FAKULTÄT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Lehrstuhl für Sicherheit in der Informatik

Resizing Threats: Developing Methodologies and Techniques for Large-scale Security Analytics

George Davis Webster III

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 19.08.2019 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 07.03.2020 angenommen.

Acknowledgements

Computer security has long been my passion, and many have fostered my interest and mentored me over the years. Foremost, I would like to express my appreciation to my advisor, Prof. Dr. Claudia Eckert. It has been an honor to have studied under her tutelage. With unwavering enthusiasm, she has supported my pursuits, mentored me in academic research, provided funding to make my efforts, and encouraged me to push through my bouts of frustration. In the same vein, I would like to thank my second advisor Prof. Dr. Jens Grossklags. He was paramount in helping to flesh out my concepts and guide me through the academic process. I always looked forward to my brainstorming session with him over coffee and Skype. They have been an wonderful advisement team and they made this work possible.

To my loving, supportive, and understanding family, thank you. You have been my ultimate cheerleaders and provided me with ample opportunities to explore my passions. In particular, thank you to my remarkable wife Kristi, my bundle of joy Sebastian, and our little monster on the way. You have brought so much happiness, love, and joy into my life.

With special gratitude to the mentors who molded me over the years. Scott Boehm, Colonel Munster, and Lisa Smith, for guiding me in the workplace, providing me with the opportunities to succeed, and pushing me to attain greatness. Dr. Clinton Edwards, Dr. Michael Caloynnides, William Parrish, and Dr. Sawyer, for sparking my interest in academia and being phenomenal professors. To Dr. Michael van Putte for your stimulating discussions and teaching me how to push the bounds of research. And to the countless others that have been there for me over the years.

Thank you to my TUM colleagues and HiWi army for creating an atmosphere conducive for collaboration, friendship, and sparking my passion in teaching. Collectively you have made my stint in academia a wonderful experience that I will always look fondly upon.

With much appreciation to my friends in the security community: Andre Ludwig, Zachary Hanif, Ryan Harris, Bruce Hembree, Jacob Torrey, and the nameless for the great discussions, imparting unique viewpoints on the concepts, and enabling my research. And finally to my colleagues at the USAF, DARPA, FBI, and other agencies who provided a sense of fun, enjoyment, and accomplishment in the work environment.

Abstract

Computer security practices are stuck in a time when the lone analyst was enough to solve the world's problems. Unfortunately, this has not been the case since the Morris worm of 1988. Yet, security analysts are still reliant on one-off tools, stove-piped processes, and immature methods with the end goal being a signature or an Indicator of Compromise for a single event. Sadly, this is a reactive process that takes months, while malicious actors move from victim A to victim B in less than 24 hours. As such, it should be no surprise that major incidents are regular news stories and 20% of companies report a major incident every year.

This dissertation explores how to break the current paradigm in computer security. As such, this work takes the approach that defensive methods must evolve to empower analysts to function across the Intelligence Cycle and pool the collective knowledge and resources of the community together. Specifically, we seek to change how the security community approaches the challenges of investigating malicious activities and generating defensive mitigation actions. In doing so, we provide the technical concepts required and guide how the analytics of malicious activities should be approached. After all, it is the process and the philosophy that matters most. To help guide achieving these goals, we develop an architecture that allows analysts to perform large-scale analysis using any object type. We then expand the architecture to create a new model for sharing and collaboration. This model allows analysts to develop a global perspective and assess threats as a collective whole. To emphasize that the concepts presented in this dissertation can apply to the real world, we then present a working prototype. This prototype has performed complex investigations and enabled active mitigation operations. Finally, we exemplify the power of the approach this dissertation prescribes by demonstrating these methods. In doing so, we reveal a hidden aspect of the PE32 file type and create two triage methods that perform rapid similarity matching and fingerprint the actor's build environment.

Zusammenfassung

Computersicherheitspraktiken stecken noch immer in einer Zeit fest, in der der einsame Analyst ausreichte, um alle Probleme der Welt zu lösen. Unglücklicherweise ist das bereits seit dem Morris Wurm von 1988 nicht mehr der Fall. Trotzdem verlassen sich Sicherheitsanalysten noch immer auf Einmal-Werkzeuge, Trichter-Prozesse und unausgereifte Methoden, um am Ende eine Signatur oder einen Indikator für die Kompromittierung eines simplen Ereignisses zu erhalten. Leider ist dies ein reaktiver Prozess der mitunter Monate dauert, während bösartige Akteure in nur 24 Stunden von Opfer A zu Opfer B springen. Es ist deshalb nicht verwunderlich, dass regelmäßig von größeren Vorfällen in den Nachrichten berichtet wird und jährlich auch von 20 Prozent der Unternehmen gemeldet werden.

Diese Doktorarbeit erforscht, wie man in der Computersicherheit aus diesen Muster ausbrechen kann. Sie verfolgt den Ansatz, dass sich defensive Methoden weiterentwickeln müssen, um Analysten darin zu stärken über den gesamten Informationszyklus hinweg zu funktionieren sowie das kollektive Wissen und die Ressourcen der Community zu bündeln. Wir versuchen insbesondere die Herangehensweise der Sicherheits-Community bei der Untersuchung von bösartigen Aktivitäten und Planen von Vorgehensweisen zur Schadensminderung zu verändern. Hierfür stellen wir die benötigten technischen Konzepte bereit und bieten einen Leitfaden für die Vorgehensweise bei der Analyse bösartiger Aktionen, denn schlussendlich sind die Prozesse und Philosophien noch immer ausschlaggebend. Um diese Ziele zu erreichen, entwickeln wir eine Architektur die es Analysten erlaubt eine groß angelegte Analyse von beliebigen Objekten durchzuführen. Anschließend erweitern wir unsere Architektur um ein neues Modell, mit dem Ziel einen besseren Datenaustausch und eine bessere Zusammenarbeit zu erreichen. Dieses Modell ermöglicht es Analysten eine umfassendere Sichtweise zu entwickeln und Bedrohungen im Kollektiv einzuschätzen. Um deutlich zu machen, dass die in dieser Doktorarbeit vorgestellten Konzepte in der realen Welt Anwendung finden können, stellen wir darauffolgend einen funktionierenden Prototyp vor. Dieser Prototyp übernahm komplexe Untersuchungen und ermöglichte aktiv Schadensminderung in laufenden Operationen. Abschließend veranschaulichen wir die Kraft unserer in dieser Doktorarbeit beschriebenen Methoden, indem wir sie an einem konkreten Fall demonstrieren. Wir zeigen hierzu einen versteckten Aspekt des PE32 Formats auf und schaffen zwei Selektierungsmethoden, die jeweils eine schnelle Ähnlichkeitssuche durchführen und so die Buildumgebung des Akteurs kategorisieren können.

Contents

List of Figures

List of Tables

List of Algorithms

Scientific Publications

George D. Webster, Ryan L. Harris, Zachary D. Hanif, Bruce A. Hembree, Jens Grossklags, and Claudia Eckert. Sharing is Caring: Collaborative Analysis and Real-time Enquiry for Security Analytics. In *Proceedings of the IEEE International Symposium on Recent Advances on Blockchain and Its Applications (BlockchainApp)*. 2018.

Bojan Kolosnjaji, Ghadir Eraisha, George Webster, Apostolis Zarras, and Claudia Eckert. Empowering convolutional networks for malware classification and analysis. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 3838–3845. Anchorage, AK, USA, 2017.

George D. Webster, Bojan Kolosnjaji, Christian von Pentz, Julian Kirsch, Zachary D. Hanif, Apostolis Zarras, and Claudia Eckert. Finding the Needle: A Study of the PE32 Rich Header and Respective Malware Triage. In Michalis Polychronakis and Michael Meier, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings*, volume 10327 LNCS, pages 119–138. Springer International Publishing, Cham, 2017.

Bojan Kolosnjaji, Apostolis Zarras, Tamas Lengyel, George Webster, and Claudia Eckert. Adaptive Semantics-Aware Malware Classification. In Juan Caballero, Urko Zurutuza, and Ricardo J Rodríguez, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings*, volume 9721, pages 419–439. Springer International Publishing, Cham, 2016.

Bojan Kolosnjaji, Apostolis Zarras, George Webster, and Claudia Eckert. Deep learning for classification of malware system call sequences. In Byeong Ho Kang and Quan Bai, editors, *AI 2016: Advances in Artificial Intelligence. AI 2016. Lecture Notes in Computer Science*, volume 9992 LNAI, pages 137–149. Springer International Publishing, Cham, 2016.

George D. Webster, Zachary D. Hanif, Andre L. P. Ludwig, Tamas K. Lengyel, Apostolis Zarras, and Claudia Eckert. SKALD: A Scalable Architecture for Feature Extraction, Multi-user Analysis, and Real-Time Information Sharing. In Matt Bishop and Anderson C A Nascimento, editors, *Information Security: 19th International Conference, ISC 2016, Honolulu, HI, USA, September 3-6, 2016. Proceedings*, pages 231–249. Springer International Publishing, Cham, 2016.

Tamas K. Lengyel, Thomas Kittel, George D. Webster, Jacob Torrey, and Claudia Eckert. Pitfalls of virtual machine introspection on modern hardware. In *1st Workshop on Malware Memory Forensics (MMF)*. New Orleans, Louisiana, USA, 2014.

Tamas K. Lengyel, Steve Maresca, Bryan D. Payne, George D. Webster, Sebastian Vogl, and Aggelos Kiayias. Scalability, fidelity and stealth in the DRAKVUF dynamic malware analysis system. In *Proceedings of the 30th Annual Computer Security Applications Conference on - ACSAC '14*, pages 386–395. New Orleans, Louisiana, USA, 2014.

Industrial Conferences

Space Based Industrial Control Security Evaluation. In *DARPA RSGS Satellite Workshop*, 2017.

George D. Webster, Zach Hanif, and Bojan Kolosnjaji. Spoilers: Effective Malware Triage Using Hidden Fields. In *Microsoft DCC*, 2017.

George D. Webster and Christian von Pentz. Size Matters: Open-Source Framework for Large Scale Analysis. In *Microsoft DCC*, 2017.

George D. Webster and Zach Hanif. From Mole Hills to Mountains: Revealing Rich Header and Malware Triage. In *RSA USA*, 2017.

Challenges in the Analysis and Visualization of Cyber Operations as Scale. In *DARPA Cyber Forum 10*, 2016.

George D. Webster and Julian Kirsch. A Study of the Rich Header and Respective Malware Triage. In *Hacktivity*, 2016.

Zach Hanif, Tamas K. Lengyel, and George D. Webster. Internet-scale file analysis. In *Black Hat USA*, August 2015.

Cyber Information Sharing. In *DARPA Cyber Forum 8*, 2015.

Chapter 1

Introduction

28% of corporations have high confidence they can detect a successful attack
20% of corporations have reported a successful attack
19% of corporations have high confidence they can prevent an attack
19% of corporations have high confidence they can recover from an attack
10s of millions in damage for each corporate attack

Microsoft's 2018 Global Cyber Risk Perception Survey [1]

The state of computer security is not pretty. In 2017, 20% of companies reported that they were victimized by a successful cyber attack [1]. However, only 28% of these companies had high confidence they could even identify if an attack occurred in the first place [1]. To put these percentages in perspective, it takes around 198 days for a company to discover they have been victimized, with some taking upwards of six years [2]. These numbers are alarming. Especially because each attack is costing upwards of 10s of millions of dollars in lost revenue [1]. Outside of financial implications though, the state of computer security becomes downright frightening. 145.5 million individuals had their sensitive personal information stolen in the recent Equifax hack [3]. Even the United States government witnessed the theft of 21 million personnel records, including employees' background investigations [4]. Sadly, the state of security becomes even more petrifying when we consider the risks to the democratic process, vulnerable critical infrastructure, and the increasing use of hacking as a weapon of war [5, 6].

The world is not idle and blind to the state of affairs. Most companies place computer security within their top five priorities and it is estimated that defensive spending will exceed one trillion dollars within the next five years [1, 7]. Governments are also working to combat the threat and have issued numerous legislative acts and executive orders hoping to stop the tsunami of malicious activity [8–11]. Yet, the security communities' defensive methods, tools, and capabilities are still falling short and struggle to counter the threat. Malicious actors are continually improving their techniques, tactics, and methods and it is commonplace to hear of a major computer security incident.

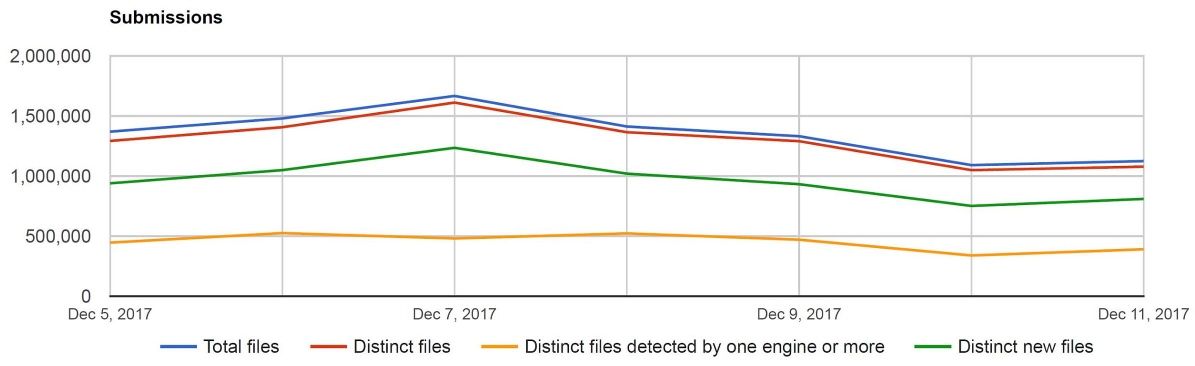

Figure 1.1.: Submissions to VirusTotal for the Week of December 2017.

1.1. Motivation

Malicious actors are no longer composed of curious individuals and misguided youths. These are dedicated teams that command highly sophisticated tools, infrastructures, and possess the resources required to stay ahead of the defender [12, 13]. With respect to just processing the sheer volume of artifacts malicious actors produce, our defensive analytic systems are struggling to cope [14–16]. To put the problem in perspective, in 2012, McAfee received over 100 thousand samples per day [17], yet in 2015, VirusTotal received over a million unique samples in just one day [18]. Unfortunately, as shown in Figure 1.1, over a million samples a day is the new norm and the rate is only growing. Case in point, on December 7th of 2017, VirusTotal received over 1.6 million samples. Unfortunately, this number becomes more overwhelming considering that 97% of those samples were unique and 74% have never been seen before by VirusTotal. Disproportionately, traditional defensive methods are lagging behind. As Figure 1.1 shows, about 30% of these artifacts can be identified as malicious by an Antivirus (AV) vendor. As a result, it should come as no surprise that old school malware analysis tools and methods are struggling to scale to meet this challenge and present a clear picture of criminal activity [14–16]. This is because, while those numbers are impressive, they only account for file-based samples that are already suspicious. However, the reality is that security practitioners need to stay abreast of these daily threats, while also analyzing the considerable volume of benign traffic and keeping track of historic activities.

One of the main culprits behind why defenders cannot keep up is because defensive tools are not designed to be used together, work in an automated fashion, and enable a collaborative environment [14–16]. This is not surprising though, as defensive toolsets grew from the early days of security, when the lone analyst was enough. As stated by MITRE, the reliance on one-off tools causes a situation where analysts often regenerate information and duplicate the work of their peers—a huge waste of time and resources [14]. However, this state of mind is still prevalent. For example, one of the major automation tools, VIPER, is designed for a single user and is expected to be installed on a workstation [19].

While other tools such as Collaborative Research Into Threats (CRITs) and Model-based Analysis of Threat Intelligence Sources (MANTIS) do support a team environment, they fall short of the goal that their acronym implies [14, 20]. One of the reasons is because they struggle to scale and provide the fault-tolerance required to support the volume of data needed to be processed; specifically, CRITs fails with a load of only 50 thousand samples [21]. This is caused by a design that is based on a linear, monolithic, and "tightly coupled" processing pipeline. Another reason these tools have failed is because they are only designed to support the extraction of information from raw data, and in the case of MANTIS, also enabling the sharing of Indicators of Compromise (IOCs). Unfortunately, they do not support other critical aspects of a security investigation, namely how to analyze and make sense of all the information the tools generate or even support the creation of robust IOCs. This model hinders the ability for teams to work together and view the problems from a holistic perspective. As a result, it is no surprise that defenders struggle to present a clear picture of criminal activity, and analysis is limited in its effectiveness.

While security tools are struggling to keep up, another major hindrance in defensive capabilities is caused by the stove-piping of security teams and the inability for them to share and collaborate. This creates multiple problems. One problem is that performing assessments on limited sets of data causes analysts to draw conclusions and make correlations that do not fully encompass the situation [22–24]. Another problem is that each team must perform their own end-to-end investigation. However, investigations are a resource-intensive and time-consuming task. For instance, it takes even a specialized company 54 days to perform an investigation and deploy a mitigation solution [25]. This is a waste of resources and as previously discussed, analytic pipelines are already overwhelmed. Sadly, this stove-piping creates another problem that is more concerning. Because the community is not working together, malicious actors have a large window in which to move between victims using the same methods. To illustrate this problem, even after a well-publicized and major security incident, a year later the malicious actors still used the same methods to steal 100s of millions from hardened banking targets [26–28].

For decades, it has been acknowledged that these stove-pipes need to be broken and that the sharing of security information and collaboration between security practitioners is a necessity. To this end, a gamut of legislative acts, executive orders, academic works, and private sector initiatives have pontificated about the problem and aimed to be the catalyst needed to fix the situation [8–11]. But it has been 30 years since these efforts started. The stove-pipes still exist; sharing and collaboration is a technically complicated, slow, and untrusted task that is regularly impeded by bureaucratic woes [29, 30]. As a result, effective sharing and collaboration is rare. However, even when sharing does occur, the shared artifacts are lacking the context needed to be useful. Thus, researchers can either blindly deploy the rules that were shared or spend considerable effort to rebuild the context behind the shared artifacts by performing their own investigations. However, malicious actors can stay ahead of shared artifacts by using methods such as polymorphism, metamorphism, and Domain Generating Algorithms (DGAs). In addition, performing an investigation is costly and time-consuming [25]. Given this, it is no revelation that it takes around 198 days for a company to discover they have been victimized, with some taking upwards of six years [2].

1.2. Contributions

The lone reverse engineer analyzing a binary or the crafty security researcher dissecting network traffic is no longer adequate. However, a technological tool or novel technique will also not overcome this challenge. A paradigm shift is needed for how defenders tackle the problems of computer security. This requires taking a holistic approach and consolidating the concepts and ideas from multiple domains. In this vein, defensive methods, tools, and techniques need to evolve to face the current and future threats. In particular, defensive methods must empower analysts to function across the Intelligence Cycle and pool the collective knowledge and resources of the community together.

This dissertation does not aim to provide a definitive technical solution or tool that solves all the problems in the security world. Instead, this work seeks to change how the security community approaches the challenges of studying malicious activities and generating defensive mitigation actions. In particular, we seek to provide the technical concepts required and guide how analytics and investigation of malicious activities should be approached. At the end of the day, it is the process and the philosophy that matters most. To help guide the achievement of these goals, this dissertation makes the following major contributions:

- **We present a novel architecture which guides the creation of analytic systems to support the investigations of malicious activities plaguing computer systems, named Skald.**
  At its core, the Skald architecture enables large-scale investigations across the Intelligence Cycle and fosters analytic collaboration. Skald guides the creation of analytic systems that can: (*i*) cope with the growing volume of data, (*ii*) be resilient to system failures, and (*iii*) be flexible enough to incorporate the latest technology trends. Skald provides this by identifying the core categories of analytic activities, based on the Intelligence Cycle, and creates a “loosely coupled” architecture around these concepts. As such, systems designed using Skald are able to receive raw data, extract valuable information from the data, perform assessments across sets of information using advanced analytics, and aid analysts in collectively making a determination. Furthermore, Skald provides a central repository for raw artifacts and analytic results that is segregated according to the Data-Information-Knowledge-Wisdom (DIKW) model. This way defenders can break the stove-pipping between teams and collaborate on a single system. Thus, the Skald architecture reduces the need for each team to reprocess artifacts and also allows each team member to focus on their core area of expertise.

We demonstrate the concepts of Skald by presenting a prototype and conduct extensive experiments. We show that our architecture has a near-linear growth rate and is able to eliminate critical failures when extracting *Information* across millions of PE32 samples. Furthermore, we show major performance gains with the ability to conduct feature extraction at a rate of 3.1 milliseconds per PE32 sample, compared to 2.6 seconds when using existing systems. Finally, we discuss how the Skald architecture provides a platform for analysis on a collective set of security artifacts, thus enabling more accurate analysis of malicious activity and real-time discovery while minimizing the need for redundant processing and thereby reducing analysis time and infrastructure cost. Showing our claims have merit, systems developed based on the Skald architecture have been used to conduct complex investigations, generate academic research, and execute defensive mitigation activities [31–36].

- **We extend the Skald architecture to create a new model for sharing and collaboration, CARE.**
  Skald provides the architectural underpinnings for performing analysis across the Intelligence Cycle. CARE, on the other hand, extends this architecture and furthers the goals of the Intelligence Cycle by encouraging collaboration and empowering the dissemination of the results. We first discuss the recent failures of real-world initiatives for sharing security artifacts and dissect the associated challenges. With an understanding of these challenges, we then extend the Skald architecture to encourage sharing and collaboration across the DIKW model. As such, CARE provides the mechanisms required to perform analytic collaboration with a collective pool of knowledge in near real-time. It does this by creating a cryptographically backed exchange for sharing, derived through a set of common, verifiable extraction methods and analytic algorithms. As a result, CARE provides the foundations for overcoming the privacy and secrecy issues with sharing; it maintains the context and lineage associated with derived information; and it provides a common structure to allow shared artifacts to be easily ingested in analytic pipelines. Furthermore, the cryptographically backed method increases overall trust in the system, while also providing the ledger and infrastructure required to develop a sharing marketplace. In turn, this provides the necessary incentives needed to encourage companies and individuals to share, and it manages the immutable records needed to identify offenders of trust. Thus, CARE allows the sharing and dissemination to occur while providing new opportunities for business models, insurance risk assessments, and government backed incentivisation. While the concepts and prototypes of CARE are in their infancy, the concepts have already gained the attention of major corporations and government agencies.

- **We demonstrate the feasibility and utility of our architecture by creating a working prototype that has aided research and real investigation.**

We present a working prototype that was created to explore and realize the concepts presented in Skald and CARE, in particular, by guiding computer security investigations to break down the process according to the Intelligence Cycle and view the derived artifacts as part of a collective whole. Thus, the prototype fuses together analytic specialties and empowers teams to collaborate. Furthermore, the design of Holmes Processing allows the system to support extremely large datasets, remain flexible to incorporate changes, and be resilient to failures.

While the prototype is academic in nature and not a robust product, it has been used to empower research and perform real investigations against sophisticated and complex actors. Adding validity to our concepts, we have been invited to speak about Holmes Processing and our derived work at numerous highly acclaimed venues, such as Black Hat USA, Microsoft DCC, RSA USA, Hacktivity, and DARPA meetings [2–8].

- **We exemplify the power of this approach by revealing a hidden aspect of the PE32 file type and creating two triage methods.**
  We use the working prototype to perform the first accurate assessment of the Rich Header and detail how to extract its clandestine information and perform advanced analytics. The breakthrough in understanding the Rich Header was achieved by leveraging the power of the prototype to study millions of malware and benign samples. We then present a series of statistical studies and describe two proof-of-concept methods that use only the extracted Rich Header information to generate knowledge about samples. The first method allows for the rapid detection of post-modified and packed binaries through the identification of anomalies. The second method can be used to identify similar malware, different versions of malware, and when malware has been built under different build environments; revealing potentially distinct actors. Furthermore, we are able to perform these operations in near real-time, i.e., in less than 6.73 ms, on commodity hardware across our studied samples.

  Of significant interest, this work is used by the security community to identify potentially malicious activities and perform investigations against complex actors. In one specific case, Kaspersky described how the Rich Header was used to reveal the tools used by the Lazarus Group and perform attribution [37]. However, what is most interesting is that Kaspersky also identified anomalies between the Rich Header information and what is contained in the actual binary; one of our recommendations for future work. As such, Kaspersky concluded that this discrepancy was deliberate and that the identified operation was a potential false flag by another actor—to the author's knowledge, an industry first. In summary, this example demonstrates how the methods presented in this dissertation can provide a major boon for performing computer security investigations.

1.3. Scope

This dissertation focuses on enhancing the capabilities and methods for security analytics. This is a wide topic area and incorporates many areas of specialization. As such, it is important to narrow the scope of the research. Specifically, this work researches the methods needed to change how the security community conducts investigations and analytics. The focus is on the concepts behind investigations of malicious activities and developing a new paradigm for how this work should be approached. While much attention has been paid to log management and Security Information and Event Management (SIEM) systems, this work is targeted towards file-based analysis and the evaluation is focused on the PE32 file type. This focus was chosen because large-scale file analysis is a challenging topic, ripe in potential, and one deserving of more attention. This is especially true due to recent breakthroughs in machine learning and other advanced analytic techniques that can provide valuable insights into the plethora of information that file-based analysis can provide. Additionally, we selected our evaluation around the PE32 file type as this provides access to quality datasets of significant size and enables comparisons to existing tools and methods.

Supplementary to the above, this dissertation is targeted on how to enable advanced analytic capabilities that can empower the security analyst to make better decisions and share the findings with the world. We have purposefully made no effort in designing techniques that center around automatic decisions that determine if a file is benign or malicious.

1.4. Organization

Chapter 2 begins the dissertation by providing the foundational knowledge on which this work is built. In this chapter, we will first discuss relevant analytic models that are commonly used in other domains. This is done to articulate the ideal role of security analysts and develop an understanding for how the divisions of labor are connected. Building upon this, we will then describe the three core categories for extracting *Information* from security artifacts. We follow this discussion by providing a quick overview of the big data analytic techniques used for turning *Information* into *Knowledge*. We then conclude this chapter by introducing the methods used for managing decentralized records and transactions as well as the techniques used for creating large-scale architectures.

The core of the dissertation's work begins with chapter 3. In this chapter, we introduce a new architecture for developing the next generation of analytic systems, SKALD. We describe how SKALD's design enables the development of security analytic platforms that are scalable, flexible, and resilient. In turn, we argue that this provides analysts with the tools needed for performing object-based analysis at real-world speed and scale. Additionally, we argue that the design of SKALD enables analysts to work across the

Intelligence Cycle and empower collaborative human-based analytics.

Chapter 4 builds upon the architecture presented in Chapter 3 to provide a method for enabling effective sharing and collaboration with peers. The chapter first illustrates the major issues surrounding the current sharing paradigm and discusses the critical need to improve the sharing paradigm. We then introduce our model for overcoming these issues, CARE. We argue that the CARE model presents a new way forward in how sharing can occur that alleviates the issues of the current paradigm while providing better opportunities for collaboration. In addition, we show how the CARE model incentivizes sharing and encourages a healthy community by providing the foundations on which an economy can be built, named CAREconomy.

Chapter 5 presents the prototype developed around the concepts prescribed by Skald and CARE. In this chapter, we describe the engineering solutions and explain our implementation decisions.

In Chapter 6, we demonstrate how the prototype empowers advanced analytics. We do this by developing an understanding of the PE32 Rich Header and creating a service that extracts the obfuscated *Information*. We next run the *Service* over 1 million samples, including labeled sets that were donated by the security community. We conclude by presenting our results and describe how our findings can enable rapid triage and perform similarity matching based on the build environment and program function.

Finally, Chapter 7 discusses our findings and highlights the potential opportunities these new methods present for security researchers.

Chapter 2 Fundamentals

"The truth is rarely pure and never simple."

Oscar Wilde

In this chapter, we present a synopsis for the topics that provide the foundation on which this dissertation is built. We begin the discussion by presenting two models for analysis. These models provide the guiding principles that shape this work's architecture. The first model focuses on how to transform raw data into forms that can aid understanding and help make a judgment. The next model presents a popular and widely used end-to-end process for performing an investigation. As we are analyzing large volumes of raw data, the next section discusses the core analytic categories used in security to extract details from raw security artifacts and then the techniques used to study these details as a collective whole. In the succeeding sections, we then shift topics to focus on the relevant techniques and technologies that are used to develop large-scale enterprise class systems. These techniques are leveraged by this work to develop the prototype that evaluates our model. We first present the major two models for concurrence programming, which enable highly parallel computation. Next, we expand beyond a single computer and discuss the architectural patterns for developing distributed systems. Widening the scope of the discussion, we then discuss the technologies for providing secure and trustworthy interactions, without requiring a 3rd party, across multiple users that are geographically distributed.

2.1. Analytic Models

This section introduces two analytic models for the purpose of identifying how analytics and investigations should be performed. A key focus is on the specific steps taken and the types of data each step generates.

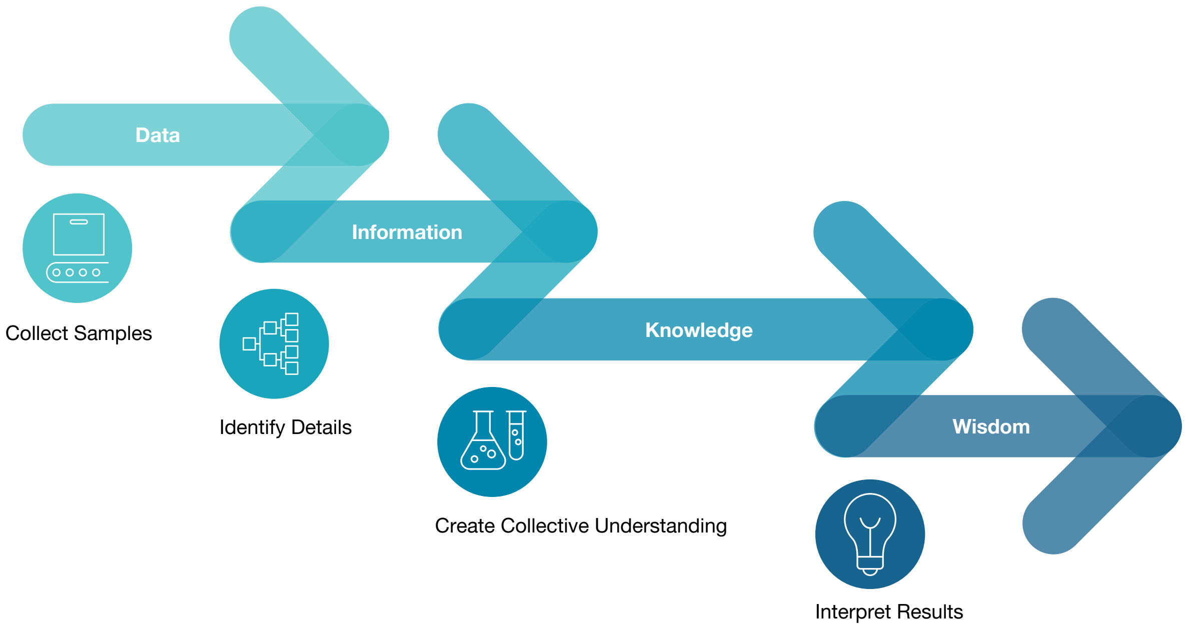

Figure 2.1.: The Layers of the DIKW Model.

2.1.1. Data Information Knowledge Wisdom (DIKW) Model

The DIKW model describes a process for transforming data into value or understanding. While the origin of the DIKW is uncertain, Ackoff receives the most credit for defining its current form and evolving its representation to that of a pyramid [38]. In this form, each layer of the DIKW model builds upon its predecessor until a determination or final judgment can be reached [39]. Furthermore, Ackoff notes that logic can define the transformation between *Data*, *Information*, and *Knowledge*. However, he states that the transformation to *Wisdom* is best left to a human because this step requires a judgment or value determination which is rooted in subtleties such as experience and context.

The definition for each layer of the DIKW model has evolved over the years and is often specific to the context in which the model is used. For the sake of clarity, in this work we leverage the original layer definitions provided by Ackoff and define each layer as follows:

- ***Data*** - A raw object with no further meaning. A PE32, Packet CAPture (PCAP), memory dump, domain name, and an Internet Protocol (IP) address are all examples of objects that are often observed during a computer security investigation.
- ***Information*** - Details obtained by asking who, what, where, when, and “how many” questions of *Data*. These details are often the outcome of static analysis, dynamic analysis, or other extraction method when applied to a piece of *Data*.
- ***Knowledge*** - Organizing a set or a subset of *Information* into useful forms by

asking "how-to" questions. This is performed using statistics, knowledge-based rules, machine learning, and similar analytic techniques across a set or subset of *Information*.

- ***Wisdom*** - The results from the development of understanding based on sets of *Knowledge* and analytic experience.

Under these definitions, the process, as shown in Figure 2.1, for how to make a judgment when presented with a single or set of security objects or samples can begin to evolve. Specifically, an object by itself is *Data*. To begin to work with this *Data*, the first analytic act is to extract details about the *Data* to create a set of *Information*. This extraction often occurs with static and dynamic analyses and help answer questions such as: "what is the size of the *Data*?", "what are the file headers and their respective values for a piece of *Data*?", "what is the control flow of the *Data*?", "what network artifacts does the *Data* produce when executed?", etc. Using multiple sets of *Information*, an analysts can then create *Knowledge* by applying techniques such as statistical modeling, machine learning, and graph theory to identify patterns and draw correlations. For example, these techniques can be used to ask questions such as: "what is the *Information* similar to?", "is the *Information* related to a known threat?", and "how is this *Information* different or similar to a benign sample?". With this *Knowledge*, an analyst is then able to create *Wisdom* by making a judgment based on what has been observed and an analysts experience. For instance, "is the PE32 file malicious?", "should the domain be blocked from the network?", and "who is responsible for this activity??".

In this dissertation, we use the DIKW model to define the different categories of security artifacts and separate the various tasks involved in performing a computer security investigation.

2.1.2. Intelligence Cycle

The Intelligence Cycle was created by the United States Intelligence Community (IC) as a guide for how to perform successful intelligence investigations [40]. This model has since been adopted by multiple military, intelligence, and law enforcement agencies around the world [41–43]. In a similar vein to the DIKW model, the IC model is focused on turning *Data* into useful forms that can aid in making a decision or judgment.

As Figure 2.2 shows, the Intelligence Cycle is broken down into six main parts. These parts flow together in a never ending cyclical fashion and continually feed itself until the overall goal has been reached. Additionally, while seemingly linear, the parts of the Intelligence Cycle can operate asynchronously and can be performed in any order. These parts are as follows:

- **Planning and Direction** - A continual process that sets the strategic goals of the investigation. Additionally, this part monitors the other parts of the Intelligence Cycle to ensure they are performing efficiently and meeting the needs of the

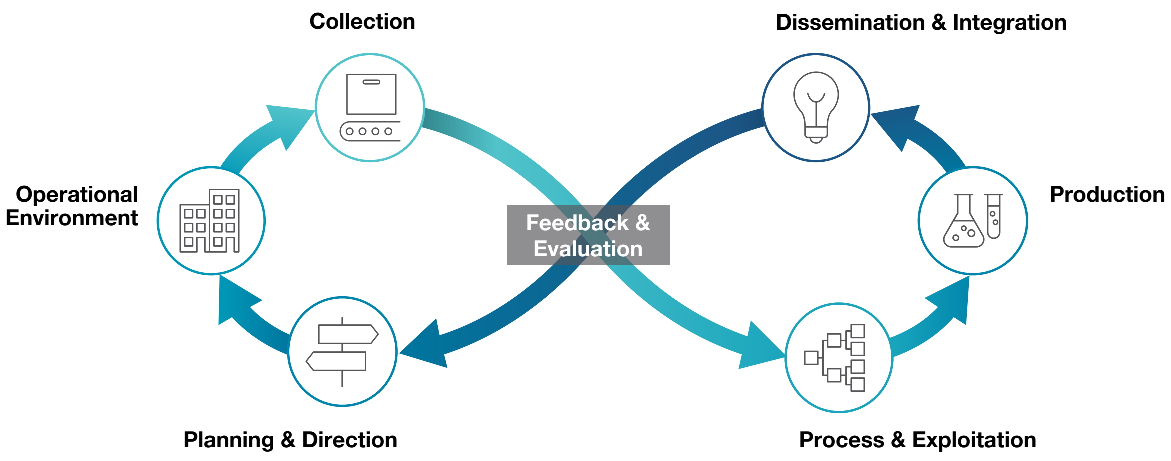

Figure 2.2.: Intelligence Cycle with The Operational Environment.

investigation. This step is an endless cycle because the investigation's goals should be adjusted as a new understanding is reached.

- **Collection** - The process of collecting needed *Data*. This is not restricted to one type of *Data* but should include a collection of different types of *Data* from multiple sources to ensure that an investigation considers multiple angles and takes into consideration a holistic viewpoint.
- **Processing and Exploitation** - The transformation of *Data* into useful forms. At its basic level, this part of the cycle includes converting the *Data* into standardized formats and storing the *Data* where it can be easily utilized. At its more advanced level, this involves interpreting the *Data* and investigating its aspects. Together, this is akin to generating *Information* in the DIKW model.
- **Production** - The analysis of *Information* to derive meaning and understanding. This includes evaluating the validity of new *Information*, analyzing sets of *Information*, integrating what is learned from analysis to identify patterns and build understanding, and drawing conclusions from the analysis. The first three stages in this step are similar to deriving *Knowledge* from *Information*, while the fourth stage is akin to creating *Wisdom* from *Knowledge*.
- **Dissemination and Integration** - The packing of *Knowledge* and *Wisdom* into usable forms that address the needs of the consumer. Additionally, this includes distributing the results and sharing what is learned.
- **Feedback and Evaluation** - A ceaseless process where the holistic effort and parts of the investigation are evaluated and refined. This can be done through informal feedback and discussion as well as formal reviews.

Besides these six parts, we have included an additional part named the **Operational Environment**. This is because security investigations often have an active component that would be outside the scope of a traditional investigation activity. For example, during an investigation an analysts might adjust a firewall configuration or deploy a signature to remove a specific file from the network. As such, in this work we define the **Operational Environment** as a step in the process where the environment is considered, and an active action is taken or contemplated on part of the defender. For instance, to mitigate existing malicious activity or proactively take measures to defend the environment.

In this dissertation, the Intelligence Cycle is used as a foundational guide for the steps needed to perform a successful computer security investigation and drives the structure. This is because the Intelligence Cycle has had wide success in performing complex investigation and producing accurate analysis. Furthermore, computer security investigations are similar to intelligence investigations in that the artifacts available during an investigation are often incomplete, designed to provide false leads, noisy, false positives, or simply not needed. The Intelligence Cycle accounts for these issues during multiple steps. For instance, the Collection part states that multiple pieces of *Data* from multiple different sources should be used to provide a holistic understanding of what is being investigated. Additionally, the Production part states that *Information* should be regularly validated to ensure it should be included or remain included during the investigation. However, the main part of the Intelligence Cycle that counters these issues is the Feedback and Evaluation phase. In this phase, the investigation is regularly assessed to ensure the scope is proper, the analytic focus is addressing the needs of the investigation, and the artifacts being collected and generated are satisfying the needs of the investigation.

As strong parallels can be drawn between an intelligence investigation and that of a computer security investigation, it should come as no surprise that many of the more successful security analytic processes resemble parts of the Intelligence Cycle. As a case in point, Recorded Future's description of the Threat Intelligence Lifecycle is described according to the phases of the Intelligence Cycle [44]. The work in this dissertation Moreover, it is not a coincidence that the DIKW model strongly reflects how the Intelligence Cycle breaks down each part. The major differences are that the DIKW model focuses on the transformation and the associated types of data, while the Intelligence Cycle focuses on the process for performing a successful investigation. Additionally, we will sometimes refer to the Intelligence Cycle as the analytic life cycle. This was done to emphasize the analytic focus of the method.

2.2. Data Based Analytic Methods

The previous section summarizes two models for performing a successful investigation and the various stages for which an object is analyzed to the point where judgment or determination can be made. In this section, we expand upon this discussion by introducing three major categories for extracting *Information* from *Data* with respect to a security investigation: static analysis, dynamic analysis, and 3$^{\text{rd}}$ party information gathering. Furthermore, it is important to note that each category has benefits and drawbacks that should be considered when developing an analytic system. Additionally, and as highlighted in the Intelligence Cycle, no one solution or category is adequate for a full investigation and they should complement each other.

2.2.1. Static Analysis

Static analysis is the act of investigating an object without execution. These methods focus on analyzing the source code, object code, and structure of the *Data* in question. Tools based on static analysis are commonly used to uncover potential vulnerabilities, identify programming errors, extract details about the *Data*, and formally verify the code will behave as expected [45].

One popular use for static analysis is to analyze software during the development cycle. These testing tools; such as Synopsys' Coverity, RogueWave Software's KlocWork, and MathWorks' PolySpace; scan the programming code to identify programming errors, maintainability issues, and detect potential security vulnerabilities before a binary is created [46]. With the adoption of agile development, these methods have seen wide adoption; upwards of 60% of major open source projects [47]. Furthermore, these techniques can be easily incorporated into agile's continuous integration process to ensure newly submitted changes can be safely merged and assessed for security vulnerabilities [48]. For example, the Travis CI and Jenkins build systems include add-ons for Coverity and a slew of other static analysis methods [49, 50]. Interestingly enough, incorporating these techniques with continuous integration can also lessen development overhead. As stated in a Microsoft report, continuous integration with static analysis accounts for a greater than 40% reduction in the overhead cost associated with merging in new code changes [51].

Specific to malware analysis, many triage systems focus on leveraging static analysis for the first assessment of new *Data*. One of the primary reasons for this is because these tools typically have a predictable and short execution time. For example, in our experiences the popular static analysis tool pefile will complete execution in approximately two seconds [52]. However, in this time, pefile will provide a wealth of *Information* that includes the headers and their respective values as well as the structure of a PE32 file. In turn, these headers can provide details on the libraries used to create the PE32, fingerprint the build environment, identify if the binary is cryptographically signed and by whom, and other key details that can be critical in an investigation [53]. Additionally, as we will show in Chapter 6, this PE32-based *Information* can reveal details that *Knowledge*-based analysis can leverage to identify if the binary is packed or obfuscated, allow the ability to identify similar malware, and fingerprint the actors.

Malware authors are not blind to the threat posed by static analysis. In turn, they

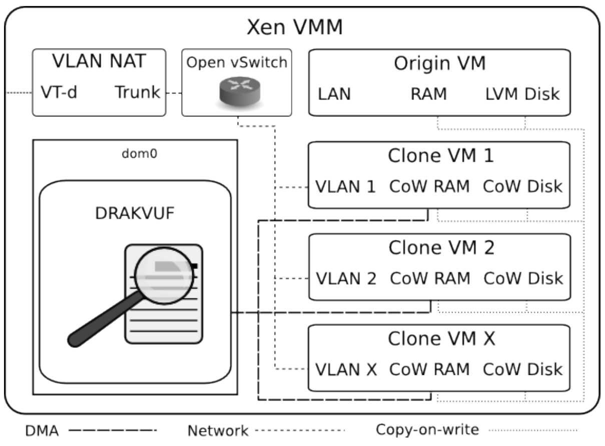

Figure 2.3.: Components for VMI Analysis with Drakvuf

use code obfuscation techniques such as packers, polymorphism, anti-reverse engineering tricks, and "hiding in the noise" to thwart the efforts of static analysis and triage systems [54, 55]. In addition to these efforts, malware authors will also leverage multi-stage attacks in which the originally deployed malware is relatively benign but will download additional components or other malware after the foothold is established [55]. Moreover, this cycle can continue beyond a couple states, as witnessed by the attacks conducted by the Lazarus Group [31]. In sum, these efforts create a cat-and-mouse scenario where each side is continually evolving their techniques to gain an advantage.

2.2.2. Dynamic Analysis

Dynamic analysis is the process of investigating an object while it executes. This is often done to observe the behavior of the object, understand code paths, and identify run-time bugs. Additionally, and of importance to computer security, dynamic analysis can be used to aid the process of unpacking a binary, decrypting code, and capturing the malware used during the different stages of an attack [56, 57].

When performing dynamic analysis to study the behavior of malware, three popular methods are traditionally used: in-guest, VMI, and bare-metal. In-guest approaches, for example Cuckoo Sandbox, leverage a software agent that lives inside the analysis environment [58]. This allows for easy monitoring but has the drawbacks of being easily tampered with and detectable. VMI seeks to overcome the challenges with in-guest based methods by inspecting the execution of an object through the hypervisor that controls the virtual machine; see, for example, the analytic system Drakvuf [59, 60]. As illustrated in Figure 2.3, this provides additional tamper resistance as the inspection element lives outside the environment being inspected and is thus more difficult to detect then in-guest methods. The downside of this approach is that it is challenging to interpret the actual meaning of the symbols passed through the hypervisor, referred to as the semantic gap.

An additional downside is that it is all but impossible to obfuscate that the malware is being executed in a virtual environment and thus potentially being analyzed. Approaches based on bare-metal, such as BareCloud, seek to investigate malware in as clean an environment as possible by not leveraging in-guest agents or virtual environments [61]. However, these systems are costly and difficult to implement. As such, pure bare-metal approaches are most often leveraged only after other methods have detected abnormalities that require further inspection.

While the above three mentioned approaches are arguably the most popular methods used during a computer security investigation, their are many other dynamic analysis methods can can provide valuable *Information* for a computer security investigation. These methods cover a wide range of area such as memory profiling, inspecting cache, fuzzing a programs inputs, generating a binary's call-graph, dynamic packet or protocol analysis, hybrid static analysis approaches such as concolic testing, and many others. However, in this dissertation we focus our research around the main three methods used for studying the behavior of malware during an investigation.

It is important to note that any of these methods have the challenge of dealing with stalling execution or executing benign code paths, known as the halting problem [62, 63]. This is because unlike static analysis, dynamic analysis can only capture what is being executed. This presents a problem as software can have an innumerable number of states and run indefinitely; whereas dynamic analysis platforms require costly resources to run and need to return a result to an analyst in a reasonable amount of time. Regarding malware analysis, this can cause major challenges as a malware program can stall execution or otherwise not exhibit malicious behavior during the allotted time for dynamic analysis. For example, a malware author only needs to incorporate a delay that is greater than 5 minutes to defeat most Cuckoo Sandbox based analysis because Cuckoo Sandbox has a default timeout of 5 minutes.

2.2.3. Third Party Information Gathering

Gathering *Information* from a third party is a method for investigating objects where the investigating process is performed or the *Information* is obtained from an outside source. This can be a powerful method because the analysts can receive details about the object that would otherwise not be obtainable. For example, when confronted with a domain name, the analysts can query current and historic Domain Name System (DNS) records. This can provide *Information* about where the domain resolves to, where it historically resolved, where the servers are hosted, who hosts the servers, how old the domain is, etc. Another example of valuable *Information* that third parties can provide is the detection status and signature labels for numerous AV products, provided by VirusTotal.

2.3. Information-Based Analytic Techniques

In this section, we will introduce three common categories in computer security for transforming *Information* into *Knowledge*: statistical and knowledge based, machine learning, and graph analysis. While other approaches exist, these techniques encompass much of the current research and product landscape. Additionally, similar to the previous section, each method and technique has a valuable role to play in providing the *Knowledge* needed to make an accurate judgment. For example, these techniques can answer questions similar to, "how is this *Information* similar to a previously observed threat?", "how is this *Information* different from a benign sample?", and "how is this *Information* linked to other activity I am observing?" Combining this *Knowledge* together aids the analysts in understanding what is being investigated.

2.3.1. Statistical and Knowledge-based Approaches

Algorithm 1 Yara Signature for APT1's LIGHTDART Malware

```
rule LIGHTDART_APT1 {

meta:
author = "AlienVault Labs"
info = "CommentCrew-threat-apt1"

strings:
$s1 = "ret.log" wide ascii
$s2 = "Microsoft Internet Explorer 6.0" wide ascii
$s3 = "szURL Fail" wide ascii
$s4 = "szURL Successfully" wide ascii
$s5 = "%s&sdate=%04ld-%02ld-%02ld" wide ascii

condition:
all of them
}
```

Statistical analysis and knowledge-based approaches seek to analyze *Information* based on known patterns and rules [64]. At its basic level, signatures are used to match the *Information* from unknown objects to previously observed objects. This can be as simple as identifying a PE32 sample based on a hash—such as MD5 or SHA-256—or identifying suspicious IP traffic based on a port and protocol. More robust forms of signatures take this a step further by seeking to identify and classify samples based on a set of defined textural or binary patterns [65]. This type of signature allows the creation of rules that flag against unique pieces of *Information* contained within a sample versus the overall structure and generic communication methods. For a concrete example, Algorithm 1[1], shows a Yara signature, a popular method used for identifying unique binary or textural patterns, for the APT1 actor's LIGHTDART malware. In this algorithm, AlienVault Labs has defined a rule that states that if all five strings, $s[1-5]$, appear in a binary then label the binary $LIGHTDART\_APT1$.

Not blind to the risk these signatures face, malware authors have adopted polymorphic and metamorphic techniques as well as random communication methods such as DGA. This allows malware to automatically adjust its code structure, execution footprint, and communication pattern on the fly. Thus, increasing the difficulty in writing a static signature [55, 66]. To contend with this new evolution, defenders have created techniques that identify and classify objects based on statistical characteristics. For instance, PEHash is based on the properties of PE32 files and impfuzzy creates a fuzzy hash of the PE32 header import table [67, 68]. Regarding network traffic, methods have been created that model the stochastic behavior of a benign network over time and seek to flag anomalies [64].

The negative of these approaches is that a signature or rule author must first reach a level of understanding for what they are searching for before rules or signatures can be created. This creates a situation where defenses are reactive in nature and defending against new and previously unseen attacks proves challenging. Additionally, as the rise in new malware is accelerating, the ability to keep pace in writing signatures becomes a losing battle. For instance, in December 2017, only 29% of newly submitted samples to VirusTotal were identified by any AV product [69]. Furthermore, if a malware author knows which *Information*, i.e. the string $s[1-5]$ in Algorithm 1 or general statistical characteristics, the rule is flagging against, then the malware authors are able to adjust any new versions of their malware family to bypass the rule.

2.3.2. Machine Learning

Machine learning algorithms fall under two primary categories, supervised learning and unsupervised learning. In supervised learning, the algorithm incorporates a training phase that uses labeled input that states what the output should be. The algorithm then processes the labeled input and tries to create a mathematical model that returns the desired output. This mathematical model is then used with new input to achieve the program's goals. In unsupervised learning, there is no training phase that uses labeled input. Instead, the algorithm identifies the patterns that exist in the input to determine what the output should look like. For example, the algorithm could cluster the input into multiple sets. However, in practice it is not uncommon to see both categories being used together. To illustrate, a program could leverage unsupervised learning to establish which input features or their weights would be best to use with a supervised learning

[1]Source: `https://github.com/AlienVault-Labs/AlienVaultLabs/blob/master/malware_analysis/CommentCrew/apt1.yara`

algorithm.

Security analysts have begun to leverage machine learning in their investigation workflows and the security products they create. This is because machine learning is effective at processing large sets of *Information* to identify patterns and answer how-questions. However, unlike statistical and knowledge-based approaches, machine learning does not require that a specific rule was previously written. For example, these algorithms can help identify if a file is likely to be malicious or related to another sample without requiring a signature [33, 34]. Regarding web traffic, these algorithms can help determine if communication to a domain is malicious without requiring the domain to be listed on a blacklist or whitelist [64, 70]. These techniques are also major boons for investigations, because they allow analysts to identify patterns of activity they might otherwise have missed. For instance, by finding similar samples to one that is being investigated or revealing similar patterns of activity to a known malicious event [35].

While powerful, machine learning is not without its downsides. One of the major challenges in using machine learning is because the effectiveness of the algorithm is based on the *Information* it receives during its training and execution. As a result, if an algorithm was trained with incomplete or improperly labeled artifacts, the result will be inaccurate. Furthermore, if the *Information* provided to the algorithm was obfuscated or incorrect, the algorithm will be non-the-wiser. Unfortunately, in computer security, well-labeled artifacts are difficult to come by and malicious actors are incentivized to make the extraction of *Information* challenging. As another downside, techniques used by malware are not always distinct from benign activity. As a prime example, it is hard to identify if the capability to download and execute a file is for deploying malware packages or for providing regular and legitimate updates. Regarding network traffic, it is not uncommon for a malicious actor to hijack benign network infrastructure for their activities and DGA infrastructure can display strong similarities to content delivery networks. Another major challenge is that it is hard to verify and validate how a machine learning algorithm is functioning. This is especially true with the subset of machine learning called deep learning. As such, it can be difficult to use the results from machine learning algorithms for legal proceedings and to ensure that regulatory violations do not occur during an investigation.

2.3.3. Graph Analysis

Graph analysis is a powerful technique used to understand complex criminal activity. While this technique has multiple names, such as link analysis and network analysis, the concept is the same. The goal of graph analysis is to identify and study the relationships between artifacts. This allows analysts to generate *Knowledge* by organizing how pieces of *Information* fit together to identify connections and infer meaning. Graph analysis is particularly useful for helping to drive the direction of an investigation [43]. Furthermore, deep insights can be brought to light by leveraging graph analysis in conjunction with the Intelligence Cycle. For example, Valdis Krebs unraveled the covert terrorist network that conducted the September 11th terrorist attack through the use of graph analysis with four types of relationships [71].

The ancient technique of graph analysis should be familiar to any fan of Sherlock Holmes, CSI, or any mystery series; for example, when seeing the characters in these series studying pictures posted to a wall with different colored string connecting them. However, the tools used for graph analysis have become an interesting field of research and have evolved under three main generations [72, 73]. Under the first generation, an analyst reviews the information to identify relationships and records them in sets of matrices for each type of relationship [74]. The analyst then uses these matrices to create a graph and the results. While this is effective, it is a time-consuming and tedious task. The second generation focuses on the use of specialized tools, such as Analyst's Notebook, Palantir, and Maltego, to help capture the relationships and graph the results [73]. This has the benefit of allowing the analysts to incorporate multiple relationship types and manipulate the resultant graphs to help in their understanding. The third generation of graph analysis, referred to as Social Network Analysis (SNA), aids the analysts in understanding the relationships by combining other analytic methods. These methods include identifying additional relationships, highlighting sub-groups, revealing patterns between artifacts, and highlighting where there is a need for more *Information* [72, 73].

Investigations into criminal activity are not the only domain of computer security that profits from graph analysis [75]. There has been much recent attention in using graph analysis to identify vulnerability paths in a computer network and the components of a system. This type of graph is referred to as an attack graph. Recent efforts in research have made great strides in how to scale the creation of these graphs and highlighting paths for weaknesses. Similar to SNA, these methods borrow from multiples domains, such as machine learning and formal methods [76–79].

Like all analytic methods, graph analysis has its weaknesses. One area of trouble is caused by trying to graph too much *Information* [80]. This can lead to a graph where everything appears related, or the graph becomes intractable for a human. Another issue is that the SNA methods for revealing relationships and clustering subgroups can be susceptible to adversarial learning, for instance, injecting noise into how artifacts are related [81]. Unfortunately, the obfuscation techniques used by malicious actions naturally have these side effects, such as using hacked infrastructure and Tor nodes.

2.4. Large-Scale Infrastructure Techniques

The previous sections focused on the steps required for performing a computer security investigation. In this section, we discuss the architectural building blocks needed to combine these steps to build a large-scale analytic system. We will first discuss how to develop highly parallelized programs and perform concurrent operations. We then expand these concepts from beyond a single computer and focus on the architectural patterns for developing and organizing distributed systems.

2.4.1. Concurrent Programming Paradigms

While there are multiple models for concurrent programming, the Actor Model and Communicating Sequential Processes (CSP) model have become the dominant methods used in large programs and programming languages for parallel processing and distributed computing. For example, modern implementations of the Actor Model include the Erlang programming language and the Akka toolkit, while the CSP model has been implemented in the Go and Clojure programming languages [82–85]. In these models, the components are "loosely coupled" computational primitives and only communicate with each other by sending a message. This is in contrast to traditional concurrent designs that were based on shared memory and the use of threads, locks, semaphores, and mutexes. This difference allows the development of highly concurrent programs that can take advantage of parallel architectures while also being easy to reason about and mathematically model. In this section, we will briefly discuss the key concepts behind the Actor Model and CSP model.

The Actor Model originated in 1973 as an architecture in which to build efficient and highly parallel artificial intelligence systems [86]. In this model, the Actor is the "fundamental unit of computation" and primary primitive that is composed of three essential elements: processing, storage, and communication [87]. Systems based on the Actor Model are then composed of multiple Actors that are linked together to perform the program's goal. The core design of the Actor Model is centered on the concepts of physics versus traditional mathematical approaches [87]. Because of this, the Actor Model makes the assumption that communication and interaction is inherently asynchronous [88]. To handle this assumption, Actors communicate with each other by only sending messages using internal mailboxes. As Actors can only process one message at a time, the internal mailbox stores newly received messages in a queue until the Actor is available to handle the message. When an Actor receives a message, it is able to processes the message and perform three types of operations: create more Actors, send a message to other Actors it knows about, and designate how the Actor will handle the next message it receives [87]. This design allows highly parallel systems, which can also be easily distributed, as the actors are not reliant on each other for processing, and messaging is decoupled. Furthermore, this allows the creation of concurrent systems that are easy to reason about [89].

The CSP model was created to be a formal algebraic mathematical model for describing concurrent programs [90]. Similar to the Actor Model, each process can only communicate with another process by exchanging messages. However, in the CSP model processes communicate with each other using the concept of channels. Under this concept, processes directly transmit messages to each other. As a process can only perform one function at a time, a process must wait until the receiver is available to accept a message before it can transmit. Additionally, the receiver block will block any other transmission. This eliminates the need in the CSP model to require any form of a queue or mailbox when transmitting messages. Thus, this creates a concurrent system that is synchronous as messages are processed in the order they were sent [91]. Furthermore, this allows the system to be easily mathematically modeled and the management or concern of queue size is no longer an issue.

The key difference between the Actor Model and CSP model comes down to the theory in how messages are transmitted—and unfortunately developer's tribal beliefs. On the one hand, the CSP model is based on direct communication that is synchronous. On the other hand, the Actor Model uses the concept of mailboxes that contain a queue and is asynchronous. Thus, a delay in processing can occur in the CSP model because blocking is required during a message transmission while the Actor Model's queue mitigates these issues. However, the Actor Model's queue can delay the processing of synchronous transitions and make reasoning about the program's logic more difficult. That said, it is possible to define a synchronous system that is asynchronous and an asynchronous system that is synchronous. In fact, current implements of CSP allow a buffer between elements and implementations of the Actor Model allow Actors to have a mailbox queue size of one [84, 85]. Even still, these theoretical differences translate to a higher level of abstraction and decoupling with Actors. This makes designing distributed systems easier with implementations of the Actor Model, while it is easier to control the flow of a programming under CSP.

2.4.2. Service Oriented Architectures

Service Oriented Architectures (SOA) is an architectural approach, and arguably a matter of philosophy, for how to structure enterprise systems [92]. At its core, SOA seeks to organize enterprise architectures around business processes [93]. While the specifics for how SOA is defined are numerous, and often in conflict, this core principal is always the same [94]. That said, Sprott and Wilkes provide the definition that we use in this dissertation. This definition states that SOA is:

> "The policies, practices, frameworks that enable application functionality to be provided and consumed as sets of services published at a granularity relevant to the service consumer. Services can be invoked, published and discovered, and are abstracted away from the implementation using a single, standards-based form of interface." [95]

The key with this definition is two-fold. First, SOA is not just an architecture for designing computer programs but incorporates business structure and practices. Second, SOA is not specific to a piece of technology but a fundamental concept. Providing merit to this definition, this same view is embodied in the works by Papazoglou [96].

Under this definition, SOA seeks to break "tightly coupled" and monolithic programs into multiple "loosely coupled" independent programs. Furthermore, SOA aims to provide a bridge to traditionally siloed applications to allow them to more freely interact [93]. In turn, these programs and applications are called *Service*s and these *Service*s fulfill a business-based operation or need [97]. To create a system, *Service*s' communicate with each other using a common interface. This communication is often supported by an Enterprise Service Bus (ESB) that provides routing, monitoring, standard enforcement, Quality of Service (QoS), and other management tasks [98].

The principals and philosophical design choices imposed by SOA have created many benefits in the deployment and development of systems [97]. One of the main reasons behind these benefits is the "loosely coupled" nature of *Service*s. This is because it allows systems to become more flexible and more easily evolve to changing business needs. Additionally, this benefit applies not only to the software architecture but the development process as well. This is because this level of abstraction between different *Service*s allows teams of developers to work independently and concurrently on different aspects of the system. An additional benefit provided by the "loosely coupled" design is that it empowers distributed computing and scalability [97].

2.4.3. Microservices

Microservices is an increasingly popular architectural style for developing enterprise systems. Perhaps microservices is nothing more than a stripped-down version of SOA. Adding worthiness to this argument, for years Netflix referred to their implementation of the microservices architectural pattern as SOA using "fine grained *Service*s" [99]. However, microservices evolved from the challenges many implementors faced with SOA deployments and SOA's ill-defined nature [100]. This has led to key conceptual differences. Specifically, the role of a *Service*, how a *Service* communicates, and how a *Service* should be deployed. Thus, it can be argued that microservices is a fundamental shift in how to design enterprise systems and merits a clear distinction from the term SOA. Further, microservices have seen wide adoption and popularity in systems that need to scale and take advantage of cloud-based infrastructures.

Akin to SOA, a system in the microservices style is still created from interconnected *Service*s. However, *Service*s in microservices focus on providing a capability instead of fulfilling a business process [100]. The distinction is that a capability provides a single functionality and is self-sufficient [100]. To illustrate this concept in computer security, a monolithic system will perform the end-to-end functionality for analysis. This can include the user interface, analytic methods focused on *Data*, analytic methods focused on *Information*, storage and retrieval, and any other features a customer desires. In SOA, the system will become separated into *Service*s based on business lines. For example, one *Service* will provide the user interface while another *Service* will perform the full scale of DIKW analysis. The microservices based approach breaks the system down further. Looking just at analytic methods for *Data*, one *Service* would orchestrate the execution of additional *Service*s for static analysis. These other *Service*s would then perform a specific static analysis task, such as extracting the headers from a PE32 file.

To enable the self-sufficient nature of this new type of *Service*, *Service*s are designed differently. Specifically, a *Service* must become its own product, able to optimally execute its capability, as "loosely coupled" as possible [101]. In this, a *Service* is expected to receive a tasking, perform its operation, and output the results. Additionally, the ESB is removed to eliminate the “tightly coupled” nature between the ESB and a *Service*. One effect of this is that a *Service* now provides its own generic lightweight communication method that is ideally asynchronous. Such as the REpresentational State Transfer (RESTful), the Advanced Message Queuing Protocol (AMQP), and Protocol Buffers (protobuf). Another effect is that routing and orchestration becomes handled by *Service*s.

This architectural pattern creates an extremely “loosely coupled” system. With this it creates complexity in the design but also immense freedom. This provides major benefits regarding the scalability, flexibility, and resiliency of the system. With respect to scale, a monolithic program can incorporate a load balancer, but the entire program still needs to be replicated. Scaling an SOA system allows more control, but this still forces scaling parts of the system that are not needed. Microservices, on the other hand, allow the scaling of only the specific *Service* that is needed. This saves resources and infrastructure cost. An additional benefit is that the “loosely coupled” nature allows systems to be immensely flexible. For example, a *Service* can be adjusted or changed without requiring changes to other components. Furthermore, this allows *Service*s to be written in whatever language or technology that is best suited for the job or the developers. With respect to resiliency, a system deployed using the microservices architecture often has multiple redundant *Service*s running in tandem while tasking is routed through load balancers or a message bus. This pattern allows redundant *Service*s to seamlessly take on the load of a failed *Service*. Additionally, the “loosely coupled” design further minimizes the risk of failure propagation because of the lack of interdependencies.

There are numerous benefits to a microservices based architecture, but it is not the optimal solution for every use case. It is important to keep in mind that the microservices pattern adds complexity to design, development, deployment, and maintenance of systems. As such, the pros and cons of microservices should be carefully considered before this approach is undertaken. For instance, when scale and cloud-based deployments are not a concern, it can often be more appropriate to select a different architectural pattern or even write a single application using a well-structured monolithic approach.

Lastly, while SOA and microservices seem similar to the Actor Model and CSP, the key difference lies in the scope of the problem these patterns are attempting to solve. Specifically, the Actor Model and the CSP model are used for concurrent programming and primarily serve to empower a single program. On the other hand, SOA and microservices are guides for how to design an overall system that is composed of multiple programs. To help illustrate this concept, you can write a *Service* using the Actor Model and some Actors can be viewed as a *Service*. However, *Service*s focus on achieving a functional goal of the overall system and are inherently independent programs.

2.5. Providing Trust for Decentralized Records and Transactions

In this section, we will discuss the techniques for realizing the Intelligence Cycle's requirement for disseminating what is uncovered during an investigation. To do this, trust in the records and transactions must be established between the exchanging parties. As such, we will present an overview for the key technology for creating a secure and immutable ledger and orchestrating secure contracts and transactions.

2.5.1. Blockchain

The concept of a blockchain was first proposed in 2009 by Satoshi Nakamoto with his paper that introduced Bitcoin [102]. In his ground breaking work, Nakamoto created the concept of a blockchain for creating a verifiable and immutable public ledger for enabling monetary transactions. At its core, blockchains are a form of distributed database that is optimized for security and byzantine fault tolerance. This has seen wide use in achieving a distributed consensus for transactions and solidifying the ownership and transfer of resources.

A blockchain is composed of blocks that are linked together based on the cryptographic hash of the previous blocks [102]. Each block contains a list of transactions that are hashed and encoded as a Merkle tree [103]. Adding blocks to the blockchain is based on distributed consensus. This consensus is reached through two core techniques, Proof of Work (PoW) and Proof of Stake (PoS). To provide a high level overview, PoW adds new blocks to the blockchain through a process where participants solve cryptographic puzzles to identify new blocks, referred to as mining [102, 104]. On the other hand, PoS adds new blocks through a consensus of validity. In this, each miner votes on the validity of a new block with their votes being weighed by how many blocks they own. Unfortunately, blockchain implementations used for consensus are not flawless. For example, under both methods, agreement on consensus by the majority of participants takes precedence. However, this can expose systems to vulnerabilities of trust, known as the 51% attack, where if one party or group gains a majority stake, they can reverse transactions, allow double spending, and halt transactions all together. Furthermore, PoW relies on solving complex computational problems that increase in difficulty. Thus, as more problems are solved, more energy and computing power is required. Eventually, a state occurs where mining becomes exceedingly expensive for a minimal return on the part of the miner.

While exceedingly popular, the blockchain is not an ideal solution for providing a trusted distributed database in all use cases [105, 106]. For example, one issue is that adjustments to the blockchain can impose a taxing latency and cause slow throughput for transactions. Additionally, the computational power required to reach consensus can add additional computational and infrastructure costs. Another issue is that public blockchains often impose transactional fees to incentivize miners which can outweigh the benefits received from the transactions. As such, care must be taken to determine if the advantages of a trusted distributed database without a 3rd party are truly needed in a specific use case. Otherwise, other techniques such as IOTA or a database with trust being provided by a 3rd party might be a more performant and cost-effective approach.

2.5.2. Smart Contract

In 1997, Nick Szabo published the theoretical concept for smart contracts [107]. This theory is based on how traditional business contracts operate and applies this concept to the digital world. At its root, a smart contract provides the means to formalize and secure a relationship, otherwise known as a contract, between parties. However, smart contracts take the concept of traditional paper contracts further. Specifically, while it formalizes the rules and penalties like a traditional paper contract, a smart contract also provides the ability to automatically execute the agreement and enforce the obligations. Furthermore, a smart contract discourages a breach of contract by making it prohibitively costly and keeps the details about the specifics of the contract private in so far as necessary. This is done without requiring a trusted 3rd party.

The first real-world realization of a smart contract system was in the form of an economic framework for peer-to-peer file sharing. This system, called KARMA, used smart contracts to encourage healthy participation by making freeloading (also known as free riding) costly. This system was implemented using a specialized Distributed Hash Table named Pastry and recorded transactions in a secured ledger [108]. This ledger allowed KARMA to create a currency for transactions. Thus, sending a file earned a user "karma" which could be used to purchase other files.

While other smart contract systems have emerged, the next major evolution in implementation is arguably Ethereum. Ethereum creates a smart contract system by providing a Turing-complete language on top of the blockchain [109]. In turn, this allows the creation of any type of contract while providing validation and enforcement through a cryptographically backed, decentralized, and immutable ledger. This is a major leap forward. However, the theoretical concept for smart contracts requires that minimal information about the contract and its execution be publicly available. Unfortunately, traditional implementations of the blockchain transparently store the contract and any transactional details.

To overcome this, recent blockchain models have implemented methods for proving work without requiring complete knowledge of what was exchanged. The research powering this ability goes back decades. For instance, the breakthrough is how to use only a string to prove a statement without requiring prior knowledge and interaction between the parties occurred in 1988 [110]. However, applying these theories at a generic level, while allowing the ability to verify computations, and keeping the verification distinct from the working being proved was finally achieved with Zero Knowledge Succinct Noninteractive ARgument of Knowledge (zk-SNARK) [111]. Simply put, zk-SNARK provides the ability to verify a blockchain without revealing any transactional information. The first major implementation of a blockchain using zk-SNARK was used to add transactional privacy with cryptocurrencies, Zerocash [112]. Finally, bringing these concepts to the realm of smart contracts to provide privacy for the transaction and contract details was realized with HAWK in 2016 and the Byzantium version of Ethereum in 2017 [113, 114]. However, zk-SNARK not only realized the original privacy goals of the smart contract theory, but also had the added benefit of reducing the size of the blockchain. This is because only the information needed to verify the transaction is required on the blockchain. The details behind the contract, the goods exchanged, and the who took part in the transaction are no longer required to exist on the blockchain.

To summarize, current implementation of smart contracts provide the ability to transfer security artifacts between multiple distributed users in a way that preserves trust and has a verifiable immutable ledger. However, the immutable nature of smart contracts and how verification of blockchains work present unique drawbacks that must be appropriately considered when implementing a smart contract system using the blockchain. For instance, one concern is that smart contracts published on the blockchain are immutable. While workarounds exist for changing published contracts, such as Ethereum's $SELFDESTRUCT$; they are challenging to use and often require adequate foresight. As such, care must be taken when publishing a contract to ensure security vulnerabilities are minimized [115].

2.6. Summary

The fundamentals presented in this chapter provide the foundational pillars needed to understand how to perform a computer security investigation and design a supporting system. In the next chapter, we will investigate how to develop an architecture that supports collaborative analysis across the Intelligence Cycle while overcoming the unique challenges computer security presents.

Chapter 3

Developing an Architecture for Large-scale Investigations and Analytics

"I will not follow where the path may lead, but I will go where there is no path, and I will leave a trail."

Muriel Strode

As discussed in Chapter 1, only 28% of companies have high confidence that they can identify malicious activity [1]. Furthermore, it takes around 198 days for a company to discover they have been victimized, with some taking upwards of six years [2]. Yet, it takes less then twenty-four hours for a malicious actor to move from the first victim to the second victim. These statistics paint a destitute situation; however, these statistic should not be shocking. Defensive tools are primarily retrospective, are predominantly made as one-offs, do not work together, and fail at supporting teams of analysts in performing an end-to-end investigation. As a result, companies lack the ability to process artifacts at the scale and fidelity required to contend with malicious activity. Unfortunately, this culminates into a situation where analysts struggle to effectively and accurately identify adversaries' activities and create effective mitigation strategies.

In this chapter, we explore how to begin to shift the current defensive paradigm by introducing an architecture that supports teams of analysts in performing large-scale end-to-end investigations against any type of artifact, named Skald. Skald brings together the concepts discussed in Chapter 2 to guide the creation of analytic systems that can: (*i*) cope with the growing volume of data, (*ii*) be resilient to system failures, and (*iii*) be flexible enough to incorporate the latest technology trends. Skald provides this by identifying the core categories of analytic activities, based on the Intelligence Cycle, and creates a "loosely coupled" architecture around these concepts. As such, systems designed using Skald are able to receive raw *Data*, extract valuable *Information* from the *Data*, perform assessments across sets of *Information* to create *Knowledge*, and aid analysts in collectively making a determination. Furthermore, Skald provides a central repository for artifacts and analytic results that is segregated according to the DIKW model. This way, defenders can break the stove-pipping between teams and

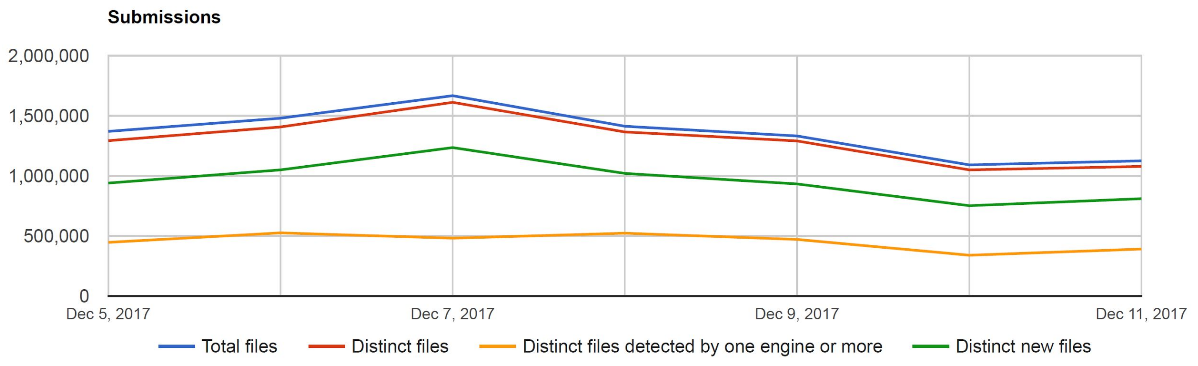

Figure 3.1.: Submissions to VirusTotal for the Week of December 2017.

collaborate on a single system. Thus, systems design based on Skald reduces the need for each team to reprocess artifacts and also allows each team member to focus on their core area of expertise. In chapter 5, we will discuss how we built a working prototype that is based on the Skald architecture.

3.1. Introduction

Malicious actors are no longer composed of curious individuals and misguided youths. These are dedicated teams that command highly sophisticated tools, infrastructures, and possess the resources required to stay ahead of the defender [12, 13]. Unfortunately, malware analysis systems are struggling to meet this challenge and present a clear picture of criminal activity [14–16]. To put the problem in perspective, in 2012, McAfee received over 100 thousand samples per day [17], yet on one day in 2015, VirusTotal received over a million unique samples [18]. Unfortunately, as shown in Figure 3.1, over a million samples a day is the new norm and the rate is only growing. Case in point, on December 7th of 2017, VirusTotal received over 1.6 million samples. Disproportionately, traditional defensive methods, such as AV, are lagging behind. As shown in Figure 3.1, only 29% of those samples could be identified by any AV vendor. We posit that a core reason behind this state of affairs is because the mindset of the defender has not evolved beyond the time when malicious actors were simple and a lone analyst was enough to conduct an investigation. As such, defensive tools have been predominantly made as one-offs, do not work together, and fail at supporting analysis across the Intelligence Cycle. Hence, analysis is limited in accurately identifying the full scale of adversaries' activities and developing effective mitigation strategies. A solution to this problem is twofold: (*i*) how can defenders analyze artifacts and retain a central repository at scale and (*ii*) how can security researchers collaborate in a timely manner without exposing sensitive data and retain essential context.

The volume of malware being released has strained the ability of security teams to analyze the volume of current threats and present a clear picture of criminal activity [14–

16]. With respect to analytic systems, a million samples a day is impressive but it becomes overwhelming when it is considered that 97% of those samples were unique and 74% have never been seen before by VirusTotal. As such, each unique sample is required to be processed and an investigation into the threat performed. However, our defensive systems cannot keep up. For example, it takes an average of 54 days for a specialized company to perform an investigation and deploy a mitigation technique [25]. This is emphasized in the low detection rate that VirusTotal reports for security products when faced with new samples. One of the main culprits behind why defenders cannot keep up is because defensive tools are not designed to be used together in an automated fashion and enable a collaborative environment [14–16]. As stated by MITRE, this causes a situation where analysts often regenerate information and duplicate the work of their peers—a huge waste of time and resources [14]. With respect to collaboration with industry peers, the second problem is that it is difficult to share and work together with security partners in a manner that is timely, retains context, and protects the collection methods. Although, rapid information sharing is an essential element of effective cybersecurity, and will lessen the volume companies need to process, companies are weary of sharing data for fear of tarnishing their business reputation, loosing market share, impairing profits, privacy violations, and revealing internal sources and methods [30, 116, 117]. As a result, it is now common practice to share only with trusted groups large sets of data with minimal context or select post-processed *Wisdom*.

In an attempt to alleviate the burden of analysis, a number of solutions have been developed to help triage *Data* and create a central repository of the collected *Information* [14, 19, 20]. Unfortunately, many of these tools struggle to scale and provide the fault-tolerance required to support the sheer volume of data needed to be processed in part due to the linear, monolithic, and tightly coupled processing pipeline. For instance, analysis tools, like CRITs [14] and MANTIS [20], are not separated from the core Django/Apache system and are executed on the same physical host, while VIPER [19] has been developed for a single user with the intention of being deployed on a workstation. Consequently, when these systems become overloaded, a bottleneck occurs that prevents the analytic tools from executing properly. To make things even worse, when one of the aforementioned tools fails, it is difficult to perform a graceful exit or cleanup, and the system becomes overwhelmed with a load of only a few thousand malware samples. This results in a situation in which analytic tasks cannot be performed quickly and at scale using current technologies and architectures. Furthermore, these tools do not incorporate what is prescribed by the Intelligence Cycle. Specifically, they are only designed to facilitate the extraction of *Information* from *Data*. These systems ignore the other critical aspects of a security investigation, namely how to analyze and make sense of all the *Information* the tools generate. This hinders the ability for teams to work together, view the problems from a holistic perspective, and perform complex investigations.

To counter the problem of lack of collaboration, the security community has developed methods for sharing based on extracted features called IOC. Unfortunately, IOCs only tackle the problem of how to capture features in a format that is understandable among peers. They do not address the larger issues of how to share analytic results in a timely manner, retain the context around the features, and protect sensitive information often attached to raw data. With respect to how to share features in a timely manner and retain the context around the features, IOCs fall short because the creation of IOCs takes time and valuable resources, and the methods used to generate the features are not always uniform and trusted [118]. Hence, the receiving parties spend additional time and resources to reprocess *Data* to generate their own artifacts before they can perform analysis. Regarding the protection of sensitive information attached to raw *Data*, IOCs are often sanitized and only released for sharing after the results are validated [29]. Sadly, this causes a bias in results as performing analysis on limited sets is problematic because it causes analysts to draw conclusions and make correlations that are inaccurate and do not fully encompass the problem [22–24].

In this chapter we present SKALD, a novel architecture to create systems that can perform analysis across the Intelligence Cycle at scale and provide a robust platform for analytic collaboration and the sharing of security artifacts. In essence, SKALD provides the required infrastructure to perform analysis that can: (*i*) cope with the growing volume of artifacts, (*ii*) be resilient to system failures, and (*iii*) be flexible enough to incorporate the latest technology trends. In addition, SKALD takes a new approach in terms of how artifacts are shared by providing a platform that grants analysts' tools access to the entire sets of *Information* without requiring the analysts, and their tools, to have direct access to the raw malware samples or other primary analytic objects, such as a domain name or an IP address. This enables correlations, clustering, and data discovery over the entire set of collected *Information*, while still protecting the raw object, the sources, and the methods used to obtain the *Data*. To this end, we develop an open-source prototype, discussed in detail in chapter 5, and conduct extensive experiments that demonstrate that our architecture has a near linear growth rate and is able to eliminate critical failures when extracting *Information* across millions of PE32 samples. Furthermore, we show major performance gains with the ability to conduct *Information* extraction at a rate of 3.1 milliseconds per PE32 sample, compared to 2.6 seconds when using existing systems. Finally, we discuss how our methodology provides a platform for analysis on a collective set of artifacts, which enables more accurate analysis of malicious activity and real-time discovery while minimizing the need for redundant processing and thereby reducing analysis time and infrastructure cost.

In summary, we make the following main contributions:

- We develop a framework for end-to-end computer security analytics that is scalable, flexible, and resilient, while it demonstrates near linear growth with zero critical errors over millions of samples.

- We display major speed improvements over traditional techniques using only 3.1 ms per sample when utilizing 100 workers.

- We exhibit the ability of our approach to allow partner organizations to submit new raw analysis objects in real-time leveraging the infrastructure from multiple organizations on different continents.
- We show SKALD's capacity to share resultant extracted features and analysis with geographically and organizationally diverse partners who are not sufficiently trusted to have unrestricted access to the raw analysis objects.

3.2. System Overview

SKALD is an architecture for developing analytic platforms for teams working to thwart cyber crime. At its core, SKALD dictates the required structure to perform the steps of the Intelligence Cycle. It scales these actions horizontally, at a near linear rate, across millions of objects, while remaining resilient to failures and providing the necessary flexibility to change analytic methods and core components. SKALD additionally provides the necessary infrastructure to perform advanced analytics to turn *Information* into *Knowledge*, while empowering analysts to retrieve and share information using their preferred tools and scripting interface. Furthermore, SKALD's design allows the sharing of artifacts with partners without requiring the release of *Data*. This is because SKALD segregates analysis according to the DIKW model and SKALD's Access Control Layer (ACL) and intelligent core components allow analytics to be executed across a combination of central instance and in-house replicas. Ergo, artifacts can be easily shared, enabling analysis over a more complete and collective set of data, which overcomes biases caused by informational gaps.

SKALD's achievements are primarily due to the approach of logically abstracting the system into "loosely coupled" themes and core components, as depicted in Figure 3.2; allowing the creation of systems that are scalable, flexible, and resilient. This level of abstraction between system elements is critically missing in widely-used systems such as CRITs, VIPER, and MANTIS. This has lead these systems to become monolithic and "tightly coupled" in design; creating a major hindrance in allowing them to evolve and scale by leveraging distributed computing techniques. However, SKALD – apart from the scalability it offers – enables systems to evolve so they can meet the challenges posed by future cyber criminals by allowing components to be easily exchanged, added, or subtracted. Thus, if a newer, better, or simply different method is discovered, this can be easily incorporated alongside existing methods or simply replace the old ones. This also allows system components to be outsourced to a company, institution, or organization specializing in that work. Finally, this design allows SKALD to orchestrate tasking, which creates an efficient system by substantially reducing the infrastructure and network overhead for transmitting data to and from multiple *Service*s.

The structure of SKALD is based on a microservices design and is composed of three main components: *Transport*, *Planner*s, and *Service*s. As Figure 3.2 illustrates, *Transport*

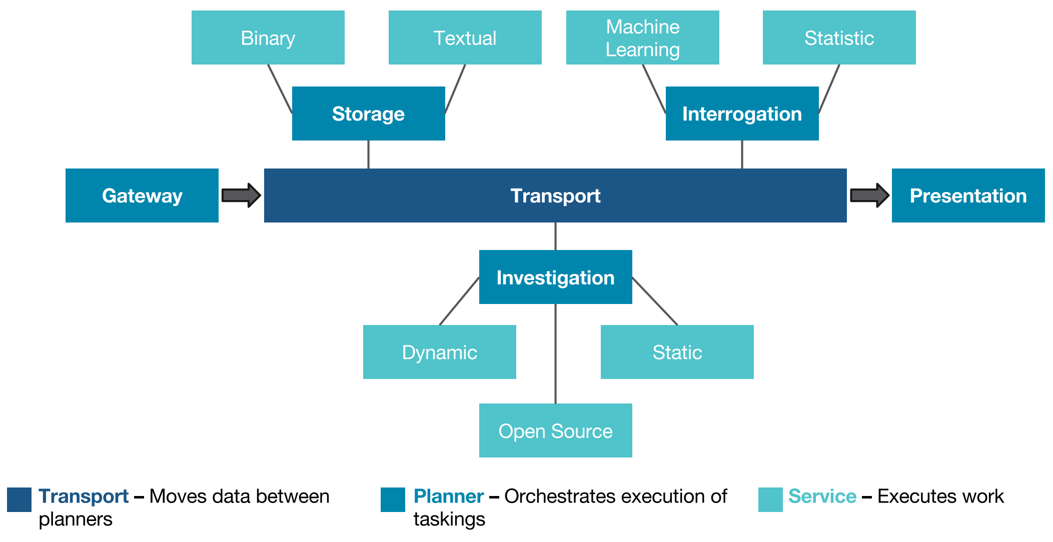

Figure 3.2.: Organization of SKALD's components and core themes.

is the main orchestrator and moves data and tasking to the *Planner*s. Then, *Planner*s allocate infrastructure, enforce security, and oversee the execution of *Service*s. *Service*s in turn perform the requested work and provide the resultant response along with pertinent meta information, such as error messages, back to *Planner*s. This is further described in the following subsections.

3.2.1. Planner

The *Planner*'s primary purpose is to serve as an intelligent orchestrator for *Service*s. At its core, it manages tasking, allocates resources, enforces the ACL, and provides an abstraction between the *Service*s and other aspects of SKALD. The *Planner* also informs the *Transport* what *Service*s are available for tasking and provides status information back to the system core. As previously mentioned, *Planner*s are "loosely coupled" with other parts of SKALD. In this way, they provide flexibility by ensuring that changes to the core aspects of a *Planner* will not affect other parts of the system. This helps to improve resiliency by allowing the *Transport* to delegate tasking to redundant *Planner*s during system failures [97, 119]. Furthermore, this allows the system to horizontally scale by permitting the *Transport* component to instantiate additional *Planner*s under heavy load while also allowing the *Planner*s to schedule the tasking of *Service*s and allocate additional resources on internal servers and cloud infrastructure.

In this section we will describe some of the core aspects of the *Planner* and discuss the five *Planner* themes in SKALD.

3.2.1.1. Service Orchestration and Management

The *Planner*'s primary function is to serve as an orchestration engine for *Service*s. However, this poses a challenge under the microservices paradigm when analyzing computer security artifacts across a distributed architecture. This is because the movement of large objects, such as a file based *Data* artifact, across the network can impose significant network transmission costs. Unfortunately, this is a common occurrence in computer security because of the nature of the artifacts being analyzed. For instance, a *Data* artifacts can be a few megabytes in the case of a PE32 file and extracted *Information* can be in the range of 100s of megs when capturing the Application Programming Interface (API) calls executed during dynamic analysis. To make matters worse, the *Service*s required for computer security investigations often have dependencies that can conflict with parts of the system or are difficult to configure.

SKALD prescribes two approaches for tackling the above challenge. The first approach calls for the isolation of *Service*s through the use of configuration management tools and containers for large *Service*s and leveraging the Actor Model or CSP model for smaller *Service*s. The second approach is to enable the *Planner* to smartly package together to optimize the execution of *Service*s. In effect, this allows a SKALD system to ship the analytic execution to the artifact instead of shipping the artifact to be analyzed.

This provides three core benefits with respect to speed, flexibility, and resiliency [120]. Regarding speed, the packaging of multiple isolated *Service*s under one *Planner* reduces the volume and frequency of large artifacts passing through the network. This in turn is a major advantage with distributed and cloud-based systems because it reduces network latency. Furthermore, packaging *Service*s increases the flexibility of SKALD-based systems by allowing the rapid deployment of new *Service*s without concern for complex dependency management while ensuring discrete versions of *Service*s and configurations. Finally, containers easily allow QoS operations to automatically re-instantiate critically failed *Service*s.

3.2.1.2. Communication with Services

A “loosely coupled” design and flexibility is a critical core tenant to the design of SKALD. This poses a challenge of how to create a generic design for communicating with a *Service* while also allowing the flexibility for how that *Service* operates. To overcome this challenge, SKALD prescribes a two pronged approach.

The first approach focuses on how a *Planner* communicates a task and the results with an independent *Service*. For more complex tasks a *Planner* communicates with *Service*s using Hypertext Transfer Protocol (HTTP) over the Transport Layer Security (TLS) protocol. wever, SKALD does not dictate the format of messages transmitted over HTTP with TLS. That said, our prototype, discussed in chapter 5 provides developers with interfaces for typed JavaScript Object Notation (JSON) message parsing for stronger message safety and a loosely-typed Map data structure when message safety is not a concern. We use this method for three main reasons. First, HTTP is widely understood and is capable of transferring a variety of data types. Second, the wide adoption of the HTTP protocol allows analysts to be able to add new analytics in the language they feel most comfortable this is because HTTP is a widely supported protocol and native to most major languages. Third, HTTP communication allows *Service*s to be deployed either locally or across network partitions. For lighter tasks, SKALD leverages the communication methods inherent to the CSP and the Actor Model.

The second approach in SKALD provides two methods for the *Planner* to communicate objects to *Service*s. For local analytics, the object is delivered via a read-only RAM disk, with fail-over to local disks based on size. This creates a fast and easily-available data-store for analytic file reads across multiple local *Service*s. For external *Service*s, the *Planner* will deliver the object via HTTP. When interacting with *Service*s that do not require a local file, the *Planner* will attach the pertinent meta-data to the tasking message. We select this method for transmitting objects as many existing analytics tools require a local file to be read. As such, this allows existing tools to maintain relevance through leveraging SKALD's ability to distribute and scale workloads. Furthermore, this improves performance since each *Service* does not require a costly network transmission to access an object.

3.2.1.3. Access Control Layer Enforcement

As previously discussed, computer security investigations must support collaborative analysis. However, most current systems do not support a multi-tenancy environment or enable a tasking to restrict how a *Service* executes. This creates an all-or-nothing approach when granting access to an analytic system and exposes a system to potential malicious abuse or data leakage by not allowing the limiting the capabilities of a *Service*.

In SKALD, this issue is overcome by having the *Planner* be is the primary element responsible for managing the ACL, further described in section 3.3.2. As such, the *Planner* is responsible for ensuring that tasking is authorized. The *Planner* also limits potential exposure caused by analyzing an object by enforcing *Service* execution restrictions through the use of ACL meta-tags. This is done by allowing tasking to state that the execution should only run, for example, on internal hardware, without Internet access, and restrict DNS lookups.

3.2.2. Planner Themes

SKALD breaks down *Planner*s into five discrete themes as depicted in Figure 3.2. This segregation is done to prevent the systems from becoming monolithic, enforce the separation of analytic tasks in line with the Intelligence Cycle, and ease the isolation between types of artifacts according to the DIKW model. For instance, the GATEWAY *Planner* receives artifacts and prepares them for processing which is akin to the Intelligence Cycle's Collection step. Similar to the Process and Exploitation step of the Intelligence

Cycle, INVESTIGATION turns *Data* into useful forms through the extraction of *Information*. Moreover, INTERROGATION acts in a similar vein to the Production step by executing analysis across sets of *Information* to generate *Knowledge* and empowering analysts to perform assessments of *Knowledge* to create *Wisdom*. In total, the *Planner* themes are one of the following: GATEWAY, INVESTIGATION, STORAGE, INTERROGATION, and PRESENTATION. In this section, we will explore these themes in detail and provide details on their functionality.

3.2.2.1. Gateway

The GATEWAY's primary purpose is to receive taskings and push them to *Transport*. When tasking is received, the GATEWAY first performs an ACL check to guarantee that the tasking is authorized. If authorized, GATEWAY then ensures taskings are valid, scheme-compliant, and that the pipeline can handle the requested work. The GATEWAY can also automatically assign tasking based on an object type. Together this ensures that pipeline resources are not wasted and provides the first level of system security. How we implemented a GATEWAY is further discussed in chapter 5.3.1.

3.2.2.2. Investigation

This theme is responsible for extracting *Information* from *Data*. When tasking is received, the *Planner* schedules the execution of its *Service*s which are capable of performing static analysis and dynamic analysis as well as gather data from third parties. During scheduling it optimizes the execution of *Service*s by packaging them together and directly providing them the *Data* needed for analysis. As taskings are executed, it performs the two-fold QoS strategy by monitoring the health of *Service*s and validating received results. Additionally, it enforces the ACL and ensures *Service*s adhere to the meta-tags configured restrictions.

Section 5.3.2 discusses how we implemented the above requirements. However, to help illustrate the high level goals of the INVESTIGATION *Planner*, the following describes an ideal execution flow. The *Transport* layer T2 (see Section 3.2.4) submits an object for INVESTIGATION along with a set of taskings and ACL tags. The *Planner* first identifies which *Service*s are available for tasking and configures them according to the tasking request and ACL meta-tags. Next, it either packages an object together with a set of *Service*s for execution on one node, or sends the object to a preexisting node dedicated to a *Service*. The *Planner* will then monitor the health of the *Service* and perform any remediation action as needed. For instance, if a *Service* is unable to gather *Information* from a third-party source due to a query cap, the *Planner* will reschedule the *Service*'s execution once the cap has expired. When the results of a *Service* are received, the *Planner* passes them to the *Transport* layer which then commands the STORAGE *Planner* to archive them. Additionally, if a *Service* returns new objects, the *Planner* will submit the object back to the *Transport* layer with the pertinent ACL tags for storage and tasking to the GATEWAY.

3.2.2.3. Storage

This *Planner*, which implementation is described in section 5.3.3, controls how artifacts are stored and retrieved in SKALD. At its core, it is an abstraction layer for database *Service*s and maintains a central repository of artifacts. The STORAGE *Planner* enforces a standard storage scheme, segregates artifacts according to the DIKW model, and passes the requests to the appropriate database elements, identified by Universally Unique IDentifier v4 (UUIDv4). This enables SKALD to be storage system agnostic and utilize a single or hybrid data storage scheme for resultant artifacts and objects of analysis. The benefit of this approach is that artifacts can be stored in databases optimized for the data type while also easing the inclusion of legacy archives. For example, objects can use Amazon's Simple Storage Service (S3) while the *Information* can be stored in a system optimized for text such as Cassandra [121]. This approach additionally has a major benefit of allowing industry partners to perform in-house replication of selected sets of *Data*, *Information*, and *Knowledge* while enforcing restrictions on more restricted artifacts. For instance, in our prototype, raw objects are stored in restricted datasets hosted by the originator while the extracted *Information* is replicated across all partners. When access to restricted artifacts is required, the *Planner* provides contact information to the requester and once approved the *Planner* will automatically configure access.

The design also enabled the STORAGE *Planner* to perform storage-based optimizations, for example, performing deduplication, compression, and deconflict updates and deletions of temporally sensitive datasets. Finally, the *Planner*s overarching view of the system allows it to automatically perform QoS-based operations such as automatically instantiating additional storage shards and coordinating multi-datacenter replication.

3.2.2.4. Interrogation

The INTERROGATION *Planner* focuses on how to turn the sets of *Information* into *Knowledge* and *Wisdom* in two distinct forms. Aside these forms, the *Planner* is responsible for scaling the number of *Service*s to balance system load, for instance, by instantiating additional Apache Spark workers when required. In the first form, INTERROGATION *Service*s process system artifacts for retrieval through mechanisms such as an API, plugin, or website. This deviates from previous systems by separating the generation of artifacts from the rendering of results. Thus, displaying *Information* is not bound to the extraction or analytic method and can change based on what a user desires. For example, a *Service* can display the VirusTotal score along side Yara rule matches and the results from a clustering algorithm. In the second form, an INTERROGATION *Service* orchestrates the execution of analytics to turn sets of *Information* into *Knowledge*, for example, by clustering objects together with the use of machine learning, identifying relationships through link analysis, and revealing similar malware through PE32 header

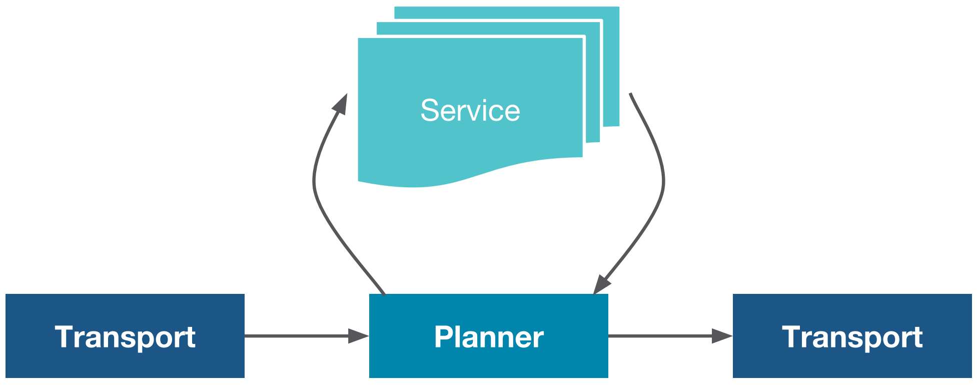

Figure 3.3.: Interaction between the core components of Skald

analysis. To do this, the *Planner* distributes the work load across available *Services* to complete the task at hand. This load balancing allows the integration with mini-batch training techniques to achieve large-scale model training, model generation, and serve as a distribution layer for pre-trained models. Details for how we have implemented this planner are presented in section 5.3.4.

3.2.2.5. Presentation

This *Planner* provides a standard mechanism for externally interacting with stored artifacts that Skald generates. When requests are received, it first ensures that the request is authenticated and scheme-compliant. As this *Planner* theme is the most likely to vary based on individual use cases, we opt to keep its definition as minimal as possible. That being said, the Presentation *Planner* can be imagined as a microservices fog surrounding the datastores queried by the Interrogation *Planner*. How we have implemented this is detailed in section 5.3.5.

3.2.3. Service

Services perform the work being orchestrated by a *Planner*. The key concepts of a Skald *Service* are that they are "loosely coupled" and only interact with their parent *Planner*, as depicted in Figure 3.3. As discussed by Papazoglou et al. [97] and prescribed by the microservices design pattern, this highly decentralized model allows *Services* to be platform-independent and scale as needed. Additionally, this improves fault-tolerance as no *Service* is reliant on the successful execution of another *Service*. Furthermore, the atomic nature of *Services* provides a great level of flexibility by allowing them to be exchanged as new technology emerges and requirements change.

In essence, a *Service* is attached to a *Planner* and performs work associated with that *Planner*'s theme. Examples of various *Services* for each *Planner* are discussed in

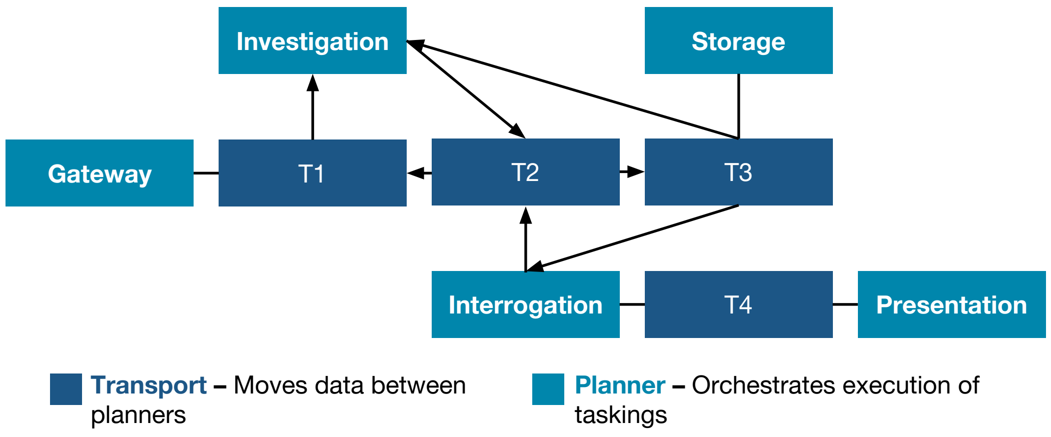
Investigation
Storage
Gateway
T1
T2
T3
Interrogation
T4
Presentation
Transport – Moves data between planners
Planner – Orchestrates execution of taskings

Figure 3.4.: Interaction between the Transports and Planners.

section 5.3. However, for a high-level illustration under the INVESTIGATION *Planner*, *Services* can gather *Information* from VirusTotal, generate a PEHash [67], and perform dynamic analysis with Cuckoo Sandbox [58] and Drakvuf [60]. An INTERROGATION *Service*, on the other hand, will perform an action across a set or subset of *Information* through statistical analysis, machine learning, or by leveraging other data mining and exploration techniques and provide a mechanism to display the results [34].

3.2.4. Transport

Transport's main task is to move data among the *Planner*s. In addition, the *Transport* layer monitors the health of the *Planner*s and performs remediation actions to support QoS. This enables a robust level of resilience by ensuring that requested work is always stored in a queue and that results and taskings are never lost. Additionally, the *Transport* layer improves SKALD's ability to scale by reducing adverse effects of system overloads by allowing the distribution of work across multiple *Planner*s. This is a huge benefit over current and openly available computer security systems because they are only designed to scale vertically.

The *Transport* consists of four main parts (T1, T2, T3, and T4) as Figure 3.4 illustrates. This permits the selection of optimized technology to handle the interaction among *Planner*s. In the following, we introduce these parts and a description of how we implemented these parts of our working prototype can be found in section 5.4.

3.2.4.1. Transport - T1

T1 is focused on moving data to the INVESTIGATION *Planner* for analysis. To do this, T1 is required to perform three primary actions. The first action is to receive tasking from the GATEWAY *Planner* and schedule their transmission to the INVESTIGATION *Planner*. The second is to receive tasking from T2 and submit them to the GATEWAY

Planner for validation. The final action is to monitor the health of the INTERROGATION *Planner* and perform QoS management. When implementing T1, we recommend the utilization of a distributed message broker such as Apache Kafka or RabbitMQ.

3.2.4.2. Transport - T2

T2 is focused on receiving objects and results from the INVESTIGATION and INTERROGATION *Planner*s. To do this, T2 has two primary actions. The first action is to receive artifacts from the INVESTIGATION and INTERROGATION *Planner*s and schedule their submission to the T3 for STORAGE. This is separated from T3 to allow a message queue service to be implemented to help throttle the storage of data during peak loads as storage operations can be costly but are often not time-critical. The secondary action is to receive objects from the INVESTIGATION and INTERROGATION *Planner*s and pass them to T1 for further analysis. Like T1, we recommend the use of a distributed message broker.

3.2.4.3. Transport - T3

T3 is focused on submitting and retrieving data from the STORAGE *Planner*. As such, T3 is responsible for three primary actions: (*i*) providing *Data* from the STORAGE *Planner* directly to the INVESTIGATION and INTERROGATION *Planner*, (*ii*) receive results from T2 and passing the artifacts on to the STORAGE *Planner*, and (*iii*) managing the QoS of the STORAGE *Planner*. We recommend the first two actions to be implemented with no message queues between the STORAGE *Planner* and the databases. This permits database *Service*s to rely on their own optimization frameworks during the retrieval of artifacts. This is because databases are often heavily optimized for retrieval of data and implement their own form of message queues.

3.2.4.4. Transport - T4

T4 handles the exchange of *Knowledge* and *Wisdom* between the INTERROGATION and PRESENTATION *Planner*s. The first action is to provide a conduit for communication between the PRESENTATION and INTERROGATION *Planner*s. The second is to monitor the health of the INVESTIGATION *Planner* and perform QoS management. As the INTERROGATION *Planner* will typically provide *Knowledge* and *Wisdom* through HTTP calls, we recommend implementing T4 as HTTP load balancers.

3.3. System Wide Aspects

In this section, we present system wide aspects of SKALD. We first introduce the QoS strategy we follow and then discuss the ACL requirements.

3.3.1. Quality of Service

Malware authors are incentivized to thwart analysis and as a result the failure of *Service*s should be expected. To counter this, SKALD automatically recovers from issues arising from the execution of *Service*s by leveraging a robust QoS pattern for resilience during analysis. It does this through a two-fold QoS philosophy of monitoring the actual *Service* and the resultant response from a *Service*. Thus, SKALD accounts for the scenario of when the returned work has failed or even if the actual *Service* has failed and accounts for them differently.

To implement the QoS strategy, the *Planner* monitors the health of the *Service* using container status messages and HTTP response codes. The *Planner* then evaluates the availability, response time, and throughput of each attached *Service*. If the evaluation responds by stating the *Service* is operating within normal bounds, the *Planner* will then send the results to the second stage to evaluate the returned work. This allows *Service* authors to specify deep level checks and perform automated remediation actions that are *Service* specific. If a failure occurs at either step, the *Planner* will determine if the *Service* has entered a failed state and perform remediation actions, such as restarting the *Service* container. If the *Service* appears to be healthy, the *Planner* will re-queue the work that has demonstrated failure while saving otherwise successful *Service* results.

The unique aspect of this strategy is that SKALD views the tasking of each *Service* as single elements and processes them individually. When the *Service* or returned work fails, SKALD will only discard failed work as opposed to abandoning combined tasking and queued work. This is a key difference between SKALD and previously proposed systems. For example, the methodology used by systems which rely on the Hadoop Distributed File System (HDFS) and MapReduce model for their data and task distribution, such as BinaryPig [122] and BitShred [123], will cease processing or discard successful results when percentages of work fail. However in SKALD, even if there is a high number of failed jobs, any successful result will be saved and failed work will be reattempted. Furthermore, in the event of pathological failure, SKALD will store the tasking in a separate queue for human intervention. Our evaluation showed this new approach provides significant performance benefits and was outright required when executing large, historical, analytic tasks across computer security *Data* as the *Data* is often designed to confound investigations and cause failures.

SKALD's QoS system also accounts for congestion during times of peak load as well as *Service* shutdown and instantiation. To do this, the *Planner* enforces throttling of *Service*s using a pull-based pattern with an "at least once" message delivery scheme [124]. In this scheme, the *Planner* is aware of the system's current message load and pulls a configurable number of messages from the *Transport* layer. Each requested message is then tracked within the *Planner* as a discrete entity. Upon completion of work, the *Planner* notifies the *Transport* of the work status, pushes the results to the queue, and pulls additional tasking. This allows SKALD to prevent the overburdening of *Planner*s while also ensuring that no work is lost due to component failure. This approach also reduces the chance that a failed *Service* state will replicate to other parts of the system and cause a work stoppage due to being overburdened.

3.3.2. Access Control Layer

The nature of handling unique and sensitive artifacts within SKALD requires the incorporation of a complex ACL system. Unfortunately, the standard ACL model used in computer security only allows for isolation based on the source of the raw *Data* or a user's role. However, the problem is that this does not address access to sensitive capabilities, differentiate *Service*s from users, or separate raw *Data* from associated artifacts such as *Information*, *Knowledge*, and *Wisdom*. This creates an "all or nothing" approach to access. Therefore, the systems cannot be designed to easily allow analysts or even a *Service* to derive *Knowledge* across a collective set of *Information* without granting the analysts access to all sources of artifacts and any sensitive capabilities.

To overcome this, SKALD creates a new ACL model by granting access based on *User*, *Capability*, *Source*, and *Meta-tags*. The *User* defines the users and components of the system. While *Capabilities* map to *Service*s and their derived *Information* and *Knowledge*, *Source* maps the origin of the raw *Data* or artifact. Finally, *Meta-tags* provide *Planner*s with *Service* execution restrictions. For example, these tags can specify that dynamic analysis can only execute without Internet access. While implemented in our prototype, the full definition of the ACL is left to future work.

3.4. Evaluation

To evaluate SKALD, we created an open-source prototype and performed a series of experiments. This prototype is described in detail in chapter 5. However, in this chapter we focus on the ability of the prototype to extract *Information* from *Data*. As such, we use first generation versions of the INVESTIGATION *Planner* and STORAGE *Planner*. This was done to allow the evaluation to compare SKALD with existing systems.

We acknowledge that SKALD focuses on the structure of a system and does not prescribe implementation methods. While this can create varying performance metrics, we feel it is prudent to present a baseline implementation to demonstrate the significant improvements afforded by the SKALD architecture.

Throughout this section, we use the prototype to evaluate the architecture's (*i*) scalability, (*ii*) resiliency, and (*iii*) flexibility.

3.4.1. Experimental Environment

We leveraged three hardware profiles for our evaluation. The first profile was used as a control and deployed CRITs using their recommended setup with the following nodes: (*i*) *Ingest VM*: 2 cores 4GB RAM, (*ii*) *MongoDB VM*: 10 cores 32GB RAM, and (*iii*) *CRITs VM*: 6 cores 32GB RAM. CRITs was selected for our control as it is an industry standard for performing multi-user analytics. Additionally, we made the assumption that CRITs performs similarly to other systems such as MANTIS [20], as the architectures are remarkably similar and Django-based. The second profile deploys our prototype using a similar hardware profile as the control. In this profile, we deployed the prototype in a cloud-based environment using the following nodes: (*i*) *Workers*[1]: three AWS EC2 M3 large instances, (*ii*) *Transport*: One AWS M3 xlarge instance, and (*iii*) *Storage*: One AWS M3 medium instance. Finally, the third profile was used to evaluate the horizontal scalability of the SKALD architecture. In this profile, we used the following nodes: (*i*) *Workers*: 100 AWS EC2 M3 large instances, (*ii*) *Transport*: One AWS M3 xlarge instance, and (*iii*) *Storage*: One AWS M3 medium instance.

In all experiments, we used a diverse set of malicious PE32 samples from VirusShare, Maltrieve, Shadowserver, and private donations. This set of binaries encompasses traditional criminal malware, highly advanced state-sponsored malware, and programs which are not confirmed as malicious but are suspicious.

3.4.2. Scalability

To provide a meaningful evaluation of SKALD's scalability, we studied the ability to ingest PE32 samples and then execute a series of three INVESTIGATION *Service*s. We selected PE32s as this provides a direct mapping to CRITs and our prototype performs at a near identical level when processing domain names, PCAPs, IP Addresses, and other *Data* artifacts. To this end, we did not perform any experiments related to the storage of *Service* results because these experiments would vary depending on the data-store used. Furthermore, we omitted experiments on the INTERROGATION *Service*s as this architectural model is relatively similar to INVESTIGATION and it is difficult to perform clear correlations among other existing systems.

During the first part of our experiments, we used four sets of *Data*: 1000, 5000, 10,000, and 50,000 randomly selected samples. As SKALD is intended to scale to support large datasets, the second part of the experiment ran SKALD with a set of one million samples. In both cases, we pushed the samples into each system through a linear sequence of RESTful calls with no delays between each call. We did this to evaluate the ability of the systems to queue work and simulate batch queues that we regularly encounter during investigation work. Additionally, to ensure that the evaluation was as fair as possible, we leveraged existing CRITs *Service*s. To do this, we added a RESTful wrapper around original CRITs *Service*s to make them compatible with our prototype. These *Service*s gather *Information* from PEInfo, check the file against the VirusTotal private API, and run each sample against 12,431 Yara signatures provided by Yara Exchange [125].

Focusing on up to 50,000 samples, our first finding was that our prototype of SKALD was able to outperforms existing systems, as shown in the first two rows of Table 3.1.

[1]We describe worker to be the combination of a *Planner* and its relevant *Service*s

Framework	1K	5K	10K	50K
CRITs	2.8000	3.1774	3.3781	1.1929
3 Workers	0.0502	0.0558	0.0616	0.1303
100 Workers	0.0032	0.0032	0.0032	0.0025

Table 3.1.: Average time to process samples in seconds.

This in an interesting finding because even with similar hardware, 3 workers, the design was able to process each sample at an average rate of 130.3 ms. In terms of speed, this is a significant improvement over CRITs' average rate of 1.1929 seconds per sample. To put this into context, it took about 28 hours and 37 minutes for CRITs and about 1 hour and 48 minutes for the 3 worker SKALD prototype to process 50,000 samples. Furthermore, these results highlight one of the critical issues plaguing existing systems. The perceived rate increase with CRITs at 50,000 samples was caused by an overload of the CRITs system. When the overload occurred, the operating system began to randomly kill processes before completion, producing a false appearance of speed improvements. However, the tasking was not completed in CRITs and an inspection of the resulting *Information* showed incomplete and corrupted entries. As such, the real speed for CRITs to process 50,000 samples is closer to 44 hours and 8 minutes. The SKALD prototype, with a 3 worker and 100 worker deployment, on the other hand was able to scale to 50,000 samples and performed all the work tasked in that 1 hour and 48 minutes.

Our second finding was that the SKALD prototype was able to scale to 100 workers with minimal effort. To scale, we initiating new workers and the *Transport* layer automatically identified each worker and began to issue tasking. No further action was required on behalf of the operator. Our second finding was that the 100 worker SKALD prototype was able to process 50,000 sample at an approximate rate of 2.5 ms per sample. As the processing rate was similar to the results found when processing the smaller sets, we are confident that the system was able to scale to a large number of workers.

The above experiment established SKALD's performance in processing was superior to existing systems with a similar hardware profile and that SKALD was able to horizontally scale its infrastructure to 100 workers. To push the SKALD prototype further, we then studied the prototype's ability to scale to meet the demand of extremely large sample sets. In this experiment we used 100 workers to process the same type of work as above but with a set of 1 million samples. Unfortunately, a direct comparison with CRITs was not possible because, try as we might[2], the CRITs system could not maintain stability beyond 50,000 samples. Accordingly, we did not use the 3 worker configuration because a direct comparison of CRITs was not possible.

We found that the SKALD prototype was able to scale to meet this extremely large

[2]We submitted numerous patches that improved stability to the CRITs' upstream repository.

sample size. With 1 million samples, the prototype performed at an approximate rate of 3.4 ms per sample using 100 workers. This rate was similar to what we found in the previous experiment when processing smaller sample sets and this gives us confidence that the system was able to handle the large work load. To perform a comparison of sorts with CRITs and put this number into perspective, the SKALD 100 worker prototype was able to process 1 million samples in about 56 minutes while CRITs, at an average processing rate of 3.174 seconds per sample [3], would have taken approximately 36.7 days.

In both experiments we noticed that the processing rates increased and decreased depending on the size of taskings. For instance, at 50,000 samples the three worker showed significant slowdown while the 100 worker showed a speed increase. Furthermore, the speed increase reverted back to the norm at 1 million samples. This increasing and decreasing rate is an interesting finding but one that could be explained through the normalization of the implemented scheduler in the *Transport* layer. Through investigation we were able to see that the large tasking size required disk IO which caused a slowdown over time that affected both configurations equally. However, the implementation of the *Transport* layer's caching mechanism had a delay before it activated. As a result, the faster pull rate with 100 workers only allowed the scheduler to begin to cache the next task for processing at around 10,000 samples where the slower pull rate with 3 workers allowed the caching mechanism to active when less samples were processed. This was shown with the near linear rate of 3.2ms per sample up to about 10,000 samples with 100 workers and a steady rate of improvement leading up to 2.5ms per sample at 50,000 samples. The rate then reverted to the norm due to the requirement of disk IO which was caused by the large tasking size. As such, the variations in speed were caused by the *Transport* layer's implementation having a caching delay and requiring disk reads because of the large tasking size. This, however, was a constant rate and we are confident that SKALD performs at a similar rate irrespective of the volume of samples.

3.4.3. Resilience

We executed two experiments to study the resiliency of SKALD for handling troublesome work and continuing processing even with failed components. For the first experiment, we wrote 26KB worth of random bytes across 20% of the samples. This was done by creating a script that used */dev/random* to randomly selected 20% of the bytes of a sample and rewrote these bytes with the values of */dev/random*. This was done to generate *Data* objects that would potentially confuse and fail analytic tasks. As such, this allowed us to evaluate SKALD's ability to cope with failed, long-running, and troublesome work that is typically found during security investigations because of the desire of malicious actors to thwart defensive investigations. For the second experiment, we studied how SKALD-based systems perform while core components of the infrastructure are unavailable or entered

[3]This rate was selected because it was the last speed at which CRITs was able to process samples with a loaded and without critical errors.

Framework	1K	5K	10K	50K
CRITs	0	0	151	17,012
3 Workers	0	0	0	0
100 Workers	0	0	0	0

Table 3.2.: Critical failures in sample processing.

a failed state. We did this by running a script that randomly killed and restarted the machines, in isolation and in unison, hosting the *Planner*, *Transport*, and *Service* components of the system. During both experiments, we reran the newly generated sample sets through each hardware profile using the same method as in the scalability experiments.

The results of the experiment showed that SKALD's design and QoS paradigm greatly outperformed current systems. In the first experiment, as Table 3.2 shows, our prototype encountered zero critical errors, defined as a *Service* failing to complete tasking. Additionally, When the *Service*s did fail, the *Planner*s successfully recovered from all errors encountered by *Service*s and continued processing with other unaffected *Service*s. This is in direct contrast with CRITs reporting 17,012 critical errors using a set of 50,000 samples. In an investigation of the results, the high critical error rate was because the CRITs platform was unable to handle the load caused by failed CRITs *Service*s. Unfortunately, when a CRITs *Service* failed, these *Service*s entered a locked state that consumed valuable resources and could not be efficiently reallocated, or were killed by the operating system before results could be submitted for storage.

During the second experiment, SKALD remained tolerant of faults. While the overall processing speed was decreased, our prototype's QoS paradigm was able to identify failed states and re-queue the tasking without any operator interaction. The final outcomes revealed that no work was lost and 100% of the tasking were completed with no critical errors.

These results showed that the SKALD design provided resilient when processing poorly performing *Service*s but also easily handled failures to critical components of the system. Furthermore, these results revealed the benefits of SKALD's QoS structure because work was never lost and the system never entered a failed state. By contrast, CRITs' failed *Service*s remained in a failed state and human intervention would be required to clean faulty results from the database, reset the *Service*s, and re-task the system.

3.4.4. Flexibility

In order to examine SKALD's flexibility, we first evaluated SKALD's ability to incorporate existing *Information* extraction methods. To do so, we created an INTERROGATION *Service* by modifying the existing CRITs PEInfo *Service*. This required the modification of less than 50 lines of code in order to remove CRITs specific commands and provide a RESTful HTTP wrapper. During evaluation, we noted that the wrapped *Service* performed with no discernible difference when compared to natively written Skald *Service*s. The prototype was able to seamlessly incorporate the *Service* to include performing QoS operations. In summary, this demonstrated Skald's ability to provide flexibility by showing a minimal level of required work to incorporate a non-native *Service*.

Next, we wanted to study Skald's flexibility in changing a core component. We did this by testing the ability of our prototype to directly work with the CRITs database. This experiment was selected because the original implementation of Skald's Storage *Planner* used Cassandra and S3 as the database back-end. In contrast, the CRITs framework uses MongoDB for documents and MongoDB's GridFS for large objects. Thus, we were not only required to change the underlying scheme for storing artifacts but also the core database technologies. To do this, we made three modifications to the original prototype: (*i*) we created a Storage *Service* that parsed the results into the CRITs database scheme, (*ii*) we introduced an additional Storage *Service* that queried the existing CRITs database system to retrieve raw objects, and (*iii*) we included logic in the Storage *Planner* to identify which Storage *Service*s to select when handling tasking. In total, we were able to make these changes using less than 100 lines of code. Furthermore, no modifications to the other parts of Skald were required.

In summary, we are confident that Skald provides the flexibility required to relatively easily change *Service*s as well as core components of a system. We are confident that these results are applicable to other parts of the system.

3.5. Use Cases

In this section, we present use cases to demonstrate how Skald can overcome many of the current limitations in sharing artifacts among industry partners. These use cases are based on our experience in using our framework to manage a collective set of millions of objects, including their associated artifacts, across three globally distributed organizations.

3.5.1. Sharing Resources with Geographically Distributed Partners

Security partners rarely share infrastructure for fear that this will lead to unintended information exposure. This fear is well founded as it has in the past tarnished business reputations, reduced market share, impaired profits, caused privacy violations, and revealed internal sources and methods [30, 116, 117]. This poses a problem where processing resources leads to duplicated efforts and large capital is required for their maintenance. While we acknowledge that Skald cannot overcome the reasons behind why artifacts are restricted, our framework can overcome the problem of how to develop a shared infrastructure.

This is because SKALD allows each partner to leverage their own STORAGE *Planner* to restrict the transfer of restricted artifacts and configure the *Transport* to acknowledge internal and external *Planner*s. This in turn allows partners to leverage a collective set of hardware resources while allowing the creation of INVESTIGATION *Service*s capable of performing data discovery, clustering, and colorations over the entire set of collective artifacts.

3.5.2. Sharing Derived Information with Partners

Computer security investigations are mostly retrospective in nature and based on identifying previously observed attack patterns. Sharing artifacts are vital in this process as it allows analysts to make better correlations and create a more accurate picture of what is happening. Unfortunately, current methods for sharing artifacts among partners require too much time to be truly effective. This is because artifacts first need to be identified as important, processed, validated as sharable, and then transmitted to partners often in the form of IOCs. This is an inherently slow process but an even more critical issue is that methods used during the process to create an IOC is not uniform and vary among industry partners [118]. This causes the receiving party to rerun analytic methods before any received artifacts can be leveraged by their analysts' systems.

SKALD overcomes the above issue by providing a platform that supports breaking the traditional paradigm of how artifacts are shared. This is accomplished by providing the infrastructure to allow partners to directly share a central repository of artifacts (segregated according to the DIKW model) and jointly perform analysis against these artifacts. This approach allows partners: (*i*) to have a common set of analytic methods that is understood by all parties, (*ii*) to have real-time access to current artifacts, and (*iii*) to have a wider view of the malicious activity during the investigation.

SKALD makes the aforementioned setup an easy task. In our deployment, we created this setup with globally distributed partners by adding a STORAGE *Service* that contained the logic required to replicate *Information* between industry peers while restricting sensitive raw *Data*. Thus, this created a uniform platform that allowed all partners to leverage a collective pool of *Information*. Furthermore, when working with INVESTIGATION tasking, the taskings can leverage the ACL meta-tags, discussed in Section 3.2.1, to inform the system if it should only use in-house resources or use collective resources.

3.6. Lessons Learned

Throughout our evaluation period a number of additional discoveries were made that were unrelated to the performance of the SKALD prototype. The first discovery is that SKALD must be carefully implemented when calling *Service*s that rely on third-party network resources. This is because the potential query rate of a SKALD based system means that users can very easily exceed API query and rate limits. This can lead to the creation of potential issues along the processing pipeline, with more pronounced effects if the Skald deployment relies on a single API key. During the experiments, our prototype's ability to remain robust in the face of such conditions was demonstrated by exceeding our VirusTotal per minute query limit. Despite this, the resultant errors were captured and marked by our Investigation *Planner* to be reprocessed while other analytic processing continued unabated. Although this slowed down processing, the Skald prototype continued to work properly. More precisely, results were emitted from the PEInfo and Yara *Service*s, while tasking intended for the VirusTotal *Service* was re-queued for a second attempt. Thus, it is strongly recommended that Skald implementors configure their *Planner*s so that they automatically rate limit their calls to external, especially third-party *Service*s. Although failing to do so will not disrupt the stability of Skald, it can slow down overall task completion. Furthermore, it will annoy the *Service* provider. In our case, we overextended VirusTotal's infrastructure and caused their API to crash.

3.7. Related Work

The need for a large-scale malware analysis framework has been well discussed in prior work. Bitshred is a prime example and can perform malware correlation based on hashes of extracted features (a subset of *Information*), thus greatly increasing the throughput of the analysis system [123]. To address the issues of how to more agilely work with extracted features, Chau et al. created Polonium [126]. Polonium identifies correlations based on a set of features. Leveraging techniques similar to Pregel [127], Polonium takes a set of known features and performs machine learning against new samples to identify the probability of "goodness". However, both systems only address the problem of dealing with already extracted features and are reliant on other systems to generate *Information* and feature extraction. In turn, Hanif et al. [122] attempted to solve the problem of performing scalable feature extraction in their work on BinaryPig. This was done by performing feature extraction in a distributed fashion through the use of a Apache Hadoop based back-end. Unfortunately, BinaryPig is limited to only performing static analysis of malware binaries to generate features. Additionally, this system does not support performing analysis over the features and the QoS schedule cannot appropriately handle failed work. While these systems are promising and major achievements, they are only a piece of the solution when performing a computer security investigation. Specifically, neither of these systems support end-to-end investigations and do not easily allow analysts to work with different types of artifacts to ask questions that are outside of the original scope of the system's design.

Significant research has been put forward in distributed setups addressing the issues of scaling, flexibility, and resilience. SOA breaks systems down into business processes, i.e., *Service*s, and integrates them in an asynchronous event-driven manner through an ESB [98]. By establishing disjoint components, SOA enables distributed systems by defining how *Service*s communicate versus their implementation. The xSOA architecture expands upon traditional SOA by allowing multiple *Service*s to be combined under a single composite *Service* [97]. This composite server then provides a management layer which can perform *Service* orchestration, routing, provisioning, as well as integrity checking. Verma et al. [128] has taken xSOA a step further in their work on large-scale cluster management with Borg. They developed a xSOA-like system and optimized it by introducing a BorgMaster. The BorgMaster serves as the master for *Service*s and schedules their execution across Borglets. Together, the BorgMaster and Borglets intelligently manage the execution of *Service*s and perform necessary actions to improve resilience, flexibility, and scalability. Additionally, Borg packages *Service*s together for execution to improve efficiency by cutting down on transmission time, network bandwidth, and resource utilization. However, these architectures are not meant to deal with computer security work as malware is often designed to cause component failures and SKALD's additional abstraction is required.

3.8. Summary

In this chapter, we introduced an architecture to support Intelligence Cycle based investigations against the ever-growing volume of malicious activities plaguing computer systems. The architecture, named SKALD, enables the design of a large-scale, distributed system that is applicable in the security domain. To this end, SKALD supports multiple users and scales horizontally to perform analysis across millions of artifacts. Furthermore, SKALD breaks the paradigm that the automated extraction of *Information* is the ultimate goal and takes the viewpoint that providing a flexible architecture to empower human analysts is king. Specifically, SKALD provides mechanisms for analysts to leverage static analysis and dynamic analysis techniques to extract *Information* and apply advanced analytics across *Information* to generate *Knowledge* and create *Wisdom* while maintaining a central repository of artifacts. Empirical results confirm that SKALD can scale these tasks horizontally, at near linear growth, and is able to process artifacts at a rate of 3.1 milliseconds with zero critical errors in contrast to existing system's rate of 2.6 seconds and thousands of critical errors. SKALD's design also creates the building blocks needed to overcome the limitation of current sharing models. This is done by providing the infrastructure needed to allow industry peers to perform analysis across collective sets of artifacts while protecting sensitive data. Thus, SKALD enables more accurate investigations of malicious activities and real-time data discovery while minimizing the need for redundant systems and thereby reduces analysis time and infrastructure cost.

In the next chapter, we expand upon the concepts of SKALD to develop a sharing platform to further overcome the stove-piping of security teams.

Chapter 4

Expanding the Architecture to Enable Collaborative Analysis and a Sharing Marketplace

"It is a capital mistake to theorize before one has data."

Arthur Conan Dole

SKALD provides the architectural underpinnings for performing large-scale analysis across the Intelligence Cycle. However, another critical aspect that hinders defensive capabilities is caused by the stove-piping of security teams and the inability to collaborate and share. For decades it has been acknowledged that sharing security artifacts and collaboration between security practitioners is a necessity. Yet, effective sharing and collaboration is rare. A gamut of legislative acts, executive orders, academic works, and private sector initiatives have discussed aspects of the problem and aimed to be the catalyst needed to fix the situation. But almost 30 years since these efforts started, and even now the state of sharing and collaboration is technically complicated, slow, untrusted, and impeded by bureaucratic woes.

Unfortunately, this creates multiple problems. One problem is that performing assessments on limited sets of data causes analysts to draw conclusions and state correlations that do not fully encompass the situation [22–24]. Another problem is that each team must perform their own end-to-end investigation. However, investigations are a resource-intensive and time-consuming task. This is a waste of resources and as we previously discussed, analytic pipelines are already overwhelmed. Sadly, this stove-piping creates another problem that is more concerning. Because the community is not working together, malicious actors have a large window in which to move between victims using the same methods.

This chapter identifies the challenges of sharing and uses real-world examples to illustrate our findings. Based on this knowledge, we propose a new model for sharing and collaboration, CARE. The CARE architecture builds upon the foundation of SKALD to ease many of the privacy, secrecy, lineage, and structure issues that plague current sharing communities and platforms. We then leverage this foundation to introduce a marketplace based on smart contracts with transactional privacy over a distributed blockchain. Therefore, CARE incentivizes sharing, combats free riding, and provides an immutable ledger for the attribution of events. This paradigm shift overcomes the challenges of sharing while providing new opportunities for business models, insurance risk assessments, and government backed incentivisation.

4.1. Introduction

The modern day security team struggles with deriving accurate analytic assessments and with overcoming the status quo of just-too-late reactive defensive strategies. We posit that a core reason behind the current state of affairs is that our defensive tools and processes struggle to enable truly collaborative environments, and sharing security artifacts between industry peers is a technically complicated, slow, untrusted, and an overly bureaucratic task [29, 30]. As a result, analytic insights and assessments are less accurate and defensive actions are delayed or ineffective [116, 117].

Compounding the issues above, shared security artifacts do not provide the impact needed to defend systems. For instance, after a well-publicized and major security incident, the malicious actors still used similar tools and techniques to steal millions from hardened banking targets [26–28]. This should be no surprise to the informed security expert. Most research in this problem area has focused on the necessities of sharing and the ontologies to use [30]. But this research does not address the underlying problems. There is a reluctance to share, and the context needed to make shared artifacts valuable is often stripped. This makes adjusting defenses challenging and hinders proactive investigations and collaboration. To illustrate, the United States Computer Emergency Response Team (US-CERT) proudly released a slew of IOCs for a significant security incident [129]. However, these IOCs were nothing more than a collection of hashes with a quick description (i.e. “Lightweight backdoor”), a set of signatures with no details on how or why they were created, and IP addresses with a port and country. Thus, researchers given these IOCs can either blindly deploy the rules, or spend considerable effort to rebuild the context and perform their own investigation. However, deploying these rules will not prevent attacks by any moderately devoted malicious actor, because these actors can simply move infrastructure and conduct a re-signature of their tools [130]. In fact, actors can automate this process with methods such as polymorphism, metamorphism, and DGAs. On the other hand, performing an investigation is costly and on average takes even specialized companies 54 days [25]. Given this, it is no revelation it takes around 198 days for a company to discover they have been victimized, with some taking upwards of 6 years [2].

To move forward, we need a change in paradigm. To do this, we propose a new model for sharing computer security artifacts, CARE, which aims to provide the mechanisms required to perform analytic collaboration with a collective pool of artifacts in near real-time. It does this by providing a cryptographically backed exchange for sharing, derived through a set of common, verifiable extraction methods and analytic algorithms. As a result, this model provides the foundations for overcoming the privacy and secrecy issues with sharing; it maintains the context and lineage associated with derived artifacts; and it provides a common structure to allow shared artifacts to be easily ingested in analytic pipelines. Furthermore, the cryptographically backed method increases overall trust in the system, while also providing the ledger and infrastructure required to develop a sharing marketplace. In turn, this provides the necessary incentives needed to encourage companies and individuals to share, and have the immutable records needed to identify offenders of trust.

Our work makes the following main contributions:

- We discuss recent failures of real-world initiatives for security information sharing and dissect the associated challenges.
- We develop an architectural model for sharing that alleviates many of the privacy, secrecy, lineage, and structure issues associated with sharing initiatives.
- We introduce a secure ledger model that records who shared what, with whom, and when; and also guarantees the lineage and structure for shared artifacts.
- We present a design pattern for a cryptographically backed method for sharing that creates a marketplace based on smart contracts with transactional privacy over a distributed blockchain.
- We describe how our model creates new opportunities through the creation of new business models, providing metrics for identifying insurance risk, and providing the structure needed for allowing governments to provide tax incentives.

4.2. The Problem in Perspective

In November 2014, Sony Pictures Entertainment became the victim of an unprecedented attack that not only leaked Sony's private business records and communications but also destroyed valuable data [31]. In response, a combined governmental and corporate initiative immediately went into action to identify the threat, perform mitigation operations, and share across the community in the hopes of preventing future attacks [129, 131]. This effort was deemed a major success and a textbook example after which future responses should be modeled; now named the "Sony Model" [131]. One of the reasons behind why the response was deemed so successful is because the Federal Bureau of Investigation (FBI) treated the victims as a partner and encouraged the proactive sharing of IOCs.

Sadly, a year later millions of dollars were stolen from a bank by means of fraudulent Society for Worldwide Interbank Financial Telecommunications (SWIFT) transactions [28].

Initially the SWIFT and Sony attacks appeared unrelated. However, security researchers were able to attribute the attacks to the same actor, the Lazarus Group, and conclude that multiple other banks were also victims [26, 27].

The quick response to the Sony attack and the fact that details of the Sony attack greatly aided the SWIFT investigation clearly demonstrates the value of sharing within the security community. For example, the breakthrough in the SWIFT investigation, and attribution to the Lazarus Group, was in large part due to the widespread sharing of IOCs [27]. However, this example also highlights many of the major failures in the current sharing paradigm. For instance, the IOCs shared by the FBI through US-CERT were not what was cited as providing significant value during the SWIFT investigation. This credit went to an independent mitigation operation that occurred two years later, called Operation Blockbuster [26, 27, 31]. This is because the information originally shared by the FBI was heavily stripped of context to protect privacy, secrecy, and tradecraft. Taking a more critical point of view, one could even consider it a significant failure that the Lazarus Group was still able to use similar tools and techniques a year after the Sony attack in the SWIFT attack; even more so since the Sony attack received much publicity.

This episode highlights the potential benefits of sharing, but also the problems with the current sharing paradigm, which are rooted in a storied history. Dating back to the 1980s, numerous legislative acts, executive orders, and private sector initiatives have singled out sharing as a necessity for effective computer security and the lack of sharing as a major weakness. In response, these acts and initiatives were intended to be the catalyst needed to fix the problems with sharing within the security community [8–11]. Subsequently, communities and organizations were formed to act as facilitators for sharing, specifically Computer Emergency Response Teams (CERTs), sector-specific Information Sharing and Analysis Center (ISACs), and private tight-knit community trust groups [9, 10, 29, 132]. Furthermore, these private trust groups are often orientated around a single mission. For example, Yara Exchange is an exclusive group of researchers that focuses on creating a collective set of Yara signatures [125], while Ops-T is a vetted community that aims to thwart malicious behavior through collective action and sharing blacklists [133]. As these associations matured, they created common ways for their participants to communicate through the development of numerous standardized ontologies for cataloging information about computer security incidents, collectively known as IOCs [30]. Furthermore, to facilitate the communication of these IOCs, multiple sharing protocols and platforms were formed along with a new business sector focusing on cyber threat intelligence feeds [30, 134, 135].

Despite the above efforts, most sharing is typically done (*i*) through an ad hoc exchange of *Data* and unstructured *Information* by means of work acquaintances, small community trust groups, and between individual ISAC members, or (*ii*) via cyber threat intelligence feeds that are delayed and of questionable value [15, 29, 136]. Despite the delay in sharing actionable details, the level of sharing that occurred after the Sony attack and the details that were provided as part of Operation Blockbuster are a positive anomaly. Simply put, widespread sharing and collaboration of timely artifacts that crosses sectors rarely happens and when it does its impact is often less than it should be [136,137].

4.3. The Realities of the Current Sharing Paradigm

The major issues surrounding why sharing and collaboration do not regularly occur and lack effectiveness, when it happens, can be summarized as follows [29,30,116,117,137–140]:

- **Privacy of Victims** - Exchanging raw *Data* can leak information regarding the victim's identity and their sensitive data. This inadvertent disclosure can directly harm the victim and their reputation, eroding market share, as well as cause contractual, legal, and regulatory violations on behalf of the sharer.
- **Secrecy of Attack Patterns** - Raw *Data* can divulge the methods and techniques used by the attacker as well as details about the victims' infrastructure and computer security posture. This can empower and encourage other malicious actors. In an infamous case, Zeus' leaked source code was used to create Citadel and ICE IX [141].
- **Tradecraft of Investigators** - Requesting and exchanging artifacts can alert attackers that an investigation is occurring. This can allow attackers to shift tactics and increase their chances of evading future detection by defensive monitoring and controls. Additionally, this can leak business secrets to peers, which can reduce competitive advantage.
- **Lack of lineage** - Shared artifacts are often stripped of vital context, and how the artifacts were obtained and generated is unknown. This causes a situation where shared artifacts are untrusted, the relevance to the recipient is not immediately apparent, and the artifacts must be reprocessed or amended through informal channels to be useful for the receiving party.
- **Lack of structure** - Shared artifacts may be unstructured, poorly structured, or in a myriad of different IOC formats. As such, significant time and manual intervention is required to ingest the feeds in the recipient's own workflow.
- **Absence of ledger** - No universal method to track verifiably who shared what, with whom, and when exists. This leaves little potential to identify offenders of trust or allow for crediting of the contributors of valuable artifacts.
- **Lack of incentives** - There is no directly apparent economic benefit to community-wide sharing. Additionally, commercial security companies may be disincentivized to share freely for fear of diminishing their competitive advantage.

In total, these reasons create an environment where organizations and individuals are reluctant to share and when sharing occurs the artifacts are stripped to the point that the

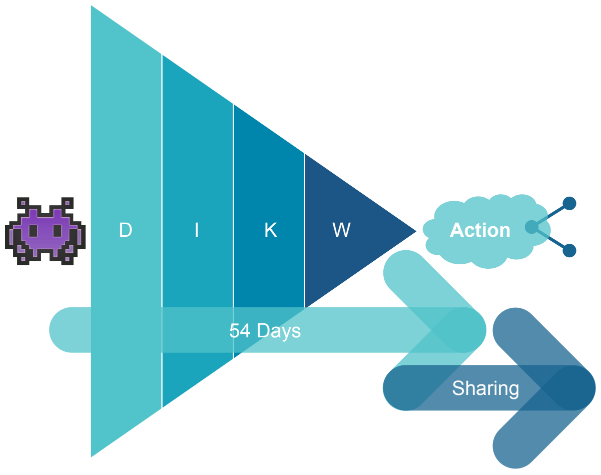

Figure 4.1.: DIKW Pyramid with Respect to Average Mitigation Time and Sharing

immediate value becomes questionable. Unfortunately, this culminates into a paradigm where potentially vital details that can mitigate threats often never reaches (potential) victims in time.

4.3.1. Wisdom Without Context is Merely Data

In the Sony case study, the response team proactively shared artifacts, including a summary of some of the attacker's tools and unstructured IOCs containing import table hashes, binary MD5s, command-and-control IP addresses, Snort Signatures, and Yara rules [129]. While sharing publicly at this level is rare, what was shared is typical of the the types of artifacts that are broadcasted via ISACs, community trust groups, and cyber threat intelligence feeds. For example, the Financial Services - Information Sharing and Analysis Center (FS-ISAC) advertises that they provide the sharing of different types of reports and the means for members to ask for further information through submitting a Request for Information (RFI) [142]. Similar to ISACs, two of the most popular trust groups, Yara Exchange, and Ops-T, regularly share Yara signatures, lists of blacklisted domains, hashes for malicious samples, and allow members to directly request additional information about an object [125, 133].

Regretfully, this proactive sharing of IOCs was not enough to hinder the Lazarus Group from using similar tools and techniques during the SWIFT attacks. The issues around why this occurred are best described when viewing security analytics and investigations under the perspective of the DIKW model [39]. As illustrated in Figure 4.1, analytic insight or action is achieved by building upon each layer of the DIKW model: *Data*, *Information*, *Knowledge*, and *Wisdom*. Applying this to computer security analytics, we can derive the following:

- ***Data*** - The raw object: PE32, PCAP, memory dump, domain, IP, file, etc.

- ***Information*** - Details about the *Data* that can be determined through static analysis, dynamic analysis, or other extraction mechanisms.
- ***Knowledge*** - Organizing a set or a subset of *Information* into useful forms using statistics, knowledge-based rules, machine learning, or other analytic techniques.
- ***Wisdom*** - Developing an understanding of the *Knowledge*, based on experience, to allow a judgment or action to be made.

Only once *Wisdom* has been derived, can the defender fully comprehend the threat and formulate an effective response. Thus, the common practice of sharing IOCs that contain nothing more than the derived *Wisdom* without the lineage of how there were generated (for instance, various lists of hashes, IP addresses, a specific import used by a binary, or a signature) does not greatly assist in the analytic loop. Shared *Wisdom* without context is just *Data*. In turn, this makes it difficult for the recipient to work with or generate *Wisdom* when they do not understand the context of how the *Information* was generated and how the *Knowledge* was pieced together. This makes it challenging to apply the received *Wisdom* to their own investigation and ask different questions to derive a different meaning. Furthermore, without a common structure it is not easy for shared artifacts to be merged into the recipient's analytic pipeline in order to identify further meaning or understanding. Hence, the situation occurs where the receiving party needs to gather the original *Data* behind shared IOCs and reprocess them before further analytics or effective action can take place. However, a catch-22 occurs because the original *Data* is often restricted and rarely exchanged due to issues of privacy, secrecy, and tradecraft.

4.3.2. The Need For Speed

The current paradigm is lacking in timeliness to be effective. For instance, the Lazarus Group used the same tools and techniques to attack multiple victims over a period lasting longer than a year. This is unsettling, but it should come as no surprise. As shown in Figure 4.1, it takes an average of 54 days for a specialized company to move from *Data* to *Wisdom* and develop a response after a malicious action has been identified [25]. Furthermore, 15% of known malicious files are still not detected, let alone mitigated, until 180 days after being released [25]. The Sony case was faster than average, sharing an initial set of IOCs in less than 30 days and then amending the IOCs about a year and a half afterwards [129]. Unfortunately, this is still much too slow, especially when considering that 75% of attacks spread from the first victim to the second in less than 24 hours [143].

Regrettably, even when information is shared, the security community still faces an issue where the sharing recipients must enter a cycle of reprocessing any received *Data*, then ask for more *Data* based on what they identified, and then reprocess this newly received *Data* to generate the *Information* and create the *Knowledge* needed to develop their own *Wisdom*. This process is required because what is shared is a static snapshot of a previous attack and does not contain any lineage. As such, given a shared IOC the recipient can only understand, at a general level, what was previously used in an attack and the details are obfuscated. Unfortunately, this allows attackers to easily overcome the threat posed by sharing. Specifically, an attacker can move infrastructure and obfuscate malicious code using any method as long as it is faster than the defender can complete this resource-intensive cycle [130].

4.3.3. Lack of Trust In Exchanged Items

The current sharing paradigm does not have a verifiable lineage. The Sony case study is no different in that the originally shared *Information* provided minimal context and little details regarding how the *Information* was generated. Unfortunately, this causes issues of trust between sharing partners and inaccurate assessments. One of the reasons behind this is that different methods can be used for generating similar types of *Information*. However, while similar, the *Information* is not always interchangeable and may even contain flaws. For example, Wesley Shields recently reported that the implementation of PEHash [67] used by widely popular tools such as Totalhash [144], CRITs [14], and VIPER [145] incorrectly generated the hashes [118]. This caused a problem when generating *Knowledge* and *Wisdom* based on shared *Information* because the hashes would not match and inaccurate results were produced. Unfortunately, this issue is not rare and as such it is common practice to validate any received IOCs until a level of trust in the originator can be established [138].

Eroding trust further, the current model's reliance on sharing by either massively distributing IOCs or providing specifically requested details among peers exacerbates the concerns surrounding privacy, secrecy, and tradecraft. This is because these methods lack a universal and trusted ledger of what was shared, with whom, and when. As such, the originator loses traceability, which hinders the identification of abuses of trust. Thus, it is nearly impossible to enforce any security and privacy controls or perform retribution against violators. This fear is evident in the exclusivity of trust groups and ISACs as well as the level of context that was stripped from the originally released IOCs in our case study.

4.4. The CARE Model

CARE is a design pattern for developing analytic systems that enable collaboration and alleviate the current issues with sharing. At its core, CARE is an architectural foundation for generating and exchanging security artifacts across the DIKW model; thus, allowing partners to overcome the challenging and time-consuming burden of rebuilding the context and *Information* behind IOCs, finished reporting, and *Data*. Building upon this foundation, CARE then leverages smart contracts on top of blockchain technology

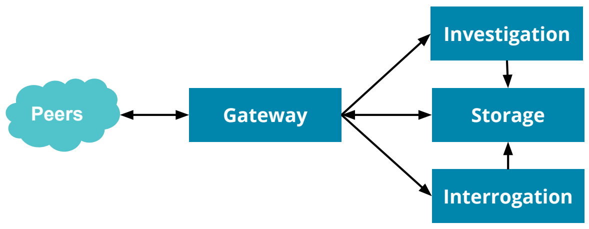

Figure 4.2.: CARE architectural components and interaction.

to enable a cryptographically backed exchange (CAREconomy).

The design for CARE is based on the Skald framework and extends the original design to enable healthy collaboration and create a sharing marketplace [21]. As discussed in the previous chapter, Skald is an architecture for performing large-scale computer security analysis across the Intelligence Cycle. While not just theoretical, prototypes of Skald have been successful in enabling multidisciplinary teams to perform rapid and complex analysis against sophisticated malicious actors, e.g., during the Blockbuster investigation [31]. However, we note that the CARE model can be extended to other frameworks, such as the Malware Information Sharing Platform (MISP) and CRITs [14, 135]. That said, Skald was particularly selected because of its uniqueness in supporting Intelligence Cycle based analytics and segregating the processes across the DIKW model. While MISP and CRITs are powerful and effective, both systems only focus on generating *Information* from *Data* and sharing a myriad of IOC formats. As such, Skald is a natural fit for breaking the paradigm and achieving the goals of CARE.

In this section, we will describe the CARE architecture, explain the CAREconomy, and lastly discuss how sharing partners interact with the system.

4.4.1. CARE Architecture

The architecture for CARE builds upon the Skald concept of *Planner*s and *Service*s. In this, the *Planner* is centered on supporting the goals for a specific theme and orchestrates the execution of *Service*s. The *Service* component is “loosely coupled” and performs the execution of a task. For example, the Investigation *Planner* smartly orchestrates *Information* extraction *Services* for *Data*. Under this structure, the CARE architecture primarily provides (*i*) the ability to manage partner interaction, (*ii*) the generation of artifacts, and (*iii*) it creates an abstract method for storing artifacts and system data. As shown in Figure 4.2, the architecture is composed of four core *Planner*s: Gateway, Investigation, Interrogation, and Storage. The 5th Skald *Planner*, Presentation, is not directly addressed in CARE. This is because while it is important for controlling user interaction, it is out of scope for creating a sharing marketplace. As will be discussed in this section, these *Planner*s are designed to overcome the privacy and secrecy issues with sharing *Data*; to maintain the lineage associated with *Information*, *Knowledge*, and *Wisdom*; and to provide a common structure for sharing. Together this

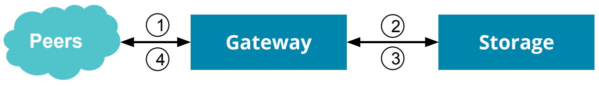

Figure 4.3.: *Planner* interaction when an artifact exists in STORAGE: (1) GATEWAY receives and validates a peer request, (2) the artifact is identified in STORAGE, (3) STORAGE transmits the artifact to GATEWAY, and (4) GATEWAY provides the results to the requestor.

empowers peers to exchange across the DIKW model and more effectively collaborate.

The GATEWAY *Planner* is central for managing partner interactions and exchanging artifacts across the DIKW model. The GATEWAY provides four key functionalities: (*i*) notifying peers what artifacts are available, (*ii*) authenticating and validating requests, (*iii*) orchestrating the exchange of artifacts, and (*iv*) administrating the CAREconomy.

GATEWAYs communicate with each other in a peer-to-peer fashion. This communication method keeps the artifacts in the control of the owner while also providing the foundational building blocks on which collaborative trust groups can be built. The peer-to-peer approach also allows peers to have fine-grained control over what artifacts are available and who can access their system. For instance, a peer can restrict the sharing of an artifact based on a taxonomy or specific *Service*s as well as which peers are able to access these artifacts. This is a dramatic difference over traditional sharing platforms, such as CRITs and MISP, because those model access based around the concept of group-level access or access based on the source of *Data* in which the artifact was generated [14, 135]. Furthermore, the peer-to-peer method removes the necessity of trusting a sharing platform provider with protecting all the assets; a key finding highlighted by Clemens et al. [30].

Illustrating how GATEWAY functions, in CARE, partners submit all requests for exchanging artifacts through GATEWAY. When GATEWAY receives a request, it first authenticates the requester and validates that the request is properly formatted. After that, GATEWAY gathers the requested artifact from STORAGE (Figure 4.3) or submits a request to INVESTIGATION or INTERROGATION to generate the requested artifact (Figure 4.4). In its final step, GATEWAY gathers the artifact, adds the pertinent metadata to the artifact, updates the distributed ledger, and submits the results back to the requester.

The next set of *Planner*s, INVESTIGATION and INTERROGATION, are charged with transforming artifacts to the next higher level in the DIKW model. Regarding the INVESTIGATION *Planner*, this *Planner* is responsible for generating *Information* from *Data*. INVESTIGATION does this by orchestrating the execution of *Service*s that perform static analysis and dynamic analysis as well as gathering *Information* from 3rd parties. In a similar vein to INVESTIGATION, the INTERROGATION *Planner* focuses on transforming information into *Knowledge* and empowering an analyst to create *Wisdom* through assessing collective *Knowledge*. For example, an INTERROGATION *Service* can execute a machine learning algorithm to cluster samples or help to label artifacts by executing

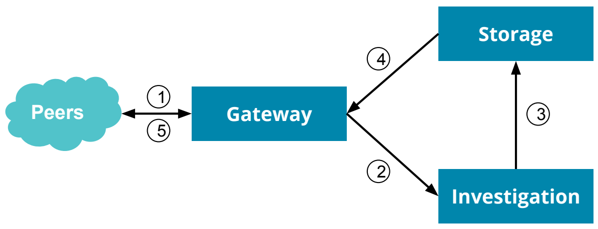

Figure 4.4.: *Planner* interaction when *Information* needs to be generated: (1) Gateway receives and validates a peer request, (2) Gateway submits a request to Investigation to generate the *Information*, (3) Investigation transmits the artifact to Storage, (4) Gateway retrieves the *Information* from Storage, and (5) Gateway provides the results to the requestor.

statistical or knowledge-based analysis. With respect to *Wisdom*, a *Service* could help with visualizing sets of artifacts, creating a standardized set of IOCs, or generating blacklists.

In this model, the Investigation and Interrogation *Services* that are available for sharing are known, with cryptography guarantees, and agreed upon by the peers or an ombudsman. This helps to overcome some major challenges with sharing unstructured artifacts or IOCs, because the lineage and structure is maintained and understood by all parties. As such, the results a *Service* provides and how the artifacts propagated through the system are known. This allows these results to be immediately incorporated into the receiver's analytic pipeline because the context is understood. Thus, it reduces the time required to process any newly shared artifact a party receives. Additionally, this overcomes the challenges that stem from only providing an exchange for IOCs even if the lineage is known. As discussed prior, and can be witnessed with the MISP platform, there are a plethora of different ontologies that can be classified as an IOC and these ontologies can have extensive vocabularies. To illustrate, the MISP document that describes the subset of vocabulary for ontologies they support is 326 pages [146]. This makes immediately leveraging a shared IOC difficult because the receiver of an IOC must first validate the format, ensure it is being used in the same way, and convert the format to the in-house style.

Lastly, this method helps to overcome the privacy and secrecy issues that stem from sharing *Data*. This is because the context behind an artifact is known and the methods for transforming *Information* and *Knowledge* are available, direct access to *Data* is not required for an investigation.

The Storage *Planner* in CARE is akin to the original design of Skald. Specific to sharing, Storage manages the repository of *Data*, *Information*, *Knowledge*, and *Wisdom* for each peer in addition to CARE-specific data. Storage also provides an abstraction layer between database systems. This abstraction allows peers to incorporate the model over existing systems, leverage a single or hybrid back-end solution, and select the database backed of their preference.

4.4.2. CAREconomy

The CARE architecture provides a foundation for generating and exchanging security artifacts in a way that alleviates the current sharing paradigm's issues of privacy, secrecy, lineage, and structure. However, many of the issues with sharing are caused from the fact that sharing platforms are based on the reputation of individual contributors [30]. This can erode trust in these groups due to accident or malice by the participants. For example, original authorship can be mis-attributed or forgotten, community engagement can go unnoticed, and breaches of confidence can occur. Unfortunately, these fears are well-founded. Greed, the desire for recognition, and forgotten ownership has caused sharing partners to release or act upon restricted details early at the detriment of the collaborative effort [147]. For example, during the Mariposa botnet take-down, the DNS registrar was successfully bribed into helping the malicious actors regain control of the botnet [148]. Furthermore, as groups grow, the problem of free riding, where participants reap the benefits but do not contribute, becomes prevalent [117, 149]. If left unchecked, these issues will erode participation and wear down the perceived benefits to sharing. To combat these issues, CAREconomy is focused on developing overall trust in CARE. This is done with a smart contract system based on the blockchain. As such, CARE provides an immutable ledger, allows for distributed trust among mistrusting peers, and providing incentives to share by creating a marketplace.

4.4.2.1. Developing a Marketplace

Algorithm 2 GATEWAY receives a transaction request for *artifact* from $user_a$ with value of n CARECOINs

```
Ensure: $user_a$ is authorized
Ensure: $ledger[requester] \geq$ CARECOIN
    $ledger[requester] \leftarrow ledger[requester] -$ CARECOIN
    $ledger[sender] \leftarrow ledger[sender] +$ CARECOIN
    transmit artifact to requester
    update global ledger
```

Conceived in the late 1990s, smart contracts provide the theoretical underpinnings for establishing, executing, and enforcing contractual clauses in the digital realm [107]. Unlike their paper predecessors, smart contracts have the added benefit of allowing the automatic execution of contract terms and remove the requirement of having a trusted third party to guarantee transactions. Modern implementations, such as Ethereum, realize the concept of smart contracts by provide a Turing complete language on top of blockchain technology [109]. This provides the ability to execute small programs while recording the interactions in an immutable and distributed ledger. Additionally, techniques such as PoW and PoS verify modifications to the blockchain and thus provide trust for the transaction and ensure contractual breaches are prohibitively costly.

In other sectors, smart contract based systems have nurtured healthy sharing. For instance, KARMA used smart contracts to create an economic system for peer-to-peer file sharing [150]. This system created healthy sharing by enforcing a balance of what was provided to the community versus what was taken. Furthermore, BitTorrent communities that leveraged similar concepts saw a drastic reduction in free riding behavior [151]. Shockingly, many of these communities reported upwards of 100 seeders per leecher, whereas open systems only averaged a ratio in the single digits. This directly benefited the communities and resulted in higher download speeds (3-5 times faster) and stronger rates of connectivity.

Using these concepts, the CAREconomy is based on using smart contracts on the blockchain to create a community-maintained marketplace for sharing security artifacts in CARE. Under CARE, the community selects who can participate and what is available for trade by approving which *Service*s are allowed to be executed to created other DIKW artifacts. The Gateway then enforces these decisions and manages the exchange of artifacts between authorized users using a cryptocurrency, hereafter referred to as CareCoins. Shown in Algorithm 2, at a basic level this permits the buying, selling, and reselling of security artifacts between peers which allows the creation of an economy and generates incentives. As explained in more details later, under this model *Data* artifacts can be added at will but the *Service*s used to generate *Information* and *Knowledge* are controlled. This limits the ability for fraud as the the market deems which *Data* artifacts, and their associated artifacts higher up the DIKW chain, are of value. Additionally, the immutable ledger records all transactions which overcomes the problems of tracking community engagement, author attribution, and identification of breakers of trust and fraud. For example, in cases where a partner releases information early or for their own gain, the immutable ledger would allow identifying of who accessed what, and when, and where the artifact came from. While this does not guarantee malicious or harmful inadvertent acts will not occur, it can provide details to track what happened and be used as a deterrent or justification for removing that participant. Lastly, the blockchain backing of these smart contracts allows for a level of trust in the system that is currently lacking among sharing communities. This is because the blockchain provides distributed trust even when multiple parties need to modify the state of the system and are fundamentally mistrusting of each other and the intentions of any third party.

Algorithm 3 Gateway receives a transaction request for an *artifact* at CareCoin

```
Ensure: $user_a$ is authorized
Ensure: $ledger[requester] \geq$ CareCoin
    $ledger[requester] \leftarrow ledger[requester] -$ CareCoin
    $ledger[sender] \leftarrow ledger[sender] +$ CareCoin
    add SNARK details to $ledger$
    update global $ledger$
    allow transmission of $artifact$ to requester
```

Chapter 4

4.4.2.2. Verifiable Lineage

One of the main issues with current sharing paradigms is that there is no way to verify what method was used to generate the artifact that is being shared and the object itself. CAREconomy addresses this issue by leveraging zk-SNARK to verify the execution of programs that live outside the blockchain [152]. As shown in Algorithm 4.4.2.1, this is done by adding the zk-SNARK proofs, to the ledger. This has two direct benefits. The first is that zk-SNARK provides the ability to verify which *Service* was used to generate the exchanged artifact and the exchanged artifact. As such, the recipient is guaranteed to understand the structure and lineage of shared artifacts as well as have trust in what was being purchased is what was requested. The second benefit is that the code used for a *Service* and the artifact can live outside the blockchain. This provides the ability to leverage more complex *Service*s, update *Service*s more easily, and it also reduces the size of the blockchain. Additionally, while CARE aims to lessen the risks associated with widely sharing, especially *Data*, these risks are not eliminated. However, zk-SNARK-based transactions do not expose the private information to the blockchain and thus still allow for the appropriate remediation to occur.

4.4.2.3. Transactional Privacy

As previously discussed, a major factor to consider when sharing security artifacts is in preserving the tradecraft of the investigators. For instance, revealing the tradecraft secrets of what is being investigated can tip off a malicious actor that an investigation into their activities is ongoing and cause them to disappear or shift tactics before a mitigation action can occur. Furthermore, revealing what is being investigated can lead to suspicion that an exposure has occurred which can cause bad publicity, degrade market value, or erode market share. CARE addresses this concern by expanding to use zk-SNARK to add trust to the system by allowing the incorporation of transactional privacy [113]. This is done by using zk-SNARK to encrypt the details recorded in the ledger behind what was shared between two parties. In turn, this further adds trust to CARE by granting peers the ability to share artifacts without exposing what was shared.

4.5. Discussion

The CARE model creates a new way forward for sharing and collaborating in the security community. In this section, we will discuss how this new model presents previously unattainable opportunities for the creation of sharing partnerships, expands how contributors can take part, and presents new possibilities for incentivizing and assessing partner collaboration and effectiveness of sharing.

4.5.1. Creating Collaborative Communities

Security communities have historically been established to fill a community need or to service a specific sector. For example, FS-ISAC was created to foster collaboration and sharing in the financial sector, Yara Exchange was created to crowdsource the creation of Yara signatures, and Ops-T brought together vetted security practitioners with the goal of exchanging information to collaborate in the mitigation of security threats. For reasons explained in the previous section, these groups are almost universally tight-knit and have processes in place that attempt to vet new members, encourage participation, and overall maintain a level of trust. Unfortunately, this is often a losing battle and over time the level of quality in what is shared degrades and member participation declines [117, 149]. In sum, while the goals of these communities are noble, the effectiveness and utility of these communities and cyber threat intelligence feeds are often left in question.

4.5.1.1. Sharing Communities

CARE empowers these communities and intelligence feeds by overcoming the challenges discussed in Section 4.3. In turn, this enables traditional security communities to more effectively share and collaborate while providing the infrastructure needed to keep them healthy and encourage participation. Additionally, the secure ledger and incentives that CAREconomy provides open the opportunity for larger collaborative efforts by lessening free riding and overcoming the risks associated with sharing data widely. CARE is envisioned as becoming the new method for collaboration and exchanging artifacts within private trust groups, ISACs and CERTs, corporate partnerships, and be the driver needed to enable the creation of large communities that span multiple sectors and security specializations.

To help illustrate the above, the new paradigm afforded by CARE allows the buying and selling of artifacts that can be immediately plugged into the analytic pipelines of the recipients. This is because the context and lineage around the original artifact or any subsequently generated artifact is known and trusted by the sharing partners. This opens up the market and allows participants to focus on an area of specialization verse being typical broker of raw material or finished analysis. As one of many possible examples, Partner A can specialize in the sourcing of *Data* artifacts and make them available for sale on the marketplace. Partner B can then specialize in the extraction of *Information* from *Data* and sell this service or the resulting *Information*. Partner C can then specialize in generating *Knowledge* based on their sets of *Information*. The resulting *Wisdom* that can highlight other *Data* that can be of value, inform what items should be removed or blocked from a network, provide insights that can assist in a mitigation operation, or produce an investigative result can then be disseminated by another Partner. In turn, each partner is able to participate in their own unique way while furthering the goals of a collective community and marketplace.

4.5.1.2. Community Management

To manage sharing communities, CARE necessitates the use of an ombudsman or other structured form of governance. In the traditional cryptocurrency world, an informal method of governance has been the dominating force [153,154]. These governance models provide many benefits and have been surprisingly long lasting. However, they also have their issues, specifically with managing access, evolving to change, and responding to abuse [155,156]. Furthermore, a smart contract that lives on the blockchain is immutable by nature. Unfortunately, these contracts can be challenging to write and are not immune to vulnerabilities [115]. For instance, the Decentralized Autonomous Organization (DAO) smart contract that provided a form of informal governance had numerous flaws in its design [155]. In one particular case, this allowed an attacker to steal 50 million dollars while the community could only watch [157].

When these issues with decentralized governance models are posed together with the unique challenges created by computer security investigations and the handling of security artifacts, it is clear that a different approach to governance is required. As such, we propose that CARE models its community management in a manner that is more akin to the ISAC model. In this, the community should have a structured form of governance that serves to foster trust and encourage collaboration within the community [158]. At a minimum, we propose that the management of a sharing community should provide:

- **Management of users** - Approve access to the community and perform traditional user management functions.
- **Organization of the CAREconomy** - Declare sanctioned contract types, maintain a set of approved *Service*s, take action against fraud, select the method for generating money and validate the blockchain, and oversee the economy.
- **Remediation for security and privacy concerns** - Mediate member disputes and perform remedies in cases of information leaks and inadvertent exposures.
- **Proactive reduction of security risks** - Vet service code and the code used for creating smart contracts.

4.5.2. Sharing Partners

The current sharing paradigm has fostered ad hoc exchanges and intelligence feeds of questionable value. As discussed, these exchanges are often delayed, and the received artifacts need to be re-validated and reprocessed before they can provide utility. Within a sharing community, this creates an all-or-nothing situation where participants typically must perform all the steps required to create and disseminate IOCs or otherwise advance the community's mission. However, the ability in CARE to share artifacts across the DIKW model presents new opportunities and grants sharing partners the ability to collaborate with asymmetric resources in near real-time. This allows participants the ability to specialize in different types of threat research while still furthering the collective knowledge of the community. For example, independent researchers can maintain sets of honeypots that collect *Data* and leverage the community's *Knowledge* to help identify what was collected. University researchers can obtain *Information* that enables research in new machine learning algorithms and return the *Knowledge* they derive. Corporate security teams can perform automated triage, such as PE32 header extraction, on new *Data* to generate *Information* and leverage the collective knowledge to better defend their networks. And threat intelligence providers can sell *Wisdom* (e.g., actionable IOCs) or specialized *Information* and *Knowledge*.

4.5.3. New Opportunities

Critics of historic efforts to promote sharing and collaboration in the security community have identified that the government needs to incentivize healthy sharing and security practices. Unfortunately, the identification of how to measure the effectiveness of a security team and their sharing practices has been a hotly discussed item for decades [159]. The underlying reasons have been discussed in this work but the core issue is that there are no good metrics for what makes a security team effective and what defines the value of shared artifacts. Moreover, the current sharing paradigm does not provide the infrastructure and records needed to create these metrics.

The CARE model provides a ledger that records all interactions; even if the details of what specifically is exchanged can be encrypted. This presents new possibilities in how to determine and rate how companies collaborate and in turn the effectiveness of the security teams' efforts. This can provide the metrics needed for developing useful government incentives. Additionally, insurance companies can use these metrics to determine the risk factors associated with insuring a company and adjust rates accordingly. These incentives and overhead adjustments combined with the ability to accumulate wealth through the CAREconomy helps break the mold of corporate security groups being taxing cost centers. In turn, this provides the potential of these groups to become not just a necessary evil in the form of a red-line on a balance sheet to mitigate risk but profit centers or at least groups that can articulate their worth.

4.6. Limitations and Future Work

The complexities of issues surrounding sharing makes the design of a sharing model challenging and imperfect. Additionally, as CARE creates a new paradigm for how sharing can be effectively conducted, it would be arrogant to foreshadow how these concepts can be implemented and evolve. In this section, we will discuss potential pitfalls we have identified and areas for future research.

4.6.1. Secrecy and Privacy Considerations

The CARE architecture allows for sharing security artifacts while maintaining the associated context and lineage. This is done by separating artifacts across the DIKW model and leveraging the immutable ledger provided by the blockchain for maintaining the lineage and context of shared items. However, the immutable natures of smart contracts and how verification of the blockchains work, present unique drawbacks and must be appropriately considered when implementing a CARE system.

One concern is that smart contracts published on the blockchain are immutable. While workarounds do exists for changing published contracts, such as Ethereum's $SELFDESTRUCT$; they are challenging to use and often require adequate foresight. As such, care must be taken when publishing a contract to ensure security vulnerabilities are minimized [115]. While we highlight this concern, we leave the handling of this issue to the governance model selected by the sharing community. This is to allow flexibility in the governance model and allow the adoption on future methods as this is a ripe area of research.

Another concern is that validating the blockchain requires agreement by the majority of peers and is often implemented using methods such as PoW and PoS. In both cases the agreement of the majority takes precedence. This can expose a system to vulnerabilities of trust, known as the 51% attack, where if one party or group gains a majority stake, they can reverse transactions, allow double spending, and halt transactions all together. Furthermore, PoW relies on solving complex problems, aka mining. As more problems are solved, more energy and computing power is required. Eventually, a state occurs where mining becomes exceedingly expensive for a minimal return on the part of the miner. However, the CARE architecture does not dictate what consensus method is used for mining. This was done to allow a community to select which consensus model best fits their organizational structure and allow the adoption of future methods. For instance, if a community wishes to be public, leveraging an existing blockchain, such as Ethereum, would allow the community to use the wider community's consensus method to reduce the possibility of cheating. However, if a different community wishes to maintain a private blockchain, CARE grants them the flexibility to create a new or unique consensus model that best fits their use case, such as one based on participation. That said, when managing a CARE sharing platform, care must be taken to ensure an appropriate consensus method is used to minimize the possibility of malicious actions and ensure a stable and functioning system.

4.6.2. Identification of Shareable Resources

The CARE model makes the assumption that partners are aware of the resources available in the CAREconomy. This was done to allow flexibility in how CARE is implemented and the CAREconomy is organized. While the model leaves the design and analysis of resource identification to further research, we provided the following to aid the discussion:

- **Bounty** - Partners can post requests for artifacts to their Gateway.
- **Gateway Inquiry** - Partners can issue an inquiry about the availability of artifacts to connected Gateways.
- **Replication of Metadata** - Partners can replicate the metadata associated with their artifacts that are available for sharing.
- **Relationship Identification** - The ombudsman can orchestrate the creation of a graph identifying related artifacts and transmit these to partners.

4.7. Related Work

The belief that sharing security information among peers will improve the overall defensive posture of the community has been well-established. For instance, after the Morris Worm attack the U.S. government created CERT, and PDD-63 created ISACs to help combat the perceived threat to critical infrastructure [9, 132]. Providing academic rigor behind the notion that sharing is critical, Gordon et al. evaluated the state of sharing and developed an economic model that analyzed the organizational cost [149]. This work was then expanded by Gal-Or et al. who used game theory to study the demand side effects occurring under the current sharing paradigm [117]. Both works are critical to understanding the major benefits that sharing provides to the participants and community at large while also identifying the underlying flaws in the current system. Specifically that sharing provides mutual benefits to security and cost savings, but the effects are negated because the current sharing paradigm does not provide incentives to prevent *free riding* and overcome participants' concerns. Unfortunately, as shown in a recent incentive study, the identification of incentives is still a ripe area for research [159].

Little research has been conducted about the technologies to enable sharing in the security community. To address this, Sauerwein et al. perform an exploratory study of twenty-two cyber threat intelligence platforms and the state of scientific research [30]. Their analysis identifies eight key findings and highlights the current issues of trust, structure, speed, and overall need to migrate these systems from sharing IOCs to sharing information across the Intelligence Cycle. Another notable work is PRACIS, which adds format-preserving and homomorphic encryption to increase trust in how information is exchanged by protecting the privacy of the request and requester [160].

The theoretical concept of smart contracts for formalizing and securing digital relationships dates back to the late 1990s [107]. One of the first implementations of this concept based on PoW was KARMA [150]. In KARMA, the researchers devised a method for overcoming freeloaders (free riding) in peer-to-peer file exchanges through the use of a secure decentralized ledger. In a similar vein, Nakamoto proposed a hash-based

PoW system for payment, which is now famously named Bitcoin [102]. While Bitcoin's blockchain has given rise to numerous applications, its scripting language is not Turing complete and difficult to retrofit. Ethereum addresses this issue and provides a blockchain with a Turing-complete language with the possibility to implement smart contracts [109]. Expanding on the concepts of Ethereum, Kosba et al. present a blockchain based smart contract system that incorporates transactional privacy, HAWK [113].

These works highlight many of the underlying problems and provide parts of the solutions to the issues of sharing security information. However, these works only identify the problems or provide solutions to specific issues that are surrounding the challenges of sharing computer security artifacts. Our work combines and builds upon these works and is the first to our knowledge that combines these concepts to tackle the holistic problem of developing a framework for securely exchanging computer security artifacts across the Intelligence Cycle, while overcoming the issues of privacy, secrecy, tradecraft, lineage, structure, ledger, and incentives.

4.8. Summary

In this chapter, we presented a new model for sharing security artifacts, CARE. We discussed how CARE breaks the existing sharing paradigm and alleviates many of the issues with why sharing is often ineffective. We first discussed the need for the new model by presenting a study of the current state of sharing and identified the associated issues. We then show how CARE overcomes these issues by providing the ability to exchange security artifacts across the DIKW model while preserving the artifacts' lineage and mitigating privacy and secrecy concerns. We then discuss the CAREconomy and describe how this cryptographically backed method incentivizes sharing through the creation of a marketplace and provides new opportunities to encourage healthy collaboration and develop trust. Finally, we discuss how CARE opens new possibilities in how security groups can collaborate, governments can foster effective security practices, and insurance companies can more accurately identify risk through the secure and distributed ledger.

Together Skald and CARE provide the architecture and model performing large-scale investigations against computer security threats and collaborate these findings with peers. The next chapter describes the prototype created from these theoretical foundations.

Chapter 5

Prototyping the Concepts

"Theory without practice cannot survive and dies as quickly as it lives."

Leonardo da Vinci

SKALD and CARE provide the architectural blueprints for how to design a large-scale and collaborative analytic system that supports investigations across the Intelligence Cycle. In this chapter, we will introduce a working prototype for these concepts, named Holmes Processing. In our description, we discuss our design decisions, the key features for the components, and outline a subset of the included analytic *Service*s. Furthermore, we will introduce other relevant prototype and analytic work we have performed. Specifically, an advanced VMI platform for efficient and large-scale dynamic analysis called Drakvuf.

5.1. Introduction

As discussed in the previous chapters, computer security tools are disjointed and do not support an end-to-end analytic process. While efforts have been attempted to fuse these tools together, they fall short because of a tightly coupled and monolithic design. Furthermore, these efforts remain stove-piped and only support a single activity in the Intelligence Cycle. This hinders the analysts' ability to derive meaning from large sets of artifacts in a reasonable time. More troubling though is that the current situation forces a reactive defensive posture and one that cannot evolve with the latest trends in malicious activity.

We developed SKALD and CARE to create the concepts needed to shift the current paradigm in computer security. However, we created the Holmes Processing prototype to develop a working system that realizes these concepts and help iterate over our concepts using real-world examples. This proved to be a boon in refining SKALD and CARE. More interestingly though, it allowed us to explore advanced engineering concepts such as concurrent programming, large-scale architectures, and advanced analytic methods. Furthermore, developing the Holmes Processing showed that our concepts are valid through its ability to empower our research and perform real investigations against sophisticated and complex (malicious) actions. Adding validity to our claims, we have been invited to speak about Holmes Processing and our derived work at numerous highly acclaimed venues, such as Black Hat USA, Microsoft DCC, RSA USA, Hacktivity, and DARPA [2–8].

In this chapter, we will present the architecture for Holmes Processing and describe its key features. We will highlight how Holmes Processing helps break down an investigation according to the Intelligence Cycle and enables analysts to view the derived artifacts as part of a collective whole through the DIKW model. Thus, Holmes Processing fuses together analytic specialties and empowers teams to collaborate. We will also present how the design of Holmes Processing supports analysis across extremely large datasets, remains flexible to incorporate changes, and is resilient to failures.

The Holmes Processing prototype makes the following main contributions:

- We present a working prototype for large-scale analysis across the Intelligence Cycle.
- We explain the techniques used to make our prototype highly scalable, easily incorporate new analytic methods, and remain resilient to failures.
- We detail the advanced engineering concepts that are incorporated into the design of Holmes Processing.
- We present the analytic *Service*s we created for enabling real computer security investigations.
- We document our lessons learned to empower future efforts.

5.2. Architectural Overview

The Holmes Processing architecture is a "loosely coupled" microservices design based on the concepts of Skald and CARE. Similar to Skald, it is composed of three main components: *Transport*, *Planner*s, and *Service*s. Together these components create a system that can perform large-scale analysis of security artifacts and empower teams of specialists to collaborate while working through the process of the Intelligence Cycle.

The design of Holmes Processing creates a system that can scale, be resilient to failure, and evolve. While there are multiple techniques used to achieve these goals, these are afforded to the system due to its "loosely coupled" and distributed microservices design. For instance, regarding scaling, the Holmes Processing's design allows any component to be easily distributed. Thus, multiple *Planner*s of the same type can live in the system and work together in parallel. Moreover, this pattern exists for the *Transport* and *Service* components. In fact, we regularly spawn multiple *Planner*s and *Service*s of the same type when operating Holmes Processing. Resiliency on the other hand is primarily achieved because key parts of the system are “loosely coupled” and not reliant on other aspects. As such, failures in any part of the system do not propagate. Additionally, the replication of components and orchestration of tasking to these components by their *Planner* allows the system to pick up tasking where it left off or automatically reissue tasking when recovery is not possible. In case of a pathological failure, the *Planner* is able to adjust the tasking, resubmit the tasking to a specialized component that can handle the failure, or ship the tasking to a failure queue for human intervention. Furthermore, this design allows the system to evolve. Specifically, the “loosely coupled” design and abstraction from the technology executing work allows parts of the system to change without requiring adjustments to other parts of the system.

In this section, we will further explore the architecture of Holmes Processing, identify where it fits in the Intelligence Cycle, and discuss our key design considerations.

5.3. Planners and Services

Each project in Holmes Processing is in essence a SKALD *Planner* along with their relevant *Service*s. These projects are designed to encapsulate a core function of the Intelligence Cycle while also providing the foundations for creating an artifact based computer security analytic system. While each project can be used individually, they are designed to be combined to create a holistic system that can perform and investigation according to the Intelligence Cycle. These projects are Holmes-Gateway, Holmes-Totem, Holmes-Totem-Dynamic, Holmes-Storage, Holmes-Analytics, and Holmes-Presentation. In the following, we will outline the goals of each project and highlight key features.

5.3.1. Holmes-Gateway

Holmes-Gateway provides the foundations needed for supporting the collection step of the Intelligence Cycle. As such, Holmes-Gateway focuses on managing the ingestion of artifacts and taskings into the Holmes Processing system while allowing multiple individuals to work together by providing the foundation for access control and user segregation. In total, its primary purpose is to provide a single interface for analysts to submit artifacts and taskings while preventing analysts from directly interfacing with other components of the system, except for Holmes-Presentation. The other purpose of Holmes-Gateway is to orchestrate the execution of *Service*s that ease the burden on the analysts and provide key functionality. Together these *Service*s:

- Authenticate users and ensure they have the appropriate permissions for their request.
- Validate that submissions are properly formatted.
- Provide an interface for collaborating with partners.

- Attach meta-information about the submission and who submitted the tasking.
- Automate the tasking of analytics for newly submitted artifacts.
- Push packaged artifacts and taskings to the appropriate *Transport* channel.

We developed Holmes-Gateway using the Go programming language. Go was selected for a few reasons. The first reason was because we wanted to explore concurrency with the CSP model. Since Go is one of the few languages based upon CSP, this was a natural choice. The second reason is that Go is a simple language with a small learning curve. As such, the *Planner* could be developed and maintained by a cadre of graduate assistance. For the most part, we found implementing the *Planner* in Go to be fairly straightforward and were impressed with its simplicity. Additionally, *Service*s in Holmes-Gateway were created as separate programming classes and not as true microservices. This is a partial deviation from the goal in SKALD to have all *Service*s function as separate entities. However, we chose to do this because the *Service*s were lightweight and not overly complex. For instance, these *Service*s perform authentication, validate that submitted tasking is properly formatted, attach submission details (which analysts, date and time, source, etc), and extract meta-information about submitted *Data* artifacts, and allow the creation of rules for automated tasking based on mime-type. As such, we worried that the overhead from providing a more strict microservices design would negatively impact performance.

Of note, the current implementation for collaboration more closely follows the design outlined in SKALD. As such, it is currently based on ACL rules and configuration details that describe what analysts and partnered corporations can task. While we originally included a more robust collaboration method that achieved some of the goals outlined in CARE, this was scratched from the current implementation. We did this because we found the method did not overcome many of the challenges faced with sharing. This is what led us to research a better way forward and create the concepts that evolved into the CARE model. In the future, we plan incorporating additional *Service*s that achieve goals of CARE.

5.3.2. Holmes-Totem and Holmes-Totem-Dynamic

Holmes Processing implements the INVESTIGATION *Planner* as two separate projects, Holmes-Totem and Holmes-Totem-Dynamic, to realize Intelligence Cycle's Process and Exploitation step. Together, these projects leverage *Data* based analytic methods to transform *Data* into *Information*. Specifically, by orchestrating the tasking of *Service*s that perform static analysis, dynamic analysis, and gathering *Information* from 3rd party sources. Of additional note, both projects are artifact agnostic and support the analysis of any type of file or identifier (such as a domain name or IP address). The only requirement for extracting *Information* from an artifact is that the *Planner* has a *Service* that is capable of performing the analytic operation.

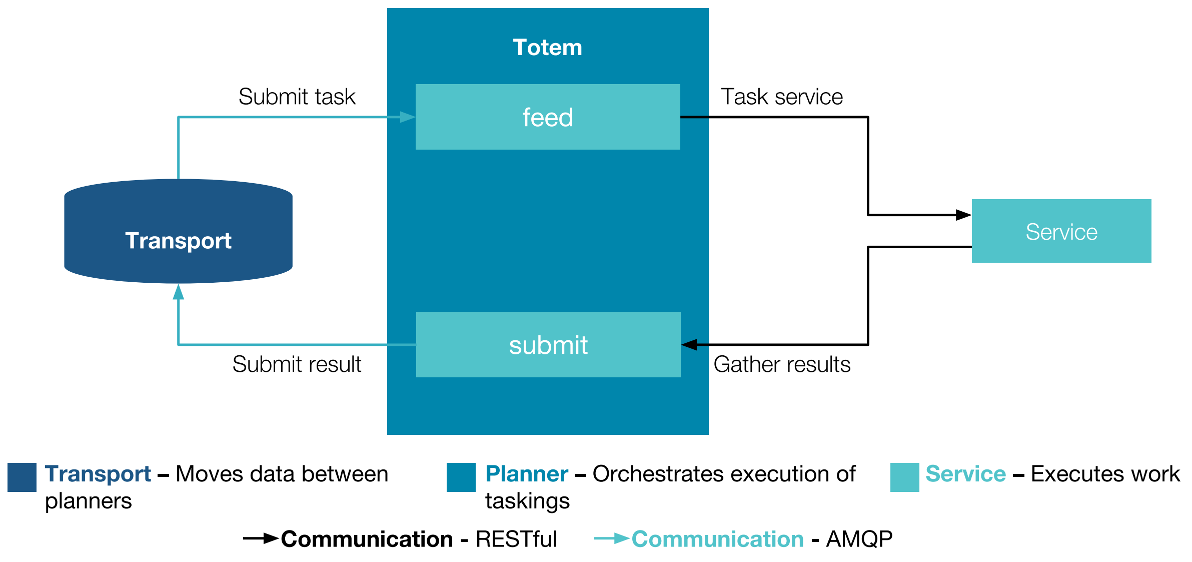

Figure 5.1.: Architecture of Holmes-Totem

As noted above, we split the development efforts of the INVESTIGATION *Planner* into two separate projects. The first project, Holmes-Totem was created to optimize the execution of *Services* that display a deterministic run-time or have a short execution time. In practice, *Services* in Holmes-Totem predominately execute static analysis and gathering *Information* from 3rd party providers. The second project, Holmes-Totem-Dynamic, focuses on the challenges of executing *Services* that have an indeterministic run-time or take a long time to complete, as is typically seen with dynamic analysis and interfacing with 3rd party providers that perform a function on our behalf; for example, when submitting a new file to VirusTotal for analysis. Nevertheless, this split was done mainly for practical reasons that are related to academic research and expediency. Specifically, Holmes-Totem allowed us to study the Actor Model, while Holmes-Totem-Dynamic enabled us to explore the CSP model and investigate techniques for managing *Services* with an indeterministic run-time.

Holmes-Totem and Holmes-Totem-Dynamic have two main differences. The first difference is that Holmes-Totem was developed using the Scala programming language and the Akka toolkit for concurrency using the Actor Model, while Holmes-Totem-Dynamic was created using the Go programming language and leverages its native support for CSP. The other major difference is in how Holmes-Totem and Holmes-Totem-Dynamic task and monitor the execution of a *Service*. In Holmes-Totem, tasking feeds directly to a *Service*, and the *Planner* leaves a connection open to receive the results, as shown in Figure 5.1. This paradigm improves performance by reducing network latency by not requiring that a communication path be reopened and removing the wait time caused by a polling operation to check if the analytic task being performed by a *Service* is complete. However, we found this method to be inefficient when dealing

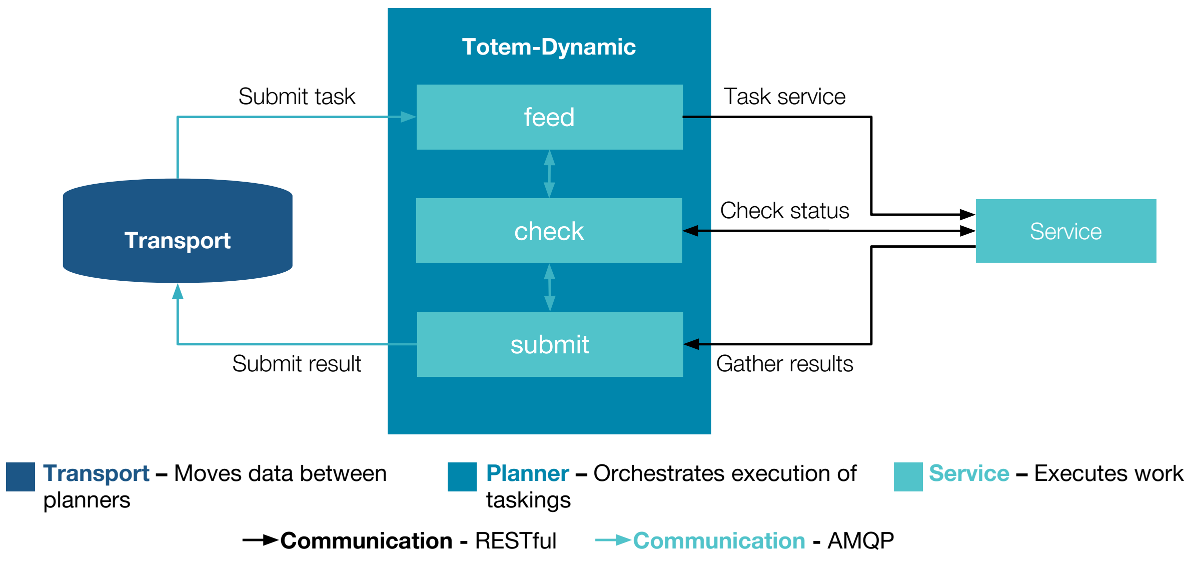

Figure 5.2.: Architecture of Holmes-Totem-Dynamic

with *Services* that are long-running or a have an indeterministic execution time. This is because the *Planner* leaves the communication path open and is unable to move on to the next task or reallocate resources. To overcome this challenge, Holmes-Totem-Dynamic incorporates an additional step, named check, when executing a *Service* and uses AMQP for communicating between each step, as shown in Figure 5.2. As such, Holmes-Totem-Dynamic first submits a task to a *Service* using the feed function and then submits a notification message via AMQP. The next step, check, receives the message and begins a polling operation to monitor the execution of the *Service* to identify when it is finished. When it has identified that a *Service* has completed its task, it sends a notification message, again via AMQP, that the *Service* has finished and where to receive the results. The final step, submit, receives the AMQP message and gathers the results from the *Service* and performs any post-processing task. Unlike Holmes-Totem, this process provides the benefit of isolating each step of the *Service* to complete its execution and staging the results due to the use of queuing thought AMQP messaging. Additionally, the reliance on AMQP allows Holmes-Totem-Dynamic to record the execution state of a *Service* and thereby allows Holmes-Totem-Dynamic to resume the last state of execution of a *Service* in cases of a *Planner* failure. As an additional benefit, this method allows Holmes-Totem-Dynamic to distribute the execution steps between any Holmes-Totem-Dynamic *Planner*. As a drawback, this method increases the complexity of executing and creating a *Service*. With the added downside of imposing performance penalties against short running tasks due to network latency.

In both Holmes-Totem and Holmes-Totem-Dynamic, *Services* contain the parts of the *Planner* that perform the actual analytic operation for extracting *Information* from

Name	Type	Description
ASNMeta	3rd party	Identifies the registered Autonomous System Number (ASN) for an IP address
cfg	Static analysis	Extracts the Control Flow Graph (CFG) from a binary using Nucleus [161]
cfgangr	Static analysis	Extracts the CFG from a binary using Angr [162]
Cuckoo	Dynamic analysis	Detonates a binary or Uniform Resource Locator (URL) using Cuckoo Sandbox
DNSMeta	3rd party	Identifies the DNS records and the true Time to Live (TTL) for a domain
Drakvuf	Dynamic analysis	Detonates a binary using the DRAKVUF VMI analysis system
goGadget	Static analysis	Extracts the gadgets from a binary using ROPGadget [163]
objDump	Static analysis	Returns details of a binary derived from objdump
PassiveTotal	3rd party	Returns historic DNS records from Passive Total
PDFParse	Static analysis	Extracts the elements of a Portable Document Format (PDF) file [164]
PEiD	Static analysis	Identifies the compiler, packer, and other build tools used to create a binary
PEInfo	Static analysis	Extracts the PE32 headers from a PE32 file using PEInfo [165]
PEMeta	Static analysis	Extracts the PE32 headers from a PE32 file
Richheader	Static analysis	Extracts the Rich Header from a PE32 file
Shodan	3rd party	Returns information about an IP from Shodan
VirusTotal	3rd party	Returns information about an binary from VirusTotal
Yara	Static analysis	Performs Yara signature matching with a default rule pack or custom rule
ZipMeta	Static analysis	Extracts the meta information contained in a zip file

Table 5.1.: Included analytic *Service*s in Holmes-Totem and Holmes-Totem-Dynamic

Data. To date, we have created a handful of private *Service*s and eighteen public *Service*s that are detailed in Table 5.1. While some *Service*s are straightforward, many of them were complex undertakings and were created to support the efforts of separate research projects. For instance, Drakvuf was developed by Tamas Lengyel as part of his PhD and co-authored by the author of this dissertation. This work was developed to provide a scalable and stealthy method for dynamic analysis that uses an agentless approach with VMI; in turn, allowing analysts to more accurately record how an artifact being analyzed behaves. In addition, PEMeta *Service* was made to create a new method for extracting the headers of a PE32 file that was ten times faster than PEInfo while also providing a more accurate extraction of these headers. This *Service* was used to gather the *Information* used to enable the research of multiple knowledge based analytics using machine learning. Lastly, the Richheader *Service* is the first publicly available method that accurately extracts this obscure and obfuscated PE32 header, while also identifying the meaning behind the features. This work empowers teams of analysts to perform rapid triage of PE32 files, identify binaries that are similar in nature, and fingerprint the build environment. The Rich Header work will be further discussed in Chapter 6.

While the Holmes-Totem and Holmes-Totem-Dynamic projects include these *Service*s as part of the repository, it is important to note a few key features. The first feature is that both projects provide support for *Service*s to be locally or remotely deployed. That said, in cases where the *Service* is locally deployed, both systems will use a RAM Disk to optimize the transmission of the artifact to the tasked *Service*s to reduce network load. This can provide significant performance gains when an analyst desires to execute multiple *Service*s against the same artifact. The second feature is that there are no hard requirements for which communication method is used to communicate between a *Planner* and *Service*. While the *Service*s packaged by both projects used a RESTful interface for communication, this is not necessary and in both projects the incorporation of a different protocol is painless. For instance, in Holmes-Totem, the Scala code file contains the logic for how the project communicates with the *Service*. As such, replacing the RESTful communication method with protobuf would only require changes to this *Service* specific file. The last feature is that both projects encapsulate *Service*s as Docker containers and use Docker Compose for container management. This is to minimize the dependency management of the system while also providing benefits from a practical and security perspective. With respect to scaling, this allowed the quick spawning of replica *Service*s when a *Service* became overburdened. Furthermore, when faced with a misbehaving *Service*, the *Service* could be restarted with a single command. Lastly, the containerization provided further robustness and security because its execution and failures were encapsulated and did not easily propagate to other parts of the system.

It is worthwhile to state that we originally attempted to extend CRITs to support the goals of the SKALD INVESTIGATION *Planner*. To this end, we significantly enhanced CRITs performance and submitted our patches to the CRITs project. However, we encountered numerous challenges that forced us to abandon this task. The first issue was that CRITs' core used a considerable amount of resources when executing a CRITs *Service*. Through our investigation, and the support we received from the CRITs' lead developers, it was determined that the issue was caused by CRITs' reliance on spawning *Service*s through the Apache Web Server and the inefficiencies of Python. Unfortunately, this limited our ability to submit large volumes of taskings because of physical resource constraints. Another challenge was that if a CRITs *Service* failed, the error would replicate through the entire system or consume resources that could not be reallocated in time. This became an intractable issue and one that caused a hard limit on executing more than fifty thousand tasks at the same time without significant cool-down periods. In practice, we found that we were only able to run one thousand *Service*s every hour. The final major challenge was that we could not scale CRITs beyond instantiating additional database shards. Unfortunately, this could not be overcome with simple patches. This is because CRITs monolithic design with "tightly coupled" *Service*s required that the entire core be rewritten. Deploying a load balancer in front of multiple CRITs' instances and creating an orchestration engine that recovered the instances when they failed would have alleviated the problem. However, the solution is a brute force approach that will reappear as the volume of artifacts and tasks increases. Furthermore, the solution would have done nothing to overcome the issue of retrieving artifacts. Thankfully, the CRITs engineers were helpful and even reviewed the pre-publication version of SKALD to help refine the design.

5.3.3. Holmes-Storage

Holmes-Storage realizes the goals of the STORAGE *Planner* that are outlined by SKALD and CARE. At its primary level, Holmes-Storage provides an abstraction layer for the

Name	Type	Description
Cassandra	textural	Distributed non-relational database inspired by Amazon's Dynamo and Google's BigTable
CRITs	textural and Object	Legacy support for the popular analytic triage solution by MITRE
Flat file	Object	Hosts files located in a folder
MongoDB	textural	Distributed document-oriented NoSQL database
S3	Object	Object-store databases compatible with Amazon's S3 protocol

Table 5.2.: Supported database back-ends in Holmes-Storage

interactions with its database *Service*s and maintains a central repository for artifacts. This repository segregates artifacts according to the DIKW model and provides optimal schemes, across multiple different databases, that support large-scale analytics, internal queries from other Holmes Processing components, and user interactions. In doing so, Holmes-Storage provides a support function for Holmes Processing that helps facilitate multiple steps of the Intelligence Cycle. These steps are namely Processing and Exploitation by organizing artifacts in standardized formation and Dissemination and Integration by adding newly derives artifacts into the systems collective whole. While not an entire list, Holmes-Storage provides the following capabilities:

- Automatic fetching analytic results via AMQP.
- Receiving new artifacts via a RESTful API.
- Providing a RESTful API for artifact retrieval.
- Managing Holmes Processing's deployment-specific configuration details.
- Segregating artifacts according to internal policy requirements.
- De-duplicating *Data* artifacts while appending details to their textural records.
- Optimizing the schemes for large-scale analytics, user interactions, and internal queries.
- Enriching artifacts with meta-information that support usability, de-duplication, the ACL model, and archiving.
- Compressing artifacts to minimize network overhead.

We developed Holmes-Storage using the Go programming language and design it to be a reference implementation for a Skald Storage *Planner* while being optimized for our use case. However, our primary concern with creating Holmes-Storage was to overcome the challenges of locking into one database technology to provide flexibility in how artifacts were stored and allow Holmes Processing to be attached to legacy systems. This challenge was overcome by providing an abstraction layer that could select various storage *Service*s for different database back-ends. At the time of this writing, five different database types are supported, as shown in Table 5.2. However, it is important to note, that any of these back-ends can be used or combined together. For instance, a user of Holmes-Storage can use a database specialized on the storage of objects for large binary blobs while delegating the storage of analytic results in a database that is better suited for text, such as NoSQL. However, as we were primarily concerned with efficient large-scale analytics, we eventually focused our efforts with S3 and Cassandra and have heavily optimized their interactions. As a testament to to this paradigm, our internal deployment regularly contains 10s of millions of *Data* artifacts and five times that volume in extracted *Information* and generated *Knowledge*. However, even with this volume we are able to receive the results of our queries in near real-time and support intensive machine learning and statistical operations across the entire dataset in under a minute.

MongoDB was our first choice as a database back-end. This was because of its simplicity with managing textural artifacts and we already had a large MongoDB cluster. However, we encountered performance issues when we started to reach millions of stored artifacts and we found it cumbersome to use with machine learning engines. While we could have continued to scale the database *Service*, this would increase the cost beyond a university budget. Our first solution allowed Holmes-Storage to use an S3 compatible database as an object store. We found this solution to be highly efficient, allowing the system to easily leverage the cloud, and significantly increasing our performance. Unfortunately, we still had performance issues when dealing with artifacts further up the DIKW chain that were more textural in nature. We explored a few different solutions but eventually selected Cassandra. Our reasons for doing this is because we found its ability to scale while remaining to be highly performant to be down right remarkable. Furthermore, it supported complex machine learning queries with remarkable efficiency. That all said, we still expanded our back-end support to show the power in the Holmes-Storage design and improved ease of use. Specifically, we created the ability to support a flat-file style object storage for small testing deployment and added legacy support for CRITs back-end. We also experimented with using Apache Hadoop HDFS but eventually dropped support because S3 was easier to use and we could not identify any performance gains. Of note, Holmes-Storage is not limited to only using these database back-ends. Expanding support to include other technologies would only require a class file that includes the logic for how Holmes-Storage should interact with the *Service*. Thus, if a user of Holmes Processing desires to incorporate an Elastic style index for searching or would like to incorporate AMQP notifications for stream processing, the amount of effort required should not exceed a few days.

We optimized the system for large-scale analytics and easy of use by mainly focusing on the Cassandra scheme design. Our first attempt was to provide support for Cassandra SASI indexing. This creates an index for searching within the Cassandra ecosystem. While initially we found this to be highly effective, we ran into trouble when our datasets expanded in size. Unfortunately, this identified a bug in the Cassandra SASI implementation and we were advised by the lead developers of Cassandra to drop support until they can identify a way forward. Fortunately, this effort led us to Material Views. Under this storage paradigm, the database creates multiple tables that contain the same data but under different scheme designs. This allowed us to create separate tables for artifacts that were optimized for machine learning queries, searching, retrieving specific artifacts, and standard user queries. This proved to be highly efficient from both a storage standpoint and drastically reduced our query time. The only remaining optimization we implemented was to compress stored textural artifacts to reduce the volume that needed to transit the network. While seemingly trivial, this resulted in massive benefits. Specifically, it reduced the time it took for a *Knowledge* based analytic method using link analysis from two-and-a-half years to minutes. Collectively, these optimizations are a major achievement and have allowed our analysts to perform investigations and research that would normally not be achievable.

In the future, we plan to expand Holmes-Storage to increase functionality and automate administrative features. Regarding functionality, we plan to add support for search engines technology such as Elastic or Solr. This is because while Apache Spark is able to accomplish the goals of searching, it requires support from an INTERROGATION style *Planner*. With respect to administrative features, we intend to provide three core aspects. The first goal is to support maintaining the database *Service*s by automating administrative tasks such as repair and compaction. The second goal is to embed an optimized proxy for communicating with database clusters. The third goal is to allow Holmes-Storage to automatically scale and de-scale the database back-ends based on the load. As a more research-oriented expansion, we also desire to investigate ways to automate the scheme design to improve the performance of Holmes-Storage based on usage.

5.3.4. Holmes-Analytics

Holmes-Analytics has two primary missions that are akin to the Production step in the Intelligence Cycle. The first mission is to orchestrate *Information* based analytic methods to enable the creation of *Knowledge*. The second mission it to present an analyst with various representations of the artifacts contained in Holmes Processing to support their investigations and create *Wisdom*. Together Holmes-Analytics supports computer security analysts with understanding the copious volumes of artifacts they are presented with and making more effective judgments and mitigation actions. Furthermore, this allows the analysts to drive the investigations because they are not restricted to only being able to work with small sets of artifacts and can manipulate the viewing of artifacts as best suits the individual.

The first mission is achieved through the scheduling of two categories of *Service*s in Holmes-Analytics, analytic engine and analytic service. Together these perform advanced analytics using techniques such as machine learning, link analysis, and statistics to make sense out of sets of *Information*. These *Service*s have been split into two categories because advanced analytic techniques often use multiple algorithms and analytic engines

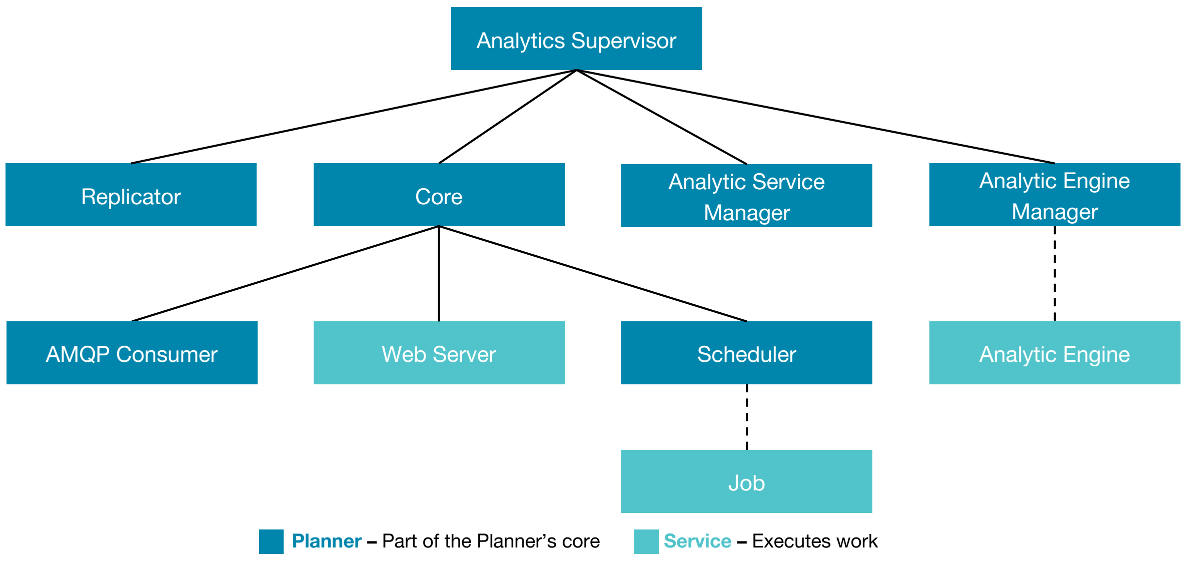

Figure 5.3.: Architecture for Holmes-Analytics

to achieve their goals. As a simple example, an analytic work-flow could leverage the Apache Spark analytic engine to gather *Information*, perform feature extraction, and vectorize the results while using Google's TensorFlow for performing clustering operations. To this end, our internal deployment of Holmes Processing provides analytic engine *Service*s for Apache Spark, Spark GraphX, TensorFlow, and multiple engines that fall under the Apache Hadoop family, while our analytic *Service* implements the logic required for the analysis operation; such as, performing clustering, similarity matching, automatic link analysis, and gathering statistics from the artifacts.

The second mission is handled through the orchestration of *Service*s that focus on providing access to artifacts through an API or display engine. This is a deliberate separation because the process of manipulating how artifacts are displayed is a vital analytic step in its own right. For example, understanding the results of graph analysis and manipulating how the graph is displayed can provide vital insights for a human analyst. Additionally, we strongly felt that an analyst should not be limited to how a tool says an artifact should be rendered and should be free to adjust this rendering as she sees fit, including using other tools that are outside of Holmes Processing. At this time, we have implemented *Service*s that empower manipulating artifacts through Apache Zeppelin, a website for viewing artifacts and pivoting on results, and an API for interfacing with custom scripts and 3rd party analytic tools.

Our initial attempts for creating Holmes-Analytics was centered on Apache Hadoop and web-based front-ends. However, this became intractable due to the steep learning curve it imposed on the analysts and the DevOps nightmare caused by maintaining the multiple analytic engines. Through multiple iterations—and frustrations—we settled on a design that leveraged Scala and the Akka toolkit to achieve our desired result. In particular, Scala with Akka allowed us to handle asynchronous requests and support multi-tenancy with a simple design. This model also allowed us to remove the analysts from the details for the infrastructure and let them focus on their analytic algorithm. As shown in Figure 5.3, Holmes-Analytics is composed of nine types of actors:

- **Analytic Supervisor** - Acts as the main overseer and initializes the Core, Analytic Engine Manager, and Analytic Service Manager.
- **Core** - Oversees the execution of the Web Server, AMQP Consumer, and Scheduler.
- **Web Server** - Provides a RESTful API and web front-ends.
- **AMQP Consumer** - Enables AMQP based communications.
- **Scheduler** - Schedules the creation of a Job and manages its execution.
- **Job** - Enables the execution of an analytic task.
- **Analytic Engine Manager** - Orchestrates the execution of an Analytic Engine *Service*.
- **Analytic Engine** - Provides the logic needed for interacting with an analytic engine *Service* and encapsulates its execution.
- **Analytic Service Manager** - Orchestrates the execution of the logic for an analytic *Service*.

We contemplated incorporating frameworks such as Apache Livy and Apache Beam to accomplish the goals of Holmes-Analytics. The rationale was that Apache Livy has promise for enabling remote queries and supporting multi-tenancy through an API, while Apache Beam appears ideal for creating algorithms that are analytic engine agnostic. Unfortunately, these products were too immature at the time of our development. As such, Holmes-Analytics implements its own methods for multi-tenancy, remote queries through AMQP and a RESTful API, and is abstracting the algorithm logic from the analytic engine. However, we plan to monitor the progress of Apache Beam and Apache Livy and incorporate these projects when it makes sense.

5.3.5. Presentation

Holmes-Presentation is a simple but necessary *Planner*. Its single purpose is to provide a unified point of entry into Holmes Processing and manage the flow of communications from outside the system, thus helping to facilitate the Dissemination and Integration step of the Intelligence Cycle. To this end, we implemented Holmes-Presentation as an HAProxy server that incorporates authentication. As such, it orchestrates the connections from outside the Holmes Processing system and provides a layer of abstraction from internal components.

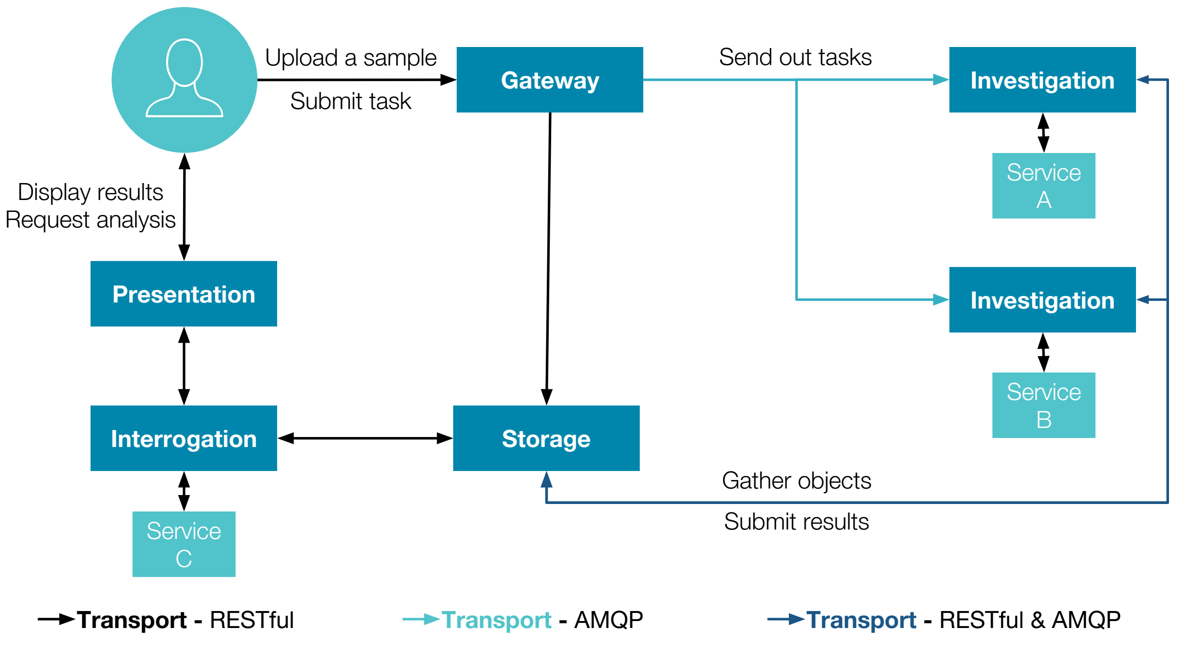

Figure 5.4.: Communication Flow in Holmes Processing

5.4. Transport

The *Transport* component in Holmes Processing manages the communication between other components in the system. Similar to SKALD, this has been implemented more as a concept and a set of best practices instead of an independent project. As such, the other *Planner*s in Holmes Processing will automatically identify how to effectively communicate and configure the servers providing the functionality. In effect, the *Transport* component is based on a RESTful API and the AMQP protocol. We selected these two methods because they are easily understood, flexible, and have robust enterprise solutions. For instance, the RESTful protocol is well suited for facilitating direct interactions. However, when performance, scheduling, and horizontal scaling is required, AMQP provides an efficient lightweight transport mechanism that inherently queues tasking.

The specific communication methods between components in Holmes Processing are illustrated in Figure 5.4. As can be determined, the managing of large-scale tasking is effectively delegated to AMQP. For instance, Holmes-Gateway provides tasking to Holmes-Totem and Holmes-Totem-Dynamic using AMQP. This allows Holmes Processing to receive massive amounts of *Data* based analytic taskings while enabling an indeterminate number of Holmes-Totem and Holmes-Totem-Dynamic *Planner*s to be instantiated. When a new INVESTIGATION *Planner* is instantiated, Holmes Processing will automatically register these *Planner*s with the AMQP server and the *Planner*s will begin to pull the tasking from the AMQP server. In practice, this has allowed us to scale the analysis of *Data* to over a hundred INVESTIGATION *Planner*s with minimal effort and overhead.

Furthermore, using AMQP allows Holmes-Storage to manage the burden of receiving massive quantities of analytic results without imposing delays or overwhelming the system. This is done by allowing Holmes-Storage to pull analytic results, provided by Holmes-Totem or Holmes-Totem-Dynamic, in bulk from the AMQP server. Thus, the influx of network connectivity directly against Holmes-Storage is minimized while allowing additional Holmes-Storage *Planner*s to be instantiated and enabling the database technologies to manage their own workload. That said, when direct asynchronous interaction is required, such as gathering a *Data* artifact or responding to an API query, the RESTful protocol is better suited.

Our internal deployment of Holmes Processing currently uses Python's Tornado, Go's HTTP library, and Scala HTTP to implement a RESTful API. These were selected because they are easy of use and exhibit high performance characteristics. On the other hand, we implemented the AMQP protocol and server using RabbitMQ. This was selected because it is simple to deploy and we have found no reason to look for an alternative. While the approach of using AMQP has allowed the components of Holmes Processing to efficiently communicate and remove networking bottlenecks, it does come at a cost. Namely, it increased the complexity of a new deployment and requires an administrator to be concerned with an additional server. As Holmes Processing evolves, we plan to research the feasibility of replacing AMQP with a more tightly coupled design such as Akka Cluster. In theory, this will allow the *Planner*s of Holmes Processing to communicate with the benefits of AMQP without requiring an external AMQP-compliant server to be deployed. Adding a further benefit, this will allow *Transport* to streamline the ACL implementation between all *Planner*s and allow the addition of logic to automatically scale and manage Holmes Processing *Planner*s.

5.5. Lessons Learned

We learned many valuable lessons through the process of implementing Holmes Processing and leveraging its capabilities. In this section, we will discuss a few highlights.

5.5.1. Languages

We originally used Python for creating the INVESTIGATION *Planner*, Holmes-Totem, and a number of extraction *Service*s. While this simplified development and allowed for a clear comparison with CRITs, the limitations of the Python language created many challenges.

One critical issue stemmed from Python's immature support for multiprocessing. When dealing with extremely large tasks, this caused Holmes Processing to require an excessive amount of resources and we identified flaws in execution due to memory caps imposed by the Python interpreter. While we originally worked to optimize our Python code base, these efforts only alleviated the immediate problems and the overall challenges persisted.

We finally solved the challenges by implementing the Investigation *Planner* using a language designed for concurrent execution, Scala with the Akka toolkit. Another challenge we identified was that *Service*s written in Python displayed clear performance penalties. While this was more or less fine for simple *Service*s, the penalty to execution time in intensive operations greatly slowed down the system when dealing with large task loads.

5.5.2. Actor versus CSP

The CSP and Actor Model have intensive followings with clear lines of loyalty. To explore this, we leveraged both models when creating *Planner*s in Holmes Processing. We found that both models provided major benefits for concurrent execution. In particular, we found that CSP performed remarkably well for Holmes-Storage and Holmes-Gateway and had a reduced development learning curve. However, we found that the Actor Model's asynchronous nature provided benefits when orchestrating the execution of multiple *Service*s. Namely, toolkits designed around the Actor Model provided native support for robust messaging. This allowed us to achieve our implementation goals, and some, with a smaller code base and less system complexity. Unfortunately, this caused a steeper development learning curve as it is more challenging to reason about. While we acknowledge that the noticed differences can be due to the languages and toolkits we selected, we found this pattern to exist in other implementations of these models. That said, we overall found that the Actor Model was well-suited to the challenges in performing an analysis of computer security artifacts.

5.5.3. The DIKW Model and Loosely Coupled Design

No design is perfect and will stand the test of time. We need to constantly seek to better our methods and leverage what we know to identify new and better ways forward. The constant evolution that occurs in computer security highlights this principle. Fortunately, we found that segregating artifacts according to the DIKW model and imposing a "loosely coupled" design made evolving the system and exploring new analytic methods an easy task. As an example, when leveraging Holmes Processing for researching new information based analytic techniques the process became simple and quick. We found that researchers began to only worry about how to tweak which features were being selected and how best to design their algorithm. This is because the *Information* already existed in the system and the time required to extract additional *Information* was no longer required. Furthermore, the "loosely coupled" design allowed the researchers to be ignorant in how the *Information* was extracted in the first place. In another example, we found that when issues arose in any *Service* the process for adjusting that *Service* was remarkably simple. In particular, the maintainer only needed to focus on the *Service* code and not how other parts of the system interacted. In practice, this allowed changes in the system to occur in days if not the same day.

5.6. Future Work

The work of a prototype is never done. In this section, we will discuss major efforts we plan to undertake in the near future. Specifically, how to improve the INVESTIGATION *Planner*, enhance system administration and management, and provide a robust ACL model.

5.6.1. Merge the Investigation Planners

The INVESTIGATION *Planner* for Holmes Processing is currently split into two projects, Holmes-Totem and Holmes-Totem-Dynamic. This was done to reduce the complexity required to design a *Planner* that can manage both the execution of *Service*s with a short and deterministic execution time while also supporting *Service*s with a long and indeterministic execution time. Furthermore, splitting the *Planner* into two parts also allowed the study of the Actor Model versus the CSP model. While this was adequate for research, this greatly increases the complexity of the system. Moving forward, we plan to merge these *Planner*s.

5.6.2. Deployment and System Management

Holmes Processing is a working prototype that has been used for complex investigations and academic research. However, it is still an academic prototype and simplifying the administration of the system was not a priority. To gain wider adoption, we plan to address this issue by easing the deployment and management of Holmes Processing. Specifically, by allowing automatic deployments using Kubernetes, Mesos DC/OS, and other DevOps tools. Furthermore, we plan to remove the requirement for Apache Hadoop when deploying Holmes-Analytics.

5.6.3. Improve ACL

Security artifacts have associated sensitivities and restrictions that require special handling. To accommodate these requirements, security analytic systems need to incorporate a design that supports isolation and an appropriate ACL model. The most basic requirement for an ACL model of this type is to segregate *Data* and associated artifacts by the source of the *Data* and/or the submitting user. While most existing analytic systems use this model, this model incurs analytic limitation and can expose sensitive details. For instance, this type of model creates an all-or-nothing situation for access. Specifically, there is no distinction between types of artifacts, where an artifact lives along the DIKW model, and analytic task. As such, an analyst focusing on generating *Knowledge* using machine learning would require full access to the source of the *Data* and all associated artifacts. Additionally, this model does not allow limited access to sensitive or rate limited analytic capabilities.

Holmes Processing only implements the foundations on which to build a robust ACL model. Namely, by allowing a fine-grained level of isolation between system components while providing flexibility in how artifacts are stored and attaching descriptive metadata. However, the prototype does not currently leverage these features beyond the traditional ACL model used in other systems. In future work, we plan to research how to replace the existing ACL. While the new model is still being researched, we currently envision the model to grant access based on a *User*, *Capability*, *Source*, and *Meta-tags*. As such, artifacts can be restricted based on their type, system capabilities or *Service*s can be protected and controlled, sensitive sources and operations are still guarded, and the execution of a *Service* does not require an analyst to have access to every artifact.

5.6.4. Providing Native Streaming Support

We heavily contemplated implementing stream processing in Holmes Processing. However, these efforts were always abandoned because streaming is more of a function that is better suited in a production system. Specifically, as we were focused on research and performing static investigations. As such, Holmes Processing did not require optimizing the continual collection and processing of new artifacts. That said, enabling stream processing is a straightforward task and one we desire to accomplish in the near future. An astute observer of the code base will notice that most of the functionally needed for streaming is implemented but not advertised.

5.7. Summary

This chapter introduced the Holmes Processing prototype that was developed from the concepts introduced in Chapters 3 and 4. Holmes Processing enables computer security investigations to work across the steps of the Intelligence Cycle and view the derived artifacts as part of a collective whole. As a result, analytic specialties are fused together and teams are able to effectively collaborate. Furthermore, the design of Holmes Processing allows the system to support extremely large datasets, remain flexible to incorporate changes, and be resilient to failures. For instance, the addition of new *Information* extraction *Service*s is painless and the developer is not burdened with details for how the entire system operates. Additionally, analysts working to generate *Knowledge* from *Information* can leverage the work from their peers and are not concerned with how the *Information* was created; all while allowing the analysts to work with a corpus of 10s of millions of artifacts in near real-time.

While the prototype is academic in nature and not a robust product, it has been used to empowered research and perform real investigation against sophisticated and complex actors [31–36]. Adding validity to our concepts, we have been invited to speak about Holmes Processing and our derived work at numerous highly acclaimed venues, such as Black Hat USA, Microsoft DCC, RSA USA, Hacktivity, and DARPA [2–8].

In the next chapter, we will begin to explore how to leverage Holmes Processing to perform a real investigation. We do this by presenting a case study that made a significant impact in the security community.

Chapter 6

Proving the Concept: PE32 Malware Triage and Similarity Matching

'Information is the oil of the 21st century, and analytics is the combustion engine."

Peter Sondergaard

The previous parts of this dissertation focused on the concepts that went into the design and architecture of a system that empowers analysts to function across the Intelligence Cycle and pool the collective knowledge and resources of the community together. This system enables teams to work together while performing large-scale analysis and complex investigations. In this chapter, we explore the paradigm shift these concepts empower; namely, the process and the philosophy behind how investigations can be performed and how individual skills can be leveraged together to tackle the challenges computer security analysts face.

We begin this discussion by using Holmes Processing and its application to the Intelligence Cycle to study a hidden PE32 field known as the Rich Header. During this process the team built upon each others unique skill sets and created an analytic pipeline that could perform rapid analysis across 964,816 malware samples. By working together and processing this large volume of *Data*, the team performed the first accurate assessment of the Rich Header. In doing so, they were able to uncovered the secrets of the Rich Header and identified how to extract the *Information* that Rich Header clandestinely contains. Though the identification of anomalies they observed during this process, the team then created a method that is able to quickly identify post-modified and obfuscated binaries through anomalies in the header. The team then worked to demonstrate how the now revealed *Information* contained in the Rich Header can derive *Knowledge* that can be leveraged to perform rapid triage across millions of samples, including packed and obfuscated binaries. This was done by exhibiting the Rich Header's utility in triage by presenting a proof of concept similarity matching algorithm which is solely based on the contents of the Rich Header. With this algorithm the demonstrate how the contents of the Rich Header can be used to identify similar malware, different versions of malware, and when malware has been built under different build environments; revealing potentially distinct actors. Furthermore, we are able to perform these operations in near real-time, less than 6.73 ms on commodity hardware across our studied samples. As a result, we establish that this little-studied header in the PE32 format is a valuable asset for security analysts and has a breadth of future potential. To put it differently, we demonstrate the power of using the concepts presented in this dissertation for performing complex computer security investigations.

6.1. Introduction

The sheer volume of malware samples that analysts have to contend with makes thorough analysis and understanding of every sample impractical. As a result, effective and timely triaging techniques are vital for analysts to make sense of the collective *Information* and focus their limited time on agglomerated tasks through uncovering commonalities and similar variants of malicious software. This in turn, allows analysis to better hone in their effort and avoid wasting costly cycles on previously analyzed or unrelated samples. Unfortunately, it is common practice for malware authors to design malware that hinders automated analysis and otherwise thwart triaging efforts; thereby allowing malware to operate under the radar for a longer period of time.

One of the common practices used in triaging samples is to leverage header information from the Portable Executable file format (PE32) [34, 166, 167]. This is primarily done as the derived *Knowledge* that can: (*i*) reveal how the executable was built and who built it, (*ii*) provide an understanding of what the executable does, and (*iii*) identify entry points for disclosing packed and obfuscated code. For example, when investigating the *Rustock* Rootkit, the PE32 Headers identified the location of the first deobfuscation routine [167]. Additionally, numerous clustering and similarity matching algorithms are often exclusively based on the *Information* derived from the PE32 file format [67, 68, 168, 169].

Unfortunately, malware authors are well aware of the valuable *Information* contained in the PE32 file format. As a result, they routinely take steps to strip or otherwise distort any useful *Information* from the PE32 format through packing binaries, adjusting compiler flags, and manually removing *Information* contained in the headers [170, 171]. While unpacking malware and performing manual reverse engineering can recover this useful *Information*, the process is extremely costly. As stated by Yan et al. [172], "Who has the time to reverse all the bytecodes given that security researchers are already preoccupied with a large backlog of malware?" Needless to say, in practice stripping the headers leaves little useful *Information* available for triage and analysis.

Fortunately for security analysis, the PE32 Header contains *Information* that is often poorly understood or simply hidden. In this work, we perform an in-depth study of one of these hidden attributes commonly known as the Rich Header. While rich in *Information*, this header is also common in malware, present in 71% of our random sample set, and is found in any PE32 file assembled by the Microsoft Linker. Through performing an in-depth investigation of the header using the Intelligence Cycle, we show how to properly dissect the *Information* and explain what the resulting *Information* means. We then use Holmes Processing to extract the *Information* from millions of malicious samples, which we gather from four distinct datasets. Leveraging what we learned in how to reveal this *Information*, we present proof of concept methods for deriving *Knowledge* that demonstrate the significant value the Rich Header can provide for triage.

In this chapter, we use Holmes Processing and apply the concept of the Intelligence Cycle to uncover the secrets of the Rich Header. We begin by providing the first accurate assessment of the Rich Header and detail how to extract its clandestine *Information*. We then present a series of statistical studies and describe two proof of concept methods for deriving *Knowledge* from the *Information* contained in the Rich Header. The first method allows for the rapid detection of post-modified and packed binaries through the identification of anomalies. The next method leverages *machine learning* to perform rapid and effective triage based solely on the values in the Rich Header's *@comp.id* field; specifically, the 516 unique *ProdID*, 29,460 distinct *ProdID* and *mCV* pairs, and their *Count* values that we have identified across 964,816 malicious samples. This method can identify *similar malware variants* and *build environments* in 6.73 ms across 964,816 malware samples using only a consumer grade laptop. As such, we prove that leveraging the *Information* contained in this often forgotten and overlooked aspect of the PE32 file format, called the Rich Header, establishes a major boon for performing analytic triage operations and opens the door for a plethora of future work. In total, we show that the Rich Header field is valuable in triage and can be a catalyst for past and future work. Through this work, we also demonstrate the paradigm shift to security that the the concepts of this dissertation creates.

In summary, we make the following main contributions:

- We present the first accurate and in-depth study of the Rich Header field and describe how to extract its *Information*.
- We demonstrate how anomalies in the Rich Header can identify 84% of the known packed malware samples.
- We present a proof of concept approach that utilizes machine learning techniques to derive *Knowledge* that identifies similar malware variants and build environments in near real-time, 6.73 ms, by only leveraging the Rich Header.
- We provide a case study that demonstrates collaborative analysis across the Intelligence Cycle and how these concepts propelled the Rich Header breakthroughs.

6.2. Background

To aid in understanding this work, this section provides a synopsis of the Portable Executable file format, commonly known as PE32, and compiler linking. For simplicity,

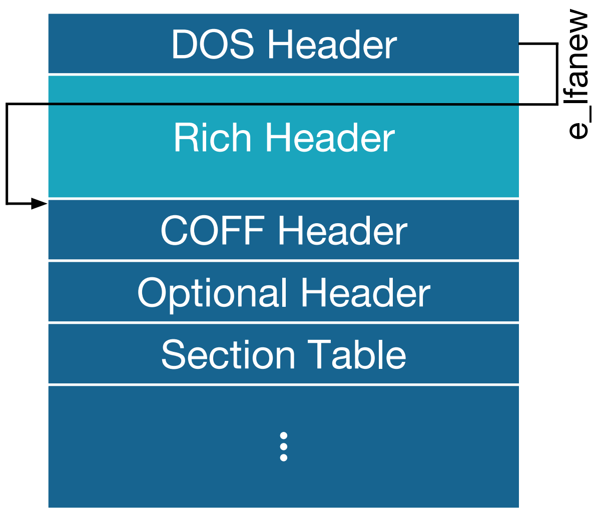

Figure 6.1.: High level view of the PE32 format

we focus our description on the 32-bit version of the PE32 file format. This is because the 64-bit version has strong parallels to the 32-bit version and the concepts are the same.

6.2.1. Portable Executable File Format Headers

The Portable Executable file format was introduced by Microsoft to provide a common format for executable files across the Windows Operating System family [53]. As such, the format is the primary standard used for shared libraries, binaries, and other types of executable code or images in Windows. The Portable Executable file format is also often called the Common Object File Format (*COFF*), *PE/COFF*, and PE32.

The PE32 format includes an MS-DOS stub for backwards compatibility, with the `e_lfanew` field pointing to the beginning of the COFF Header, as Figure 6.1 illustrates. The COFF Header, in turn, is followed by optional headers that control (among others) imports, exports, relocations, and segments [173]. Together, these headers contain valuable *Information* that enable program execution, identification, and debugging; including the base address of the image in virtual memory, the execution entry point, imported and exported symbols, and *Information* on the code and data sections that form the program itself.

The PE32 Header is openly documented by Microsoft and as such its internal mechanics are well understood by the development community [174]. However, as we will discuss in this work, the PE32 format does contain undocumented sections that have eluded understanding for over a decade.

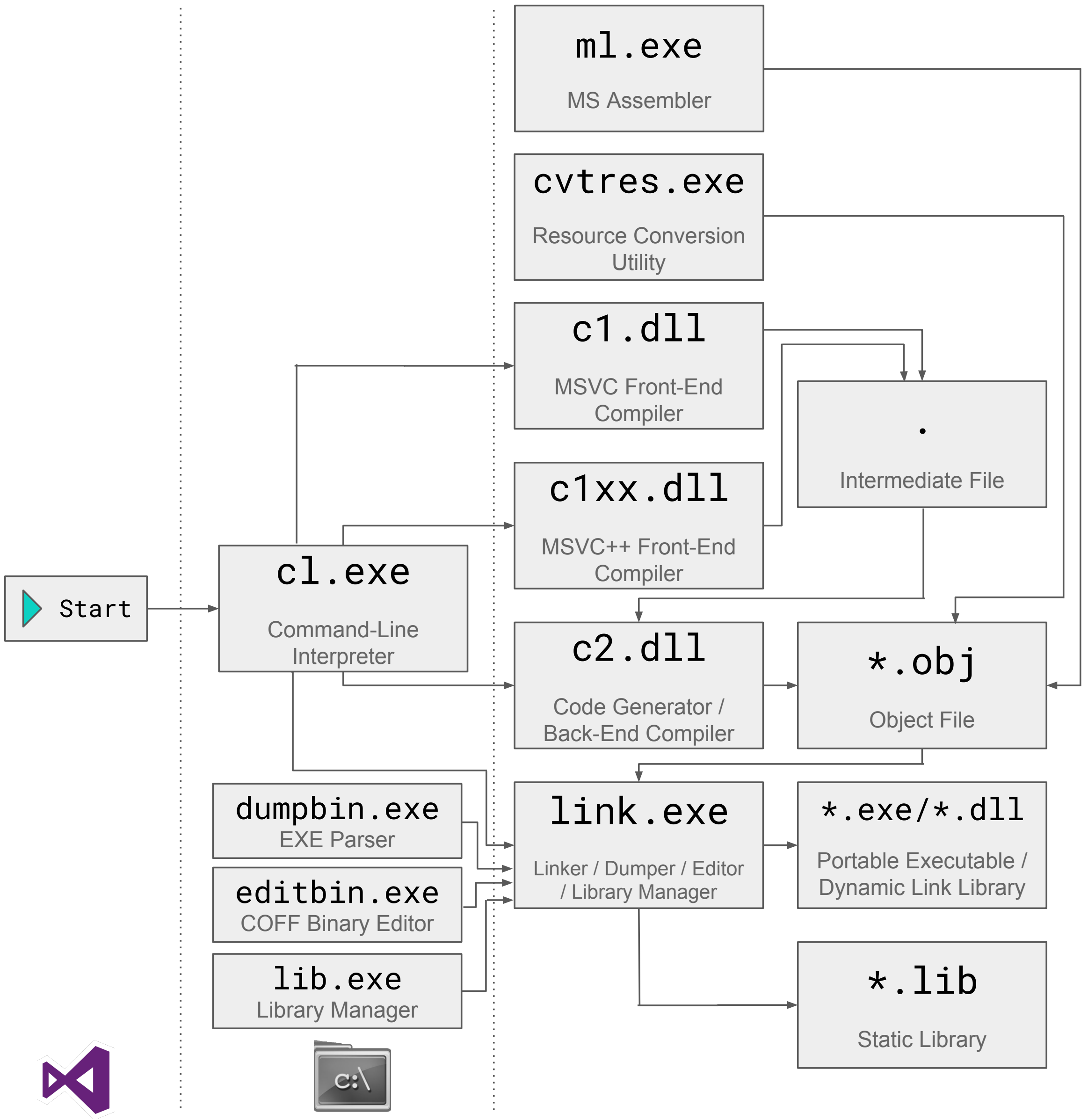

Figure 6.2.: Illustration of the MSVC Compiler Toolchain

6.2.2. Compiler Linking

The typical process of building an executable image is subdivided into two parts: the compilation phase in which the compiler translates code written in a high-level language into machine code and the linking phase where the linker combines all produced object files into one executable image. These are both compartmentalized processes introducing a one-to-one relation between compile units (usually files containing source code) and the resulting object files.

The Microsoft Visual C++ (MSVC) Compiler Toolchain is a commonly used solution for building executable images from source code written in a variety of programming languages. The MSVC system is composed of multiple components and stages, depicted in Figure 6.2. During the compilation phase `cl.exe` provides the interface to the Front-End compilers (`c1.dll` and `c1xx.dd`) as well as the Back-End compiler `c2.dll`. The Front-End compilers create intermediate files from source code. Then, `c2.dll` will create object files (`.obj`) from intermediate files or, in the case where an intermediate file is not required, from source code. While creating the objects, the Microsoft Compiler assigns each object file an ID, in this work referred to as the *@comp.id*, and stores the ID in the header of the respective object file. It is important to note that also the Microsoft assembler as well as the part of the tool chain, which is responsible for converting resource files into a format suitable for linking, generated *@comp.id*s.

Once the compilation phase is complete, `link.exe` will collect the objects needed and begin to stitch the PE32 file or static library (`.lib`) together. Consequently, static libraries consisting of more than one object contain multiple *@comp.id*s. For executables and dynamic link libraries, `link.exe` builds up the Rich Header during generation of the appropriate PE32 Headers.

6.3. Rich Header

We begin the investigation by collecting publicly available details about the Rich Header and analyzing the results. The goal of this collection was to uncover what was known about the Rich Header to focus our efforts. This action is akin to a mini Intelligence Cycle investigation with the primary purpose being to help focus the Planning and Direction step for the larger investigation. While we were able to uncover the history of the Rich Header and some details, it was surprising how little was publicly available. This section documents what we were able to uncover about the historic efforts and how the Rich Header is currently treated by the major PE32 header extraction *Service*s.

6.3.1. History of Previous Investigation into the Rich Header

As previously mentioned, the header is an undocumented field in the PE32 file format that exists between the MS-DOS and the COFF Headers. Even though the field is undocumented, it is referred to at the Rich Header because one of the elements in the header contains the ASCII values for "*Rich*." While rumors of its existence and speculation on its purpose have existed across multiple communities for a long time, it was not until July 2004 that an article by *Lifewire* started to unveil details about the Rich Header [175]. Unfortunately, this article provided limited technically correct details and the drawn conclusions—especially regarding the purpose of the *@comp.id* field—were incorrect. Four years later, on January 2008, Trendy Stephen furthered the understanding of the Rich Header by discovering some of the meaning behind the *@comp.id* field and a relatively correct assessment of how the checksum is generated [176]. A few months later, Daniel Pistelli released an article that provided a guide for extracting the Rich Header and portions of the *@comp.id* field [177]. Then, two years later, in November 2010, Daniel Pistelli updated his article with information describing how the high value bits in *@comp.id* correspond to a "Product Identifier" (referred to in this work as *ProdID*) [177]. However, while we found that the work of these pioneers provided crucial details on how to reverse engineer the Rich Header, aspects of the header were still poorly understood and these articles often conflicted with each other. Specifically, we determined that the full structure of the *@comp.id* has not been identified, details about how to map the *ProdID* were unknown, and mistakes were made in the methods proposed to extract the *Information*.

6.3.2. Previous Efforts to Extract The Rich Header

Above we discussed that the Rich Header had been identified for a number of years but no articles completely and accurately explained the header's structure. As a result, we found that most common triage engines and libraries that parse the PE32 Headers either ignore, do not fully process, or perform incorrect parsing of the Rich Header. For example, two of the most common and openly available malware triage systems, VIPER and MITRE's CRITs, do not properly extract this field. In the case of VIPER, the supplied PE32 Header extractor ignores the Rich Header field entirely, whereas CRITs will attempt to process the Rich Header but performs an incomplete extraction. Specifically, CRITs will only process the first `0x80` bytes of the Rich Header and does not extract the fields contained in the *@comp.id* data structure. Unfortunately, this is not unique to the *Service*s in triage systems. When looking at major PE32 parsing libraries, we found that the very popular pefile has a similar issue to CRITs in that it also only parses the first `0x80` bytes of the Rich Header and does not extract the values contained in the *@comp.id*. Furthermore, the PE32 extraction script, `pescanner.py`, from the Malware Analyst's Cookbook [178] ignored the Rich Header field entirely. In summary, we found that any common method to extract the PE32 headers did not include the Rich Header or their Rich Header parsing was inaccurate. Further, if the parsing of the Rich Header was attempted, it falsely assumed that the start location could only occur at `0x80` in the *Data* artifact.

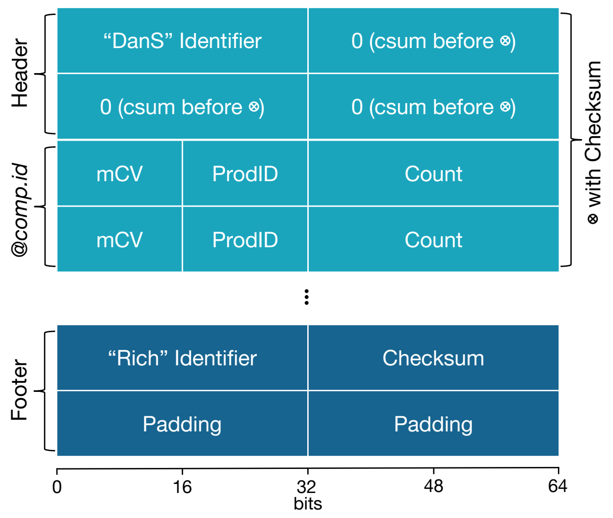

Figure 6.3.: Structure of the Rich Header

6.4. Revealing the Rich Header

Our previous investigation into this history of the Rich Header found significant gaps in understanding and accuracy. To overcome this issue, we developed a custom Holmes Processing *Service* to accurately extract the Rich Header *Information*[1]. The process in creating this *Service* was iterative and closely followed the Intelligence Cycle. Specifically, we used our *Service* to extract the Rich Header on a set of *Data* artifacts. We then processed the results using a statistical approach powered by Apache Zeppelin to find anomalies that required further investigation. Based on these anomalies, we focused on efforts in reverse engineering the MSVC Compiler Toolchain and manually investigated the structure of the Rich Header in any abnormal *Data* artifacts. This process was repeated with a larger sample set until we were confident that we were able to perform a successful *Information* extraction.

This section documents the details we uncovered in the Rich Header, which has not previously been completely and accurately described in any single source. We then explain how the header is added to PE32 files, reveal the meaning behind the *ProdID*s, and present our algorithm for generating the hashes used to obfuscate the header.

6.4.1. Core Structure

By calling the function *CbBuildProdidBlock* [177], the Microsoft Linker (`link.exe`) adds the Rich Header to the resulting binary during building. Although this action is performed during the building of the COFF Header, it is undocumented in the Microsoft specification [174] and begins before the official start of the COFF Header, as designated

[1]The source code is available in Appendix A

by the symbol `e_lfanew` in the MS-DOS stub. Additionally, this field is ubiquitous and cannot be disabled by compilation flags or by selecting different binary formats. The only notable exception is when the linker is not leveraged. For example, *.NET* executable files do not use the MSVC linker and these executables do not contain a detectable Rich Header.

The Rich Header has been added as far back as 1998 with the release of Microsoft VC++ 6. Since then, each iteration of the Microsoft Toolchain adjusts how the header is generated and updates the *ProdID* mapping that the MSVC can generate. However, we suspect that this header has been included prior to VC++ 6. This is because we have seen evidence of a potential Rich Header like field in samples that were generated before the release of VC++ 6. Unfortunately, it was not possible to confirm this belief because we were unable to obtain an older version of the Microsoft Linker and received only a smattering of samples generated before 1998.

Diving deeper, the generated structure of the Rich Header is composed of three distinct sections: the header, an array of *@comp.id* blocks, and the footer, as Figure 6.3 depicts. Together, these provide four core pieces of information: (*i*) a checksum composed from a subset of the PE32 MS-DOS header and the *@comp.id*s, (*ii*) the *ProdID* used when building the binary, (*iii*) the minor version information for the compiler used when building the product, and (*iv*) the number of times the linker leveraged the product during building.

The header of the Rich Header is composed of four blocks where each is `0x04` bytes in length. The first block contains the ASCII representation of "*DanS*"—it is speculated that "*DanS*" probably refers to *Daniel Spalding* who ran the linker team in 1998 [179]—while the next three blocks contain null padding. During linking, this section is XORed with a generated checksum value that is contained in the footer of the Rich Header.

The next section of the Rich Header is represented by an array of *@comp.id* blocks. Each block is `0x08` bytes in length and contains information related to the Product Identifier (*ProdID*), the minor version information for the compiler used to create the product (*mCV*), and the number of times the product was included during the linking process (*Count*). All fields are stored in little endian byte order and XORed with the previously mentioned checksum value. The *@comp.id* block consists of the following three values:

1. The *mCV* field contains the minor version information for the compiler used to build the PE32 file. This version information allows the establishment of a direct relationship between a particular version of the Microsoft Toolchain and this *@comp.id* block in the Rich Header. For example, Microsoft's latest Visual Studio 2015 release ships version `14.00.23918` of the MSVC compiler (`cl.exe`). Therefore, object files created by this compiler will contain the value of `0x5d6e`. During the linking process for the building of a PE32, the value will be added into the produced PE32's Rich Header in the *mCV* field of the *@comp.id* block representing this object.

2. The *ProdID* provides information about the identity or type of the objects used to build the PE32. With respect to type, each Visual Studio Version produces a distinct range of values for this field. These values indicate whether the referenced object was a C/C++ file, an assembly blob, or a resource file before compilation as well as a subset of the compilation flags. For example, a C file compiled with Visual Studio 2015 will result in the value `0x104` being copied into the Rich Header as *ProdID* in all PE32 files that include the respective object file.

3. The *Count* field indicates how often the object identified by the former two fields is referenced by this PE32 file. Using a simple C program as an example, this field will hold the value `0x1` zero-extended to span 32 bits, indicating that the object file is used once by the PE32.

The final section of the Rich Header, the footer, is composed of three blocks of information. The first block is `0x04` bytes in length and represents the ASCII equivalent of "*Rich*". The next `0x04` bytes are the checksum value that are used as the XOR key for enciphering the Rich Header. The final block section is used as padding, typically null, and ensures that the total length of the Rich Header is a multiple of 8. Unlike the previous two sections, the footer is not XORed with the checksum value.

6.4.2. Hashes Contained Within the Rich Header

In the Rich Header, the checksum value appears at four distinct places, as shown in Figure 6.3. The first three occurrences are located immediately after the ASCII-equivalent of "DanS". As the linker initially places null values in this location, they only appear after the header is XORed with the checksum value. The final checksum is located in the footer and immediately after the ASCII-equivalent of "Rich".

While checksum values are traditionally straightforward to generate, the Rich Header's checksum has interesting properties. Specifically, only 37 of each *@comp.id*'s 64 bits are calculated. This appears to have been a previous point of failure in other works. However, by iterating over numerous samples, we were able to detect the anomaly and focus our reverse engineering efforts. As such, we present the following algorithm which produces a valid Rich Header checksum.

The Rich Header checksum is composed of two distinct values c_d and c_r that are summed together. To calculate c_r, we define the rol operator, which zero extends its first argument to 32 bits and then performs a rotate left operation equal to the second argument's value of the first argument's bits. We define rol as:

$$\begin{aligned}\mathrm{rol}(val, num) := ((val \texttt{ << } num)\ \&\ \texttt{0xffffffff})\ |\\ |\ (val \texttt{ >> } (32 - num))\end{aligned}$$

where `<<` and `>>` denote logical left and right shift, and `|` and `&` are the binary OR/AND operators. Then, the distinct parts of the checksum *csum* are calculated in the following way:

1. For c_d, all bytes contained in the MS-DOS header with the "*e_lfanew*" field (offset `0x3c`) set to `0` are rotated to the left by their position relative to the beginning of the MS-DOS header and summed together. Zeroing the "*e_lfanew*" field is required as the linker cannot fill in this value because it does not know the final size of the Rich Header, and, Therefore, is unable to calculate the offset to the next header. Let n denote the length of the MS-DOS header in bytes (most commonly `0x80`) and let $\textsc{dos}_i$ be the i-th byte of the (modified) MS-DOS header:

$$c_d = \sum_{i=0}^{n} \mathrm{rol}(\textsc{dos}_i, i)$$

2. To calclulate c_r, the algorithm first retrieves the list of m *@comp.id* blocks. Then the algorithm combines the corresponding *mCV* and *ProdID* parts into one 32 bit value. Finally, this value is rotated to the left by its respective *Count* value:

$$c_r = \sum_{j=0}^{m} \mathrm{rol}(\mathit{ProdID}_j \ \texttt{<<}\ 16 \mid \mathit{mCV}_j, \mathit{Count}_j \ \&\ \texttt{0x1f})$$

   It is noteworthy that despite the fact that *Count* is a 32 bit field, the checksum algorithm only considers the least significant byte value (& `0xff`). Combined with the fact that $m \equiv n \mod 32 \implies \mathrm{rol}(v, n) = \mathrm{rol}(v, m)$, it is sufficient to perform the calculation as indicated above.

The two values c_d and c_r, and the size of the MS-DOS header (`0x80`) are then added together to form the final checksum value:

$$\mathrm{csum} = \texttt{0x80} + c_d + c_r$$

6.4.3. Generation of @comp.id and ProdID

The *@comp.id* is generated for each object file before linking. The type of the object being created is determined during the creation of the object file. With this information, the respective generator (see Table 6.1) will then assign a *ProdID* and *mCV* that maps to the object type and the Visual Studio release version in which the object was compiled.[2] For instance, a *ProdID* value of 255 to 261 corresponds to a Visual Studio 2015 Resource, Export, Import, Linker, Assembler, C, and a C++ file respectively. The same range of values can be shifted to base values `0xab`, `0xcf`, and `0xe1` which correspond to Visual Studio 2010, 2012, and 2013. Additionally, the *ProdID* is adjusted based on the compilation flags used to create the object. To date, we have identified that the MSVC Toolchain is capable of assigning 265 *ProdID*. During our research, we found that the generated *ProdID*s cannot be manually changed without patching the compiler backend.

[2]For a mapping of *ProdID*s that the MSVC Toolchain can generate, see Appendix B.

ProdID	VS Release	Object Type	Generator
`0x105`	2015	C++	`c2.dll` via `cl.exe`
`0x104`	2015	C	`c2.dll` via `cl.exe`
`0x103`	2015	Assembly	`c2.dll` via `ml.exe`
`0x102`	2015	Linker	`link.exe`
`0x101`	2015	Imported sym.	`c2.dll` via `cl.exe`
`0x100`	2015	Exported sym.	`c2.dll` via `cl.exe`
`0xff`	2015	Resource file	`cvtres.exe`

Table 6.1.: Subset of *ProdID*s generated by Visual Studio 2015

In cases where a *ProdID* is already present, such as a third-party static library (`.lib`) containing multiple object files, the linker uses the preexisting *ProdID*s and *mCV*s. Inside of the library, the data is represented as a linked list.

Interestingly enough, in our investigation we have found that these *ProdID*s do not necessarily correlate to what the MSVC Toolchain can generate. Specifically, we have identified 251 *ProdID*s that cannot be generated by MSVC. We analyzed these oddities over a few million PE32 *Data* artifacts and identified that they appear correlated to a bundled library or the libraries supplied by major corporations. However, we could not find hard evidence to support this claim.

6.4.4. Adding the Rich Header to the PE32 File Format

During the build process, the section that generates the data contained in the Rich Header is located in the Microsoft Compiler backend (c2.dll). The Microsoft Linker (link.exe) then collects the data required to build the Rich Header and places it in the generated PE32 file.

6.5. Knowledge Based Statistical Analysis

The previous parts of our investigation revealed the structure of the Rich Header and guided our ability to create an effective Holmes Processing *Service* to extract the *Information* this header contains. Using the refined *Service*, this Section studies the effectiveness of using the Rich Header for triage operations. To do so, we used our Holmes Processing *Service* to perform statistical analysis against the *Information* extracted across approximately one million malware samples.

6.5.1. Data Sources

To focus our study, we used four sets of PE32 samples. The first set is composed of 964,816 randomly selected malicious PE32 samples from 2015 and is supplied by VirusShare [180]. The second set contains 1875 samples from the Mediyes dropper [181].

Family	Total	Rich Header	Percent
Random Set	964,816	683,238	71%
APT1	292	286	98%
Zeus-Citadel	1928	717	37%
Mediyes	1873	30	2%

Table 6.2.: Samples containing a Rich Header with total percentages rounded

The third set contains 2031 samples related to the Zeus derivative Citadel [141]. The final set is composed of 293 samples associated with the APT1 espionage group [182].

In total, these binaries represent a diverse set of malware types that range from traditional criminal malware, highly advanced state-sponsored malware, and programs which have not been confirmed to be malicious but are highly suspicious. It is worth to mention that in order to study the effectiveness of the Rich Header during triage, we made no efforts towards unpacking or deobfuscating these samples.

6.5.2. Information Gathering

To generate sets of *Information* against these sample sets, we used the Holmes Processing prototype, out automated infrastructure based on SKALD and CARE, and executed five *Service*s. These *Service*s (*i*) extracted the Rich Header using our custom tool, (*ii*) performed Yara signature matching with 12,693 signatures provided by Yara Exchange [125], (*iii*) retrieved malicious scan results from VirusTotal, (*iv*) performed identification on the compiler and any potential packer used to create the sample, and (*v*) produced decompiled C code generated via IDA Pro.

6.5.3. Statistical Results

With our gathered *Information* we performed a series of statistical studies. This was done to better understand the Rich Header's prevalence in malicious samples and identify which packers or compilers omit the Rich Header. Additionally, we developed a statistical check that is capable of rapidly identifying packed and post-modified PE32 files, leveraging data only contained within the Rich Header.

6.5.3.1. Samples with a Rich Header

We identified that a surprisingly high percentage of samples contain the Rich Header, as shown in Table 6.2. For instance, 71% of the random sample set and 98% of the APT1 sample set contained parseable versions of the Rich Header. This is surprising as our initial assumption was that malware authors would use a variety of compilers when creating samples and potentially attempt to strip the Rich Header. However, the results

Family	No Rich	dUP	MinGW	Borland
Random Set	157,497	135,115	25,387	63,283
APT1	6	4	1	0
Zeus-Citadel	1211	673	93	398
Mediyes	1843	1787	0	194

Table 6.3.: Samples not containing a Rich Header

show that the majority of malware authors are in fact leveraging the Microsoft Linker and pay no mind to the Rich Header.

Based on the above information, we conclude that the Rich Header is commonly found in malware and that malware authors do not deliberately strip the Rich Header. Furthermore, we can conclude that compilation of malicious binaries are most often done using compilers that leverage the Microsoft Linker.

6.5.3.2. Compilers and Packers without a Rich Header

While the high rate for malware containing a Rich Header is positive for *Knowledge* based triage, this was not a uniform result. Specifically, some malware variants reported a low match for samples containing the Rich Header, such as Mediyes reporting 2%. In Section 6.2.2, we discussed that the Rich Header is generated by the Microsoft Linker. This implies that compilation tools not using the official Microsoft Linker should not generate the Rich Header. While this can explain why some samples do not include the Rich Header, in this section, we further explore other reasons behind the absence of the field. Specifically, we identify common tools and packers used by malware to either strip or corrupt the Rich Header.

To do this, we used our *Service* that performs compiler and packer identification to scan all samples without a Rich Header. This was done to identify if there are any commonalities with these samples. As Table 6.3 shows, the percentage of samples built by either Borland C++ Builder or MinGW, which is based on GCC, is relatively high and accounts for approximately a third of all samples that do not contain a Rich Header in the random and Citadel datasets. However, this was not the case in the APT1 and Mediyes dataset. Upon further analysis, we identified that most packers, while sometimes introducing anomalies, did not often strip the Rich Header from samples. With respect to the Mediyes set, we had a high rate of matches for the Themida Packer [183]. As we discuss further in Section 6.5.3.2, Themida is one of the packers that rewrites the entire PE32 file and does not include the Rich Header. Instead, we identified that the absence of a Rich Header was a result of corruption caused during the packing of the sample.

To identify if other packers caused similar corruption, we leveraged our identification *Service* again to detect the most common packers used by our malware datasets. Our results showed that UPX, ASPacker, mingw, dUP, and the Nullsoft Scriptable Install System were the top five most commonly used packers. As we already understood that

Family	Total	Dup. ID	csum Err
Random Set	683,238	15,006	137,965
APT1	286	0	34
Zeus-Citadel	717	17	357
Mediyes	30	0	0

Table 6.4.: Samples containing a Rich Header that have duplicate entries and invalid checksums

samples created with mingw and dUP will remove or otherwise corrupt the Rich Header, we manually created test samples with variants of UPX (v1, v2, and v3.91), ASPacker, and Nullsoft. In every manual test case, we were unable to cause a corruption or exclusion of the Rich Header field.

6.5.3.3. Identifying Modified Binaries Based on Rich Header Corruption

In our previous results, we found that it was uncommon for malware authors to deliberately strip the Rich Header. As such, we re-evaluated our samples to search for cases where the Rich Header was inadvertently corrupted.

The first approach we took was to identify cases where the Rich Header contained duplicate *@comp.id* blocks. We took this approach because under normal operation, the Microsoft Linker should never produce duplicate entries. This is because during the linking process, the Microsoft Linker will search for existing instances of the *ProdID* and *mCV* and if identified, will increment the number of times it was used, *Count*, to the existing entry.

The second approach we applied was to re-calculate the Rich Header checksum and compare it to the sample's reported Rich Header checksum. This was done as an unsuccessful check would indicate that either the MS-DOS Header or the Rich Header was modified after the linking process; potentially revealing Trojanized or post modified binaries.

As Table 6.4 shows, the amount of malicious samples containing a corrupted Rich Header varies and can rise upwards to 50% based on the malware family. Additionally, across the random one million dataset, this corruption occurred approximately 31% of the time. Knowing this and the fact that no official Microsoft Linker should produce these forms of corruption, identifying corruption of the Rich Header can be a fast and efficient triage step to use for screening samples for potential maliciousness.

6.5.3.4. @comp.id and mCV Values Present in Malware

To develop an understanding of how we can potentially leverage the Rich Header for more advanced triage operations, we studied the *@comp.id* values in our malware datasets. By doing so, we identified 516 unique *ProdID*s. This was surprising as all versions of the

MSVC Toolchain, dating back to VS++ 6, are only capable of generating 265 *ProdID*s. While researching the 251 unknown *ProdID*s, we identified that these appear to more than likely correlate to bundled libraries and major corporations. However, while in practice this assumption appears to be accurate, we cannot conclusively confirm this.

Digging in deeper, we discussed in Section 6.3 that the *ProdID* is paired with the mCV. Thus, potentially providing more fine-grained *Information* for identifying specific objects. To confirm this, we created tuples of all the *ProdID* and mCV pairings. We then single out 29,460 distinct *ProdID* and mCV pairs across our approximately one million malware samples. These numbers show relatively substantial variability in the *@comp.id*s found in malware and malware authors' build environments.

6.6. Machine Learning Based Analysis

The *Knowledge* obtained in Section 6.5 showed promise in using the Rich Header for more complex triage operations. This is especially true considering the vast majority of our datasets are from the same date range and the fact that the Rich Headers of malicious samples contain numerous *@comp.id*s along with the number of times the object was used during linking.

To demonstrate the potential of leveraging the Rich Header in future work, we created a basic proof of concept *Knowledge* based *Service* for Holmes Processing. This *Service* utilized machine learning to process the newly revealed *Information* of the Rich Header, specifically the *@comp.id* values. Specifically, the values for *ProdID*, *Count*, and mCV. As the Rich Header identifies linked objects and version information of the build environment, our *Service* is specifically focused on identifying similar samples, based on linked objects, and also samples using a similar build environment. In crafting the machine learning algorithm, we used a feature hashing strategy which transformed the *Information* into a *50-dimensional* vector. We then leveraged a *Stacked Autoencoder* to turn our vectorized features into a denser, lower-dimensional space. Finally, in order to improve performance and allow us to scale to support datasets containing millions of malware samples, we utilized a *Ball Tree* for fast storage and retrieval of the vectors.

In the following case studies, we demonstrate the ability of solely using the Rich Header to perform similarity matching leveraging our proof of concept *Service* that is based on a custom machine learning algorithm. In the case studies, we compare the exemplar samples, selected at random, with the collected vector similarities from the *Ball Tree* populated by the random one million, APT1, and Citadel datasets, and analyze their closest matches. For our ground truth in the case studies, we compare the results of our algorithm to the results returned by Kaspersky and Symantec Antivirus, as implemented by VirusTotal, and perform manual reverse engineering. We selected this ground truth method primarily due to the limited matches across Yara and the high percentage of no detection or generic signatures across other popular AV vendors.

6.6.1. Similarity Matching with the APT1 Dataset

We selected three exemplar samples from the APT1 dataset for our first case study. APT1 was selected as the actor is a relatively skilled Advanced Persistent Threat (APT) and 98% of the samples in the APT1 dataset contain a Rich Header.

We randomly selected our first exemplar sample, E1. Kaspersky classifies this sample as *HEUR:Trojan.Win32.Generic* which means that through heuristic analysis, Kaspersky believes that this is a Trojan but has not classified the sample further. When querying our algorithm, we identified that it had an identical Rich Header feature vector with another APT1 sample, which we will refer to as E1-R1. Inspecting E1-R1, Kaspersky classified the sample also as *HEUR:Trojan.Win32.Generic*. While a generic classification does not tell us much, manual analysis of the generated source code, produced by IDA Pro, confirmed that these two samples were in fact identical.

Going a step further, we then queried the nearest neighbor to E1. This returned three samples: E1-N-R1, E1-N-R2, and E1-N-R3. All three matches were also contained in the APT1 dataset and shared the *HEUR:Trojan.Win32.Generic* Kaspersky classification. Our algorithm reported that the distance between these vectors was `1`; the smallest possible difference without the vectors being identical. We then performed manual analysis and identified that the generated source code produced by IDA Pro for E1-N-R1 was identical to our exemplar. However, as the vectors were slightly off, we further analyzed the cause for this and concluded that the variance was caused by slightly different build environments when compiling the binaries.

The other two nearest neighbor matches, E1-N-R2 and E1-N-R3, produced even more interesting results. In both cases, the generated source code produced by IDA Pro had slight differences. In the case of E1-N-R3, E1-N-R3 adds a call to function $FlushFileBuffers$ right after it writes the buffer to a file. Furthermore, E1-N-R2 seemed to build upon the changes made to E1-N-R3. Specifically, E1-N-R2 includes an additional change in that E1-N-R2 adjusted how it wrote the buffer to files. In our exemplar sample, E1 first writes the buffer to the file and then performs a second write that adds `\r\n` to the file. In the case of E1-N-R2, the sample does not write `\r\n` to the file and instead calls `strcat` on the buffer in order to add `\n` to the buffer before it writes the buffer to the file.

Our second exemplar, E2, was selected from the APT1 dataset because its signature was different than E1 and it shares its feature vector with no other samples. After running our algorithm, we identified E2-N-R1 as the nearest neighbor to E2 at a distance of 1.732. While it is reasonable to argue that the distance is very near, it is indicative of a clear similarity between the samples.

When analyzing the results, both E2 and E2-N-R1 are classified by Kaspersky as "Agents". However, the generated source code produced by IDA Pro for both samples is quite different and the programs have different functionality. To understand why our algorithm identified this as a match, we performed additional research on the binaries and found that both E2 and E2-N-R1 are very small, had a nearly identical import table with only one variation, and were packed with Armadillo v1.71. Looking at the Rich Header vectors, we found that the vast majority of the objects imported all had identical version information; which led us to conclude that the samples were more than likely built on the same machine or the machines at least had an identical build environment. Open-source research further validated this opinion as both samples were used by APT1 in cyber operations [184]. While not a direct match in terms of functionality, this example demonstrates the power in using the Rich Header to identify not only similarly behaving malware but also malware that is related because the malware is presumably built on the same machine.

Our final exemplar, E3, was selected as it had five samples that shared the same Rich Header feature vector: E3-R1, E3-R2, E3-R3, E3-R4, and E3-R5. In all cases, these samples ended up being members of the APT1 dataset and shared the *HEUR:Trojan.Win32.Generic* Kaspersky classification. Manual reverse engineering also showed that the samples shared a nearly identical code base and performed the same functionality.

We then queried our algorithm for the nearest neighbors to E3 that were not in the APT1 sample set. The query returned six samples at a distance of 2.236: E3-N-R1, E3-N-R2, E3-N-R3, E3-N-R4, E3-N-R5, and E3-N-R6. Kaspersky classified E3-N-R2, E3-N-R4, E3-N-R5, and E3-N-R6 as *HEUR:Trojan.Win32.Generic*. E3-N-R1 and E3-N-R3 were classified by Kaspersky at *Net-Worm.Win32.Cynic.in* and *Net-Worm.Win32.Cynic.am*, respectively. However, although the Kaspersky classifications were different, manual analysis revealed that all E3-N-R* samples were nearly identical to each other. The only differences between the samples in this cluster were caused by artifacts left by the obfuscation engines and by the language settings on the build environment. Furthermore, the two clusters for the same vector, E3 and E3-R*, and nearest neighbors, E3-N-R*, were remarkably similar in functionality.

During the evaluation, we identified that the similarity matching algorithm produced very strong results for the exemplar samples E1 and E3. However, with E2 and E3, the algorithm further identified samples of similar nature and with a similar build environment.

6.6.2. Similarity Matching with the Citadel Dataset

In this case study, we opted to explore the results of two exemplar samples from the Citadel botnet. We selected this dataset because the Citadel actors are typical of basic cyber criminals and as such have a different target and mission than the actors behind the first test case.

Kaspersky classified our first exemplar, E4, as a generic Trojan. Our algorithm though was able to identify 23 similar samples, E4-R*, which shared the feature vector of E4. Kaspersky classified them as either *HEUR:Trojan.Win32.Generic* or *not-a-virus:AdWare.Win32.FakeDownloader.ac* whereas Symantec identifies all the E4-R* samples as *Trojan.Gen* or *Trojan.Zbot*. When comparing the IDA Pro generated source code, we confirmed that the E4 and E4-R* samples were nearly identical; the differences in E4 and E4-R1 are that sections of the code were moved under different functions and that E4 uses a “for loop” while R4-R1 uses a “do while” for their XOR algorithm. Thus, the difference in E4 and E4-R* appear to be caused by slightly different source code versions, compiler optimization, or artifacts left by the obfuscation engines.

When looking at the nearest neighbor cluster for E4 we identified four additional samples: E4-N-R1, E4-N-R2, E4-N-R3, E4-N-R4. While Kaspersky classified the sample as *HEUR:Trojan.Win32.Generic*, Symantec identified E4-N-R1 and E4-N-R3 as clean. However, when looking at the samples, we observed only a slight variation in that E4-N-R* ran the XOR loop 220,712 times where E4 and E4-R* ran the XOR loop 51,700 times. As with E4-R*, E4-N-R* also moved code segments into different functions. When verifying the Rich Header, we observed that the reason for being classified as a nearest neighbor was because of variations in the number of times one product was included. This is a clear example where using Rich Header values as a triage system could prove useful for an investigative team by identifying similar malware samples from potentially different versions.

The next exemplar, E5, shares its vector hash with 36,606 samples. This is notably high, no less so due to the fact that Kaspersky fails to identify 16,123 of those samples with a classification of any kind, not even the most generic of names. However, when comparing the IDA Pro generated source code, we observed only small variations; specifically the value for a constant was changed.

The nearest neighbor grouping for E5 has a distance of `2` and contained a total of 1,567 samples, where 511 samples have no Kaspersky listing. In fact, the majority of samples in both groups are listed as generic Trojans, *HEUR:Trojan.Win32.Generic*. While the inclusion of a known Citadel sample is not enough to convict these samples as Citadel members, it provides an interesting jumping off point for analysis.

6.6.3. Similarity Matching with the Mediyes Dataset

In our final case study, we selected a random Mediyes sample, E6, to use as our exemplar. We chose this sample because it would allow us to perform out-of-set comparisons as the Mediyes dataset was not originally vectorized and included in the *Ball Tree*. As such, this was used as a comparison metric to see how our algorithm would cope with the inclusion of an entirely new dataset, which was unlikely to have been included in the *Ball Tree* through other means.

Querying our algorithm for identical Rich Header feature vectors, we received a list of 266 other samples: E6-R*. When querying for the nearest neighbors we receive 86 additional samples, E6-N-R*, with a distance of `1`. Analysis of the IDA Pro generated source code showed a strong correlation between the samples. Furthermore, the vast majority of both E6-N-R* and E6-R* were classified by Kaspersy as Zango samples; an instance of adware frequently associated with Mediyes.

6.7. Future Work and Limitations

In this chapter, we study an important yet little-studied header of PE32, the Rich Header. We show that the Rich Header has been largely ignored by malware authors and is not removed by most packers and obfuscation engines. In fact, 71% of the 964,816 samples in our random dataset include the Rich Header. Our investigation revealed that the Rich Header contains useful *Information* that can be leveraged by defenders; and analysis of the *Information* contained within the header allows for the rapid triage of samples using a cost effective approach. This is true even for samples that are stripped and contain little to no PE32 Header information.

We strongly believe that by leveraging the Rich Header, current and future triage algorithms will perform a more accurate and cost-effective triage functionality. In future work, we will explore how to combine the Rich Header features with other aspects of the PE32 file format to generate robust similarly matching and clustering algorithms. As the Rich Header artifacts help to identify similar malware as well as characterizing the build environment in which the malware was built, this presents new opportunities for attribution and tool-chain identification.

Furthermore, as *Knowledge* of the Rich Header grows, it is understandable that malware authors will attempt to obfuscate the *Information* in this header. This is an expected outcome but also presents interesting future work potentials when the Rich Header is combined with additional features. This is because leveraging compiler fingerprinting and additional PE32 header information can be used to determine if the Rich Header should be included and approximate the expectant values of this field. As such, future *Knowledge* based analysis efforts can identify anomalies in the Rich Header field. This increases the complexity required when performing obfuscation and adds resiliency.

In fact, Kaspersky successfully performed this recommendation and identified anomalies between the Rich Header *Information* and what is contained in the actual *Data* artifact. Through additional analysis, Kaspersky concluded that this discrepancy was deliberate and was an indicator of a potential false flag by another actor—to the authors knowledge, an industry first.

6.8. Discussion

The breakthrough we achieved in understanding the Rich Header and identifying its utility during security investigations was greatly aided by the Holmes Processing prototype. Holmes Processing allowed our team of experts, with different specializations, to seamlessly to work together across the Intelligence Cycle and rapidly test hypothesis across very large sample sets. In turn, this allowed the team to more accurately and quickly accomplish their individual tasks and uncover details that would have been otherwise missed.

Our research into the Rich Header started by the identification of irregularities in how the Rich Header was handled by various PE32 header information extraction techniques. As previously discussed, these extraction methods would sometimes skip over the Rich Header or extract *Information* that was not uniform or complete in appearance. This piqued our interest and created our investigation requirement to uncover how to properly extract the Rich Header and identify its potential utility in computer security investigations.

With our overarching requirement for the Planning and Direction phase, we began to structure our efforts according to the other steps of the Intelligence Cycle. For the Collection step, we sourced approximately one million PE32 (*Data* artifacts) that included random samples, labeled samples from previous investigations, samples built under various tool chains, and synthetic samples we created from every MSVC compiler we could find. Focusing on the Process and Exploitation step, we created an *Information* extraction *Service* for Holmes-Totem that could successfully extract the *Information* from the Rich Header. We also leveraged existing Holmes-Totem *Service*s that extracted other sets of *Information* based on 12,693 Yara rules, retrieves VirusTotal results, performed a rough fingerprinting of the samples build environment with PEiD, produced decompiled C code, and extracted the PE32 headers with PEMeta. To distill these large sets of *Information*, 7 million artifacts, we then created Holmes-Analytic *Service*s that could perform statistics and eventually a similarity matching machine learning algorithm. To Disseminate our finding, we created polished Holmes Processing *Service*s that could perform complete and accurate extraction of the Rich Header, perform statistical triage of PE32 samples, and perform a similarity matching for the malware family and the malware authors build environment.

This was a monumentally challenging process and required multiple experts in various fields to focus on their specific task while receiving input from their peers. However, because of Holmes Processing, including these experts was a seamless process and any achievements made was immediately Disseminated and Integrated to the rest of the team. For instance, as we made headway into extracting the Rich Header, we updated our *Information* extraction *Service* for the Rich Header and quickly ran this new version across millions of samples in a matter of minutes. This updated set of *Information* was then immediately available to our machine learning and statistical experts, with no changes to their algorithm required. This in turn allowed these large-scale experts to help identify discrepancies and anomalies in our findings which helped focus the reverse engineering efforts.

The ability of Holmes Processing to support the above described cyclical process and test the finding across such a large data set directly enabled our biggest breakthrough. For instance, the ratio of successful to unsuccessful extraction of the rich header, based on our estimates and the information from PEiD, is what caused the reverse engineers to identify the error in how the MSVC was creating the Rich Header checksum. Furthermore, it was the unusual clustering of malware families and the statistical patterns from the Rich Header 's *@comp.id* values that lead us to suspect a fingerprinting of the build environment was occurring and focused our efforts on creating the *@comp.id* matching table, Appendix B. Lastly, understanding all of the collective whole allowed us to document the Rich Header and show its utility has proved of significant value to the security community [37].

6.9. Related Work

Leveraging the *Information* derived from the PE32 file format has been widely explored for triage purposes. One common technique, as shown by Mandiant's Imphash, is to generate a hash of the values located in the PE32 Import Address Table (IAT) [169]. These hashes are then used in analytic queries and by machine learning algorithms to identify similar strains and families of malware. In this vein, JPCERT recently released impfuzzy to improve upon this technique through incorporation of a fuzzy hash [68]. However, these algorithms require accurate IAT and their accuracy is greatly reduced if the malware strips or otherwise provides a misleading IAT.

In light of this issue, more advanced techniques use additional *Information* that can be derived from the PE32 file format. For example, PEHash uses the structural characteristics of the PE32 file format to generate hashes that are then used in clustering operations [67]. Unfortunately, the above methods only work well when the *Information* is available and not being misconstrued.

Identifying the weaknesses in these approaches, specifically PEHash's lack of robustness, Jacob et al. [168] expand upon these methods by focusing their efforts on PE32's code section. While this approach is more tamper resistant, it is still not immune. On the other side of the spectrum, Perdisci et al. [185] focused their efforts on using pattern recognition to identify packed samples and then send those samples to universal unpacking algorithms before matching occurs. However, while this process does reduce the cost and improves the accuracy in clustering and similarity matching, unpackers are known to be unreliable and exceedingly expensive [185, 186].

In a change of pace, our investigation, using the methods prescribed in this dissertation, illustrates a hidden aspect of the PE32 file format, the Rich Header, that has been largely ignored by malware authors. Using this section of the PE32 file formation, we show how to cheaply identify packed malware, perform similarity matching solely using this field, and identify malware that was created using similar build environments. Our work does not aim to directly compete with the existing research. Instead the *Wisdom* gained from our novel approach aims to be a catalyst for triage when combined with these, and other, triage techniques. In turn, this work enables existing and future algorithms to provide better results, be more resistant to tampering due to the wider scope, and improve returned *Knowledge* by allowing matching based not just on the samples' characteristics but also the characteristics of the build environment used to create the samples.

6.10. Summary

In this chapter, we leveraged the Holmes Processing prototype to perform an Intelligence Cycle investigation of the Rich Header. As a result, we showed the Rich Header has significant potential in being leveraged for triaging malicious samples. To the best of our knowledge, this assessment of the Rich Header is the most complete and accurate report for this hidden and undisclosed section of the PE32 header so far. With this knowledge, we created a custom Rich Header parser and extracted the headers' contents in over 964,816 malicious samples. We demonstrated the Rich Header's potential in enabling the rapid triage of malicious samples. By doing so, we showed how to leverage the *Information* contained in the Rich Header to create a *Knowledge* based *Service* that can identify post-modified and packed PE32 files, detecting 84% of the known packed malware samples. We also demonstrate the value in leveraging the Rich Header by developing a proof of concept *Knowledge* based *Service* using machine learning and performing three case studies. In these studies, we are capable of rapidly returning results in 6.73 ms using a single CPU core, identifying similar malware variants, and highlight malware developed under the same build environments.

In total, we demonstrated that the approaches prescribed in this dissertation provided the catalysis needed to uncover the secrets of the Rich Header. Specifically, we showed how Holmes Processing provided the flexibility, scalability, and resilience needed to analyze millions of artifacts. Furthermore, the method of performing an Intelligence Cycle allowed this breakthrough to occur. This is because it fused the specializations between the team members and allowed them to work as a collective whole, while analyzing massive sets of *Data* artifacts.

Chapter 7

Conclusion

"The game is afoot."

Arthur Conan Doyle

At last this dissertation must end and this chapter serves to summarize the work presented. The first section of this chapter reflects on the challenges of security analytics and provides a high-level description of the proposed solution. The next section of the chapter outlines the major contributions of this work. We then discuss the path forward this dissertation creates for developing new defensive methods and empowering security research. Finally, we conclude with final thoughts.

7.1. A Moment of Reflection

Performing security investigations is necessary for identifying and countering the threats from malicious actors. Without the wisdom that investigations provide, security defenses are blind and unfocused. However, malicious actors are in a rapid state of evolution and security analytics have fallen behind. The traditional paradigm of a lone analyst conducting investigations in isolation and performing manual tasks has become insufficient, inefficient, and ineffective. Traditional approaches and methods cannot keep pace with the growing volume of artifacts, the sophistication of the actors, and proactively defend against malicious activity.

Unfortunately, within five years companies are expected to spend over one trillion dollars on defensive methods and governments continue to issue ineffective legislative acts that cannot overcome the challenges faced by malicious activities. Without a change in paradigm for how analytics and investigations are conducted, the defensive tools, techniques, and mitigation strategies that are created under these traditional methods will remain insufficient to tackle the challenges posed by malicious activities.

7.2. Contributions

This dissertation has discussed the need for a new approach to computer security analytics. New techniques and methods must empower analytics across the Intelligence Cycle and pool the collective knowledge and resources of the community together. Without this evolution, defensive methods, tools, and techniques will be insufficient and ineffective.

- **Provided an Architecture for Large-scale Investigations**
  We began our discussions by presenting a novel architecture that guides the design of analytic systems that support the investigations against the malicious activities plaguing computer systems, named SKALD. SKALD creates analytic systems that can: (*i*) cope with the growing volume of artifacts, (*ii*) be resilient to system failures, and (*iii*) be flexible enough to incorporate the latest technology and analytic trends. SKALD provides this by identifying the core categories of analytic activities, based on the Intelligence Cycle, and creates a "loosely coupled" architecture around these concepts. As such, systems designed using SKALD can receive raw *Data*, extract valuable *Information* from the *Data*, perform assessments across sets of *Information* using advanced analytics to generate *Knowledge*, and aid analysts in collectively making a determination. Furthermore, SKALD provides a central repository for raw artifacts and analytic results that are segregated according to the DIKW model. This way defenders can break the stove-pipping between teams through enabling analytics across a single system. As a result, the SKALD architecture reduces the need for each team to reprocess artifacts and also allows each team member to focus on their core area of expertise.

  The overall result of SKALD is that computer security defenders can more rapidly perform investigations that present a clearer picture of malicious threats. For instance, the SKALD prototype can extract *Information* from artifacts at a rate of 3.1 milliseconds with zero critical errors in contrast to the baseline system's rate of 2.6 seconds and thousands of critical errors. Showing our claims have merit, systems developed from the SKALD architecture have been used to conduct complex investigations, generate academic research, and execute defensive mitigation activities [31–36].

- **Enabled Collaboration and Sharing of Security Artifacts**
  Leveraging the foundation provided by SKALD, we then created a new model for sharing and collaboration amongst security practitioners, named CARE. CARE furthers the goals of the Intelligence Cycle by encouraging collaboration between analysts and empowering the wide dissemination of the results they generate. This is achieved by using real-world collaborative and sharing initiatives and identifying their challenges and where these initiatives have failed. Using this wisdom, we then extended the SKALD architecture to create a platform that breaks the existing sharing paradigm and alleviates many of the issues with why sharing is often

ineffective. CARE overcomes these issues by providing the ability to exchange security artifacts across the DIKW model while preserving the artifacts' lineage and mitigating privacy and secrecy concerns. We then discuss the CAREconomy and describe how this cryptographically backed method incentivizes sharing through the creation of a marketplace and provides new opportunities to encourage healthy collaboration and develop trust. We then presented a discussion on how CARE opens new possibilities in how security groups can collaborate, governments can foster effective security practices, and insurance companies can more accurately identify risk through the secure and distributed ledger.

While our work on CARE is conceptual in nature, it paves the way forward for enabling a truly collaborative environment that empowers analytics and raises the effectiveness of mitigation activities. Even though the concepts and prototypes of CARE are in their infancy, the concepts have already gained the attention of major corporations and government agencies.

- **Introduced a Working Prototype**
  We described our prototype, Holmes Processing, that enables computer security investigations to work across the steps of the Intelligence Cycle and view the derived artifacts as part of a collective whole. As a result, analytic specialties are fused together and teams are able to effectively collaborate. Furthermore, the design of Holmes Processing allows the system to support extremely large datasets, remain flexible to incorporate changes, and be resilient to failures. For instance, the addition of new *Information* extraction *Service*s are painless and the developer is not burdened with details for how the entire system operates. Additionally, analysts working to generate *Knowledge* from *Information* can leverage the work from their peers and are not concerned with how the *Information* was created. All while, allowing the analysts to work with a corpus of 10s of millions of artifacts in near real-time.

  While the prototype is academic in nature and not a robust product, it has been used to empower research and perform real investigation against sophisticated and complex actors [31–36]. Adding validity to our concepts, we have been invited to speak about Holmes Processing and our derived work at numerous highly acclaimed venues, such as Black Hat USA, Microsoft DCC, RSA USA, Hacktivity, and DARPA [2–8].

- **Demonstrated the Concepts by Uncovering the Rich Header**
  We demonstrated the power of our analytic architecture by revealing a hidden aspect of the PE32 file type and creating two effective triage methods. Using the working prototype for Skald and CARE, we performed the first accurate assessment of the Rich Header and detail how to extract its clandestine *Information* and perform advanced analytics. The breakthrough in understanding the Rich Header was achieved by leveraging the power of the prototype to study millions of

malware and benign samples. We then presented a series of statistical studies and described two proof-of-concept methods that use only the *Information* extracted from the Rich Header to generate *Knowledge* about the samples. The first method allows for the rapid detection of post-modified and packed binaries through the identification of anomalies. The second method can be used to identify similar malware, different versions of malware, and when malware has been built under different build environment; revealing potentially distinct actors. Furthermore, we showed how we are able to perform these operations in near real-time; i.e., in less than 6.73 ms on commodity hardware.

Of significant interest, this work has been successfully used by the security community to identify potentially malicious activities and perform investigations against complex actors. In one specific case, Kaspersky described how the Rich Header was used to reveal the tools used of the Lazarus Group and perform attribution [37]. However, what is most interesting is that Kaspersky also identified anomalies between the Rich Header information and what is contained in the actual binary; one of our recommendations for future work. As such, Kaspersky concluded that this discrepancy was deliberate, and that the identified operation was a potential false flag by another actor set—to the authors knowledge, an industry first. This example demonstrates how the methods presented in the dissertation can provide a major boon for performing computer security investigations.

7.3. Looking Forward

The foundations presented in this dissertation pave the way forward for future work and effective defensive strategies. Specifically, SKALD and CARE enable large-scale and complex computer security investigations and effective collaborative analytics. This is done by breaking the problems of an investigation down according to the Intelligence Cycle and archiving the results in alignment with the DIKW model. This creates a paradigm where the workflow of a security analyst feeds directly into an overall system that serves to empower the work of other analysts. As such, analytic teams are able to work together to derive meaning from the ever-growing corpus of malicious activity and deduce understanding behind the actions of malicious actors.

Moving forward, the methods presented in this dissertation open new opportunities for artifact-based analytics. This is due to the ability behind the methods presented in this dissertation to extract *Information* from large and diverse sets of artifacts in near real-time, orchestrate advanced analytics from sets of *Information* to derive *Knowledge*, present this *Knowledge* in multiple ways that empower the human analysts to make a judgment, and share their results with their peers. In total, this allows analysts to not only focus on the immediate threats that they are presented with and have knowledge of, but view the problem as a collaborative investigation activity that incorporates a global perspective and historic context.

Leveraging the architecture of SKALD and CARE, these methods enable the identification of new analytic techniques and seamlessly allow them to plug into existing architectures. This allows the specializations across the research community to come together to tackle the problems of security as a collective whole. For instance, reverse engineers and networks specialists can deduce new methods for extracting *Information* from raw *Data*. While, data scientists can focus on the challenges of deducing *Knowledge* from a plethora of current and historic *Information* and presenting this *Knowledge* in ways that can empower judgments to be made of the threat faced and effective mitigation strategies to be crafted. This foundation enables future work to tackle how to create a holistic picture of malicious activity that identifies the methods behind malicious actions, the techniques actors are using, and understand an actor's motivations and weaknesses.

7.4. Final Words

This dissertation has been a culmination of years of thought and effort derived from tackling some of the hardest problems facing computer security, orchestrating complex investigations, forcing collaboration across multiple agencies, and tracking some of the most elusive actors. The work in this dissertation has effectively aided complex investigations and spurred innovative research initiatives. Moving forward, I hope this work can empower other activities and serve as a guide for how organizations can work together and develop advanced investigative solutions. In an ideal world, the research and prototypes developed by this dissertation will see adoption by a collective of industry partners and be fueled by a non-profit initiative. To that end, I will focus my efforts on these goals and look forward to seeing how this work evolves.

Appendix A Availability

The Holmes Processing prototype has been released under the Apache2 license. It can be located at the following location: `https://www.holmesprocessing.com/`

Appendix B

Mapping of Known ProdIDs in the Rich Header Generated by MSVC

The following table provides a mapping of the known MSVC *ProdID*s in the *@comp.id* field of the Rich Header.

ProdID	**VS Release**	**Release Number**	**Object Type**
0x0000	Visual Studio prior	00.00	prodidUnknown
0x0001	Visual Studio prior	00.00	prodidImport0
0x0002	Visual Studio prior	00.00	prodidLinker510
0x0003	Visual Studio prior	00.00	prodidCvtomf510
0x0004	Visual Studio prior	00.00	prodidLinker600
0x0005	Visual Studio prior	00.00	prodidCvtomf600
0x0006	Visual Studio prior	00.00	prodidCvtres500
0x0007	Visual Studio prior	00.00	prodidUtc11_Basic
0x0008	Visual Studio prior	00.00	prodidUtc11_C
0x0009	Visual Studio prior	00.00	prodidUtc12_Basic
0x000a	Visual Studio prior	00.00	prodidUtc12_C
0x000b	Visual Studio prior	00.00	prodidUtc12_CPP
0x000c	Visual Studio prior	00.00	prodidAliasObj60
0x000d	Visual Studio prior	00.00	prodidVisualBasic60
0x000e	Visual Studio prior	00.00	prodidMasm613
0x000f	Visual Studio prior	00.00	prodidMasm710
0x0010	Visual Studio prior	00.00	prodidLinker511
0x0011	Visual Studio prior	00.00	prodidCvtomf511
0x0012	Visual Studio prior	00.00	prodidMasm614
0x0013	Visual Studio prior	00.00	prodidLinker512
0x0014	Visual Studio prior	00.00	prodidCvtomf512
0x0015	Visual Studio prior	00.00	prodidUtc12_C_Std
0x0016	Visual Studio prior	00.00	prodidUtc12_CPP_Std
0x0017	Visual Studio prior	00.00	prodidUtc12_C_Book

0x0018	Visual Studio prior	00.00	prodidUtc12_CPP_Book
0x0019	Visual Studio prior	00.00	prodidImplib700
0x001a	Visual Studio prior	00.00	prodidCvtomf700
0x001b	Visual Studio prior	00.00	prodidUtc13_Basic
0x001c	Visual Studio prior	00.00	prodidUtc13_C
0x001d	Visual Studio prior	00.00	prodidUtc13_CPP
0x001e	Visual Studio prior	00.00	prodidLinker610
0x001f	Visual Studio prior	00.00	prodidCvtomf610
0x0020	Visual Studio prior	00.00	prodidLinker601
0x0021	Visual Studio prior	00.00	prodidCvtomf601
0x0022	Visual Studio prior	00.00	prodidUtc12_1_Basic
0x0023	Visual Studio prior	00.00	prodidUtc12_1_C
0x0024	Visual Studio prior	00.00	prodidUtc12_1_CPP
0x0025	Visual Studio prior	00.00	prodidLinker620
0x0026	Visual Studio prior	00.00	prodidCvtomf620
0x0027	Visual Studio prior	00.00	prodidAliasObj70
0x0028	Visual Studio prior	00.00	prodidLinker621
0x0029	Visual Studio prior	00.00	prodidCvtomf621
0x002a	Visual Studio prior	00.00	prodidMasm615
0x002b	Visual Studio prior	00.00	prodidUtc13_LTCG_C
0x002c	Visual Studio prior	00.00	prodidUtc13_LTCG_CPP
0x002d	Visual Studio prior	00.00	prodidMasm620
0x002e	Visual Studio prior	00.00	prodidILAsm100
0x002f	Visual Studio prior	00.00	prodidUtc12_2_Basic
0x0030	Visual Studio prior	00.00	prodidUtc12_2_C
0x0031	Visual Studio prior	00.00	prodidUtc12_2_CPP
0x0032	Visual Studio prior	00.00	prodidUtc12_2_C_Std
0x0033	Visual Studio prior	00.00	prodidUtc12_2_CPP_Std
0x0034	Visual Studio prior	00.00	prodidUtc12_2_C_Book
0x0035	Visual Studio prior	00.00	prodidUtc12_2_CPP_Book
0x0036	Visual Studio prior	00.00	prodidImplib622
0x0037	Visual Studio prior	00.00	prodidCvtomf622
0x0038	Visual Studio prior	00.00	prodidCvtres501
0x0039	Visual Studio prior	00.00	prodidUtc13_C_Std
0x003a	Visual Studio prior	00.00	prodidUtc13_CPP_Std
0x003b	Visual Studio prior	00.00	prodidCvtpgd1300
0x003c	Visual Studio prior	00.00	prodidLinker622
0x003d	Visual Studio prior	00.00	prodidLinker700
0x003e	Visual Studio prior	00.00	prodidExport622
0x003f	Visual Studio prior	00.00	prodidExport700
0x0040	Visual Studio prior	00.00	prodidMasm700
0x0041	Visual Studio prior	00.00	prodidUtc13_POGO_I_C

0x0042	Visual Studio prior	00.00	prodidUtc13_POGO_I_CPP
0x0043	Visual Studio prior	00.00	prodidUtc13_POGO_O_C
0x0044	Visual Studio prior	00.00	prodidUtc13_POGO_O_CPP
0x0045	Visual Studio prior	00.00	prodidCvtres700
0x0046	Visual Studio prior	00.00	prodidCvtres710p
0x0047	Visual Studio prior	00.00	prodidLinker710p
0x0048	Visual Studio prior	00.00	prodidCvtomf710p
0x0049	Visual Studio prior	00.00	prodidExport710p
0x004a	Visual Studio prior	00.00	prodidImplib710p
0x004b	Visual Studio prior	00.00	prodidMasm710p
0x004c	Visual Studio prior	00.00	prodidUtc1310p_C
0x004d	Visual Studio prior	00.00	prodidUtc1310p_CPP
0x004e	Visual Studio prior	00.00	prodidUtc1310p_C_Std
0x004f	Visual Studio prior	00.00	prodidUtc1310p_CPP_Std
0x0050	Visual Studio prior	00.00	prodidUtc1310p_LTCG_C
0x0051	Visual Studio prior	00.00	prodidUtc1310p_LTCG_CPP
0x0052	Visual Studio prior	00.00	prodidUtc1310p_POGO_I_C
0x0053	Visual Studio prior	00.00	prodidUtc1310p_POGO_I_CPP
0x0054	Visual Studio prior	00.00	prodidUtc1310p_POGO_O_C
0x0055	Visual Studio prior	00.00	prodidUtc1310p_POGO_O_CPP
0x0056	Visual Studio prior	00.00	prodidLinker624
0x0057	Visual Studio prior	00.00	prodidCvtomf624
0x0058	Visual Studio prior	00.00	prodidExport624
0x0059	Visual Studio prior	00.00	prodidImplib624
0x005a	Visual Studio 2003	07.10	prodidLinker710
0x005b	Visual Studio 2003	07.10	prodidCvtomf710
0x005c	Visual Studio 2003	07.10	prodidExport710
0x005d	Visual Studio 2003	07.10	prodidImplib710
0x005e	Visual Studio 2003	07.10	prodidCvtres710
0x005f	Visual Studio 2003	07.10	prodidUtc1310_C
0x0060	Visual Studio 2003	07.10	prodidUtc1310_CPP
0x0061	Visual Studio 2003	07.10	prodidUtc1310_C_Std
0x0062	Visual Studio 2003	07.10	prodidUtc1310_CPP_Std
0x0063	Visual Studio 2003	07.10	prodidUtc1310_LTCG_C
0x0064	Visual Studio 2003	07.10	prodidUtc1310_LTCG_CPP
0x0065	Visual Studio 2003	07.10	prodidUtc1310_POGO_I_C
0x0066	Visual Studio 2003	07.10	prodidUtc1310_POGO_I_CPP
0x0067	Visual Studio 2003	07.10	prodidUtc1310_POGO_O_C
0x0068	Visual Studio 2003	07.10	prodidUtc1310_POGO_O_CPP
0x0069	Visual Studio 2003	07.10	prodidAliasObj710
0x006a	Visual Studio 2003	07.10	prodidAliasObj710p
0x006b	Visual Studio 2003	07.10	prodidCvtpgd1310

0x006c	Visual Studio 2003	07.10	prodidCvtpgd1310p
0x006d	Visual Studio 2005	07.10	prodidUtc1400_C
0x006e	Visual Studio 2005	08.00	prodidUtc1400_CPP
0x006f	Visual Studio 2005	08.00	prodidUtc1400_C_Std
0x0070	Visual Studio 2005	08.00	prodidUtc1400_CPP_Std
0x0071	Visual Studio 2005	08.00	prodidUtc1400_LTCG_C
0x0072	Visual Studio 2005	08.00	prodidUtc1400_LTCG_CPP
0x0073	Visual Studio 2005	08.00	prodidUtc1400_POGO_I_C
0x0074	Visual Studio 2005	08.00	prodidUtc1400_POGO_I_CPP
0x0075	Visual Studio 2005	08.00	prodidUtc1400_POGO_O_C
0x0076	Visual Studio 2005	08.00	prodidUtc1400_POGO_O_CPP
0x0077	Visual Studio 2005	08.00	prodidCvtpgd1400
0x0078	Visual Studio 2005	08.00	prodidLinker800
0x0079	Visual Studio 2005	08.00	prodidCvtomf800
0x007a	Visual Studio 2005	08.00	prodidExport800
0x007b	Visual Studio 2005	08.00	prodidImplib800
0x007c	Visual Studio 2005	08.00	prodidCvtres800
0x007d	Visual Studio 2005	08.00	prodidMasm800
0x007e	Visual Studio 2005	08.00	prodidAliasObj800
0x007f	Visual Studio 2005	08.00	prodidPhoenixPrerelease
0x0080	Visual Studio 2005	08.00	prodidUtc1400_CVTCIL_C
0x0081	Visual Studio 2005	08.00	prodidUtc1400_CVTCIL_CPP
0x0082	Visual Studio 2005	08.00	prodidUtc1400_LTCG_MSIL
0x0083	Visual Studio 2005	08.00	prodidUtc1500_C
0x0084	Visual Studio 2008	09.00	prodidUtc1500_CPP
0x0085	Visual Studio 2008	09.00	prodidUtc1500_C_Std
0x0086	Visual Studio 2008	09.00	prodidUtc1500_CPP_Std
0x0087	Visual Studio 2008	09.00	prodidUtc1500_CVTCIL_C
0x0088	Visual Studio 2008	09.00	prodidUtc1500_CVTCIL_CPP
0x0089	Visual Studio 2008	09.00	prodidUtc1500_LTCG_C
0x008a	Visual Studio 2008	09.00	prodidUtc1500_LTCG_CPP
0x008b	Visual Studio 2008	09.00	prodidUtc1500_LTCG_MSIL
0x008c	Visual Studio 2008	09.00	prodidUtc1500_POGO_I_C
0x008d	Visual Studio 2008	09.00	prodidUtc1500_POGO_I_CPP
0x008e	Visual Studio 2008	09.00	prodidUtc1500_POGO_O_C
0x008f	Visual Studio 2008	09.00	prodidUtc1500_POGO_O_CPP
0x0090	Visual Studio 2008	09.00	prodidCvtpgd1500
0x0091	Visual Studio 2008	09.00	prodidLinker900
0x0092	Visual Studio 2008	09.00	prodidExport900
0x0093	Visual Studio 2008	09.00	prodidImplib900
0x0094	Visual Studio 2008	09.00	prodidCvtres900
0x0095	Visual Studio 2008	09.00	prodidMasm900

0x0096	Visual Studio 2008	09.00	prodidAliasObj900
0x0097	Visual Studio 2008	09.00	prodidResource
0x0098	Visual Studio 2010	10.00	prodidAliasObj1000
0x0099	Visual Studio 2010	10.00	prodidCvtpgd1600
0x009a	Visual Studio 2010	10.00	prodidCvtres1000
0x009b	Visual Studio 2010	10.00	prodidExport1000
0x009c	Visual Studio 2010	10.00	prodidImplib1000
0x009d	Visual Studio 2010	10.00	prodidLinker1000
0x009e	Visual Studio 2010	10.00	prodidMasm1000
0x009f	Visual Studio 2010	10.00	prodidPhx1600_C
0x00a0	Visual Studio 2010	10.00	prodidPhx1600_CPP
0x00a1	Visual Studio 2010	10.00	prodidPhx1600_CVTCIL_C
0x00a2	Visual Studio 2010	10.00	prodidPhx1600_CVTCIL_CPP
0x00a3	Visual Studio 2010	10.00	prodidPhx1600_LTCG_C
0x00a4	Visual Studio 2010	10.00	prodidPhx1600_LTCG_CPP
0x00a5	Visual Studio 2010	10.00	prodidPhx1600_LTCG_MSIL
0x00a6	Visual Studio 2010	10.00	prodidPhx1600_POGO_I_C
0x00a7	Visual Studio 2010	10.00	prodidPhx1600_POGO_I_CPP
0x00a8	Visual Studio 2010	10.00	prodidPhx1600_POGO_O_C
0x00a9	Visual Studio 2010	10.00	prodidPhx1600_POGO_O_CPP
0x00aa	Visual Studio 2010	10.00	prodidUtc1600_C
0x00ab	Visual Studio 2010	10.00	prodidUtc1600_CPP
0x00ac	Visual Studio 2010	10.00	prodidUtc1600_CVTCIL_C
0x00ad	Visual Studio 2010	10.00	prodidUtc1600_CVTCIL_CPP
0x00ae	Visual Studio 2010	10.00	prodidUtc1600_LTCG_C
0x00af	Visual Studio 2010	10.00	prodidUtc1600_LTCG_CPP
0x00b0	Visual Studio 2010	10.00	prodidUtc1600_LTCG_MSIL
0x00b1	Visual Studio 2010	10.00	prodidUtc1600_POGO_I_C
0x00b2	Visual Studio 2010	10.00	prodidUtc1600_POGO_I_CPP
0x00b3	Visual Studio 2010	10.00	prodidUtc1600_POGO_O_C
0x00b4	Visual Studio 2010	10.00	prodidUtc1600_POGO_O_CPP
0x00b5	Visual Studio 2010	10.10	prodidAliasObj1010
0x00b6	Visual Studio 2010	10.10	prodidCvtpgd1610
0x00b7	Visual Studio 2010	10.10	prodidCvtres1010
0x00b8	Visual Studio 2010	10.10	prodidExport1010
0x00b9	Visual Studio 2010	10.10	prodidImplib1010
0x00ba	Visual Studio 2010	10.10	prodidLinker1010
0x00bb	Visual Studio 2010	10.10	prodidMasm1010
0x00bc	Visual Studio 2010	10.10	prodidUtc1610_C
0x00bd	Visual Studio 2010	10.10	prodidUtc1610_CPP
0x00be	Visual Studio 2010	10.10	prodidUtc1610_CVTCIL_C
0x00bf	Visual Studio 2010	10.10	prodidUtc1610_CVTCIL_CPP

0x00c0	Visual Studio 2010	10.10	prodidUtc1610_LTCG_C
0x00c1	Visual Studio 2010	10.10	prodidUtc1610_LTCG_CPP
0x00c2	Visual Studio 2010	10.10	prodidUtc1610_LTCG_MSIL
0x00c3	Visual Studio 2010	10.10	prodidUtc1610_POGO_I_C
0x00c4	Visual Studio 2010	10.10	prodidUtc1610_POGO_I_CPP
0x00c5	Visual Studio 2010	10.10	prodidUtc1610_POGO_O_C
0x00c6	Visual Studio 2010	10.10	prodidUtc1610_POGO_O_CPP
0x00c7	Visual Studio 2012	11.00	prodidAliasObj1100
0x00c8	Visual Studio 2012	11.00	prodidCvtpgd1700
0x00c9	Visual Studio 2012	11.00	prodidCvtres1100
0x00ca	Visual Studio 2012	11.00	prodidExport1100
0x00cb	Visual Studio 2012	11.00	prodidImplib1100
0x00cc	Visual Studio 2012	11.00	prodidLinker1100
0x00cd	Visual Studio 2012	11.00	prodidMasm1100
0x00ce	Visual Studio 2012	11.00	prodidUtc1700_C
0x00cf	Visual Studio 2012	11.00	prodidUtc1700_CPP
0x00d0	Visual Studio 2012	11.00	prodidUtc1700_CVTCIL_C
0x00d1	Visual Studio 2012	11.00	prodidUtc1700_CVTCIL_CPP
0x00d2	Visual Studio 2012	11.00	prodidUtc1700_LTCG_C
0x00d3	Visual Studio 2012	11.00	prodidUtc1700_LTCG_CPP
0x00d4	Visual Studio 2012	11.00	prodidUtc1700_LTCG_MSIL
0x00d5	Visual Studio 2012	11.00	prodidUtc1700_POGO_I_C
0x00d6	Visual Studio 2012	11.00	prodidUtc1700_POGO_I_CPP
0x00d7	Visual Studio 2012	11.00	prodidUtc1700_POGO_O_C
0x00d8	Visual Studio 2012	11.00	prodidUtc1700_POGO_O_CPP
0x00d9	Visual Studio 2013	12.00	prodidAliasObj1200
0x00da	Visual Studio 2013	12.00	prodidCvtpgd1800
0x00db	Visual Studio 2013	12.00	prodidCvtres1200
0x00dc	Visual Studio 2013	12.00	prodidExport1200
0x00dd	Visual Studio 2013	12.00	prodidImplib1200
0x00de	Visual Studio 2013	12.00	prodidLinker1200
0x00df	Visual Studio 2013	12.00	prodidMasm1200
0x00e0	Visual Studio 2013	12.00	prodidUtc1800_C
0x00e1	Visual Studio 2013	12.00	prodidUtc1800_CPP
0x00e2	Visual Studio 2013	12.00	prodidUtc1800_CVTCIL_C
0x00d3	Visual Studio 2013	12.00	prodidUtc1800_CVTCIL_CPP
0x00e4	Visual Studio 2013	12.00	prodidUtc1800_LTCG_C
0x00e5	Visual Studio 2013	12.00	prodidUtc1800_LTCG_CPP
0x00e6	Visual Studio 2013	12.00	prodidUtc1800_LTCG_MSIL
0x00e7	Visual Studio 2013	12.00	prodidUtc1800_POGO_I_C
0x00e8	Visual Studio 2013	12.00	prodidUtc1800_POGO_I_CPP
0x00e9	Visual Studio 2013	12.00	prodidUtc1800_POGO_O_C

0x00ea	Visual Studio 2013	12.00	prodidUtc1800_POGO_O_CPP
0x00eb	Visual Studio 2013	12.10	prodidAliasObj1210
0x00ec	Visual Studio 2013	12.10	prodidCvtpgd1810
0x00ed	Visual Studio 2013	12.10	prodidCvtres1210
0x00ee	Visual Studio 2013	12.10	prodidExport1210
0x00ef	Visual Studio 2013	12.10	prodidImplib1210
0x00f0	Visual Studio 2013	12.10	prodidLinker1210
0x00f1	Visual Studio 2013	12.10	prodidMasm1210
0x00f2	Visual Studio 2013	12.10	prodidUtc1810_C
0x00f3	Visual Studio 2013	12.10	prodidUtc1810_CPP
0x00f4	Visual Studio 2013	12.10	prodidUtc1810_CVTCIL_C
0x00f5	Visual Studio 2013	12.10	prodidUtc1810_CVTCIL_CPP
0x00f6	Visual Studio 2013	12.10	prodidUtc1810_LTCG_C
0x00f7	Visual Studio 2013	12.10	prodidUtc1810_LTCG_CPP
0x00f8	Visual Studio 2013	12.10	prodidUtc1810_LTCG_MSIL
0x00f9	Visual Studio 2013	12.10	prodidUtc1810_POGO_I_C
0x00fa	Visual Studio 2013	12.10	prodidUtc1810_POGO_I_CPP
0x00fb	Visual Studio 2013	12.10	prodidUtc1810_POGO_O_C
0x00fc	Visual Studio 2013	12.10	prodidUtc1810_POGO_O_CPP
0x00fd	Visual Studio 2015	14.00	prodidAliasObj1400
0x00fe	Visual Studio 2015	14.00	prodidCvtpgd1900
0x00ff	Visual Studio 2015	14.00	prodidCvtres1400
0x0100	Visual Studio 2015	14.00	prodidExport1400
0x0101	Visual Studio 2015	14.00	prodidImplib1400
0x0102	Visual Studio 2015	14.00	prodidLinker1400
0x0103	Visual Studio 2015	14.00	prodidMasm1400
0x0104	Visual Studio 2015	14.00	prodidUtc1900_C
0x0105	Visual Studio 2015	14.00	prodidUtc1900_CPP
0x0106	Visual Studio 2015	14.00	prodidUtc1900_CVTCIL_C
0x0107	Visual Studio 2015	14.00	prodidUtc1900_CVTCIL_CPP
0x0108	Visual Studio 2015	14.00	prodidUtc1900_LTCG_C
0x0109	Visual Studio 2015	14.00	prodidUtc1900_LTCG_CPP
0x010a	Visual Studio 2015	14.00	prodidUtc1900_LTCG_MSIL
0x010b	Visual Studio 2015	14.00	prodidUtc1900_POGO_I_C
0x010c	Visual Studio 2015	14.00	prodidUtc1900_POGO_I_CPP
0x010d	Visual Studio 2015	14.00	prodidUtc1900_POGO_O_C
0x010e	Visual Studio 2015	14.00	prodidUtc1900_POGO_O_CP

List of Abbreviations

ACL Access Control Layer.

AMQP Advanced Message Queuing Protocol.

API Application Programming Interface.

APT Advanced Persistent Threat.

ASN Autonomous System Number.

AV Antivirus.

CERT Computer Emergency Response Team.

CFG Control Flow Graph.

CRITs Collaborative Research Into Threats. *See Glossary:* CRITs.

CSP Communicating Sequential Processes. *See Glossary:* CSP.

DAO Decentralized Autonomous Organization.

DGA Domain Generating Algorithm.

DIKW Data-Information-Knowledge-Wisdom. *See Glossary:* DIKW.

DNS Domain Name System.

ESB Enterprise Service Bus.

FBI Federal Bureau of Investigation.

FS-ISAC Financial Services - Information Sharing and Analysis Center. *See Glossary:* Information Sharing and Analysis Center.

HDFS Hadoop Distributed File System. *See Glossary:* HDFS.

HTTP Hypertext Transfer Protocol.

IAT Import Address Table.

IC Intelligence Community.

IOC Indicator of Compromise. *See Glossary:* IOC.

IP Internet Protocol.

ISAC Information Sharing and Analysis Center. *See Glossary:* Information Sharing and Analysis Center.

JSON JavaScript Object Notation.

MANTIS Model-based Analysis of Threat Intelligence Sources. *See Glossary:* MANTIS.

MISP Malware Information Sharing Platform. *See Glossary:* MISP.

MSVC Microsoft Visual C++.

PCAP Packet CAPture.

PDF Portable Document Format.

PE32 Portable Executable.

PoS Proof of Stake.

PoW Proof of Work.

protobuf Protocol Buffers.

QoS Quality of Service.

RESTful REpresentational State Transfer.

RFI Request for Information.

S3 Amazon's Simple Storage Service.

SIEM Security Information and Event Management.

SNA Social Network Analysis.

SOA Service Oriented Architectures.

SWIFT Society for Worldwide Interbank Financial Telecommunications.

TLS Transport Layer Security.

TTL Time to Live.

URL Uniform Resource Locator.

US-CERT United States Computer Emergency Response Team.

UUIDv4 Universally Unique IDentifier v4. *See Glossary:* UUID.

VMI Virtual Machine Introspection.

zk-SNARK Zero Knowledge Succinct Noninteractive ARgument of Knowledge.

Glossary

Actor Model The Actor Model is a model for designing concurrent programs that uses asynchronous mailboxes for message passing. *See Section:* 2.4.1.

Apache Hadoop A framework for processing very large data sets that are reliable, scalable, and distributed. Originally designed as an open source implementation of the MapReduce algorithm, Apache Hadoop is now comprised of multiple components that support the analysis of structured and unstructured data.

Apache Kafka Apache Kafka is a distributed message broker that is focused on high performance and scalability. While originally designed for log files, Kafka is commonly used for moving messages in stream processing systems.

APT1 An APT actor commonly attributed to the Chinese Government.

blockchain A distributed database that is optimized for security and byzantine fault tolerance. This has been popularized by Bitcoin because blockchain provides the technical foundations.

CARE A new model for sharing computer security artifacts which aims to provide the mechanisms required to perform analytic collaboration with a collective pool of knowledge in near real-time. *See Chapter:* 4.

CareCoin An economic unit that is used for transmitting artifacts between peers under the CARE model.

CAREconomy An economic model for how to incentive sharing under the CARE model.

Citadel A family of malware based on the leaked Zeus Trojan targeting backing activity. Often called Citadel or Zeus-Citadel.

CRITs A popular analytic tool developed by MITRE for conducting triage of objects. CRITs is designed to support multi-users and be deployed on servers.

CSP Communicating Sequential Processes is a model for designing concurrent programs that uses synchronous channels for message passing. *See Section:* 2.4.1.

Cuckoo Sandbox A widely popular host-based dynamic analysis system.

Data A layer of the DIKW model which is defined as a raw object with no further meaning.

DIKW A model for how to transform raw *Data* into usable forms that can empower understanding and a judgment. *See Section:* 2.1.1.

Drakvuf A dynamic analysis system based on VMI.

dynamic analysis The process of extracting *Information* from *Data* through executing *Data* and observing its behavior. *See Section:* 2.2.2.

Feature extraction The process of refining *Information* for the purpose of machine learning.

free riding The situation where participants in an economy reap the benefits of the collective whole but do not provide a meaningful contribution.

Gateway A SKALD *Planner* whose primary purpose is to receive tasking. *See Section:* 3.2.2.1.

HDFS A core component of the Apache Hadoop ecosystem that provides a distributed file system for managing very large data sets.

Holmes Processing An open-source platform that realized the goals of SKALD and enables large-scale analysis of security artifacts. More information can be learned in chapter 5 and `https://www.holmesprocessing.com/`.

impfuzzy Performs fuzzy hashing of the IAT that is reported in the headers of the PE32 file format.

Information A layer of the DIKW model which is defined as the details obtained by asking who, what, where, when, and how questions of *Data*.

Information Sharing and Analysis Center Created by the United States Federal Government in 2003 with the mission of facilitating sharing between peers in critical sectors and the federal government.

Intelligence Cycle A process for performing investigations that has been widely adopted by the intelligence and law enforcement communities. *See Section:* 2.1.2.

Interrogation A SKALD *Planner* whose primary purpose is to turn *Information* into *Knowledge*. *See Section:* 3.2.2.4.

Investigation A SKALD *Planner* whose primary purpose is to orchestrate *Information* extraction against *Data*. *See Section:* 3.2.2.2.

IOC An ontology for cataloging details about an object or event.

Knowledge A layer of the DIKW model which is defined as organizing a set or subset of *Information* into useful forms by asking "how-to" questions.

Lazarus Group An APT actor responsible for the Sony and SWIFT attacks.

Loosely coupled A design pattern where a component of a system is not reliant on another component.

Machine Learning A field of computer science that focuses on creating algorithms that learn how to solve a problem without being provided specific instructions. *See Section:* 2.3.2.

Maltrieve A distribution feed that collects malware samples from blacklists.

MANTIS Created by Siemens Corporations to be a scalable competitor to CRITs that also focused on sharing IOCs. This system was abandoned by Siemens as the popularity of MISP increased.

MapReduce A programming model for processing large data sets across distributed clusters.

MD5 A one way hashing algorithm that creates a message digest of 128 bits. Originally was the dominant hashing algorithm used to identify malicious binaries.

Mediyes A signed malware dropper that used a valid signature and heavily targeted western countries, particularly Germany.

Microservices Is an architectural design pattern similar to SOA. However, *Service*s in the microservice pattern focuses on small tasks that are required by the overall application. *See Section:* 2.4.3.

MISP An IOC sharing and storage platform that is sponsored by the European Union.

Ops-T A cyber security trust group that aims to promote collaboration amongst vetted security professionals to promote responsible action against malicious activities.

pefile A popular library for extracting the headers and structure from a PE32 binary.

PEHash A popular hashing technique for PE32 binaries.

PEInfo A commonly used static analysis tool, originally part of CRITs, for extracting the headers from PE32 binaries.

Planner A component of SKALD that operates as an intelligent orchestrator for *Service*s. *See Section:* 3.2.1.

Presentation A SKALD *Planner* whose primary purpose is to provide a standard mechanism for interacting with stored data and the data the INTERROGATION *Planner* generates. *See Section:* 3.2.2.5.

RabbitMQ RabbitMQ is a popular and feature rich message broker. It provides native support for multiple message formats, such as AMQP, and programming languages.

Service A subset of a system that executes an element of work or a business function. *See Section:* 3.2.3.

SHA-256 A one way hashing algorithm originally designed by the NSA that creates a message digest of 256 bits. Recently has begun to overtake MD5 as the default hash for malicious binaries.

Shadowserver A cyber security trust group that focuses on gathering intelligence on malware, botnets, and computer fraud.

Skald An architecture which guides the creation of analytic systems that support the investigations of malicious activities plaguing computer systems, *See Chapter:* 3.

smart contract The evolution of paper contracts applied to the digital world. *See Section:* 2.5.2.

static analysis The process of extracting *Information* from *Data* by investigating the code or structure of the *Data*. *See Section:* 2.2.1.

Storage A SKALD *Planner* whose primary purpose is to optimize the storage and retrieval of security artifacts and details that support the functioning of the system. *See Section:* 3.2.2.3.

Tightly coupled A design pattern where a component of a system is reliant on another component.

Transport A component of SKALD that moves data between *Planner*s. *See Section:* 3.2.4.

UUID A Universally Unique IDentifier (UUID) that guarantees uniqueness across space and time.

VIPER An analytic tool for extracting information from raw computer security data *Data*. VIPRE is designed to empower a single security practitioner and run on their workstation.

VirusShare A distribution feed for raw malware samples.

VirusTotal A service provided by Google for analyzing if a file or URL is detected by an AV product.

Wisdom A layer of the DIKW model which is defined as the results from the development of understanding based on sets of *Knowledge* and experience.

Yara A language used to write a signature that is often based on textural or binary patterns.

Yara Exchange A cyber security trust group which specializes in the exchange of Yara signatures.

Bibliography

[1] Microsoft and Marsh, "By the Numbers: Global Cyber Risk Perception Survey," 2018.

[2] S. Farhang and J. Grossklags, "When to Invest in Security? Empirical Evidence and a Game-Theoretic Approach for Time-Based Security," in *Workshop on the Economics of Information Security*, June 2017. [Online]. Available: http://arxiv.org/abs/1706.00302

[3] D. Borak and K. Vasel, "The Equifax Hack Could Be Worse Than We Thought," February 2018. [Online]. Available: http://money.cnn.com/2018/02/09/pf/equifax-hack-senate-disclosure/index.html

[4] J. Chaffetz, M. Meadows, and W. Hurd, "The OPM Data Breach: How the Government Jeopardized Our National Security for More than a Generation," *US House of Representatives Committee on Oversight and Government Reform, 114th Congress*, 2016.

[5] D. Volz and T. Gardner, "In a First, U.S. Blames Russia for Cyber Attacks on Energy Grid," March 2018.

[6] S. J. Shackelford, "From Nuclear War to Net War: Analogizing Cyber Attacks in International Law," *Berkeley J. Int'l Law*, vol. 27, no. 192, 2009.

[7] S. Morgan, "Cybersecurity Market Report," Tech. Rep., 2017. [Online]. Available: https://cybersecurityventures.com/cybersecurity-market-report/

[8] 100th Congress, "H.R. 145 - Computer Security Act of 1987," pp. 100–235, 1988.

[9] B. Clinton, "Presidential Decision Directives/NSC-63," p. 68, 1998.

[10] 107th Congress, "Homeland Security Act of 2002," pp. 107–296, 2002.

[11] B. Obama, "Promoting Private Sector Cybersecurity Information Sharing, Executive Order 13691," 2015.

[12] K. Choo, "The Cyber Threat Landscape: Challenges and Future Research Directions," *Computers and Security*, vol. 30, no. 8, pp. 719–731, 2011.

[13] G. Ollmann, "Behind Today's Crimeware Installation Lifecycle: How Advanced Malware Morphs to Remain Stealthy and Persistent," Damballa, Tech. Rep., 2011.

[14] The MITRE Corporation, "Collaborative Research Into Threats (CRITs)," 2014.

[15] P. Vixie, "Internet Security Marketing: Buyer Beware," *CircleID*, April 2015. [Online]. Available: http://www.circleid.com/posts/20150420_internet_security_marketing_buyer_beware/

[16] A. Stamo, "The failure of the security industry," p. 7, April 2015. [Online]. Available: http://www.scmagazine.com/the-failure-of-the-security-industry/article/403261/

[17] McAfee Labs, "McAfee Threats Report: Fourth Quarter 2011," *Intel Corporation*, vol. Q4, pp. 1–24, 2011.

[18] VirusTotal, "File Statistics," April 2015. [Online]. Available: https://www.virustotal.com/en/statistics/

[19] L. Zeltser, "SANS - Managing and Exploring Malware Samples with Viper," June 2014. [Online]. Available: https://digital-forensics.sans.org/blog/2014/06/04/managing-and-exploring-malware-samples-with-viper

[20] B. Grobauer, S. Berger, J. Göbel, T. Schreck, and J. Wallinger, "The MANTIS Framework: Cyber Threat Intelligence Management for CERTs," *Proceedings of the 26th Annual FIRST Conference on Computer Security Incident Handling*, 2014.

[21] G. Webster, Z. Hanif, A. Ludwig, T. Lengyel, A. Zarras, and C. Eckert, "SKALD: A Scalable Architecture for Feature Extraction, Multi-user Analysis, and Real-Time Information Sharing," in *Proceedings of the 19th International Conference on Information Security*. Springer, 2016, pp. 231–249.

[22] K. Murphy, *Machine Learning: A Probabilistic Perspective.* MIT Press, 2012.

[23] D. D. Woods, E. S. Patterson, E. M. Roth, and K. Christoffersen, "Can We Ever Escape from Data Overload? A Cognitive Systems Diagnosis," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 43, no. 3, pp. 174–178, 1999.

[24] B. Anderson, C. Storlie, and T. Lane, "Improving Malware Classification: Bridging the Static/Dynamic Gap," in *ACM Workshop on Security and Artificial Intelligence*, 2012.

[25] Damballa Inc, “3% to 5% of Enterprise Assets are Compromised by Bot-driven Targeted Attack Malware,” pp. 2008–2010, March 2008. [Online]. Available: https://www.prnewswire.com/news-releases/3-to-5-of-enterprise-assets-are-compromised-by-bot-driven-targeted-attack-malware-61634867.html

[26] S. Shevchenko and A. Nish, “Cyber Heist Attribution,” *BAE Systems Threat Research Blog*, 2016.

[27] Symantec, “SWIFT Attackers’ Malware Linked to More Financial Attacks,” May 2016.

[28] T. Bergin and J. Finkle, “Exclusive: SWIFT Confirms New Cyber Thefts, Hacking Tactics,” *Reuters*, December 2016.

[29] G. Fisk, C. Ardi, N. Pickett, J. Heidemann, M. Fisk, and C. Papadopoulos, “Privacy Principles for Sharing Cyber Security Data,” in *Proceedings of the IEEE International Workshop on Privacy Engineering.* IEEE, 2015, pp. 193–197.

[30] C. Sauerwein, C. Sillaber, A. Mussmann, and R. Breu, “Threat Intelligence Sharing Platforms: An Exploratory Study of Software Vendors and Research Perspectives,” in *Proceedings of the 13th International Conference on Wirtschaftsinformatik (WI 2017)*, 2017, pp. 837–851.

[31] Novetta Threat Research Group, “Operation Blockbuster - Unraveling the Long Thread of the Sony Attack,” February 2016.

[32] Novetta, “Operation SMN: Axiom Threat Actor Group Report.”

[33] B. Kolosnjaji, G. Eraisha, G. Webster, A. Zarras, and C. Eckert, “Empowering convolutional networks for malware classification and analysis,” in *2017 International Joint Conference on Neural Networks (IJCNN)*, Anchorage, AK, USA, 2017, pp. 3838–3845.

[34] B. Kolosnjaji, A. Zarras, T. Lengyel, G. Webster, and C. Eckert, “Adaptive Semantics-Aware Malware Classification,” in *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings*, J. Caballero, U. Zurutuza, and R. J. Rodríguez, Eds. Cham: Springer International Publishing, 2016, vol. 9721, pp. 419–439.

[35] G. Webster, B. Kolosnjaji, C. von Pentz, J. Kirsch, Z. Hanif, A. Zarras, and C. Eckert, “Finding the Needle: A Study of the PE32 Rich Header and Respective Malware Triage,” in *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings*, M. Polychronakis and M. Meier, Eds. Cham: Springer International Publishing, 2017, vol. 10327 LNCS, pp. 119–138.

[36] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, “Deep Learning for Classification of Malware System Call Sequences,” in *AI 2016: Advances in Artificial Intelligence. AI 2016. Lecture Notes in Computer Science*, B. H. Kang and Q. Bai, Eds. Cham: Springer International Publishing, 2016, vol. 9992 LNAI, pp. 137–149.

[37] GReAT, “The Devil’s in the Rich Header,” *Securelist*, March 2018. [Online]. Available: https://securelist.com/the-devils-in-the-rich-header/84348/

[38] J. Rowley, “The Wisdom Hierarchy: Representations of the DIKW Hierarchy,” *Journal of Information Science*, vol. 33, no. 2, pp. 163–180, 2007.

[39] R. Ackoff, “From Data to Wisdom,” *Journal of Applied Systems Analysis*, vol. 16, no. 1, p. 3–9, 1989.

[40] Director Of National Intelligence (ODNI), “IC Consumers Guide,” 2011.

[41] UK Ministry of Defence, “Understanding and Intelligence Support to Joint Operations (JDP 2-00),” *Joint Doctrine Publication*, p. 155, 2011.

[42] A. S. Hulnick, “What’s Wrong With the Intelligence Cycle,” *Intelligence and National Security*, vol. 21, no. 6, pp. 959–979, December 2006.

[43] United Nations, “Criminal intelligence - manual for analysts,” *United Nations Office on Drugs and Crime*, 2011.

[44] Recorded Future, “5 Phases of the Threat Intelligence Lifecycle.” [Online]. Available: https://www.recordedfuture.com/threat-intelligence-lifecycle/

[45] B. Chess and G. McGraw, “Static Analysis for Security,” *IEEE Security and Privacy*, vol. 2, no. 6, pp. 76–79, November 2004.

[46] P. Emanuelsson and U. Nilsson, “A Comparative Study of Industrial Static Analysis Tools,” *Electronic Notes in Theoretical Computer Science*, vol. 217, no. C, pp. 5–21, 2008.

[47] M. Beller, R. Bholanath, S. McIntosh, and A. Zaidman, “Analyzing the State of Static Analysis: A Large-Scale Evaluation in Open Source Software,” in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016, pp. 470–481.

[48] M. Fowler, “Continuous Integration,” *Integration: The VLSI Journal*, vol. 26, no. 1, pp. 1–6, 2006.

[49] Synopsys, “Coverity Scan - Travis CI Integration.” [Online]. Available: https://scan.coverity.com/travis_ci

[50] Jenkins, "Coverity Plugin." [Online]. Available: https://plugins.jenkins.io/coverity

[51] A. Miller, "A Hundred Days of Continuous Integration," *Agile, 2008. AGILE'08. Conference*, pp. 289–293, 2008.

[52] M. Reddy, "Building Example Services For Holmes Processing." [Online]. Available: https://www.holmesprocessing.com/gsoc/#portfolioModal5

[53] M. Pietrek, "An In-Depth Look into the Win32 Portable Executable File Format," *MSDN Magazine*, vol. 17, pp. 1–15, February 2002.

[54] P. Okane, S. Sezer, and K. McLaughlin, "Obfuscation: The Hidden Malware," *IEEE Security and Privacy*, vol. 9, no. 5, pp. 41–47, 2011.

[55] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, "A View on Current Malware Behaviors," *Proceedings of the 2nd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, p. 8, 2009.

[56] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: Malware Analysis via Hardware Virtualization Extensions," in *Proceedings of the 15th ACM Conference on Computer and Communications Security.* New York, New York, USA: ACM Press, 2008, pp. 51–62.

[57] A. Moser, C. Kruegel, and E. Kirda, "Exploring Multiple Execution Paths for Malware Analysis," in *2007 IEEE Symposium on Security and Privacy.* IEEE, May 2007, pp. 231–245.

[58] The Cuckoo Foundation, "Cuckoo Sandbox," 2011. [Online]. Available: http://cuckoosandbox.org

[59] B. Jain, M. B. Baig, D. Zhang, D. E. Porter, and R. Sion, "SoK: Introspections on Trust and the Semantic Gap," in *2014 IEEE Symposium on Security and Privacy.* IEEE, May 2014, pp. 605–620.

[60] T. Lengyel, S. Maresca, B. Payne, G. Webster, S. Vogl, and A. Kiayias, "Scalability, Fidelity and Stealth in the DRAKVUF Dynamic Malware Analysis System," in *Proceedings of the 30th Annual Computer Security Applications Conference on - ACSAC '14*, New Orleans, Louisiana, USA, 2014, pp. 386–395.

[61] D. Kirat, G. Vigna, and C. Kruegel, "BareCloud: Bare-metal Analysis-based Evasive Malware Detection," in *Proceedings of the 23rd USENIX Security Symposium*, 2014, pp. 287–301.

[62] A. M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society*, vol. s2-42, no. 1, pp. 230–265, January 1937.

[63] B. J. Copeland, "The Essential Turing Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life plus The Secrets of Enigma," *Cryptologia*, vol. 29, p. 613, 2004.

[64] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Ferná Ndez, and E. Vá Zquez, "Anomaly-based Network Intrusion Detection: Techniques, Systems and Challenges," *Computers & Security*, vol. 28, pp. 18–28, 2009.

[65] V. M. Alvarez, "Yara," 2015. [Online]. Available: https://virustotal.github.io/yara/

[66] M. Antonakakis and R. Perdisci, "From Throw-away Traffic to Bots: Detecting the Rise of DGA-based Malware," in *Proceedings of the 21st USENIX Security Symposium*, 2012, p. 16.

[67] G. Wicherski, "peHash: A Novel Approach to Fast Malware Clustering," in *Proceedings of the 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2008.

[68] S. Tomonaga, "Classifying Malware Using Import API and Fuzzy Hashing -Impfuzzy-," May 2016. [Online]. Available: http://blog.jpcert.or.jp/2016/05/classifying-mal-a988.html

[69] VirusTotal, "File Statistics," 2017. [Online]. Available: https://www.virustotal.com/en/statistics/

[70] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a Dynamic Reputation System for DNS," *Proceedings of the 19th USENIX Conference on Security*, pp. 1–17, 2010.

[71] V. E. Krebs, "Mapping networks of terrorist cells," *Connections*, vol. 24, no. 3, pp. 43–52, 2002.

[72] P. Klerks, "The network paradigm applied to criminal organisations: Theoretical nitpicking or a relevant doctrine for investigators? recent developments in the netherlands," in *Transnational Organised Crime.* Routledge, 2004, pp. 111–127.

[73] J. Xu and H. Chen, "Criminal network analysis and visualization," *Communications of the ACM*, vol. 48, no. 6, pp. 100–107, 2005.

[74] W. R. Harper and D. H. Harris, "The application of link analysis to police intelligence," *Human Factors*, vol. 17, no. 2, pp. 157–164, 1975.

[75] A. Singhal and X. Ou, "Security risk analysis of enterprise networks using probabilistic attack graphs," in *Network Security Metrics.* Springer, 2017, pp. 53–73.

[76] S. Jha, O. Sheyner, and J. Wing, "Two formal analyses of attack graphs," in *Computer Security Foundations Workshop, 2002. Proceedings. 15th IEEE.* IEEE, 2002, pp. 49–63.

[77] J. L. Obes, C. Sarraute, and G. Richarte, "Attack planning in the real world," *arXiv preprint arXiv:1306.4044*, 2013.

[78] R. W. Ritchey and P. Ammann, "Using model checking to analyze network vulnerabilities," in *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on.* IEEE, 2000, pp. 156–165.

[79] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Security and privacy, 2002. Proceedings. 2002 IEEE Symposium on.* IEEE, 2002, pp. 273–284.

[80] E. Casey, *Digital evidence and computer crime: Forensic science, computers, and the internet.* Academic press, 2011.

[81] Y. Chen, Y. Nadji, A. Kountouras, F. Monrose, R. Perdisci, M. Antonakakis, and N. Vasiloglou, "Practical attacks against graph-based clustering," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 2017, pp. 1125–1142.

[82] The Go Project, "The Go Programming Language." [Online]. Available: https://golang.org/project/

[83] R. Hickey, "The Clojure Programming Language," in *Proceedings of the 2008 Symposium on Dynamic Languages.* New York, New York, USA: ACM Press, 2008, pp. 1–1.

[84] R. Virding, C. Wikstrom, and M. Williams, *Concurrent Programming in ERLANG*, 2nd ed., J. Armstrong, Ed. Hertfordshire, UK, UK: Prentice Hall International (UK) Ltd., 1996.

[85] P. Haller, "On the Integration of the Actor Model in Mainstream Technologies," in *Proceedings of the 2nd Edition on Programming Systems, Languages and Applications Based on Actors, Agents, and Decentralized Control Abstractions.* New York, New York, USA: ACM Press, 2012, p. 1.

[86] C. Hewitt, P. Bishop, and R. Steiger, "A Universal Modular ACTOR Formalism for Artificial Intelligence," in *Proceedings of the 3rd International Joint Conference on Artificial Intelligence.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1973, pp. 235–245.

[87] E. Meijer, C. Hewitt, and C. Szyperski, "The Actor Model (Everything You Wanted to Know, but Were Afraid to Ask)," 2012.

[88] G. A. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott, “A Foundation for Actor Computation,” *Journal of Functional Programming*, vol. 7, no. 1, pp. 1–72, 1997.

[89] G. A. Agha, “ACTORS: A Model of Concurrent Computation in Distributed Systems,” Massachusetts Inst of Tech Cambridge Artifical Intelligence Lab, Tech. Rep., 1985.

[90] C. Hoare, “Communicating Sequential Processes,” *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, August 1978.

[91] R. Cox, “Bell Labs and CSP Threads.” [Online]. Available: https://swtch.com/~rsc/thread/

[92] A. Arsanjani, G. Booch, T. Boubez, P. C. Brown, D. Chappell, J. DeVadoss, Thomas Erl, N. Josuttis, D. Krafzig, M. Little, B. Loesgen, A. T. Manes, Joe McKendrick, S. Ross-Talbot, S. Tilkov, C. Utschig-Utschig, and H. Wilhelmsen, “SOA Manifesto,” *SOA Manifesto*, p. 35, 2009. [Online]. Available: http://www.soa-manifesto.org/

[93] M. P. Papazoglou, “Service-Oriented Computing: Concepts, Characteristics and Directions,” *Proceedings - 4th International Conference on Web Information Systems Engineering*, pp. 3–12, 2003.

[94] M. Fowler, “Service Oriented Ambiguity,” *MartinFowler.com*, 2005. [Online]. Available: https://martinfowler.com/bliki/ServiceOrientedAmbiguity.html

[95] D. Sprott and L. Wilkes, “Understanding Service-Oriented Architecture,” *The Architecture Journal*, vol. 1, no. 1, pp. 10–17, 2004.

[96] M. P. Papazoglou and V. D. Heuvel, “Service-Oriented Design and Development Methodology,” *International Journal of Web Engineering and Technology*, vol. 2, no. 4, pp. 412–442, 2006.

[97] M. P. Papazoglou and W. J. Van Den Heuvel, “Service Oriented Architectures: Approaches, Technologies and Research Issues,” *VLDB Journal*, vol. 16, no. 3, pp. 389–415, March 2007.

[98] D. Krafzig, K. Banke, and D. Slama, *Service-Oriented Architecture Best Practices*. Prentice Hall Professional, 2005.

[99] B. P. Padmanabhan, K. Sadekar, and G. Krishnan, “What’s trending on Netflix?” Netflix, Tech. Rep., 2015. [Online]. Available: https://medium.com/netflix-techblog/whats-trending-on-netflix-f00b4b037f61

[100] J. Lewis and M. Fowler, “Microservices: A Definition of This New Architectural Term,” *MartinFowler.com*, 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html

[101] S. Newman, *Building Microservices: Designing Fine-Grained Systems.* O'Reilly Media, Inc., 2015.

[102] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[103] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," *Conference on the Theory and Application of Cryptographic Techniques*, pp. 369–378, 1988.

[104] M. Jakobsson and A. Juels, "Proofs of Work and Bread Pudding Protocols," in *Secure Information Networks*, B. Preneel, Ed. Boston, MA: Springer US, 1999, pp. 258–272.

[105] K. Wüst and A. Gervais, "Do you need a Blockchain?" *IACR Cryptology ePrint Archive*, pp. 1–7, 2017. [Online]. Available: https://eprint.iacr.org/2017/375.pdf

[106] S. Popov, "The Tangle, IOTA Whitepaper," pp. 1–28, 2017. [Online]. Available: https://iota.org/IOTA_Whitepaper.pdf

[107] N. Szabo, "Formalizing and Securing Relationships on Public Networks," *First Monday*, vol. 2, no. 9, September 1997.

[108] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, R. Guerraoui, Ed. Springer Berlin Heidelberg, 2001, pp. 329–350.

[109] G. Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," *Ethereum Project Yellow Paper*, 2014. [Online]. Available: http://www.cryptopapers.net/papers/ethereum-yellowpaper.pdf

[110] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing - STOC '88.* New York, New York, USA: ACM Press, 1988, pp. 103–112.

[111] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference on - ITCS '12.* New York, New York, USA: ACM Press, 2012, pp. 326–349.

[112] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," *Proceedings - IEEE Symposium on Security and Privacy*, pp. 459–474, 2014. [Online]. Available: http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf

[113] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts,” in *Proceedings of the 2016 IEEE Symposium on Security and Privacy*, 2016, pp. 839–858.

[114] E. Team, “Byzantium HF Announcement,” 2017. [Online]. Available: https://blog.ethereum.org/2017/10/12/byzantium-hf-announcement/

[115] A. Mavridou and A. Laszka, “Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach,” in *Proceedings of the 22nd International Conference Financial Cryptography and Data Security*, 2018.

[116] L. Gordon, M. Loeb, and W. Lucyshyn, “An Economics Perspective on the Sharing of Information Related to Security Breaches: Concepts and Empirical Evidence,” in *Workshop on the Economics of Information Security (WEIS)*, 2002.

[117] E. Gal-Or and A. Chose, “The Economic Incentives for Sharing Security Information,” *Information Systems Research*, vol. 16, no. 2, pp. 186–208, 2005.

[118] W. Shields, “Problems with PeHash Implementations,” 2014. [Online]. Available: https://gist.github.com/wxsBSD/07a5709fdcb59d346e9e

[119] F. Cristian, “Understanding Fault-tolerant Distributed Systems,” *Communications of the ACM*, vol. 34, no. 2, pp. 56–78, February 1991.

[120] S. B. Fan, J. Shaw, H. L. Han, and L. P. Zhang, “Software configuration management,” in *Google Patents*, 2012, p. 14.

[121] A. Lakshman and P. Malik, “Cassandra: A Decentralized Structured Storage System,” *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, p. 35, 2010.

[122] Z. Hanif, T. Calhoun, and J. Trost, “BinaryPig: Scalable Static Binary Analysis Over Hadoop,” *Black Hat USA 2013*, p. 5, 2012.

[123] J. Jang, D. Brumley, and S. Venkataraman, “BitShred,” in *Proceedings of the 18th ACM Conference on Computer and Communications Security - CCS ’11*, 2011, p. 309. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2046707.2046742

[124] HiveMQ, “MQTT Essentials Part 6: Quality of Service 0, 1 & 2,” 2015. [Online]. Available: http://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels

[125] DeepEnd Research, “YaraExchange,” May 2017.

[126] D. Horng, P. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos, “Polonium: Tera-Scale Graph Mining and Inference for Malware Detection,” in *Siam International Conference on Data Mining*, 2011, pp. 131–142.

[127] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a System for Large-scale Graph Processing," in *Proceedings of the 2010 International Conference on Management of Data*, 2010, p. 135.

[128] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale Cluster Management at Google with Borg," in *Proceedings of the 10th European Conference on Computer Systems*, 2015, pp. 1–17.

[129] US-CERT, "Alert (TA14-353A)," December 2014. [Online]. Available: https://www.us-cert.gov/ncas/alerts/TA14-353A

[130] D. Bianco, "The Pyramid of Pain," *Enterprise Detection & Response*, 2013.

[131] A. Boyd, "How FBI Cyber Division Helps Agencies Investigate Intrusions," *Federal Times*, October 2015.

[132] CERT-Coordination Center, "CSIRT Frequently Asked Questions (FAQ)," pp. 1–10, 2016. [Online]. Available: https://www.cert.org/incident-management/csirt-development/csirt-faq.cfm?

[133] Operations Security Trust, "Ops-T," 2017.

[134] J. Connolly, M. Davidson, and C. Schmidt, "The Trusted Automated eXchange of Indicator Information (TAXII ™)," The MITRE Corporation, Tech. Rep., 2014. [Online]. Available: http://taxii.mitre.org/about/documents/Introduction_to_TAXII_White_Paper_May_2014.pdf

[135] C. Wagner, A. Dulaunoy, G. Wagener, and A. 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*. ACM, 2016, pp. 49–56.

[136] T. Moore and R. Clayton, "The consequence of non-cooperation in the fight against phishing," in *Proceedings of the eCrime Researchers Summit (eCrime)*, 2008. [Online]. Available: http://www.cl.cam.ac.uk/~rnc1/ecrime08pre.pdf

[137] Communications Security, Reliability and Interoperability Council, "Working Group 5: Cybersecurity Information Sharing - Information Sharing Barriers," no. jun, 2016.

[138] C. Johnson, L. Badger, D. Waltermire, J. Snyder, and C. Skorupka, "Guide to Cyber Threat Information Sharing," *NIST Special Publication*, vol. 800, no. 150, 2016.

[139] P. Cichonski, T. Millar, T. Grance, and K. Scarfone, "Computer Security Incident Handling Guide - Recommendations of the National Institute of Standards and Technology," *NIST Special Publication*, vol. 800, no. 61, 2012.

[140] T. Moore, R. Clayton, and R. Anderson, “The Economics of Online Crime,” *Journal of Economic Perspectives*, vol. 23, no. 3—Summer, pp. 3–20, 2009.

[141] J. Milletary, “Citadel Trojan Malware Analysis,” *Dell SecureWorks*, 2012.

[142] FS-ISAC, “Financial Services Information Sharing and Analysis Center,” 2015.

[143] Verizon, “2015 Data Breach Investigations Report,” 2015.

[144] Team Cymru, “#totalhash,” 2018.

[145] C. Guarnieri, “Viper - Time to Do Malware Research Right,” 2015.

[146] MISP Project, “MISP Taxonomies and Classification as Machine Tags,” Tech. Rep., 2018.

[147] D. Dittrich, “So You Want to Take Over a Botnet ...” *Proceedings of the 5th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, pp. 1–8, 2012.

[148] B. Krebs, “’Mariposa’ Botnet Authors May Avoid Jail Time,” March 2010.

[149] L. Gordon, M. Loeb, and W. Lucyshyn, “Sharing Information on Computer Systems Security: An Economic Analysis,” *Journal of Accounting and Public Policy*, vol. 22, no. 6, pp. 461–485, 2003.

[150] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, “KARMA: A Secure Economic Framework for Peer-to-Peer Resource Sharing,” in *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[151] M. Meulpolder, L. D’Acunto, and M. Capota, “Public and Private BitTorrent Communities: A Measurement Study,” *Proceedings of the 9th International Workshop on Peer-to-Peer Systems (IPTPS)*, p. 10, 2010.

[152] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, “Succinct Non-Interactive Arguments for a von Neumann Architecture,” *Proceedings of the 23rd USENIX Security Symposium*, pp. 781–796, 2013.

[153] V. Buterin, “Notes on Blockchain Governance,” 2017. [Online]. Available: https://vitalik.ca/general/2017/12/17/voting.html

[154] F. Ehrsam, “Blockchain Governance: Programming Our Future,” *Coinbase*, 2017. [Online]. Available: https://medium.com/@FEhrsam/blockchain-governance-programming-our-future-c3bfe30f2d74

[155] D. Mark, V. Zamfir, and E. G. Sirer, “A Call for a Temporary Moratorium on The DAO,” *Hacking, Distributed*, 2016.

[156] R. Böhme, N. Christin, B. Edelman, and T. Moore, “Bitcoin: Economics, Technology, and Governance,” *Journal of Economic Perspectives*, vol. 29, no. 2, pp. 213–238, 2015.

[157] K. Finley, “A $50 Million Hack Just Showed That the DAO Was All Too Human,” 2016. [Online]. Available: https://www.wired.com/2016/06/50-million-hack-just-showed-dao-human/

[158] Financial Services Information Sharing & Analysis, “Operating Rules,” Tech. Rep., 2016. [Online]. Available: https://www.fsisac.com/sites/default/files/FS-ISAC_OperatingRules_June2016.pdf

[159] Department of Homeland Security Integrated Task Force, “Executive Order 13636: Improving Critical Infrastructure Cybersecurity, Incentives Study Analytic Report,” 2013.

[160] J. M. de Fuentes, L. González-Manzano, J. Tapiador, and P. Peris-Lopez, “PRACIS: Privacy-preserving and Aggregatable Cybersecurity Information Sharing,” *Computers and Security*, vol. 69, pp. 127–141, August 2017.

[161] D. Andriesse, A. Slowinska, and H. Bos, “Compiler-agnostic function detection in binaries,” in *Security and Privacy.* IEEE, 2017, pp. 177–189.

[162] Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel, and G. Vigna, “SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis,” in *IEEE Symposium on Security and Privacy*, 2016.

[163] J. Salwan, “ROPGadget,” 2011. [Online]. Available: http://github.com/JonathanSalwan/ROPgadget

[164] D. Stevens, “PDF Tools,” 2008. [Online]. Available: https://blog.didierstevens.com/programs/pdf-tools

[165] The MITRE Corporation, “PEInfo Service.” [Online]. Available: https://github.com/crits/crits_services/tree/master/

[166] K. Kendall and C. McMillian, “Practical Malware Analysis,” in *Black Hat Conferences, USA*, 2007.

[167] K. Chiang and L. Lloyd, “A Case Study of the Rustock Rootkit and Spam Bot,” in *Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, 2007, p. 10.

[168] G. Jacob, P. M. Comparetti, M. Neugschwandtner, C. Kruegel, and G. Vigna, "A Static, Packer-Agnostic Filter to Detect Similar Malware Samples," in *Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2012.

[169] Mandiant, "Classifying Malware using Import API and Fuzzy Hashing – impfuzzy –," January 2014. [Online]. Available: https://www.mandiant.com/blog/tracking-malware-import-hashing/

[170] R. Lyda and J. Hamrock, "Using Entropy Analysis to Find Encrypted and Packed Malware," *IEEE Security and Privacy*, vol. 5, no. 2, pp. 40–45, 2007.

[171] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang, "On the Analysis of the Zeus Botnet Crimeware Toolkit," in *2010 8th International Conference on Privacy, Security and Trust*, 2010, pp. 31–38.

[172] Y. Wei, Z. Zheng, and N. Ansari, "Revealing Packed Malware," *IEEE Security and Privacy*, vol. 6, no. 5, pp. 65–69, 2008.

[173] Microsoft, "Common Object File Format - KB121460," pp. 1–15, 2009. [Online]. Available: https://support.microsoft.com/en-us/kb/121460

[174] ——, "Microsoft Portable Executable and Common Object File Format Specification," p. 97, 2010.

[175] Lifewire, "Things They Didn't Tell You About MS Link and the PE Header," no. 29A, 2004.

[176] T. Stephen, "Rich Header," January 2008. [Online]. Available: http://trendystephen.blogspot.de/2008/01/rich-header.html

[177] D. Pistelli, "Microsoft's Rich Signature," 2010.

[178] M. Ligh, S. Adair, B. Hartstein, and M. Richard, *Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code.* Wiley Publishing, 2010.

[179] OpenRCE, "Microsoft's Rich Signature (Undocumented) - Comments," March 2012.

[180] J.-M. Roberts, "Virus Share," 2011. [Online]. Available: https://virusshare.com/

[181] V. Zakorzhevsky, "Mediyes - The Dropper With a Valid Signature," March 2012. [Online]. Available: https://securelist.com/mediyes-the-dropper-with-a-valid-signature-8/32397/

[182] Mandiant, "APT1 Exposing One of China's Cyber Espionage Units," *Report*, vol. 27, no. 4, pp. 1–76, 2013. [Online]. Available: http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf

[183] Oreans Technologies, "Themida - Advanced Windows Software Protection System," January 2016. [Online]. Available: http://www.oreans.com/themida.php

[184] S. Sarméjeanne, "The HTran Tool Used to Hack Into French Companies," August 2011.

[185] R. Perdisci, A. Lanzi, and W. Lee, "Classification of Packed Executables for Accurate Computer Virus Detection," *Pattern Recognition Letters*, vol. 29, no. 14, pp. 1941–1946, 2008.

[186] L. Martignoni, "OmniUnpack: Fast, Generic, and Safe Unpacking of Malware," in *Annual Computer Security Applications Conference*, 2007.