Provenance data are usually represented as a directed acyclic graph throughout the literature. However, since we are interested only in the association and not the sequence of events, we represent the underlying data as a heterogeneous information network, hence the provenance HIN.

Table 4.1: Relevant events available from Sysmon

EventID	Description
1	Process created
2	Process changed file creation timestamp
3	Network Connection
5	Process terminated
7	Image loaded
8	Remote thread created
10	Process access
11	File Created
12	Registry object created or deleted
13	Registry value set
14	Registry key or value renamed
15	NTFS file stream created
17	Pipe created
18	Pipe connected
22	DNS request
23	File deleted

Process Execution

A process represents the core unit of execution in today's operating systems, encapsulating and separating all physical and virtual resources bound to a single run of a particular program [286].

Modeling processes in a generalized form that allows unique and global identification can be a challenging task; one has to consider the initial image, open handles, security context, and shared libraries. In this work, we attempt the best effort by identifying a process based on its initial image hash, normalized command line, and parent chain. This approach covers the most relevant inputs given to a process before it can perform actions and the context in which it is executed, modeling the initial state of execution.

Malware is often not standalone but rather accesses a network of related executables from the operating system (like scripts and pre-installed tools) and other sources, e.g., the use of Mimikatz is a prominent characteristic of many APT attacks [112]. Sysmon logs the creation of processes under event ID 1. The event provides information about the newly created process and its parent process, thus the relationship `Process—(spawns)→Process`.

Malware often uses process injection techniques [260] to disguise the origin of action. While there are many different sub-techniques (e.g., Thread Execution Hijacking, Process Hollowing, or Asynchronous Procedure Calls), all have in common that one process can execute under the context of another process. With the `Process—(injects)→Process` relationship, we model this behavior to preserve the associations between the actions of the different processes. One can capture this relationship from Sysmon Event ID 8.

Loaded Executables/Libraries

Relationship `Process—(loads)→File` captures an association between a process and a particular library. A library shared between multiple executables indicates that the executables have access to similar functionality. In Microsoft Windows, system libraries scope the used operating system APIs [11].

Whenever a process loads an additional image (e.g., a shared library) into its process space, Sysmon emits events with the Event ID 7. The loaded image is described by its image path and its corresponding Hash, i.e., `File—(hasContent)→Content`.

File Operations

A file is a kernel object that exposes content under a certain path. Each file object contains a set of relevant properties, such as file name and path, that are used in our modeling. Paths are usually volatile and ambiguous, thus requiring extensive normalization and resolution. A path can be split into a volume and location reference. The volume reference contextualizes the location. One can consider local and remote filesystems, whereby remote disks are identified by their network address. Local filesystems can be either the boot and system disk or an internal or removable data store. The location reference is relative to the volume. For a globally identifiable address, symbolic links and junctions need to be resolved, and system-specific path components like usernames need to be removed. As provided by Sysmon Event 15, NTFS file streams could be considered an extension of the path.

A set of relationships captured by `Process—(FTouch)→File` that emphasizes file accesses/modifications could be used to fingerprint software behavior [274]. In this regard, program file creation, modification, and removal play a special role - Droppers are typically used to install malware executables [149], triggering a file creation event. Viruses work by injecting themselves into executables and libraries, therefore causing modification events. A similar setting applies to self-modifying executables, which is a widespread obfuscation technique [191]. Most malware contains self-removal functionalities that cause delete events [159].

These relationships and entities can be extracted from the Sysmon Event ID 11 (file create), 15 (file create stream), and 23 (file delete), respectively. Note that capturing file operations via Sysmon-like tools is limited to only `TargetFilename` and not content hash.

While in this modeling, we distinguish process from file and content for finer control over the inference, one can combine all these into a single entity represented by its image hash to simplify the modeling.

System Observations

With the intuition, that good applications are typically used by many users, whereas potentially malicious applications tend to only appear on a few computers [40]. One can capture such association by observing a particular file or process within a particular system, thus `File/Process—(observedIn)→System`. A system is a computing element such as a Windows workstation uniquely identified with an identifier such as `MachineGuid`.

Registry Operations

Registry entries are key-value pairs in a system-wide hierarchical database. Keys are hierarchically structured container objects that can hold values and subkeys, and values map a key to a value.

While this database is only available on a few operating systems like Microsoft Windows, it holds various high-relevance entries for malware. Examples include network hooks, autostart entries for systems and applications, user profiles, and virus scanner properties [16]. Research also shows success in malware detection by analyzing registry accesses [255].

`File–(RKTouch/RVTouch)→Registry` relationships attempt to capture such association. Note that to avoid ambiguous hive names (i.e., those referring to the user's profile), one should normalize paths to ensure global identification across different machines (e.g., by normalizing user SID components). Registry events can be extracted from Sysmon event ID 12 (create and delete), 13 (set), and 14 (rename).

Furthermore, as the Windows Registry is structured by component, the hierarchy of a key-value entry carries information about the scope of its entries. One can utilize this property, i.e., `InNameSpace`, to infer risk in the proposed graph structure. One can further parse the registry values attempting to extract relevant association to certain files, e.g., for malware setting auto-start entries for an executable. The relationship `pointsTo` attempts to capture such meta-paths.

Named Pipes

Windows provides a series of kernel objects to synchronize cooperating processes: *Mutexes*, *Semaphores*, *Timers*, *Events*, *Pipes*, and *Mailslots*. Pipes [273] can be used to pass data between two processes. Pipes on Windows are either anonymous or can have a name by which other processes can connect to the pipe. It is possible to use pipes across multiple devices with UNC naming and SMB as an RPC transport. Some malware families are well-known to use named pipes for synchronization and communication, for example, the Cobalt Strike framework [182]. Pipe access can be modeled as `Process–(PTouch)→Pipe` indicating the creation of and connection to a pipe. Sysmon Event 17 indicates the creation of a new pipe, and Event 18 shows the connection of another process to the pipe.

DNS Requests

`File–(resolves)→Domain` is a relationship that models the executables' attempt to make a DNS query for a domain name (DNS record type A, AAAA) to the respective operating system API. This relationship attempts to capture techniques, such as spear phishing scenarios [106] and domain generating algorithms [208], with the intuition that attempting to resolve a malicious domain/IP should not be seen from a benign context.

Note that, if possible (i.e., data source support), one can expand this via other DNS record types such as the `resolvesPTR` relationship modeling PTR records, `MailServerFor` MX records, `NameServerFor` NS records, and `AliasFor` CNAME record.

Another relationship extractable from this event is `subdomainOf`, which attempts to capture the hierarchy of subdomains (e.g., DGA and domain shadowing). For example, in shared or cloud hosting, it is common to provide subdomains to different customers. If such a host predominantly hosts malicious content, one can assume malicious intents. The same principle applies to the upper levels of domain hierarchy [15], as seen in the SpamHaus ranking of the most abused top-level domains[1]. Techniques like domain shadowing require a differentiation between the different domain levels. Sysmon logs all DNS resolution requests through the local system resolver via Event ID 22.

Network Connections

`File—(connectsTo)→Service` attempts to capture TCP- and UDP-based network connections of executables to an *IP address* with a certain *port*. This combination of IP address and ports attempts to uniquely identify a service running on a host and distinguish different activities related to the same IP address.

MITRE ATT&CK[2] defines a set of use cases for which malware might need to open network connections, including lateral movement, command and control, and data exfiltration. These network connections are characteristic of malware and are therefore used for fingerprinting in anti-virus products [274]. As IPv4 addresses are an increasingly rare and expensive resource [64], the probability of an attacker reusing their addresses increases. This is of particular relevance as command and control services need persistence when used without indirection. Only a few hosting organizations don't comply with abuse reports or provide anonymous services [188]. At the same time, IP blackholing increases the pressure for attackers to build reliable communication systems [60]. One can collect network connection events by filtering for the Sysmon Event ID-3.

1 https://www.spamhaus.org/statistics/tlds
2 https://attack.mitre.org/matrices/enterprise

4.2.2 Entities & Relationships Extracted from Proxy & DNS

Due to the fact that web traffic is typically allowed by most of the firewalls, HTTP, HTTPS, and DNS traffic are extensively abused by cybercriminals [47, 178] such as bots communication with command-and-control servers, hence leading to the popularity of proxy and DNS log analysis in the security domain.

Oprea et al. [193] discuss the set of features extractable from proxy logs (e.g., domain connectivity, the referrer string, the user-agent string) that aid in the detection of malicious domains. Ma et al. [162, 163] address the same problem using URL' lexical and host-based features (e.g., number of dots) with the intuition that malicious URLs exhibit certain common distinguishing features. Zhang et al. [289] use term frequency/inverse document frequency (TF-IDF) algorithm to tackle malicious URL detection. Bilge et al. [26] introduce EXPOSURE, a system that employs large-scale passive DNS analysis to detect malicious domains using features such as the number of distinct IP addresses per domain, average TTL, the percentage of numerical characters, etc. Antonakakis et al. [14] propose Notos, a similar system to EXPOSURE, while distinguishing itself by incorporating complementary information such as the registration, DNS zones, BGP prefixes, and AS information. In later research [15] Kopis is introduced, which separates itself from previous work by analyzing the DNS traffic at the upper level of the DNS hierarchy rather than local recursive DNS servers.

These works highlight the value of proxy and DNS log analysis for the purpose of threat detection/hunting. Unlike these efforts, which mostly target local features, we focus on global features extracted from proxy and DNS logs correlated with DHCP. Subsequently, we are going to further expand on each relationship, describing the intuition behind its importance for maliciousness inference.

Network Connection

The relationship *requestedAccessTo* captures an HTTP/HTTPS request or a DNS query from a client/workstation (with a MAC address) to/for a domain/IP. This can be extracted from proxy or DNS logs where a client is requesting a domain or an IP. The intuition here is that if a workstation is known to access a large number of malicious domains/IPs, it is possible that there exists malware on the machine which is trying to reach out, e.g., reaching out to the command and control [167]. It is worth noting that the majority of the DNS or proxy servers do not log the client's MAC address. Instead, they tend to log the client's IP address. That is why it is important to be able to correlate the client IP with the corresponding MAC address according to the DHCP logs to identify the endpoint.

HTTP-based Global Features

The relationship *referedTo* is extracted from the referer field in the HTTP request header logged in proxy logs capturing a relationship between two domain/IP nodes if one has referred to the other. The main intuition here is that the majority of the malware serving networks are composed of a tree-like structure in which the victims are usually redirected through various hops before landing on the main distribution site (e.g., exploit kits, drive-by-download, malvertising) [169]. Although different victims might land on totally different sites, the redirection paths usually overlap. Furthermore, the HTTP referer is also set while a domain is loading its modules from potentially different servers, therefore, indicating association among different domains/IPs. Online advertising networks can introduce a challenge due to their association with many different entities.

The relationship *uses* intends to capture the user agents used by each endpoint with the intuition that even if malware is trying to disguise itself as an innocent application (e.g., a browser) to reach out using HTTP, the user agent string might still differ from the major UA used by the workstation (e.g., different browser version). The user-agent string is extractable from proxy logs (i.e., user agent field).

DNS-based Global Features

Similar to EDR's DNS-related events, enterprise DNS servers have the ability to log all DNS requests and responses; however, they are much more comprehensive and dependable since they are accountable for all resolutions.

In addition to the relationships described in the EDR section (4.2.1), *nameServerFor and mailServerFor*, are the underrated relationships that can be extracted from DNS Resource Record type NS and MX, which indicates the deligations of a domain to a set of name servers (NS) or mail servers (MX), respectively. The intuition here is infrastructure reuse. Similarly, *aliasFor*, a relationship extracted from DNS RRs type CNAME with the intuition that domains connected by canonical name records share intrinsic relation and are likely to be in a homophilic state [205].

4.2.3 OSINT Enrichment

We would like to define Open-Source Intelligence as any information gathered from publicly available sources (i.e., open-source) that provide context to those observed entities extracted from our SIEM-based data.

OSINT can potentially improve the inference and reasoning about the maliciousness of an entity (e.g., IP range or ASN for an IP). Antonakakis et al. [14], Khalil et al. [130], and Mahjoub [165] make use of ASNs, as part of their feature set to detect malicious activities with the intuition that cybercriminals tend to use few hosting services, which allow hiding their identities and don't react to complaints. Mishari et al. [175], Holz et al. [105], and Cooper et al. [46] provide a comprehensive analysis of X.509 certificates and the most important features one can extract for detecting malicious X.509 certificates.

Similar to event logs, OSINT could also pivot endlessly. Therefore, it is important also to define a scope for the related OSINT. OSINT Framework[3] provides a good overview of all available OSINT sources, Enaqx[4] provides a comprehensive collection of OSINT tools. However, due to the fact that our event logs can reach up to 10 terabytes (TB) generated per day, it is also important to select those OSINT which can be collected/crawled at scale. Lastly, we would like to distinguish between passive and active collections. We define active as those that require an active engagement with a server or an API for the collection, e.g., DNS RRs. While passives are those that can be collected in bulk without a per-entry interaction (e.g., ASN).

Thus for the purpose of this research, we limit our OSINT to *IPRanges*, *ASN*, *X.509 certificates, and DNS Resource Records.*

OSINT data does not provide actions but relationships between used resources.

CIDR and ASN

IP ranges and Autonomous Systems (AS) define the ownership structure of an IP address. IANA assigns globally-routable IP addresses to autonomous systems in chunks of different-sized subnets. As all regional Internet registries require a reason for new allocations, IP range assignments are often separated by customers, use cases, or applications. This style of segmentation even applies to internal networks for routing purposes. Consequently, neighboring IPs are typically associated with a shared purpose and AS ownership. This relationship has been exploited in IP neighborhood clustering [115, 189]. As malicious hosting companies have been observed in the wild, autonomous system numbers can carry a reputation similar to IP addresses [14, 130, 142, 165][5].

It is possible to collect IP ranges and their relationship to autonomous systems as a passive BGP listener or using RIRs measurements such as those published by APNIC (https://thyme.apnic.net). IP ranges are uniquely identified by an IP address, netmask, and ASs by the RIR-assigned autonomous system number (ASN).

3 http://osintframework.com/
4 https://github.com/enaqx/awesome-pentest#osint-tools
5 https://www.spamhaus.org/statistics/botnet-asn

Certificate

X.509 is a standard for Public Key Infrastructure (PKI) and has been adopted by the Internet Engineering Task Force (IETF) as the PKI for several IETF protocols such as HTTPS, IMAPS, SMTPS, and POP3S [46]. X.509 Certificates can build a hierarchy of trust through cryptographic signatures. To use an X.509 certificate publicly, it needs to be signed by a Certification Authority (CA) that is trusted by the operating system or software which is used to connect to a service. PKIX standardizes various procedures for CA operations, which ensure that public keys are bound to reference entities like an X.509 Distinguished Name (DN) or Alternative Name (AN), e.g., an e-mail address or a DNS entry. These certificates are integral parts of today's internet protocols' security, providing the mean for trust between two parties. X.509 certificates contain valuable information, such as details related to the issuer, subject, creation date, and associated domains/IPs. In this regard, Mishari et al. [175] discuss how features from SSL certificates can be used to detect fraudster domains, Holz et al. [105], and Cooper et al. [46] provide a comprehensive analysis of X.509 certificates and the most important features one can extract for detecting malicious X.509 certificates.

This was particularly relevant before the introduction of free-of-cost TLS certificates with StartSSL and Let's Encrypt; TLS certificates were expensive and, therefore, a rare resource, making them often subject to reuse. X.509 certificate fingerprint is a threat intelligence type that allows fingerprinting several malware families' C&C servers [13, 46][6].

Another use of X.509 certificates is code signing, which is required by several operating systems like macOS and Windows for regular software and kernel drivers. Measurements by Kim et al. showed that many Windows malware samples have a valid signature [133]. As these signatures are expensive and hard to obtain, one can expect to reuse (compromised) developer certificates. A certificate is uniquely identified by its hash (within the means of collision resistance). One can collect certificates from certificate transparency logs, executables, TLS-logging web proxies/DPI instances, and large-scale web scanning of public-facing services.

Although numerous features can be utilized for examining X.509 certificates in the context of threat detection, our focus is solely on global characteristics that might suggest a malicious link. These global features comprise the certificate's connection to other entities such as domains, IPs, and files, as well as the signing hierarchy [13, 46, 210].

6 https://sslbl.abuse.ch/statistics

Registrar and Organization

Registrars allow the registration of publicly reachable domains. While typical TLD policies require personal identifications, some registrars allow bypassing these measures, thus attracting more criminal activities. In addition, registrars play a unique role in detecting fast-flux scenarios, where IP addresses are changed at a very high pace [66].

A domain registrar can be requested using the WHOIS [53] or Registration Data Access Protocol[7] (RDAP) for most public TLDs. This allows querying authoritative metadata on domains and IP addresses. Registrars are uniquely identified by their IANA-issued registrar ID.

Organizations are legal entities that can hold virtual resources. References to legal entities are provided in X.509 certificates and RDAP/WHOIS responses to requests on domain names and IP addresses. For many legitimate businesses, these references correlate across their activities, such that benignness can be inferred between legitimate resources. However, malicious actors don't have any incentive to provide this form of correlation between their activities but sometimes reuse identifiers.

DNS Records

DNS data is among those valuable information sources that have a lot of potential for threat detection. Significant research exists exploring threat detection via DNS log analysis [15, 26], and many others that expand on the relationships extracted from DNS data logs [110, 130, 186, 187, 293].

Even though the process endpoint of proxy logs may only capture an effort by a process to resolve a domain name or IP address, such resolution might not always be available. Therefore is possible to enrich all observed domains and IPs by obtaining their corresponding DNS records (for instance, through the use of the Gieben DNS library [8]). Nevertheless, it is essential to remember that the results may not always be accurate as responses may vary during the query due to DNS-level load balancing, CDNs, traffic manipulation, fast-flux networks, and other factors.

4.2.4 CTI Enrichment

As mentioned, our approach exercises guilt-by-association. In this regard, it is essential to introduce seed maliciousness for propagation and inference. This

7 https://www.icann.org/rdap
8 https://github.com/miekg/dns

can be achieved by matching the HIN's entities against Cyber Threat Intelligence (CTI). The term cyber threat intelligence or threat intelligence is defined and used differently throughout the literature. However, we would like to use Chismon's definition [42]. Threat intelligence is information that can aid with threat detection tasks. Similarly to OSINT, CTI could also be endless (e.g., indicators, TTPs). For the purpose of this research, we focus only on Indicators of Compromise (IOCs) related to our observed entities, particularly *file hashes*, *domain names*, and *IP addresses*. Furthermore, we limit our sources to those that can be used for passive collection at scale with no API limitation. Slatman provides a curated list of threat intelligence [241] resources.

4.3 MalRank: Graph-based Inference Algorithm

This section begins by defining the problem and requirements that led to the development of our proposed graph inference algorithm, MalRank.

4.3.1 Problem Definition

At a high level, we would like to reason about an entity based on its association with other entities, with the intuition that malicious entities tend to share some global properties. In this regard, graphs or heterogeneous information networks are ideal for this task due to their capability to preserve the correlation and association among different entities. That is why we formulate our problem as a graph-based inference problem. More specifically,

Given:

- A directed weighted graph or a heterogeneous information network $G = (\mathcal{V}, \mathcal{E})$, with a particular network schema $T_G = (\mathcal{A}, \mathcal{R})$.
- A Prior score s^o and prior confidence c defined over all $x \in \mathcal{V}$, where $s^o(x) \in \{0, 1\}$ and $c_{s^o(x)} \in [0, 1]$. Where $s^o(x) = 0$ denotes x as a neutral node and $s^o(x) = 1$ denotes x as a known malicious node. c represents the confidence in the label, 0 being no confidence, and 1 absolute confidence in the trustworthiness of that label (expected to be set according to the TI source).
- An edge weight ω defined over each edge type $r \in \mathcal{R}$ and its inverse r^{-1}, denoting the influence of relationship type in each direction. $\omega_r, \omega_{r^{-1}} \in [0, 1]$. The higher the edge weight, the higher the influence.

Find:

- Maliciousness score $\mathcal{S}(x)$ of a node x, i.e., $\mathcal{S}(x) \in [0, 1]$. A higher score indicates a higher risk.

4.3.2 Graph Inference Requirements

The graph-based inference has been studied widely in a variety of domains. Although it is referred to differently depending on the domain (e.g., influence, diffusion, propagation, transductive classification, meta-path-based similarity), at its core, the problem can be simplified to the inference of nodes' properties based on their neighbors. In our case, inferring the maliciousness of a node based on the maliciousness of its neighbors. This is also known as guilt-by-association throughout the literature [257].

Before presenting our graph-based inference algorithm, it is essential to establish our primary requirements for the use case. By doing so, we can assess the constraints of existing algorithms and recognize the necessity for a new graph inference algorithm, namely MalRank.

Single Diffused Label

Since CyberHIN is constructed from entities and relationships observed in an enterprise's SIEM, it is quite unlikely that the number of benign and malicious entities is proportional, i.e., the majority of the entities are expected to be benign. This is due to the fact that the most traffic within an organization is expected to be benign. That is why it is important for us to consider only one label (maliciousness). Therefore, the algorithm should be able to infer a maliciousness score for any given node based on its neighbors' maliciousness score while considering the number of neutral neighbors to reduce the maliciousness. In other words, if a node has a high degree with a large number of neutral neighbors, it is less likely to be malicious (Figure 4.3). For example, a malicious domain is more likely to be accessed by a few of enterprise workstations rather than the majority of them [167]. This requirement also allows us to eliminate supernodes (nodes with a high degree, e.g., content delivery networks, web hosting services, or advertising networks). In this regard, while the majority of the previous efforts take supernodes as a challenge, thus eliminating them from the final graph, with this requirement, we can ensure that supernodes won't affect the algorithm due to a high number of neutral neighbors.

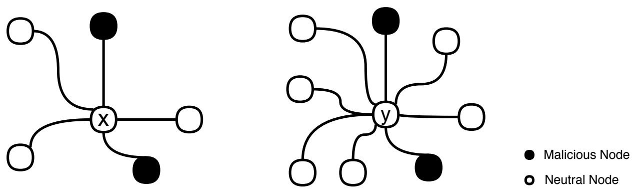


Figure 4.3: The effect of the neutral neighbors on a node's MalRank score.

Directed Weighted Propagation

The next important requirement is the ability to define edge weights. Since the knowledge graph is expected to consist of various types of nodes and edges, the algorithm must be capable of considering how maliciousness should be propagated through a particular association. For instance, a *resolvesTo* edge should have a much greater influence than *requestedAccessTo*. Furthermore, although the majority of the relationships described in the previous section can be treated as bidirectional edges, the algorithm should be able to incorporate not only edge directions but also different edge weights in different directions. This would allow one to have much more control over the influence weights and their directions. This is important as it can stop an adversary from rendering the algorithm ineffective by connecting to a large number of neutral nodes (e.g., referring to a large number of benign domains or adding CNAME records pointing to other legitimate domains). Although it's quite unlikely that this is happening at the moment, one must also consider this as part of threat modeling.

Maliciousness Influence Maximization

Maliciousness should be treated like a disease, i.e., the more malicious a node is, the greater its influence. This ensures that the maliciousness does not fade within a graph of extreme bias towards benignness. Thus, the algorithm should have a mechanism to adjust the edge weights depending on the source's maliciousness score, i.e., if a node gets more malicious, the edge weights on the edges connecting that node to others should be increased accordingly, thus allowing maliciousness to be propagated more effectively.

Scalable and Iterative

Lastly, due to the amount of our data, which can easily reach 10 TB per day, the algorithm should be able to scale both vertically and horizontally to allow parallelization across multiple machines. Furthermore, having the possibility to have an iterative implementation that is better supported by the current state of the art for large-scale graph analytics.

4.3.3 MalRank Formulation

Let us denote the maliciousness score (MalRank score) of a node $x \in \mathcal{V}$ as $\mathcal{S}(x)$; following our earlier intuition and definition, the simplified $\mathcal{S}(x)$ can be calculated with:

$$\mathcal{S}(x) = c_{s^o(x)} s^o(x) + (1 - c_{s^o(x)}) \frac{\sum\limits_{y \in N(x)} \sum\limits_{r \in \mathcal{T}_{xy}} \mathcal{S}(y).\hat{\omega}_{xy^{r^{-1}}}}{\sum\limits_{y \in N(x)} \sum\limits_{r \in \mathcal{T}_{xy}} \hat{\omega}_{xy^{r^{-1}}}} \tag{4.1}$$

Where $s^o(x) \in 0, 1$ refers to the prior of node x. If x is a known malicious node $s^o(x) = 1$, and 0 otherwise (usually set if x is observed in a TI source). $c_{s^o(x)} \in [0, 1]$ is the prior confidence of $s^o(x)$. This indicates the trust level of the prior. The value is decided according to the trust level for the corresponding TI source. This is introduced to control low-quality threat intelligence; we shall discuss this later. $N(x)$ is the set of nodes neighboring node x, $\mathcal{T}_{xy} \subset \mathcal{R}$ is the set of relationship types between x and y. $\hat{\omega}^{i}_{xy^{r^{-1}}}$ is the maximized/minimized edge weight on the relationship type r directed from y to x.

▶ **Definition 4.3.** *Maximized/Minimized Edge Weight ($\hat{\omega}$)*
As discussed previously, there are three main requirements to control the propagation and influence: first, the ability to decay the influence differently on different edge types. This is achieved by introducing edge weights, ω_r denoting the weight on the edge of type r. Second is the ability to have different weights in different edge directions. This is achieved by distinguishing the direction of the influence; hence $\omega_{r^{-1}}$ is the inverse direction of ω_r. For instance, if r is a relationship *spawns* between node x and y (x—*spawns*—y), r^{-1} is the inverse relationship (y—*isSpawnedBy*—x) which can have a different weight. Thus, ω_{xy^r} does not need to be the same as $\omega_{xy^{r^{-1}}}$. This allows us to differentiate the impact of influence via a relationship in each direction. It is also worth mentioning that this is how the algorithm sees the directions. In this regard, although our knowledge graph is a directed graph, from the algorithm perspective, all edges are bidirectional, but the influence can be different in each direction. This way, one could define the $\omega_{xy^r} = \omega_r = d$ and $\omega_{xy^{r^{-1}}} = \omega_{r^{-1}} = 0$ if the edge type r has only one direction, i.e., from source vertex x to destination vertex y. Lastly, the ability to adjust this decay based on the score

of the influencer. Thus, introducing the maximized/minimized edge weight ($\hat{\omega}_{xy^r}$). This value is calculated by taking the weighted average of the original edge weight and the source maliciousness score:

$$\hat{\omega}_{xy^r} = \begin{cases} 0, & \text{if } \omega^o_r = 0 \\ k\mathcal{S}(x)^+(1-k).\omega^o_r, & \textit{otherwise} \end{cases} \tag{4.2}$$

where ω^o_r is the original edge weight for relationship type r. Note that initial edge weights are defined over relationship types and not on each edge, whereas the $\hat{\omega}_{xy^r}$ are defined over each edge. k is the maximizer factor which is expected to take a value between 0.5 and 0.8. The higher k values enforce a higher maximization/minimization for the new weight ($\hat{\omega}$) according to the influencer's score. ◀

While the equation 4.1 is exact for *singly-connected* (acyclic) and directed graph, i.e., there are no loops or bidirectional edges in the graph, this is not the case in a directed graph in which there could be loops or connected components. In particular, in the case of *singly-connected* networks, the computation starts from the nodes at the root of the graph and only computes scores when the parent scores are calculated. In that case, it is easy to recognize that each score needs to only be computed once. In other words, the whole computation takes time proportional to the number of links in the graph.

However, when loops are present, the algorithm eventually runs into trouble, but if we ignore the existence of loops and continue with the calculation with no guarantee for convergence, the scores will mostly converge to a stable equilibrium as time goes on. This phenomenon is very similar to what we observed in loopy belief propagation [184]. For this purpose MalRank can be calculated iteratively via message passing as follows:

$$\mathcal{S}(x)^{i+1} = c_{s^o(x)} s^o(x) + (1 - c_{s^o(x)}) \frac{\sum_{y \in N(x)} \sum_{r \in \mathcal{T}_{xy}} m^i_{xy^{r-1}}}{\sum_{y \in N(x)} \sum_{r \in \mathcal{T}_{xy}} \hat{\omega}^i_{xy^{r-1}}} \tag{4.3}$$

$$m^{i+1}_{xy^r} = [\mathcal{S}(x)^i - \underbrace{(1 - c_{s^o(x)}) \frac{\sum_{r \in \mathcal{T}_{xy}} m^i_{xy^{r-1}}}{\sum_{y \in N(x)} \sum_{r \in \mathcal{T}_{xy}} \hat{\omega}^i_{xy^{r-1}}}}_{\text{echo cancellation}}].\hat{\omega}^{i+1}_{xy^r} \tag{4.4}$$

$$\hat{\omega}^{i+1}_{xy^r} = \begin{cases} 0, & \text{if } \omega^o_r = 0 \\ k\mathcal{S}(x)^i + (1-k).\omega^o_r, & otherwise \end{cases} \tag{4.5}$$

where $\mathcal{S}(x)^i$ is the maliciousness score of a node $x \in \mathcal{V}$ in iteration i.

$m^i_{xy^r}$ is the message (influence) sent from node x to node y in iteration $i+1$ over edge type r, and $m^{i+1}_{xy^{r-1}}$, is the message sent from node y to node x in iteration i. Note the echo cancellation term designed to eliminate the continuous increase of maliciousness among two nodes, i.e., node x sends a high maliciousness influence to itself via a neighbor y ($x \longrightarrow y \longrightarrow x$). This is achieved by removing the influence of a node from the previous iteration. In other words, the message node x sends to node y in iteration $i+1$ should eliminate the influences of node y on x in the previous iteration (via any link between these two nodes).

$\hat{\omega}^{i+1}_{xy^r}$ is an instance of maximized/minimized edge weight on the relationship type r between node x and y in iteration $i+1$. ω^o_r is the original edge weight for relationship type r.

Figure 4.4 shows the effect of running 20 iterations of MalRank on a syntactically created HIN. Figure 4.5 shows how the MalRank scores and the maximized/minimized edge weights evolve over time and eventually converge after several iterations.

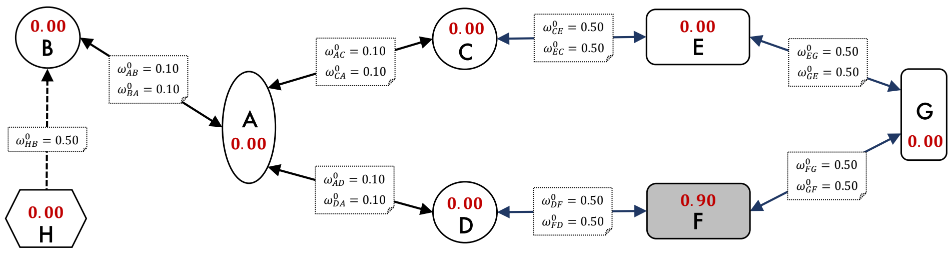
4.3.4 MalRank vs. Belief Propagation

The majority of the literature in this domain has adopted belief propagation as the inference algorithm. During the previous section, we briefly described why there is a core fundamental limitation with belief propagation that prevents it from being effectively used as a malicious inference algorithm based on our requirements. Here we are going to expand this more by providing a simple example showcasing how MalRank scores are compared with BP under a very similar setting.

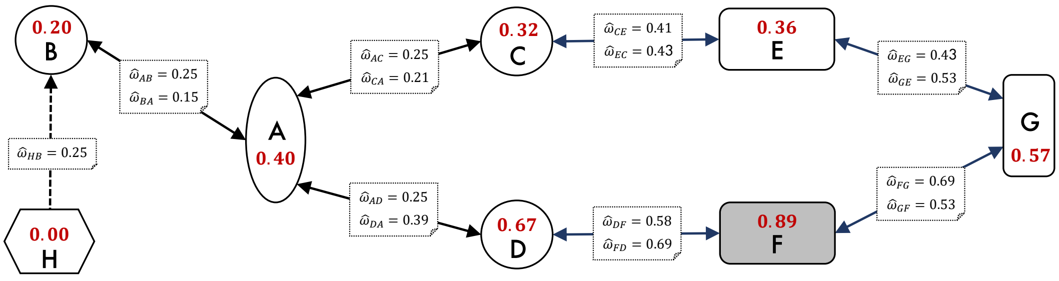
Table 4.2: MalRank and Belief Propagation algorithms' parameters for the Zachary experiment.

MR Parameter
$\forall r \in \mathcal{R} : \omega^o_r = \omega^o_{r^{-1}} = 0.5$
$k = 0.7$
$s^o(x) = \begin{cases} 0.9, & \text{if } x \in \{6, 24\} \\ 0, & \text{otherwise} \end{cases}$ $c_{s^o(x)} = \begin{cases} 0.99, & \text{if } x \in \{6, 24\} \\ 0, & \text{otherwise} \end{cases}$

BP Node Potential			BP Edge Potential		
Node	$P(mal)$	$P(ben)$	$\psi_{ij}(x_i, x_j)$	$x_j = ben$	$x_j = mal$
Malicious (6, 24)	0.99	0.01	$x_i = ben$	$0.5 + \epsilon$	$0.5 - \epsilon$
Unknown (rest)	0.5	0.5	$x_i = mal$	$0.5 - \epsilon$	$0.5 + \epsilon$



(a) Original Graph


0.20 B
$\hat{\omega}_{AB} = 0.25$
$\hat{\omega}_{BA} = 0.15$
$\hat{\omega}_{HB} = 0.25$
0.00 H
A 0.40
$\hat{\omega}_{AC} = 0.25$
$\hat{\omega}_{CA} = 0.21$
$\hat{\omega}_{AD} = 0.25$
$\hat{\omega}_{DA} = 0.39$
0.32 C
0.67 D
$\hat{\omega}_{CE} = 0.41$
$\hat{\omega}_{EC} = 0.43$
$\hat{\omega}_{DF} = 0.58$
$\hat{\omega}_{FD} = 0.69$
0.36 E
0.89 F
$\hat{\omega}_{EG} = 0.43$
$\hat{\omega}_{GE} = 0.53$
$\hat{\omega}_{FG} = 0.69$
$\hat{\omega}_{GF} = 0.53$
G 0.57

(b) After 20 Iterations

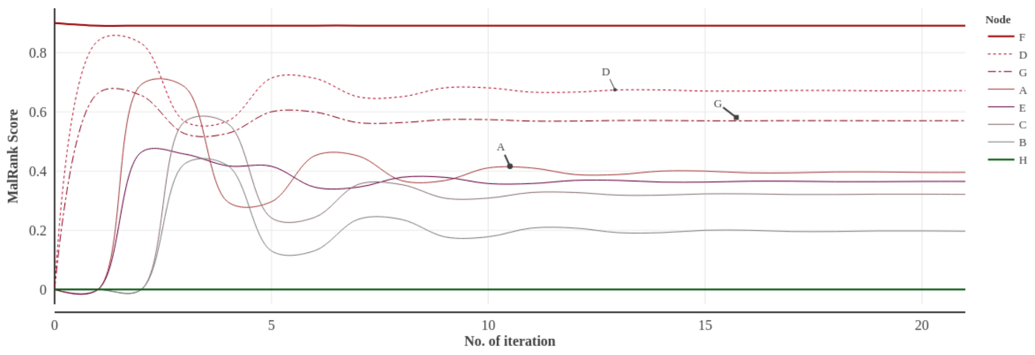
Figure 4.4: How MalRank scores and edge weights are evolved in a synthetically created graph after 20 iterations.

In this regard, table 4.6 (b) shows the effect of running 10 iterations of MalRank and 10 iterations BP on the graph presented in figure 4.6 (a) under a similar setting described in table 4.2 and table 4.2.

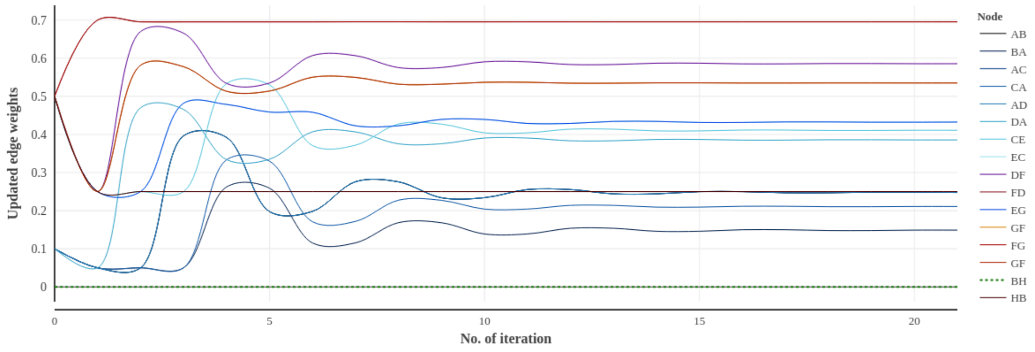
While BP can provide meaningful results under a balanced setting in which there is symmetry among classes (e.g., malicious and benignness/unknown), it fails when there is a huge bias towards a particular class, in our case, benignness. This is also shown in the results. For instance, 10 iterations of MalRank change the MalRank score of node-17 from 0 to 0.6, whereas 10 iterations of BP change its BP score from 0.5 to 0.508 (which would result in a miss-classification).

4.3.5 Naive Incremental Calculation

In a real deployment, the HIN is expected to be extremely dynamic in which new entities (vertices) or links (edges) are added or priors (maliciousness indicators) modified over time. In such a setting, it is not infeasible to expect to run MalRank



(a) MalRank Score, $\mathcal{S}(x)$.



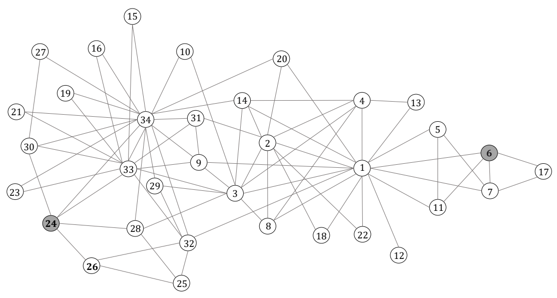
(b) Adjusted edge weights, $\hat{\omega}^{i}_{xyr}$.

Figure 4.5: MalRank scores and edge weights evolve and converge over iterations.

inference on the entire graph every time such updates occur. Moreover, these updates should not significantly affect the entire graph. In this regard, to avoid the re-computation of MalRank scores from scratch, we outline a variety of MalRank which allows us to perform incremental batch updates for the areas of the graph that is most affected by the addition of the new vertices, edges, and priors.

The intuition behind *Naive Incremental MalRank* is that these updates should result in minor changes in the immediate neighbors of those inserted/updated nodes and edges. At the same time, the effect will not be strong enough to propagate to the rest of the graph. To achieve this, we adopt the underlying idea behind Single-Pass Belief Propagation [88] and NetProbe [195].

An update includes either the addition of new nodes with particular links to each other or existing nodes; note that each new node is expected to have its own prior (0 if unknown, 1 if known malicious). It could also be the addition of new links



(a) Zachary karate club graph.

	MalRank		Belief Propagation	
Node	$s^0(x)$	$s^{10}(x)$	$P(mal)^0$	$P(mal)^{10}$
6	0.9	0.826	0.9	0.900
24	0.9	0.823	0.9	0.900
17	0	0.622	0.5	0.508
11	0	0.491	0.5	0.508
7	0	0.484	0.5	0.508
26	0	0.473	0.5	0.508
28	0	0.425	0.5	0.508
30	0	0.414	0.5	0.508
5	0	0.375	0.5	0.500
25	0	0.345	0.5	0.500
27	0	0.321	0.5	0.500
32	0	0.270	0.5	0.500
33	0	0.266	0.5	0.508
1	0	0.256	0.5	0.508
12	0	0.255	0.5	0.500

(b) MalRank and BP scores.

Figure 4.6: Changes in MalRank scores vs. BP scores after 10 iterations on Zachary karate club graph with two nodes marked as malicious.

between already existing nodes. Lastly, changing the priors of existing nodes, e.g., marking an existing node as malicious.

At a high level, whenever an update occurs, particularly with a set of newly added/updated vertices $\mathcal{V}'$ each with their own prior, and set of edges $\mathcal{E}'$, we shall re-calculate the MalRank score of all nodes which are at most h hops away from any node in set $\mathcal{V}^*$, or $h-1$ away from any source or destination vertex of an edge in set $\mathcal{E}^*$, where h is configurable noting the trade-off between computational cost and influencing the larger portion of the graph.

```
Algorithm 1: Naive Incremental MalRank Pseudocode
input : MalRank Graph with Vertices and Edges   $G(V, E)$
        New Vertices   $V'$
        New Edges   $E'$
        Vicinity   $h$
Output: Updated MalRank Graph $G'$
// upsert new nodes/links to the MalRank scored graph G
$G' \leftarrow upsert(G, V', E')$;
$H = \{\}$;
for $v$ in $V'$ do
    // find all nodes that are at most $h$ hops away from
       vertices in $V'$
    $H.insert(BFS(G', h, v))$ ;
end
for $(s_v, d_v)$ in $E'$ do
    // find all nodes that are at most $h-1$ hops away from
       the src or dest vertex of the edges in $E'$
    $H.insert(BFS(G', h-1, s_v))$ ;
    $H.insert(BFS(G', h-1, d_v))$ ;
end
while $i < h+1$ or scores are not converged do
    for $(s_v, d_v)$ in $G'$ do
        // upgrade messages and weights on all edges that
           their destination vertex is in $H$
        if $d_v \in H$ then
            $updateEdgeWeights(d_v, s_v)$ ;
            $updatedMsg(d_v, s_v)$
        end
    end
    // upgrade MalRank score of all nodes in $H$, by
       aggregating the newly sent msgs as well as
       previous msgs
    for $v$ in $G'$ do
        if $v \in H$ then
            $updateMalRankScore(v)$ ;
        end
    end
end
```

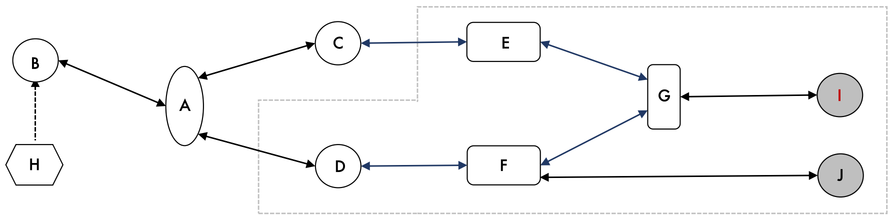


Figure 4.7: Showing the $2-vicinity$ of two newly added nodes I and J for incremental MalRank.

It is important to pay particular attention to the boundary nodes, i.e., those nodes that are exactly h hops away. In this regard, it is important not to eliminate the rest of the graph when re-calculating the scores or messages surrounding these boundary nodes. Eliminating those untouched areas completely will disrupt the global equilibrium. Furthermore, note that MalRank needs to be calculated at least $h + 1$ iterations allowing the desired inference/influence.

Figure 4.7 shows $h-$vicinity of two newly added nodes: $\mathcal{V}^{'} = \{I, J\}$ and and two newly added bi-directional edges: $\mathcal{E}^{'} = \{JF, FJ, IG, GI\}$, where $h = 2$. In this regard, as shown in the graph, finding the $h = 2-$vicinity of $G' \mid \mathcal{V}^{'}, \mathcal{E}^{'}$ will give us $\{I, J, G, F, E, D\}$. Next, we will update all messages (using equation 4.4 and 4.5) on all edges whose destination vertex is in 2−vicinity, i.e., *IG, GI, JF, FJ, GE, EG, GF, FG, FD, DF, AD, CE*. Having the updates messages, for each node in 2−vicinity, we will update the MalRank scores using equation 4.3. This process shall be repeated at least 3 times.

Lastly, it is worth mentioning that by using this naive approach, the re-calculation of the MalRank scores of the whole graph should be done every now and then. Furthermore, note that we have not discussed the removal of vertices or edges in this naive approach.

4.4 MalLink: a Framework for CyberHIN & MalRank

In this section, we delve into the design and implementation of MalLink, a system/framework created to run on SIEMA, interface with the underlying storage, load and parse the event logs stored within, generate the CyberHIN described in Section 4.2, perform OSINT enrichment and CTI marking, and ultimately execute MalRank to identify potentially unknown malicious entities.

4.4.1 MalLink Architecture

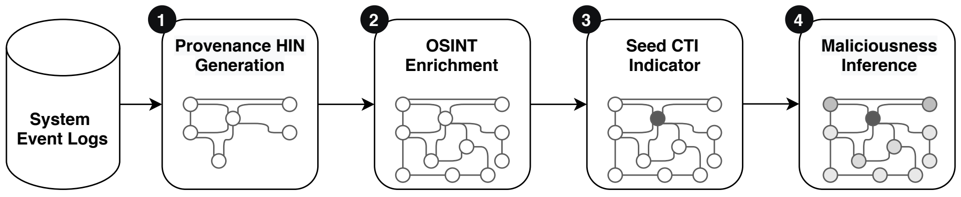


Figure 4.8: MalLink's high-level approach to threat detection.

Figure 4.8 shows an overview of the proposed approach, and Figure 4.9 shows the architectural design of the MalLink system. Due to scalability requirements, we decided to build MalLink using Apache Spark, which would allow us to scale

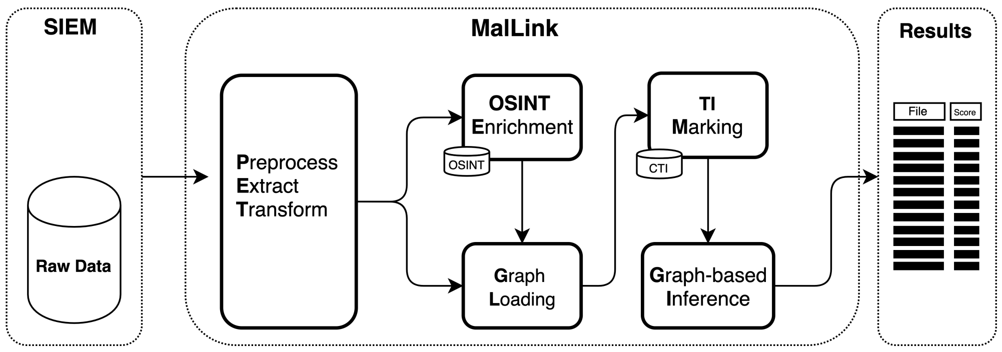


Figure 4.9: Overview of MalLink architecture.

our data processing pipeline as well as the inference algorithm (using GraphX). MalLink is expected to run as a plugin on top of the proposed SIEMA in Chapter 3.

In the following section, we elaborate on the function of each module and provide an overview of the design choices made for their incorporation within the architecture of MalLink.

Log Collection and Processing

MalLink is designed to attach to a data lake (e.g., Amazon S3, Azure Data Lake Storage) to ingest the raw, semi-structured event logs of interest. These logs are then preprocessed (e.g., cleaned, deduplicated, parsed, and validated). Lastly, entities and relationships of interest (according to desired network schema) are extracted and transformed into independent vertex and edge objects which are then passed to the HIN construction and the OSINT enrichment modules.

Each vertex object has a *vid* (vertex identifier), *name*, *type*, *tiObservation*, and *mrScore*. Each edge has a *srcId*, *dstId*, *srcV* (the whole source vertex object), *dstV*, and *eType* (edge type). The intuition behind this specific design is to embrace micro-service, stateless, and distributed design patterns. In this regard, despite duplicating each vertex within each edge object, the system can scale out more efficiently. This is because the loading module can independently process the received vertices and edges, regardless of order or distribution.

OSINT Enrichment

As mentioned, we are interested in further enriching certain entities of interest with related open-source intelligence. In this regard, MalLink consists of various enricher modules awaiting particular entity types to enrich accordingly, e.g., CIDR/ASN enricher module configured to enrich all IP entities with their corresponding CIDR range and ASN.

HIN Construction and CTI Marking

Given a set of preprocessed vertex and edge objects, MalLink builds a HIN. To provide the maliciousness seed for later inference, the *TI Marking* module marks those vertices observed within previously collected CTI.

All extracted and processed entities and relationships arrive independently at the loading module. This module is responsible for de-duplicating, indexing, cleaning, and combining all the vertices and edges. It is also responsible for labeling all vertices according to the TI collected previously while marking some for evaluation. The output of this layer is the final labeled and processed distributed graph.

Graph Inference

After generating the HIN marked with the seed CTI, next, MalLink runs a distributed and iterative implementation of the MalRank algorithm described in Section 4.3 on the generated graph.

MalRank algorithm is implemented using Pregel's computational model using Apache Spark GraphX. Pregel, initially introduced by Malewicz et al. [166], brings graph algorithms into the map-reduce world by expressing graph algorithms as a sequence of iterations, in each of which a vertex can receive messages sent in the previous iteration, send messages to other vertices, and modify its own state and that of its outgoing edges or mutate graph topology. Using this vertex-centric intuition ("think like a vertex"), one can express a broad set of algorithms while parallelizing its computation across any number of nodes. GraphX is Apache Spark's API for parallel and fault-tolerant graph computation at scale.

Algorithm 2 presents MalLink inference pseudocode. In each iteration for every edge in the graph, a map function updates the edge weights and calculates the message to be sent to the destination vertex (according to Eq. (4.4), and (4.5)). Intuitively, by the end of this mapper round, each vertex receives a message for every incoming edge (from other vertices). Then the reduce function combines all messages at each vertex, Eq. (4.3). The reduce function is written to handle only two messages at a time but will be repeated until all messages have collapsed into a single message.

The described system is designed to work both in streaming and batch mode. However, in this research, we only utilized its batch mode. In other words, one must re-run the MalRank algorithm to score newly added vertices. We would like

Algorithm 2: MalLink Graph-based Inference Algorithm Pseudocode

```
input:
        Vertices $V$ ($\mathcal{S}, s^o, c_{s^o}$, InDgrWeightSum )
        Edges $E$ ($x, y, m_{xy}, m_{xy^{-1}}, r, \hat{\omega}_r, \hat{\omega}_{r^{-1}}, \omega^o_r, \omega^o_{r^{-1}}$ )
        k
Output: Graph $G$
1  begin
2      $cleanAndInitialize(V, E)$ ;
3      while $i$ < max iteration or "scores are not converged" do
           // mapper:  upgrade weights and messages on all edges
4          for $e$ in $E$ do
               // using equation 4.5
5              $\hat{\omega}_r \leftarrow$ updateEdgeWeight ($\mathcal{S}(x), k, \omega^o_r$)
6              $\hat{\omega}_{r^{-1}} \leftarrow$ updateEdgeWeight ($\mathcal{S}(y), k, \omega^o_{r^{-1}}$) ;
               // using equation 4.4, simultaneously update
7              $m_{xy} \leftarrow$ updateEdgeMsg ($\mathcal{S}(x), c_{s^o(x)}, \hat{\omega}_r, m_{xy^{-1}}$, InDgrWeightSum) ;
8              $m_{xy^{-1}} \leftarrow$ updateEdgeMsg ($\mathcal{S}(y), c_{s^o(y)}, \hat{\omega}_r, m_{xy}$, InDgrWeightSum) ;
           // reducer:  aggregate messages and weights to update scores
9          for $x$ in $V$ do
10             sumOfInAggregatedMsgs $\leftarrow 0$ ;
11             InDgrWeightSum $\leftarrow 0$ ;
12             for $e$ in $-$ degree of $x$ do
13                 sumOfInAggregatedMsgs $+= m_{xy^{-1}}$ ;
14                 InDgrWeightSum $+= \hat{\omega}_{r^{-1}}$ ;
               // using equation 4.3
15             last_$\mathcal{S}(x) \leftarrow \mathcal{S}(x)$ ;
16             $\mathcal{S}(x) \leftarrow$ updateNodeScore(last_$\mathcal{S}(x), s^o(x), c_{s^o(x)}$,
17             sumOfInAggregatedMsgs, InDgrWeightSum ) ;
```

to leave this to our future work, expanding the implementation to support the temporal incremental mode of MalRank, which not only operates on streams but also takes time (first-seen and last-seen) into consideration.

4.5 Experiment Setup

Infrastructure

For this research, we utilized two variants of the proposed SIEMA (a big data platform that runs on top of a legacy SIEM system).

More specifically, a research workbench used for case study 4.7 consisting of one Dell PowerEdge (R730, R820) and five Fujitsu Primergy RX600 with a total of 1,864 GB RAM, 24 CPUs (200 total cores), and 4 TB storage interconnected via 10 Gb optical fiber. In addition, the data was initially stored on an external Network

Attached Storage (NAS) connected to the cluster via 3x 10Gb optical fiber. A cluster backed by Kubernetes[9] for orchestration, Apache Spark[10] for distributed processing, and Apache Kafka[11] for distributed queueing.

As well as commercial alternatives for case study 4.6 consisting of Azure Databricks (configured with 8 "Standard_D32s_v4" workers). Thus, having a big data platform backed up by Apache Spark with a total of; 1024-GB Memory, 256 vCPU Core, having access to Azure Data Lake Storage, where the company's EDR logs were residing.

MalLink Configuration

When configuring the MalRank algorithm throughout our experiments and case studies, we decided to rely on expert knowledge to initialize the parameters of each vertex (entity) and edge (relationship) according to Table 4.3.

Table 4.3: MalRank configuration for the experiment

MR Parameter	Description
$\forall r \in \mathcal{R} : \omega_r^o = \omega_{r^{-1}}^o = 0.4$	All edge weights initialized with 0.4 in both directions
$k = 0.75$	The maximizer factor for the calculation of $\hat{\omega}$
$s^o(x) = \begin{cases} 1, & \text{if } x \in X_{mal} \\ 0.0001, & \text{otherwise} \end{cases}$	Node x initial score depending on whether it was observed as an IOC by a TI source
$c_{s^o(x)} = \begin{cases} CTI_c, & \text{if } x \in X_{mal} \\ 0, & \text{otherwise} \end{cases}$	Prior's confidence, derived from TI confidence

MalRank has a mechanism to incorporate confidence in an indicator ($c_{s^o(x)}$). Almost all of our TI sources provided a trust level that we adjusted for the MalRank confidence scores during our experiment.

4.6 Case Study: Malware Detection

In this section, we present the details of our experiment, deploying MalLink in a real-world environment using EDR logs. The objective was to generate the EDR-based CyberHIN depicted in Figure 4.1 and utilize MalRank to detect new malware variants.

9 https://kubernetes.io/
10 https://spark.apache.org/
11 https://kafka.apache.org/

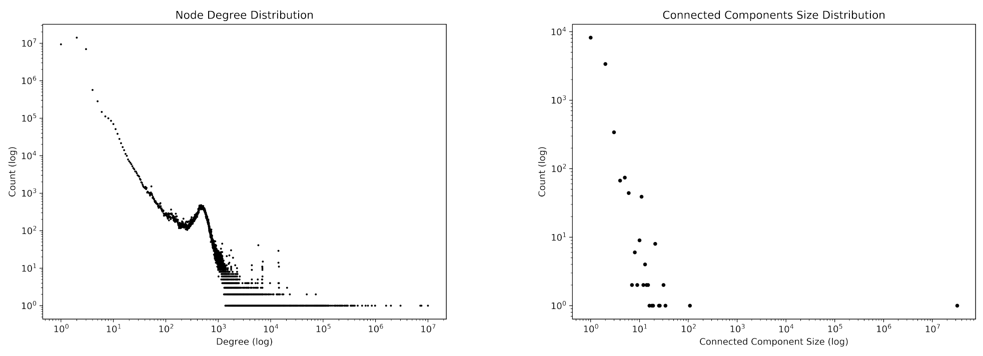


Figure 4.10: Node degree distribution and connected component size of the final graph in log-log scale.

4.6.1 CyberHIN Construction Details

Dataset— This research used 8 days of process activity logs collected by a large international enterprise SIEM, spanning over 20 TB. These logs were generated from a commercial tool similar to Sysmon. However, due to the lack of certain event types, our final generated HIN lacked certain entities and relationships, see Table 4.8.

The Indicators of Compromises (IOCs) that were used as the seed for known maliciousness during this experiment were collected from several commercial Threat Intelligence providers that were available within the company's landscape. In addition, several other resources were used to collect indicators (e.g., malicious files previously detected by the endpoint protection solutions or closed incidents). We focused particularly on IOCs related to our observed entities, i.e., *domains*, *IPs*, and *Files*. Furthermore, we limited the IOCs to those that were last seen during the experiment time frame. This is important as CTI is time-sensitive. In total, we collected 60 thousand *IPv4*, 19 thousand *File:sha256*, and 4 thousand *Domain* indicators. Note that one could also utilize various open-source and free resources [12].

The Generated HIN Characteristics

Before proceeding to the evaluation, it is essential to understand some characteristics of the final network. The final network was generated from 32 million unique vertices and 128 million unique edges spanning over 50 GB in memory. Table 4.8 depicts each entity and relationship counts after passing the raw data through the first two layers (PET and OSINT enrichment). Note the malicious count for each

12 https://github.com/hslatman/awesome-threat-intelligence

entity type. These values are the portion of the collected CTI that matched against the entities of the final generated HIN. For example, out of the 19 thousand total malicious file hashes collected, only 500 hundred were observed in our data.

Figure 4.16 shows the degree distribution and the connected component sizes. It is worth noting that the giant component of the generated graph (the point at the right side) accounts for 99.9% of the nodes in the graph. Additionally, all matched IOCs were also part of this giant connected component.

Table 4.4: The count of each vertex and edge type loaded into the final graph.

Edges		Vertices		
Type	#	Type	#	#Malicious
observedIn	70 million	Domain	14m	1.3k
resolvesD	18m	Registry:value	6.5m	-
RVTouch	6.6m	Registry:key	5.8m	-
hasValue	6.5m	File/Content/Process	4.3m	0.5k
pointsTo	6.2m	IP/Service	1.2m	2.5k
RKTouch	5.7m	System	134k	-
subDomainOf	5.6m	CIDR	70k	-
FTouch	4m	AS	12k	-
connectsTo	4m			
inRange	0.9m			
loads	0.8m			
spawns	0.4m			
assignedTo	70k			

4.6.2 Evaluation

Stratified Shuffle Split Cross-Validation

Like most related work in this domain, we decided at first to evaluate MalLink using cross-validation, particularly Stratified Shuffle Split Cross-Validation with different metrics discussed in Table 4.5. Note that MalLink requires seeds for maliciousness inference; thus, for each test, we divided the collected TI into seeds and test sets. More specifically, we passed the output of the *TI Marking* layer to an evaluation layer responsible for generating k (where $k = 10$) splits in which k% of each class (strata) is randomly sampled/marked for the evaluation. This resulted in the generation of 10 splits, each with 50 randomly sampled known malicious files for testing the positive class and 429950 unknown samples to test the negative class. The rest of the TI (1300 domains, 25000 IPs, 450 files) were used as the seeds. Note that while we used all IOC types (domains, IPs, files) as the maliciousness

Table 4.5: Metrics for evaluation [78]

Metric	Description
True Positives (TP)	Malicious item correctly scored as malicious
False Positives (FP)	Benign item incorrectly scored as malicious
True Negatives (TN)	Benign item correctly scored as benign
False Negatives (FN)	Malicious item incorrectly scored as benign
True Positive Rate (TPR)	TP/(TP+FN)
False Positive Rate (FPR)	FP/(FP+TN)
Accuracy	(TP+TN)/(TP+FP+TN+FN)
Precision	TP/(TP+FP)
Recall	TP/(TP+FN)
F1	2×(precision · recall) / (precision + recall)

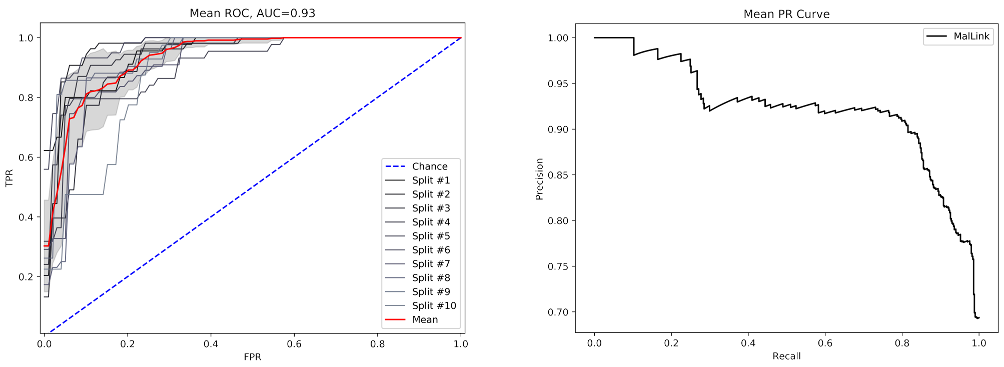


Figure 4.11: Average Receiver Operating Characteristics (ROC, left) and Precision-Recall (PR, right) curves for 10 Stratified Shuffle Split Cross-Validation

seeds, we sampled only files for testing the positive class (maliciousness). Lastly, we ran 10 iterations of the inference on each set, thus generating 10 outputs for evaluation (10 strata).

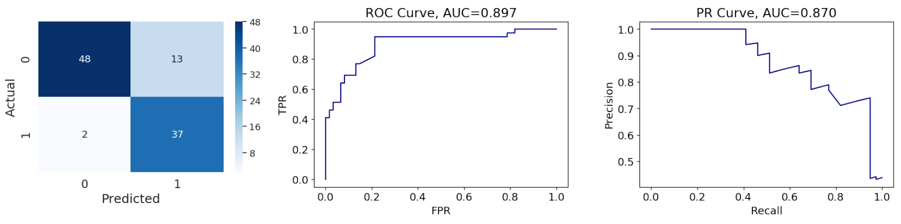


Figure 4.12: Confusion matrix (1 representing positive class, i.e., malicious), ROC, and PR curves for the manual evaluation of highest and lowest scored nodes.

Figure 4.11 shows the Receiver Operating Characteristics (ROC) curve plotting the True Positive Rate (TPR) versus the False Positive Rate (FPR) as the threshold varies through the range of data values for each testing set.

While the ROC Area Under Curve (AUC) of 0.93 seems impressive at first, it is not a correct metric for evaluation due to the significant skew in class distribution, thus testing. Intuitively in such a situation, we will end up testing a much larger number of benign/unknown samples for every malware sample, as the number of negative cases would tend to be much larger than the set of positive cases. Thus, increasing the chance of correct classification of a negative class (True Negatives) and dominating the ROC analysis with TNs, making False Positive Rate (FPR), not such a useful metric.

In this regard, it makes more sense to replace FPR with another metric that does not directly take TN, i.e., *precision* [245]. That is why Precision and Recall (PR) curves are much better suited in such an imbalanced setting. However, in that case, we can be very harsh on the algorithm as testing on that large number of unknown samples poses a much greater chance of finding FPs. Figure 4.11 shows the precision-recall curve for 10-split stratified shuffle cross-validation with a downsampled unknown class, i.e., before PR evaluation, for each split, the test samples for the unknown class were randomly reduced to match the size of the malicious class (50).

Previously Unknowns

While we attempted to evaluate MalLink similarly to the related work, i.e., using Cross-Validation (dividing the data into training and testing sets), it is not the most appropriate evaluation. That is not because we had to downsample but because the underlying problem is not a generic classification problem but rather an inference problem. In other words, the idea is not to classify every node as malicious or not (two-class classification) but rather to infer maliciousness (single-class inference). We explicitly tried to avoid using the term "benignness" as we are more interested in determining maliciousness through inference. That means that a false positive does not necessarily imply a wrong classification of a benign entity as malicious but rather that an unknown entity was sampled. Despite the expectation of unknown classification, the algorithm classified it as malicious, which could be, in fact, true. Furthermore, our labeled data (previously known malicious files) accounted for only 0.04% of all data.

To ensure our approach's effectiveness, we manually investigated the 50 highest and the 50 lowest-scored nodes with prior 0 (previously unknowns with no label). We decided to make this split to simulate a SOC analyst's reasoning and hunting.

In this regard, we expect an analyst to pick the top k nodes with the highest scores and no prior to find previously unknown malware samples. In contrast, the nodes with the lowest scores should represent highly reputable applications that should be benign.

For this investigation, we utilized various external sources such as Hybrid Analysis[13], VirusTotal[14], and ANY.RUN[15] as well as internal resources such as the enterprise SIEM system and other security solutions. We concluded maliciousness by examining whether the aforementioned external or internal resources suggested unanimously or, in their majority, clear maliciousness. In addition, the examination of our entity's behavior within the SIEM was also considered.

Figure 4.12 shows the Receiver Operating Characteristic (ROC) curve, Precision-Recall curve, and confusion matrix. As presented in the figures, out of 100 investigated files, we validated 37 to be true positives (F1-score of 0.85). This evaluation proves the ability of this approach to detect previously unknown malicious files.

Investigation of FPs and FNs

We decided to further investigate the false positives and false negatives, attempting to identify the reason for the misclassifications. In this regard, we noticed that the fundamental reason for the misclassifications was the ingestion of bad TI. Notably, we ingested anti-virus (AV) alerts as another source of known malicious files. However, at the time, we did not notice that some of the alerts were associated with the usage of a particular file, not the file itself (e.g., the command line being marked as malicious). Similarly, some of the other ingested commercial TI marked primary core Windows services as malicious due to their misuse by criminals.

These bad IOCs were the reason for the presence of FPs and FNs, which proved another hypothesis of this approach, i.e., *exempt-by-reputation*. Although certain nodes were falsely identified as malicious, not all entities connected to the bad TI were misclassified.

Figure 4.13 illustrates how bad TI affected MalLink inference. While CMD (representing a bad IOC) affects node M to be wrongly classified as malicious, it does not affect A, B, C, due to their high reputation (association to other entities). Interestingly these FPs were files that had a similar structure to node M; however, they were only observed on a few hosts that were not necessarily malicious such as gaming clients.

13 https://www.hybrid-analysis.com/
14 https://www.virustotal.com/
15 https://any.run/

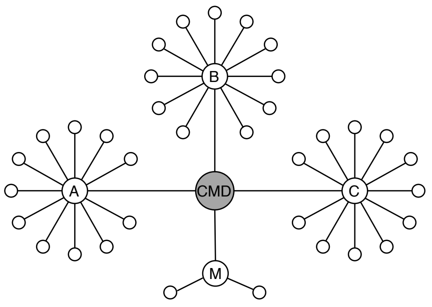


Figure 4.13: Illustrating how node *M* was classified as malicious (false positive) due to a bad TI (*CMD* ingested as IOC) influencing all its neighborhood, while nodes *A*, *B*, *C*, were not influenced greatly due to other connections, node *M* got influenced greatly due to limited association to other nodes.

Comparison with Polonium

Lastly, we decided to compare our approach with one of the most related works, Polonium [40]. While an exact comparison is impossible due to proprietary components of Polonium, e.g., machine reputation or file prevalence, 2) the difference between the underlying data sources, 3) the labeling mechanism, we tried to keep the comparison as close as possible. In this regard, we followed a similar process to that described in Figure 4.9. We limited our constructed HIN to only file observations in a system, attempting to build the undirected bipartite machine-file graph of Polonium. The underlying algorithm used in Polonium is Belief Propagation. In this regard, we adopted Hewlett Packard's implementation of BP in Apache Spark [161]. BP was then configured as shown in Table 4.9. Since we intentionally did not use benignness in our experiment, we adapted Polonium proposals for node potential to only malicious and unknown labels.

Table 4.6: Node and edge Potential configuration for Polonium experiment.

Edge Potential			Node Potential		
$\psi_{ij}(x_i, x_j)$	$x_j = unk$	$x_j = mal$	Node	$P(mal)$	$P(ukn)$
$x_i = unk$	$0.5 + \epsilon$	$0.5 - \epsilon$	Malicious	0.98	0.02
$x_i = mal$	$0.5 - \epsilon$	$0.5 + \epsilon$	Unkown	0.5	0.5

Figure 4.14 shows the ROC and PR curve comparison of Stratified Shuffle Split Cross-Validation for MalLink and Adapted Polonium. While it is difficult to identify the exact reason, it is possible to observe the overall MalLink's superiority compared to Polonium. In our future work, we would like to explore the reason for

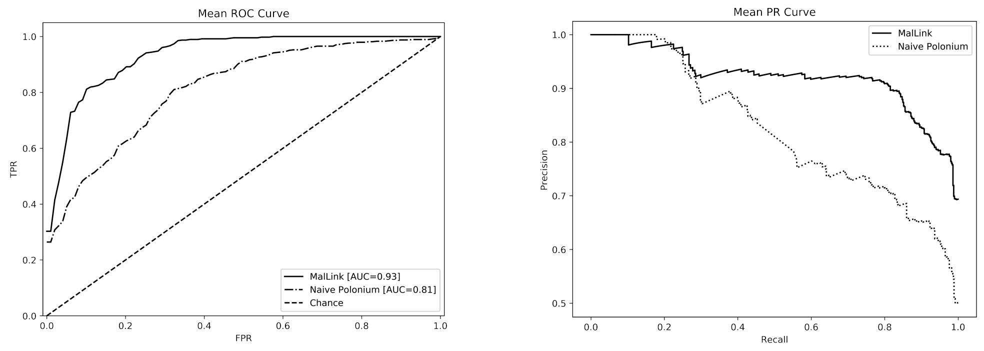


Figure 4.14: The mean ROC and PR curves for 10-Stratified Shuffle Split Cross-Validation compering MalLink and Polonium [40].

such improvement further, particularly by examining the effect of switching the underlying algorithms and measuring the effectiveness of each relationship in our HIN.

4.6.3 Discussions

Convergence and Effect of Iterations

Although the inference algorithm is not exact for HIN, if one continues the iterative calculation without a guarantee for convergence, the scores will mostly converge to a stable equilibrium as time progresses. This phenomenon is similar to what we observed in loopy belief propagation [184]. Figure 4.15 shows the transformation of scores and the maximized/minimized edge weights ($\hat{\omega}$) over iterations coming to a stable equilibrium after the first few iterations. Moreover, the figure highlights the direct effect of a change in a node's score on the weight of its outgoing edges.

Resilience to Inaccurate Threat Intelligence

When investigating false negatives, we noticed how MalLink could, to some extent, correct the impact of incorrect IOCs. Therefore, we decided to check whether this approach could be potentially used to evaluate the accuracy of IOCs. In this regard, we adjusted our original hypothesis to examine IOC quality. In particular, we hypothesized that an indicator is more malicious if it shares certain characteristics with other malicious entities and less malicious if it is highly reputable across the landscape.

To evaluate this hypothesis, we experimented with known malicious files, especially those with the most significant drop in their maliciousness score. Note that,

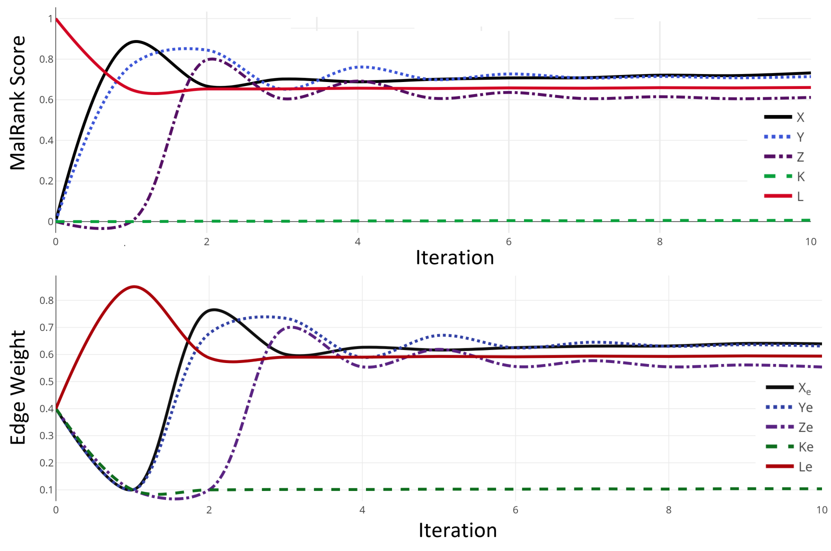


Figure 4.15: Node scores and edge weights over iterations. Node X is one of the high-scored nodes, and node Y is a high-scored node with a high degree (163). Node Z is a high-scored node with no immediate malicious neighbor. K is a low-scored node, and L is a TI node with a significant adjustment in its prior (i.e., bad TI node). X_e is one of the randomly selected outgoing edges of node X.

as mentioned in the experiment setup section (Table 4.3), we set the CTI nodes' prior ($s^o(x)$) equal to 1; however, with different confidence ($c_{s^o(x)}$) according to the TI source confidence score (provided by the sources itself). These scores were mainly set from 0.6 to 0.9. Since these confidence values were less than 1, the corresponding nodes' scores could drop in theory.

We picked those nodes with prior 1 that had the most drop in their score, bringing it closer to $s^o(x).c_{s^o(x)}$. This indicated that the algorithm did not believe the node to be as malicious as it was initially considered. For example, a node with prior 1 and confidence 0.7 would have a final score of 0.7.

Investigating the top 20 files with the mentioned behavior presented compelling observations: First, the approach's capability to identify bad indicators. In this regard, we noticed several bad ingested TI, such as "SVC host" or "CMD", which are not malicious themselves but rather have the potential to be misused for malicious activities, thus scored down due to their association with a large number of entities (thus high reputation). A similar observation was made when one of the TI sources, particularly the AV, alerted on files that were part of a tailored solution of that company, hence, scoring them low due to their presence in a large number of hosts. Second, irrespective of an entity's maliciousness, if it's associated with a large number of nodes (e.g., registries and files), its maliciousness score would be reduced. This points to one of the limitations of this work, i.e., adversarial attacks and manipulation, further discussed in the limitation and future work section.

In summary, we have seen opportunities to use MalLink's approach to increase and measure the quality of CTI whilst acknowledging how certain factors of the algorithm can be potentially misused to force a good score. This explains how one has to better engineer initial edge weights and directions to combat such misuse, especially those that can easily be misused.

4.7 Case Study: Malicious Domain/IP Detection

This section presents a detailed description of our experimental approach, which involved deploying MalLink in a real-world setting utilizing Proxy and DNS logs. Our primary objective was to generate the Proxy and DNS-based CyberHIN as illustrated in Figure 4.2, with the purpose of utilizing MalRank to detect novel malicious domains and IPs.

4.7.1 CyberHIN Construction Details

Dataset— For the purpose of this research, we used two days of Proxy, DNS, and DHCP logs (almost 3 billion events) collected by a large international enterprise SIEM, spanning over 3 TB. Table 4.7 shows the statistics related to the raw event logs.

Table 4.7: The statistics of our SIEM logs for the experiment

Source	Size	#Events
DNS Logs	2TB (120GB gzip compressed)	2 billion
Proxy Logs	1TB (100GB gzip compressed)	755 million
DHCP Logs	12GB (800MB gzip compressed)	4m

The described event logs were later enriched with related OSINT as described in Section 4.2, i.e., ASN, X.509 certificates, and DNS RRs. In this regard, we used the sanitized version of the BGP prefixes, origin ASNs[16], and ASN to organization name mapping[17] available at thyme.apnic.net. These files span approximately 20 MB in total. For X.509 certificates, we used a Censys[18] IPv4 snapshot which contains a full

16 http://thyme.apnic.net/current/data-raw-table
17 http://thyme.apnic.net/current/data-used-autnums
18 https://censys.io/

portscan of the entire IPv4 address space. This data spans over 1.2 TB of disk space. Note that we were only interested in port 443 scans. Alternative sources for X.509 certificates are Certificate Transparency Logs and the automated collection of TLS handshakes. Due to the enterprise's configuration for DNS servers to not log the DNS responses (DNS RRs), we had to pass all the DNS queries (logged by the DNS server) to our active OSINT-DNS enricher (based on Gieben's DNS library[19] and MassDNS[20]) and log the responses ourselves.

Lastly, we utilized various sources (e.g., Google's Safe Browsing, malwaredomains.com, etc.[21]) to collect our threat intelligence that was used as the ground truth throughout our experiments. Ultimately, we managed to passively collect a total of 1.5 million malicious indicators (domains and IPs) and 1 million benign domains (from Cisco's top 1 million domains, one can also use Alexa's top 1 million domains). Note that our algorithm does not rely on benignness, and this list was collected only for evaluation.

HIN Characteristics

After passing the described data set through the event processor, 13 million vertices and 122 million edges were extracted. This process took approximately 4 hours on the described research workbench setup. Next, after passing through the enrichment layer, an extra 6 million vertices and 12 million edges were added, making a total of 19 million vertices and 134 million edges passed to the loading module. The enrichment layer processing time was about 2 hours. After the loading module stage, the final graph was created with 15 million unique vertices and 132 million unique edges.

Table 4.8 shows the count of each vertex and edge type, and figure 4.16 shows the degree and component distribution of the created knowledge graph. The distribution follows a power-law distribution, indicating an extremely sparse graph with very few edges between the majority of the node and a minority with high-degree connected clusters. This is understandable since most of our relationships enforce low degrees when we have no global view of the association but rather an enterprise-level view of only observed entities. The majority of high-degree nodes were entities associated with the enterprise itself, e.g., enterprise domains and workstations.

19 https://github.com/miekg/dns
20 https://github.com/blechschmidt/massdns
21 https://github.com/hslatman/awesome-threat-intelligence/

Table 4.8: The count of each vertex and edge type loaded into the final knowledge graph.

Edges		Vertices	
Type	#	Type	#
requestedAccessTo	103 million	Domain	12.4m
subDomainOf	10m	Ipv4	1.8m
resolvedTo	7m	Organization	0.28m
uses	3m	X509cert	0.27m
aliasFor	2.5m	Mac	0.12m
referedTo	2m	ipRange	0.08m
associatedWith	1.7m	Useragent	0.07m
isInRange	1.3m	Asn	0.02m
mailServerFor	1m		
issuedBy	0.26m		
issuedFor	0.26m		
signedBy	0.23m		
nameServerFor	0.12m		
assignedTo	0.08m		
belongsTo	0.03m		

4.7.2 Evaluation

Previously Known Malicious

Those are nodes that were known to be malicious at first (e.g., indicated by a TI source). However, they were marked as unknown (prior set to 0) when passed to the algorithm so that later they could be utilized to evaluate the algorithm detection capability (i.e., the testing set). Following the standard practices, in order to evaluate the detection capability of our approach, we decided to utilize a Receiver Operating Characteristic (ROC) curve as well as the Precision and Recall (PR) curve.

It is worth remembering the testing set could only be derived from the 20K labeled data points that are connected. This is due to the fact that, first, the rest of the data points did not have any label in the first place that could be used for evaluation. Second, taking random samples in a sparse graph with a low degree distribution could result in samples that have no connection to any other labeled nodes, thus eliminating the ability to evaluate the inference. Figure 4.17 illustrates this idea. If black nodes are previously known malicious nodes and white nodes are unknown (unlabeled) nodes, we only consider nodes such as A and B for our test samples, as choosing X or Y will give us no value because they are not connected to any other labeled nodes to allow effective maliciousness propagation. More specifically, to select the testing samples for a class (e.g., maliciousness) in an evaluation run, the loading module first calculates the connected components (clusters) for that class,

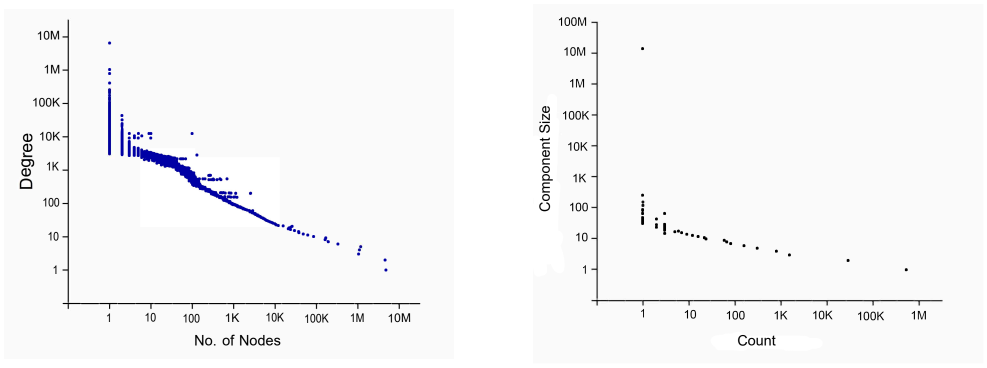


Figure 4.16: Node degree distribution and connected component size of the final graph in log-log scale.

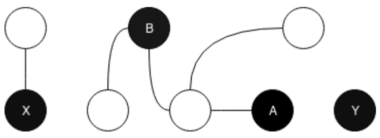


Figure 4.17: Pseudo-random sampling for the purpose of evaluation. In this regard, to select the testing set, we only consider connected nodes such as A and B.

next, from each of those clusters that have more than one member, selects $\sqrt{k}$ nodes at random (where k is the number of labeled nodes in each cluster). For instance, if Figure 4.17 is our knowledge graph, we would take only A or B at random. In our experiment, this process led to the random selection of approximately 2, 000 nodes (1, 000 known malicious and 1, 000 known benign nodes) in each evaluation run.

Figure 4.18 shows the ROC and PR plot for 9 iterations of MalRank with the configuration described in Table 4.3 on the described data set with the described testing set. Note that the whole experiment (including the sampling) was repeated 4 times to flatten out the outliers. The results show a high accuracy (AUC = 96%, with the peak *F1-score* = 0.905, and *Accuracy* = 0.900)

In order to better understand the algorithm's results, we investigated the false positives (FPs) and false negatives (FNs). In this regard, we had the following observations; first, in contrast to our original intuition and the common practice used in past efforts, top-ranked domains by Cisco Umbrella or Alexa did not necessarily reflect benign domains. In this regard, there were multiple instances of domains being marked as malicious after MR due to association with multiple malicious entities despite appearing on Umbrella's top 1 million domains (therefore FP). This was also confirmed by [222]. A good example of this was *world.rickstudio.ru*, which appeared among the top 1 million domains under the umbrella while being ma-

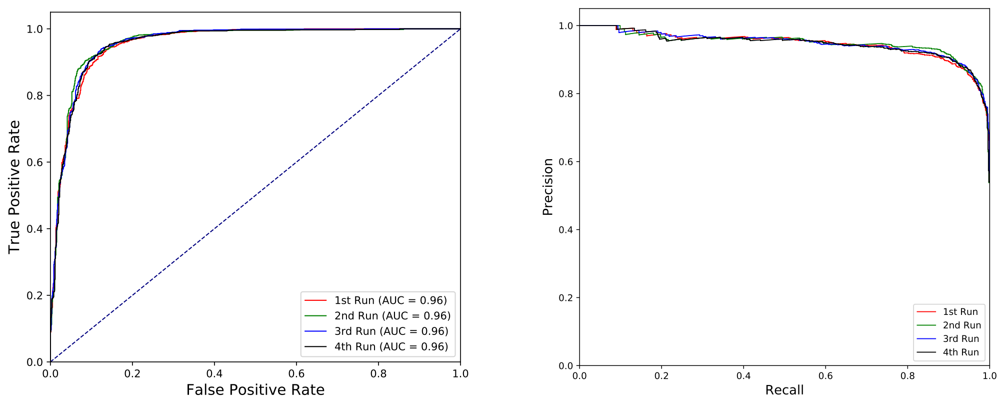


Figure 4.18: Receiver Operating Characteristic and Precision-Recall curves of 9 iterations of MalRank ran 4 times.

licious. This was also due to our random selection of benign samples from the entire 1 million, one should at least ensure that the samples are from the top *k* thousand. Other legitimate false positives were due to the association of malicious IPs to benign domains, this could be explained by the web hosters that might share IPs among domains.

The Majority of the false negatives were also due to bad-quality Threat Intelligence. For instance, ingesting a TI source where *github.com* and *google.de* were marked as malicious by the TI. These were later marked as non-malicious by our algorithm due to their association with major neutral nodes. Other FNs were also due to content delivery networks (CDN) in which a TI was reporting an IP malicious while it was also associated with a couple of legitimate domains through a proxy server.

Investigating these FNs and FPs confirmed one of our initial intuitions in which we hypothesized that MalRank could also be used to improve the quality of Cyber Threat Intelligence. We would like to explore this further in our future work.

Comparison with Belief Propagation

In order to evaluate MalRank's efficiency and accuracy, we decided to compare it with Hewlett Packard's implementation of Belief Propagation implemented in Apache Spark [161]. BP is the most popular algorithm used throughout the literature as a graph-based inference algorithm in the context of security.

Table 4.9: Node and edge Potential configuration for Belief Propagation experiment.

Edge Potential			Node Potential		
$\psi_{ij}(x_i, x_j)$	$x_j = ben$	$x_j = mal$	Node	$P(mal)$	$P(ben)$
$x_i = ben$	$0.5 + \epsilon$	$0.5 - \epsilon$	Malicious	0.99	0.01
$x_i = mal$	$0.5 - \epsilon$	$0.5 + \epsilon$	Benign	0.5	0.5

Figure 4.19 shows the ROC plot for 9 iterations of MR vs. 9 iterations of BP on the same knowledge graph with the same testing set.

Table 4.9 and 4.3 shows BP and MR algorithm configuration for this experiment, respectively. It is also worth noting that due to the fact that MalRank utilizes only one class label (maliciousness), whereas BP requires at least two class labels (maliciousness and benignness), we decided to configure BP by initializing all unknown and benign nodes with 0.5 as the probability of maliciousness (i.e., $P(x_{mal}) = P(x_{ben}) = 0.5$) and 1 for those previously known malicious nodes (i.e., $P(x_{mal}) = 1$, $P(x_{ben}) = 0.5$).

As shown in Figure 4.19, MR is outperforming BP not only in terms of accuracy but also the run-time. In this regard, with almost an identical implementation in GraphX, MalRank finishes 9 iterations within 20 minutes, whereas BP takes about 2 hours. It is worth mentioning that BP memory utilization was almost 6 times higher than MalRank. It is also important to note that while BP shows high accuracy, this is not true in all cases. The main reason for this accuracy in this experiment is that the majority of the testing set were nodes with a low degree, and as we discussed in the previous sections, BP starts introducing errors as the node's degree increases. This was observed mostly in our next experiment when investigating previously unknown threats.

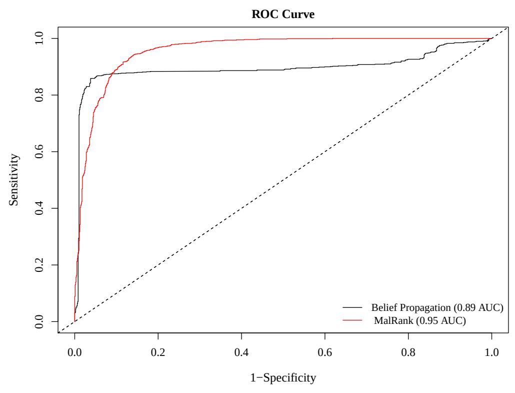


Figure 4.19: ROC curve for 9 iterations of MalRank vs. Belief Propagation.

Investigation of FPs and FNs

ROC and PR curves are useful for evaluating a threat detection technique. Nevertheless, plotting such curves requires a testing set, and as mentioned before, the choice of the testing set for our approach is a challenging task. More specifically, our approach is not a generic classifier that can classify any arbitrarily given entity as malicious or non-malicious. Instead, it is an inference model designed to increase the quantity and the quality of our threat intelligence by discovering new malicious entities associated with previously known malicious entities. Therefore, the most relevant validation for us is the evaluation of previously unknown and inferred maliciousness. In this regard, we decided to manually investigate top high MalRank-scored nodes that did not have a prior (not observed in our TI). For this manual investigation, we utilized *ThreatCrowd, VirusTotal, ThreatMiner, URLVoid, AlienVault, Robtex and MXToolBox*. We categorized our investigation depending on the type of entity.

When investigating the top 200 Domains/IP, we were able to find an indicator for 67% of those. While the majority of those were the result of maliciousness inference on *resolvesTo* relationship, there were those high-degree nodes that were scored mostly due to *mailServerFor*, *isInRange*, and *referedTo*. As a result, we were able to identify a large number of previously unknown malicious domains and IPs. We were also able to identify a surprising number of pornographic domains that were ranked high. We assume this is due to malvertising, and clickjacking techniques widely adopted by such domains.

When investigating the top high-scored X.509 certificates, we were mostly capable of identifying parking domains (i.e., domains registered solely for the purpose of displaying web advertisements with typically no real content [147]) and rogue web hosters (e.g., **.000webhostapp.com* whom its subdomains are regularly misused by cybercriminals to host scams). Hence allowing us to capture further potentially unknown malicious Domains/IPs.

We had the same observation when investigating the top malicious organizations, as one of the top MalRanks scored ones was the organization responsible for **.000webhostapp.com*. We were also able to identify a number of self-signed certificates associated with an organization which led us to find associated Domains/IPs which were in fact, classified as malicious by VirusTotal.

We didn't investigate MAC address, ASN or User Agent (UA) as the majority of nodes did not come up with high scores (less than 0.2). This was reasonable, considering we had only access to two days of data.

Lastly, we investigated a set of malicious domains which was identified by the enterprise's SOC analysts to be associated with malware beaconing on a number of clients. More specifically, these were domains starting with imp (i.e., `^imp\\..+`) such as *imp.searchlff.com*[22]. When we checked the MalRank score of these previously unknown malicious domains, we noticed that the algorithm scored them as malicious (0.6 - 0.7) due to association with TI (through IP address and range sharing).

Interestingly, when we investigated the BP score for the above findings, we could verify our initial intuition, i.e., BP's limitation to infer maliciousness for high-degree nodes with unbiased labeled neighbors. In this regard, the majority of previously unknown malicious entities were scored between 0.51-0.56, which makes them susceptible to misclassification by BP.

In summary, although we were unable to validate all high-score nodes, according to our investigations, MalRank proved to be an effective method to increase the quality and quantity of threat intelligence. While one could argue that these were low-hanging fruits, we still see the value in our approach. Furthermore, It is also worth noting that the SIEM logs used within this research were from an international enterprise that already utilizes various security measures and practices (e.g., IDS, AV, Proxy/DNS blacklisting, signature checking, and etc.) therefore making it rare to encounter various threats, yet MalRank was capable of detecting valuable previously unknown malicious entity, i.e., the detection of a potential malware beaconing case.

It is worth noting that, throughout our investigation, we came across a number of nodes and cases in which the nodes were scored high (malicious), but we could not validate the maliciousness as it seemed harmless (e.g., parked domains, link farms, and other dubious domains/IPs). Even though such entities seemed non-malicious (FPs), we argue that blocking them at the enterprise level should not have a drastic effect, as the main reason for their false classification was having a number of associations with high-scored nodes.

4.7.3 Discussions

Number of Iterations

In the majority of our experiments, we chose 9 as the maximum number of iterations. The main reason for this choice is that, within our knowledge graph, the inference from more than 4 hops away does not make much sense. Consider the *requestedAccessTo* edge isolated from all the others. This edge captures the relationship between a MAC address and a set of domain/IP nodes (shaping a bipartite graph). In order to decide the label for a domain, it makes sense to traverse back

22 https://www.threatcrowd.org/domain.php?domain=imp.searchlff.com

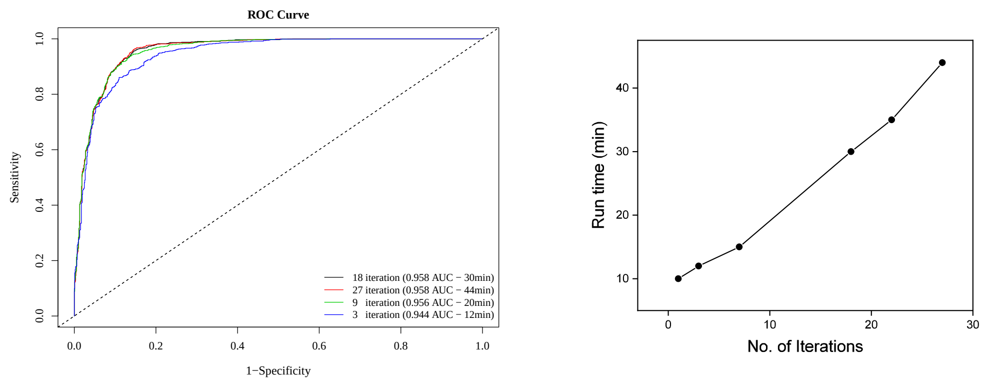


Figure 4.20: Receiver Operating Characteristic and Precision-Recall curves of 9 iterations of MalRank ran 4 times.

to the MAC address that requested this domain and check the score for all other domains visited by that MAC (perhaps an indication of malware trying to reach out to malicious IPs or Domain for C&C), i.e., inference from two hops away. It also makes sense to check another two hops, i.e., check whether there exist other workstations which connect to similar known malicious nodes. However, going deeper than that loses the intuition entirely. This could also be observed in Figure 4.20, which shows the ROC curve for a different number of iterations. As shown in the figure, while the results do not vary drastically after 7 iterations, the algorithm runtime increases drastically. For instance, when we change from 9 to 18 iterations, we increase the accuracy by 0.2% and the run-time by 50%.

4.8 Case Study: Maliciousness in Common Crawl Web Graph

The utilization of private data in our main experiment poses a significant challenge for the evaluation of MalRank, especially given that the algorithm itself is a key contribution to this study. As a result, we attempted to apply MalRank to a public dataset. Regrettably, we were unable to replicate our SIEM-based knowledge graph due to the absence of publicly available and recent Proxy and DNS. While one can find Proxy and DNS logs, we cannot use them due to the fact that half of our cyber graph is composed of OSINT and TI, which are challenging to obtain for events logs that are several years old.

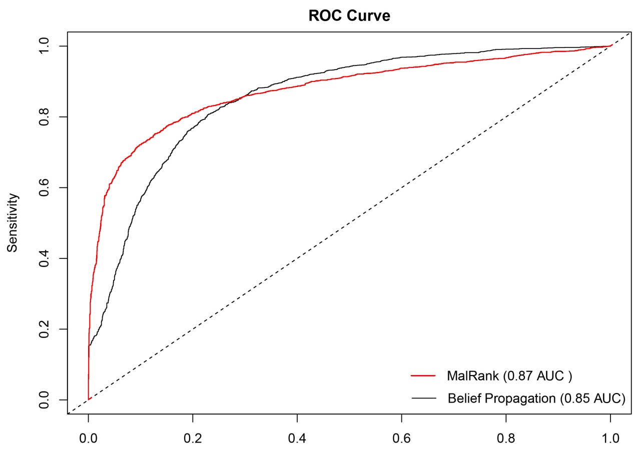


Figure 4.21: ROC Curve comparison between 3 iterations of MalRank and 3 iterations of Belief Propagation on CommonCrawl web graph.

The CommonCrawl web graph[23], which consists of host-level and domain-level web crawls, was the most relevant alternative graph structure data that we found online. We employed the most recent version of the CC domain-level web graph (Feb 2018) in our system and labeled the nodes with the same TI that we collected previously. We subsequently executed both MR and BP and compared the outcomes. The generated graph's size was around 600GB in memory, with a high degree of connectivity. This caused MR to take several hours, and BP took a day to complete three iterations. Figure 4.21 displays the ROC curve for this experiment.

4.9 Limitations and Future Work

CTI Quality

The quality of Cyber threat intelligence (CTI) is a crucial factor that affects the effectiveness MalRank. In our work, we encountered significant limitations due to the quality of CTI, which resulted in a high false positive rate. MalRank relies on a small set of previously known malicious nodes as seeds to infer maliciousness. These seeds are expected to be validated CTI. However, most publicly available CTI sources exhibit low quality, containing a significant number of false positives. Utilizing such false TI can lead to further false positives, thus negatively impacting the overall performance of MalRank.

23 http://commoncrawl.org/

Although MalRank incorporates a mechanism to incorporate the quality of CTI, our experiments lacked an approach to evaluate the sources of TI, resulting in a significant number of false positives caused by the ingestion of bad TI from a source. Future work should focus on using better-quality CTI, even if that means imposing some API limitations.

Additionally, we propose the utilization of MalRank as a CTI validation and measuring algorithm first before using it for threat detection. In other words, run MalRank to find the confidence of the CTI seed nodes and run it a second time with the adjusted weights, thereby enhancing its overall effectiveness in detecting and mitigating malicious activities.

In addition, we suggest using MalRank as a tool to validate and measure Cyber Threat Intelligence (CTI) before utilizing it for threat detection (as discussed in Section 4.6.3). This involves running MalRank once to determine the confidence levels of the CTI seed nodes and then running it again with adjusted weights based on the initial results. By doing so, MalRank can better detect and mitigate malicious activities, thereby improving its overall effectiveness.

Adversarial Attacks

A significant concern with our approach is the potential for adversaries to influence MalRank scores. For example, a malicious entity might attempt to increase its reputation by forcing associations with a large number of other entities, thereby reducing its score. This issue was observed when investigating false negatives (FN).

To address this, it is crucial to set edge weights in a way that makes it difficult for malicious entities to manipulate their reputation. Specifically, relationships that can be easily abused should receive lower weights. This approach increases the effort required for malicious entities to build their reputation and makes manipulation more challenging.

Despite the algorithm's capability to support directional edge weights, we relied on naive expert knowledge during our experiments and specified edge weights in both directions as 0.4. MalRank has a mechanism to adjust weights within each iteration, reducing the effect of benignness. However, for future work, we plan to investigate the importance of each edge and its direction, hypothesizing that not all edges should be treated equally. Some edges are more important than others for malicious propagation, and the weights of both directions of an edge should not be the same. Careful setting of edge weights should allow us to combat adversarial attacks more effectively.

HIN Schema Expansion

Our research focused on process and networking logs, explicitly modeling the main entities and relationships present in EDR, proxy, and DNS logs. However, there are still some areas that we have not explored in detail. For example, parsing command line or registry values, extracting additional entities and relationships, examining remote executions and services (e.g., via WinRM, WMI) that are frequently exploited by attackers, and incorporating scheduled tasks and services could be valuable research directions. Additionally, we simplified the original HIN due to limited data sources during our experiments. Therefore, investigating different HIN network schemas to identify new entities and relationships and further correlating them with other log types and open-source intelligence may yield exciting results.

Algorithmic Improvements

We would also like to explore MalRank algorithmic improvements. More specifically, first, finding the closed formula in terms of matrix operation should improve its efficiency and better reason for its convergence. Second, expand the naive incremental MalRank to better incorporate time and node/edge expiry for truly incremental calculation. Last, experimenting with other aggregator functions such as LSTM and Pooling [95] as opposed to the weighted average.

Graph-based Outlier Detection— In our study, we approached threat detection as an inference problem that relies heavily on the seed CTI. Without validated CTI, it becomes impossible to determine the maliciousness of a node. For example, suppose we have two nodes, one with only five connections to unknown neighbors and the other with over a thousand connections. In this scenario, MalRank will score both nodes equally at zero since there is no evidence of maliciousness. Due to the challenge of acquiring validated CTI, it may be worthwhile to investigate a modified version of MalRank that puts more emphasis on exception-by-reputation.

Benignness— As previously mentioned, we focused specifically on single-class diffusion, or the propagation of maliciousness, and considered all other nodes as benign unless they were guilty by association or intern except by reputation. However, it would be an interesting experiment to include validated benignness in our approach.

Incremental Mode

Our experiment was designed as a batch job, i.e., one would have to run the whole system (*PET*, *OSINT Enrichment*, *Graph Loading*, *TI Marking*, *Graph-based Inference*) every time a new node or link is added, or a prior is modified.

In a real-world setting, such a large-scale re-run is computationally intensive and redundant, especially considering how such updates should only affect a certain portion of the graph. In this regard, we would like to outline a naive incremental mode of the proposed approach 4.3.5, which shall operate in a streaming mode as opposed to batch, with the underlying intuition that an update should result in minor changes in the immediate neighbors of those inserted/updated nodes and edges. At the same time, the effect will not be strong enough to propagate to the rest of the graph [88, 195].

Following Figure 4.9, at a high level, as new logs are presented, the *PET* extracts and prepares the desired entities and relationships, *OSINT Enrichment* enriches certain entities, *Graph Loading*, updates the previous graph with the new set of relationships and entities, *TI Marking* changes priors if necessary (by removing adding TI on existing nodes or to the newly added nodes). Lastly, considering the newly added/updated vertices, $\mathcal{V}^{'}$ each with their own prior, and set of edges $\mathcal{E}^{'}$, *Graph-based Inference* re-calculates the score of all nodes that are at most h hops away from any node in set $\mathcal{V}^{'}$, or $h-1$ away from any source or destination vertex of an edge in set $\mathcal{E}^{'}$, where h is configurable noting the trade-off between computational cost and influencing the larger portion of the graph.

Ensemble Systems

Integrating our approach with previous works that focused on local features would be interesting. One could utilize the local features to derive an initial score for each node (e.g., 0.5) and then run MalRank to obtain the MR scores. By looking back at the nodes, if the MR score was increased further, it would be possible to conclude maliciousness with higher confidence, as the node was marked malicious based on not only its local features but also global ones.

An alternative approach would be to combine MalRank scores with other features and train another machine learning (ML) classifier to improve detection accuracy further. We anticipate that by ensembling MalRank with other approaches, the false positive rate could be significantly reduced.

Graph-based Threat Hunting

All our investigations of high-scored nodes were so far manual, which can be an exhausting task. It would be beneficial to have an additional layer in our system that sorts and enriches high-scoring nodes and their connected components based on their degree and type. This layer could actively gather further threat intelligence (TI) and open-source intelligence (OSINT) and present the cases as a subgraph that can be more easily evaluated by a SOC analyst. For example, for a high-scored domain, this layer could check it against VirusTotal and ThreatMiner APIs to obtain better quality TI, enrich it with registrar details, and present the domain its connected components, and add OSINT/TI as a subgraph. In our research, we focused on collecting OSINT/TI at scale and did not discuss active enrichment. However, for evaluation purposes, as the number of potentially malicious nodes is lower, it is possible to enrich the results further with active queries.

4.10 Literature Review

Significant research exists on the application of data mining for malware detection [244, 263, 281]. This section will only focus on the most related work, particularly those that mine host-level event logs and primarily consider intrinsic and global characteristics for threat detection.

Malware Detection via Graph Inference

Chau et al. [40] introduce Polonium as one of the first and arguably the most innovative and successful works that tackle the problem of malware detection using large-scale graph inference with the intuition that, good applications are typically used by many users, whereas unknown (i.e., potentially malicious) applications tend to only appear on a few computers. The authors achieve this by running an adapted version of Belief Propagation [283] on an undirected, unweighted bipartite machine-file graph. The authors perform a large-scale evaluation of Polonium over a real, billion-node machine-file graph, demonstrating the capability of such an approach in detecting previously unknown malware.

Ye et al. [282] hypothesize that combining file content (API calls) and file relationships (co-occurrence of files, e.g., shared across multiple clients, can improve malware detection. The authors formulate the classification problem as a graph regularization framework. In later research, Chen et al. [41] extend Ye's work by focusing solely on file relation graphs and adopt belief propagation as the inference algorithm. Tamersoy et al. [257] propose Aesop, which tackles the same problem using locality-sensitive hashing to measure the similarity between files to eventually construct a file-file bipartite graph and running BP to infer files' goodness based on its neighbors.

Other similar efforts include: Kwon et al. [149] proposing malware detection using features derived from downloader-graph. The authors show that the graph structures for download activity on hosts differ when comparing benign and malicious downloads. Karampatziakis et al. [125] explore maliciousness propagation via a logistic regression classifier. The authors focus on file-file relationships based on co-occurrences of files within a certain containment, such as a zip archive. Invernizzi et al. [115] build undirected neighborhood graphs using HTTP traffic on each host, hypothesizing the shared behavior across IP addresses, domain names, FQDNs, URLs, URL paths, file names, file hash, and hosts can aid with malware drive-by.

The underlying hypothesis of our work is similar to those mentioned above, particularly Polonium. However, while they focus on bipartite graphs (e.g., file-file, file-machine) to infer maliciousness based on co-occurrences of files within a system/machine, we construct a comprehensive heterogeneous information network. We emphasize only the co-occurrences of files within a system/machine as the shared attribute but also on multiple other global attributes such as registry keys touched, files written, and network connections made to improve the inference process.

While the majority of these works focus on bipartite graphs (e.g., file-file, file-machine), we construct a comprehensive heterogeneous information network emphasizing not only the co-occurrences of files within a system/machine but also multiple other global attributes such as registry keys touched, files written, network connections made, to improve the inference process. Furthermore, in comparison to the previous works which adopt Belief Propagation [283] as the core, we adopt MalRank [186], which allows us to better tackle a number of issues such as the incorporation of directed edge weights, reduction of the score based on the number of neighbors, and most importantly the resilience of the high skew in a graph with large biased towards benignness.

Threat Detection via Graph Embedding

With the recent advancement in graph embedding [84, 93, 95, 206, 258], a number of works have utilized graph embedding techniques to tackle malware detection. Fan et al. [76] utilize both content- and relation-based features for malware detection. To extract relation-based features, the authors propose the construction of a HIN. Entities are file, archive, machine, API, and DLL. Relationships are file-archive, file-machine, file-file, API-DLL, and file-API relations. The authors evaluate different graph embedding techniques such as DeepWalk, and LINE and eventually propose metagraph2vec to learn the low-dimensional representation of nodes in the HIN.

Peng C et al. [204] propose MalShoot, utilizing LINE algorithm [258] to embed a domain-IP bipartite graph, attempting to detect malicious domains using passive DNS. Wenxuan et al. [101] address the same problem but emphasize more on the construction of a domain-domain graph and utilize a modified version of Deep-Walk as a graph embedding algorithm. Yuta et al. [127] propose the usage of a Graph Convolutional Network (GCN)-based method attempting to estimate the maliciousness of IoCs.

In this research, we consider graph embedding techniques out of context. Graph embedding is a new field based on the advancement of neural networks and deep learning. While they have shown promising results in other fields [33, 92], the lack of explainability and parallelization makes it challenging when applied in the Cybersecurity domain, considering the adversarial setting and highly skewed and noisy data [197]. We leave this to our future work to evaluate different graph embedding techniques and compare them to our proposed inference and propagation approach.

Wang et al. propose PROVDETECTOR [270], a system that aims to detect stealthy impersonation malware using kernel-level provenance monitoring to capture the dynamic behaviors of each target process. The system models process activity data, i.e., a program's runtime behaviors such as file read, write, execute, or network connections as a Directed Acyclic Graph (DAG). Next, these DAGs are embedded using Doc2vec and are eventually passed through Local Outlier Factor (LOF) to detect anomalous patterns, hence previously unseen attacks. Several other works have taken a very similar approach to PROVDETECTOR, [62, 99] exploring anomalous process behaviors.

Malicious Domain Detection via Graph Inference

Manadhata et al. [167] address the problem of malicious domains/IPs detection using enterprise HTTP proxy logs. A host-domain bipartite graph is then constructed to capture workstations' connections to external domains, BP is then run to infer maliciousness from a set of seed malicious nodes. The authors show the effectiveness of graph inference for malicious domain detection. Khalil et al. [130] address the same problem using passive DNS data, focusing on a domain-IP bipartite graph. Several other works adopt Belief Propagation as an inference algorithm attempting to infer maliciousness in graphs constructed from DNS or Proxy logs [131, 187, 293].

There exists further research that tackles the same problem (malicious domain/IP detection via DNS, Proxy data), but by taking a different graph mining approach, e.g., meta path-based transductive classification [251]. Najafi et al. [186] propose MalRank, a graph-based inference algorithm designed to infer a node maliciousness in knowledge graphs, taking into consideration the edge types, weights, and directions for better propagation of maliciousness. They evaluate their proposal on a knowledge graph constructed from Proxy and DNS logs enriched with related OSINT. They show that MalRank can outperform Belief Propagation.

Our approach is very similar to Najafi's MalRank [186]. However, it differs in the underlying problem definition and data sources used. While Najafi et al. construct their graph from Proxy and DNS data to detect malicious Domains/IPs, we focus on detecting malicious files (malware) through the use of host-level system events and discuss those global features observed in EDR or similar tools' logs.

The main difference between our work on these works is the underlying problem and data. While these works mostly construct their graph from Proxy and DNS data to detect malicious Domains/IPs, we focus on process activity logs and discuss those global features observed in Windows Event Logs, Sysmon, or EDR logs to detect malicious files (malware).

This section provides an overview of the most relevant and influential work in the context of graph-based inference for cybersecurity.

Chau et al. [40] introduce Polonium as one of the first and arguably the most successful works that tackle the problem of malware detection using large-scale graph inference with the intuition that good applications are typically used by many users, whereas unknown (i.e., potentially malicious) applications tend to only appear on few computers. The authors achieve this by running an adopted version of belief propagation on an undirected, unweighted bipartite machine-file graph. In similar research, Tamersoy et al. [257] propose Aesop, which tackles the same problem using locality sensitive hashing to measure the similarity between files to eventually construct a file-file bipartite graph and running BP to infer files' goodness based on its neighbors.

Manadhata et al. [167] address the problem of detecting malicious domains by using enterprise HTTP proxy logs to construct a host-domain bipartite graph capturing workstations' connection to external domains, then running BP to discover malicious domains based on a set of seed malicious nodes. The intuition in this research is that infected hosts are more likely to visit various malicious domains, whereas user behavior on benign hosts should result in benign domain access.

Khalil et al. [130] address the same problem using passive DNS data focusing on a domain-IP bipartite graph with the intuition that a domain/IP is malicious if it has a strong association with a previously known malicious domain/IP. While the authors evaluate BP as part of their evaluation, their main proposal takes a different approach. In this regard, the authors formulate the problem as a similarity measure between a pair of domains based on the number of IPs shared to derive a domain-domain similarity graph and use a path-based algorithm to infer a maliciousness score for each domain according to their topological connection to known malicious domains. In a later research, Khalil et al. [131] discuss the limitations of their previous work [130], which is the computational complexity leading them to adopt belief propagation again on an adjusted graph while emphasizing ASN.

Zou et al. [293] takes a similar approach focusing on DNS logs. In this regard, the authors focus on three main relationships extractable from DNS logs: 1) connection request from an enterprise's workstation to a domain, 2) resolves to a relationship (DNS record type A), which indicates a domain resolving to an IPv4 address, and 3) CNAME DNS RRs which indicates a domain being an alias for another domain.

Najafi et al. [187] also tackle the problem of malicious domain/IP detection using BP on a property graph focusing on domain-to-IP resolution (DNS record type A), domain-to-domain referral (proxy log referer header) and sub-domain relationship.

There are also some other works that take a different approach using advancement in the field of neural networks. In this regard, Peng C et al. [204] propose MalShoot, a graph embedding technique to detect malicious domains using passive DNS. The authors utilize Line algorithm [258] on a domain-IP bipartite graph to calculate feature vectors to eventually train a RandomForest classifier to detect malicious domains. Wenxuan et al. [101] address the same problem but emphasize more on the construction of a domain-domain graph. The authors utilize a modified version of DeepWalk as a Graph embedding algorithm to extract local structure features, combined with other domain and PDNS structural, linguistic, and statistical features to train a RandomForest classifier to detect malicious domains.

Yuta et al. [127] take a different angle by proposing the same approach but to estimate the maliciousness of IoCs with the hypothesis that traditional CTI cannot be trusted and one should evaluate maliciousness by combining IoCs' local features with their global features (associations). For this, the authors propose gathering network-based and OSINT-based information related to IoCs and utilizing a graph convolutional network (GCN)-based method attempting to estimate the maliciousness of IoCs more accurately.

Other works include, Xiaoqing et al. [251] propose HinDom, a malicious domain detection system that treats DNS data as a Heterogeneous Information Network (HIN), allowing meta path-based transductive classification to infer maliciousness.

Huang et al. [110] investigating the connection between domain, IP, and URL. Oprea et al. [193] addressing the early-stage APT detection using BP on host-domain graph extracted from proxy logs. Rahbarinia et al. [214] propose Segugio to detect new malware-control domains based on DNS traffic analysis with a very similar intuition to Manadhata et al. [167]. Mishsky et al. [176] explore the same issues from a slightly different angle. Simeonovski et al. [239] approach the problem using taint-style techniques for the propagation of labels in a property graph built from nodes consisting of domains, organizations, and ASNs. Peng et al. [205] build a domain-domain graph using DNS CNAME RRs with the intuition that domains connected by DNS CNAME RRs share intrinsic relations and are likely to be in a homophilic state.

While the majority of the mentioned works focus on single edge type isolated (i.e., a bipartite graph), we construct a comprehensive knowledge graph that incorporates various types of nodes and edges. To the best of our knowledge, this is the first work exploring knowledge graphs at this scale within the security domain. Furthermore, in contrast to other works, while we evaluate BP, we introduce a much more effective and efficient algorithm that allows us to better infer maliciousness in knowledge graphs. Lastly, while the majority of the other works (e.g., graph embedding-based approaches) work well in theory or a test environment, they fail in a real-world application, particularly in an enterprise environment due to the vast amount of data which requires massive parallel and distributed computing which is challenging for the current state-of-the-art neural networks.

4.11 Chapter Summary

In this chapter, we first introduced the intuition behind global features for threat detection. Next, we presented the CyberHIN, a SIEM-based Heterogeneous Information Network (knowledge graph), which is constructed from entities and relationships observed within data captured by an enterprise's SIEM. More specifically, endpoint EDR logs and network Proxy and DNS logs. We also covered the most relevant OSINT and CTI that can be collected at scale.

We formulated threat detection as a large-scale graph inference problem based on two fundamental principles: *guilt-by-association* and *exempt-by-reputation.* This led us to the introduction of our proposed algorithm named MalRank, a scalable graph-based inference algorithm designed to infer a node's maliciousness score based on its association with other nodes. We also discussed MalRank's unique characteristics that set it apart from other graph-based inference algorithms.

Next, we proposed a framework/system named MalLink designed to automate the process of tailoring the generation of CyberHIN and executing MalRank in a real-world setting. MalLink and MalRank are built on top of Apache Spark to satisfy the scalability requirements, making them suitable to run on SIEMA described in Chapter3.

In the evaluation phase, we configured MalLink for two distinct use cases: malware detection and the identification of malicious domains and IPs. The results of the study demonstrated the superiority of MalRank over other graph-based inference algorithms, mainly Belief Propagation. Additionally, MalLink demonstrated effective detection capabilities, confirming the initial guilt-by-association hypothesis. In essence, the proposed approach was successful in identifying previously unknown threats by inferring maliciousness from the known malicious entities (i.e., CTI, including malware hashes or malicious domains) that were utilized as seed nodes.

In addition, we have discussed certain intrinsic features of this approach, particularly its ability to correct erroneous threat intelligence data and its potential to evaluate and measure the accuracy of CTI.

In summary, the successful implementation of MalLink serves as a significant contribution to the cybersecurity community. The results of this study provide valuable insights into the effectiveness of graph-based inference for identifying malicious entities, highlighting the applicability of guilt-by-association for cybersecurity applications. Despite some of its limitations and shortcoming, MalLink has demonstrated significant capabilities to detect previously unknown low-hanging fruits and has proven to be efficient and feasible for large-scale deployment in a real-world setting. The integration of MalLink with other detection approaches, such as PROVDETECTOR [270], has the potential to enhance our detection capabilities and provide a better fighting chance against the continuously evolving and advancing malware landscape.

5 Human-assisted Outlier Detection

The constantly evolving and sophisticated nature of cyber threats, such as the emergence of fileless malware [148], APT attacks [213], supply chain attacks [75] is putting the efficiency of heuristic-based detection techniques in question [171]. According to CrowdStrike, adversaries' utilization of traditional malware decreased significantly from 61% in 2018 to 29% in 2022 [48]. As the number of sophisticated threats continues to rise, the shortcomings of heuristic-based techniques, which are primarily developed to identify known threats, are becoming more apparent.

As a result, it has become imperative to explore more advanced detection techniques that utilize state-of-the-art data-driven analytics, such as statistical analysis, machine learning [140, 270], and graph mining [40, 98, 99, 186]. However, the lack of high-quality labeled data in this domain [7] renders most supervised data-driven approaches irrelevant, and unsupervised methods such as Anomaly/Outlier Detection (OD) [132, 280] become the most viable option.

Outlier detection has been extensively studied across various domains, such as NLP, bioinformatics, stock market analysis, financial fraud detection [103, 201], health diagnosis [89], industrial defect detection [36], e-commerce [215], social media [287], and others [269].

One of the domains that have also been associated highly with outlier detection is cybersecurity [7, 19, 132], largely due to the widely accepted hypothesis that *"deviation from normal behavior (e.g., of a user or an entity) indicates a potential threat"*. In theory, outlier detection has the potential to identify a wide range of previously unknown threats. This is because anomalous actions may be indicative of previously unknown threats.

However, despite extensive research on anomaly/outlier detection in cybersecurity, particularly in the context of HIDS[123] or NIDS [69], there is still a lack of systematic understanding and demonstration of real-world value [237]. This is largely due to the continuously evolving nature of the cybersecurity domain, which results in unreliable and context-specific features. Additionally, the existence of adversaries that are constantly adapting which further contributes to this challenge [183].

Outlier detection is not transferable, i.e., one cannot develop a universally applicable outlier detection method. Instead, individual outlier detection methods must be tailored to specific applications [10, 39].

In this section, we focus on the challenges of outlier detection in cybersecurity and propose a simple yet effective outlier detection framework. Our proposed approach builds on how an analyst would reason about possible maliciousness by asking simple questions from different angles, i.e., evaluating univariate features within a specific context.

We argue that SOC analysts are indispensable. They rely on their tacit knowledge and experience to develop analytical questions and reasoning that help them investigate the issue at hand [18]. Therefore, an outlier/anomaly detection approach should promote human-machine collaboration acknowledging the importance of the analyst's expertise rather than attempting to automate blindly. In line with this principle, we propose the Human-assisted Ensemble Outlier Detection (HEOD) framework. HEOD leverages the analytical capability of SOC analysts to design outlier detection systems that can be easily interpreted and utilized by the analysts themselves. By emphasizing the importance of analysts' insight and involvement.

Chapter Contribution

The main contributions of this chapter are summarized below:

- **HEOD**: Propose a simple yet effective statistics-based outlier detection framework that emphasizes scalability, simplicity, and interoperability.
- **Discussion and Evaluation**: Evaluate the proposed framework not only on public data sets but also in a real-world enterprise environment. Furthermore, discussing the challenges and lessons learned over a decade of research and industrial application of outlier detection in cybersecurity.

Chapter Structure

The chapter begins by providing the requisite background knowledge for outlier detection (Section 5.1). Section 5.2 discusses the challenges in the adoption of OD in cybersecurity, paving the way for Section 5.3 wherein we unveil HEOD, our proposed framework for outlier detection. Section 5.4 outlines the implementation details and experimental setup used for evaluating HEOD. Following this, in Section 5.5, we present the reproducible results obtained from evaluating HEOD against other well-known OD algorithms. A case study of HEOD's application in a real-world setting for detecting LOLBins is presented in Section 5.6. In Section 5.7, we discuss the lessons learned from operating OD in real-world settings, attempting to separate myth from reality. The limitations and future work directions of HEOD are presented in Section 5.8. This leads to a comprehensive review of related work in outlier detection and their applicability in the cybersecurity domain in Section 5.9. Finally, the chapter concludes with a summary in Section 5.10.

5.1 Background: Outlier Detection

While defining outliers can itself be a complex task [20], we consider outliers as data points that are significantly dissimilar to other data points and do not imitate the expected typical behavior of the other points [104]. Thus, we define outlier detection as the process of finding data that is significantly different from or inconsistent with the rest of the data within a given dataset.

Over the years, the process of outlier identification has carried many names in machine learning and data mining, such as outlier mining, novelty detection, outlier modeling, anomaly detection, etc. [269]. Another similar concept to outlier detection is clustering. Clustering finds the majority of patterns in a data set and organizes the data accordingly. While outlier detection and clustering analysis serve different purposes, one can formulate outlier detection as a clustering problem.

5.1.1 Types of Outliers

Outliers can be categorized under global outliers, contextual Outliers, and collective outliers [96].

Global outliers— Data points that deviate significantly from the rest of the data. These outliers are the simple type, and most outlier detection methods aim at finding them.

Contextual outliers— Data objects that deviate significantly with respect to a specific context. Therefore, in contextual outlier detection, the context has to be specified as part of the problem definition. Contextual outliers are often referred to as local outliers. Global outlier detection can be regarded as a special case of contextual outlier detection where the set of contextual attributes is empty.

Collective outliers— A subset of data that, as a whole, deviates significantly from the entire data set. Importantly, the individual data may not be outliers. However, their collective occurrence is considered an outlier.

5.1.2 Outlier Detection Types

At a high-level, outlier detection methods can be divided into supervised, semi-supervised, and unsupervised techniques.

Supervised Outlier Detection— Supervised methods model outlier detection as a classification problem where a domain expert is expected to label a sample of the underlying data to be used to train a classifier that can recognize the outliers. In other variations, the expert can label the normal data points and have the classifier trained to recognize normality; hence the inverse being the anomaly. However, in real-world applications, supervised techniques may face limitations due to the nature of imbalanced data in real-world problems, where the number of outliers may be too small to train an effective classifier. Although there are techniques to overcome this issue, such as oversampling or undersampling, the applicability of supervised methods in outlier detection is still limited. Moreover, similar to signature-based approaches, supervised techniques are restricted to previously seen anomalies [96].

Unsupervised Outlier Detection— Unsupervised techniques do not require labeled data. Instead, these techniques assume that normal objects are clustered together in feature space, while outliers are expected to occur far away from other observations. The main advantage of unsupervised techniques is that they are not limited by the number of known anomaly samples, allowing the model to recognize new anomalies. However, developing unsupervised techniques on a large scale, especially in real-world scenarios, can be challenging due to their tendency to generate substantial false positives, despite their intuitive concept.

Semi-supervised Outlier Detection— Semi-supervised methods combine elements of both supervised and unsupervised approaches. In this technique, a small set of labeled data is used to train a model that can recognize outliers. Then the model is applied to the remaining unlabeled data to identify any additional outliers. Semi-supervised outlier detection can be more effective than unsupervised techniques since it can leverage some prior knowledge about the data while still being flexible enough to identify new types of outliers.

5.1.3 Outlier Detection Techniques

One can categorize the outlier detection techniques as statistical methods, proximity-based methods, and clustering-based methods [96, 104].

Statistical Methods

Statistical methods make assumptions of data normality. They assume that normal data objects are generated by a statistical (stochastic) model and that data not following the model are outliers. These methods focus on outlier detection using statistical properties of the underlying distribution, such as median, mean, variance, and statistical tests (e.g., box-plot, trimmed mean, extreme studentized deviate, and the dixon-type test [268]).

Statistical-based methods are usually classified into two main groups - the parametric and non-parametric methods. Parametric statistical models have assumptions about the underlying distribution in given data (e.g., Gaussian or Poisson distribution), and it estimates the parameters of the distribution model. Two well-known methods adopted for outlier detection are the Gaussian Mixture model [279] and the Regression model [227]. The effectiveness of parametric statistical methods highly depends on whether the assumptions made for the underlying distribution hold true for the given data. On the other hand, non-parametric statistical methods make no assumptions about the underlying distribution of the data [72]. Kernel Density Estimation (KDE) is a common non-parametric approach for detecting outliers by comparing each point's local density to that of the neighbor's local density[151, 200]. Histogram-Based Outlier (HBOS) is another non-parametric method that uses static and dynamic bin width histograms to model univariate feature densities and score data instances[91].

Proximity-based Methods

Proximity-based methods assume that an object is an outlier if the nearest neighbors of the object are far away in feature space, i.e., the proximity of the outlier object to its neighbors significantly deviates from the proximity of most of the other objects to their neighbors in the same data set.

The effectiveness of proximity-based outlier detection techniques largely depends on the choice of distance measure used to calculate the proximity or distance between data points. There are various distance measures available, such as Euclidean, Manhattan, Cosine, Hamming, and Mahalanobis distances [55, 59, 236].

There are two major types of proximity-based outlier detection: distance-based and density-based. Density-based outlier detection methods identify outliers as data points that are located in areas of low data density, where data density is defined as the number of data points in a given neighborhood. In contrast, distance-based outlier detection methods identify outliers based on the distance or proximity of a data point to its nearest neighbors. Local Outlier Factor (LOF) is an example of a loosely related density-based outlier detection method [31], and k-nearest neighbor (KNN) is a type of distance-based [54].

Clustering-based Methods

The underlying assumption of clustering-based outlier detection methods is that normal data objects belong to dense and large clusters, whereas outliers either belong to small or sparse clusters or do not belong to any clusters at all.

The performance of clustering-based techniques is highly dependent on the effectiveness of the clustering algorithm and its parameters. Nevertheless, The unsupervised nature of clustering-based methods makes them appealing for a wide range of applications.

Clustering-based methods share many characteristics with proximity-based methods, as they both rely on measures of distance or density. Some of the most widely used clustering-based algorithms include K-Means [164], and DBSCAN [73].

Ensemble-based Methods

Ensemble-based methods combine multiple outlier detection methods into a single framework in order to improve the overall performance of outlier detection. Example of ensemble-based includes Isolation forest [157] and Extreme Gradient Boosting Outlier Detection (XGBOD) [290].

Other Methods

Outlier detection has gained significant attention in recent years, resulting in the development of new techniques and approaches. Notably, there has been a surge of interest in active learning[57, 207], deep learning [38], and graph-based outlier detection [9].

5.2 Challenges and Requirements

Before delving into the specifics of our proposed approach, it is crucial to first address the primary challenges, hence requirements that must be considered for a successful anomaly/outlier detection in cybersecurity.

Contextualization

In the rapidly changing field of cybersecurity, where the distinction between normal and malicious behavior can be subtle, contextualization is essential to enhance our ability to discern and make more informed decisions. [10, 18]

Consider an employee that is found to have logged in to several computers. This act might initially appear unusual, as it could indicate a malicious attempt to move laterally within a network. However, if further investigation reveals that the employee in question is an administrator, this activity would be deemed acceptable and no longer anomalous, as administrators frequently access multiple assets. This illustrates the necessity for contextualization, for instance, grouping individuals based on their roles prior to conducting any analysis.

Interoperability and Explainability

The term "interpretability" or "explainability" is defined as the ability to provide understandable explanations for a human and an end-user.

Sejr et al. [233] discuss how outliers are context-dependent and, therefore, can only be detected via domain knowledge, algorithm insight, and interaction with end-users. Emphasizing how interpretation, explanation, and user involvement have the potential to provide the missing link to bring complex outlier algorithms from research into real-life applications.

This is particularly relevant in cybersecurity, where outliers do not always indicate malicious activity, leading to alert fatigue in real-world settings [99].

Analysts' involvement in SOCs is indispensable, and the lack of interpretability and explainability poses difficulties for human analysts to effectively decide whether the anomaly is of interest (i.e., malicious and true positive) or not (i.e., benign and false positive). This is because, without explainability, the analysts would spend most of their time investigating why an alert is handed to them in the first place.

However, many of today's complex and advanced outlier detection algorithms (e.g., deep learning-based, autoencoder-based) lack interpretability [10, 226, 277].

As a result, it is highly desirable to develop outlier detection algorithms that provide interpretable explanations in addition to outlier scores to facilitate more efficient analysis and decision-making by human analysts.

It is also essential to distinguish between explainable and interpretable. Even in the face of explainable outlier detection, an analyst would still have difficulty comprehending the justification in a multi-dimensional setting. In contrast, it is much easier to comprehend a simple univariant feature given a context.

Scalability

As the amount of data that companies collect continues to grow, the concept of big data, which encompasses data generated at a high volume, diverse variety, and rapid velocity, becomes increasingly relevant. A company of moderate size can easily collect and process tens of terabytes of data daily in near real-time. This data can come from various sources, including endpoints (e.g., EDR, Windows

Event Logs), network devices (e.g., proxy and DNS servers), and servers, among others [183]. This highlights the necessity for outlier detection algorithms that are efficient and scalable, utilizing distributed systems and parallel computation for scaling outs [145].

Incremental and Adaptive Learning

In this rapidly evolving cybersecurity landscape, new learnings can significantly influence the interpretation of existing knowledge. As such, it is crucial for outlier detection algorithms to have the capability to adapt continuously in response to changing data and its distribution.

Consider file hashes as a feature in an anomaly detector. Within a typical enterprise setting, the number of unique hashes observed daily can be substantial, on the order of thousands. An anomaly detection algorithm may become ineffective without the ability to incorporate new data efficiently.

Many traditional outlier detection methods necessitate the complete retraining of models when new data is introduced. This can be computationally intensive and may not be practical when dealing with large, dynamic datasets. Therefore, outlier detection algorithms need to have the capability of quickly and efficiently incorporating new data into the model without the need for complete retraining. This ensures the continued effectiveness and robustness of the outlier detection system.

Feature engineering represents a critical step in creating an effective outlier detection algorithm. This is especially important in the continuously evolving realm of cybersecurity. In the face of evolving threats, some knowledge becomes obsolete while new knowledge arises. Therefore, it is also imperative that the outlier detection algorithm possess the capability to not only support incremental learning but also facilitate adaptation to changing conditions through the continual update and modification of features to ensure its ongoing effectiveness.

Categorical Data

Categorical features often need to be encoded into numeric form, as the majority of outlier detection algorithms take attributes as either discrete or continuous. However, this can be a difficult task if the categorical variables have a large number of categories or if there is a high or low degree of cardinality, making identifying outliers based on their frequency or distribution difficult. Furthermore, the inherent property of categorical data, which lacks ordinality and a numerical scale, presents a challenge for applying traditional outlier detection techniques that utilize metrics for measuring the dissimilarity between observations (e.g., density or distance measures) [254]. Therefore, the ability to treat categories simply as categories is favored.

Non-Parametric

The selection of appropriate hyperparameters is a significant challenge in many outlier detection algorithms, as it can significantly impact the quality of the results [155]. For example, in k-means clustering, the number of clusters, k, is a hyperparameter that needs to be specified in advance. The optimal value of k depends on the structure of the data and the problem at hand. A common approach is to use trial and error to find the best value of k, which can be time-consuming and computationally expensive. Additionally, because parametric models are based on assumptions about the distribution of the data, their results can be unreliable for practical situations and applications when prior knowledge about the underlying distribution is limited. This is particularly relevant for outlier detection in the field of cybersecurity, where the distribution of data can be highly skewed and varied across different contexts.

This makes non-parametric algorithms that do not assume a particular distribution particularly appealing.

Event Count vs. Prevalence

Outliers can be understood as data points with a low frequency of occurrence and typically indicate the number of times an event with a particular attribute has been observed. However, in some instances, the focus may not be on the frequency of observation of the primary attribute but rather on the number of unique occurrences of a secondary attribute that have been observed in relation to the primary attribute. As an example, consider the scenario of executable file hashes. In this case, the focus may not be on the frequency of the file hash occurrence but rather on the number of unique assets that have been observed running the executable. In such a scenario, the emphasis would be on identifying the count of unique asset IDs associated with the file hash rather than counting the events associated with that file hash.

5.3 HEOD: Human-assisted Ensemble Outlier Detection

In this section, we present our proposed approach, taking into account the challenges and requirements previously discussed. We begin with a motivating example that serves as a basis for introducing the framework. We then provide an overview of the approach and conclude with a detailed explanation of the technique. This includes an explanation of the underlying algorithm and a justification for each design decision.

Consider network traffic, particularly proxy logs. There is valuable information in such logs, including the protocol, URL, HTTP method, duration, response code, request/response length, user-agent, client IP, and destination IP [186]. To decide on whether a request is potentially malicious based on its corresponding event log, an analyst may consider various factors, such as the rarity of the destination, the rarity of the destination given the source, the response code in relation to the domain, and the request length in relation to the domain given the HTTP-method.

Following this mindset, we propose a framework that mirrors the thought process of an analyst who evaluates a log entry from multiple perspectives, i.e., different features within their corresponding contexts. Hence, an ensemble of simple univariate models, each specifically designed to examine a single, carefully selected feature within a given context tree to determine the level of abnormality for the observation.

The core idea is that by combining simple well-thought submodels, each of which examines different sub-spaces of the data from different perspectives, we can achieve a thorough and holistic evaluation of log entries that are explainable and intuitive. This is possible as every submodel's unique contribution to the overall score is measurable and explainable.

At a high level, a submodel consists of several histograms of frequency observations representing a single feature within different contexts, e.g., request length (feature) per destination and HTTP method (context). Now, given a histogram, we can define outliers as data points that occur in low-density parts and are distant from the main densities. The lower the observation density and the further from the closest density, the more significant the outlier. Finally, the submodels' independent output is combined to derive a single outlier score representing the overall outlierness of an entry, hence ensembling.

In Summary, the main features of the proposed HEOD are:

- Simplify detection to a single well-thought feature within a context.

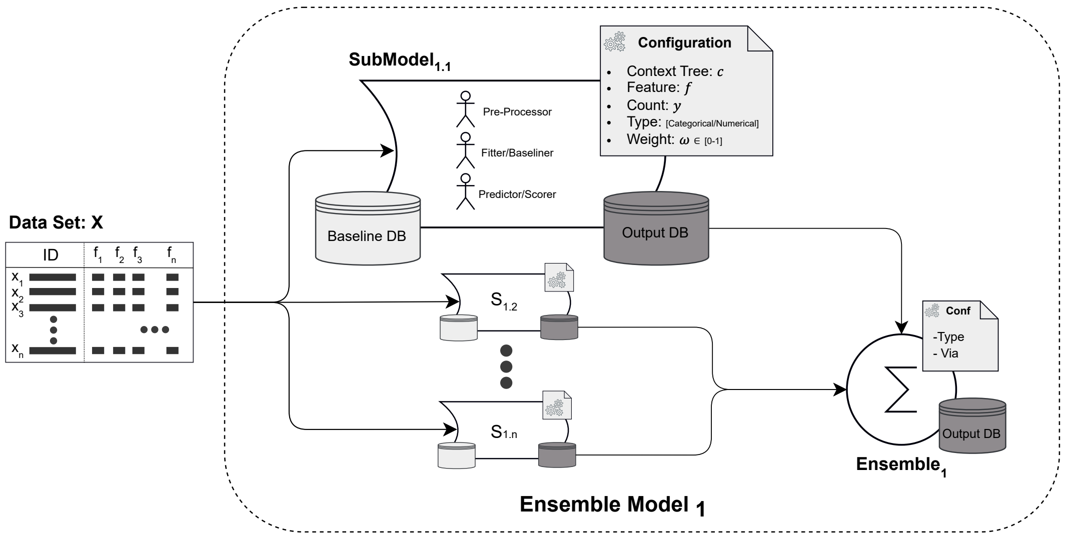


Figure 5.1: The proposed HEOD framework.

- Incorporate expert/domain knowledge, i.e., feature engineering and defining how every feature should be contextualized.
- Density and distance-based scoring functions that are unique to cybersecurity with an emphasis on skewness.
- Every detection can be easily explained and interpreted due to the simplicity of the submodels and the ensembling process.

More formally, suppose we have n data points $\mathbf{X} = X_1, X_2, ..., X_n$, each with f dimensions or features. The objective is to compute the aggregated or ensembled score $\bar{O}_i$ for each data point X_i passed through all submodels in a set of fine-grained univariate models denoted by $\mathbf{M} = M_1, M_2, ..., M_n$.

Figure 5.1 shows the high-level architecture of the proposed approach. In the following, we expand on the details of each component. Table 5.1 presents a summary of the symbols and notations that will be used throughout the rest of this section.

5.3.1 Submodels

Submodels are univariate statistical models that are designed to baseline and score observation within a defined context. Hence a submodel $M_i := \mathcal{M}^{\omega}_{f|c,y}$ will have the following configuration.

Table 5.1: The list of symbols and notations used in this section.

Symbol	Description
$\mathbf{X}$	the set of data, e.g., the whole table
X_i	a data point (e.g., a row in a table)
$\mathcal{X}_i^{f\|c}$	the value under feature f for the data point X_i with context tree c, corresponding to row i and column f, while also having access to other columns denoted by c in row i.
$\mathbf{M}$	the set of all submodels.
$M_i := \mathcal{M}^{\omega}_{f\|c,y} \in \mathbf{M}$	a submodel that is configured with feature f, of type f_t, context tree c, count of y, and submodel weight of ω
$\bar{O}_i$	the final ensembled score for data point X_i that has passed through all submodels in $\mathbf{M}$
$hist(M_i)$	the baseline of model M_i
$hist(c \mid M_i)$	the portion of baseline table (histogram) for model M_i where the context is c
$histStats(M_i)$	the statistical points derived from the baseline for the submodel M_i
$histStats(c \mid M_i)$	the portion of baseline statistics table for model M_i where the context is c
$F(X_i \mid M_i)$	an estimate of the observation count of data point X_i from the perspective of model M_i, according to its configured feature and context tree
$S_{density}(X_i \mid M_i)$	the density score for data point X_i given by model M_i
$S_{distance}(X_i \mid M_i)$	the distance score for data point X_i given by model M_i
$S(X_i \mid M_i)$	the outlier score (a composite of the distance and density scores) for data point X_i given by model M_i
$S_c(X_i \mid M_i)$	the confidence in the outlier score given to data point X_i by model M_i
$O(X_i) = \{(S(X_i \mid m), S_c(X_i \mid m)), \forall m \in M\}$	the set of outlier scores and outlier confidence scores for X_i given by all models in the set of $\mathbf{M}$
$\bar{O}_i = \bar{E}_i = E(O(X_i))$	the ensembled score for data point X_i after passing through all models $\mathbf{M}$

- *feature* (f): the dimension/feature the submodel is designed to evaluate.

- *feature type* (f_t): the type of the feature, i.e., numerical or categorical. That is important as we use a different hypothesis to score a categorical feature, even though the categorical feature might be represented by numeric values (e.g., label encoding or one hot encoding).

- *context tree* (c): the context to which the feature is bound. This context can include a sequence of features. For instance, bounding the feature `bytes_sent` to context `dest`→`http_method`, indicating that the bytes sent for different destinations and different HTTP methods (e.g., POST vs. GET) should not be compared together.

- *count* (y): the secondary feature to define the count rule. This feature would allow us to treat outlier detection as a prevalence problem. For instance, instead of counting how many times (event count) we see a particular file hash, we count unique endpoint observations (i.e., on how many assets we observed executing that file). This functionality is particularly significant since there may be instances in which the focus is not on the observation of a given feature value but instead on the unique observation of another feature that corresponds to the original feature f.

- *weight* ($\omega \in [0, 1]$): the weight of the submodel defined by the expert.

Each submodel is expected to possess three primary functionalities: *data preparation*, *baselining/fitting*, and *scoring/predicting*. The data preparation function is relatively straightforward, entailing the ability to pre-process an input source (e.g., table, dataframe, CSV), select relevant contextual information and features of interest, and subsequently, validate them. On the other hand, the fitting and scoring function is comparatively more complex. In the following discussion, we expand further into these functions.

Submodel Fitting

Fitting is simply counting the observation of a feature within a context. This count can take the form of a simple count of the occurrences of a particular value or a unique count of a secondary feature associated with the primary feature. The fitting can be conceptualized as a method of grouping and aggregating data as follows:

```
data.groupBy(c_1,c_2,...,c_n, f)
    (1) .agg(count())
    (2) .agg(countDistinct(y))
```

Fitting/Baselining leads to the creation of a table conceptualized as a collection of multiple histograms separated by the context tree, histograms that denote the feature values on the x-axis, and the count of observations on the y-axis. Throughout this paper, the terms frequency, observation count, and height are utilized interchangeably to denote the number of observations on the y-axis.

Consider events associated with the execution of executable files, identified by `sha256`, in an endpoint (`computerName`), which we have enriched with departmental information (`peerGroup`). Now consider two almost equivalent models configured with the following:

$$(1)\ \mathcal{M}^{1.0}_{\texttt{sha256}|\texttt{peerGroup},\texttt{-}} \qquad (2)\ \mathcal{M}^{1.0}_{\texttt{sha256}|\texttt{peerGroup},\texttt{computerName}}$$

Two models designed to evaluate feature f=`sha256` within context tree c =`peerGroup`, but one with `y=None(-)`, i.e., simply counting event observations, and the other with `y=computerName`, i.e., counting unique observation of endpoints associated with the sha256. In this regard, one can conceptualize the baselining process of these two models as follows.

```
(1) data.groupBy(peerGroup,sha256).agg(count())
(2) data.groupBy(peerGroup,sha256).agg(countDistinct(computerName)
```

In this example, the first model determines the frequency of event observations related to a given file within a specific peer group (e.g., the HR department). Addressing the questions: *How many times have we seen this file executed in each peer group?* Whereas the second model focuses on determining the count of unique machines on which a particular file has been executed within a given peer group. Answering the questions; *What is the prevalence of this file within each peer group?* This latter metric is often deemed more informative as it provides insight into the distribution of a file across different machines rather than simply counting the number of events associated with that file.

Categorical Feature Baselining: Baselining a categorical feature is straightforward. It simply involves counting the number of observations of the different values within a defined context, such as the frequency of different HTTP methods for a particular destination.

Numerical Feature Baselining: In the case of numerical features, the process becomes more complex. While it is still possible to count all observations, the continuous nature of numerical data and the potential for different resolution scales must be considered. In the following sub-section on scoring, we elaborate on the technique used to estimate frequency (count of observations) by utilizing a kernel density estimation-like function. While our proposed technique does not require data binning or clustering, one can deploy various optimization strategies to reduce the data and tier it into different layers (i.e., bronze, silver, gold). For example, when baseline the feature `bytes_sent`, rather than counting two data points $x_1 = 1.0001$ and $x_2 = 1.0003$ as two distinct values with a count of one each, they can be combined and represented by a single value (e.g., $x_1 = 1.0002$) with a count of two. There are many strategies for data reduction, such as discretization/binning [27, 37], clustering[118], and sampling[235].

To gain a deeper understanding of the baselining process, let's consider the example mentioned at the beginning of this section. Suppose that one of the models is configured with the following configuration:

$$\mathcal{M}^{1.0}_{f=\texttt{request\_length}|c=\texttt{destination-http\_method},y=\texttt{-}}$$

A model that is constructed to analyze the occurrences of *request lengths*, differentiated by *destination* and *HTTP method.* The hypothesis underlying this model is that the destination and HTTP method (context tree) plays a role in determining the distribution of request lengths. For instance, the request lengths for GET requests should not be directly compared with those of POST requests.

An example of the data beneath the baseline table for the mentioned model is presented in Table 5.2. The first row indicates that there have been five instances of a POST event to `dest1.com` with a request length of 10.

Table 5.2: Example for a submodel baseline table.

$c-1$: `dest`	$c-2$: `http_method`	f: `request_length`	y: `EventCount`
dest1.com	POST	10	8
dest1.com	POST	12	1
dest1.com	POST	13	3
dest1.com	POST	17	3
dest1.com	POST	25	1
dest1.com	GET	1	2
dest1.com	GET	3	3
dest1.com	GET	25	3
dest2.com	POST	120	2

It is important to note that this baseline table can be conceptually understood as a collection of histograms. Specifically, a set of histograms defined by the context trees. For instance, in the above example, given the context tree of `dest` $\rightarrow$ `http_method`, one can conceptualize three distinct histograms, one for each branch in the context tree.

Through the process of submodel baselining, a series of tables can be generated to facilitate scoring and prediction.

- Frequency table, $hist(M_i)$: a table that records the frequency of each observation value within a specific context tree.
- Frequency stats table, $histStats(M_i)$: a table used to maintain critical statistical points of the observation table, such as the median, average value, and hyperparameters for any relevant techniques such as the bin-width or kernel bandwidth.

Submodel Scoring

At a high level, submodel scoring is based on two fundamental principles, the lower the frequency observation of the value, the higher the outlier score, referred to as the *density score*, and the further the distance of the value from the closest high-density region, the more significant the outlier, referred to as the *distance score*.

The scoring process aims to seek answers to the following questions:

1. What is the frequency of occurrence for this value or similar values?
2. How does the observation count for this value compare to observation counts?

3. How does the value of this observation compare to other observed values?

Let's consider the model described previously (Table 5.2) to better grasp the difference between the density and the distance score. Given an average request length of k bytes for dest1.com and the POST method, while observations of $k+1000$ and $k + 1000000$ would both fall within low-density regions and thus be assigned a high outlier density score, the second observation may be considered a more significant outlier due to its greater deviation. The distance score provides a means of assessing the relative significance of outliers based on their distance from the high-density regions in the data. This highlights the importance of considering both density and distance when assessing the outlierness.

As mentioned before, scoring techniques vary depending on the type of feature being considered. For categorical features, distance is not a relevant factor since they lack the concept of distance. Additionally, determining the frequency count of a value X_i for a categorical feature is a straightforward process of joining the relevant baseline table to find the frequency. However, with numerical features, because they may have a continuous nature, it is necessary to be able to estimate the frequency. Below, we detail the specific procedure for calculating scores based on the feature type.

Categorical Feature Scoring: In order to predict the density score of a data point X_i, the model must first isolate the corresponding baseline, specifically, the histogram that aligns with the context tree values of X_i. As an example, in the case of a model that is evaluating the feature `response_code` and is contextualized by the `http_method`, if the data to be scored has an HTTP method of POST, the model must first isolate the histogram associated with the POST context, i.e., the portion of the baseline table that pertains to the POST context (denoted as $hist(c \mid M_i)$ and $histStats(c \mid M_i)$).

Next, determine the observation count for the given response code. This can be accomplished by simply joining the observed context and feature value to find the observation count.

More formally, $F(X_i \mid M_i) \in \mathbb{R}$ is the frequency observation of data point X_i with respect to model $M_i := \mathcal{M}^{\omega}_{f|c,y}$ which evaluates feature/dimension f within context tree c and counting observations of y.

After the count estimation, it is necessary to transform the unbounded frequencies into a density-based outlier score that is both bounded and interpretable, such as probabilities [86], i.e., $S_{density}(X_i \mid M_i) \in [0, 1]$. This conversion process is commonly referred to as data scaling or normalization in data mining. This step is necessary as the independent scores must ultimately be integrated into a unified ensemble score.

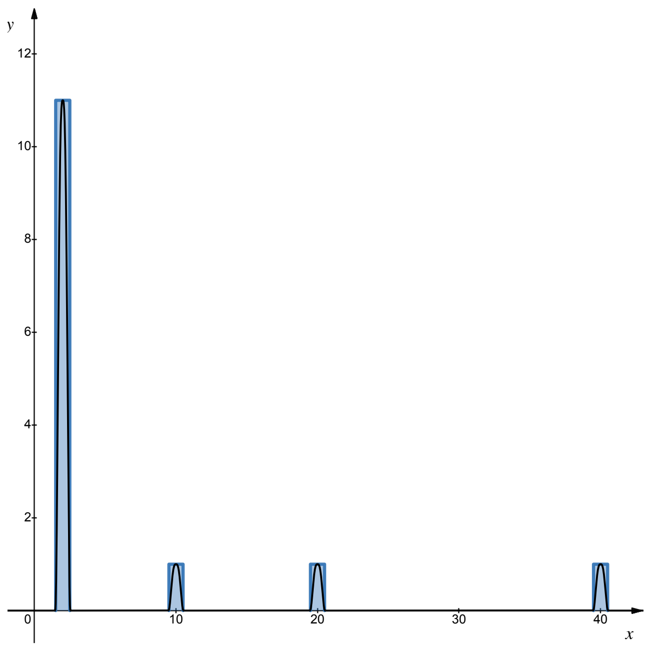


Figure 5.2: The scaling challenge in a highly skewed distribution.

There are several techniques for achieving data scaling or normalization, such as Min-Max or MaxAbs scaling. More generally, this can be viewed as a data transformation problem that involves applying mathematical functions or algorithms to the data to alter its distribution, range, or format. This generalization enables the use of transformers, including Quantile, Power, Logarithmic, Box-Cox, Sigmoid, Square root, and Exponential transformations [8].

In this work, we draw inspiration from the technique of quantile transformation. Our proposed approach is particularly beneficial in cybersecurity, where we often encounter highly skewed distributions, and it is imperative to preserve the relative order of values. For instance, when dealing with a dataset comprising a vast number of small values and a few large values, our approach mitigates the impact of the large common values on the transformed data, allowing for an improved focus on outliers.

Consider a highly skewed distribution (long-tailed) where a few values are observed frequently, such as 1 KB being observed over ten million times (See Figure 5.2). The same histogram also contains values in the tail that are only observed at the scale of thousands. If we were to use a Min-Max scaler, the large number of 1 KB observations would dominate the scaling process, resulting in a score of 0.0 for 1KB and a score above 0.9 for all other observations. This would make it difficult to distinguish between observations that fall between 1 and 100 observation counts, e.g., 20, 40 KBs.

In this work, we propose a dynamic transform function that is interpolated based on a few given thresholds. The transformer (Function 5.3) is a multi-condition function that oscillates between Equation 5.1 and Equation 5.2

$$f_{denA}(x, y_s, y_e, x_s, x_e) = y_s - (y_s - y_b)\frac{(x - x_s)^2}{(x_s - x_e)^2} \tag{5.1}$$

$$f_{denD}(x, y_s, y_e, x_s, x_e) = y_s - (y_s - y_b)\frac{(x - x_s)^3}{(x_s - x_e)^3} \tag{5.2}$$

$$f_{den}(x, t_l, t_c, t_u) = \begin{cases} f_{denA}(x,1.00,0.75,0,t_l-\epsilon), & \text{if} \quad x \leq t_l - \epsilon \\ f_{denD}(x,0.75,0.50,t_l-\epsilon,t_c), & \text{if} \quad t_l-\epsilon < x \leq t_c \\ f_{denA}(x,0.50,0.25,t_c,t_u+\epsilon), & \text{if} \quad t_c < x \leq t_u+\epsilon \\ f_{denD}(x,0.25,0.0,t_u+\epsilon,1+2\cdot(t_u-t_c)), & \text{if} \quad t_u+\epsilon < x \leq 1+2\cdot(t_u-t_c) \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

Where x represents the frequency (i.e., the estimated number of observations) of a random variable, and t_l, t_c, and t_u are thresholds derived from the underlying distribution. These thresholds regulate how the observed frequency is compared to other frequencies. For instance, consider a histogram where numerous observed values (on the x-axis) have a frequency count of 1 (on the y-axis). While an individual observation may appear to be an anomaly (i.e., only once observed), the fact that this frequency occurs frequently should reduce the significance of the outlier.

The epsilon is used to control scenarios where all thresholds are equal. For instance, if many values have only one observation count, in this case, all thresholds will be equal. Such a scenario is prevalent in reality as novel things are consistently emerging, resulting in observation counts of one.

Figure 5.3 (a) shows an example of the proposed density function interpolated from thresholds of $t_l = 2$, $t_c = 4$, and $t_u = 10$. The dotted red and blue lines on the figure represent the underlying functions utilized to oscillate between different thresholds (using Equations 5.1 and 5.2). The figure illustrates that observation counts from 0 to $t_l = 2$ are relevant, and their score should decline from 1 to 0.75 accordingly.

Therefore, the density score for data point X_i given by model $M_i := \mathcal{M}^{\omega}_{f|c,y}$ that evaluates feature value f within context tree c, is calculated as follows:

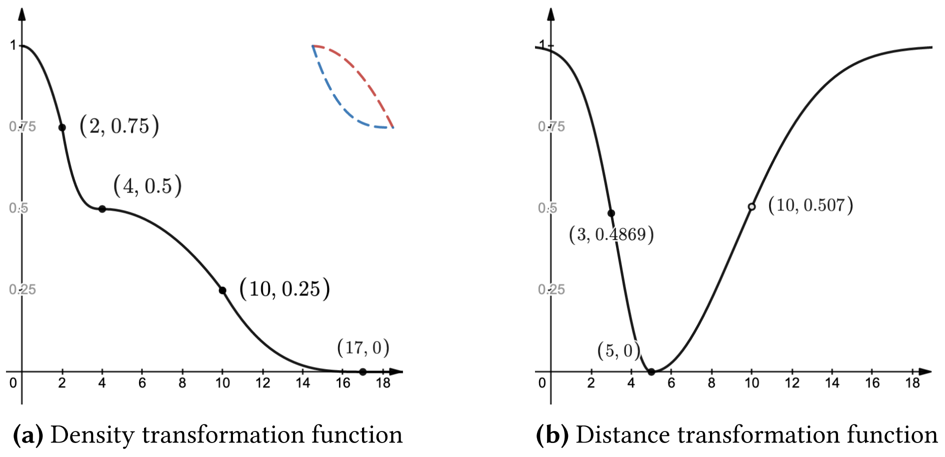


Figure 5.3: Density and distance transformation function with threshold examples.

$$S_{density}(X_i \mid M_i) := S_{density}(\mathcal{X}_i^{f|c} \mid \mathcal{M}_{f|c,y}^{\omega}) = f_{den}(F(\mathcal{X}_i^{f|c}), t_l, t_c, t_u)$$

$$\{t_l, t_c, t_u\} \in histStats(c \mid M_i)$$

Where $F(\mathcal{X}_i^{f|c})$ represents the frequency observation of data point X_i's feature f given context tree c, one can conceptualize a table where the row is i, the column is f, and we also have access to other columns c in that row (context tree).

Note that the thresholds ($\{t_l, t_c, t_u\}$) should be determined based on the underlying distribution of the mode's baseline corresponding to the related context tree, represented as $histStats(c \mid M_i)$. These thresholds should reflect the essential statistical properties of the frequencies in the distribution. For instance, providing insight into the average observed frequency enables the comparison of the observation counts at hand with others in the distribution.

To better understand the underlying concept, let's consider a model that is designed to evaluate the rarity of an executable path (a categorical feature) contextualized by the executable's name. The hypothesis is that executables run from uncommon locations may pose security risks. After one month of baseline, all paths for a given executable end up with a table that can be conceptualized by a histogram per executable, where the x-axis represents the observed normalized paths, and the y-axis represents the number of times we have seen that particular executable within that path. Suppose we observe a new path associated with that executable. The density score can tell us how anomalous this observation is compared to the others. However, to give a more accurate score, the model will also take into consideration the variety of path observations for that executable. More specifically, if we have never seen this path before and we have observed many different paths only once (with an average observation count of 1), then the outlier may not be as significant. However, suppose we have mainly observed one path (resulting in a much higher average of path observation count), and the path in question has never been observed before; the outlier score should be much more significant. This is the role of the thresholds, allowing us to control the decaying better.

For instance, one can derive the thresholds from the underlying statistics as follows:

$$\begin{cases} t_l = GM(min, q1, mean, median) \\ t_c = GM(mean, median) \\ t_u = GM(max, q3, mean, median) \end{cases} \qquad GM = \left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}}$$

Note that the properties are of the count observations, e.g., *min* is the minimum count observation in the table/histogram, *q1* is the first quartile of count observations, and so on. We found through experimentation that this particular setting was the most suitable for our experiment, as it allowed the minimum and maximum values to have an impact. However, one could experiment with different statistical properties, e.g., skewness and kurtosis.

Although we only used three thresholds to interpolate the defined function, one can achieve similar results using linear interpolation with more points. Furthermore, note that the transformation function changes the underlying distribution. However, it does not affect our ability to detect outliers. In fact, it improves by emphasizing outliers and reducing the importance of the majority class.

Hence for the categorical feature, the final outlier score S for data point X_i given by model M_i is:

$$S(X_i \mid M_i) = S_{density}(X_i \mid M_i)$$

Numerical Feature Scoring: Since numerical values can be continuous and not limited to a set of discrete options, it is not possible to perform an exact join on a table, as is the case with categorical features. To address this issue, we must employ a method to estimate the frequency or count of a data point and its feature.

Formally, $F(X_i \mid M_i) \in \mathbb{R}$ denotes the frequency observation of data point X_i relative to model $M_i = \mathcal{M}^{\omega}_{f|c,y}$, which assesses feature or dimension f within context tree c and tallies the observations of y. We refer to the histogram or data points extracted from the baseline table of model M_i for which the context tree is c as $hist(c \mid M_i)$. Let $\mathbf{Z} = z_1, z_2, ..., z_n$ denote the data tuples derived from $hist(c \mid M_i)$, where each tuple z_k contains $z_k^{(i)}$ as the x-axis value and $z_k^{(j)}$ as the y-axis count. The task at hand is to estimate the count observation for the feature value f of data point X_i, i.e., $\mathcal{X}_i^{f|c}$, given $hist(c \mid \mathcal{M}^{\omega}_{f|c,y})$.

To achieve this, we adopt Kernel Density Estimation [198], with a slight alteration to give us the height (count/frequency) rather than density. This technique allows us to estimate regardless of underlying data distribution which is an essential requirement in cybersecurity.

$$KHE(x \mid Z) = \sum_{k=1}^{n} z_k^{(j)} \cdot K(\frac{x - z_k^{(i)}}{h})$$

$$K(u) = e^{-0.5 \cdot u^2}$$

$$F(X_i \mid M_i) := F(\mathcal{X}_i^{f|c} \mid \mathcal{M}^{\omega}_{f|c,y}) = KHE(\mathcal{X}_i^{f|c}, hist(c \mid \mathcal{M}^{\omega}_{f|c,y})) \tag{5.4}$$

Where K is the adjusted Kernel, a non-negative function, $h > 0$ is a smoothing parameter called the bandwidth; intuitively, one wants to choose h as small as the data will allow; however, there is always a trade-off between the estimator's bias and its variance. A substantial amount of academic literature exists that explains the most optimal selection of bandwidth (h) [230]. However, throughout our experiments, we utilized Silverman's rule of thumb [238].

$$h = 0.9 \cdot min(\hat{\sigma}, \frac{IQR}{1.34}) \cdot n^{\frac{-1}{5}}$$

where $\hat{\sigma}$ is the standard deviation derived from $hist(c \mid M_i)$, IQR is the interquartile range $Q3 - Q1$, and n is the total sample size in the histogram

As mentioned before, one could perform various optimization to improve the computational cost, e.g., sub-sample data for the KHE or cluster data to reduce the baselines table.

Numerical Feature Density Score: After frequency estimation, similar to the categorical feature, we use the same formulas to convert the frequencies to a density-based outlier score $S_{density}(X_i \mid M_i) \in [0, 1]$.

To enable this transform function, we require a set of thresholds derived from the statistical properties of the frequencies observed (e.g., what is the average count/hights observed per histogram). However, while in the case of categorical data, one can easily find the count statistics (e.g., mean of counts), in the case of numerical data obtaining the basic statistics of the count can be challenging.

There are several methods to obtain these statistics, including identifying all critical points in the derived KHE function (such as global and local maxima) using techniques like the First Derivative Test [246], sampling data to estimate the count statistics, or binning the values and computing the statistics based on the heights of the bins. While all of these approaches produced comparable thresholds in our experiments, we ultimately opted to utilize binning due to its efficiency, particularly in a distributed map-reduce fashion.

More specifically, we employed modified versions of Freedman–Diaconis [82], and Scott's [231] rules taking into account the skewness of the data to determine the optimal bin size and calculate the statistical measures for the bin heights (e.g., min, median, average bin height). These measures were then inserted into the $histStats(M_i)$ table, which was later used to derive the necessary thresholds for the transformation.

Numerical Feature Distance Score: As mentioned, with numeric features, we introduce a novel aspect - the outlier distance score. This property allows for the detection of significant anomalies that may not be detected through density-based techniques alone.

Figure 5.2 illustrates the challenge posed by relying solely on a density-based score. In such a scenario, values of 10 and 40 will receive similar density scores, as they are both situated in areas of low density (i.e., their estimated observation count will be identical).

In the field of cybersecurity, the detection of extreme values is often of particular interest. For instance, consider the feature of bytes sent. Suppose the value 10MB is assigned the same density score as a value of 10GB. In that case, the 10GB outlier will be of significantly greater interest, thereby highlighting the importance of utilizing a distance score.

To derive the distance score $S_{distancce}(X_i \mid M_i) \in [0, 1]$, we measure how far an observation value is from the closest density (mode). Our implementation of HEOD uses global density (or the largest mode) as the reference point. However, one can further improve this by finding the closest local density.

In this regard, similar to the density score, we interpolate a transformer function that maps a given feature value to a score between 0 and 1 based on a set of thresholds.

$$f_{disA}(x, x_s, x_e) = 1 - e^{\frac{-2(x-x_s)^2}{e(x_s-x_e)^2}} \tag{5.5}$$

$$f_{dis}(x, t_l, t_c, t_u) = \begin{cases} f_{disA}(x, t_c, t_l - \epsilon), & \text{if } x < t_c \\ f_{disA}(x, t_c, t_u + \epsilon), & otherwise \end{cases} \tag{5.6}$$

Figure 5.3 (b) shows an example of the distance transformer function, highlighting that as we move from the densest area (5) to further values, e.g., 16, the distance score gets closer to one.

$$S_{distance}(X_i \mid M_i) := S_{distance}(\mathcal{X}_i^{f|c} \mid \mathcal{M}^{\omega}_{f|c,y}) = f_{dis}(\mathcal{X}_i^{f|c}, t_l, t_c, t_u)$$

$$\{t_l, t_c, t_u\} \in histStats(c \mid M_i) \qquad \begin{cases} t_l = d \cdot (mean + stddev) + (1 - d) \cdot min \\ t_c = mean \\ t_u = d \cdot (mean + stddev) + (1 - d) \cdot max \end{cases}$$

Note that here the stats are from the distribution, e.g., *min* is the minimum value observed in the related table/histogram, *q1* is the first quartile of values, and so on. We set the parameter d to 0.8 during our experiments.

Therefore, when the feature is numerical, the outlier score $S(X_i \mid M_i)$ is calculated as follows:

$$S(X_i \mid M_i) = \omega_{density} \cdot S_{density}(X_i \mid M_i) + \omega_{distance} \cdot S_{distance}(X_i \mid M_i).$$

Throughout our experiment, we set the density score weight ($\omega_{density}$) to 0.8 and the distance score weight ($\omega_{distance}$) to 0.2.

Figure 5.4 illustrates the kernel height estimation function, the density, and the distance score of an example distribution.

Outlier Confidence Score

In this section, outliers were defined as observations with different characteristics to other data points. However, this definition lacks clarity on what constitutes "different". For instance, if our baseline consists of only one item and we encounter a dissimilar observation that is also different, however, it should not be considered a top-priority outlier as there is no well-established baseline. Thus, it is crucial to not only identify outliers but also to quantify the degree of certainty for the detection. Confidence scores prove helpful in practice as they assist in reducing potential false positives and prioritizing which outliers to investigate first.

Confidence is particularly important in our proposed framework, as contextualization may introduce discrepancies between the baselines. Therefore, it becomes crucial to establish a high degree of confidence in the identification of outliers,

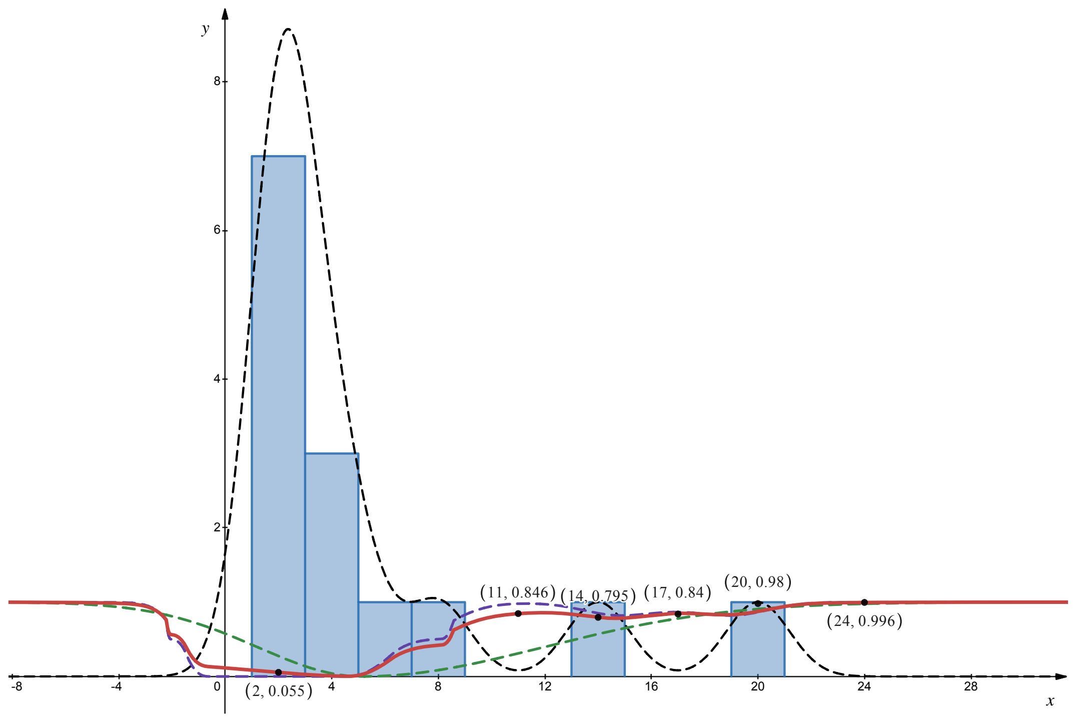


Figure 5.4: HEOD functions on an example distribution highlighted by the blue bins. The black dotted lines express the kernel height estimation function (KHE) of the underlying distribution. The purple dotted line represents the density score for a given value of x, while the green dotted line represents the distance score. The red line represents the ultimate final score, which is obtained by combining both the density and distance scores.

to account for variations in contexts and baselines. Suppose that in a given scenario, the feature under consideration is *bytes sent* contextualized based on the *HTTP method*. Consider we have established a baseline of 90 POST events and 10 GET events (i.e., two separate histograms). Now, we have identified two outliers: one POST event that differs significantly in bytes sent compared to other POST events and one GET event that substantially differs from other GET events. In this scenario, both events are considered outliers since they deviate significantly from their respective baselines. However, there is more confidence in identifying the POST event as an outlier. This is because there were more observations available to compare its byte sent value to, in contrast to the GET event, which had fewer observations to compare against. Thus, the POST event should carry more weight in the outlier analysis as it is the more confident outlier identification.

In this work, we propose another transform function $C_w \in [d, 1]$, to derive a confidence weight for an outlier score given the total number of observations beneath the corresponding histogram. The fundamental concept entails starting from the minimum weight (d) and elevating the outlier score as we approach the anticipated minimum baseline size expectation.

$$C_w(n, d) = 1 - (1 - d).e^{\frac{-en^2}{h^2}}$$

$$S_c(X_i \mid M_i) = C_w(n, d)$$

Where d denotes the minimum weight value (throughout our experiment, set to 0.8), h is the parameter allowing one to control the minimum expectations of the baseline size (e.g., 100). n is the actual size of the underlying baseline/histogram (defined by a context) taken from the $histStats(c \mid M_i)$.

Note that while we derived the confidence from baseline size here, one could expand with other concepts such as age.

5.3.2 Ensemblers

The ensemble modules rely on a distinct attribute to establish a correlation between submodel outputs. Examples of such attributes include event ID, username, and asset. This attribute is highlighted as the "via" field in Figure 5.1. Once the ensemble module has determined the correlation attribute, it combines the independent outputs generated by each submodel when evaluating X_i to produce a unified and meaningful value called the ensemble score for that particular entry. This score is then utilized to rank the entry X_i.

In the realm of probabilities, numerous approaches exist for deriving joint probabilities. Nonetheless, in this study, we have opted to avoid probabilities due to their numerical instability when handling highly skewed distributions. Ensembling techniques have been studied extensively, particularly in the context of classification problems, e.g., bagging[29], boosting [83], stacking[44, 242, 275], random forests [30], model averaging [61], and a bucket of models [65]. Furthermore, numerous outlier detection algorithms underneath utilized ensemble techniques [91, 155, 157].

In this work, we have separated the ensembling process to not only formalize it but also facilitate continuous improvement of the underlying technique over time.

Formally, the objective is to derive the ensemble score $\bar{E}_i$ for a given data point X_i passed through all submodels in $\mathbf{M}$, where each submodel ($M_i := \mathcal{M}^{\omega}_{f|c,y}$) outputs the following attributes for every data point X_i:

- Outlier score $S(X_i \mid M_i) \in [0, 1]$

- Outlier confidence score $S_c(X_i \mid M_i) \in [0,1]$
- Submodel's weight $\omega \in [0,1]$

Our objective is to compute the final ensemble score $\bar{O}_i = \bar{E}(X_i \mid M)$ for data point X_i by aggregating all the outlier scores $\boldsymbol{S}$ and outlier confidence scores $\boldsymbol{S_c}$ assigned to it by the models in $\mathbf{M}$ while taking into consideration the submodel's weight ω.

Let $O(X_i) = \{(S(X_i \mid m), S_c(X_i \mid m)), \forall m \in M\} = \{(s_1, s_{c1}), (s_2, s_{c2}), ..., (s_n, s_{cn})\}$ denote a set of tuples that contains all the outlier scores and their corresponding outlier confidence scores assigned to data point X_i by the models in $\mathbf{M}$. Let $W = \{\omega_1, \omega_2, ..., \omega_n\}$ be the set of submodel weights in $\mathbf{M}$. Function 5.7 is designed to compute the ensemble score from a set of independent outlier scores and their corresponding confidence levels.

$$E(O, W) = \sum_{i=1}^{n} (s_i \cdot s_{ci} \cdot \omega_i)^3 \tag{5.7}$$

Hence, the final ensemble score is calculated as follows:

$$\bar{O}_i = \bar{E}_i = E(O(X_i), W)$$

The main intuition behind the ensembling functions is to magnify significant outlier scores to prevent them from averaging out due to the presence of numerous low scores. In essence, if poorly derived models are present, irrelevant scores from various components can weaken the overall outlier score.

Consider the following sets of scores, where each item in a set represents a submodel's score for an item.

```
(1) X1=[0.2, 0.2, 0.2, 0.2]
(2) X2=[0.0, 0.0, 0.4, 0.4]
(3) X3=[0.0, 0.0, 0.0, 0.8]
```

When considering the three items, selecting the average as the combination function results in equal outlier scores for all three scenarios. However, such an approach is not desirable, especially when considering domain knowledge. Item (3) should be regarded as more important as a model outputs a very high outlier score. While using the average as the combination function results in an equal value of 0.2 for all three scenarios, our function yields $E(X1) = 0.032$, $E(X2) = 0.128$, and $E(X3) = 0.512$, which better reflects the desired rank, i.e., (3), (2), and (1).

One can use various other methods to obtain comparable outcomes, such as utilizing the maximum function, logarithmic sum, damped averaging, and pruned averaging [5].

Explainable Results

The paper emphasized the significance of producing explainable interpretable outliers. HEOD's simple, intuitive design enables it to add interpretable justifications to each element. Every time a submodel scores an event, it approximates the number of times a particular feature value or values close to it have been observed under the given context. Each submodel also contains a baseline with statistics on its previous observations. The baseline statistics facilitate the submodel to generate not only the prediction but also the statistical focal points that led to that prediction, i.e., the thresholds used for the transformer functions.

Let us take an example of a submodel configuration with the feature bytes sent and the context of the HTTP method. When this submodel generates the density score for a given value of bytes sent, it can also provide the estimated frequency, average, standard deviation, and median of the underlying distribution (i.e., other bytes sent within the same context). This information enables the end-user to investigate easier why a specific event has received a particular score by a particular submodel.

Similarly, the ensmebler can aggregate and summarize all reasoning attached to the submodel outputs along with their contribution to the overall ensemble score. This feature enables the end-user to promptly analyze an outlier.

5.3.3 HEOD Framework: Ensemble of Ensembles

Having grasped the notions of submodels and ensemblers, we are now equipped to construct comprehensive models comprised of diverse submodels, each examining distinct data from various perspectives. These submodels can be combined through ensemblers to generate comprehensive models.

Figure 5.5 demonstrates how it is possible to link various submodels and their corresponding ensemblers to construct comprehensive models.

5.4 Experimental Setup

In this section, we present the details of our experimental setup for evaluating the proposed framework.

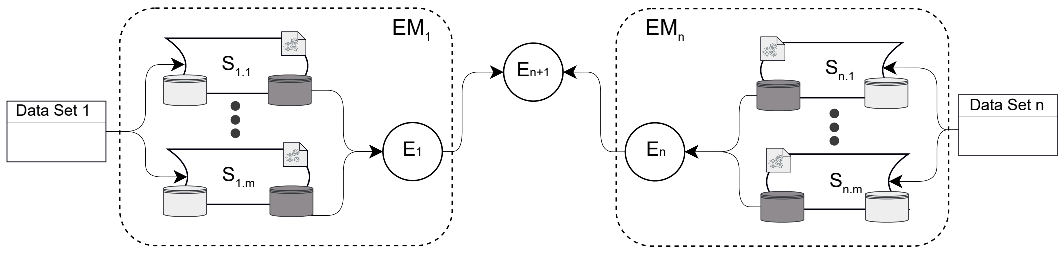


Figure 5.5: HEOD framework.

5.4.1 Implementation

One of our main requirements was the ability to handle big data in a real-world setting. As our framework enables us to embrace modular and distributed setups, we decided to implement the proposed framework in Apache Spark, an open-source distributed computing framework specially designed for big data processing and analytics. In addition, we specifically tailored the implementation for Databricks.

The proposed framework is intended to be operated and configured by an expert. Therefore we emphasized the importance of keeping the end user (in our case, the security analyst) in the loop for the design of the outlier detection framework. To achieve this, we asked a security analyst to evaluate each dataset used in the assessment thoroughly. They identified important features and how they could be contextualized. Then we utilized this information to derive our expert sub-models.

Nevertheless, we have also implemented a default version of the framework to ensure comparability with other algorithms. This version spawns default submodels for each data column (i.e., feature) with no context. The default submodel is assigned according to the feature type (i.e., categorical or numerical) with no context tree and a default weight of 1.

5.4.2 Infrastructure

We deployed our framework on a big data platform consisting of Azure Databricks and Azure Data Lake Storage. We used this infrastructure to evaluate our framework on public datasets and run it in a real-world setting. Specifically, the infrastructure was integrated into a large international organization, where a portion of the company's SIEM logs was residing.

Throughout our experiments, we configured Databricks with 8 "Standard_D32s_v4" workers. Thus, having a big data platform backed up by Apache Spark with a total of; 1024-GB Memory, 256 vCPU Cores having access to petabytes of data. We expand on the details of the data in the next section.

5.5 Evaluation

Outlier analysis is considered the most challenging core data mining problem to evaluate, particularly on real datasets, due to its unsupervised nature and small sample space. This is especially true in the cybersecurity domain. However, to ensure the effectiveness of our proposed technique, we have followed best practices in the literature by evaluating it alongside other algorithms in a cybersecurity-specific setting using a relevant dataset.

It is important to note that in cybersecurity, the focus is not solely on identifying outliers. Instead, the primary objective is to detect malicious activity, which outliers may not always reflect. Therefore, it is critical to have a comparison of algorithms that considers the specific requirements and objectives of the cybersecurity domain.

More specifically, we aim to answer the following questions as part of our evaluation:

- EQ1. How does the performance of our proposed outlier detection algorithm compare to other well-known algorithms when applied to public cybersecurity-related datasets?

- EQ2. Is our hypothesis valid that human-assisted outlier detection will perform better, meaning do the expert models defined by the analyst aid in enhancing the results and better transition from outlierness to maliciousness?

Finding a suitable dataset in the cybersecurity domain presents a significant challenge. Most well-known datasets, such as KDD CUP, suffer from issues, such as uneven distributions and imbalanced classes due to synthetic attack generation [132, 218, 240]. As a result, there is a need for more high-quality reference data points in the cybersecurity domain for algorithmic evaluation. Despite the challenges, we endeavored to identify datasets that could best meet our requirements. Specifically, we sought labeled datasets that closely resemble logs collected by a typical enterprise, preferably collected from a real-world setting. Lastly, a dataset consists of negative class labels (anomalies or attacks) that are significantly less prevalent than normals. In this regard, we selected Kyoto 2006+ [243] and UNSW-NB15 [181] as our preferred datasets.

The *UNSW-NB15* dataset [181] comprises the raw network packets generated by the IXIA PerfectStorm tool in the Cyber Range Lab of UNSW Canberra. The dataset is composed of a hybrid of real normal activities and synthetic contemporary attack behaviors, mainly nine types of attack categories known as the Analysis, Fuzzers, Backdoors, DoS Exploits, Reconnaissance, Generic, Shellcode, and Worms. The raw traffic was captured using the tcpdump tool, resulting in 100 GB of traffic, which was then processed using the Argus and Bro-IDS tools to generate a total of 49 features, each with a corresponding class label.

The whole dataset available on USNW website[24] spawns over 0.58 GB with a total of $2,540,044$ events stored in the four CSV files, namely, UNSW-NB15_1.csv, UNSW-NB15_2.csv, UNSW-NB15_3.csv, and UNSW-NB15_4.csv. The dataset contains $2,218,761$ legitimate flows and $321,283$ attack flows, with the attacks accounting for 14% of the whole dataset. Moustafa et al. [211] provide detailed information about the dataset and analyze the statistical properties of its features.

The original **Kyoto 2006+** [243] dataset [25] was built on the three years of real traffic data from November 2006 to August 2009, then extended to December 2015. It consists of fourteen statistical features derived from the KDD Cup 99 dataset and ten additional features. It was collected using honeypots, darknet sensors, email servers, and web crawlers. Protić [211] provides a comprehensive analysis and review of the dataset.

The Kyoto 2006+ dataset spawns over 19.683 GB compressed (over 138 GB uncompressed). It includes $806,095,624$ events classified into three main labels: normal (1), known attacks (-1), and unknown attacks (-2). More specifically, $160,873,849$ (19.95%) are normal, $640,618,555$ (79.47%) are known attacks, and $4,603,220$ (0.57%) are unknown attacks.

5.5.1 Data Preprocessing

Both datasets present similar features, capturing network traffic. The features include basic flow features such as source and destination IP address/port, source/destination bytes, etc., as well as context-specific features. For the detailed features description, refer to the provided references. These features primarily comprise numerical features with a few categorical features. Although our proposed algorithm can handle categorical data, other algorithms require numeric representations. Therefore, to ensure a fair comparison, we encoded all categorical data using label encoding for other algorithms.

Both Kyoto 2006+ and UNSW-NB15 raw data are quite large. Kyoto 2006+ 138GB

24 https://research.unsw.edu.au/projects/unsw-nb15-dataset
25 https://www.takakura.com/Kyoto_data/

with more than 808 million events, and UNSW-NB15 dataset with more than 2 million events

Although our algorithm can easily handle these datasets, even with our large-scale deployment, we were still unable to process the entire datasets when evaluating other outlier detection algorithms under PyOD[26]. This is because the PyOD implementation of the algorithms only includes single-machine parallel processing. To address this, we had to reduce the amount of data used. For UNSW-NB15, we only used one-quarter of the data (UNSW-NB15_1.csv file), which amounted to $700,001$ events ($677,786$ normal and $22,215$ attack).

For Kyoto 2006+, we only utilized the data from July to December 2015. Another issue with the Kyoto dataset is that most of the data consist of attacks. While this may be appropriate for a supervised learning task, it is not realistic for evaluating outlier detection in the field of cybersecurity, where we expect attacks to be extremely disproportionate to normal traffic. Therefore, we had to refine the data further by reducing the known attacks to match realistic expectations. Ensuring proper downsampling while preserving the underlying distribution can be exceedingly challenging. We attempted random stratified sampling of the known attack class for our experiments to be approximately less than 1% of the entire data. To ensure the best practices in preserving the original distribution, we used the months as the strata for sampling. This downsizing reduced the dataset to a total of $2,939,350$ ($2,912,578$ normal, $25,888$ known attacks, and 884 unknown attacks).

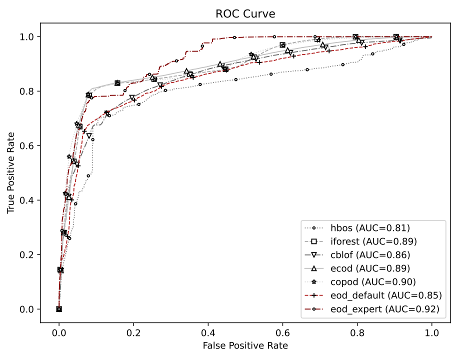
Lastly, we removed any features that are unrealistic in a real-world scenario and may cause a bias toward the label. For instance, we excluded the IDS_detection, Malware_detection, Ashula_detection features from the dataset.

5.5.2 Results

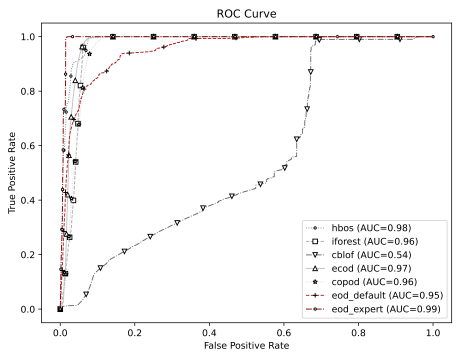
We have followed the standard practices in the field to evaluate the effectiveness of our proposed algorithm against other commonly used algorithms. The evaluation was conducted using the metrics described in Table 4.5.

Figure 5.6 (a) and 5.6 (b) presents the Receiver Operating Characteristics (ROC) curve, which depicts the performance of the HEOD algorithm against the selected algorithms using the UNSW-NB15 and Kyoto 2006+ datasets. Receiver Operating Characteristic (ROC) curve is a commonly used tool to evaluate the performance of a binary classifier. However, when dealing with imbalanced datasets and problems (e.g., outlier detection in the real world), where one class significantly outweighs the other, ROC might not be the best metric to highlight the classifier's performance.

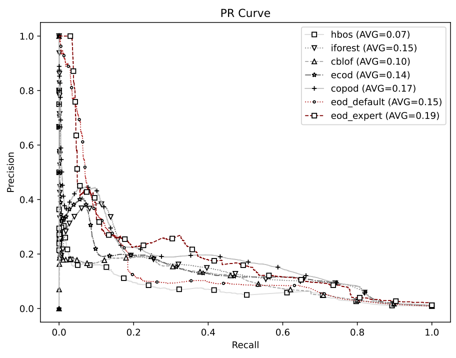
26 https://pyod.readthedocs.io/en/latest/



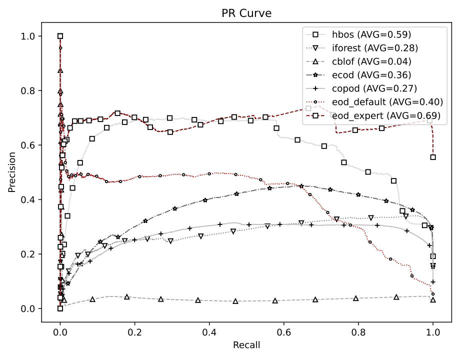
(a) The ROC curve on the Kyoto 2006+.



(b) The ROC curve on the UNSW-NB15.



(c) The PR curve on the Kyoto 2006+.



(d) The PR curve on the UNSW-NB15.

Figure 5.6: The Receiver Operating Characteristics (ROC) and Precision-Recall (PR) curves for evaluating the performance of the HEOD outlier detection algorithm against other commonly used algorithms on UNSW-NB15 and Kyoto 2006+ datasets

This is because ROC is insensitive to class imbalance and instead focuses on the overall accuracy of the classifier, which can be misleading. In imbalanced datasets, the classifier may achieve high accuracy by simply predicting the majority class (TNs) most of the time, ignoring the minority class. A more appropriate metric would be the precision-recall curve, which better represents the classifier's performance in an imbalanced setting as it replaces TN with another metric that does not directly take TN, i.e., *precision* [18, 245]. Figure 5.6 (d) and 5.6 (c) shows the precision-recall curve when evaluating the performance of the HEOD algorithm on the two USNW-NB15 and Kyoto 2006+ datasets.

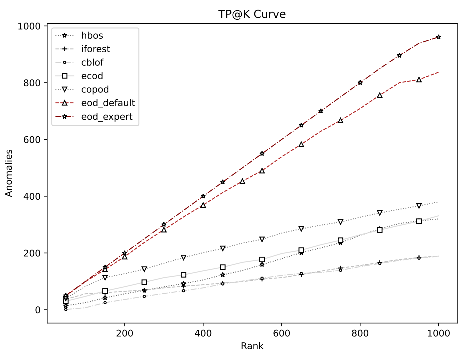
While the precision-recall curve is an excellent metric to evaluate overall performance, it is not well-suited for real-world scenarios. We can see that even in this public dataset, if we ensure feasibility (i.e., scale the outlier proportion to reflect the real world), the precision-recall curve demonstrates poor performance.

In the real world, where the data scale can reach terabytes with billions of events, typical precision-recall metrics would not be realistic. In other words, an analyst would never expect to find all true positives in the entire dataset. Instead, they expect to see the best results at the top k according to the rate they can manage. This is important as outlier detection in practice is not a binary classification problem but a ranking problem.

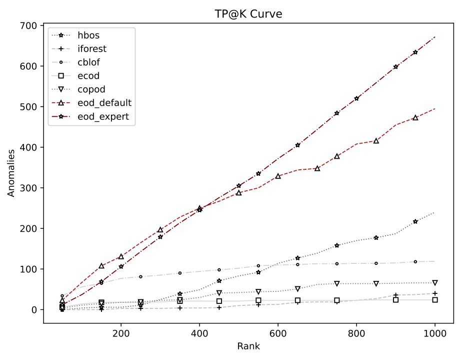
To better reflect the ranking as the additional objective of the outlier detection, we will be introducing three additional metrics: TP@K, Precision@K, and F1@K curves reflected in Figure5.7

The TP@K curve provides valuable information on the number of true positives detected as we increase the threshold for the top k-ranked outliers. This helps to answer the question of how many true positives are identified when considering the top k outliers. In addition, the Precision@K curve offers further insights into false positives by analyzing the number of true positives and false positives among the top k results. This aids in determining the level of precision the algorithm achieves at different values of k. Furthermore, the F1@K curve provides additional insights into missed outliers (TNs) by reflecting the F1 score, which is the harmonic mean of precision and recall, against the threshold k. This addresses what happens when the top k-ranked outliers are analyzed, i.e., how many are true outliers (TP), how many are missed (FN), and how many are false outliers (FP)?

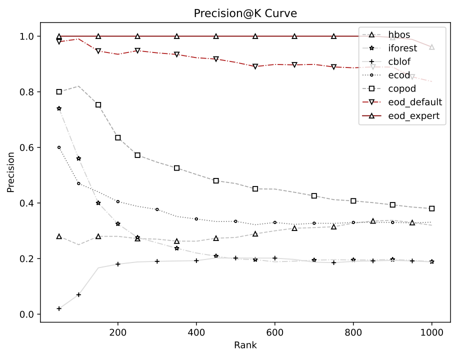
These new metrics are particularly useful for evaluating the performance of our algorithm against others, as they reflect the mindset of a SOC analyst who is limited by time and resources and is therefore focused on identifying the most significant outliers [10]. As a result, the analyst expects to find true positives at the beginning of the ranked list while minimizing the number of false positives encountered during the analysis.



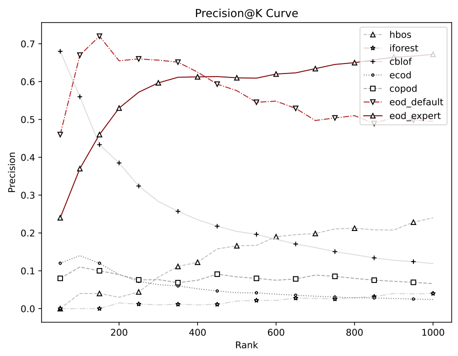
(a) The TP@K curve on the Kyoto 2006+.



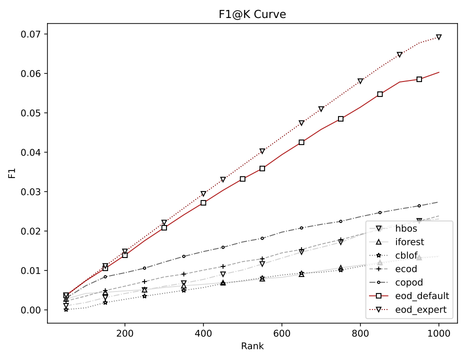
(b) The TP@K curve on the UNSW-NB15.



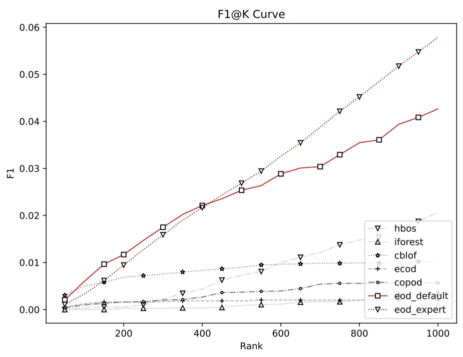
(c) The Precision@K curve on the Kyoto 2006+.



(d) The Precision@K curve on the UNSW-NB15.



(e) The F1@k curve on the Kyoto 2006+.



(f) The F1@k curve on the UNSW-NB15.

Figure 5.7: The TP@K, Precision@K, and F1@K (k=1000) curves for evaluating the performance of the HEOD outlier detection algorithm against other commonly used algorithms on the UNSW-NB15 and Kyoto 2006+ datasets.

The experiments demonstrate that our proposed algorithm consistently performs well compared to other well-known algorithms. Furthermore, it validates our hypothesis that human-assisted outlier detection, specifically an expert model where a domain expert contextualizes features, can significantly improve the results by enabling a more accurate transition from outlierness to maliciousness.

5.6 Case Study: LOLBins Detection

While the previous sections focused on evaluating HEOD as an outlier detection algorithm, this section will assess its potential for threat detection, especially in a real-world scenario. Particularly aimed at answering the following questions as part of the evaluation:

- EQ3. How feasible is it to run our proposed framework in a real-world setting?
- EQ4. What is the experience of running HEOD as an outlier detection use case for threat detection in a real-world setting?

In this regard, we have implemented HEOD on the described infrastructure (Section 5.4.2) and have productized it in a real-world setting, following the described implementation (Section 5.4.1).

The described big data platform was part of a larger cyberdefense infrastructure belonging to a major international organization with over 100, 000 assets. In this regard, we had access to the enterprise's EDR logs that were sent to the central SIEM. These logs were being streamed (near real-time) at scale (4TB compressed JSON per day), a true big data arriving at a high volume and velocity. This data was used to evaluate the HEOD framework in production over one month.

Data

The Endpoint Detection and Response (EDR) logs are among one the most valuable sources of logs as they provide visibility into activities and events that occur on the endpoints. EDR logs can include information on processes executed, network connections established, system events, security events, and other endpoint activities.

There are several commercial EDR solutions available, including CrowdStrike Falcon[27] and Microsoft Defender for Endpoint. Microsoft offers a comprehensive

27 https://www.crowdstrike.com/products/endpoint-security

overview of all events captured by such EDR solutions [172]. Other alternative solutions to allow one to capture similar events to those captured by EDR includes Windows Event Logs, and Sysmon [70].

While EDR logs contain a broad range of events, such as network connections, DNS resolutions, files, and registry modifications, for this experiment, we focused solely on one of the most valuable event types: *process creation*. This event includes details regarding the process executed, including but not limited to a device identifier, initiating process file name and file hash, execution path, command line, process integrity level, account SID, parent process ID, and others. To better understand the features captured by this event type, readers may refer to Microsoft Defender for endpoint event type `DeviceProcessEvents` [1] or Sysmon event ID 1 (Process Creation) [253].

Data Pre-processing

As mentioned, the raw data arrived in the form of compressed JSON files on ADLS, with each file containing various event types. The pipeline included an ETL workload that involved opening the compressed JSON files, parsing, duplicating, and partitioning events based on the event type and the date, and writing to ADLS as partitioned snappy compressed parquet files.

This data was also enriched with contextual information, such as attaching inventory data (information about the asset, users, and peer groups).

As previously stated, for this case study, we concentrated solely on process execution. Specifically, incremental baselining was performed over one month of data, which spanned over 12TB of compressed parquet files and contained over 4.1 billion events. Following this, a week's worth of data was utilized for scoring purposes, spanning over 2.7TB and containing over 2 billion in events.

5.6.1 HEOD Setup

We formulated the problem as detecting advanced threats that involve the use of Living-off-the-Land Binaries (LOLBins). LOLBins is a term used to describe legitimate executables, tools, or scripts that attackers can exploit to carry out malicious activities on a victim's system while also evading detection by security tools [148]. The misuse of LOLBins has become increasingly pertinent in today's security landscape, as their detection presents significant challenges. It is difficult to create signatures or heuristics to identify misuse of valid system tools, such as word.exe or cmd.exe, and blocking them is not a feasible solution. Thus making their detection a good case study for an anomaly-based detection approach via the proposed HEO framework.

To achieve this objective, we filtered the executions only to include those that were associated with LOLBin executables [160]. We solicited the input of SOC analysts to design a set of submodules for EDR's process execution events. These submodules were created with the aim of targeting specific dimensions or features within particular contexts, with a focus on identifying possible behavior outliers that could indicate LOLBins.

Table 5.3 presents examples of the finalized submodules obtained following the consolidation process with the SOC analysts and performing Exploratory Data Analysis (EDA). For information about the features used, refer to [1]. In addition to the default features provided by DeviceProcessEvents, we engineered a few additional ones, including asset peer group ID based on the enterprise inventory list and command line features [190], such as length, entropy, and the number of special characters.

Table 5.3: Example of submodel configuration for evaluating HEOD on process execution logs.

#SM	Context Tree (c)	Feature (f)	count (y)	Type (f_t)	Weight (ω)
1	-	ProcessChain(Parent-Child)	-	Categorical	1.00
2	-	InitiatingProcessSHA256	DeviceId	Categorical	1.00
3	InitiatingProcessFileNameNormalized	InitiatingProcessSHA256	-	Categorical	1.00
4	PeerGroupId -> ProcessChain	ProcessIntegrityLevel	-	Categorical	0.85
5	PeerGroupId -> ProcessChain	InitiatingProcessNormaAccountSid	-	Categorical	0.90
6	PeerGroupId -> ProcessChain	InitiatingProcessNormalizedFolderPath	-	Categorical	1.00
7	PeerGroupId -> NormProcessFilename	ProcessCommandLineLength	-	Categorical	0.65
8	PeerGroupId -> NormProcessFilename	ProcessCommandLineEntropy	-	Numerical	0.70
9	PeerGroupId -> NormProcessFilename	ProcessCommandLineSpecialChars	-	Numerical	80
10	PeerGroupId -> NormProcessFilename	ProcessCommandLineParamCount	-	Numerical	0.70

5.6.2 Results

As previously noted, the HEOD framework does not approach outlier detection as a classification problem but rather as a ranking problem. Consequently, all events are assigned an ensemble score. The goal is to achieve high precision when taking the top k events on a daily basis, whereby all top k events (alerts) should result in the identification of an event of interest in the best-case scenario.

However, in practice, achieving a binary categorization where the top k alerts are classified as either false positives (normal behavior with no threat) or true positives (a threat) is a challenge. To address this, we proposed the following categorization when evaluating the top k alerts per day:

- *False Positive*: The outliers that retain anomalous characteristics but do not exhibit any malicious intent.
- *Suspicious*: Outliers that initially appeared suspicious, but upon investigation, they were found to be not a threat, including instances of uncommon human behavior that may seem malicious.
- *Risky*: Outliers that indicate a potential risk that requires mitigation but may not have the most significant impact. Examples of such risks include indications of misconfigurations and policy violations (such as file transfer, tunneling, and crypto miners).
- *Incident*: An outlier that led to the escalation of an incident requiring attention from the response team.

We request the assistance of the enterprise's threat hunters and monitoring team. Specifically, we requested that they allocate time to review the top unique k alerts produced daily. We have represented the results of the investigations in Table 5.4, where each row corresponds to the findings for a particular day.

Table 5.4: HEOD case study results for detecting LOLBins in a real-world setting.

Day	0 (False Positive)	1 (Suspicious)	2 (Risky)	3 (Incident)	Total Alerts (k)
1	74	17	5	2	98
2	84	14	1	1	100
3	81	5	2	0	88
4	75	7	3	0	85
5	78	11	2	2	93
6	74	20	10	3	107
7	69	12	1	1	83
Total	535	86	24	9	654

After combining all classes into one (class TP) and calculating precision, the resulting value was 0.18, revealing a common misconception in cybersecurity about outlier/anomaly detection, particularly in real-world scenarios where high detection rates are not realistic and false positives remain a significant challenge [10]. This is not necessarily due to the outlier detection algorithm, as we have demonstrated that our proposed method performs equally well as other algorithms, particularly when contextualizing features with expert knowledge. Instead, the fault lies with the domain itself, where outlier behavior does not always translate to maliciousness. Effective outlier detection in cybersecurity requires extensive effort in feature engineering, threat modeling, and problem definition.

It is also worth noting that the number of successful attacks in large enterprises with numerous security solutions may be relatively low. In this regard, out of the nine incidents reported, only a handful was deemed to have significant value after triaging.

As a part of the case study, we also requested the analysts to share their feedback on hunting for results. The initial response indicated that the quality of the results was poor as the analysts were accustomed to evaluating alerts generated from heuristics (signatures), making their evaluation relatively straightforward. However, hunting for outlier results requires a different mindset, even though every outlier was adequately explained by the HEDO framework (thus ensuring explainability). These findings underscore the interpretability challenges associated with outlier detection, even when the results are explainable.

5.7 Discussion: Outlierness vs. Maliciousness

What is an "outlier"? Typically, an outlier refers to a data point or observation that falls beyond the normal range or distribution of a given dataset. In our case, this is characterized by density and distance. However, as mentioned before, it is important to note that the definition of an "outlier" can vary significantly depending on the context. One philosophical consideration is that the notion of an outlier is inherently subjective. The determination of what is deemed "typical" or "normal" can differ based on the observer's perspective, and what may be considered an outlier in one setting may not be regarded as such in another. Furthermore, the definition of an outlier can evolve over time. This is particularly relevant as we introduce more complex features, i.e., the curse of dimensional [129]. In this regard, the likelihood of finding outliers increases as the number of dimensions grows.

Outlierness Implies Maliciousness! Virtually all forms of anomaly detection in cybersecurity have two implicit assumptions: (i) there is some way to define the normal patterns, and (ii) deviations from the norm are indicators of undesirable activities. What if that is not the case? Outliers, or data points that deviate significantly from the norm, are often viewed as potential indicators of malicious activity in cybersecurity. However, it is essential to note that outlierness does not necessarily translate to maliciousness. In many cases, outliers can be the result of legitimate behavior, such as system fault, changes in system behavior, misconfiguration, noise, data quality issues (e.g., collection, parsing, etc.), human error, operational failures, environmental changes, and generally random human behavior. These are particularly common in the continuously changing IT landscape.

For example, a system administrator may perform tasks that deviate from the norm, such as running software updates or installing new programs, but these actions are not malicious in nature. Similarly, a user accessing the network from an unusual location or time may be flagged as an outlier, but this does not necessarily indicate malicious intent.

This way, using outlier detection alone as a method for identifying malicious activity can lead to a high number of false positives and many alerts that need to be investigated. This can be time-consuming and resource-intensive for security teams, leading to alert fatigue and reduced ability to respond to real threats [10]. That is also what we observed during our investigations.

This is a paradigm change as we are challenging the philosophy of outlier detection in cybersecurity, as legitimate but unusual behavior is more common than one thinks.

How to get to Maliciousness? In this chapter, we emphasized the significance of feature engineering and contextualization in enhancing the effectiveness of an outlier detection algorithm. This is critical in facilitating our transition from outlierness to maliciousness. In other words, define the problem as much as possible, incorporate domain knowledge, and select contextualized features that their anomalies can let us deduce maliciousness.

Consider an attacker who attempts to misuse standard system tools (LOLBins) like powershell.exe and cmd.exe to carry out their malicious intentions. In such cases, the application itself may not be anomalous, but its anomalous behavior, such as spawned processes and established network connections, can enable the detection. Therefore well-thought problem definition and threat modeling, contextualization, and feature engineering play a crucial role.

Nevertheless, it remains challenging to deduce malicious intent accurately. Hence, solely relying on anomaly detection to identify malicious activity is not recommended. As suggested by [134] and supported by our own experience. Outlier detection is based on statistics and probability, which means that analysts should exercise caution when relying solely on the system's prediction. Instead, analysts should use the system's explanations to determine the best curse of action and form their own further investigation [10]. Hence, it's crucial to develop such mindsets recognizing the limitations of outlier detection and approaching anomaly detection with a threat-hunting mindset rather than threat detection.

A promising avenue for future research involves merging outlier detection with expert knowledge and heuristic-based detection sequentially, thereby bridging the gap between human-centric and ML-centric detection. Furthermore, involving the analysts in the process helps establish trust and minimize skepticism [10].

5.8 Limitations and Future Work

Multivariate Features— In our approach, one fundamental limitation is the inability to capture the intricate relationships and dependencies between multiple features or attributes that may exist in a multivariate dataset. As a result, our method may not detect outliers that exist in multiple dimensions or only become apparent when multiple features are examined simultaneously.

We addressed this problem by contextualizing features to capture their dependencies based on expert knowledge. We emphasized the importance of contextualization in this domain. Nevertheless, one can further extend this approach by enabling submodels that support multivariate features, such as considering both request length and response length together and contextualizing them by destination and HTTP method.

One could also introduce more advanced submodels, particularly for the numerical features. For instance, a model that combines features such as transferred bytes and the duration into a normalized sparse vector contextualized by the destination and HTTP method and measures the density and distance in high-dimensional feature space. LOF [31], and GMM [219, 220] are notable examples of algorithms capable of density estimation in high-dimensional spaces. In addition, Mahalanobis [59], Euclidean [55], Manhattan [236], Cosine, and Hamming are widely recognized distance measurement techniques.

Algorithmic Improvement— In this chapter, we presented the rationale behind each aspect of our proposed algorithms. We also made an effort to keep the design as modular as possible, recognizing possible future improvements and optimizations to the underlying techniques, such as enhancing the density measurement, approximating height/count observations, and data reduction.

An area of improvement for our approach is the kernel height estimation function. Currently, one of the most computationally intensive tasks in our approach is the calculation of the estimated height, which requires extensive data shuffling and data collection, notably, if the data is not reduced (sampled or clustered). An intriguing future direction could involve adopting signal processing techniques and replacing the KDE with Fourier coefficients. By doing so, the coefficients can be stored and utilized for estimating the observation counts. Another potential algorithmic improvement involves the ensembling function, where the combination function could be learned to improve performance.

Highly Correlated Submodels— One of the limitations of our approach is its reliance on expert-defined submodels. If submodels are not carefully chosen, irrelevant outputs may be generated even though the ensembling process attempts to discard them. Furthermore, if the expert creates highly dependent submodels in which a high score in one submodel results in a high score in the others, in that case, the results may be unintentionally biased towards that particular submodel or feature. Therefore, it is crucial to perform basic exploratory data analysis (EDA) and feature correlation analysis to determine the submodel configuration.

In addition, it may be worthwhile to explore the possibility of learning submodel weights over time, which would introduce a feedback loop into the system.

Dependence Ensemble Framework— Our work introduced an ensemble framework that combines multiple submodels to create more comprehensive models for outlier detection. However, it is possible to introduce more complex logic to connect models and integrate signature-based filtering or dynamic thresholding into the pipeline for even greater effectiveness.

5.9 Literature Review

Outlier Detection

As a result of the inherent importance of outlier detection in various areas, considerable research efforts in the survey of outlier detection methods have been made [4, 6, 9, 38, 39, 94, 150, 199, 216, 256, 269, 288, 291].

One of the most comparable unsupervised outlier detection techniques similar to ours is the Histogram-Based Outlier (HBOS) algorithm [91]. This method uses static and dynamic bin width histograms to model the univariate feature densities and calculate the outlier score for each data instance. HBOS has shown high performance while maintaining computational efficiency, but it has been criticized for its lack of theoretical foundation and sensitivity to bin width. Nevertheless, HBOS remains a popular and widely used method for unsupervised anomaly detection.

Our proposed algorithm follows in the footsteps of HBOS but has several key differences. First, we use kernels instead of histograms to estimate the density. Second, we propose a scaling function that considers skewness rather than the default MaxAbs scaling. Third, we introduce a distance function in addition to the density-based score of HBOS. Lastly, we use different ensemble functions.

Other well-known algorithms in the domain include Isolation Forest [157], Local Outlier Factor [31], COPOD[154], and ECOD[155]. Although deep learning has been successful in many domains, there are still some technical uncertainties regarding its adoption for real-world anomaly detection applications, such as scalability and interoperability [196].

Anomaly/Outlier Detection in Cybersecurity

Several academic works have conducted surveys on the applications of anomaly detection in cybersecurity [7, 24, 69, 77, 123], particularly applied to Intrusion Detection Systems (IDS) [7, 19, 132]. Here we go through the most relevant works.

Yen et al. [284] introduce Beehive, a novel system designed to improve upon signature-based approaches for detecting security incidents using proxy logs. Beehive is a three-stage system that involves parsing, filtering, and normalizing data, followed by feature generation and clustering-based detection. The authors propose a set of 15 features, categorized as destination-based, host-based, policy-based, and traffic-based. These features are combined into a feature vector, which is then subjected to Principal Component Analysis (PCA) to reduce its dimensionality. Beehive uses an adopted K-means clustering method for anomaly detection, which does not require the number of clusters to be specified in advance. The authors evaluate Beehive using terabytes of event logs from a real-world setting at EMC and highlight its ability to detect malware infections and policy violations.

Our proposed HEOD framework follows a similar logic to the Beehive feature calculation; for instance, the destination prevalence feature of Beehive is equivalent to one of the potential HEOD submodels. Beehive composes the independent features into a vector, reduces its dimensionality via PCA, and uses K-means clustering for anomaly detection, while HEOD uses an ensemble function on top of independent univariate submodels' outputs. Beehive's emphasis on feature engineering and its deployment in a real-world setting makes it a valuable contribution to the field of threat detection. HEOD formalizes this mindset further as a generic framework for outlier detection applied to threat detection.

Lei [152] proposes an anomaly traffic detection algorithm that utilizes Support Vector Machine (SVM) algorithm with particle swarm optimization (PSO) to estimate parameters with greater accuracy. The proposed technique is evaluated on both the DARPA and KDDCup99 datasets. Yin et al. [285] present a deep learning-based approach for developing an anomaly intrusion detection system using a recurrent neural network (RNN). The RNN is employed for supervised classification learning, incorporating feedback from previous information and applying it to the current output. The authors evaluated the model's performance on the

NSL-KDD dataset through two experiments: binary classification and multiclass classification. Moustafa et al. [180] proposed a collaborative anomaly detection framework (CADF). The proposed approach involves capturing and logging network data, followed by pre-processing and employing the Gaussian Mixture Model (GMM) in combination with the interquartile range to identify abnormal patterns. Kim et al. [134] proposed an anomaly detection approach to identify unknown intrusions in the endpoint environment by utilizing Local Outlier Factor (LOF) and Autoencoder. The authors also introduced an attack profiling concept to create rules based on particular scenarios (e.g., Ransomware, Drive-by-Download) to identify malicious behavior from a set of anomalous suspicious events. The feature set used for anomaly detection includes the process name, local IP address, remote IP address, UNIX timestamp, file name, and event type. The authors confirmed the effectiveness of their proposed approach by detecting 107 new suspicious processes that were not identified previously using their self-collected dataset. In a study, Alhawi et al. [12] developed a decision tree classifier to identify Windows ransomware by utilizing features such as protocol type, IP addresses, packet and byte counts, and duration of network traffic. The researchers evaluated their approach and showed that the model achieved high accuracy in identifying ransomware using the created feature database.

Ongun et al. [192] propose an Active Learning framework called LOLAL to detect LOLBins. The authors introduce a novel command-line vectorization method, cmd2vec, which involves tokenizing and word embedding (using word2vec and FastText) to transform command lines into feature vectors. The active learning module in LOLAL uses a non-linear boosting classifier and a naïve Bayes anomaly detector, along with an adaptive sampling strategy, to iteratively select anomalous and uncertain samples for labeling by a human analyst. The authors demonstrate that LOLAL can achieve an F1 score of 96%. In a similar study, Utz [190] discusses an unsupervised approach for the same problem of LOLBins detection. The author proposes a set of features derived from the execution command line, including the existence of URL encoding, potential obfuscation, anomalous arguments, and others, in addition to the parent-child relationship of the process. These features are used as part of outlier detection using various algorithms such as Isolation Forest, LOF, and One-Class SVM. Both papers emphasize the importance of contextualization and feature engineering as integral parts of threat modeling.

As supported by the relevant literature, there is substantial potential for using unsupervised machine learning, particularly in cybersecurity, for identifying outliers and possibly detecting threats. One thing in common in most studies was the importance of feature engineering and threat modeling. Notably, the best outcomes were achieved when the features were generated with a cybersecurity-oriented perspective. For example, rather than blindly use Sha256 as part of a feature vector, calculate the prevalence based on the Sha256 and use that as a feature. The HEOD framework has been developed to facilitate this process while providing an intuitive and scalable algorithm tailored to cybersecurity requirements.

5.10 Chapter Summary

This chapter challenges the traditional assumptions associated with anomaly detection in cybersecurity and elaborates on the requirements for successful adoption. Leading to the proposed HEOD framework designed to bring outlier detection to cybersecurity.

HEOD emphasizes explainability and interoperability, adopting a SOC analyst mindset that examines independent features within their specific contexts. The framework consists of an ensemble of simple uni-variate submodels specifically designed to examine a single, carefully selected feature within a given context (hence human-assisted). Every submodel consists of baselines which can be conceptualized as a set of histograms showing frequency observations of a single feature within different contexts. Given a histogram, the submodel is capable of scoring observation values. The scoring is based on two principles: the lower the observation density and the further from the closest density, the more significant the outlier. The submodels' independent outputs are then combined to derive a single outlier score representing the overall outlierness of an event.

We assessed the HEOD algorithm's performance against other commonly used outlier detection algorithms on the USNW-NB15 and Kyoto 2006+ datasets. Apart from the conventional metrics, such as ROC and PR curves, we introduced two novel metrics - Precision@K and F1@K. These metrics are based on the expectation that a reliable outlier detection algorithm should keep true positives (TP) in the top k-scored entries while minimizing the false positives (FP), where k reflects the analyst's capacity to handle alerts in a real-world scenario. Our findings revealed that HEOD surpassed all other algorithms, particularly in Precision@K. Furthermore, the results validate our hypothesis that integrating domain expert knowledge (contextualized features) significantly enhances overall performance.

Finally, as part of a case study, we deployed the HEOD framework in a real-world setting next to the SIEM system of a larger international enterprise, analyzing 14.7 TB of data with more than 6.1 billion events. The system was configured with a set of submodels tackling the problem of LOLBins detection via Endpoint Detection and Response (EDR) logs, particularly process execution events. The results demonstrated that a good anomaly detection technique has the potential to identify unknowns while highlighting the unavoidable challenge of high false positive rates in a real-world setting. This led us to a discussion and future work on how we can continue to improve anomaly detection in cybersecurity, not just as an algorithm but as a paradigm shift, by challenging the mindset of threat detection versus threat hunting.

In Summary, the HEOD framework is intended to assist cybersecurity SOC analysts in effectively utilizing outlier detection. The approach focuses on threat modeling and collaboration with domain experts to generate submodels with univariate and contextualized features. As a result, the framework can be customized to address various issues, such as command-and-control detection through analysis of proxy logs (e.g., infrequent destinations, user agents, duration, etc.), malware detection using process execution logs (e.g., Sha256 prevalence, path, SID, etc.), and so on. Moreover, the theories used in HEOD, which involves breaking down large problems into smaller ones and utilizing independent submodels with a mechanism to combine their scores, can be applied to address problems that don't necessarily require outlier detection. For example, a system designed to detect beaconing could be composed of multiple submodels, each intended to score a specific attribute such as periodicity of connection, executable prevalence, and rarity of other features [185].

6 Conclusions & Outlook

Data-driven decision-making is the process of making informed and objective decisions by analyzing relevant data and information. This process typically involves applying various techniques, such as data mining, statistical analysis, and machine learning, to extract insights from large datasets. The overarching goal of data-driven decision-making is to improve the accuracy and efficiency of decision-making by basing decisions on empirical evidence and objective analysis rather than intuition or personal experience alone.

Many of today's organizations use data-driven decision-making to optimize their operations, enhance their performance, and achieve their strategic objectives, such as more effective maintenance, marketing, recommendations, etc. [28, 212].

Despite being one of the most data-rich domains, the cybersecurity industry has failed to fully incorporate data-driven decision-making into its practices. While there have been significant advancements in cybersecurity technologies and techniques, decision-making within the industry continues to rely heavily on human intuition and experience, with the underutilization of data-driven approaches [10].

The cybersecurity research community has been actively exploring advanced data mining techniques to tackle the ever-increasing complexity of cybersecurity problems. However, while there has been significant progress in this area, many of these techniques are yet to be successfully adopted in real-world settings. Many of the techniques being discussed in the cybersecurity research community are considered unrealistic, and lack practicality [10, 18].

6.1 Contributions of this Thesis

The contributions of this thesis can be summarized as follows:

Security Information, Event Management, and Analytics (SIEMA)

Identifying the limitations of current SIEM systems, specifically their inability to perform advanced analytics and incorporate cutting-edge data mining, machine learning, and graph mining approaches. To address this gap, it introduces the concept of SIEMA (Security Information/Event Management and Analytics), a next-generation SIEM that enables advanced analytical capabilities.

The reference architecture for SIEMA is drawn from best practices and design patterns of big data architectures and pipelines.

Two versions of SIEMA were implemented and deployed: a research workbench using open-source technologies (e.g., Apache Spark, Flink, Arrow, HDFS, Yarn) for academic purposes and an industrial PoC with commercial alternatives (e.g., Databricks and Azure Data Lake Storage) in a real-world environment alongside an international organization's traditional SIEM system.

The analytical capabilities of SIEMA contribute to advancing research in data mining for threat detection and developing realistic and successful use cases in an industrial setting.

Data-driven Threat Detection

With architectural support, one can tackle many SOC operations. This thesis specifically focused on data-driven threat detection through the application of outlier detection Chapter 5 and graph mining Chapter 4. More specifically:

CyberHIN, MalRank, MalLink: Formulating threat detection as a large-scale graph inference problem based on two fundamental principles: *guilt-by-association* and *exempt-by-reputation*. This led to the introduction of our proposed graph-inference algorithm named MalRank, a scalable graph-based inference algorithm designed to infer a node's maliciousness score based on its association with other nodes. MalRank offers several unique characteristics that distinguish it from other graph-based inference algorithms, as evidenced by its superior performance when compared to other algorithms, including Belief Propagation.

MalRank is designed to operate on a Heterogeneous Information Network (HIN) that is derived from events/logs within SIEM systems. This thesis proposes CyberHIN, a knowledge graph that is constructed from entities and relationships extracted from endpoint EDR logs and network Proxy and DNS logs, further enriched with related OSINT and CTI. The entities and relationships in CyberHIN emphasize on global features - shared characteristics and associations among different entities such as written/read files, loaded libraries, network connections, ASN, and registrar associations.

Subsequently, MalLink is introduced, a system designed to automate the process of tailoring the generation of CyberHIN and executing MalRank in a real-world setting. Both MalLink and MalRank utilize Apache Spark to meet the scalability requirements, enabling them to run effectively on SIEMA.

In a subsequent real-world deployment, MalLink was leveraged to detect variants of previously known malicious entities. The obtained results validate the original hypotheses of guilt-by-association and exempt-by-reputation. Moreover, MalLink has highlighted several intrinsic features, including its ability to rectify erroneous threat intelligence data and its potential to assess and quantify the accuracy of CTI.

HEOD: Challenging the philosophy of outlier detection when applied in cybersecurity and emphasizing the importance of keeping the SOC analysis in the loop, leading to the introduction of Human-assisted Ensemble Outlier Detection (HEOD). HEOD framework is designed to bring outlier detection to cybersecurity by stressing explainability and interoperability, adopting a SOC analyst mindset that examines independent features within their specific contexts. The framework consists of an ensemble of simple univariate submodels specifically designed to examine a single, carefully selected feature within a given context (hence human-assisted).

The algorithm underlying the HEOD framework has demonstrated superiority compared to other widely used outlier detection algorithms, particularly when evaluating the top k scored outliers. Additionally, the real-world implementation of HEOD for detecting LOLBins has been found to be feasible, but not without its challenges.

Despite this, the research underscores the challenging task of transitioning from outlierness to maliciousness.

6.2 Future Work

This thesis primarily focuses on addressing one of the most critical operations of today's Security Operations Center (SOC), which is threat detection. The study has incorporated learnings from outlier detection and graph mining to design advanced use cases. However, there are numerous other research avenues that could be explored to enhance SOC operations further. For instance, the incorporation of Natural Language Processing (NLP) techniques [158, 270] or time series analysis [111]. Furthermore, data-driven problem-solving approaches can also be applied to address other SOC-related challenges such as risk assessment, alert prioritization [10, 22], vulnerability prediction [122], etc.

One of the principal contributions of this thesis was the development of custom algorithms that are unique in their design. Although we provided the rationale for each design decision, we acknowledge the potential for improvements, as discussed in the future work section of the respective chapters.

Lastly, modern organizations require more than a next-generation SIEM with advanced analytical capabilities; they need a fully managed system that is also capable of taking actions, i.e., SOAR and XDR. However, this thesis solely focused on the SIEM and analytical capabilities and excluded data collection and incident response from its scope. Nevertheless, it is plausible to extend the system's functionality by assuming the presence of dedicated agents that can manage and enable new capabilities, including enhanced event collection, correlation, edge processing, asset discovery, vulnerability assessment and scanning, policy and configuration checking, file integrity, incident response, and many others. With such a system, one can take data-driven decision-making one step further and automate decision execution.

6.3 Final Remarks

Applying a data-driven mindset to security operations is challenging due to several factors, including the architectural limitations of current systems, the complexity of cybersecurity data, the shortage of skilled data scientists with expertise in cybersecurity, and the domain's rapidly evolving and adversarial nature. However, despite these challenges, this thesis advocates the shift from human-centered security operations toward a more data-centric approach, promoting next-generation SOCs.

Bibliography

[1] *365 Defender DeviceProcessEvents*. Accessed: 2023-03-04. Microsoft. 2023. URL: https://learn.microsoft.com/microsoft-365/security/defender/advanced-hunting-deviceprocessevents-table (see pages 140, 141).

[2] **A Practical Guide to Next-Generation SIEM**. Tech. rep. SENSAGE (see page 10).

[3] Anne Aarness. *XDR VS SIEM VS SOAR*. 2022. URL: https://www.crowdstrike.com/cybersecurity-101/what-is-xdr/xdr-vs-siem-vs-soar/ (visited on 03/04/2023) (see page 21).

[4] Charu C Aggarwal. "An introduction to outlier analysis." In: *Outlier analysis*. Springer, 2017, 1–34 (see page 146).

[5] Charu C Aggarwal. **Outlier ensembles: position paper**. *ACM SIGKDD Explorations Newsletter* 14:2 (2013), 49–58 (see page 131).

[6] Shikha Agrawal and Jitendra Agrawal. **Survey on anomaly detection using data mining techniques**. *Procedia Computer Science* 60 (2015), 708–713 (see page 146).

[7] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. **A survey of network anomaly detection techniques**. *Journal of Network and Computer Applications* 60 (2016), 19–31 (see pages 105, 147).

[8] Md Manjurul Ahsan, MA Parvez Mahmud, Pritom Kumar Saha, Kishor Datta Gupta, and Zahed Siddique. **Effect of data scaling methods on machine learning algorithms and model performance**. *Technologies* 9:3 (2021), 52 (see page 121).

[9] Leman Akoglu, Hanghang Tong, and Danai Koutra. **Graph based anomaly detection and description: a survey**. *Data mining and knowledge discovery* 29:3 (2015), 626–688 (see pages 110, 146).

[10] Bushra A Alahmadi, Louise Axon, and Ivan Martinovic. **99% False Positives: A Qualitative Study of {SOC} Analysts' Perspectives on Security Alarms**. In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022, 2783–2800 (see pages 1, 3, 4, 21, 36, 43, 105, 110, 111, 137, 142, 144, 145, 151, 153).

[11] Mamoun Alazab. **Profiling and classifying the behavior of malicious codes**. *Journal of Systems and Software* 100 (2015), 91–102 (see page 52).

[12] Omar MK Alhawi, James Baldwin, and Ali Dehghantanha. "Leveraging machine learning techniques for windows ransomware network traffic detection." In: *Cyber threat intelligence*. Springer, 2018, 93–106 (see pages 51, 148).

[13] Osama Almanna. *StartSSL Domain validation (Vulnerability discovered).* Accessed 06-11-2018. 2016 (see page 59).

[14] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. **Building a Dynamic Reputation System for DNS**. In: *USENIX Security Symposium*. 2010, 273–290 (see pages 56–58).

[15] Manos Antonakakis, Roberto Perdisci, Wenke Lee, Nikolaos Vasiloglou, and David Dagon. **Detecting Malware Domains at the Upper DNS Hierarchy.** In: *USENIX Security Symposium*. Vol. 11. 2011, 1–16 (see pages 55, 56, 60).

[16] Frank Apap, Andrew Honig, Shlomo Hershkop, Eleazar Eskin, and Sal Stolfo. **Detecting malicious software by monitoring anomalous windows registry accesses**. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2002, 36–53 (see page 54).

[17] Michael Armbrust, Ali Ghodsi, Reynold Xin, and Matei Zaharia. **Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics**. In: *Proceedings of CIDR*. 2021, 8 (see page 28).

[18] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. **Dos and don'ts of machine learning in computer security**. *arXiv preprint arXiv:2010.09470* (2020) (see pages 4, 43, 106, 110, 137, 151).

[19] Stefan Axelsson. **Intrusion detection systems: A survey and taxonomy** (2000) (see pages 105, 147).

[20] Aya Ayadi, Oussama Ghorbel, Abdulfattah M Obeid, and Mohamed Abid. **Outlier detection approaches for wireless sensor networks: A survey**. *Computer Networks* 129 (2017), 319–333 (see page 107).

[21] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. **Fast incremental and personalized pagerank**. *Proceedings of the VLDB Endowment* 4:3 (2010), 173–184 (see page 48).

[22] Tao Ban, Ndichu Samuel, Takeshi Takahashi, and Daisuke Inoue. **Combat security alert fatigue with AI-assisted techniques**. In: *Cyber Security Experimentation and Test Workshop*. 2021, 9–16 (see page 153).

[23] Adam Bates and Wajih Ul Hassan. **Can data provenance put an end to the data breach?** *IEEE Security & Privacy* 17:4 (2019), 88–93 (see page 51).

[24] Monowar H Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal K Kalita. **Network anomaly detection: methods, systems and tools**. *Ieee communications surveys & tutorials* 16:1 (2013), 303–336 (see page 147).

[25] David Bianco. **The Pyramid of Pain** (2014) (see pages 14, 43).

[26] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. **EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis**. In: *Ndss.* 2011 (see pages 56, 60).

[27] Lucien Birgé and Yves Rozenholc. **How many bins should be put in a regular histogram**. *ESAIM: Probability and Statistics* 10 (2006), 24–45 (see page 118).

[28] Alexandros Bousdekis, Katerina Lepenioti, Dimitris Apostolou, and Gregoris Mentzas. **A review of data-driven decision-making methods for industry 4.0 maintenance applications**. *Electronics* 10:7 (2021), 828 (see page 151).

[29] Leo Breiman. **Bagging predictors**. *Machine learning* 24:2 (1996), 123–140 (see page 129).

[30] Leo Breiman. **Random forests**. *Machine learning* 45:1 (2001), 5–32 (see page 129).

[31] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. **LOF: identifying density-based local outliers**. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data.* 2000, 93–104 (see pages 109, 145, 147).

[32] Sergey Brin and Lawrence Page. **The anatomy of a large-scale hypertextual web search engine**. *Computer networks and ISDN systems* 30:1-7 (1998), 107–117 (see page 47).

[33] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. **A comprehensive survey of graph embedding: Problems, techniques, and applications**. *IEEE Transactions on Knowledge and Data Engineering* 30:9 (2018), 1616–1637 (see page 99).

[34] Qiang Cao, Michael Sirivianos, Xiaowei Yang, and Tiago Pregueiro. **Aiding the detection of fake accounts in large scale social online services**. In: *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation.* USENIX Association. 2012, 15–15 (see page 48).

[35] *CAPEC: Common Attack Pattern Enumeration and Classification.* MITRE Corporation. 2023. URL: https://capec.mitre.org/ (visited on 03/01/2023) (see page 14).

[36] Silvia Cateni, Valentina Colla, Marco Vannucci, Jesus Aramburo, and Antonio Ramirez Trevino. **Outlier detection methods for industrial applications**. *Advances in Robotics, Automation and Control* (2008), 265–282 (see page 105).

[37] Zeynel Cebeci and Figen Yıldız. **Unsupervised discretization of continuous variables in a chicken egg quality traits dataset**. *Turkish Journal of Agriculture-Food Science and Technology* 5:4 (2017), 315–320 (see page 118).

[38] Raghavendra Chalapathy and Sanjay Chawla. **Deep learning for anomaly detection: A survey**. *arXiv preprint arXiv:1901.03407* (2019) (see pages 110, 146).

[39] Varun Chandola, Arindam Banerjee, and Vipin Kumar. **Anomaly detection: A survey**. *ACM computing surveys (CSUR)* 41:3 (2009), 1–58 (see pages 105, 146).

[40] Duen Horng "Polo" Chau, Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. **Polonium: Tera-scale graph mining and inference for malware detection**. In: *Proceedings of the 2011 SIAM International Conference on Data Mining*. SIAM. 2011, 131–142 (see pages 3, 44, 53, 81, 82, 97, 100, 105).

[41] Lingwei Chen, Tao Li, Melih Abdulhayoglu, and Yanfang Ye. **Intelligent malware detection based on file relation graphs**. In: *Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015)*. IEEE. 2015, 85–92 (see page 97).

[42] D Chismon and M Ruks. **Threat intelligence: Collecting, analysing, evaluating**. *MWR InfoSecurity Ltd* (2015) (see pages 17, 61).

[43] *CIS Critical Security Controls*. Center for Internet Security. 2021. URL: https://www.cisecurity.org/control (see page 8).

[44] Bertrand Clarke. **Comparing Bayes model averaging and stacking when model approximation error cannot be ignored**. *Journal of Machine Learning Research* 4:Oct (2003), 683–712 (see page 129).

[45] *COBIT*. ISACA Knowledge Center. 2019. URL: https://www.isaca.org/resources/cobit (see page 8).

[46] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and William Polk. **RFC5280: Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile**. Tech. rep. IETF, 2008 (see pages 58, 59).

[47] MITRE Corporation. *ATT&CK: Commonly Used Port*. 2018. URL: https://attack.mitre.org/wiki/Technique/T1043 (see page 56).

[48] CrowdStrike, Inc. **Global Threat Report**. Tech. rep. 2023. URL: https://go.crowdstrike.com/rs/281-OBQ-266/images/CrowdStrike2023GlobalThreatReport.pdf (visited on 03/17/2023) (see page 105).

[49] Chris Crowley and John Pescatore. **Common and best practices for security operations centers: Results of the 2019 SOC survey**. *SANS, Bethesda, MD, USA, Tech. Rep* (2019) (see page 4).

[50] *CVE: Common Vulnerabilities and Exposures*. MITRE Corporation. 2023. URL: https://attack.mitre.org/ (visited on 03/01/2023) (see page 13).

[51] *CWE: Common Weakness Enumeration*. MITRE Corporation. 2023. URL: https://attack.mitre.org/ (visited on 03/01/2023) (see page 13).

[52] Critical Infrastructure Cybersecurity. **Framework for improving critical infrastructure cybersecurity**. *URL: https://nvlpubs. nist. gov/nistpubs/CSWP/NIST. CSWP* 4162018 (2018) (see page 7).

[53] Leslie Daigle. **WHOIS protocol specification**. Tech. rep. 2004 (see page 60).

[54] Taurus T Dang, Henry YT Ngan, and Wei Liu. **Distance-based k-nearest neighbors outlier detection method in large-scale traffic data**. In: *2015 IEEE International Conference on Digital Signal Processing (DSP)*. IEEE. 2015, 507–510 (see page 109).

[55] Per-Erik Danielsson. **Euclidean distance mapping**. *Computer Graphics and image processing* 14:3 (1980), 227–248 (see pages 109, 145).

[56] R. Danyliw. *The Incident Object Description Exchange Format Version 2*. RFC 797. 2022. URL: https://github.com/mandiant/OpenIOC_1.1 (see page 14).

[57] Shubhomoy Das, Weng-Keen Wong, Thomas Dietterich, Alan Fern, and Andrew Emmott. **Incorporating expert feedback into active anomaly discovery**. In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE. 2016, 853–858 (see page 110).

[58] Brian Davison. **Propagating trust and distrust to demote web spam** (2006) (see page 48).

[59] Roy De Maesschalck, Delphine Jouan-Rimbaud, and Désiré L Massart. **The mahalanobis distance**. *Chemometrics and intelligent laboratory systems* 50:1 (2000), 1–18 (see pages 109, 145).

[60] Christoph Dietzel, Anja Feldmann, and Thomas King. **Blackholing at ixps: On the effectiveness of ddos mitigation in the wild**. In: *International Conference on Passive and Active Network Measurement*. Springer. 2016, 319–332 (see page 55).

[61] Pedro Domingos. **Bayesian averaging of classifiers and the overfitting problem**. In: *ICML*. Vol. 747. 2000, 223–230 (see page 129).

[62] Boxiang Dong, Zhengzhang Chen, Hui Wang, Lu-An Tang, Kai Zhang, Ying Lin, Zhichun Li, and Haifeng Chen. **Efficient discovery of abnormal event sequences in enterprise security systems**. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 2017, 707–715 (see page 99).

[63] Sumeet Dua and Xian Du. **Data mining and machine learning in cybersecurity**. CRC press, 2016 (see pages 3, 15, 19, 20).

[64] Alain Durand, Ralph Droms, James Woodyatt, and Y Lee. **RFC6333: Dual-stack lite broadband deployments following IPv4 exhaustion**. Tech. rep. IETF, 2011 (see page 55).

[65] Saso Dzeroski and Bernard Zenko. **Is combining classifiers better than selecting the best one?** In: *ICML*. Vol. 2002. Citeseer. 2002, 123e30 (see page 129).

[66] Michel J van Eeten and Johannes M Bauer. **Economics of malware: Security decisions, incentives and externalities**. *OECD Science, Technology and Industry Working Papers* 2008:1 (2008). DOI: https://doi.org/10.1787/241440230621. URL: https://www.oecd-ilibrary.org/content/paper/241440230621 (see page 60).

[67] Mohamed G Elfeky, Walid G Aref, and Ahmed K Elmagarmid. **Periodicity detection in time series databases**. *IEEE Transactions on Knowledge and Data Engineering* 17:7 (2005), 875–887 (see page 37).

[68] Mohamed G Elfeky, Walid G Aref, and Ahmed K Elmagarmid. **WARP: time warping for periodicity detection**. In: *Fifth IEEE International Conference on Data Mining (ICDM'05)*. IEEE. 2005, 8–pp (see page 37).

[69] Sohaila Eltanbouly, May Bashendy, Noora AlNaimi, Zina Chkirbene, and Aiman Erbad. **Machine learning techniques for network anomaly detection: A survey**. In: *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT)*. IEEE. 2020, 156–162 (see pages 105, 147).

[70] *Encyclopedia: Sysmon and Windows Security Log Events*. Accessed: 2023-03-04. Ultimate IT Security. 2023. URL: https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/ (see page 140).

[71] Thomas Erl, Wajid Khattak, and Paul Buhler. **Big data fundamentals: concepts, drivers & techniques**. Prentice Hall Press, 2016 (see page 22).

[72] Eleazar Eskin. **Anomaly detection over noisy data using learned probability distributions** (2000) (see page 109).

[73] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. **A density-based algorithm for discovering clusters in large spatial databases with noise.** In: *kdd*. Vol. 96. 34. 1996, 226–231 (see page 110).

[74] Dhivya Eswaran, Stephan Günnemann, Christos Faloutsos, Disha Makhija, and Mohit Kumar. **Zoobp: Belief propagation for heterogeneous networks**. *Proceedings of the VLDB Endowment* 10:5 (2017), 625–636 (see page 47).

[75] European Union Agency for Cybersecurity. **Threat Landscape for Supply Chain Attacks**. Tech. rep. 2021. URL: https://www.enisa.europa.eu/publications/threat-landscape-for-supply-chain-attacks (visited on 03/17/2023) (see page 105).

[76] Yujie Fan, Shifu Hou, Yiming Zhang, Yanfang Ye, and Melih Abdulhayoglu. **Gotcha-sly malware! scorpion a metagraph2vec based malware detection system**. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, 253–262 (see page 98).

[77] Gilberto Fernandes, Joel JPC Rodrigues, Luiz Fernando Carvalho, Jalal F Al-Muhtadi, and Mario Lemes Proença. **A comprehensive survey on network anomaly detection**. *Telecommunication Systems* 70:3 (2019), 447–489 (see pages 3, 15, 147).

[78] César Ferri, José Hernández-Orallo, and R Modroiu. **An experimental comparison of performance measures for classification**. *Pattern Recognition Letters* 30:1 (2009), 27–38 (see page 78).

[79] Stephen Fewer. *Reflective DLL injection*. 2008 (see page 14).

[80] Barbara Filkins. **An Evaluator's Guide to NextGen SIEM**. *Gartner Group Research Note* (2020) (see page 1).

[81] FireEye. **Highly evasive attacker leverages SolarWinds supply chain to compromise multiple global victims with SUNBURST backdoor**. *FireEye Threat Research* (2020) (see page 2).

[82] David Freedman and Persi Diaconis. **On the histogram as a density estimator: L 2 theory**. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete* 57:4 (1981), 453–476 (see page 126).

[83] Yoav Freund and Robert E Schapire. **A decision-theoretic generalization of on-line learning and an application to boosting**. *Journal of computer and system sciences* 55:1 (1997), 119–139 (see page 129).

[84] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. **Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning**. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 2017, 1797–1806 (see page 98).

[85] Ajit Gaddam. **Securing your big data environment**. *Black Hat USA* 2015 (2015) (see page 31).

[86] Jing Gao and Pang-Ning Tan. **Converting output scores from outlier detection algorithms into probability estimates**. In: *Sixth International Conference on Data Mining (ICDM'06)*. IEEE. 2006, 212–221 (see page 120).

[87] Joseph Gardiner, Marco Cova, and Shishir Nagaraja. **Command & Control: Understanding, Denying and Detecting-A review of malware C2 techniques, detection and defences**. *arXiv preprint arXiv:1408.1136* (2014) (see page 37).

[88] Wolfgang Gatterbauer, Stephan Günnemann, Danai Koutra, and Christos Faloutsos. **Linearized and single-pass belief propagation**. *arXiv preprint arXiv:1406.7288* (2014) (see pages 68, 96).

[89] Gebeyehu Belay Gebremeskel, Chai Yi, Zhongshi He, and Dawit Haile. **Combined data mining techniques based patient data outlier detection for healthcare safety**. *International Journal of Intelligent Computing and Cybernetics* (2016) (see page 105).

[90] Dr A SHAJI GEORGE, AS Hovan George, T Baskar, and Digvijay Pandey. **XDR: The Evolution of Endpoint Security Solutions-Superior Extensibility and Analytics to Satisfy the Organizational Needs of the Future**. *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)* 8:1 (2021), 493–501 (see page 21).

[91] Markus Goldstein and Andreas Dengel. **Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm**. *KI-2012: poster and demo track* 9 (2012) (see pages 109, 129, 146).

[92] Palash Goyal and Emilio Ferrara. **Graph embedding techniques, applications, and performance: A survey**. *Knowledge-Based Systems* 151 (2018), 78–94 (see page 99).

[93] Aditya Grover and Jure Leskovec. **node2vec: Scalable feature learning for networks**. In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, 855–864 (see page 98).

[94] Manish Gupta, Jing Gao, Charu C Aggarwal, and Jiawei Han. **Outlier detection for temporal data: A survey**. *IEEE Transactions on Knowledge and data Engineering* 26:9 (2013), 2250–2267 (see page 146).

[95] Will Hamilton, Zhitao Ying, and Jure Leskovec. **Inductive representation learning on large graphs**. In: *Advances in neural information processing systems*. 2017, 1024–1034 (see pages 49, 95, 98).

[96] Jiawei Han, Jian Pei, and Hanghang Tong. **Data mining: concepts and techniques**. Morgan kaufmann, 2022 (see pages 107, 108).

[97] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. **Unicorn: Runtime provenance-based detector for advanced persistent threats**. *arXiv preprint arXiv:2001.01525* (2020) (see page 51).

[98] Wajih Ul Hassan, Adam Bates, and Daniel Marino. **Tactical provenance analysis for endpoint detection and response systems**. In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, 1172–1189 (see pages 3, 105).

[99] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. **Nodoze: Combatting threat alert fatigue with automated provenance triage**. In: *Network and Distributed Systems Security Symposium*. 2019 (see pages 3, 51, 99, 105, 111).

[100] Taher H Haveliwala. **Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search**. *IEEE transactions on knowledge and data engineering* 15:4 (2003), 784–796 (see page 48).

[101] Wenxuan He, Gaopeng Gou, Cuicui Kang, Chang Liu, Zhen Li, and Gang Xiong. **Malicious Domain Detection via Domain Relationship and Graph Models**. In: *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*. IEEE. 2019, 1–8 (see pages 98, 101).

[102] Raphael Hiesgen, Marcin Nawrocki, Thomas C Schmidt, and Matthias Wählisch. **The race to the vulnerable: Measuring the log4j shell incident**. *arXiv preprint arXiv:2205.02544* (2022) (see page 3).

[103] Waleed Hilal, S Andrew Gadsden, and John Yawney. **A review of anomaly detection techniques and applications in financial fraud**. *Expert Systems with Applications* (2021), 116429 (see page 105).

[104] Victoria Hodge and Jim Austin. **A survey of outlier detection methodologies**. *Artificial intelligence review* 22:2 (2004), 85–126 (see pages 107, 108).

[105] Ralph Holz, Lothar Braun, Nils Kammenhuber, and Georg Carle. **The SSL landscape: a thorough analysis of the x. 509 PKI using active and passive measurements**. In: *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM. 2011, 427–444 (see pages 58, 59).

[106] Jason Hong. **The state of phishing attacks**. *Communications of the ACM* 55:1 (2012), 74–81 (see page 55).

[107] Md Nahid Hossain, Sadegh M Milajerdi, Junao Wang, Birhanu Eshete, Rigel Gjomemo, R Sekar, Scott Stoller, and VN Venkatakrishnan. **{SLEUTH}: Real-time attack scenario reconstruction from {COTS} audit data**. In: *26th {USENIX} Security Symposium ({USENIX} Security 17)*. 2017, 487–504 (see page 51).

[108] **How Many Alerts is Too Many to Handle?** (). URL: https://www2.fireeye.com/StopTheNoise-IDC-Numbers-Game-Special-Report.html (see page 35).

[109] Xin Hu, Jiyong Jang, Marc Ph Stoecklin, Ting Wang, Douglas L Schales, Dhilung Kirat, and Josyula R Rao. **BAYWATCH: robust beaconing detection to identify infected hosts in large-scale enterprise networks**. In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2016, 479–490 (see pages 36, 37).

[110] Yonghong Huang and Paula Greve. **Large scale graph mining for web reputation inference**. In: *Machine Learning for Signal Processing (MLSP), 2015 IEEE 25th International Workshop on*. IEEE. 2015, 1–6 (see pages 60, 101).

[111] Martin Husák, Jana Komárková, Elias Bou-Harb, and Pavel Čeleda. **Survey of attack projection, prediction, and forecasting in cyber security**. *IEEE Communications Surveys & Tutorials* 21:1 (2018), 640–660 (see page 153).

[112] Noora Hyvärinen. *Breaking down the NCSC's top five hacking tools*. Jan. 2018. URL: https://blog.f-secure.com/breaking-down-the-ncscs-top-five-hacking-tools/ (see page 52).

[113] **Improve Threat Detection with Big Data Analytics and AI**. Tech. rep. Databricks (see page 28).

[114] Bill Inmon. **Data Lake Architecture: Designing the Data Lake and avoiding the garbage dump**. Technics publications, 2016 (see page 28).

[115] Luca Invernizzi, Stanislav Miskovic, Ruben Torres, Christopher Kruegel, Sabyasachi Saha, Giovanni Vigna, Sung-Ju Lee, and Marco Mellia. **Nazca: Detecting Malware Distribution in Large-Scale Networks.** In: *Network and Distributed System Security Symposium*. Vol. 14. 2014, 23–26 (see pages 58, 98).

[116] *ISO 31000:2018*. International Organization for Standardization. 2018. URL: https://www.iso.org/standard/65694.html (see page 8).

[117] *ISO 31000:2018*. Mandiant Corporation. 2022. URL: https://github.com/mandiant/OpenIOC_1.1 (see page 14).

[118] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. **Data clustering: a review**. *ACM computing surveys (CSUR)* 31:3 (1999), 264–323 (see page 118).

[119] Vandana P Janeja. **Data analytics for cybersecurity**. Cambridge University Press, 2022 (see pages 3, 15, 19).

[120] Glen Jeh and Jennifer Widom. **SimRank: a measure of structural-context similarity**. In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2002, 538–543 (see page 48).

[121] jivoi. *awesome-osint*. https://github.com/jivoi/awesome-osint. Accessed: 2023-03-04. 2016 (see page 17).

[122] Hyeonseong Jo, Yongjae Lee, and Seungwon Shin. **Vulcan: Automatic extraction and analysis of cyber threat intelligence from unstructured text**. *Computers & Security* 120 (2022), 102763 (see page 153).

[123] Shijoe Jose, D Malathi, Bharath Reddy, and Dorathi Jayaseeli. **A survey on anomaly based host intrusion detection system**. In: *Journal of Physics: Conference Series*. Vol. 1000. 1. IOP Publishing. 2018, 012049 (see pages 3, 15, 105, 147).

[124] Adel K. *awesome-threat-detection*. https://github.com/0x4D31/awesome-threat-detection. Accessed: 2023-03-04. 2016 (see page 17).

[125] Nikos Karampatziakis, Jack W Stokes, Anil Thomas, and Mady Marinescu. **Using file relationships in malware classification**. In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2012, 1–20 (see page 98).

[126] Kelly Kavanagh, Toby Bussa, and Gorka Sadowski. **Magic quadrant for security information and event management**. *Gartner Group Research Note* (2020) (see pages 1, 39).

[127] Yuta Kazato, Yoshihide Nakagawa, and Yuichi Nakatani. **Improving Maliciousness Estimation of Indicator of Compromise Using Graph Convolutional Networks**. In: *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE. 2020, 1–7 (see pages 99, 101).

[128] David Kempe, Jon Kleinberg, and Éva Tardos. **Maximizing the spread of influence through a social network**. In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2003, 137–146 (see page 48).

[129] Eamonn J Keogh and Abdullah Mueen. **Curse of dimensionality.** *Encyclopedia of machine learning and data mining* 2017 (2017), 314–315 (see page 143).

[130] Issa Khalil, Ting Yu, and Bei Guan. **Discovering malicious domains through passive DNS data graph analysis**. In: *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. ACM. 2016, 663–674 (see pages 43, 58, 60, 99–101).

[131] Issa M Khalil, Bei Guan, Mohamed Nabeel, and Ting Yu. **A Domain is only as Good as its Buddies: Detecting Stealthy Malicious Domains via Graph Inference**. In: *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*. ACM. 2018, 330–341 (see pages 99, 100).

[132] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. **Survey of intrusion detection systems: techniques, datasets and challenges**. *Cybersecurity* 2:1 (2019), 1–22 (see pages 105, 133, 147).

[133] Doowon Kim, Bum Jun Kwon, and Tudor Dumitraş. **Certified malware: Measuring breaches of trust in the windows code-signing pki**. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, 1435–1448 (see page 59).

[134] Sujeong Kim, Chanwoong Hwang, and Taejin Lee. **Anomaly based unknown intrusion detection in endpoint environments**. *Electronics* 9:6 (2020), 1022 (see pages 51, 144, 148).

[135] Samuel T King and Peter M Chen. **Backtracking intrusions**. In: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. 2003, 223–236 (see page 51).

[136] Samuel T King, Zhuoqing Morley Mao, Dominic G Lucchetti, and Peter M Chen. **Enriching Intrusion Alerts Through Multi-Host Causality.** In: *NDSS*. 2005 (see page 51).

[137] Mariam Kiran, Peter Murphy, Inder Monga, Jon Dugan, and Sartaj Singh Baveja. **Lambda architecture for cost-effective batch and speed big data processing**. In: *2015 IEEE international conference on big data (big data)*. IEEE. 2015, 2785–2792 (see page 28).

[138] John Klein, Ross Buglak, David Blockow, Troy Wuttke, and Brenton Cooper. **A reference architecture for big data systems in the national security domain**. In: *2016 IEEE/ACM 2nd International Workshop on Big Data Software Engineering (BIGDSE)*. IEEE. 2016, 51–57 (see page 31).

[139] Martin Kleppmann. **Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems**. " O'Reilly Media, Inc.", 2017 (see page 22).

[140] Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert. **Deep learning for classification of malware system call sequences**. In: *Australasian Joint Conference on Artificial Intelligence*. Springer. 2016, 137–149 (see pages 3, 105).

[141] Chris Konrad. *XDR Is Not Just Another Fancy Buzzword*. 2021. URL: https://www.wwt.com/article/xdr-is-not-just-another-fancy-buzzword (visited on 12/07/2021) (see page 21).

[142] Maria Konte, Roberto Perdisci, and Nick Feamster. **Aswatch: An as reputation system to expose bulletproof hosting ases**. In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 2015, 625–638 (see page 58).

[143] Danai Koutra. **Exploring and making sense of large graphs**. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2015 (see page 45).

[144] Danai Koutra, Tai-You Ke, U Kang, Duen Horng Polo Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. **Unifying guilt-by-association approaches: Theorems and fast algorithms**. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2011, 245–260 (see pages 44, 48).

[145] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. **Outlier detection in axis-parallel subspaces of high dimensional data**. In: *Pacific-asia conference on knowledge discovery and data mining*. Springer. 2009, 831–838 (see page 112).

[146] Vijay Krishnan and Rashmi Raj. **Web spam detection with anti-trust rank.** In: *AIRWeb*. Vol. 6. 2006, 37–40 (see page 48).

[147] Marc Kührer, Christian Rossow, and Thorsten Holz. **Paint it black: Evaluating the effectiveness of malware blacklists**. In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2014, 1–21 (see page 90).

[148] Sushil Kumar et al. **An emerging threat Fileless malware: a survey and research challenges**. *Cybersecurity* 3:1 (2020), 1–12 (see pages 14, 105, 140).

[149] Bum Jun Kwon, Jayanta Mondal, Jiyong Jang, Leyla Bilge, and Tudor Dumitraş. **The dropper effect: Insights into malware distribution with downloader graph analytics**. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015, 1118–1129 (see pages 53, 97).

[150] Donghwoon Kwon, Hyunjoo Kim, Jinoh Kim, Sang C Suh, Ikkyun Kim, and Kuinam J Kim. **A survey of deep learning-based network anomaly detection**. *Cluster Computing* 22:1 (2019), 949–961 (see page 146).

[151] Longin Jan Latecki, Aleksandar Lazarevic, and Dragoljub Pokrajac. **Outlier detection with kernel density functions**. In: *International Workshop on Machine Learning and Data Mining in Pattern Recognition*. Springer. 2007, 61–75 (see page 109).

[152] Yang Lei. **Network anomaly traffic detection algorithm based on SVM**. In: *2017 International Conference on Robots & Intelligent System (ICRIS)*. IEEE. 2017, 217–220 (see page 147).

[153] John Leitch. *Process hollowing*. 2013 (see page 14).

[154] Zheng Li, Yue Zhao, Nicola Botta, Cezar Ionescu, and Xiyang Hu. **COPOD: copula-based outlier detection**. In: *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2020, 1118–1123 (see page 147).

[155] Zheng Li, Yue Zhao, Xiyang Hu, Nicola Botta, Cezar Ionescu, and George Chen. **Ecod: Unsupervised outlier detection using empirical cumulative distribution functions**. *IEEE Transactions on Knowledge and Data Engineering* (2022) (see pages 113, 129, 147).

[156] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. **Intrusion detection system: A comprehensive review**. *Journal of Network and Computer Applications* 36:1 (2013), 16–24 (see page 9).

[157] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. **Isolation forest**. In: *2008 eighth ieee international conference on data mining*. IEEE. 2008, 413–422 (see pages 110, 129, 147).

[158] Fucheng Liu, Yu Wen, Dongxue Zhang, Xihe Jiang, Xinyu Xing, and Dan Meng. **Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise**. In: *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 2019, 1777–1794 (see page 153).

[159] Wu Liu, Ping Ren, Ke Liu, and Hai-xin Duan. **Behavior-based malware analysis and detection**. In: *2011 first international workshop on complexity and data mining*. IEEE. 2011, 39–42 (see page 53).

[160] *Living Off The Land Binaries, Scripts and Libraries*. Accessed: 2023-03-04. Github. 2023. URL: https://lolbas-project.github.io/ (see page 141).

[161] *Loopy Belief Propagation*. https://github.com/HewlettPackard/sandpiper. Accessed: 2018-08-10. 2018 (see pages 81, 88).

[162] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. **Beyond blacklists: learning to detect malicious web sites from suspicious URLs**. In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2009, 1245–1254 (see page 56).

[163] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. **Identifying suspicious URLs: an application of large-scale online learning**. In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, 681–688 (see page 56).

[164] J MacQueen. **Some methods for classification and analysis of multivariate observations**. In: *Proc. 5th Berkeley Symposium on Math., Stat., and Prob.* 1965, 281 (see page 110).

[165] Dhia Mahjoub. **Monitoring a fast flux botnet using recursive and passive DNS: A case study**. In: *eCrime Researchers Summit (eCRS), 2013*. IEEE. 2013, 1–9 (see page 58).

[166] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. **Pregel: a system for large-scale graph processing**. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM. 2010, 135–146 (see page 73).

[167] Pratyusa K Manadhata, Sandeep Yadav, Prasad Rao, and William Horne. **Detecting malicious domains via graph inference**. In: *European Symposium on Research in Computer Security*. Springer. 2014, 1–18 (see pages 56, 62, 99–101).

[168] Bernard Marr. **Big data in practice: how 45 successful companies used big data analytics to deliver extraordinary results**. John Wiley & Sons, 2016 (see page 22).

[169] Niels Provos Panayiotis Mavrommatis and Moheeb Abu Rajab Fabian Monrose. **All your iframes point to us**. In: *USENIX Security Symposium*. 2008, 1–16 (see page 57).

[170] Brenda M Michelson. **Event-driven architecture overview**. *Patricia Seybold Group* 2:12 (2006), 10–1571 (see page 28).

[171] TREND MICRO. **FUTURE TENSE: Trend Micro Security Predictions for 2023** () (see pages 3, 105).

[172] *Microsoft 365 Defender: Understand the advanced hunting schema*. Accessed: 2023-03-04. Microsoft. 2023. URL: https://learn.microsoft.com/microsoft-365/security/defender/advanced-hunting-schema-tables (see page 140).

[173] Sadegh M Milajerdi, Birhanu Eshete, Rigel Gjomemo, and VN Venkatakrishnan. **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*. 2019, 1795–1812 (see page 51).

[174] Sadegh M Milajerdi, Rigel Gjomemo, Birhanu Eshete, Ramachandran Sekar, and VN Venkatakrishnan. **Holmes: real-time apt detection through correlation of suspicious information flows**. In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, 1137–1152 (see page 51).

[175] Mishari Al Mishari, Emiliano De Cristofaro, Karim El Defrawy, and Gene Tsudik. **Harvesting SSL certificate data to identify Web-fraud**. *arXiv preprint arXiv:0909.3688* (2009) (see pages 58, 59).

[176] Igor Mishsky, Nurit Gal-Oz, and Ehud Gudes. **A topology based flow model for computing domain reputation**. In: *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer. 2015, 277–292 (see page 101).

[177] *MITRE ATT&CK: Adversarial Tactics, Techniques, and Common Knowledge*. MITRE Corporation. 2023. URL: https://attack.mitre.org/ (visited on 03/01/2023) (see page 14).

[178] MITRE Corporation, ed. *ATT&CK: Standard Application Layer Protocol*. 2018. URL: https://attack.mitre.org/wiki/Technique/T1071 (see page 56).

[179] AC Mora, Y Chen, A Fuchs, A Lane, R Lu, P Manadhata, et al. **Top ten big data security and privacy challenges**. *Cloud Security Alliance* 140 (2012) (see page 31).

[180] Nour Moustafa, Gideon Creech, Elena Sitnikova, and Marwa Keshk. **Collaborative anomaly detection framework for handling big data of cloud computing**. In: *2017 military communications and information systems conference (MilCIS)*. IEEE. 2017, 1–6 (see page 148).

[181] Nour Moustafa and Jill Slay. **UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)**. In: *2015 military communications and information systems conference (MilCIS)*. IEEE. 2015, 1–6 (see page 133).

[182] Raphael Mudge. *Learn Pipe Fitting for all of your Offense Projects*. Ed. by Cobaltstrike by HelpSystems. 2022. URL: https://www.cobaltstrike.com/blog/learn-pipe-fitting-for-all-of-your-offense-projects/ (see page 54).

[183] Joseph Muniz. **The Modern Security Operations Center: The People, Process, and Technology for Operating SOC Services**. Addison-Wesley, 2021 (see pages 1, 105, 112).

[184] Kevin Murphy, Yair Weiss, and Michael I Jordan. **Loopy belief propagation for approximate inference: An empirical study**. *arXiv preprint arXiv:1301.6725* (2013) (see pages 65, 82).

[185] Pejman Najafi, Feng Cheng, and Christoph Meinel. **SIEMA: Bringing Advanced Analytics to Legacy Security Information and Event Management**. In: *Security and Privacy in Communication Networks: 17th EAI International Conference, SecureComm 2021, Virtual Event, September 6–9, 2021, Proceedings, Part I 17*. Springer. 2021, 25–43 (see page 150).

[186] Pejman Najafi, Alexander Mühle, Wenzel Pünter, Feng Cheng, and Christoph Meinel. **MalRank: a measure of maliciousness in SIEM-based knowledge graphs**. In: *Proceedings of the 35th Annual Computer Security Applications Conference*. 2019, 417–429 (see pages 34, 44, 60, 98–100, 105, 114).

[187] Pejman Najafi, Andrey Sapegin, Feng Cheng, and Christoph Meinel. **Guilt-by-association: detecting malicious entities via graph mining**. In: *Security and Privacy in Communication Networks: 13th International Conference, SecureComm 2017, Niagara Falls, ON, Canada, October 22–25, 2017, Proceedings 13*. Springer. 2018, 88–107 (see pages 34, 43, 44, 60, 99, 101).

[188] Antonio Nappa, M Zubair Rafique, and Juan Caballero. **Driving in the cloud: An analysis of drive-by download operations and abuse reporting**. In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2013, 1–20 (see page 55).

[189] Amirreza Niakanlahiji, Mir Mehedi Pritom, Bei-Tseng Chu, and Ehab Al-Shaer. **Predicting Zero-day Malicious IP Addresses**. In: *Proceedings of the 2017 Workshop on Automated Decision Making for Active Cyber Defense*. 2017, 1–6 (see page 58).

[190] Utz Nisslmueller. **LOLBin detection through unsupervised learning: An approach based on explicit featurization of the command line and parent-child relationships**. MA thesis. University of Twente, 2022 (see pages 141, 148).

[191] Philip OKane, Sakir Sezer, and Kieran McLaughlin. **Obfuscation: The hidden malware**. *IEEE Security & Privacy* 9:5 (2011), 41–47 (see page 53).

[192] Talha Ongun, Jack W Stokes, Jonathan Bar Or, Ke Tian, Farid Tajaddodianfar, Joshua Neil, Christian Seifert, Alina Oprea, and John C Platt. **Living-off-the-land command detection using active learning**. In: *Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses*. 2021, 442–455 (see pages 51, 148).

[193] Alina Oprea, Zhou Li, Ting-Fang Yen, Sang H Chin, and Sumayah Alrwais. **Detection of early-stage enterprise infection by mining large-scale log data**. In: *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*. IEEE. 2015, 45–56 (see pages 56, 101).

[194] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. **The PageRank citation ranking: Bringing order to the web.** Tech. rep. Stanford InfoLab, 1999 (see page 47).

[195] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. **Netprobe: a fast and scalable system for fraud detection in online auction networks**. In: *Proceedings of the 16th international conference on World Wide Web*. 2007, 201–210 (see pages 68, 96).

[196] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. **Deep learning for anomaly detection: A review**. *ACM computing surveys (CSUR)* 54:2 (2021), 1–38 (see page 147).

[197] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. **The limitations of deep learning in adversarial settings**. In: *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE. 2016, 372–387 (see page 99).

[198] Emanuel Parzen. **On estimation of a probability density function and mode**. *The annals of mathematical statistics* 33:3 (1962), 1065–1076 (see page 125).

[199] Animesh Patcha and Jung-Min Park. **An overview of anomaly detection techniques: Existing solutions and latest technological trends**. *Computer networks* 51:12 (2007), 3448–3470 (see page 146).

[200] M Pavlidou and G Zioutas. "Kernel density outlier detector." In: *Topics in Nonparametric Statistics*. Springer, 2014, 241–250 (see page 109).

[201] Amruta D Pawar, Prakash N Kalavadekar, and Swapnali N Tambe. **A survey on outlier detection techniques for credit card fraud detection**. *IOSR Journal of Computer Engineering* 16:2 (2014), 44–48 (see page 105).

[202] Judea Pearl. **Probabilistic reasoning in intelligent systems: networks of plausible inference**. Elsevier, 2014 (see page 46).

[203] Kexin Pei, Zhongshu Gu, Brendan Saltaformaggio, Shiqing Ma, Fei Wang, Zhiwei Zhang, Luo Si, Xiangyu Zhang, and Dongyan Xu. **Hercule: Attack story reconstruction via community discovery on correlated log graph**. In: *Proceedings of the 32Nd Annual Conference on Computer Security Applications*. 2016, 583–595 (see page 51).

[204] Chengwei Peng, Xiaochun Yun, Yongzheng Zhang, and Shuhao Li. **MalShoot: Shooting Malicious Domains Through Graph Embedding on Passive DNS Data**. In: *International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Springer. 2018, 488–503 (see pages 98, 101).

[205] Chengwei Peng, Xiaochun Yun, Yongzheng Zhang, Shuhao Li, and Jun Xiao. **Discovering Malicious Domains through Alias-Canonical Graph**. In: *Trustcom/BigDataSE/ICESS, 2017 IEEE*. IEEE. 2017, 225–232 (see pages 57, 101).

[206] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. **Deepwalk: Online learning of social representations**. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, 701–710 (see pages 48, 98).

[207] Tiago Pimentel, Marianne Monteiro, Juliano Viana, Adriano Veloso, and Nivio Ziviani. **A generalized active learning approach for unsupervised anomaly detection**. *stat* 1050 (2018), 23 (see page 110).

[208] Daniel Plohmann, Khaled Yakdan, Michael Klatt, Johannes Bader, and Elmar Gerhards-Padilla. **A comprehensive measurement study of domain generating malware**. In: *25th USENIX Security Symposium (USENIX Security 16)*. 2016, 263–278 (see page 55).

[209] Oskars Podzins and Andrejs Romanovs. **Why siem is irreplaceable in a secure it environment?** In: *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*. IEEE. 2019, 1–5 (see page 1).

[210] J Ronald Prins and Business Unit Cybercrime. *DigiNotar Certificate Authority breach 'Operation Black Tulip'*. Accessed 06-11-2018. 2011 (see page 59).

[211] Danijela D Protić. **Review of KDD Cup '99, NSL-KDD and Kyoto 2006+ datasets**. *Vojnotehnički glasnik/Military Technical Courier* 66:3 (2018), 580–596 (see page 134).

[212] Foster Provost and Tom Fawcett. **Data science and its relationship to big data and data-driven decision making**. *Big data* 1:1 (2013), 51–59 (see page 151).

[213] Santiago Quintero-Bonilla and Angel Martín del Rey. **A new proposal on the advanced persistent threat: A survey**. *Applied Sciences* 10:11 (2020), 3874 (see page 105).

[214] Babak Rahbarinia, Roberto Perdisci, and Manos Antonakakis. **Segugio: Efficient behavior-based tracking of malware-control domains in large ISP networks**. In: *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*. IEEE. 2015, 403–414 (see page 101).

[215] Jagdish Ramakrishnan, Elham Shaabani, Chao Li, and Mátyás A Sustik. **Anomaly detection for an e-commerce pricing system**. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, 1917–1926 (see page 105).

[216] Stephen Ranshous, Shitian Shen, Danai Koutra, Steve Harenberg, Christos Faloutsos, and Nagiza F Samatova. **Anomaly detection in dynamic networks: a survey**. *Wiley Interdisciplinary Reviews: Computational Statistics* 7:3 (2015), 223–247 (see page 146).

[217] Faraz Rasheed and Reda Alhajj. **STNR: A suffix tree based noise resilient algorithm for periodicity detection in time series databases**. *Applied Intelligence* 32:3 (2010), 267–278 (see page 37).

[218] ANM Bazlur Rashid, Mohiuddin Ahmed, Leslie F Sikos, and Paul Haskell-Dowland. **Anomaly detection in cybersecurity datasets via cooperative co-evolution-based feature selection**. *ACM Transactions on Management Information Systems (TMIS)* 13:3 (2022), 1–39 (see page 133).

[219] Carl Rasmussen. **The infinite Gaussian mixture model**. *Advances in neural information processing systems* 12 (1999) (see page 145).

[220] Douglas A Reynolds et al. **Gaussian mixture models.** *Encyclopedia of biometrics* 741:659-663 (2009) (see page 145).

[221] Michael Riley, B Elgin, D Lawrence, and C Matlack. *Missed Alarms and 40 Million Stolen Credit Card Numbers: How Target Blew It. Bloomberg. com, March 17, 2014.* 2016 (see pages 2, 3).

[222] Paul Royal. **Maliciousness in top-ranked alexa domains**. *https://www. barracudanetworks. com/blogs/labsblog* (2012) (see page 87).

[223] Donald Rumsfeld. *Rumsfeld-Matrix*. Most famous statement by the then US Secretary of Defense Donald Rumsfeld , which he made during a press conference in the February 2002. Feb. 2002 (see page 13).

[224] Carl Sabottke, Octavian Suciu, and Tudor Dumitraș. **Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits**. In: *24th {USENIX} Security Symposium ({USENIX} Security 15)*. 2015, 1041–1056 (see page 17).

[225] Gorka Sadowski and Kelly and Bussa Toby andKavanagh. **Critical Capabilities for Security Information and Event Management**. *Gartner Group Research Note* (2020) (see page 11).

[226] Wojciech Samek, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus-Robert Müller. **Explainable AI: interpreting, explaining and visualizing deep learning**. Vol. 11700. Springer Nature, 2019 (see page 111).

[227] Mehmet Hakan Satman. **A new algorithm for detecting outliers in linear regression**. *International Journal of statistics and Probability* 2:3 (2013), 101 (see page 109).

[228] Onur Savas and Julia Deng. **Big data analytics in cybersecurity**. CRC Press, 2017 (see pages 3, 15, 19).

[229] Joshua Saxe and Hillary Sanders. **Malware data science: attack detection and attribution**. No Starch Press, 2018 (see pages 3, 15, 19).

[230] David W Scott. **Multivariate density estimation: theory, practice, and visualization**. John Wiley & Sons, 2015 (see page 125).

[231] David W Scott. **On optimal and data-based histograms**. *Biometrika* 66:3 (1979), 605–610 (see page 126).

[232] IBM Security. **Cost of a Data Breach Report 2022**. *IBM* (2022) (see page 1).

[233] Jonas Herskind Sejr and Anna Schneider-Kamp. **Explainable outlier detection: What, for Whom and Why?** *Machine Learning with Applications* 6 (2021), 100172 (see page 111).

[234] Andrii Shalaginov, Katrin Franke, and Xiongwei Huang. **Malware beaconing detection by mining large-scale dns logs for targeted attack identification**. In: *18th International Conference on Computational Intelligence in Security Information Systems. WASET*. 2016 (see page 37).

[235] Gaganpreet Sharma. **Pros and cons of different sampling techniques**. *International journal of applied research* 3:7 (2017), 749–752 (see page 118).

[236] Shrawan Kumar Sharma and Shiv Kumar. **Comparative analysis of Manhattan and Euclidean distance metrics using A* algorithm**. *J. Res. Eng. Appl. Sci* 1:4 (2016), 196–198 (see pages 109, 145).

[237] Xiaokui Shu, Danfeng Daphne Yao, and Barbara G Ryder. **A formal framework for program anomaly detection**. In: *International Symposium on Recent Advances in Intrusion Detection*. Springer. 2015, 270–292 (see page 105).

[238] Bernard W Silverman. **Density estimation for statistics and data analysis**. Routledge, 2018 (see page 125).

[239] Milivoj Simeonovski, Giancarlo Pellegrino, Christian Rossow, and Michael Backes. **Who controls the internet?: Analyzing global threats using property graph traversals**. In: *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2017, 647–656 (see page 101).

[240] Geeta Singh and Neelu Khare. **A survey of intrusion detection from the perspective of intrusion datasets and machine learning techniques**. *International Journal of Computers and Applications* (2021), 1–11 (see page 133).

[241] Herman Slatman. *Awesome Threat Intelligence*. https://github.com/hslatman/awesome-threat-intelligence. Accessed: 2018-08-10 (see pages 17, 61).

[242] Padhraic Smyth and David Wolpert. **Linearly combining density estimators via stacking**. *Machine Learning* 36:1 (1999), 59–83 (see page 129).

[243] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Masashi Eto, Daisuke Inoue, and Koji Nakao. **Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation**. In: *Proceedings of the first workshop on building analysis datasets and gathering experience returns for security*. 2011, 29–36 (see pages 133, 134).

[244] Alireza Souri and Rahil Hosseini. **A state-of-the-art survey of malware detection approaches using data mining techniques**. *Human-centric Computing and Information Sciences* 8:1 (2018), 3 (see page 97).

[245] Mark Stamp. **Introduction to machine learning with applications in information security**. Chapman and Hall/CRC, 2017 (see pages 79, 137).

[246] James Stewart, Daniel K Clegg, and Saleem Watson. **Calculus: early transcendentals**. Cengage Learning, 2020 (see page 126).

[247] Elizabeth Stinson and John C Mitchell. **Towards Systematic Evaluation of the Evadability of Bot/Botnet Detection Methods.** *WOOT* 8 (2008), 1–9 (see page 43).

[248] *STIX Version 2.1.* OASIS Open, Standard, 2021. 2022. URL: https://docs.oasis-open.org/cti/stix/v2.1/os/stix-v2.1-os.pdf (see page 14).

[249] Amarnag Subramanya and Partha Pratim Talukdar. **Graph-based semi-supervised learning**. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8:4 (2014), 1–125 (see page 48).

[250] Roshan Sumbaly, Jay Kreps, and Sam Shah. **The big data ecosystem at linkedin**. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 2013, 1125–1134 (see pages 21, 31).

[251] Xiaoqing Sun, Mingkai Tong, Jiahai Yang, Liu Xinran, and Liu Heng. **HinDom: A Robust Malicious Domain Detection System based on Heterogeneous Information Network with Transductive Classification**. In: *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. 2019, 399–412 (see pages 99, 101).

[252] Yizhou Sun and Jiawei Han. **Mining heterogeneous information networks: principles and methodologies**. *Synthesis Lectures on Data Mining and Knowledge Discovery* 3:2 (2012), 1–159 (see page 46).

[253] *Sysmon Event ID 1: Process Creation.* Accessed: 2023-03-04. Ultimate IT Security. 2023. URL: https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventid=90001 (see page 140).

[254] Ayman Taha and Ali S Hadi. **Anomaly detection methods for categorical data: A review**. *ACM Computing Surveys (CSUR)* 52:2 (2019), 1–35 (see page 113).

[255] Asghar Tajoddin and Mahdi Abadi. **RAMD: Registry-Based Anomaly Malware Detection Using One-Class Ensemble Classifiers**. en. *Applied Intelligence* 49:7 (July 2019), 2641–2658. ISSN: 1573-7497. DOI: 10.1007/s10489-018-01405-0. URL: https://doi.org/10.1007/s10489-018-01405-0 (visited on 09/17/2020) (see page 54).

[256] Jinita Tamboli and Madhu Shukla. **A survey of outlier detection algorithms for data streams**. In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE. 2016, 3535–3540 (see page 146).

[257] Acar Tamersoy, Kevin Roundy, and Duen Horng Chau. **Guilt by association: large scale malware detection by mining file-relation graphs**. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2014, 1524–1533 (see pages 44, 62, 97, 100).

[258] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. **Line: Large-scale information network embedding**. In: *Proceedings of the 24th international conference on world wide web*. 2015, 1067–1077 (see pages 49, 98, 101).

[259] **The Critical Role of Endpoint Detection and Response**. Tech. rep. Osterman Research, Apr. 2019 (see page 13).

[260] The MITRE Corporation. *Process Injection*. 2021. URL: https://attack.mitre.org/techniques/T1055/ (see page 52).

[261] Ashish Thusoo, Zheng Shao, Suresh Anthony, Dhruba Borthakur, Namit Jain, Joydeep Sen Sarma, Raghotham Murthy, and Hao Liu. **Data warehousing and analytics infrastructure at facebook**. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 2010, 1013–1020 (see pages 21, 31).

[262] Emmanuel Tsukerman. **Machine Learning for Cybersecurity Cookbook: Over 80 recipes on how to implement machine learning algorithms for building security systems using Python**. Packt Publishing Ltd, 2019 (see pages 3, 15, 19).

[263] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. **Survey of machine learning techniques for malware analysis**. *Computers & Security* 81 (2019), 123–147 (see page 97).

[264] Jeroen Van Splunder. **Periodicity detection in network traffic**. *Technical Report, Mathematisch Instituut Universiteit Leiden* (2015) (see page 37).

[265] Verizon. **Data breach investigations report (DBIR)** (2022) (see page 2).

[266] Manfred Vielberth, Fabian Böhm, Ines Fichtinger, and Günther Pernul. **Security operations center: A systematic study and open challenges**. *IEEE Access* 8 (2020), 227756–227779 (see page 1).

[267] Cynthia Wagner, Alexandre Dulaunoy, Gérard Wagener, and Andras Iklody. **Misp: The design and implementation of a collaborative threat intelligence sharing platform**. In: *Proceedings of the 2016 ACM on Workshop on Information Sharing and Collaborative Security*. 2016, 49–56 (see page 14).

[268] Steven Walfish. **A review of statistical outlier methods**. *Pharmaceutical technology* 30:11 (2006), 82 (see page 109).

[269] Hongzhi Wang, Mohamed Jaward Bah, and Mohamed Hammad. **Progress in outlier detection techniques: A survey**. *Ieee Access* 7 (2019), 107964–108000 (see pages 105, 107, 146).

[270] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Chen, Wei Cheng, C Gunter, et al. **You are what you do: Hunting stealthy malware via data provenance analysis**. In: *Symposium on Network and Distributed System Security (NDSS)*. 2020 (see pages 3, 51, 99, 103, 105, 153).

[271] Xianmin Wang, Jing Li, Xiaohui Kuang, Yu-an Tan, and Jin Li. **The security of machine learning in an adversarial setting: A survey**. *Journal of Parallel and Distributed Computing* 130 (2019), 12–23 (see page 43).

[272] James Warren and Nathan Marz. **Big Data: Principles and best practices of scalable realtime data systems**. Simon and Schuster, 2015 (see page 22).

[273] Steven White, Kent Sharkey, David Batchelor, and Michael Satran. *Pipes (Interprocess Communication)*. 2021. URL: https://docs.microsoft.com/en-us/windows/win32/ipc/pipes (see page 54).

[274] Carsten Willems, Thorsten Holz, and Felix Freiling. **Toward automated dynamic malware analysis using cwsandbox**. *IEEE Security & Privacy* 5:2 (2007), 32–39 (see pages 53, 55).

[275] David H Wolpert. **Stacked generalization**. *Neural networks* 5:2 (1992), 241–259 (see page 129).

[276] Wenpu Xing and Ali Ghorbani. **Weighted pagerank algorithm**. In: *Communication Networks and Services Research, 2004. Proceedings. Second Annual Conference on*. IEEE. 2004, 305–314 (see page 48).

[277] Feiyu Xu, Hans Uszkoreit, Yangzhou Du, Wei Fan, Dongyan Zhao, and Jun Zhu. **Explainable AI: A brief survey on history, research areas, approaches and challenges**. In: *CCF international conference on natural language processing and Chinese computing*. Springer. 2019, 563–574 (see page 111).

[278] Jiawei Yang, Susanto Rahardja, and Pasi Fränti. **Outlier detection: how to threshold outlier scores?** In: *Proceedings of the international conference on artificial intelligence, information processing and cloud computing*. 2019, 1–6 (see page 37).

[279] Xingwei Yang, Longin Jan Latecki, and Dragoljub Pokrajac. **Outlier detection with globally optimal exemplar-based GMM**. In: *Proceedings of the 2009 SIAM International Conference on Data Mining*. SIAM. 2009, 145–154 (see page 109).

[280] Danfeng Yao, Xiaokui Shu, Long Cheng, and Salvatore J Stolfo. **Anomaly detection as a service: challenges, advances, and opportunities**. *Synthesis Lectures on Information Security, Privacy, and Trust* 9:3 (2017), 1–173 (see page 105).

[281] Yanfang Ye, Tao Li, Donald Adjeroh, and S Sitharama Iyengar. **A survey on malware detection using data mining techniques**. *ACM Computing Surveys (CSUR)* 50:3 (2017), 1–40 (see pages 14, 97).

[282] Yanfang Ye, Tao Li, Shenghuo Zhu, Weiwei Zhuang, Egemen Tas, Umesh Gupta, and Melih Abdulhayoglu. **Combining file content and file relations for cloud based malware detection**. In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2011, 222–230 (see page 97).

[283] Jonathan S Yedidia, William T Freeman, and Yair Weiss. **Understanding belief propagation and its generalizations**. *Exploring artificial intelligence in the new millennium* 8 (2003), 236–239 (see pages 47, 97, 98).

[284] Ting-Fang Yen, Alina Oprea, Kaan Onarlioglu, Todd Leetham, William Robertson, Ari Juels, and Engin Kirda. **Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks**. In: *Proceedings of the 29th annual computer security applications conference*. 2013, 199–208 (see pages 3, 147).

[285] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xinzheng He. **A deep learning approach for intrusion detection using recurrent neural networks**. *Ieee Access* 5 (2017), 21954–21961 (see page 147).

[286] Pavel Yosifovich, Alex Ionescu, Mark E Russinovich, and David A Solomon. **Windows Internals. System architecture, processes, threads, memory management, and more**. Microsoft Press, 2017 (see page 52).

[287] Rose Yu, Xinran He, and Yan Liu. **Glad: group anomaly detection in social media analysis**. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 10:2 (2015), 1–22 (see page 105).

[288] Ji Zhang. **Advancements of outlier detection: A survey**. *ICST Transactions on Scalable Information Systems* 13:1 (2013), 1–26 (see page 146).

[289] Yue Zhang, Jason I Hong, and Lorrie F Cranor. **Cantina: a content-based approach to detecting phishing web sites**. In: *Proceedings of the 16th international conference on World Wide Web*. ACM. 2007, 639–648 (see page 56).

[290] Yue Zhao and Maciej K Hryniewicki. **XGBOD: improving supervised outlier detection with unsupervised representation learning**. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2018, 1–8 (see page 110).

[291] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. **A survey on unsupervised outlier detection in high-dimensional numerical data**. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 5:5 (2012), 363–387 (see page 146).

[292] Michael Zipperle, Florian Gottwalt, Elizabeth Chang, and Tharam Dillon. **Provenance-based intrusion detection systems: A survey**. *ACM Computing Surveys* 55:7 (2022), 1–36 (see page 51).

[293] Futai Zou, Siyu Zhang, Weixiong Rao, and Ping Yi. **Detecting malware based on DNS graph mining**. *International Journal of Distributed Sensor Networks* 11:10 (2015), 102687 (see pages 60, 99, 101).

List of Publications

Articles in Refereed Journals

[6] **You Are Your Friends: Detecting Malware via Guilt-by-association and Exempt-by-reputation**. *Computers & Security* (2023), 103519. Joint work with Wenzel Puenter, Feng Cheng, and Christoph Meinel.

Articles in Refereed Conference Proceedings

[1] **Detect me if you can: Spam bot detection using inductive representation learning**. In: *Companion Proceedings of The 2019 World Wide Web Conference*. 2019, 148–153. Joint work with Seyed Ali Alhosseini, Raad Bin Tareaf, and Christoph Meinel.

[2] **SIEMA: Bringing Advanced Analytics to Legacy Security Information and Event Management**. In: *Security and Privacy in Communication Networks: 17th EAI International Conference, SecureComm 2021, Virtual Event, September 6–9, 2021, Proceedings, Part I 17*. Springer. 2021, 25–43. Joint work with Feng Cheng and Christoph Meinel.

[3] **NLP-based Entity Behavior Analytics for Malware Detection**. In: *2021 IEEE International Performance, Computing, and Communications Conference (IPCCC)*. IEEE. 2021, 1–5. Joint work with Daniel Koehler, Feng Cheng, and Christoph Meinel.

[4] **A Comprehensive Review of Anomaly Detection in Web Logs**. In: *2022 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT)*. IEEE. 2022, 158–165. Joint work with Mehryar Majd, Seyed Ali Alhosseini, Feng Cheng, and Christoph Meinel.

[5] **MalRank: a measure of maliciousness in SIEM-based knowledge graphs**. In: *Proceedings of the 35th Annual Computer Security Applications Conference*. 2019, 417–429. Joint work with Alexander Mühle, Wenzel Pünter, Feng Cheng, and Christoph Meinel.

[8] **Guilt-by-association: detecting malicious entities via graph mining**. In: *Security and Privacy in Communication Networks: 13th International Conference, SecureComm 2017, Niagara Falls, ON, Canada, October 22–25, 2017, Proceedings 13*. Springer. 2018, 88–107. Joint work with Andrey Sapegin, Feng Cheng, and Christoph Meinel.

Articles in Review

[7] **Context is Key: Detecting Cyber Threats Others Miss**. In: *Review for Computers & Security Journal*. 2024. Joint work with Wenzel Pünter, Feng Cheng, and Christoph Meinel.