CYBERSECURITY INCIDENT RESPONSE ORCHESTRATION USING AGILE COGNITIVE ASSISTANTS

by

Steven W. Meckl

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

_____ Dr. Gheorghe Tecuci, Dissertation Director

_____ Dr. Mihai Boicu, Committee Member

_____ Dr. Xinyuan Wang, Committee Member

_____ Dr. Sanjeev Setia, Committee Member

_____ Dr. Huzefa Rangwala, Department Chair

_____ Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date:_____ Fall Semester 2019
George Mason University
Fairfax, VA

Cybersecurity Incident Response Orchestration Using Agile Cognitive Assistants

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

by

Steven W. Meckl
Master of Science
George Mason University, 2010
Bachelor of Science
University of Michigan, 1998

Director: Gheorghe Tecuci, Professor
Department of Computer Science

Fall Semester 2019
George Mason University
Fairfax, VA

DEDICATION

This is dedicated to my father, Roger Meckl, whose clever sense of humor and seemingly endless well of knowledge inspired in me a lifelong desire to learn.

ACKNOWLEDGEMENTS

This research was sponsored by the Air Force Research Laboratory (AFRL) under contract number FA8750-17-C- 0002, and by George Mason University. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS

Advanced Persistent Threat ..... APT
Application Programming Interface ..... API
Cognitive Agents for APT Detection ..... CAAPT
Cybersecurity Operations Center ..... CSOC
Decision Support System ..... DSS
Denial of Service ..... DoS
Distributed Denial of Service ..... DDoS
Domain Name System ..... DNS
Dynamic-Linked Library ..... DLL
Endpoint Detection and Response ..... EDR
Google Rapid Response ..... GRR
Host Intrusion Detection System ..... HIDS
Hypertext Transport Protocol ..... HTTP
Incident Response ..... IR
Indicator of Compromise ..... IOC
Internet Protocol ..... IP
Intrusion Detection System ..... IDS
JavaScript Object Notation ..... JSON
Machine Learning ..... ML
Managed Detection and Response ..... MDR
Megabits Per Second ..... Mbps
Network Intrusion Detection System ..... NIDS
Operational Technology ..... OT
People's Republic of China ..... PRC
Personally-Identifiable Information ..... PII
Receiver Operating Characteristic ..... ROC
Representational State Transfer ..... REST
Root to Local R2L
Secure Sockets Layer ..... SSL
Security Information and Event Management System ..... SIEM
Self-Organizing Map ..... SOM
Subject Matter Expert ..... SME
Threat Intelligence Platform ..... TIP
Time To Live ..... TTL
Tool, Technique, or Procedure ..... TTP
Transport Layer Security ..... TLS
User to Root U2R

ABSTRACT

CYBERSECURITY INCIDENT RESPONSE ORCHESTRATION USING AGILE COGNITIVE ASSISTANTS

Steven W. Meckl, Ph.D

George Mason University, 2019

Dissertation Director: Dr. Gheorghe Tecuci

In this work, I explore the problem of autonomously orchestrating cybersecurity incident response using agile cognitive assistants. Detection of sophisticated cyber threat activity has become more complex over time, as the threat landscape has shifted from cyber vandals and pranksters to multi-billion-dollar criminal enterprises and state-sponsored Advanced Persistent Threats. What was once the realm of criminals with a small collection of easily discovered automated tools is now ruled by well-funded and highly sophisticated sets of hackers carefully orchestrating intrusions as a means to advance their criminal enterprise or intelligence collection mission. This research identifies a new approach to intrusion detection and security incident response aimed at leveraging advances in the field of artificial intelligence to improve the ability of a CSOC to detect these sophisticated attacks. More specifically, it demonstrates how agile cognitive assistants leveraging knowledge-based learning and evidence-based reasoning

can be used to improve the effectiveness of attack detection for both known and unknown threats. Building on the Disciple learning agent theory and technology, I researched, developed, and demonstrated a prototype framework for agile cybersecurity. The key idea is to integrate a special type of a knowledge-based learning assistant into cybersecurity operations centers. This cognitive assistant can be trained by cybersecurity experts, based on threat intelligence, to automate the investigation of alerts from a variety of intrusion detection devices, integrating multiple detection techniques with automated network forensics, to significantly increase the probability of accurately detecting intrusion activity while drastically reducing the workload of the operators of the cybersecurity operations centers. This dissertation presents the following novel contributions: (1) conceptual modeling of the automatic APT detection process; (2) ontology design for APT detection; (3) automatic generation of abductive triggers from basic intrusion detection systems; (4) autonomous, hypothesis-driven search for evidence; (5) selection and integration of multiple, collaborative, search and collection agents working together to detect and investigate threats; and (6) development of Collection Manager software for translating and optimizing abstract searches into searches executable by real collection agents.

1. INTRODUCTION

The field of cybersecurity has rapidly expanded over the last decade, evolving into multiple disciplines including cryptography, intrusion detection, threat analysis, managed detection and response (MDR), security orchestration and automation, and intrusion tolerance. While the state of the art has seen steady progress, the volume and sophistication of attacks have outpaced the rate at which network defenses have evolved (Verizon, 2014). History has shown security to be a constant war of escalation between attackers and defenders, and high-profile intrusions including attacks on Sony Pictures Entertainment (USATODAY, 2015), the coordinated network exploitation campaign of China's People's Liberation Army (Mandiant, 2013), and the alleged attacks on the White House by Russian actors (CNN, 2015) show that attackers are still successful in spite of record high cybersecurity budgets (Cybersecurity Ventures, 2015).

The science and art of intrusion detection and prevention has evolved over the last decade, largely due to the shift from cyber vandals and pranksters to multi-billion-dollar criminal enterprises and state-sponsored Advanced Persistent Threat (APT) intrusion methodology (Zimmerman, 2014). What was once the realm of criminals with a small collection of easily discovered automated tools is now ruled by well-funded and highly sophisticated sets of hackers carefully orchestrating intrusions as a means to advance their criminal enterprise or intelligence collection mission. State-sponsored attack groups

such as the People's Republic of China's (PRC) APT1 have demonstrated that organization, funding, and lack of consequences can be more effective than use of sophisticated intrusion tools (Mandiant, 2013).

Commercial intrusion detection tools have not adapted well to this shift in attacker methodology. Attackers' sophistication has outpaced that of network defenders. This is most apparent in data analysis published in Verizon's 2014 Data Breach Investigations Report (Verizon, 2015). In 2013, the attacker's time to compromise was measured in days over 90% of the time while the defender's time to detection was measured in days less than 25% of the time, leaving large windows of time for attackers to operate undetected. What's worse is that the attackers' time to compromise is shrinking at a faster rate than defenders' time to detection. Technological advances are necessary to shrink the gap between the two.

Well-organized cybersecurity operations centers (CSOCs) leverage analysts with a wide variety of skills to constantly monitor and adjust their security infrastructure to adapt to intrusion methodology changes. Even advanced CSOCs, leveraging state-of-the-art intrusion detection system (IDS) technology, receive too many alerts for their analysts to handle. Investigation of each alert has a cost to the organization, measured in time, man-hours, and infrastructure expenses. The cost of missed detections can range from negligible to catastrophic. While many simple network compromises are routine and easy to overcome, the average cost of a data breach has risen to nearly $4 million in recent years (Ponemon Institute, 2017). False positives are also expensive (Zimmerman, 2014), as network defenders waste time investigating incorrectly identified security

incidents. In large enterprises, that see thousands of network events per day, even a 1% false positive rate can be unmanageable.

This dissertation details a new approach to autonomous orchestration of security incident response aimed at leveraging advances in the field of artificial intelligence and machine learning. More specifically, this approach uses agile cognitive assistants leveraging knowledge-based learning and evidence-based reasoning to improve the effectiveness of a CSOC's ability to detect both known and unknown sophisticated threats. Building on the Disciple agents learning theory and technology (Tecuci et al., 2016a), I researched, developed, and demonstrated a prototype framework for agile cybersecurity. The key idea is to integrate a special type of a knowledge-based learning assistant into CSOCs. This cognitive assistant is trained by cybersecurity experts, based on threat intelligence, to automate the investigation of alerts from a variety of intrusion detection devices, integrating multiple detection techniques with automated network forensics, to significantly increase the probability of accurately detecting intrusion activity while drastically reducing the workload of the operators of the CSOCs. This approach is *agile* because these agents can rapidly learn the subject matter expertise of skilled CSOC analysts to accurately identify both the existence and scope of network intrusion events. They learn general models of threats in the form of explicit reasoning patterns that are used to hypothesize intrusions, to direct the sensors to collect evidence for these hypotheses, and to test the hypothesized intrusions based on the collected evidence.

1.1. Motivation

This section provides more detail of the challenges modern CSOCs face, and why advancements in the field are necessary.

1.1.1. Intrusion Detection Overview

Intrusion detection is typically done by searching through network data, host-based data, or logs, either in real-time or after the fact, and applying detection algorithms to the data to identify intrusion activity. In some cases, multiple types of data can be fused to provide more robust detection capability, although there are not too many capabilities to do data fusion at present.

Intrusion detection algorithms focus on two major approaches: *anomaly detection* and *misuse detection*. Anomaly detection focuses on building a model of what is normal and then using that model to identify behavior that is abnormal. It can work in real-time and is capable of detecting new threats, but often has a high false positive rate. Misuse detection uses information about attacks, often in the form of static signatures or Indicators of compromise (IOCs), to detect malicious activity. It has a high detection rate for existing threats and a low false positive rate, but normally cannot detect new threats (Kemmerer & Vigna, 2002).

Machine learning, in the form of neural networks, statistical methods, or a combination of learning models, is a major area of anomaly detection research. Bhuyan et al. (2014) provides an excellent overview of the topic. While there have been some improvements in the state of the art, machine learning methods still have not solved two important problems for network defenders:

1. While machine learning methods (e.g., neural networks) perform well, they cannot tell the user why they arrived at a conclusion.
2. They still have unacceptably high false positive rates for large CSOCs.

Misuse detection is a common approach used in production systems today. Misuse detectors are designed to use IOCs to scan network, disk, or volatile memory to detect known tools, techniques, or procedures (TTPs) used by attackers. Common intrusion detection system (IDS) software packages, such as SNORT (Northcutt et al., 2007) and BRO (Paxson, 1999), along with traditional antivirus programs, are misuse detectors. Most desktop anti-virus programs also fall into this category. They are popular because they are easy to implement and share indicators for. The general weakness of misuse detection is the inability to detect new threats, although they have a high detection rate for known threats (Kemmerer & Vigna, 2002).

1.1.2. Threat-Driven Analysis and the Cybersecurity Operations Center

CSOCs employ teams of network defense experts, analysts, system administrators, and forensics experts. CSOCs leverage a rich tool set including intrusion detection systems (IDSs), data collection tools, analysis tools, and visualization tools. CSOCs receive incident information from high-value sources – law enforcement, user reporting, or threat intelligence from other CSOCs – and unconfirmed alerts from security infrastructure such as antivirus software, IDSs, heuristic alerts, or machine learning algorithms. The analyst's responsibility is to monitor alerts and logging information from all available information sources, each having differing levels of

reliability, and use them to make a determination about the presence or absence of intrusion activity (Zimmerman, 2014).

TTPs used by attackers change rapidly and can be discovered on almost any network. In an effort to adapt, there has been a movement toward making the sharing of indicators between CSOCs fast and easy. IOCs can now be shared among communities of interest using multiple open formats and transports including OpenIOC (2015), created by Mandiant, and the STIX/TAXII architecture (Barnum, 2014; Conolly et al., 2012), created by The MITRE Corporation. Additionally, security companies have begun publishing threat intelligence to customers and the public, such as the now famous Mandiant report on the APT1 intrusion set (Mandiant, 2013), which attributed an aggressive group of computer intrusion operators to China's People's Liberation Army. While these initiatives can be effective in sharing signatures and other threat artifacts, each receiving organization still must implement their own process to integrate the information into their security infrastructure, using the myriad anomaly and misuse detection systems on the market.

1.1.3. Advanced Persistent Threats

The most sophisticated type of attacker group is called the Advanced Persistent Threat. APTs are characterized by the use of superior knowledge, resources, or tactics to gain and maintain access to the networks of their intended victims to accomplish missions ranging from information theft (Mandiant, 2013) to destruction of data (Pagliery, 2015). Many APT groups are known or suspected to be large state-sponsored

teams of highly trained and well-funded hackers, following a structured attack process called the Attacker Lifecycle (Mandiant, 2013).

APT groups have some distinct differences from traditional hackers making them difficult to deal with. Due to the fact that they are often government employees or contractors performing their attacks as a part of their job and have been authorized to conduct such activity by their government, it is low risk/high reward activity. Unlike criminal hackers, there is no risk of incarceration should they get identified. As such, they can afford to optimize for volume over stealth, meaning they can attack a lot of potential victims simultaneously, exhausting defenders' resources. They are also often acting to fulfill the intelligence requirements of their government. They are persistent in pursuing that mission, frequently returning to previously compromised networks with different apparent TTPs. Many organizations are not prepared for this level of persistence.

The structured nature of APT activity also provides some opportunities defenders can exploit. Since they are often large organizations, they are subject to the limitations of large information technology teams, namely the slow evolution of their malware tools and the predictability of their structured attacker lifecycle.

I use the APT attack model as a model for study in this research because prior research into APT attackers has yielded important insights regarding the structured approach many sophisticated attackers use, including modern criminal enterprises and even some "lone wolf" attackers. While this research was built around the theoretical

model of APT detection, the developed approach is applicable to the autonomous orchestration of security incident response to any modern cyber attacker profile.

1.1.4. Evolution of Malware

One of the main reasons that misuse detection fails is the reliance on matching static IOCs in malware files, on disk, or in volatile memory. It is trivial for attackers to change their malware to avoid signature-based detectors and it is done in many ways, including the following:

- **Packers/Obfuscators** – Code packers or obfuscators are tools that can compress, encrypt, or otherwise change the look of a compiled executable program file. At run-time, the code is de-obfuscated in memory prior to execution.
- **Reconfiguration** – Attackers often change the way artifacts of their malware look when executed during an intrusion to evade static signature detection. For example, the attacker may change the name of any domains, Internet Protocol (IP) addresses, Registry keys, or file names used for each attack, but the malware otherwise behaves the same.
- **Evolutionary Development** – Malware is written using the same techniques as legitimate software systems. The tools evolve over time as malware developers learn new techniques. Each incremental development step helps evade static signatures developed to detect previous versions.

These phenomena have been observed many times over a long period of time (Gupta et al., 2009). In their report on the APT1 intrusion set Mandiant (2013) describes

the set of malware programs used by the group of attackers over several years. It also contains lists of domains, digital certificates, IP addresses, and accounts used by the group. APT1 would use the same malware for multiple attacks, reconfiguring the programs to communicate with different domains, register themselves as Windows services with different display names, and use different file and process names. Heavy use of *reconfiguration* is one of the factors allowing the group to be successful for so long.

APT1, among other attacker groups, also practices *evolutionary development* to adapt to changes in network defense technology or simply to increase efficiency. Further analysis of APT1 by Mila (2013) at contagiodump.blogspot.com shows a timeline of the attacker group's tool usage from 2004 to 2012, including information on dozens of samples of malware. The group evolved their tool set slowly over the course of at least eight years.

These changes in the way malware presents itself on the network and on host computers made it difficult for signature-based intrusion detection tools to detect attacks because the attacks can change static information in their malware faster than defenders can adapt. However, the patterns of behavior change more slowly and with less variance.

Analysis of the IOCs published by Mandiant (2013) shows that APT1 malware demonstrates clusters of behavior. Subsets of the programs share sets of techniques for communicating on the network, persisting through a reboot, or storing data on disk. One example of this is the cluster of malware programs comprised of AURIGA, BANGAT, SEASALT, and KURTON. While the features of the software changed fairly drastically

over time, the main differences in forensic evidence generated by those four tools is the persistence mechanism used to survive a reboot, the strings used to register as a service or device driver, and the names of files and Registry keys.

Clusters of malware programs are often called *malware families* in published research. Each member of the family incrementally builds on previous versions as they get detected and become less effective. The Sobig virus (Stewart 2003a; 2003b) is an example of a malware family. It was used in 2003 in a widespread email phishing attack. The Sobig virus evolved over five different revisions. Over the course of these revisions, the author changed how the malware set its expiration timer, where the command-and-control servers were located, and how encryption was used to improve the effectiveness of the malware.

Security analysts and network defenders can take advantage of the slow evolution of behavior in malware families to be more effective in detecting malware and implementing new security controls. This uses the same methods I employed to train a knowledge-based learning agent to improve the ability of a CSOC to detect attacks.

1.2. Problem Statement

The fast evolution of malware, increasing number of security events, and relatively high false positive rates have become problematic for CSOC analysts and network defenders. This combination of factors has resulted in increases in the number of successful attacks, the number of attacks that go undetected, and an overwhelming number of security events for CSOC analysts to investigate. Dealing with this complex security environment has become very expensive for network defense organizations.

The evolution of malware and network intrusion TTPs has allowed attackers to continue to evade security controls. Despite the tens of billions of dollars spent on security every year, the number of successful attacks has increased steadily year over year. According to Symantec's 2019 Internet Security Threat Report (O'Gorman et al., 2019), both the volume and sophistication of attacks continues to increase. To make matters worse, the time required for attackers to compromise a network has gone down while the time required for defenders to detect the intrusion has gone up. This means that attackers have a larger window of time to accomplish their goals before detection.

There are several reasons why intrusion detection systems and CSOCs have remained unsuccessful against evolving malware and TTPs. Most security systems deployed today are simple rule-based misuse detection systems that rely on a current set of IOCs to be effective. As TTPs change, the set of IOCs cannot be maintained fast enough to keep up. Development of IOCs is a complicated process, first requiring detection of an intrusion, then detailed analysis of the malware and attacker TTPs, followed by careful development of signatures that can distinguish between legitimate and malicious traffic. This is typically a time-consuming and manual process, requiring a highly skilled analyst. Analysis of the rising number of new attacks has caused a resource strain on network defense organizations. In a large enterprise thousands of alerts can be reported daily, and most organizations report they are able to investigate less than 50 in a typical work week (Ponemon Institute, 2017).

The large and increasing number of alerts and the time required for their manual analysis creates a very complex, expensive, and non-sustainable security environment for

network defense organizations where most alerts are not investigated, increasing the risk to the enterprise.

There has been a lot of promising research into classifier-based intrusion detection systems (Bhuyan et al., 2014). Network anomaly detection systems based on neural networks, in particular, have vastly improved the detection rate of new attacks. However, neural network-based systems still have unacceptably high false positive rates. They are also limited in their expressiveness. While they can distinguish between legitimate and malicious traffic, they are unable to articulate how they arrived at a decision. The inability to do so leaves the analyst with more investigative work.

In order to improve the effectiveness of intrusion detection systems and reduce the cost of investigating security incidents, CSOCs require improvements in some key areas:

- Accuracy of a CSOC's detection of attacks, improving both detection rates for known and new attacks, and reducing false positive rates;
- Enhancement of CSOC processes to increase the number of security events that can be investigated per day;
- Rapid development and agility in automation and orchestration in tier 1 incident triage and tier 2 incident response investigations;
- Expressive intrusion detection systems that don't just detect suspected malicious activity but provide detailed information to aid in the investigation.

1.3. Purpose of Research

The purpose of this research was to determine if a knowledge-based learning assistant and evidence-based reasoning could be used to increase the effectiveness of attack detection and improve the efficiency of CSOC operations through orchestration and automation of security incident response. During this research a prototype framework for a knowledge-based learning assistant was developed that can respond to security alerts, use abductive, inductive, and deductive reasoning to hypothesize intrusions, collect evidence about these hypotheses, analyze them based on the collected evidence, and present the results to a CSOC analyst to act upon.

The system leveraged a scalable learning assistant platform to process many simultaneous alert investigations and interface, via a flexible JavaScript Object Notation (JSON) API, with a mix of commercial, open source, and custom collection and analysis agents to evaluate the elementary hypotheses. Results of automated analysis were used to synthesize conclusions regarding the presence of intrusion activity and the scope of the intrusion. The prototype platform provides CSOC analysts with a flexible environment to model their expertise in natural and flexible way that can be applied directly to the enterprise network environment to detect and respond to intrusion activity.

The system addresses critical gaps in the current capabilities and workflow of modern CSOCs. Leveraging a knowledge-based learning assistant to process security alerts both reduces the error rate resulting from human operators performing repetitive investigations and enables CSOCs to scale the system to process every alert, which is currently infeasible. The system also adds value to existing security infrastructure and

threat intelligence services by tying together multiple, diverse, low-confidence security sensors to provide high-confidence detection using the expertise of security experts.

The prototype system was used to model the methodology a skilled CSOC analyst would use to investigate a set of known APT malware, specifically the APT1 malware published by Mandiant (2013). Experiments were conducted to show the system can be trained to both detect known threats and predict future attacks by the same APT group, as well outperform similar activity conducted in modern CSOCs. Results of these experiments are included in this dissertation.

1.4. Research Questions

This research answers several questions regarding the ability of knowledge-based learning assistants and evidence-based reasoning to accurately model and detect APT intrusions and increase the accuracy and efficiency of cybersecurity operations centers. These questions are listed below, grouped into *intrusion detection improvement, analyst support*, and *performance* categories.

1.4.1. Intrusion Detection Improvements

A core goal of this study was to make improvements in the theoretical model used to detect computer intrusions, specifically those performed by APT groups. Much of the effort in this research involved creation of a mathematical model, rooted in computational theory, to accurately detect APT activity through active network monitoring and automated computer forensics. A primary objective of this work was to determine whether a knowledge-based learning assistant, using evidence-based reasoning is able to:

- effectively fuse evidence from multiple data sources, including host-based information, network information, and evidence collected from external sources such as domain registrars, IP address registrars, malware analysis services, and threat intelligence streams;
- use explicit logic to aggregate weak intrusion indicators into strong ones, showing clearly all the reasoning steps and the evidence used.

1.4.2. Analyst Support

The functional questions regarding this study concern agent training with subject matter expertise, ease of knowledge base development, understandability of analysis, and ease of use by analysts. The study determined that an agent can:

- use mixed-initiative reasoning to assist CSOC analysts in identifying new attacker TTPs;
- be trained to automatically detect threats based on CSOC analyst expertise and threat intelligence, such as the Mandiant report on the APT1 intrusion set;
- support the analyst in identifying previously unknown threats, and learn to automatically detect future intrusion attempts of the newly discovered type;
- exhibit flexible autonomy in its interactions with the CSOC operators, from being strictly guided by the operators, to mixed-initiative, and to full autonomy.

1.4.3. Performance

Wherever possible, the attributes of the system were measured through experimental testing and calculated from observed measurements. In terms of measured

performance, this study determined that knowledge-based reasoning with evidence-based argumentation can:

- improve the detection and false positive rates of intrusion detection systems;
- improve the efficiency of CSOCs by using knowledge of cyber threats to automate repetitive investigative tasks on behalf of CSOC analysts;
- scale efficiently, meaning the number of alerts analyzed scales linearly with respect to processing power and storage.

1.5. Research Contributions

This section provides an overview of the theoretical and architectural contributions produced by this research.

1.5.1. Theoretical Contributions

The theoretical contributions achieved center around development of a theoretical model allowing for the creation of autonomous agents to detect APTs through learned security orchestration and automation. More specifically, the theoretical contributions from this research are:

1. Conceptual modeling of the automatic APT detection process;
2. Ontology design for APT detection;
3. Automatic generation of abductive triggers from basic IDSs (e.g., BRO);
4. Autonomous, hypothesis-driven search for evidence.

1.5.2. Architectural Contributions

Development of a system of agents allowing integration into a wide variety of CSOC environments in order to collect and evaluate digital artifacts in an autonomous

fashion required a substantial amount of design and development. To this aim, the following architecture contributions were achieved:

5. Selection and integration of multiple, collaborative, search and collection agents working together to detect and investigate threats;
6. Development of the Collection Manager software for translating and optimizing abstract searches into searches executable by real collection agents.

1.6. Importance of the Study

Large corporations, particularly those in the retail, banking, and energy sectors, as well as government agencies have a vested interested in making their networks more secure. As discussed above, numerous resources are expended by modern CSOCs to manually gather and analyze data in response to both true positive and false positive security alerts. In both cases, the investigative and analytical work is repetitive because the breadth and scope of responses are finite. Automating much of that repetitive work with a knowledge-based learning assistant and evidence-based reasoning will free CSOC analysts to pursue other important security tasks such as intelligence analysis, threat hunting, and improving network security posture.

The Center for Strategic International Studies estimates that the net annual net losses to the global economy due to cybercrime are more than $400 billion (Center for Strategic International Studies, 2014). *Cybersecurity Ventures* estimates that cybersecurity will be a $155 billion market by 2019 (Cybersecurity Ventures, 2015). As

security budgets rise, an effective and efficient CSOC will be a competitive advantage. Resources saved on security can be used to fund the profit centers of the organization.

This research successfully determined that much of the repetitive and analytical work can be automated by a knowledge-based learning assistant, which can learn from observing how skilled CSOC analysts respond to security events, increasing the efficiency of CSOCs, improving detection and false positive rates, and enabling CSOC analysts to pursue more creative and rewarding tasks.

1.7. Scope of the Study

Intrusion detection is a very large topic. With the emergence of mobile platforms, embedded systems, and operational technology (OT) systems – including manufacturing, building controls, and other industrial control systems – there are dozens of potential platforms, techniques, and attack surfaces that can be explored. The threat space is also very large. Attacker motivation and skill varies wildly and there are hundreds of thousands of malware variants discovered in the wild. It would be unreasonable to attempt to build a system to cover all intrusion detection problems. In order to produce usable research results in a reasonable amount of time, the scope of the project must be limited. This section describes the scope of the research project and the reasoning behind its scope limitations.

1.7.1. Knowledge-Based Learning Agent Shell

Ideally, a system that learns from demonstration by a subject matter expert (SME) would be able to observe the same intrusion detection events as the SME, watch everything they do in response to the event, analyze it to determine the effectiveness of

each action and learn from each demonstrated intrusion investigation to develop applicable rules and patterns. Unfortunately, such a system would have required development of a substantial number of auxiliary technologies that do not directly support the stated goals of this research.

Instead, the research was conducted using a more controlled environment: the *Disciple* knowledge-based learning agent shell. *Disciple* is a tool developed and supported by the Learning Agent Center at George Mason University's Volgenau School of Engineering. It is a Java-based learning agent platform that facilitates creation of purpose-built knowledge bases and learning agents covering a wide variety of subject matter areas. Disciple allows subject matter expertise to be modeled in the form of easily understandable reasoning trees, using natural language description. Problems can be generalized by utilizing well-defined ontologies. Disciple's ability to elegantly handle the difficult aspects of creating a knowledge-based learning agent with evidence-based reasoning enabled a focus on modeling the intrusion detection and investigative processes and developing the required data stores and collection agents to support this research.

1.7.2. Operating System Environment

The software systems used in this research were developed for Microsoft Windows operating systems. Due to sustained market dominance, Windows computers are the primary target of computer intrusions and the vast majority of malware is developed to target Windows (Symantec, 2015). To have the largest potential impact, it made sense to develop a prototype system for Windows.

1.7.3. Attacker Profile

This research project focused on detecting APT type intrusions into enterprise networks. APT is loosely defined as sets of well-funded and well-organized actors who gain and maintain access to sensitive computer networks for long periods of time, exfiltrating intellectual property, personally identifying information (PII), and other sensitive records to fulfill their organization's mission. APT was chosen as the target attacker profile because it has become a serious problem for network defenders in the last several years. As such, there was ample finished analysis of APT TTPs to facilitate development of a knowledge-based learning assistant capable of detecting APT activity.

1.8. Limitations

While this research involved a flexible solution applicable to a wide variety of investigative and analytical problems in intrusion detection and security incident response orchestration, there were limitations to the research. This section discusses identified limitations of the research, alternative approaches, and steps taken to mitigate those limitations.

Ontology-based systems carry an authorship burden, as the system can only reason about the concepts and facts it is aware of. Creation of a large, comprehensive ontology can take an extensive amount of time. This challenge was overcome in two ways. First, there is a fair amount of research published research into security taxonomies and ontologies. These were used where applicable. Second, because the focus of this research was to create an agent capable of intelligently responding to security alerts by conducting autonomous intrusion investigations, the ontology was built

incrementally as concepts required by the cognitive agent were discovered via research into APT malware and TTPs. This limited the scope of the ontology and relieved the authorship burden.

Intrusion detection requires analysis of multiple types of evidence, including captured network traffic, network flow data, volatile memory, Registry keys, executable binary files, data files, logs, and other non-volatile hard disk artifacts in a wide variety of formats. While it is possible to automate much of this analysis given sufficient time and resources, it is infeasible to automate all possible types of analysis. An example of this is live memory capture. Available tools for capturing volatile memory are often expensive, error prone, and difficult to use over the network (Ligh et al., 2014). For this research project, the priority was to automate analysis of data types applicable to the most possible use cases and the largest set of malware in the training and test data sets.

There are no standard data sets for testing hybrid network-based and host-based intrusion detection systems. Most network-based intrusion detection systems use the DARPA 99 data set (DARPA, 1999) or one derived from it. Host-based systems and hybrid systems such as DIDS (Snapp et al., 1991) have yet to settle on a standardized dataset for testing. As such, a new data set was created for this research that includes APT malware samples, published analytical research, and IOCs. Since the training and test data sets for machine learning are a critical part of algorithm development, the research was limited by the assembled data set.

Subject-matter expertise was also a limiting factor for this research. System's performance is heavily dependent on the quality of expertise the system is modeled after.

While I was able to develop a knowledge-based learning assistant that uses evidence-based reasoning to accurately model one SME's analytical approach, there likely exist analysts with different knowledge and experience, and therefore other effective ways of investigating, analyzing, and detecting intrusions. The intent of this research was not to create the best possible agent, but to demonstrate that the knowledge-based approach is effective for modeling and automating the investigative and analytical tasks of CSOC analysts.

Lastly, issues regarding secure collection of data from compromised computers was outside the scope of this research. Sophisticated attackers routinely use techniques to hide the presence of their malware by overriding system calls, either by hooking exported functions in user-mode dynamic-linked libraries (DLLs) or by hooking calls in various system call tables in the operating system kernel (Hogland & Butler, 2005). The problems related to collecting accurate data from an operating system compromised with one of these techniques is well-known and, in some cases, commercial products exist to help reveal these techniques or allow secure data collection. Wherever possible during this research project, best practices were followed regarding collecting evidence from workstations. However, novel solutions to this problem are outside of the scope of this research.

2. RELATED WORK

Intrusion detection is a broad, multi-disciplined area of computer security, the complexity of which is limited only by the knowledge, skill, and creativity of both attackers and defenders. Attackers have learned to exploit almost every aspect of operating systems, leverage exploits in network protocols, and take advantage of weaknesses in human operators. Likewise, successful defenders must become experts on operating systems, networks, forensics tools, scripting languages, and the specific TTPs used by attackers as they evolve their craft. Due to the speed at which attacks happen, most of them are discovered after the fact by incident responders and forensic analysts (Verizon, 2014). As such, knowledge about intrusions is accumulated relatively slowly. As repetitive tasks are identified or new technologies emerge, analysts and developers create tools to automate the detection of known threats, increasing over time the security posture of computer systems.

This chapter provides a review of academic research to understand the methods used in the past to detect intrusions into computer networks by external attackers and the strengths and weakness of those approaches. By examining those topics, research gaps were identified which were addressed by this research. These topics will be addressed through a thorough review of the state-of-the-practice and state-of-the-art in automated intrusion detection. It will describe the major sub-topics in intrusion detection, the types of data that must be examined to detect intrusions, architectural trade-offs in IDS design, and datasets used to evaluate intrusion detection systems. A brief overview of machine

learning (ML) techniques will then be provided, along with recent developments in applying ML to various intrusion detection problems.

2.1. Cybersecurity Operations Centers

In organizations with large networks, the job of computer network defense (CND) has been consolidated within a CSOC, which contains the tools, expertise, network visibility, and authority to monitor, detect, and react to security events (Zimmerman, 2014). While they can vary in size, scope, and specific construction, they generally have a workflow similar to Figure 1.

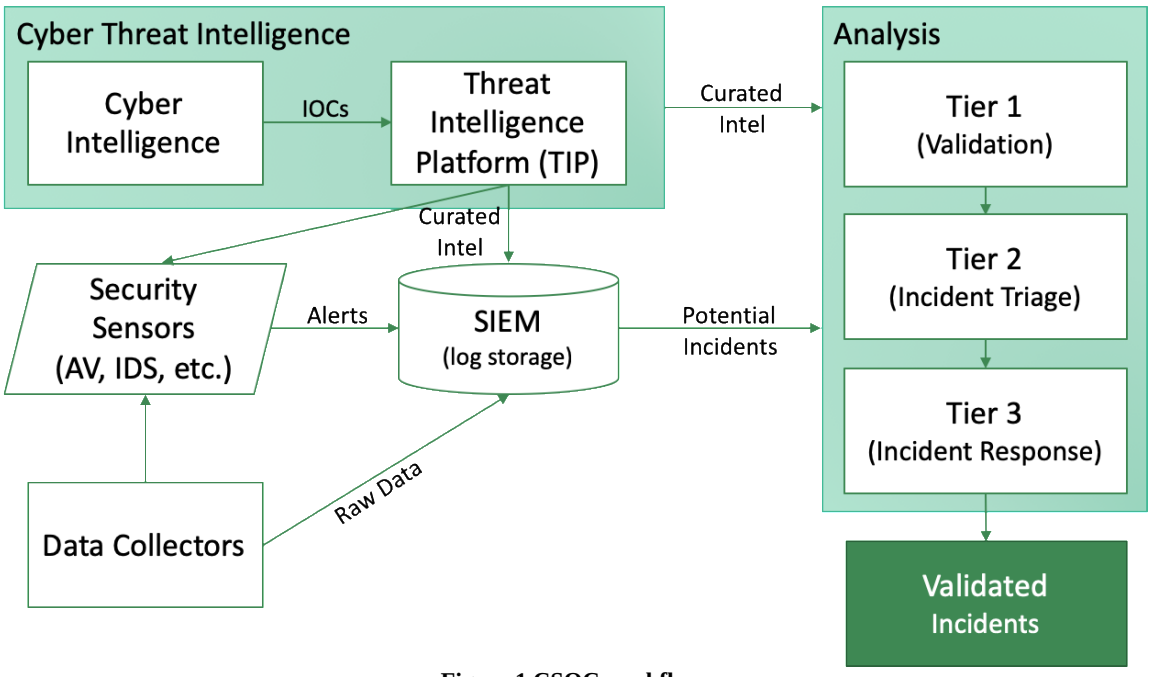

Figure 1 CSOC workflow

The Cyber Threat Intelligence component is responsible for understanding the network environment and the threats to it. Its responsibilities include actively monitoring threat intelligence reports to understand trends in attack methodology. Intelligence reports are published as subscription feeds by security companies including FireEye

(2015), Symantec (2015), and iSIGHT Partners (2015). By monitoring, analyzing, and fusing information in these reports with information from the internal network, the CSOC intelligence component can effectively deploy and tune security infrastructure.

In sophisticated CSOCs, the Cyber Threat Intelligence component operationalizes threat intelligence using a Threat Intelligence Platform (TIP). Intelligence analysts examine incoming intelligence for relevance and accuracy and deploy curated sets of IOCs to sensors and the security information and event management system (SIEM) for use in threat detection. When security sensors match an IOC against collected data, a security alert is sent to the SIEM for correlation.

At the center of a CSOC is its SIEM, which is responsible for storing logs and network configuration information (Zimmerman, 2014). The SIEM is the central tool used by analysts for organizing, tracking, and analyzing the information needed to identify and respond to attacks. The databases can be in the form of object storage, relational database, or flat storage, although a recent trend is to use unstructured storage databases like Splunk (2015) and Elasticsearch (2015). Analysts use the SIEM for correlation analysis, combining multiple alerts and raw logs to generate *potential incidents* that must be further examined.

Tier 1 analysts are responsible for validating *potential incidents* using the historical data stored in the SIEM, along with open source information and external data sources. The main objective of Tier 1 analysis is to quickly determine if a *potential incident* is a false positive or a true detection requiring triage analysis by a Tier 2 analyst.

Tier 2 analysts are more experienced in analysis and perform *incident triage* to quickly answer a few key questions about a *validated incident*:

- Can they confirm the analysis performed by the Tier 1 analyst?
- If it is a real security incident, what is the scope of the attack?
- Is it possible to quickly recover from the attack?

By answering these and other questions, the Tier 2 analyst will determine if they can remediate the attack or whether Tier 3 incident response is required.

Tier 3 analysis is often called Incident Response. Tier 3 analysts have a high level of expertise in incident investigations and are responsible for conducting in-depth analysis and forensics when an incident happens. They perform costly and time-consuming analysis of network data, logs, disk images, malware, and copies of volatile memory. This analysis is done with a wide variety of commercial and open source tools. The results of this analysis determine the response from the organization, law enforcement, or outside parties.

While the CSOC infrastructure provides a robust set of tools for data analysis, it still has limitations. Most of the analytical work is done manually or automated with simple scripts. In a large enterprise, thousands of alerts can be reported daily, and most organizations report they are able to investigate less than 50 in a typical work week (Ponemon Institute, 2017). Therefore, even sensors with a false positive rate of one percent may have enough missed detections and false positives to be unmanageable by even mature CSOCs. Increasing the efficiency and accuracy of Tier 1 and threat

intelligence tasks could drastically lower the cost of running a CSOC and reduce risk to the organization.

2.2. Overview of Intrusion Detection Concepts

An intrusion attempt is a deliberate attempt to gain unauthorized access to a computer system or network to compromise its *confidentiality*, *availability*, or *integrity* (Sabahi & Movaghar, 2008). The *confidentiality* is the property of a system which ensures information be accessed only by those persons or entities explicitly authorized to do so. A compromise of this property often manifests itself in the form of intellectual property, financial information, or PII being stolen from a computer network. Denial of service (DoS) and distributed denial of service (DDoS) attacks impact the *availability* property of a system, consuming more resources than a system has or exploiting a defect to make the system unavailable to its intended users. *Integrity* is the property of a system assuring that data stored on it remain in the state it was intended. An integrity compromise might involve changing key words in a contract to change its meaning or modifying bank account records to facilitate theft (Zevin, 2009).

In general, there are two venues for intrusion detection, defined by where an analyst or automated tool must look to identify the intrusion. *Host-based intrusion detection systems* (HIDS) attempt to detect unauthorized access by examining programs running on computer or searching for artifacts in memory or non-volatile storage. Desktop anti-virus programs are the most commonly used HIDS. They use static signatures, heuristics, and machine learning to identify attack tools as they are written to and read from hard disks. *Network-based intrusion detection systems* (NIDS) detect

intrusions by examining network data in real-time or stored in a database. The goal of a NIDS is to detect intrusion attempts, in near-real-time or real-time, before the attackers successfully compromise hosts. Rare, but not uncommon, are hybrid intrusion detection systems that fuse both network-based and host-based information to more effectively detect intrusion activity.

Intrusion detection systems are typically evaluated using three metrics: *detection rate*, *false positive rate*, and the *receiver operating characteristic* (ROC) curve (McHugh, 2000).

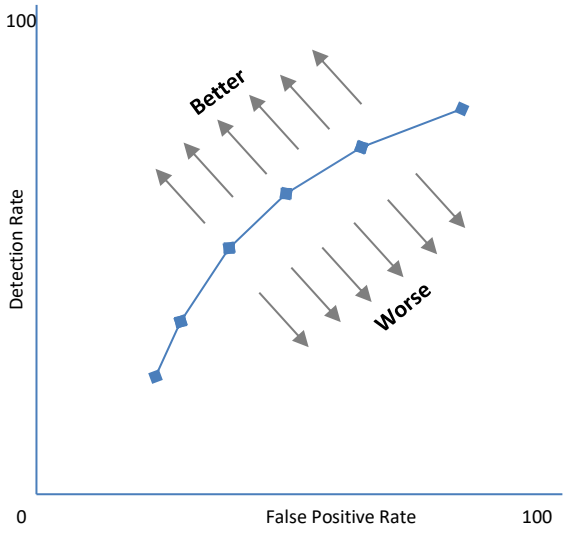

Figure 2 Receiver operating characteristic curve

The detection rate is the percentage of real attacks that are detected by the system. The false positive rate is the percentage of events incorrectly flagged as attacks. The ROC curve is a plot of detection rate versus false positive rate. Figure 2 shows an

example of what a ROC curve looks like. It can be difficult to quantitatively compare ROC curves. Instead, they are often visually compared, with the curve of superior performing systems lying to the upper left of the curve of an inferior system.

2.2.1. Cost Management in the CSOC

A major factor of CSOC operations is the tradeoff between security value and cost. The costs of operations can be measured in terms of money required to purchase security infrastructure and hire staff, as well as the time required to implement and maintain security controls and threat detection. Since no security system can be perfect at defending against attacks, CSOC managers must decide how many resources to apply based on the value of the assets being protected and the overall assessed risk to those assets.

To manage costs effectively, a CSOC must effectively balance use of automated and manual analysis. Automated analysis, typically in the form of host-based or network-based threat detection systems, is used to quickly detect known threats or identify anomalies for further investigation by human analysts. As discussed above, automated systems are measured by the ROC curve. Automated systems are employed as much as possible because they scale well and, compared to human analysts, are low in cost. In a sense, automated systems are designed to cover the area under the ROC curve, as shown in Figure 3.

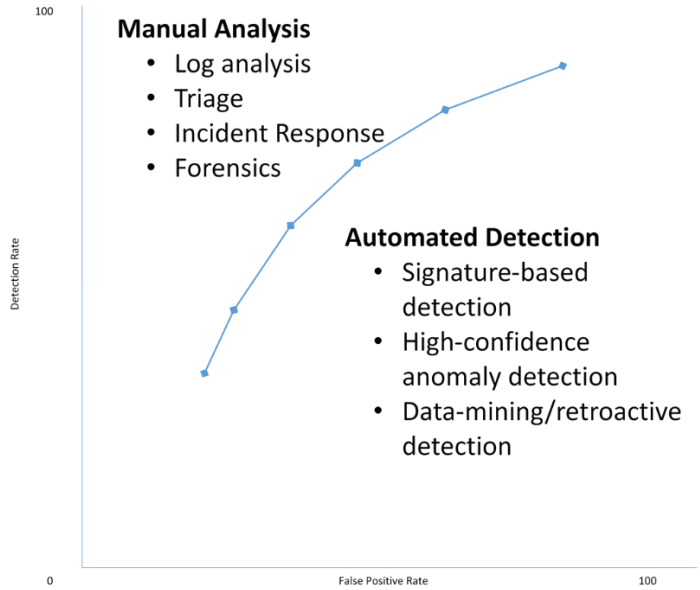

Figure 3 Cost management using the ROC curve

To address gaps in the detection and false positive rates of the CSOC security infrastructure, manual analysis is employed. Effectively, manual analysis is responsible for the area above the ROC curve. Human analysts follow up on alerts generated by automated analysis to verify their correctness prior to taking action. They are also sometimes employed at "threat hunting," where human analysts leverage their experience and knowledge of threats to manually search through all available data to look for attacks missed by automated analysis. Manual analysis can reduce the overall false positive rate of the CSOC and find new attacks but is very costly and does not scale well.

A common strategy in managing the cost of CSOC operations is to evaluate the cost of tuning automated analysis infrastructure to maximize detection rates (below the ROC curve) and the cost of manual analysis to address missed detections and false

positives (above the ROC curve). Once those costs are understood, a CSOC manager can make informed decisions in balancing the two halves of their operation to minimize cost.

2.2.2. Misuse Detection Systems

Misuse detection is the most common technique in existing intrusion detection systems. These techniques are designed to use knowledge of a threat, usually encoded in the form of static signatures, to detect known threats. These signatures, often referred to as *IOCs*, come in three types, as described in (Hutchins, et al., 2011):

- **Atomic** – Distinct indicators that cannot be broken down into smaller components
- **Computed** – Indicators derived from threat data, including hashes, regular expressions, and anti-virus signatures
- **Behavioral** – Collections of atomic and computed indicators, often tied together with combinatorial logic

While NADIR (Hochberg et al., 1993) was one of the first automated misuse detection systems, many have been created since. SNORT (Northcutt, et al., 2007) and BRO (Paxson, 1999) are two common network-based intrusion detection systems that detect misuse on a network. Most desktop anti-virus programs also fall into this category. These systems operate by loading them with *signatures*, or indicators, of malicious activity and are built to look through network- or host-based data for items matching those signatures. They are popular because they are easy to implement, and the indicators are easy to share. The general weakness of misuse detection is the inability to

detect new threats, although they have a high detection rate for known threats (Kemmerer & Vigna, 2002).

Beyond the simple use of static signatures to detect computer system misuse based on simple events, there has been much research into better mechanisms for creating complex event signatures. Lin et al. (1998) developed a system whereby abstract simple event signatures were used and reasoned with based on the context of the system objects they operated on. The misuse signatures – or MuSigs – developed from their research, were designed to deal with situations where system objects could have multiple aliases that changed over time. By tracking the history of events on the system, MuSigs could provide more accurate detection of system misuse. Meier (2004) added enriched simple signatures with semantics about the system being protected. While his research was constrained to the domain of active databases, the concepts are applicable to other domains. Naldurg et al. (2004) presented a formalization and language for reasoning about temporal logic. The language can correlate events over time and is best applied to the types of attacks that are distinguishable from normal use activity using temporal information.

Many improvements on misuse detection systems came in the form of improved *attack languages.* Attack languages combine atomic, computed, and heuristic IOCs into intuitive formal languages that can be used by a software system to detect intrusions. (Vigna et al., 2000) defined six different categories of attack languages:

- **Event languages** are the basic input for analysis. They describe atomic events that occur on a system.

- **Response languages** describe actions to be take in response to an intrusion. They describe modifications to the systems security controls to change its security posture.
- **Reporting languages** are used to describe output formats for alerts and logs of security events.
- **Correlation languages** are used to fuse alerts from different detection systems in order to gain a higher-level understanding of security events.
- **Exploit languages** describe the steps taken by an attacker during an intrusion.
- **Detection languages** describe the manifestation of an attack and are used by intrusion detection systems to create detection rules.

STATL (Eckmann et al., 2002), SHEDEL (Meier et al., 2002), P-BEST (Lindqvist & Porras, 1999), as well as the popular NIDS software systems Bro and Snort, are examples of systems that leverage detection languages to detect misuse in networks and computer systems. Over the years, there has been other research into attack languages for graph-based detection of intrusion events (Staniford-Chen, 1998), describing distributed patterns (Krügel & Toth, 2002), and detecting attacks involving multiple systems (Vorobiev et al. 2008).

Misuse detection systems are useful for detecting known attacks because they can be programmed intuitively, the rules can be derived directly from intrusion analysis work, and the rule sets can be shared between organizations to spread intrusion detection knowledge and capability. Misuse detection systems have a high detection rate against known attacks. However, misuse detection systems are generally incapable of detecting

unknown threats. If a new intrusion technique is not identified by the rule set, then no alert will be triggered.

2.2.3. Anomaly Detection Systems

In contrast to misuse detection systems, anomaly detection systems build a mathematical model of what normal behavior looks like, then use that model to detect abnormal behavior, or anomalies. In an anomaly detection system, the detection engine is trained using labeled or unlabeled data from a training set using a wide variety of mathematical models. Once trained, the anomaly detection system works by gathering data from live hosts or the network, processing that data through the trained mathematical model, and detecting events that are recognized as anomalies (Bhuyan, et al., 2014).

In the context of intrusion detection, an anomaly detection system is a classifier-based machine learning system that examines security-related information to determine if suspicious activity is present. Such systems typically have the following components:

- **Raw input data** – this is the data, either in the form of training data sets or real-time data that may or may not contain malicious activity.
- **Proximity measures** – Statistical measures that take pairs of data objects and return a numerical value that is greater the more the two objects are alike
- **Labeled data** – A set of data annotated with information denoting, for each record, whether it is normal or anomalous
- **Classification method** – Depending on the availability of labeled data, this is either a *supervised* (there exists a labeled training data set), *semi-supervised*

(there is only training data for normal class of data), or *unsupervised* (no training data required)

- **Feature selection** – A method of selecting which parts of the data will be used in the classification algorithm
- **Reporting method** – The output when an anomaly is flagged, which typically includes a score or label, depending on the classification method

In a network-based anomaly detection system, the raw input data consists of captured network data, in the case of training data, or live network traffic monitored in real-time, during active anomaly detection. The most common data set used for network-based anomaly detection is the KDD Cup data set published by DARPA in 1999 for evaluating network intrusion detection systems (DARPA, 1999). Analysis of this data set (Tavallaee et al., 2009) reveals that this data set includes approximately 4,900,000 network connections labeled as either normal or an attack. There are four categories of attack present in the data: *Denial of service* (DoS), where the intent is to reduce the availability of the network system or service; *User to root* (U2R), where the attacker attempts to elevate privileges; *Remote to local* (R2L), where the intent is to gain remote access to local resources on a machine; and *Probing attacks*, which gather information about a system to learn how to circumvent security controls.

In addition to describing the methods used, Bhuyan, et al. (2014) highlighted the overall strengths and weaknesses of network anomaly systems. In general, these systems were judged to exhibit high performance and were capable of real- or near-real-time detection of anomalies, had reasonable detection rates for known attacks, and were

capable of detecting unknown attacks. However, the vast majority of network anomaly detection methods reviewed had high false positive rates. These systems used network-only datasets, ignoring data from internal and external hosts that could improve detection rates or reduce false positive rates. As such, most anomaly-based network intrusion algorithms are inadequate by themselves and would be more effective if incorporated into a system of algorithms that can exploit their strengths and minimize their weaknesses.

Host-based anomaly detectors work in much the same way as network-based systems, except they examine different types of data, to include system call traces, audit log data, and command-line information (Yeung & Ding, 2003). Although there is ample information on hosts – API call patterns, event logs, disk access patterns, memory usage, etc. – to train an anomaly detection engine, there is not much research into host-based anomaly detectors to date.

2.2.4. Hybrid Systems

In some cases, there can be advantages to combining techniques to build a stronger system. The hybrid system can combine host- and network-based detection or anomaly and misuse detection (or some combination of all four) to make a system that is more accurate than each of the component parts individually. Some examples of this type of system are described in this section.

Depren et al. (2005) created a hybrid network-based IDS that uses both an anomaly detector and a misuse detector, running in parallel. The anomaly detector used a self-organizing map (SOM) algorithm, and the misuse detector used a decision tree algorithm. The input to these systems was network data from the KDD99 dataset. The

output was sent through a rule-based decision support system (DSS), which was responsible for making a final decision on whether a specific network packet was malicious or not.

DT-SVM (Peddabachigari et al., 2007) combined decision trees and support vector machines in a hybrid learning approach to network-based anomaly detection. The system prototype was very limited, but it was successful in detecting some classes of attacks from network data.

Although some improvement to detection or false positive rates can be achieved by mixing and matching different types of detection techniques, the research reviewed offered only incremental gains.

2.2.5. Ontology-Based Intrusion Detection Systems

Research into using ontologies for security began as an evolution of research into taxonomies, which are hierarchical structures for classifying things in a topic area. Taxonomies in computer security work in the same manner as the classification of plants and animals. The taxonomy is a tree structure of mutually exclusive concepts and sub-concepts, with leaf nodes consisting of buckets where individual items that fit the category are placed. They are useful for understanding the depth and breadth of a research topic.

There have been taxonomies published with relevance to many computer and network security topics. Research spanning a decade described taxonomies for software security flaws (Landwehr et al., 1994; Aslam et al., 1996; Weber et al., 2005). Hansman & Hunt (2005) developed a taxonomy of network and computer attacks. Igure &

Williams (2008) published a taxonomy of attacks and vulnerabilities in computer systems. Lindqvist & Jonsson (1997), building on prior research, refined a taxonomy of computer security intrusions. None of these taxonomies are relevant to the problem of detecting APT intrusions.

The most relevant taxonomy research was published by Killourhy, Maxion, and Tan. In their paper "A defense-centric taxonomy based on attack manifestations" (Killourhy et al., 2004), they describe the creation of a system to test attack techniques versus known vulnerable programs and simulated a series of attacks. After examining the artifacts and IOCs resulting from this testing, they created a defense-centric taxonomy with a focus on how the attacks manifested themselves in the system from a network defender's point of view. The taxonomy was then used to predict whether or not an IDS would be able to successfully detect the intrusion based on the attack sequence's distinguishability from normal traffic.

While these taxonomies, the research by Killhourny et al., in particular, provided some structure to the computer security and intrusion detection research spaces, it became clear that they were not expressive and detailed enough to describe the domain knowledge and reason about it. This realization lead to research into ontologies.

Gruber (1993) defines an *ontology* as an explicit specification of an abstract, simplified view of the world that is to be represented for some purpose. Formally, it consists of sets of concepts (C) and attributes (A), a hierarchy of concepts (H), and a set of semantic relations (R_T).

Equation 1 Ontology

$$O = \{C, A, H, R_T\}$$

While there is some variation to this general model, an ontology formalized in this way is sufficient for representing the required elements of a knowledge base (Colace et al., 2012).

Undercoffer et al. (2003) as well as Raskin et al. (2001) argued that taxonomies are insufficient for use in an intelligent IDS. Taxonomies organize information and concepts with the goal of classifying them. However, they do not provide a knowledge representation of a field of study, as ontologies do. Because an ontology is a formalized knowledge representation of a topic area, it can allow a machine to reason about a collection of data by applying the concepts represented in that knowledge base.

Security-related ontology research first developed from research into the Semantic Web. Two review efforts from Gomes et al. (2009) and Blanco et al. (2008; 2011) provide an overview of security ontology development. Early security ontologies were used to describe many security-related domains, including the following:

- Sharing access control information and normalizing database schemas to facilitate inter-organizational database sharing (Mitra et al., 2006)
- Describing computer and network attacks (Vorobiev et al., 2008)
- Defining and describing the trust relationships and security properties of Semantic Web services (Denker et al., 2003; 2005)
- Firewall-based access to Semantic Web services (Ashri et al., 2004)
- Assisting in security operations (Tsoumas & Gritzalis, 2006)
- Describing security incidents (Martimiano & Moreira, 2005)

- Describing authorization and privacy policies for semantic web services (Kagal et al., 2004)

Other efforts to create security ontologies have been published, but were not included in the review of Gomes, et al. (2009), only some of which are useful in informing creation of an ontology for this research.

Schumacher (2003) published a core ontology for security, which contains a small number of key security concepts and relationships. While small, this ontology or variations of it appear to be a central component of many of the ontologies described in this section.

Fenz & Ekelhart (2009) describes a security ontology designed from the perspective of a security manager who needs to identify vulnerable assets and identify security gaps in their infrastructure. While it appears comprehensive in that regard, it does not incorporate any of the operational concepts and relationships required for automated intrusion detection and network forensics.

García-Crespo et al. (2011) outlines an ontology for adding access controls to semantic web applications.

An Wang et al. (2010) created an ontology that allows software vendors, security researchers, and tool developers to reason about vulnerabilities and countermeasures to avoid or remediate them.

The STUCCO project (Iannacone et al., 2015) includes an ontology intended to represent knowledge, from multiple information databases with differing formats, in one cohesive knowledge base. It represents knowledge about attackers, hosts, the software

packages running on them, and other information valuable to security practitioners in identifying attacks and vulnerable systems. While this ontology has a solid foundation, it does not represent many key concepts relevant to detection of APT-style attacks.

OntoSec (Martimiano & Moreira, 2005; Martimiano & dos Santos Moreira, 2006) is a security incident ontology developed with the goal of describing security incidents. It is similar in structure to the ontology developed by Undercoffer et al. (2003). Like that ontology, it does not contain concepts sufficient to represent the knowledge and expertise of a CSOC analyst. Specifically, it lacks knowledge of the internal network, digital forensics concepts, and knowledge of APT TTPs.

While many of these ontologies were useful in achieving the goals of the projects they were developed for, there are not sufficient for describing, in detail, knowledge of the security domain required to reason about APT activity and create a robust automated intrusion detection agent. Researchers from the MITRE Corporation were the first to outline the knowledge requirements for such an ontology (Obrst et al., 2012). A security ontology must be able to represent the properties of malware, languages for describing security incidents, attack patterns and process models including the attack kill-chain (Hutchins et al., 2011), persons and groups, time and geospatial information, events and situations, and network operations. Their research was formalized in the form of the CybOX schema (MITRE, 2015) and the emerging STIX format (Barnum, 2014).

Building on this research into security ontologies, several systems were developed that use ontologies to perform relevant security tasks, including intrusion detection, network forensics, or security event correlation.

One of the earliest systems built with this design was created by Undercoffer and his colleagues (Undercoffer et al., 2003b; 2004). They created an ontology to describe attacks based on information that the attack target would be able to observe during the attack. The ontology was created based on analysis of over 4000 attacks and was created using the DARPA agent markup language (DAML) and implemented using a Java-based knowledge base and agent shell (DAMLJessKB). The system was used in a distributed IDS where each host had its own anomaly-detection IDS, with coalitions of host IDS's sharing an ontology and knowledge base. The goal of the system was to reason over events triggered by the IDS's running on individual hosts to DDoS attacks. While this system was successful in detecting a specific DDoS attack, it is limited to real-time detection of attacks based solely on network traffic analysis. It did not have the ability to use host-based information or to detect the intrusion after it has occurred.

Abdoli & Kahani (2008; 2009) created a distributed IDS that utilizes an ontology and standard messaging format to analyze intrusion reports from host-based network intrusion detection systems to determine if they are true positives or false positives, or if similar intrusion activity was seen on other machines in the network. As an early example of this type of system, it provides a foundational design for more comprehensive ontology-based IDS's. However, it has some key limitations. While it uses host-based sensors, it only looks at network traffic and not at memory- or disk-based artifacts on the host computer. It does correlate events from multiple systems but does not go as far as collecting further information that could aid in determining if the alert is a true or false

positive. Finally, it does not incorporate the intrusion kill chain or information about attacker methodology that is useful in determining the scope and severity of the attack.

Vorobiev et al. (2008) outlined an ontology and ontology-driven distributed IDS designed to detect simple distributed attacks. The ontology described in this paper reduces to an attack language for detecting attacks comprised of events involving multiple computers. While it may be useful for describing distributed attacks, it is unclear how the system would use the ontology to reason about events to detect attacks either in real-time or after the fact. Further, the ontology includes no information about the network being protected by the distributed IDS, which would facilitate reasoning about the feasibility of attacks succeeding and removing false positives.

Hung & Shing-Min Liu's system (2008) uses the DAML+OIL security ontology to describe the intrusion detection strategy at an abstract level, without specific knowledge of the underlying network and security infrastructure. It maps the concepts from this intrusion detection strategy onto specific security and network primitives used in the network the intrusion detection system will protect. Then, the DAMLJessKB expert system shell (a Java-based expert system shell) is used to describe forward-chaining reasoning rules to detect specific attacks.

Isaza et al. (2009a; 2009b) created an intrusion detection and prevention system that models attack signatures and prevention actions using an ontology. It then leverages multiple techniques, including neural networks, K-means clustering, and support vector machines to detect intrusion activity. The system was tested using the DARPA KDD99 dataset. This system does not incorporate host-based intrusion artifacts or evidence-

based reasoning into its learning model. As such, it is best suited for classifier-based detection of malicious network events. Also, given its use of the outdated DARPA KDD99 dataset, its relevance to modern attacks is limited.

uCLAVS (Martinez et al., 2010) is a cloud-based system that uses multiple antivirus engines to scan potential malware. This system uses an ontological approach to marry the antivirus results with data from other types of sensors, including firewalls and captured network traffic. The ontology in this system is used to define attack signatures. The system does not use a learning approach. Instead, the ontology is used to create more complex rules than would be possible in a normal rule-based antivirus system. Further, uCLAVS does not examine any host-based artifacts when determining if a file is malicious or not. It makes its determination purely based on the contents of the file and network-based data.

Li & Tian (2010) present an architecture for an ontology-based system to correlate alerts from multiple types of security sensors, including network- and host-based sensors. The aim of the system is to correlate many atomic alerts to describe a larger attack on the system. While this paper describes similar architecture to other systems of its type, it is unique in that it uses a formal security state model to track the security of the system. The system does not incorporate learning or evidence-based reasoning in its approach and limits what goes into the knowledge base to alert information collected from host- and network-based sensors. It does not collect amplifying information useful in increasing accuracy of intrusion alerts.

Saad & Traore (2010a, 2010b) have conducted significant research into ontology-driven systems. They describe an ontology designed specifically to aid in automated computer and network forensics. Their ontology contains 111 concepts, as well as a collection of binary and n-ary relations between those concepts. The ontology supports deductive, abductive, and inductive reasoning. Its unique contribution is its model of multi-stage attacks. Saad & Traore's ontology can understand the different phases of an attack and use that knowledge to drive acquisition of forensic evidence. While this approach is similar to the approach described in this research proposal, it does not go as far. My approach models the APT group's multi-stage kill chain to drive the collection of forensic evidence to support intrusion detection. While their ontology appears to adequately cover concepts related to attacks, at a basic level it does not incorporate formalized information regarding the internal network and assets that are being protected or concepts required to reason about information obtainable by examining external data sources (Domain Name System (DNS) records, network location, or geolocation), or threat intelligence feeds. These sources of information, and the ontological concepts that are encompassed by them, did not mature until after the research was published.

In a follow-up paper (Saad & Traore; 2011), they proposed a method to aggregate multiple IDS alerts through semantic analysis to create meta-alerts. The method relies on an intrusion detection domain ontology and groups alerts together based on semantic similarity using a simple algorithm. The method only considers network-based information and was evaluated using the DARPA99 data set.

Later systems built upon this work to develop methods for extracting an attack scenario from a large volume of alerts by correlating the alerts based on semantic similarity (Ahmed, 2014; Saad et al., 2014). These systems are designed to identify the steps taken by the attacker during the attack. It does not incorporate any learning techniques and focuses solely on network-based data for alerts.

Colace, et al. (2012) developed a system that uses multiple ontologies describing attacks, effects, actions, and the environment, to facilitate a slow intelligence approach to increasing the accuracy of an intrusion detection and prevention system. In principle, the ontological approach taken by Colace and his colleagues is similar to my research project. However, their system only examines network data processed by the Snort IDS and uses a Bayesian network for its learning approach. Additionally, this system is focused on specific attack types and was tested using attacks from the BackTrack penetration testing tool set. It does not incorporate knowledge of APT methodology and techniques.

A system described by More et al. (2012) uses a knowledge-based architecture with three major components - a set of data streams, a knowledge-base, and a reasoning algorithm. The data streams include network- and host- based logs and sensor data. The knowledge-based reasoning engine leverages an attack language built using an extended version of the ontology proposed by Undercoffer. The knowledge base is built using a series of OWL assertions. The reasoning algorithm infers the existence of an attack by comparing information from the data streams to assertions in the knowledge base. The system was able to detect a buffer overflow attack in Adobe Reader.

Finally, Salahi & Ansarinia (2013) describe a model whereby attacks are predicted by reasoning over ontological concepts. Their ontology was built by incorporating concepts and knowledge from multiple taxonomies, including Capec, CWE, and CVE. The approach focuses on a simplistic attack model, whereby an attack consists of an attack pattern defining the properties of a single attack, a set of weaknesses causing attacks to happen, and vulnerabilities. Reasoning is done via OWL triples. The system does not yet incorporate learning, but the authors proposed using machine classification in future research.

2.3. Forensic Methods: Investigating the Attacker Lifecycle

When a security incident is generated by an intrusion detection system, it is sent to an analyst in the CSOC, who is responsible for determining if it is a true detection or false positive, understanding the severity, scope, and scale of the attack, and executing containment, remediation, and recovery actions. This part of the CSOC process is called Incident Response (IR) and is primarily driven by collection and analysis of digital forensic artifacts.

At a high level, the IR process follows a structured analytical approach (see Figure 4) whereby the analyst begins with one or more investigative hypotheses, determines a plan to collect evidence to support or refute those hypotheses, based on their knowledge of attacker methodology, possible locations of relevant evidence, and available collection tools, executes the collection plan, and then analyzes the evidence to synthesize conclusions as to which, if any, of their hypotheses were likely to be true.

This final step often leads to additional hypotheses, leading this to be a cyclical process ending when all remaining hypotheses are addressed.

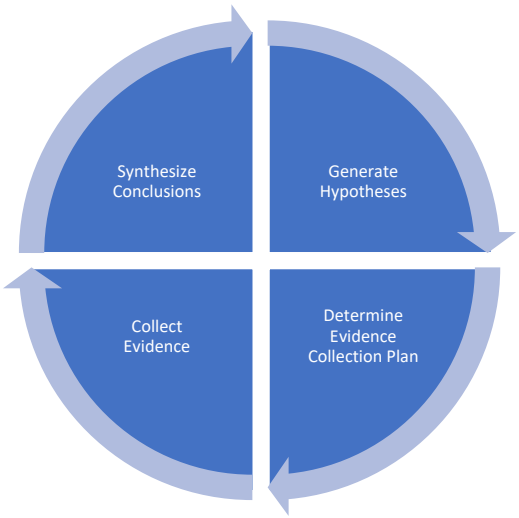

Figure 4 Incident Response Process

These processes are driven by both the investigator's knowledge of specific attack methodologies, as well as the attack lifecycle. There have been a few efforts to create abstract models of the process APT actors use to conduct hacking activity. The two most noteworthy are the Kill Chain model developed by researchers at Lockheed Martin (Hutchins et al., 2011), which identifies seven phases of an attack, and the Mandiant/FireEye Attack Lifecycle (Mandiant, 2013), consisting of eight phases. In the time since those two models were published, there has been much research into APT-style attacks, revealing more granular detail about the phases of the attacks.

For this research, we used a nine-step attacker lifecycle based on the Mandiant/FireEye model. Each step of the attacker lifecycle is a discrete step in the overall process APT actors use to gain unauthorized access to victim networks, find and gather information of interest, and exfiltrate it. A brief overview of each phase of the attack lifecycle is below.

1. **Reconnaissance** – During this phase, the attacker probes the target network and uses information available to the public – such as names, phone numbers, and email addresses from public web sites, social media sites, or public records databases – to learn as much as possible about the target network to make the attack more successful.
2. **Delivery** – This is the phase where the exploit or malware is delivered to the target network. Malware can be delivered to the victim network in a variety of ways. The most popular include SQL injection, web server exploits, email messages with malicious links or attachments, or watering hole attacks where a web site the target is likely to view is compromised.
3. **Initial Compromise** – When an external exploit works or the malicious email message is successful, the malware delivered is called the Stage 1 or *initial compromise*. It is the initial foothold the attacker obtains on the target network. The Stage 1 malware is typically a lightweight tool configured to communicate on a regular interval to a command and control (C2) server to ask for further instructions. Often the instruction is to re-contact the C2 at a

later time or to move to another C2 server. When the attack progresses, the C2 will tell the infected computer to download additional malware.

4. **Gain Foothold** – In this phase, the attacker delivers more sophisticated malware capable of providing direct, remote access to the target network. Typically, this malware, also called *Stage 2 malware*, contains a reverse shell function, the ability to download additional tools, and other functions allowing surveillance of the target network.
5. **Escalate Privileges** – When an attacker initially compromises the target network, it often has only normal user privileges. During this next phase of the attack, the APT actor will seek to gain administrator rights on the network. Such access can be gained by executing a local privilege escalation attack or exfiltrating and cracking password hashes. The result of this phase is often that the attacker will obtain legitimate credentials for logging into the network. In this case, phases 6-9 will not require the use of malware to execute.
6. **Internal Reconnaissance** – Once a foothold is gained and the attacker has sufficient privileges, they can begin to learn the layout of the target network. They will conduct network scans, locate critical servers (such as the domain controller, email server, and file servers), and identify security tools and infrastructure in place. The internal reconnaissance will enable the next phase of attack.

7. **Lateral Movement** – The attacker will then move around the network, exploiting open file shares and network services, to increase their ability to surveil the network, evade network defenses, locate data to exfiltrate, and enable redundant access to the network. Most networks do not monitor internal network communications, so it is common for this phase to go completely undetected.
8. **Maintain Presence** – APT actors are professionals and often have long-term collection requirements against their targets. As such, they have a vested interest in returning to the target network at a later time to collect additional data. This phase enables that persistence by establishing multiple, redundant points of access on the network that can survive reboots, system updates, and forensic investigations.
9. **Complete Mission** – There is always a specific mission purpose for an APT actor's activity. The last phase of the mission is to fulfill that purpose, often by collecting and exfiltrating information of value from the network through the attackers' collection of C2 servers. Exfiltration can be done in a wide variety of ways, often using the same methods that legitimate users would use to send information to external partners.

In general, modern security infrastructure is good at detecting the Delivery, Initial Compromise, Gain Foothold, and Complete mission phases of the attack. These phases of the attack cycle involve the use of malware, infected documents, delivery through email and compromised web sites, as well as bulk transfer of sensitive information out of

the victim network. Traditional IOC-based firewalls, intrusion detection system and endpoint protection software can detect many of these types of techniques, assuming the tools are known, and valid IOCs are available. Specialized tools like application aware network proxies and email scanning tools can detect delivery methods.

2.3.1. Collecting Against the Attacker Lifecycle

When collecting evidence to understand each phase of the attacker lifecycle, analysts must consider four major questions:

- **What are they collecting evidence of?** During each phase of the attacker lifecycle, different types of actions take place. Understanding which kind of activity needs to be discovered is critical when establishing an investigative plan.
- **What is the location of the evidence?** Depending on the phase being investigated, evidence can reside in a wide variety of network locations in a myriad of formats.
- **What type of evidence must be collected?** This includes knowledge of the physical format and level of volatility of the evidence to be collected.
- **What tools or methods are required to collect and analyze the evidence?** Answers to the other three questions and the level of human readability of the data help determine the tools and methods required to collect and analyze it.

Experienced incident responders understand how to use this information to collect and analyze forensic evidence to investigate a wide variety of threats by methodically collecting and analyzing data against the attacker lifecycle. Forensic collection and

analysis are large topics. For brevity, a subset of the attack actions, evidence locations, evidence types, and collection tools required for collecting forensic evidence against the attacker lifecycle is shown in Figure 5.

Attacker Lifecycle Phases	Evidence Of	Evidence Location	Evidence Types	Collection Methods/Tools
Reconnaissance	Port Scanning	Network Gateway	Netflow	Network Sensor
Delivery	Phishing Email	Email Server	Server Logs	Passive Network Tap
Initial Compromise	Watering Hole	Web Server	PCAP	Logical File Copy
Gain Foothold	Malware Execution	Compromised Host	DNS Logs	Log Search
Escalate Privileges	C2 Activity	Domain Controller	Firewall Logs	Raw Memory Copy
Internal Reconnaissance	Service Install	SIEM	Email Server Logs	Raw Disk Acquisition
Lateral Movement	Temp File Creation	Network Sensors	Email Content	Raw Artifact Parsers
Maintain Persistence	Network Mapping	File Server	Endpoint Logs	Host Collection Agent
Accomplish Mission	Use of Admin Tools	Database Server	Memory Images	Log Shippers
	Admin Acct. Creation		Filesystem Artifacts	
	Data Gathering		File-based Artifacts	
	File Encrypt/Compress		Authentication Logs	
	Data Exfiltration			

Figure 5 Collecting against the Attacker Lifecycle

During a forensic investigation, an analyst will use the phases of the Attacker Lifecycle to guide collection of evidence of types of attack activity. The type of activity to be collected against informs the investigator of where the evidence may be located in the network. This in turn helps determine which evidence types to collect from those locations and which tool swill be needed to collect those evidence types. Because specific attack activity can span across multiple Attacker Lifecycle phases and the evidence locations, types, and collection methods likewise have a many-to-many

relationship with each other there is no strict segmentation between the columns. For example, an incident responder looking for *Gain Foothold* phase activity may look for evidence of *C2 Activity* by collecting *Firewall Logs* from the *Network Gateway* using *Logical File Copy*. She might also look for *Gain Foothold* phase activity by collecting evidence of *Malware Execution* from a *Compromised Host*, which might require a *Memory Image* to be collected and examined using *Raw Memory Copy* and a *Raw Artifact Parser*. The combinations of collection and analysis capabilities an incident responder must leverage to identify sophisticated threats is very large, which is why their skillset is rare an in high demand.

Our research leveraged a subset of this methodology to train cognitive agents to be able to orchestrate and automate the threat detection process. The details will be discussed in Section 3.3.1.

2.4. Knowledge-Based Learning and Evidence-Based Reasoning

The learning model used in this research is rooted in the theory of knowledge-based learning and evidence-based reasoning. This is an area of research focusing on the creation of collaborative computational processes of evidence in search of hypotheses (through *abductive* reasoning which shows that something is *possibly* true), hypotheses in search of evidence (through *deductive* reasoning which shows that something is *necessarily* true), and evidentiary testing of hypotheses (through *inductive* reasoning which shows that something is *probably* true). This section will summarize this area of theory, as published in *Knowledge Engineering: Building Cognitive Assistants for Evidence-Based Reasoning* (Tecuci et al., 2016a).

Figure 6 is a high-level overview of the framework for the evidence-based reasoning process (Tecuci et al., 2016a). This framework is based on the work of intelligence analysis and is an iterative, collaborative process between an analyst and the knowledge-based learning assistant.

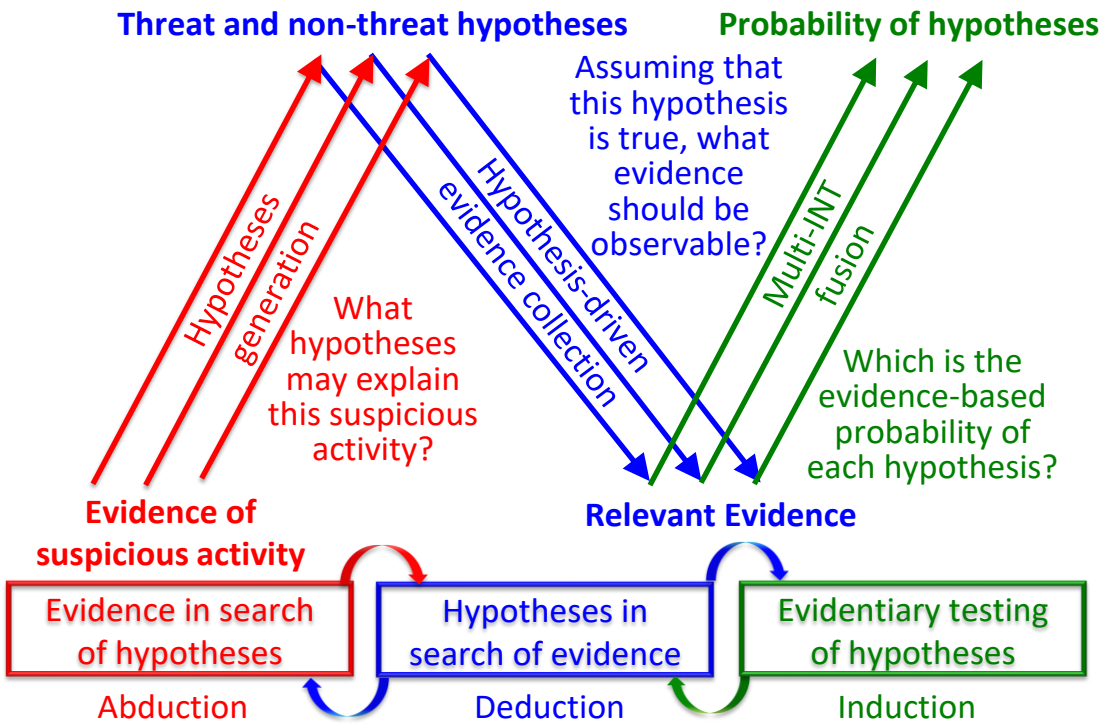

Figure 6 Evidence-based reasoning as the discovery of hypotheses, evidence, and arguments

2.4.1. Evidence in Search of Hypotheses

The first step in the evidence-based reasoning process is evidence in search of hypotheses which involves *abductive* reasoning. Formally, if **a** → **b**, and evidence **b** is observed, then **a** is *possibly* true. We write: **b** → *possibly* **a**. It could also be true that **c** →

b, where **c** is some competing hypotheses explaining **b**. The examination of multiple hypotheses that could explain the existence of evidence of a specific event is called abductive (or imaginative) reasoning. Figure 6 shows an example of how this process works. Given an item of evidence, multiple, competing hypotheses could explain its existence.

2.4.2. Hypotheses in Search of Evidence

Once a set of hypotheses are identified, they are used to generate searches for discrete elements of additional evidence that can be used to evaluate them. This is a process of *deductive* reasoning. Formally, if **a** → **b & c** and **a** is a hypothesis, then **b** and **c** must be true. We write: **a** → *necessarily* **b & c**. **b** and **c** can be broken down further into a tree of sub-hypotheses until only elementary hypotheses, or observables remain. The leaves of the deductive reasoning tree drive searches, or collection tasks, to identify evidence that can be used to test the hypotheses, as discussed below.

2.4.3. Evidentiary Testing of Hypotheses

When evidence is found it is either stored as evidence and/or placed into the knowledge base as instances of concepts in the ontology. This evidence is evaluated based on its *credibility*, or the degree to which it can be believed, *relevance*, or the probability of the hypotheses if the evidence were true, and the *inferential force*, or the probability of the hypothesis given only this evidence. These values are represented on a probability scale, as shown in Figure 7.

Probability scale

L11	100%	certain
L10	95-99%	almost certain
L09	90-95%	
L08	85-90%	very likely
L07	80-85%	
L06	75-80%	
L05	70-75%	more than likely
L04	65-70%	
L03	60-65%	likely
L02	55-60%	
L01	50-55%	barely likely
L00	50%	lacking support

Figure 7 Probability scale

Once all sub-hypotheses of a node in the tree are evaluated, then their probabilities are combined to determine the overall probability of the parent hypotheses. This is an *inductive* reasoning process. Formally, **b** → *probably* **a**. When there are multiple evidence items $\mathbf{b}_1$, $\mathbf{b}_2$, etc., their probabilities must be combined to calculate the probability of **a**. This uses the min/max probability combination rules common to the Baconian probabilities (Cohen, 1977; 1989) and Fuzzy probabilities (Zadeh, 1983). In particular the probability of a conjunction of probabilities is the minimum of these probabilities, while the probability of a disjunction of probabilities is the maximum of these probabilities.

The process of combining the probabilities of sub-hypotheses to calculate the probabilities of parent hypotheses continues until the probability of the root hypothesis is calculated.

2.5. Ontologies and Learning

Once a reasoning tree has been created by an analyst, the agent will learn rules from it. The rules are learned as generalizations of reasoning tree fragments, using the ontology as a generalization hierarchy for learning. The generalization of a reasoning tree fragment (or example) into a rule is also guided by the explanation of why that reasoning tree fragment is correct. The type of learning from examples and their explanations has advantages over other machine learning methods because the agent can learn from as few as one example where classifier methods such as neural networks or decision trees need many examples to learn a function.

Evidence-based reasoning has been used successfully to aid analysis in multiple disciplines. TIACRITIS (Tecuci et al., 2011), Disciple-CD (Tecuci et al., 2014; Tecuci et al., 2016b), and Cogent (Tecuci et al., 2015; 2018a) are all knowledge-based agents that assist in analysis or teach analysts critical thinking skills.

While the demands of APT analysis and detection require much more automation than the agents created for other topics, it follows the same intelligence analysis methodology. Rather than having an analyst manually searching for evidence, as is done in Disciple-CD, a robust search agent can be created, customized for a specific network environment, that can automate the search for and analysis of evidence required to reason about the presence and scope of an APT-style computer intrusion.

3. RESEARCH OVERVIEW

The information in this chapter has been previously published in peer-reviewed conference proceedings (Meckl et al., 2015; Meckl et al., 2017; Meckl et al., 2018) and ACM's Computing in Science and Engineering (Tecuci et al., 2018b).

For this research, I developed and evaluated a prototype system capable of using cognitive agents to autonomously orchestrate security incident response and investigation for sophisticated cybersecurity threats. It is a complex software system which required research and development of a theoretical model of APT detection, which uses knowledge-based learning agents with evidence-based reasoning, integrated into a CSOC environment using a custom-developed collection management system and a curated selection of collection and analysis agents to detect APT activity in an autonomous fashion. At its core, this research built on top of the theoretical models of learning and reasoning of the Disciple approach described in Section 2.4 and extended it to leverage knowledge of cybersecurity, sophisticated attacker behavior models, and knowledge of a network environment to automate collection and analysis of digital evidence to detect threats in an agile manner.

While this research is based on my original idea and vision, it was conducted as part of a larger team from the Learning Agents Center, with whom I built a system called Cognitive Agents for APT Detection (CAAPT). The research was funded via a contract from Air Force Research Labs. The research team contributed a substantial amount of research and development work on novel and necessary new learning and reasoning

capabilities required to support this research. My primary role in the CAAPT project was to leverage my cybersecurity subject matter expertise to develop the theoretical model of APT detection, design the overall system architecture, develop the search, collection, and evidence gathering strategy (to include development of the novel Collection Manager system necessary to enable Disciple cognitive agents to integrate with CSOC infrastructure), design and build the test network environment, and design and execute the testing and evaluation protocol for CAAPT. In this section, I will provide an overview of the CAAPT system, focusing primarily on my contributions to the research.

3.1. CAAPT Architecture Overview

At a high level, CAAPT is a collection of specialized agents, each designed to autonomously handle a specific phase of the detection process. While each agent includes only the reasoning or processing modules required to carry out its specialized responsibility, the agents use shared knowledge bases as represented in the left-hand side of Figure 8. The agents are integrated into a specific CSOC and collaborate in intrusion detection, as explained below.

The **Alert Generation Agent** receives alerts from a variety of sources, such as a network IDS, network anomaly detection, or endpoint protection alerts. In the current implementation, however, CAAPT only processes BRO (Paxson, 1999) alerts.

The **Trigger Agent** represents each alert into a different knowledge base which inherits knowledge from the shared knowledge base (consisting of the General Knowledge Base and the APT1 knowledge base). These knowledge bases are organized

into a *hypotheses generation queue* from which they are extracted by the Hypothesis Generation Agent.

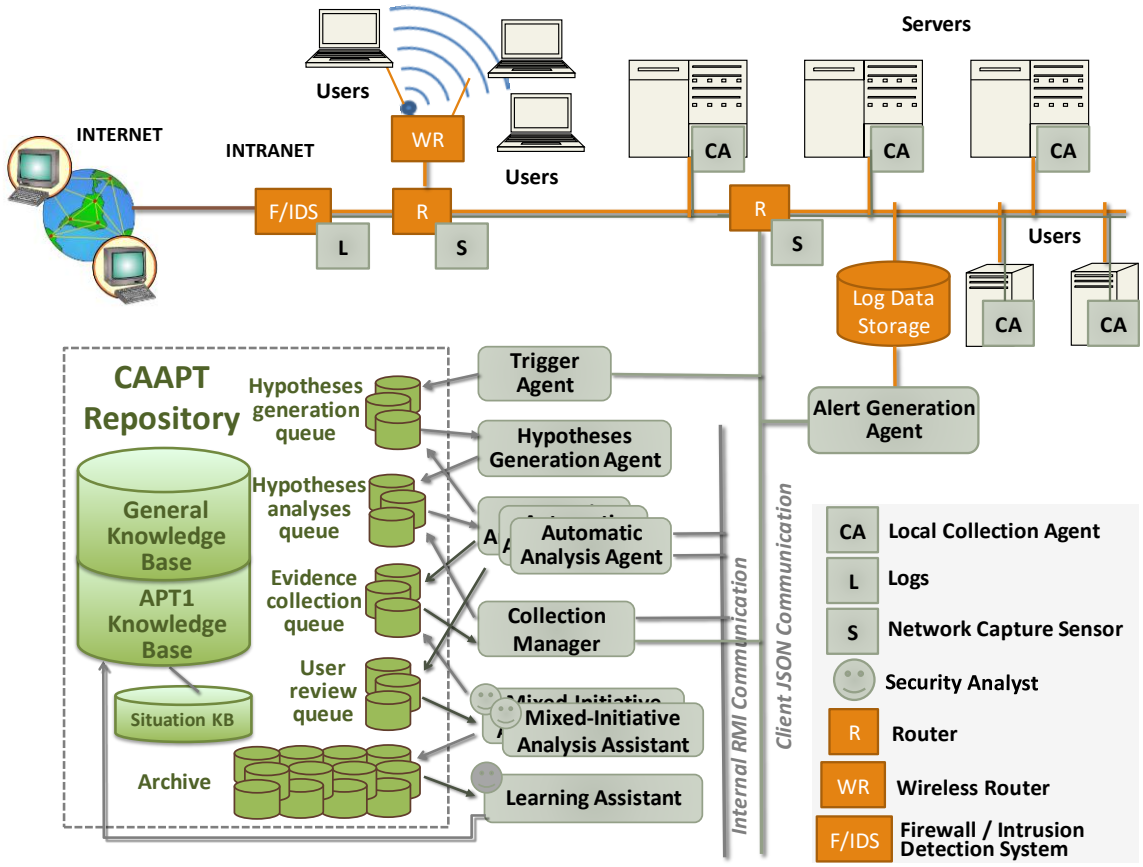

Figure 8 CAAPT system architecture overview

The **Hypotheses Generation Agent** generates the hypotheses corresponding to a trigger and places the knowledge base into the *hypotheses analyses queue* from which they are extracted by an Automatic Analysis Agent.

The **Automatic Analysis Agent** decomposes the hypotheses from such a knowledge base, as much as possible, down to the level of evidence collection requests.

Then it places the knowledge base into the *evidence collection queue* from where they are extracted by the Collection Manager.

The **Collection Manager** invokes specialized collection agents to search for evidence on the network. Then it represents the retrieved evidence into the corresponding knowledge bases and places these knowledge bases back in the hypothesis analyses queue.

When an automatic analysis agent has performed the most complete analysis possible of the alternative hypotheses corresponding to a trigger, it places the knowledge base into the *user review queue*, to be used by the Mixed-Initiative Analysis Assistant and the cyber analyst.

The **Mixed-Initiative Analysis Assistant** interacts with the cyber analyst, either by alerting the analyst of a detected intrusion, or by collaborating with them to finalize the analysis. After an analysis corresponding to a trigger is completed and necessary actions have been taken by the cyber analyst, the knowledge base is placed into an archive by the mixed-initiative analysis assistant.

The knowledge bases from the archive are used by the **Learning Assistant** and an expert cyber analyst to further refine the ontology and rules shared by the specialized agents.

3.2. Theoretical Model of Attacker Behavior

For an autonomous system to reason about and detect attack behavior it requires both knowledge describing attack activity and rules for applying that knowledge. This information is stored in the knowledge base and is authored by a collaborative team

comprised of a knowledge engineer and a subject matter expert. This section describes the theoretical model for attack detection, using the APT1's AURIGA malware (Mandiant, 2013) as a case study.

3.2.1. Ontology and Knowledge Requirements

Cybersecurity experts intuitively use a variety of knowledge domains in their analysis of sophisticated threats. For CAAPT, that knowledge is formalized into an ontology comprised of several knowledge domains. An illustrative ontology fragment is shown in Figure 9. This is the core knowledge generally used by a CSOC analyst to investigate the technical aspects of a sophisticated attack.

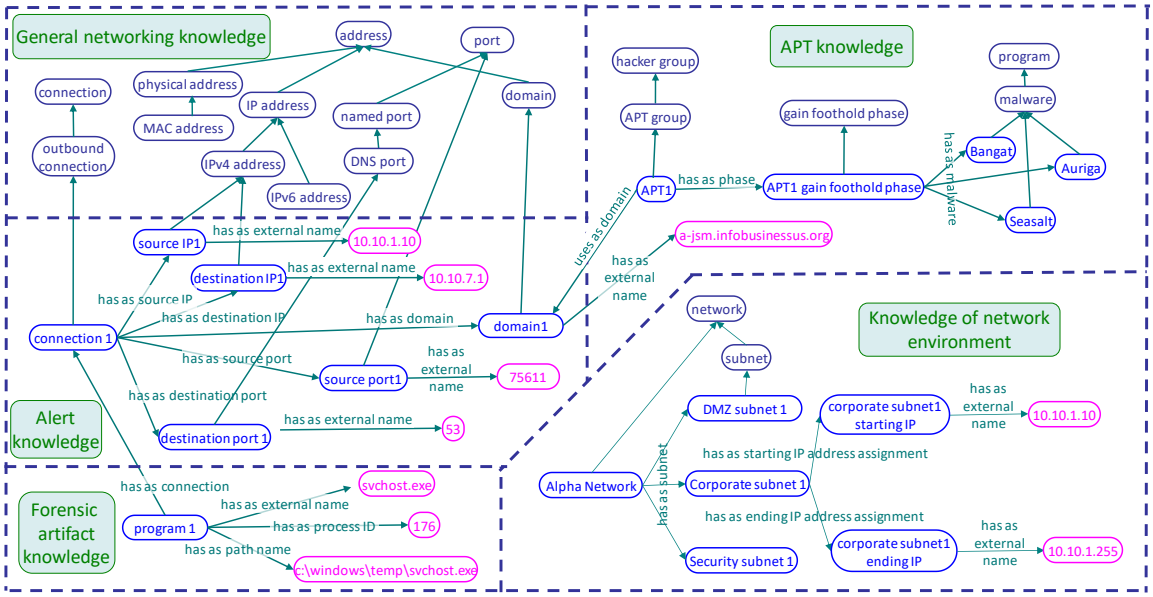

Figure 9 CAAPT ontology overview

General Networking Knowledge involves understanding of network devices and protocols, and how they relate to each other in a modern networking environment. This knowledge is used by the system, for example, to know what IP address a domain is mapped to and whether the IP address is a routable public IP address or non-routable internal IP address.

Alert Knowledge represents what specific information is learned when a security alert is raised by a CSOC's security infrastructure. Figure 10 shows an example of the

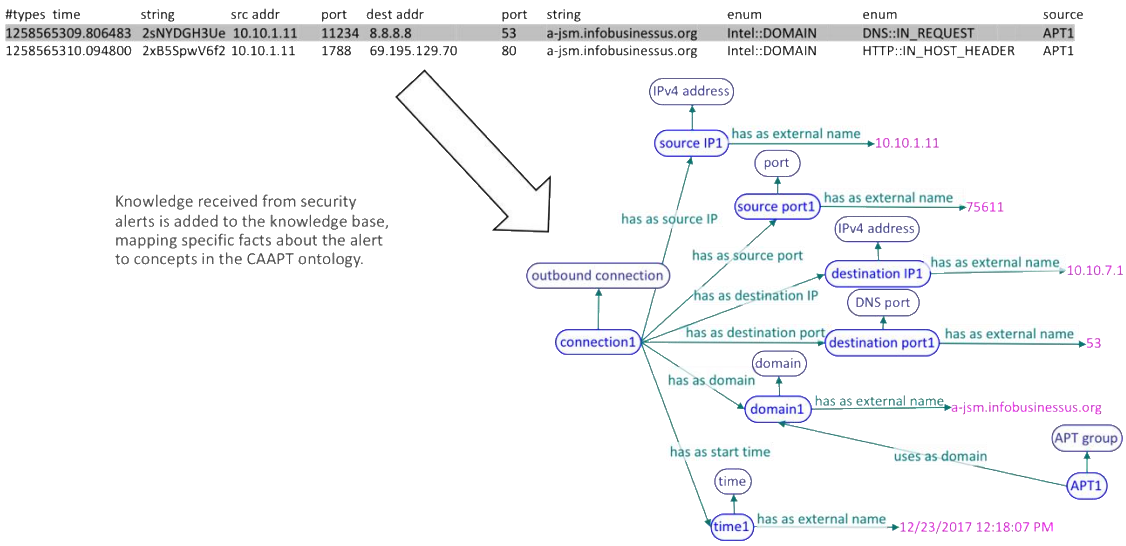

Figure 10 Alert knowledge generated from BRO alert

security knowledge learned when an alert is raised by the BRO intrusion detection system. In this example, BRO triggered an intrusion alert because the computer with IP address 10.10.1.11 (an internal, non-routable IP address) performed a DNS lookup for a known APT1 domain. When the alert is ingested by CAAPT, the facts in the BRO alert

are mapped to the ontological concepts they are instances of. The result is an ontology fragment representing the knowledge added to the knowledge base from the BRO alert.

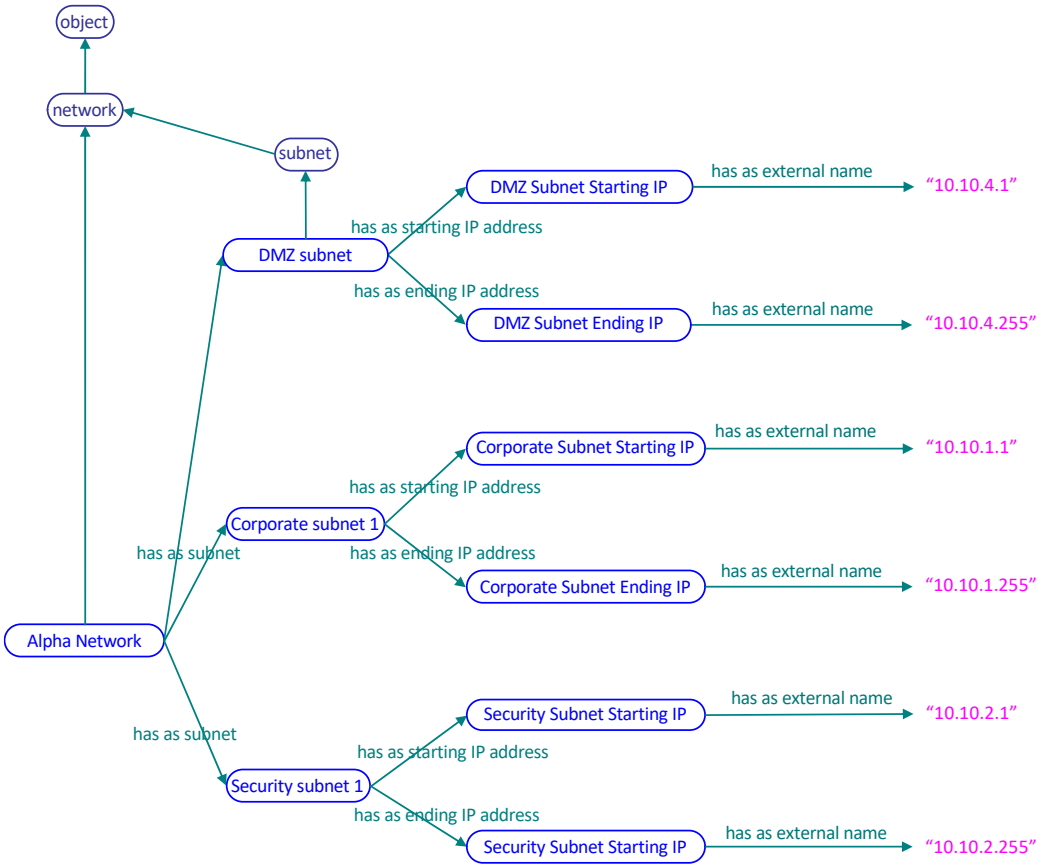

Figure 11 Simplified ontology for network topology

Knowledge of the Network Environment covers information specific to the network the CSOC is charged with monitoring. It includes knowledge about the network topology, what classes of users operate on segments of the network, critical assets, and vulnerability data. For experiments with CAAPT, a simplified ontology was used to

describe the network topology, as shown in Figure 11. The simplified ontology describes, at a high level, the subnets available on the network.

An important aspect to detection of sophisticated threats is **Knowledge of the Attacker Lifecycle**. As discussed previously, due to the organized nature of a sophisticated threat group's operations, security researchers have been able to create a semi-formal model of their methodology, called the Kill Chain (Hutchins et al., 2011) or Attacker Lifecycle (Mandiant, 2013). As shown in Figure 12, we have created a formal ontology for the attacker lifecycle, allowing CAAPT to reason about it.

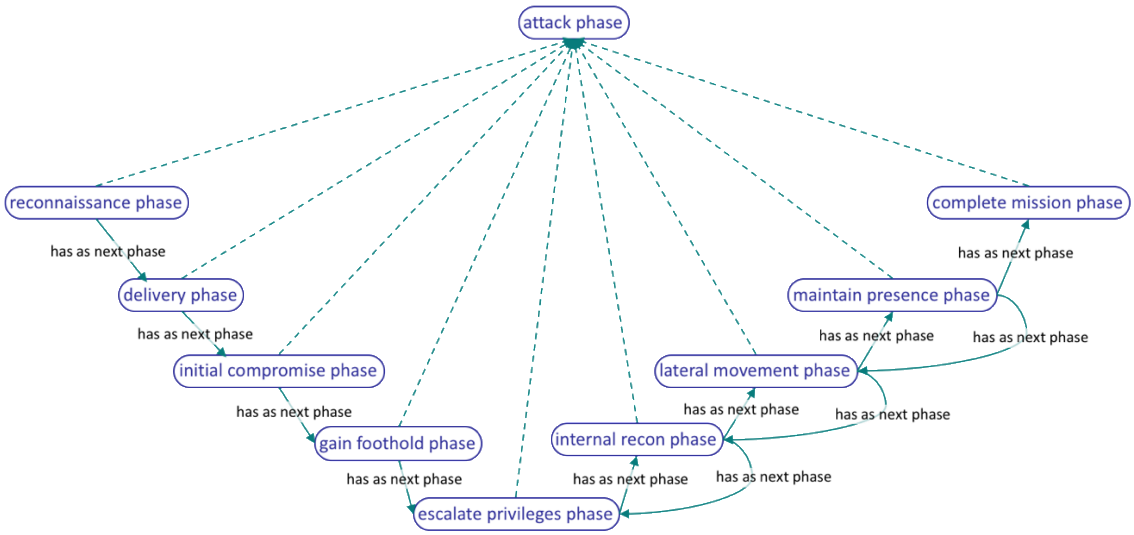

Figure 12 Ontology for the Attacker Lifecycle

Figure 12 shows an ontology for the steps of the attacker lifecycle described in **Section 2.3**. Generally speaking, an attacker must go through these steps to execute an attack on a target network. It should be noted that once an attacker gains administrator

access to a network in the *escalate privileges phase*, and begins internal recon and lateral movement, the process becomes iterative. To address this, the ontology includes *has as next phase* attributes from some of the attacker lifecycle phases pointing back to previous phases. This construct allows the ontology to more accurately describe the methodology of a sophisticated attack.

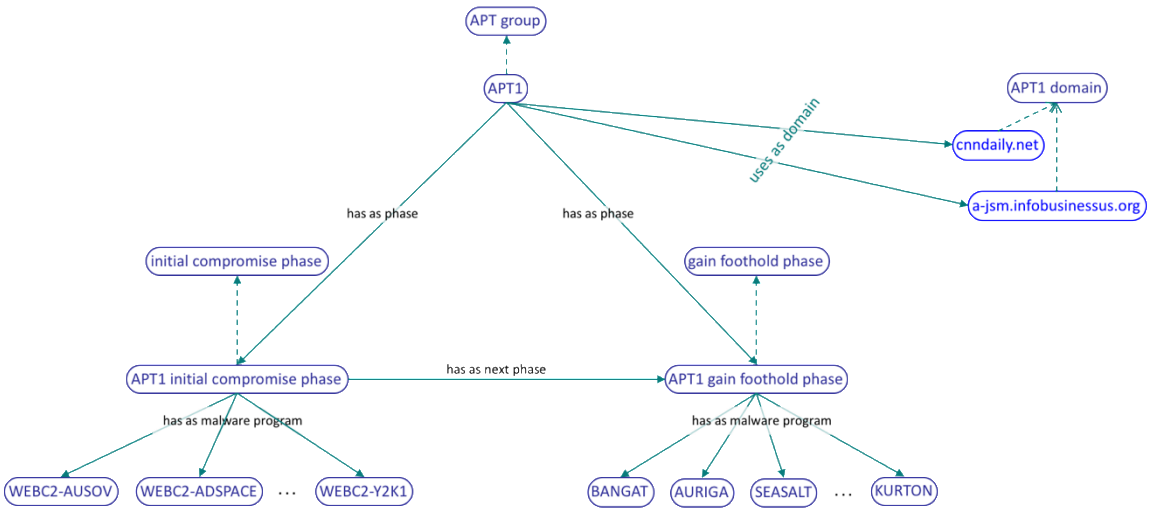

Figure 13 APT1 knowledge fragment

Attacker Knowledge is based on either publicly available threat intelligence or is the result of manual analysis of a threat. Formally, it is a description of a specific group's attacker lifecycle, using the ontological construct from Figure 13. Because this research primarily used APT1 as a case study, I have encapsulated attacker knowledge under the *APT Group* concept in the ontology. *APT Group* is a sub-concept of the *hacker group* concept as shown in the top right of Figure 9. Figure 13 shows a fragment of CAAPT's

ontology for the attacker group APT1. For each phase of the attacker lifecycle, we have a sub-concept for the attack group APT1. In each of those phases, knowledge of malware, IOCs, or other methodology is encapsulated. For example, we know from (Mandiant, 2013) APT1 uses different malware and other tools for each phase of their attack. *WEBC2-AUSOV, WEBC2-ADSPACE,* and other Stage 1 malware is used during the *APT1 initial compromise phase* to gain initial access to a network. *BANGAT, AURIGA, SEASALT, KURTON*, and other Stage 2 malware is used during the *APT1 gain foothold phase* of the attack.

There is some knowledge of an attacker group, including people associated with it, IP addresses, and domains used for command and control, not specifically associated with phases of the attacker lifecycle. To account for this type of attacker knowledge, the CAAPT ontology has some features associated directly with the *APT Group* concept. In Figure 13, the domains associated with the group are associated with *APT1* using the *uses as domain* feature.

Finally, the CAAPT knowledge base includes **Forensic Artifact Knowledge**, also referred to as **Malware Knowledge**. This includes knowledge of the data used or left behind by the malware used by an attacker group and where to look for it on a network. In our ontology, specific forensic artifacts, often called indicators of compromise (IOCs), are mapped to the malware programs that generate them. The malware programs, in turn, are mapped to the specific APT group or groups using them.

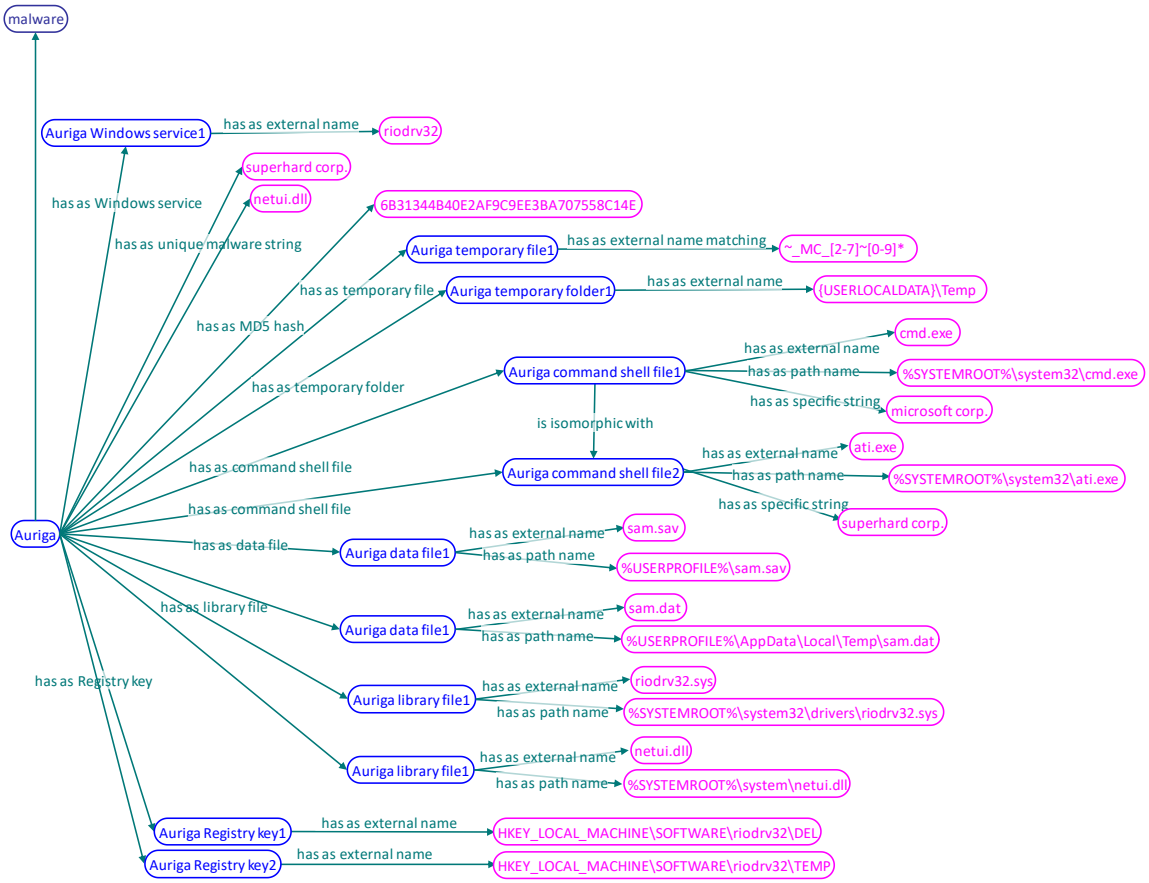

Figure 14 Forensic artifact knowledge

Figure 14 above shows an ontology fragment describing the malware knowledge of the AURIGA malware used by APT1 during the *APT1 gain foothold phase* of their attack methodology. Using threat intelligence published in (Mandiant, 2013), we have identified several forensic artifacts left behind by the malware, which are mapped to the *AURIGA* concept instance in the ontology. AURIGA persists through a system reboot by registering itself as a Windows service with name *riodrv32*. This is captured in the knowledge base by associating the service name *riodrv32* with *AURIGA* using the attribute *has as Windows service* relationship. Other forensic artifacts associated with the

AURIGA malware are similarly associated with it in the ontology, including unique strings included in the malware file, Registry keys, temporary files, data files, library files installed with the malware, and the hash of the binary file itself.

3.2.2. Abductive Trigger Generation Using Threat Intelligence

This research is primarily focused on a use case where a security alert is generated by some part of the CSOC's security infrastructure and an analyst is required to conduct follow-on analysis to determine whether a threat was accurately identified, the root cause of the attack, and the scope of the attack. The first step in this process is to use one or more detection technologies to identify potential threats using available threat intelligence, and use the information provided to trigger the abductive reasoning process. This section describes the process CAAPT uses to generate abductive triggers from threat intelligence.

At its core, security alerts are created by combining three things: 1) **Data** from the network or a host collected or scanned in real-time; 2) a **Security Sensor**, which applies an algorithm to the data collected or scanned; and 3) **Threat Intelligence** data, used by the Security Sensor to identify threats in the data. An overview of this structure is shown in Figure 15.

The output of this process is a set of Security Alerts which come in a variety of formats but are often sent to a SIEM system based on tools such as Elasticsearch, Splunk, or QRadar. In the case of CAAPT, all security alerts are forwarded to our Elasticsearch server which is used as the central repository for all log data.

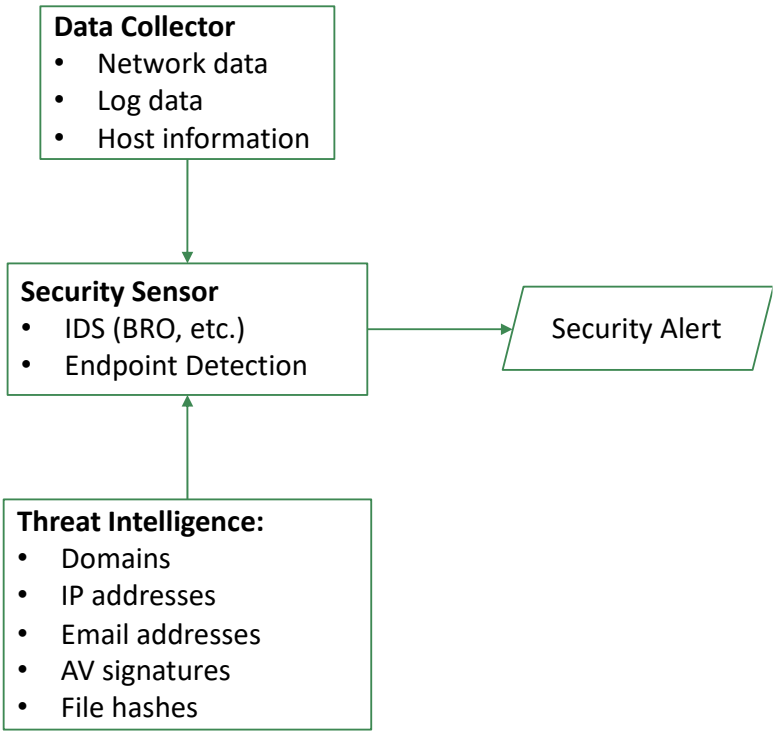

Figure 15 Generating alerts from threat intelligence

For CAAPT's development and test network, BRO (Paxson, 1999) was chosen as the IDS. On top of its ability to easily consume threat intelligence and efficiently apply it to identify threats, it also generates logs for other data with security value, including DNS lookups, connection information (network flow data), HTTP connection, URL strings, and digital certificates used for SSL connections. BRO logs all of this data into flat files in the comma-separated value (CSV) format. Because the rest of CAAPTs collection systems use JSON as the standard format, the BRO log entries must be re-formatted for use. Figure 16 shows an overview of the process by which a BRO alert log entry becomes an alert message sent to the CAAPT Trigger Agent.

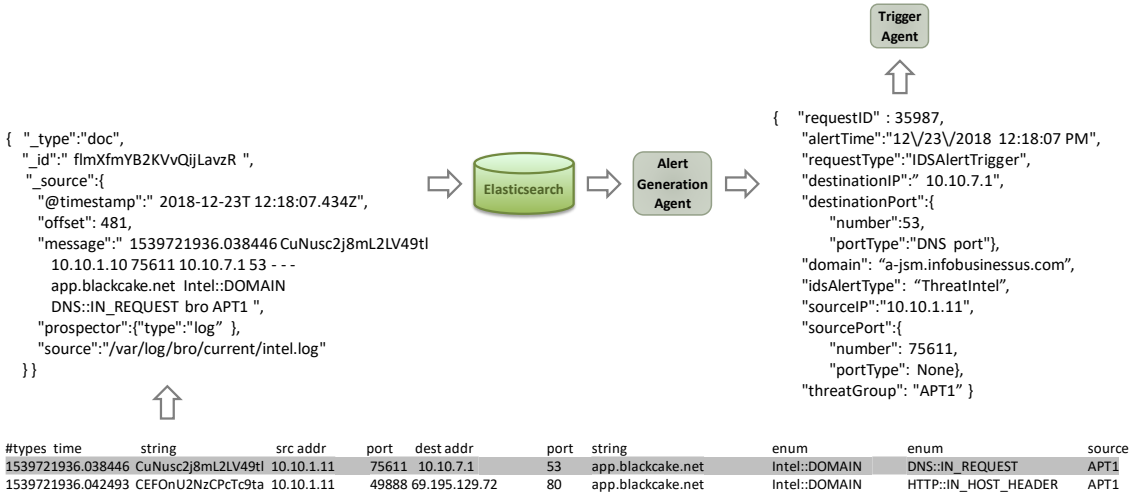

Figure 16 How a BRO alert becomes an abductive trigger

The first step in the process is to convert logs from the CSV format to a JSON message and transport the log entry to our Elasticsearch database. This is done using a program called *FileBeat* (2018). It is a program provided by Elasticsearch for log shipping, which constantly reads the end of specified files, looks for new entries, converts them to JSON, and sends them to Elasticsearch.

Next, a process is required to look in Elasticsearch for new alerts and send them to the *Trigger Agent*. I developed a custom Windows service program, called the *CAAPT Alert Generation Agent* to perform this task. This agent simply polls Elasticsearch on a specified interval, looking for log entries generated by BRO. For each one found, a new *Trigger Agent* message is created using relevant information from the BRO alert. This new message is then sent to the *Trigger Agent* to start the abductive reasoning process.

In the example in Figure 16, an alert was generated by BRO because a computer it was monitoring made a DNS request to resolve a domain known, via threat intelligence,

to be associated with APT1. Filebeat is an open source application supporting a wide variety of uses. As such, the message sent to Elasticsearch contains several extemporaneous data elements. For the purposes of threat detection, the system is primarily concerned with the information contained in the *message* data element. The *Alert Generation Agent* parses that field, converting the relevant data fields into appropriate data elements required for the JSON message on the right.

When this message is received by the *Trigger Agent*, it is added to the knowledge base in the form of an ontology fragment, as shown in Figure 17. Each field in the JSON message is ingested as an instance of a concept in the CAAPT ontology. In this example, knowledge of the connection triggering the BRO alert is captured. A connection has a source and destination IP address and port, and a timestamp. By applying learned rules for what it should do when receiving a BRO alert of this type, the *Trigger Agent* goes further, identifying port 53 as a *DNS port*, associating the domain *a-jsm.infobusinessus.org* with the connection, and further recognizing the domain's association with APT1 using knowledge of the attacker group already in the knowledge base.

When the process described in Figure 17 is complete, the *Trigger Agent* places a new knowledge base to be used for further analysis into the *hypothesis generation queue*. The *Hypothesis Generation Agent* then uses the new knowledge base for the abductive reasoning process, using learned rules to create a set of competing hypotheses which could explain why the alert was generated. First, an *indicator rule* is matched, which

generates a hypothesis from the suspicious connection that there is an active APT1 intrusion on the network. Then a *question rule* is matched, to generate a question which

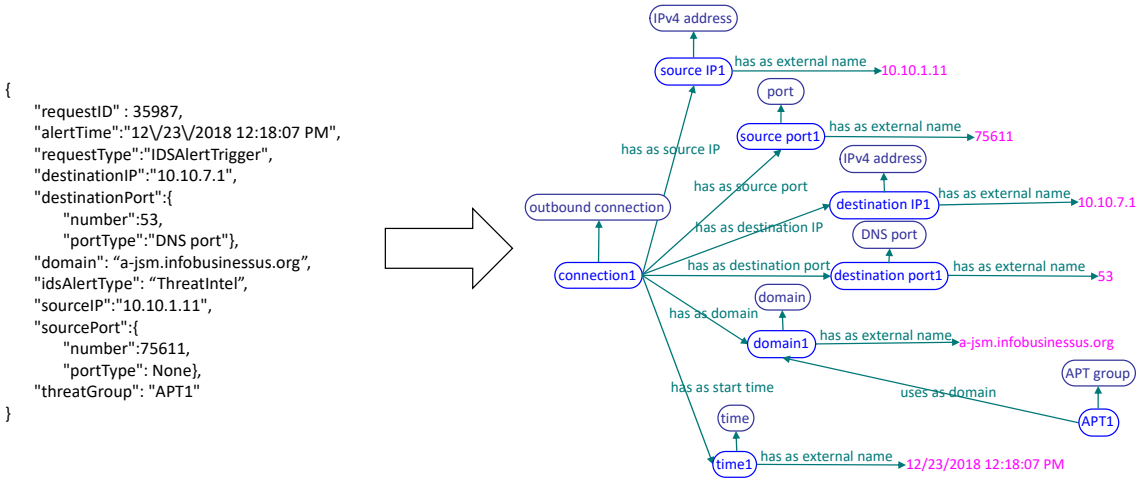

Figure 17 Trigger ontology fragment

the previously generated hypothesis could answer. From the *question rule*, multiple competing plausible hypotheses are created, which also could answer the question. This completes the first phase of the theoretical model of threat detection.

Figure 18 shows an example of the abductive reasoning process. Using an indicator rule, a plausible hypothesis is generated, based on new knowledge of a suspicious connection to an APT1 domain identified by BRO. In this case, the hypothesis is the connection is part of an APT1 intrusion. However, there are multiple hypotheses which could explain the connection, including those that would conclude the threat detection is a false positive. In this example, we offer two plausible false positive

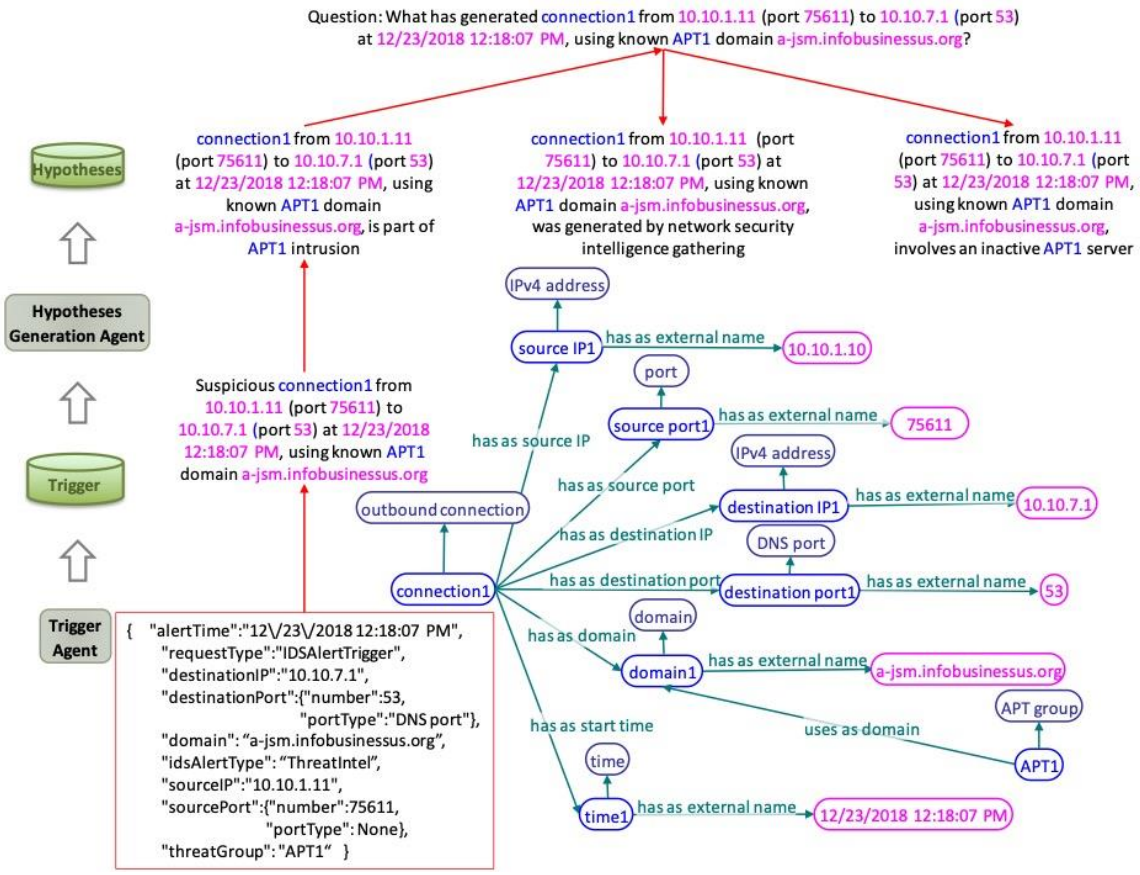

Figure 18 Hypothesis generation process

hypotheses. The first is the connection was generated as part of security intelligence gathering. Security operations or research personnel often accidentally trigger security alerts performing their duties. The second example is the situation where the C2 server is inactive. Sophisticated threat actors often use dynamic DNS providers to "park" their C2 domains (e.g., mapping them to localhost or a non-routable IP address) while not in use and then mapping them to a real C2 server when the group begins operations. They will also abandon domains once they have been discovered by threat researchers to evade future discovery. This false positive hypothesis encapsulates this scenario.

3.2.3. Search Agents for Hypothesis-Driven Search

Once a set of hypotheses are generated, the next phase is the deductive reasoning process, where each top-level hypothesis is decomposed into one or more sub-hypotheses. The process continues until a set of leaf hypotheses are generated requiring one or more searches for evidence. This overall process is called *hypothesis-driven search*.

Figure 19 shows an example of the initial hypothesis decomposition tree used to for detection of an APT1 intrusion. At the top level, we decompose the hypothesis stating the network connection which caused the BRO alert is the result of an APT1 intrusion into two sub-hypotheses. The sub-hypothesis on the left states the connection involves an active C2 server. This hypothesis is further broken down to its two components: the domain *a-jsm.infobusinessus.org* was active at the time of the connection and was registered using a dynamic DNS provider. These two sub-hypotheses are typically true when there is an active APT1 attack. The sub-hypothesis on the right states the program used in the attack is APT1 malware.

The leaf nodes of the decomposition tree result in three different searches for evidence. All three searches will eventually lead to evidence being added to the knowledge base for this security alert investigation. The search for the program that made the network connection will result in that branch of the decomposition tree being further decomposed, asking more detailed questions about the behavior of the malware.

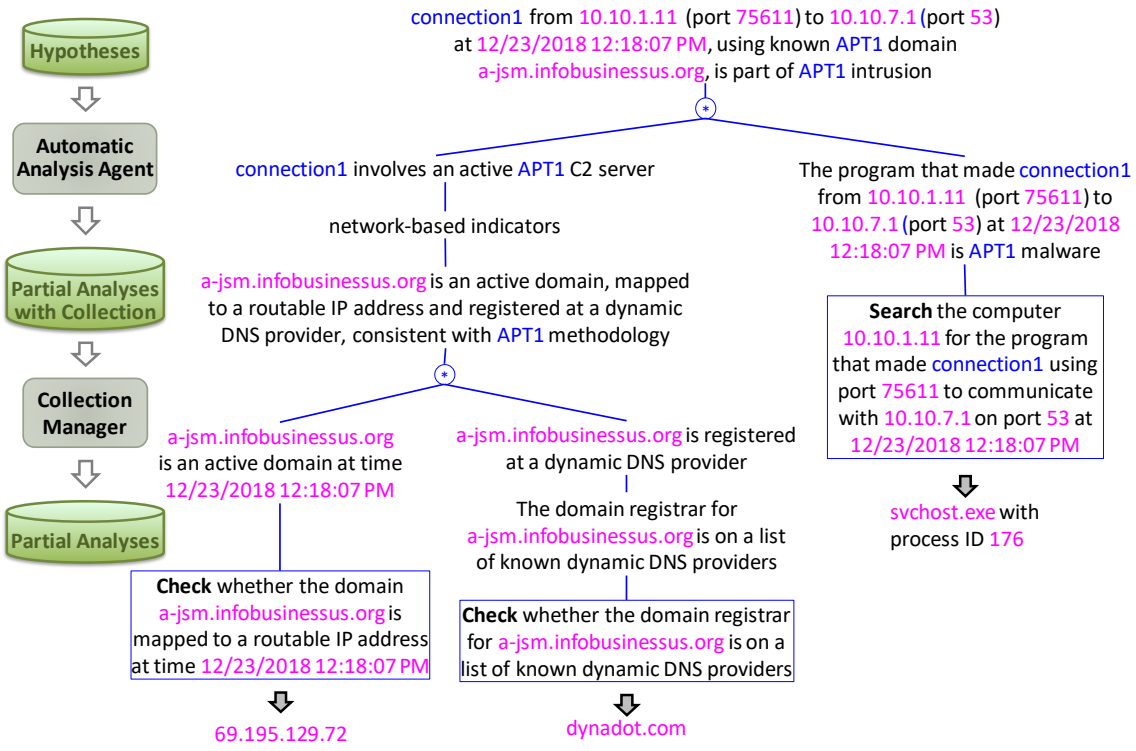

Figure 19 AURIGA example of hypothesis-driven search

The abstract searches from the bottom of Figure 19 must be turned into concrete searches for real evidence on the network. The Collection Manager is responsible for that process. First, though, the Disciple agent must be able to request the search to be performed by the Collection Manager. Let's take, for example, the left-most search from Figure 19, which is a check to determine if the domain was mapped to an active IP address at the time the BRO alert was generated. The Disciple analysis agent will create a JSON-formatted search message and send it to the Collection Manager, as shown in Figure 20. When the Collection Manager receives this message, it will call the function *CheckDomainMappedToActiveIP*, which is one of the programmed search functions

supported by the Collection Manager, mapping data elements from the search into function parameters.

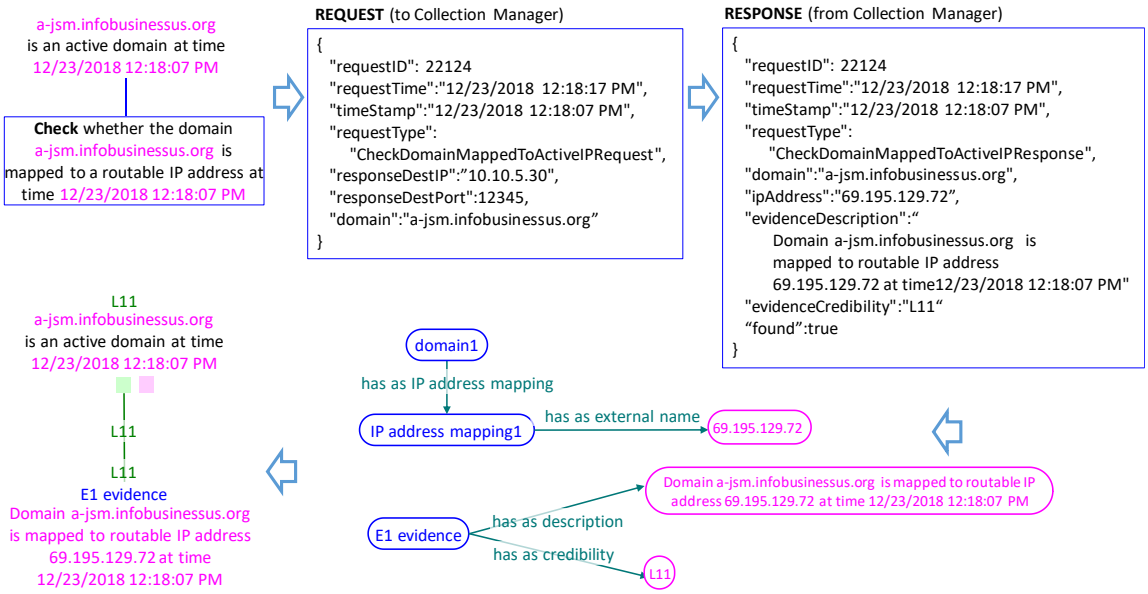

Figure 20 Search function example

When the Collection Manager completes the search, it will respond to the calling agent with a response message, which is also in the JSON format. Using learned rules, the Disciple analysis agent will convert data elements from the response message into an ontology fragment and store it in the knowledge base as evidence. An example of this is shown at the bottom of Figure 20.

In addition to simply storing the search results as an ontology fragment in the knowledge base, evidence has a **credibility** value assigned to it. For CAAPT, this is

handled using the *has as credibility* attribute with all instances of the *evidence* concept. In the example above, the credibility is L11, or certain.

In some cases, such as the leftmost two searches in Figure 19, the search satisfies all evidence collection requirements for its branch in the tree and can be decomposed no further. In other cases, such as the search for the program that generated the suspicious connection, the returned evidence will satisfy the conditions for the parent hypothesis to be further decomposed, driving further search for evidence in an autonomous fashion. This allows CAAPT to be trained to model the iterative Incident Response process outlined in Figure 4, where evidence found by a forensic analyst answers some questions only to reveal additional questions requiring investigation

Further decomposing the right side of Figure 19 after finding the process that made the connection triggering the BRO alert, we get the deductive reasoning tree in Figure 21. At the top level, the analysis requires reasoning about the features of the program that made the connection and their similarity to features we know AURIGA to have and file system artifacts created by the process.

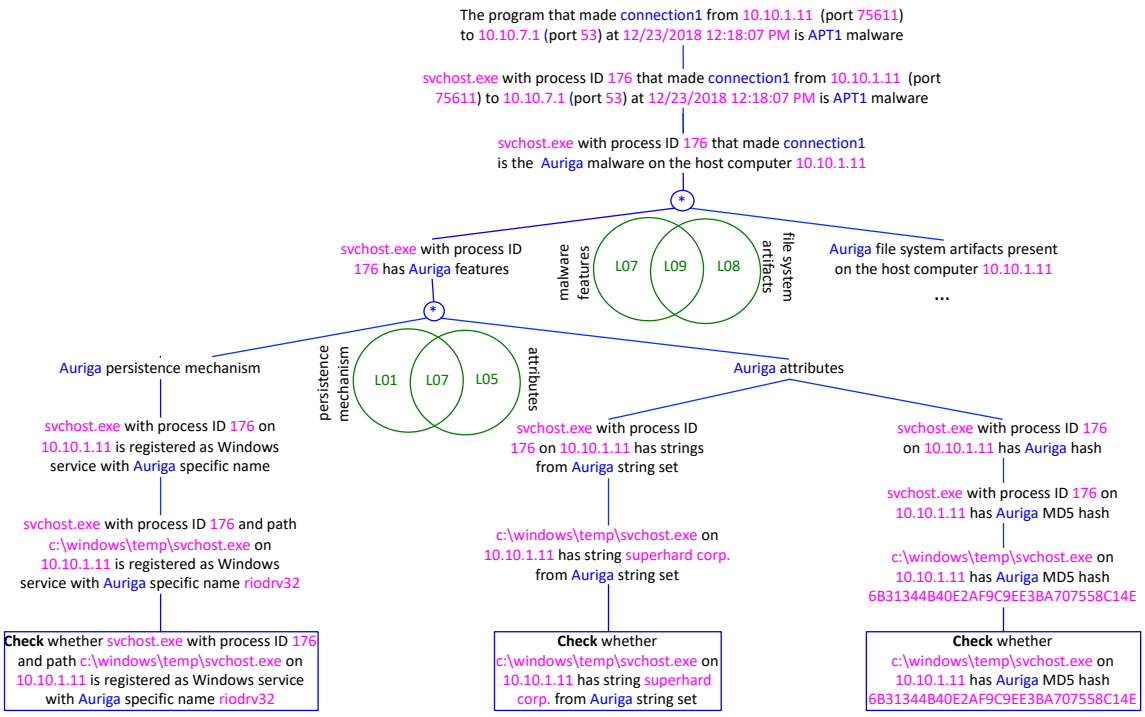

Figure 21 Search for AURIGA program features

Figure 21 shows the portion of analysis related to the persistence mechanism used and the attributes of the program including unique strings found inside the file and the MD5 hash of the file. A persistence mechanism is a way for the malware to use operating system features to survive a reboot of the computer. In this case, we have threat intelligence indicating the AURIGA malware creates a Windows service with the name *riodrv32* to start itself again after reboot. On the right side of the tree, analysis of the program executable file is required to determine if it includes any unique printable strings that can be found inside of it. The MD5 hash is also examined to see if it matches the hash of a known sample of the AURIGA malware.

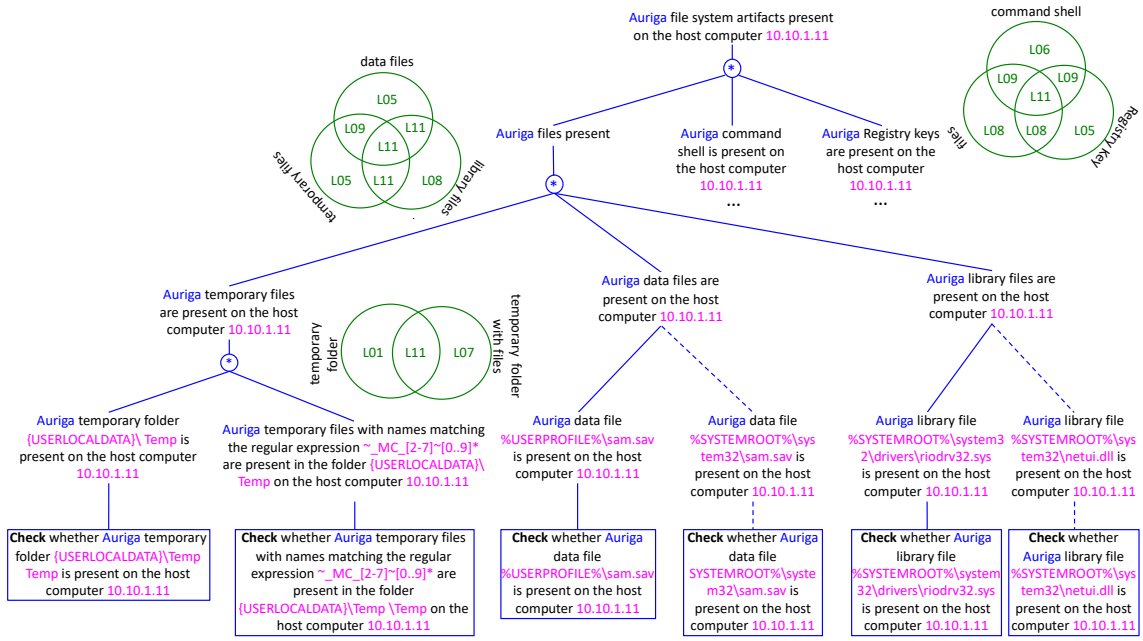

Figure 22 Search for AURIGA files

When the right side of Figure 21 is further decomposed to look for file system artifacts, it yields the decomposition tree shown in Figure 22, where the cognitive agent must look for the presence of AURIGA files, its unique command shell TTP, and Registry keys associated with the malware. Figure 22 shows the searches for AURIGA files, which I broke down into three categories: temporary files, data files, and library files. Temporary files are unique in that the file names are generated at run-time and always match the regular expression "~_MC_[2-7]~[0..9]*". Data files are used to store keystrokes and other data recorded by the malware. Library files includes the DLLs, executables, and device driver binary executable files used by the malware.

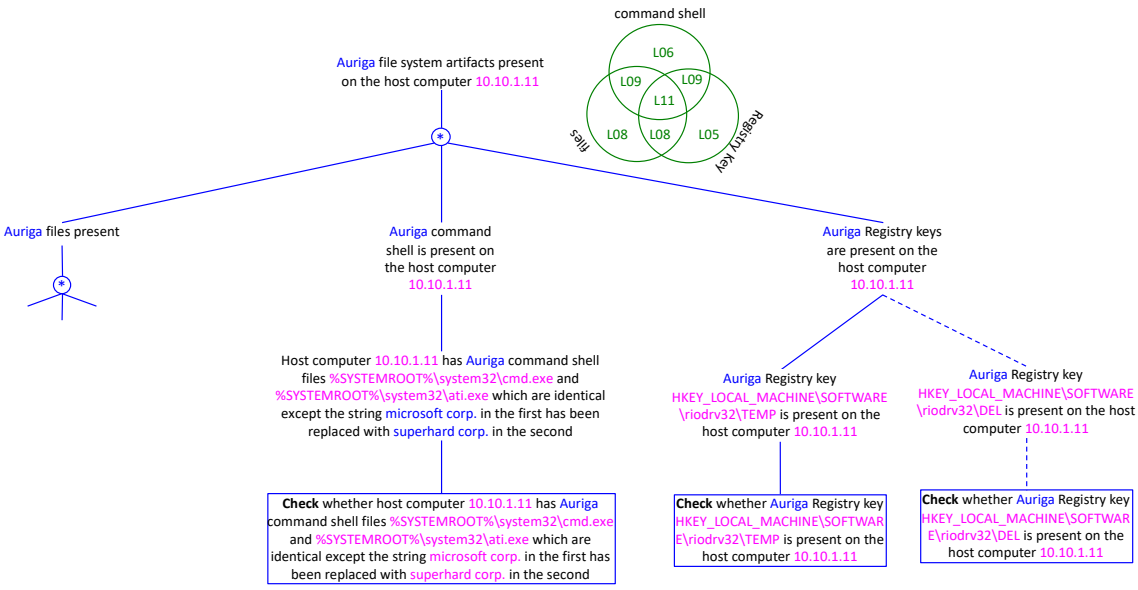

Figure 23 Search for AURIGA command shell and Registry keys

The tree shown in Figure 23 shows the decomposition of the other top-level hypotheses from Figure 22. In the middle is the search for the unique command shell TTP used by APT1. In some of their malware families, including AURIGA and BANGAT, they make a copy of the standard Windows command shell program (cmd.exe) with the path %SYSTEMROOT%\system3\ati.exe. They then replace all instances of the string "Microsoft Corp." with the string "Superhard Corp.". Figure 22 also shows the searches for AURIGA-specific Registry keys.

The massive amounts of data required to be examined by cybersecurity experts to detect threats has become a major problem as networks have become more complex and threats have been more sophisticated. The state of a network can change billions of times per second as processes on thousands of machines perform their tasks. It is infeasible to collect all possibly relevant data all the time due to prohibitively high storage costs. The

hypothesis-driven search for evidence process can drastically reduce the amount of data that must be collected to detect threats. Through the modeling process in Disciple, cognitive agents learn what data is important to subject matter experts, so only what is needed is collected.

Furthermore, because of the abstraction layer the Collection Manager provides between abstract searches generated by Disciple agents and concrete searches of real security infrastructure, CAAPT can be more easily integrated into CSOCs with vastly different security tools and infrastructure available to the analysis team. This will allow for easier adoption by transition partners.

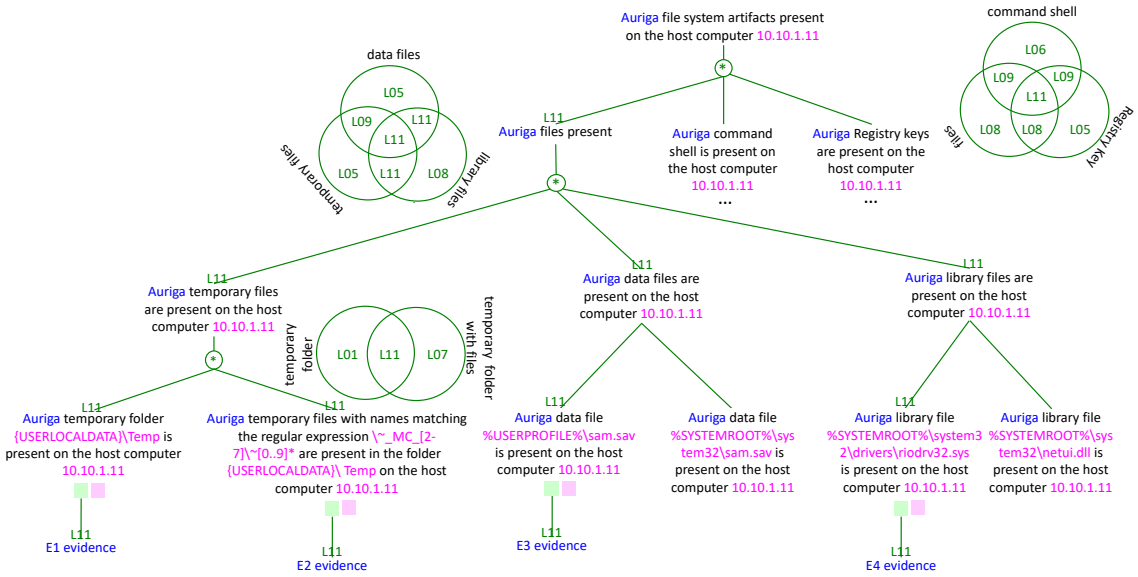

Figure 24 Automatic analysis of AURIGA files

3.2.4. Automatic Analysis of Evidence

When the hypothesis tree cannot be decomposed any further and all the available evidence has been collected the Disciple agent will begin synthesizing a conclusion using inductive (probabilistic) reasoning. This is a bottom-up approach, where favoring and disfavoring evidence for each hypothesis is evaluated to determine the probability of each leaf hypothesis. Then the probabilities of the upper-level hypotheses are determined by combining the probabilities of their sub-hypotheses. This section continues the AURIGA example and describes the automatic analysis process in depth.

Starting with analysis of the presence of AURIGA files, Figure 24 shows how the automatic analysis process works. At the leaf level of the tree, each item of evidence is evaluated based on its *credibility* (the probability that the evidence is true), and its *relevance* to the parent hypothesis (the probability of the hypothesis assuming that the evidence is true). The combination of these two values is called the *inferential force* of the evidence (the probability of the hypothesis based only of this item of evidence) and is computed as the minimum between the credibility of evidence and its relevance. In many areas of analysis, the credibility of evidence can be less than certain. For example, it can be based the reliability of a witness, their skill at interpreting the observed event, or a host of other factors. In computer forensics and threat detection, the credibility of *found evidence* is generally considered *certain* unless there is reason to believe an attacker falsifies evidence as a part of their attack methodology. APT1 was not known to leave fake forensic artifacts on victim computers or generate artifacts to trick analysts into

believing the attack was conducted by a different attack group. Therefore, the credibility of all evidence used in this work is considered to be *certain*.

All credibility, relevance, and inferential force values in CAAPT use the probability scale in Figure 7. To make the scale granular enough to allow for accurate modeling of sophisticated threats, a twelve-step probability model was chosen with assigned probability ranges. For some probability ranges, special names have been chosen (e.g. “likely” and “almost certain”) to provide more clarity in the reasoning models.

Once the inferential force of the evidence is computed for the leaf nodes it is assigned as the probability value of the parent nodes for use in computing the probability for hypotheses further up the synthesis tree. When a hypothesis has two or more child hypotheses, the value of the hypothesis is calculated using one of three operators: min, max, and *. Max, used for a parent hypothesis with alternative sub-hypotheses (arguments), takes the highest value of the inferential forces of all sub-hypotheses (arguments) and assigns it to the parent. Min, denoted by the & symbol, uses the lowest value among the probabilities of the sub-hypotheses (representing the & argument) and the relevance of the & argumentto determine the parent hypothesis’ value.

For CAAPT, the *-operator was created to allow the system to model the combinatorial force of the sub-hypotheses which are indicators of the hypothesis, which is a novel operator enabling CAAPT’s ability to analyze intrusion incidents. In the analysis of sophisticated threats, it is common for one piece of evidence to have significance on its own, but when combined with other evidence to gain substantially

more significance. It is also common for malware to leave behind multiple low-relevance artifacts that, when combined, result in high likelihood by the analyst that the threat is present. The *-operator allows a subject matter expert to accurately model this in the Disciple learning agent shell to train cognitive agents how to autonomously apply the knowledge.

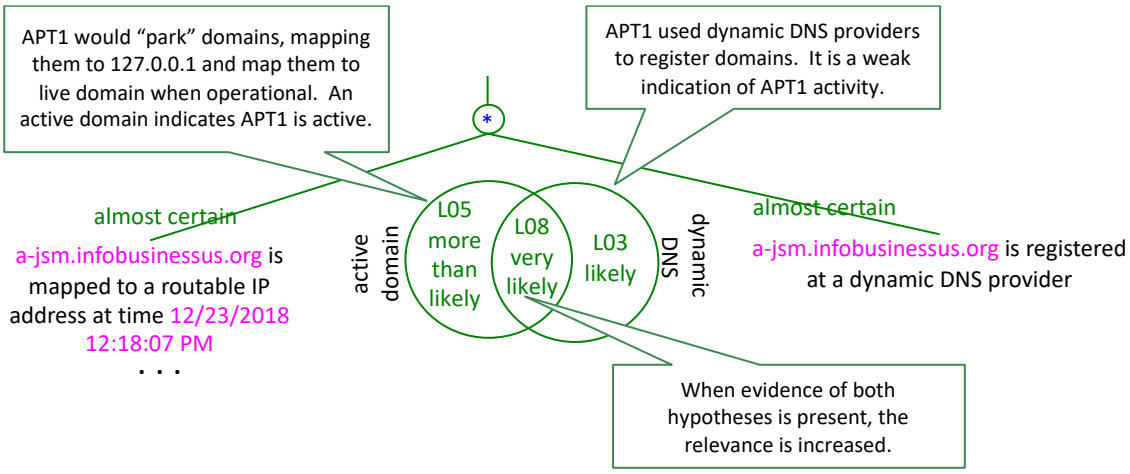

Figure 25 *-operator example

Figure 25 shows an example of the *-operator, taken from the left most branch of the inductive reasoning tree in Figure 19**.*** In this example, the system is required to determine how likely it is that an identified suspicious connection (connection1) is part of an active APT1 intrusion based on whether or not the domain is active at the time of the connection and the domain is registered by a dynamic DNS provider. APT1 was known to *park* domains by mapping them to 127.0.0.1 (localhost) or some other non-routable IP when there was no active attack in progress. Just before an attack, it would re-map the

domain to the IP address of a C2 server. While this was a good indicator of an active APT1 attack, we cannot say for certain it indicates an attack on its own. There are plausible explanations for why it would not indicate an attack on its own, such as the domain being taken over by a security researcher or law enforcement. Likewise, while APT1 was known to use dynamic DNS providers for its C2 domains, dynamic DNS providers are popular for legitimate use cases. Therefore, the relevance of dynamic DNS usage is low. However, when you combine the two indicators, the likelihood of an active APT1 intrusion goes up substantially as the likelihood of someone legitimately mapping a known APT1 domain to an active IP address using a dynamic DNS provider becomes negligible.

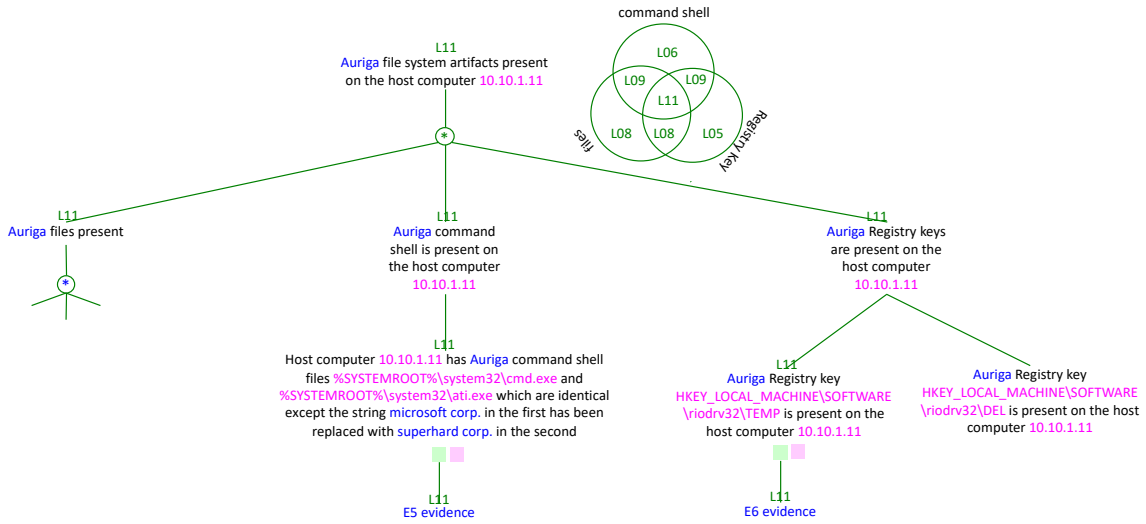

Figure 26 Automatic analysis of AURIGA command shell and Registry keys

The automatic analysis continues with Figure 26, which shows the analysis of the AURIGA command shell TTP and AURIGA Registry keys. It is common that not all of the modeled forensic artifacts will be present on the infected computer. In the case of Figure 26, only one of the Registry keys is present. However, because a *max* operator was used, the probability that the presence of Auriga registry keys is L11.

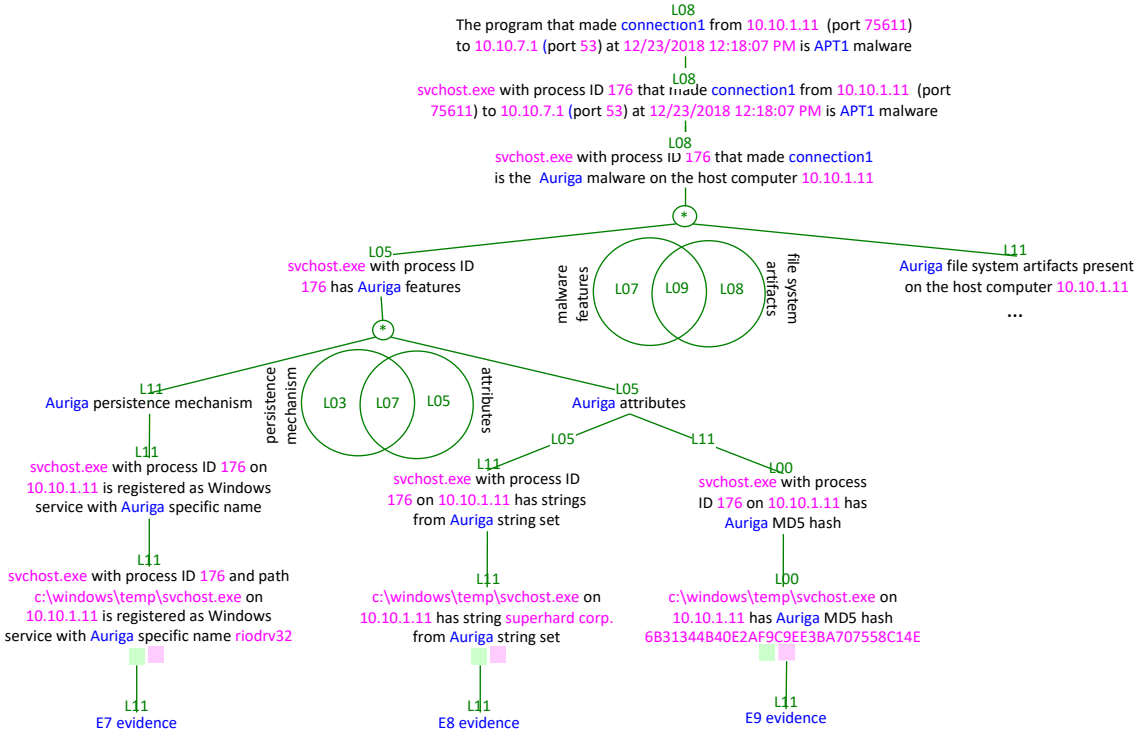

Figure 27 Automatic analysis of AURIGA program features

The analysis of AURIGA program features in Figure 27 shows another unique and powerful aspect of the theoretical model of attack detection used in CAAPT. In this case, the MD5 hash of the program that made the connection triggering the BRO alert did

not match the hash of any known samples of AURIGA. This is an example of where the attacker has made some sort of configuration change to the malware or used a different packer/obfuscator to evade detection. In Figure 27, evidence E9 is disfavoring as a result. However, we can still conclude with L08 probability that the program has AURIGA features because of the combination of the other indicators: unique strings and the persistence mechanism. Because of the modeling used, even the absence of critical indicators like MD5 hashes will not cause a failure to detect the attack.

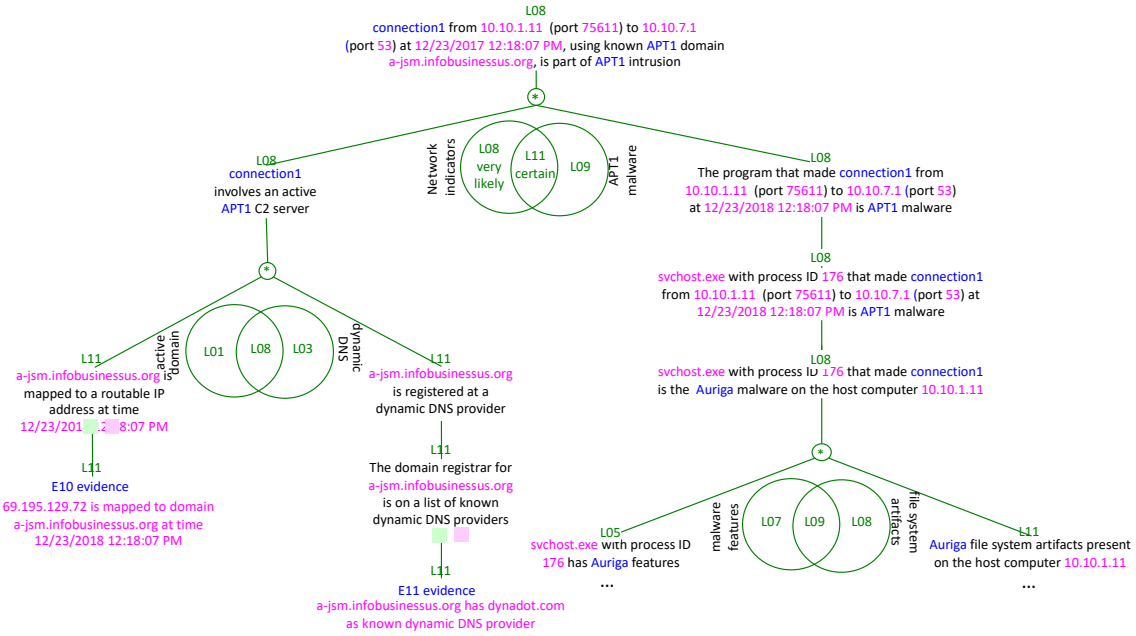

Figure 28 Top-level automatic analysis for AURIGA

At the top level, as shown in Figure 28, the modeling shows the probability that the BRO alert which triggered the process is L08. In this example the analysis shows the left and right side of the analysis have equal probability.

3.3. Integrating Cognitive Agents into a Cybersecurity Operations Center

The most significant architectural challenge in the CAAPT research is creation of a flexible design allowing the cognitive agents to integrate with a wide variety of hardware and software security sensors and controls. There are hundreds of commercial security products on the market for CSOC managers to choose from and many more open source or homegrown solutions. For CAAPT to be successful, it has to be designed for adaptability. Solving this problem required two main architectural contributions: 1) selection and integration of multiple, collaborative, search and collection agents working together to support the evidence collection requirements of APT detection, and 2) development of a Collection Manager server application for translating and optimizing abstract searches into searches executable by real collection agents. This section describes how CAAPT integrates into a CSOC and why this contribution is significant for enabling cognitive agents to work in real-world APT detection scenarios.

3.3.1. Selection of Collection Agents

The abstract searches requested by the analysis agents require evidence from multiple types of data sources available on a typical network. There are hundreds of security appliances, log source, and data store combinations in real-world networks. For abstract searches requested by the cognitive agents to be carried out, a diverse set of collection agents is required. Gartner research (Chuvakin, 2018) has determined that the most critical technologies are a network detection/collection solution, and a host detection and query solution, and a SIEM, so the selection of the agents focused on those

areas. I have chosen to use those critical technologies, as well as others, as needed, broken down into the following categories from the taxonomy in Figure 29.

Passive collectors monitor raw data sources, such as network traffic or process activity, and forward it to a datastore such as Elasticsearch or Splunk to be searched for when it is needed later. Data collected in this fashion is often needed for its historical value.

On-demand search agents listen for requests to retrieve specific artifacts from a system. In response to those requests, they collect the specified information in a forensically sound way and send it to the requestor.

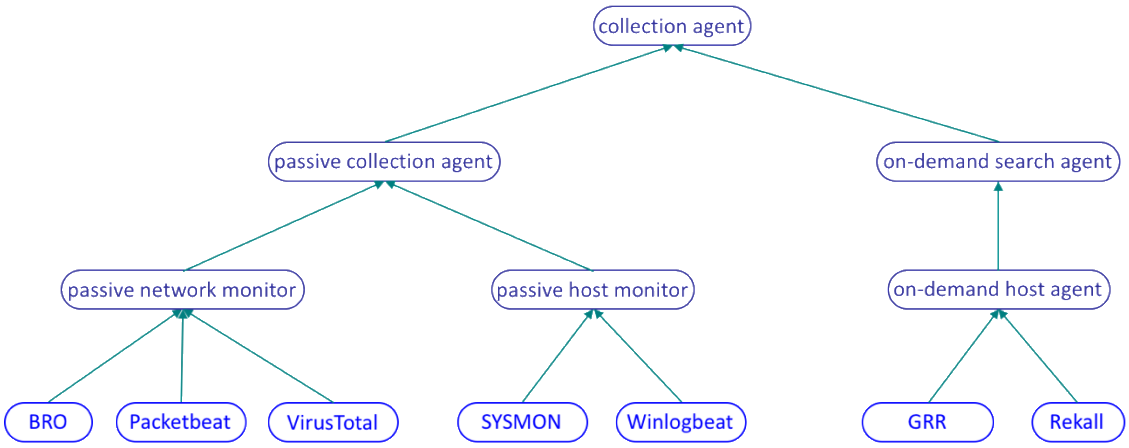

Figure 29 Collection agent taxonomy

Passive collectors include ***passive network monitors*** such as BRO (Paxson, 1999) for collecting network log data, Packetbeat (2019) and Symantec Security Analytics

which collect and store full packet data, and API-based systems such as VirusTotal which collects passive DNS and malware data. ***Passive host monitors*** include tools such as Winlogbeat (2019), which collects Windows log data and forwards it to the Elasticsearch data store and endpoint detection and response (EDR) agents like Microsoft SYSMON, CarbonBlack (2019), and Symantec EDR, which monitor and log real-time process, file access, and connection information for storage in a central database.

On-demand collection agents are primarily used for host data. ***On-demand host agents*** including Google Rapid Response (GRR) (2017), Encase Endpoint Investigator (2017), and memory forensics tools such as Volatility (2015) are examples of on-demand search agents. They are responsible for collecting and, in some cases analyzing, raw forensic artifacts from network hosts.

For CAAPT, collection agents were chosen based primarily on their ability to query and collect the types of data required for detecting sophisticated attacks. Based on the requirements for modeling detection for APT1 malware, we chose a collection of agents for netflow (network connection) data, full packet capture, DNS logs, volatile memory artifacts, Windows Registry keys and values, file-based artifacts, endpoint logs, domain controller logs, EDR logs, and passive DNS data. Next, free or open source solutions were prioritized. CAAPT must be integrated into a CSOC in order to perform its functions. The choice of open source solutions reduces barriers to integration, as there is no cost for software licenses. Lastly, I chose tools supporting a RESTful API (MuleSoft, 2016) for uniformity of integration. A substantial portion of the Collection Manager code involves use of search or collection agent APIs. Using agents with

RESTful APIs as much as possible simplified the Collection Manager code and minimizes future agent integration efforts. Table 1 below shows the collectors or agents chosen for use in the CAAPT development and test network and for initial modeling of APT1 detection.

Table 1 Collection agents for CAAPT

Data Type	Collector/Agent	Type
Netflow	BRO	Passive
Packet Capture	Packetbeat	Passive
DNS Logs	BRO	Passive
Firewall/IDS Logs	BRO	Passive
Volatile Memory	Rekall and GRR	On-demand search
Registry Keys	Google Rapid Response (GRR)	On-demand search
File-based Artifacts	GRR	On-demand search
Host Logs	Winlogbeat	Passive
Domain Logs	Winlogbeat	Passive
EDR Logs	SYSMON and Winlogbeat	Passive
Passive DNS	VirusTotal	On-demand search

On top of being an efficient network-based intrusion detection system, BRO is also a passive network collection agent, monitoring network traffic and generating logs for different events including digital certificates used for SSL/TLS connections, HTTP connection strings, DNS requests and the IP addresses resolved, and netflow data. It also has a flexible packet analysis programming language and interface, allowing for additional detection or logging features to be added, as needed.

The developers of Elasticsearch have created a collection of passive log collection agents, called “Beats” (2019). Beats are designed to collect specific types of log information and send the log entries to Elasticsearch as JSON documents. For CAAPT,

Packetbeat is used for full packet data, Filebeat is used for collecting logs stored in flat files (such as those generated by BRO), and Winlogbeat enables collection of Windows Event logs, including those created by SYSMON. All logs collected by Beats are sent to our Elasticsearch data store.

SYSMON, a passive host monitor, is a system service and device driver designed to log system change activity to a Windows Event Log file. It logs detailed information about process execution, network connections, and changes to the file system, including the Registry. SYSMON log information provides the basic EDR functionality required to understand the behavior of malware on a host computer. All SYSMON logs are forwarded to the Elasticsearch data store using Winlogbeat and can be searched using Elasticsearch's RESTful API.

3.3.2. Collection Agent Architecture

Figure 30 shows a high-level overview of CAAPT's passive collector architecture. A BRO IDS box listens in promiscuous mode on the network segment between the border router of the network and the outermost internal router. This allows the BRO server to see all network traffic passing through a network's Internet egress point. BRO generates several types of logs, including alerts of suspicious connections from threat intelligence, storing them in flat files on the server. When full packet capture capability is required, Packetbeat can be run on the BRO server, taking advantage of its visibility on the network to capture all network data passing through the Internet egress point. Microsoft SYSMON runs on every Windows endpoint on a network, logging real-

time process activity to the Windows Event Log. All logs stored in Elasticsearch can be queried using its RESTful API.

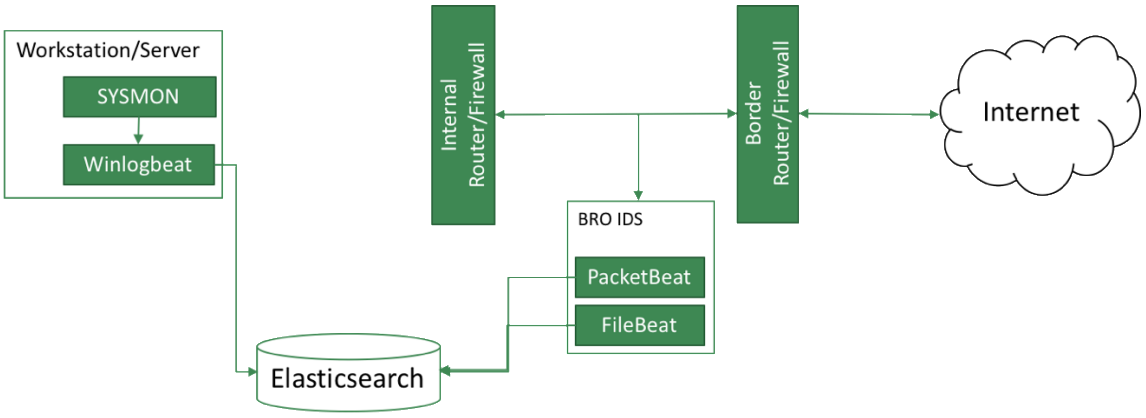

Figure 30 CAAPT passive collection architecture

Integration with on-demand search agents is simpler. The search agent runs as a service on one or more computers on the network. It can be queried using a search interface using either a synchronous or asynchronous call model. An example of the on-demand search agent architecture is described in more detail in **Section 3.3.3.**

3.3.3. The CAAPT Collection Manager

The Collection Manager is the main integration point between the agents and CSOC infrastructure. The analysis agents know what information is needed to expand their analyses, but the search requests are in abstract form. They are not tied to specific data sources. The primary function of the Collection Manager is translating high-level (abstract) search instructions into specific API calls to host and network agents, determining which such agent to send the search request to on behalf of the analysis

agents, and wrapping calls to specific search agents with a JSON API. Results returned from a specific search agent to the Collection Manager are then converted into evidence and added to the knowledge bases of the analysis agents.

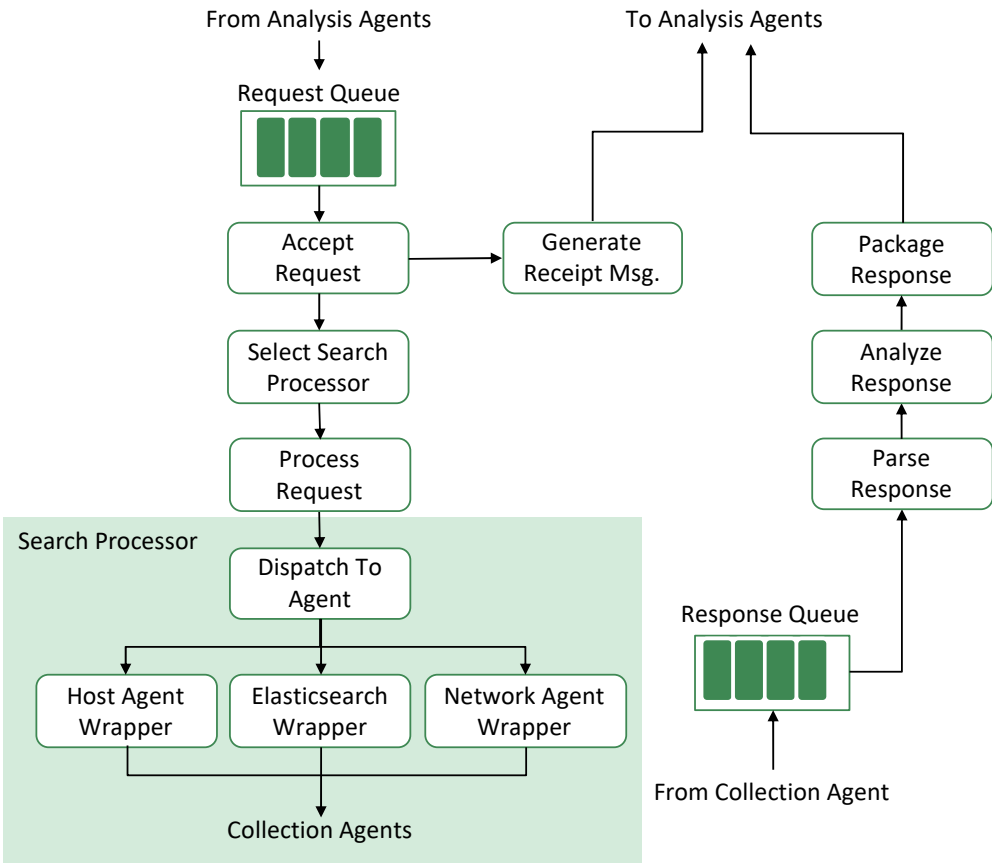

Figure 31 Collection Manager process

Figure 31 is an overview of the Collection Manger process. When the analysis agents analyze competing hypotheses, the searches generated by the hypothesis-driven search process (such as the ones illustrated in Figure 19) are sent to the Collection

Manager and added to the request queue. Requests are then dispatched for processing and a receipt message is sent back to the caller. The receipt includes the *requestID* and the IP address and port the caller will listen on for the search response.

The abstract searches requested by analysis agents require evidence from multiple types of data sources available to CSOC security infrastructure. There are hundreds of security appliance, log source, and data store combinations in real-world networks. In order for the analysis agents to integrate with real networks, the Collection Manager uses a plugin architecture with search agent wrappers, allowing it to easily translate abstract search requests into requests for information from real data stores.

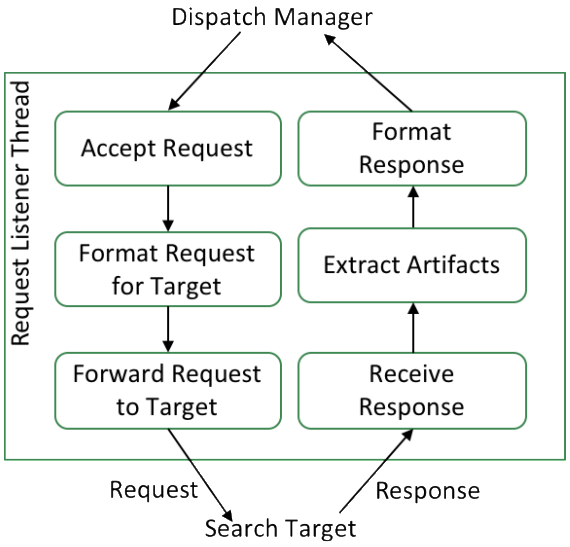

Figure 32 Synchronous wrapper flow

Depending on the amount of time required to collect the information, requests to an on-demand search agent can be either synchronous or asynchronous. From the

perspective of analysis agents, all requests to the Collection Manager are asynchronous, but internally, the Collection Manager supports both a synchronous and asynchronous call model. Figure 32 is an overview of the synchronous call model flow. The Dispatch Manager thread dequeues an abstract search request from the queue, formats and prepares a concrete search for a specific search or collection agent and forwards the request to the search or collection agent. It then waits on the TCP connection for a synchronous response. When it is received, the response is parsed to extract digital artifacts, formatted as evidence, and sent the response to back to the caller. The entire call flow happens in a single thread.

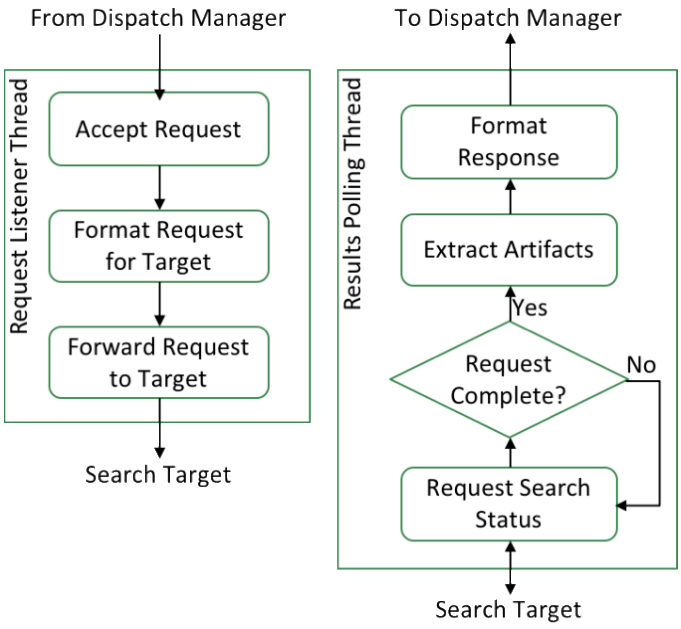

Figure 33 Asynchronous wrapper call flow

Figure 33 shows an overview of the asynchronous wrapper call model. In contrast to the synchronous call model, the call happens in two threads. In one thread, the abstract search request is received from the Dispatch Manager. The request is parsed, and a concrete search request is prepared using relevant data from the abstract request. The concrete search request is then sent to the intended search target. A second thread is responsible for polling the search target on an interval for the response. When it is available, the response data is parsed, artifacts are extracted, and a response to the analysis agent is prepared and sent to the Dispatch Manager.

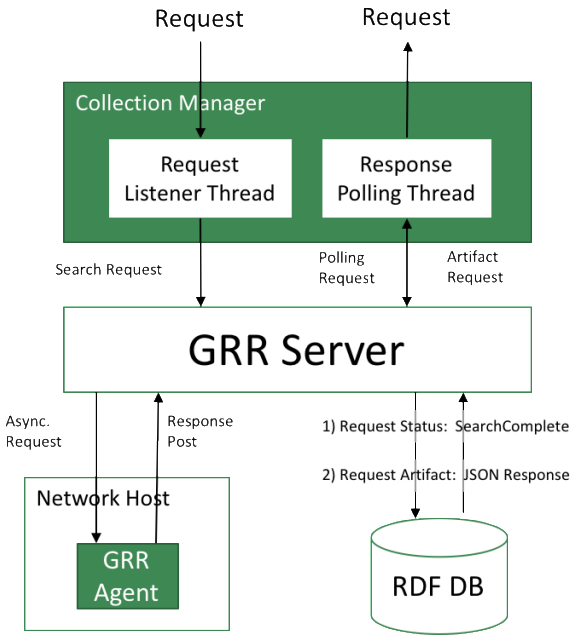

Figure 34 GRR asynchronous call flow example

GRR is the primary search agent where the asynchronous wrapper call model is used in the Collection Manager. As shown in Figure 34, GRR Server software runs on a

server in the CSOC environment. The GRR Agent is installed on all endpoints on the network. When GRR Server receives a request for a host artifact, the request goes into a queue in its database. GRR Agents poll GRR Server on an interval, looking for new requests. When one is found, it retrieves the request and executes it. When the request is complete, the response is sent to GRR Server and stored in its database. The caller must poll GRR Server to determine when the request is complete. The caller can then make a separate, synchronous call to retrieve the artifact from GRR Server's database. Using this model, we can query any host artifact supported by GRR.

To illustrate the operation of the Collection Manger, let us consider the search from the right side of Figure 19. This will lead to the invocation of the ***GetProgramByNetworkConnection*** search function. An example of how this search is performed is shown in Figure 35.

The request to the collection is in the form of a JSON document containing the name of the search to be performed and the input parameters required to carry out the search function. Parameters are extracted and the ***GetProgramByNetworkConnection*** function is called. To carry out this search, the Collection Manager must first connect to the Elasticsearch using its RESTful API. It then searches the Elasticsearch datastore for the SYSMON log from the computer with the IP address in the *sourceIP* field matching the search parameters (*sourceIP, destinationIP, sourcePort, destinationPort,* and *timestamp*). After receiving and processing the response from Elasticsearch, the Collection Manager generates the response JSON document shown in the bottom half of Figure 35, containing the name of the process, its process ID, and its full path on the file

REQUEST (to Collection Manager)


```
{
   "requestID": 0,
   "requestTime":"12/13/2017 7:46:56 PM",
   "requestType":"GetProgramByNetworkConnectionRequest",
   "responseDestIP":" 10.10.5.30 ",
   "responseDestPort":12345,
   "destinationIP":"69.195.129.70",
   "destinationPort":53,
   "sourceIP":"10.10.1.11",
   "sourcePort":11234,
   "timestamp”:”12/23/2017 7:46:56 PM“
}
```


Search Function: GetProgramByNetworkConnection

Input Parameters…	Output Parameters…

Search Process:

1. Connect to Elasticsearch via RESTful API
2. Search for the SYSMON log from computer with IP address *sourceIP* matching the search parameters
3. If the record exists:
    1. Add the program name, process ID, and full path to the output parameters
4. The Collection Manager is alerted that the search request has been completed.


```
{
        "requestID": 22124,
        "requestTime":" 12/13/2017 7:46:56 PM ",
        "requestType":"GetProgramByNetworkConnectionResponse",
        "requesterSourceIP":"127.0.0.1",
        "responseDestIP":"127.0.0.1",
        "connectionName":"connection1",
        "destinationIP":"69.195.129.70",
        "destinationPort":53,
        "sourceIP":"10.10.1.11",
        "sourcePort":11234,
        "timeStamp":"12/23/2017 7:46:56 PM“,
        "processID":176,
        "programName":”svchost.exe",
        "programPath":"c:\\windows\\temp\\svchost.exe"
}
```


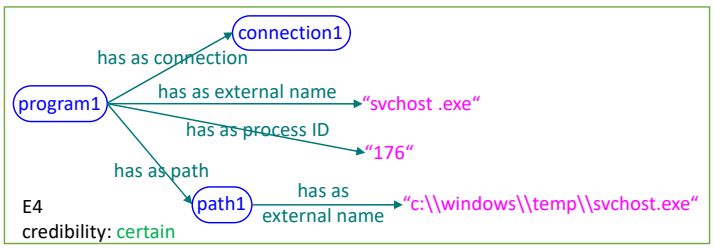

Figure 35 Search example

system. This JSON document is sent back to the calling analysis agent, which converts it into evidence with credibility *certain*, and adds it to the knowledge base. The ontology fragment in the bottom of Figure 35 is the representation of evidence found by the ***GetProgramByNetworkConnection*** function.

The selection of search and collection agents for CAAPT and the Collection Manager design offer key contributions to the state of the art for integration of cognitive agents into real-world CSOC environments:

(1) The agents selected allow for effective detection of sophisticated threats with minimal network visibility and cost to the CSOC owner.

(2) The Collection Manager provides seamless translation of abstract search requests generated by cognitive agents into concrete searches against real search agents and automatic conversion of digital artifacts and search results into evidence which can be used by cognitive agents to perform complex reasoning and generate conclusions about the probability of multiple hypotheses.

(3) A centralized collection architecture will allow for caching of search results, reusing results from identical searches and reducing the time required to execute duplicate searches.

(4) Centralizing integration between the Disciple agents and the CSOC security infrastructure will provide for other means of optimization, such as automatically scheduling large volumes of searches to reduce the

likelihood of overconsumption of network bandwidth of computing resources.

3.4. Automatic Generation and Use of Incident Investigation Playbooks

One of the key contributions of this research is the automatic generation and use of incident investigation playbooks. Not only is CAAPT capable of learning from an expert cybersecurity analyst how to conduct autonomous analyses to detect sophisticated threats, but it is also able to help analysts detect new threats by suggesting analysis and collection *playbooks* as it encounters new threats.

Sophisticated attackers' malware evolves slowly over time. Small configuration changes are made during use of a malware variant to make IOC-based detection difficult, and larger changes are made as the malware development teams add or remove features from the malware to make it better. CAAPT can detect malware when configuration changes are made because sophisticated attackers don't change everything from one attack campaign to the next.

The new versions often contain some features of the old malware, adding new features on top of them. From the perspective of a CSOC analyst, the new malware version still creates digital evidence consistent with old versions, making modeling detection for the new version much easier. It also means CAAPT agents can use modeling of known malware from an attack group to suggest what analysis to perform and which evidence to collect to detect new malware version. These suggested analyses are very similar to playbooks created in security orchestration products on the market. However, CAAPT advances the state of the art in this area because the playbooks are

generated by cognitive agents by applying rules learned from detection of known malware. This means that as the system grows to learn to detect tens or hundreds of threats, analysts will not have to search through a huge library of reasoning trees to find one that works. CAAPT will suggest a small set of playbooks likely to apply to the given detection task, drastically increasing efficiency of CSOC operations.

As discussed previously, APT1 malware evolved over several years. In this research, I have studied the evolution of APT1 malware starting with Auriga. Based on analysis from contagiodump.blogspot.com (Mila, 2013) and my analysis of malware clusters, it appears the malware lineage progressed as follows:

- Auriga is an early malware program used during the *gain foothold phase* of their attacker lifecycle.
- Bangat was then developed, removing some features of the Auriga code and reducing the footprint of digital evidence created during an attack.
- Seasalt was then created, containing shared features of Auriga and Bangat, but making the network communication cleartext, which enabled analysts to detect it using its HTTP user-agent and GET strings.
- Kurton followed Seasalt, containing shared features of Auriga, Bangat, and Seasalt. The ability to view the HTTP GET string was removed.

Figure 36 shows an example of the CAAPTs ability to use rules learned from the analysis of a malware program to suggest playbooks for the analysis of a descendent in its evolution. *Step 1* shows a high-level overview of the process of learning to detect the Auriga malware. When Auriga was analyzed for the first time, CAAPT had not been

trained to handle the security alert from the Trigger Agent. The alert was presented to a CSOC analyst (me), who along with the CAAPT research team modeled the analysis and evidence collection in Disciple-EBR. When modeling was complete, Disciple-EBR learned rules for automatic analysis and stored them in the knowledge base. When Auriga was encountered in the future (*Step 2*), CAAPT applied the learned rules to detect the malware.

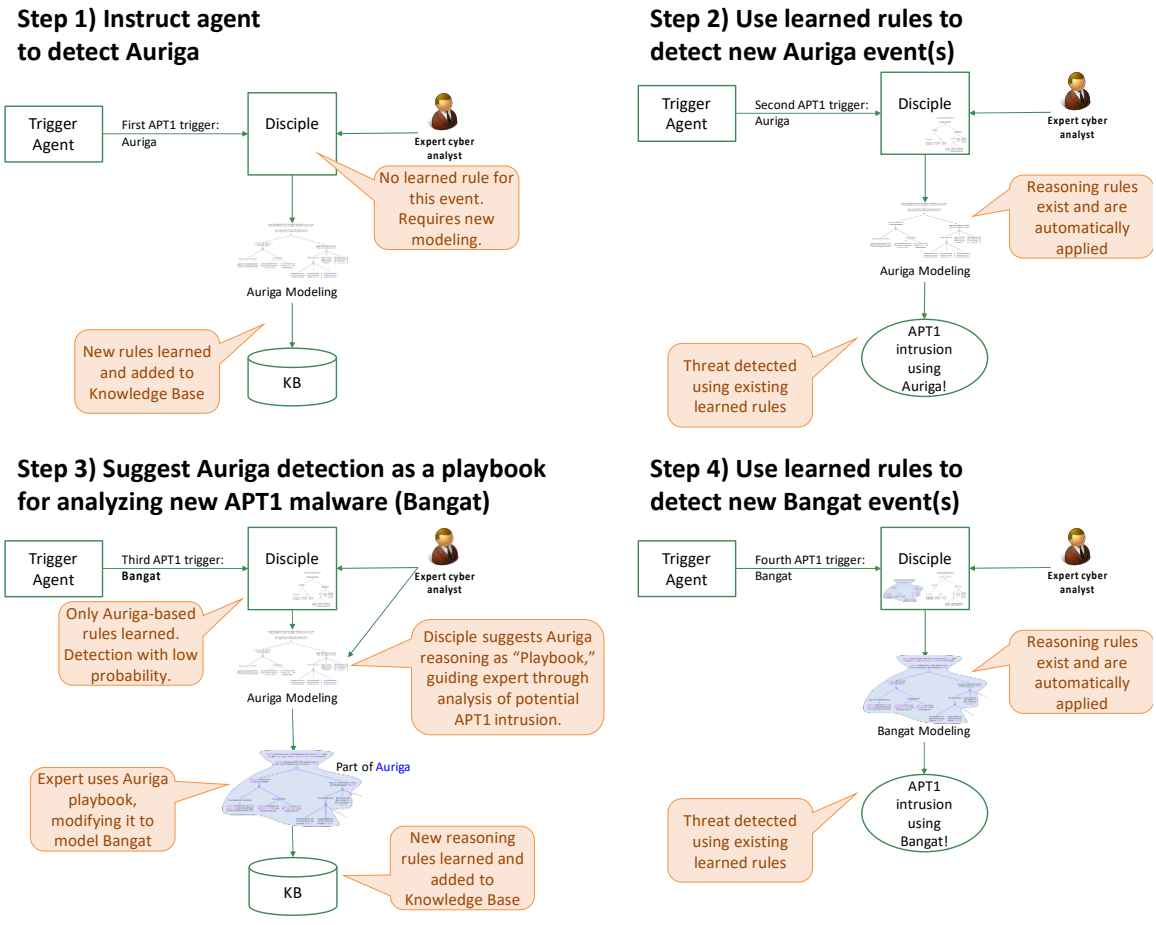

Figure 36 Example of automatic generation of detection playbooks

Later, when Bangat was analyzed for the first time, CAAPT was not able to detect it with high probability, causing it to send the alert to an analyst for modeling. Because the rules learned from the analysis of Auriga also applied to the security alert from BRO, Disciple suggested Auriga's analysis trees as a possible solution. Since the analyses are modeled using natural language and are easily understood by human analysts, the analyst was able to use the suggested analysis trees to conduct the manual analysis required to understand the new threat and model it in Disciple for use in detecting Bangat. The new analysis and learned rules for Bangat were stored in the knowledge base for future use.

When Bangat was encountered again, CAAPT used what it had learned to autonomously detect the malware. As the malware continued to evolve from Bangat to Seasalt to Kurton, the same method allowed CAAPT agents to guide me through the detection process and for me, in turn, to train the CAAPT cognitive agents to detect the new malware.

By taking advantage of the mixed-imitative reasoning and learning in Disciple, I can not only train cognitive agents to detect sophisticated threats but can also leverage previously learned knowledge to guide me through the analytical process by using the generated reasoning trees as detection and analysis playbooks. Because Disciple suggests only applicable reasoning trees for use in new scenarios, it drastically speeds up the manual analysis and modeling required to detect new threats.

3.5. Development and Test Network Environment

The CAAPT project required a simulated network environment to develop and test the cognitive agents. The network was not required to be large in scope but needed

to contain a representative architecture and selection of computers similar to a real network. Figure 37 shows the layout of the simulated network environment used for development and testing of CAAPT.

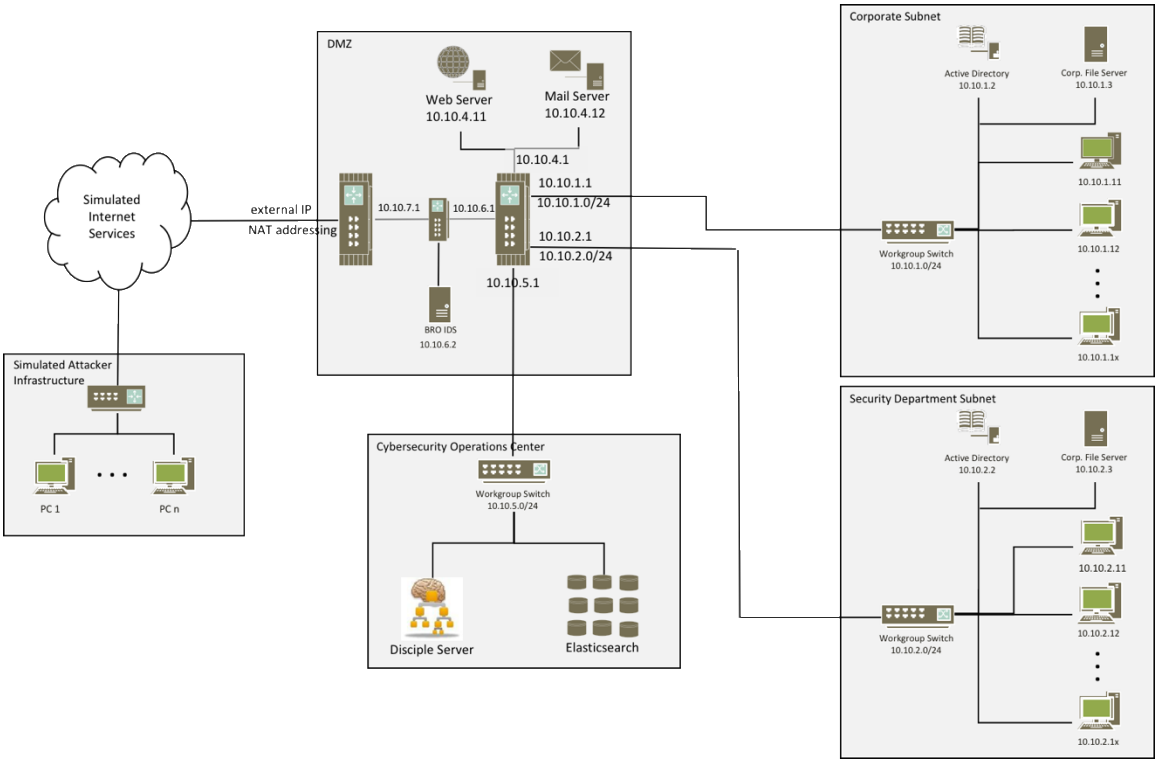

Figure 37 Logical network topology for test environment

The cognitive agents we developed were required to represent and understand real-world networks and be able to identify real threats on them. It was infeasible for development and testing of the cognitive agents to be done on a real network because the agents needed to be tested using real APT1 malware or simulated attack artifacts. This represented myriad security concerns including:

- While APT1 malware is widely known to be manually operated and does not spread from computer to computer in an automated fashion and it is unlikely to represent a significant risk, use of real malware violates the security policy of most real-world networks.
- APT1 malware is configured to call out to command-and-control (C2) servers. Those servers are believed to have been abandoned by the APT1 threat group, but it is unknown if security researchers or law enforcement continue to surveil those servers. Connections to the domains and IP addresses may trigger security alerts or unintended action by security researchers or law enforcement.
- Simulated attack artifacts, including files, registry keys, or network connections, may trigger the same types of security events or unintended security researcher or law enforcement response.

For this research, I have created a small-scale network, in a virtual environment, representative of a small corporate network, for these purposes. This virtual network contains all of the relevant network services, including workstations, domain controllers, and network security appliances. The key components of the development and test network are:

- Virtualization Platform used to host the virtual environment.
- Network Design, including networking infrastructure, operating systems and enterprise infrastructure.
- Simulated CSOC, which contains a security alert and data repository and the CAAPT cognitive assistants.

- Simulated Internet Services, allowing realistic network traffic to occur without providing malware access to the Internet.
- Simulated Attacker Network, which was used during testing and evaluation of the system.

Each of these subsystems will be discussed in detail in this section.

3.5.1. Virtualization Platform

Modern virtualization software is capable of simulating not just single workstations, but large-scale network environments for development, testing, and large simulated exercises. They provide a safe, isolated environment for testing software such as CAAPT. One of the most common software platforms for this type of testing is VMWare's vSphere environment. Depending on the hardware it is running on, it can be used to simulate dozens, hundreds, or thousands of computers in complex network environments. Through the use of "DevOps" (development operations) technology built into the vSphere tool set, it is also capable of standing up new, clean versions of an environment quickly, so the network environment can be reset repeatedly during a development and testing cycle.

Many of the key features of vSphere – virtualized computers, virtual networking and routing, etc. – can be done in VMWare Workstation, which is a desktop virtualization product. It is a mature product used by software engineers for testing and by malware analysts for examining malicious software. However, since it is designed to run on workstations or developer laptops, it cannot be used to simulate a large network environment necessary for full-scale development and testing of the system.

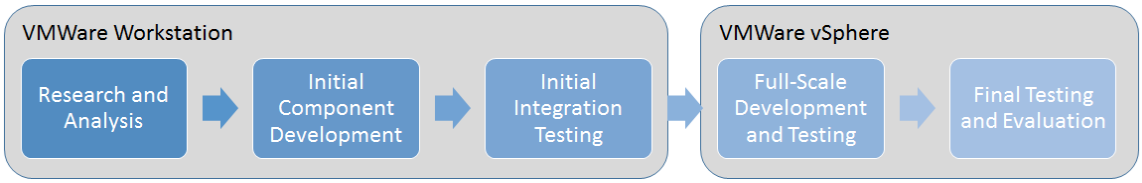

Figure 38 Virtualization platform usage in development and testing

To bridge the gap between the two needs, I combined usage of these virtualization tools throughout the development and testing process, as shown in Figure 38. For the early phases of the research project – Research and Analysis, Initial Component Development, and Initial Integration Testing – VMWare Workstation was appropriate because it could be used on the smaller developer workstations used for this early work:

- *Research and Analysis* – small-scale evaluation of network traffic, malware, or individual collection agents prior to development or selection of system components.
- *Initial Component Development* – Creation of new software components, such as the Hypothesis Generation Agent or host-based collection agents, was created on developer workstations and laptops and tested in a single virtual machine in VMWare Workstation.
- *Initial Integration Testing* – VMWare Workstation was used to test component integration on a smaller scale, prior to full-scale development.

Once development of most of the individual components was completed, I proceeded to Full-Scale Development and Testing of the system, followed by Final Testing and Evaluation. For this work, the CAAPT team needed a network similar to the

one shown in Figure 37, which required a larger server running VMWare vSphere to simulate.

By leveraging the different VMWare packages in this way, I was able to quickly start initial analysis, development, and testing prior to purchase of a server. The virtual machines are compatible across the different tools, so work done using VMWare Workstation was easily ported to the larger VMWare vSphere environment when it was ready.

3.5.2. Network Design

The development and test network was created inside of a VMware vSphere 6.5 server and its architecture is similar to Figure 37. In total, 18 virtual machines were created for the purposes of simulating a simple corporate network, implementing the collection agents, and testing the system:

- Two virtual machines (VMs) were used as virtual router appliances. This was done using Linux's built-in routing features. One of the routers is the gateway router, responsible for routing network traffic between the internal network and the Internet. The other is an internal router, responsible for routing traffic between the different subnets in the development and test network.
- GRR was installed on a standalone Linux server and was used to manage the GRR agents installed on Windows workstations.
- Elasticsearch was installed in its own virtual machine, along with the Kibana front end. Kibana was primarily used for manual analysis during

development and testing of the system. It was also an invaluable resource in debugging.

- One VM was used as a DNS server so that we could spoof DNS resolutions of APT1 domains. This was necessary because the university network our research server was connected to also has the ability to detect APT1 threats and I did not want to trigger security alerts in the university network.
- I used two VMs for BRO. One was placed behind the gateway router appliance so it could see all traffic exiting the test network. One additional BRO server was used on the 10.10.4.0/24 network so it could monitor traffic going to and from the DNS server. This was required because spoofed DNS queries would not exit the test server, preventing the original BRO server from seeing the traffic.
- The Collection Manager and Alert Generation Agent software required one VM. Both server software programs ran on a Windows Server 2016.
- The Disciple agents ran on one virtual machine.
- Nine VMs were configured as corporate workstations and were used to simulate intrusions supporting the testing use cases described below.

While the average corporate network is more complex than what was simulated in the development and test network, I created a network architecture that captures the critical issues involved in simulating, monitoring, and detecting APT intrusions in the early stages. Additional issues arise as APT intrusions progress and the attackers gain

administrator privileges on the network, move laterally, and complete their mission. The development and test network, along with the selected set of collection agents may not be sufficient to detect these later stages of the attacker lifecycle. However, with additional research, the system could be expanded to include the features required to detect the later stages of the attack.

4. EXPERIMENTATION AND TEST RESULTS

This chapter expands upon information previously published in peer reviewed conference proceedings (Tecuci et al., 2019a; 2019b).

Testing and validation of a system like CAAPT is challenging due to a lack of standardized data for use in comparing it against other systems or approaches. It is also challenging due to a lack of like systems to compare CAAPT to. It is a novel approach to the problem of APT detection and autonomous analysis of security incidents. As such, the only reasonable approach to compare CAAPT to is manual analysis by an expert. As such, a series of experiments was planned and executed to demonstrate the value of the contributions of this research by testing the following claims:

1. *Ability to automatically detect the training malware*: Once trained to detect a malware program, CAAPT is able to detect the same malware program when it is used again on the network. This creates a baseline for the evaluation.
2. *Ability to detect variants of the training malware*: Once CAAPT has been trained to detect a malware program, it is able to detect variants of the malware with different configurations (i.e. it will create a different set of forensic evidence on the network).
3. *Some ability to detect evolved malware*: Once it is trained to detect one malware family from an APT group (e.g. Auriga and Bangat of APT1), it may be able to detect an intrusion by a new malware family from the same APT group.

4. *Limited incremental training needed to detect a new malware from the same group:* Once CAAPT is trained to detect one malware family from an APT group, only incremental training is required to train it to detect a later malware family from the same APT group (e.g., limited incremental training needed to detect the Bangat family, after it was trained to detect the Auriga family).
5. *Efficient and high-quality analysis*: CAAPT can rapidly detect APT1 intrusions through a rigorous and transparent analysis, as judged by the training expert.

The first three objectives show CAAPT's ability to detect malware from the same APT group. The fourth shows efficiency in training, allowing CSOCs to be more agile in responding to new threats. The fifth shows CAAPT's ability to increase efficiency of CSOC operations. The following sections will detail the testing approach and the experimental results that justify the above claims.

5.1. Use of APT1 for Experimentation

This research used APT activity as a case study for orchestration and automation of cybersecurity incident response because APT groups follow a rigorous attack methodology and, as they are likely state-sponsored and part of a large organization, their malware and attack techniques change slowly over time compared to criminal or hacktivism attacks, allowing analysts time to adapt to the changes and put detection mechanisms in place. The evolution of an APT group's methodology and malware is an ideal case study for this research as it allows, during modeling and testing, to show the system's ability to predict changes in attack behavior and cope with a changing threat.

APT1 was chosen specifically as the attack group for this study primarily because of the abundance of freely available information about it. Freely available intelligence regarding APT groups, most of which is either classified or only offered by paid commercial subscription services, is rare. APT1 was outed publicly in 2013. Of all the known APT groups, it is the group with the largest amount of publicly available intelligence, including IOCs, malware samples, and details of how the group operated.

Included in the detailed knowledge of APT1 activity are details about the evolution of APT1 malware, which was used in demonstrating CAAPT's ability to adapt to changing APT methodology. Reporting on APT1 shows an evolution of malware used by them over the eight years they were known to operate, including the malware evolution chain used for CAAPT, which started with the Auriga implant, and then evolved over time to Bangat, Seasalt, and Kurton.

The biggest challenge with using APT1 in this study is that the group is either no longer active or is not using the malware and techniques that were made public in 2013. While APT1 malware was run in VMs to simulate attacks, and threat intelligence was used in other cases to simulate attacks, it is not possible to test the learning and reasoning models against real attacks by APT1. So, while we can test the system against APT1 technology and processes, I could not test it against the people who used to conduct the attacks.

APT1's dormancy also led to difficulties with CAAPT's false positive hypotheses. False positive scenarios related to APT detection, similar to what has been modeled in CAAPT, have to do with whether or not the C2 server is active or not. Since

APT1 has not been active for several years, all of their C2 domains have been dormant. While I could simulate aspects of an attack using data sources within the development and test network, the data sources used in the false positive hypotheses are provided by external providers, such as VirusTotal and public Whois servers, and are outside of my control. As a result, the false positive hypotheses had high probability even in simulated attack scenarios.

Lastly, simulating APT attacks was challenging because simulated connections to APT1 domains triggered security alerts in university security infrastructure, such as the FireEye network IDS appliance in use, even though the domains are dormant. Prior to implementing DNS spoofing inside of the development and test network, I accidentally triggered security alerts, resulting in access to the server being shutoff temporarily by the university's security team. This was overcome via DNS spoofing, which I implemented in the development and test network prior to final testing of the system.

For most of the experiments, attacks were simulated by planting forensic artifacts on Windows hosts consistent with those generated by real APT1 malware. Available threat intelligence provides very detailed information on the files, unique strings, Registry keys, persistence mechanisms, and network indicators created during an attack. When that information was unavailable or insufficient, I used both static and dynamic techniques to reverse engineer the malware binaries to identify forensic artifacts created by them. Using this information, I manually created attack scenarios recreating the patterns of indicators created by each malware program. Network-based artifacts, such as DNS requests, network connections, and HTTP User Agent strings, were created using

tools such as *nslookup* or the *User Agent Switcher* extension for the Chrome web browser. Attack models were then tested in isolated VMs by infecting hosts with APT1 malware and running CAAPT against them.

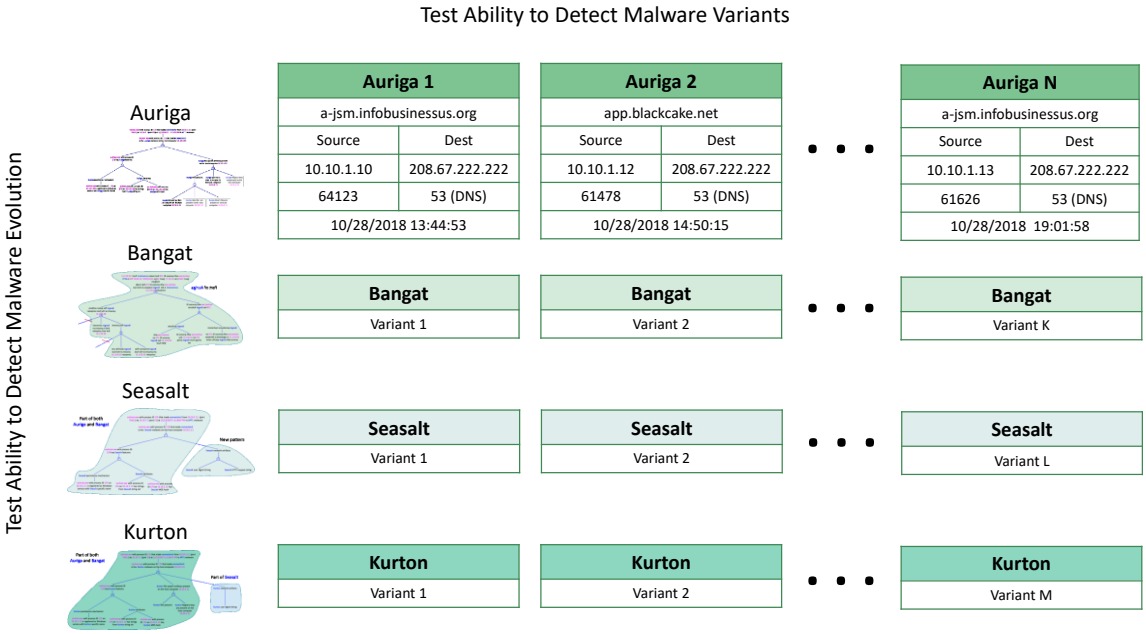

Figure 39 Overview of experiment protocol

I performed an experiment to test both the training of CAAPT to detect APT1 malware and the use of CAAPT to detect such intrusions. As shown in Figure 39, experiments were designed to test CAAPT's ability to detect configuration changes in the same malware and new malware versions as the attackers' tool set evolves over time. The experiment simulated the following evolution of APT1: Auriga → Auriga variants → Bangat → Bangat variants → Seasalt → Seasalt variants → Kurton → Kurton variants.

For this experiment only one running instance of the Automatic Analysis agent was used.

5.2.Auriga Experiment

First, detection of an Auriga intrusion was modeled along with the corresponding ontology. Based on the developed Auriga modeling, CAAPT learned 31 context-independent hypothesis patterns, two trigger rules, two indicator rules, 16 hypothesis analysis rules, 15 collection tasks, and 15 collection rules. Eight collection agents were also defined. CAAPT's detection capabilities were then tested in three scenarios:

- with the Auriga intrusion used in training (to test the system;
- with an intrusion by a variant of Auriga (to test system's capability of detecting such variants);
- with a Bangat intrusion (to test system's ability of detecting a new malware from the same family).

Table 2 provides an overview of the artifacts related to each malware variant tested in the Auriga experiment.

The variant of Auriga in the second test used a different APT1 domain to trigger the security alert and the malware process %SYSTEMROOT%\Temp\svchost.exe did not contain unique APT1 strings. Figure 40 shows a fragment of the analysis automatically generated by CAAPT when it investigated an intrusion with a variant of Auriga.

Table 3 summarizes the results of this experiment. Notice in column 3 that CAAPT succeeded in detecting the simulated attack using a variant of Auriga with the same probability as the training example.

Bangat, the next malware program in APT1 malware evolution was then tested to see if it could be detected even though CAAPT was not yet trained to detect it. As shown

Table 2 Auriga and Bangat experiment artifacts

Artifact Type	Auriga (Training)	Auriga (Variant)	Bangat
C2 Domain	a-jsm.infobusinessus.org	app.blackcake.net	canada.cnndaily.com
Service Name	riodrv32	riodrv32	(svchost) iprip
Service Display Name	N/A	N/A	Remote Access Auto Connection Manager
Service Description	N/A	N/A	N/A
Service Binary	N/A	N/A	%SYSTEM32%\rasauto32.dll
Temp File(s)	%TEMP%\~_MC_3~327	%TEMP%\~_MC_4~436	%TEMP%\~_MC_4~
Data File(s)	%TEMP%\sam.dat %USERPROFILE%\sam.sav	%TEMP%\sam.dat %USERPROFILE%\sam.sav	%TEMP%\sam.sav
Executable File	%WINDIR%\temp\svchost.exe	%WINDIR%\temp\svchost.exe	%WINDIR%\temp\svchost.exe
Auxiliary Program	N/A	N/A	N/A
Unique String(s)	superhard corp.	Not Present	superhard corp.
Library File(s)	%SYSTEM32%\drivers\riodrv32.sys %SYSTEM16%\netui.dll	%SYSTEM32%\drivers\riodrv32.sys	%SYSTEM32%\rasauto32.dll
Registry Key(s)	HKLM\SOFTWARE\riodrv32\TEMP HKLM\SOFTWARE\riodrv32\DEL	HKLM\SOFTWARE\riodrv32\TEMP HKLM\SOFTWARE\riodrv32\DEL	N/A
User Agent String	N/A	N/A	N/A

in Table 2, the main differences between Auriga and Bangat are that Bangat: (a) does not have the library files riodrv32.sys and netui.dll; (b) uses a different regular expression for its temporary file names; and (c) stores its data files in different folders. Bangat also uses different Windows Service names for its persistence mechanisms.

Notice in column 4 of Table 3 that CAAPT determined the probability L06 (75-80%) that there is an APT1 intrusion with the same probability of being Auriga intrusion.

This is because detection of an APT1 intrusion is based on network IOCs attributed to the APT1 group, but not necessarily to a specific malware program. By structuring the modeling in this way, the system can detect unknown APT1 malware even if all they do in a new attack is reuse old C2 infrastructure.

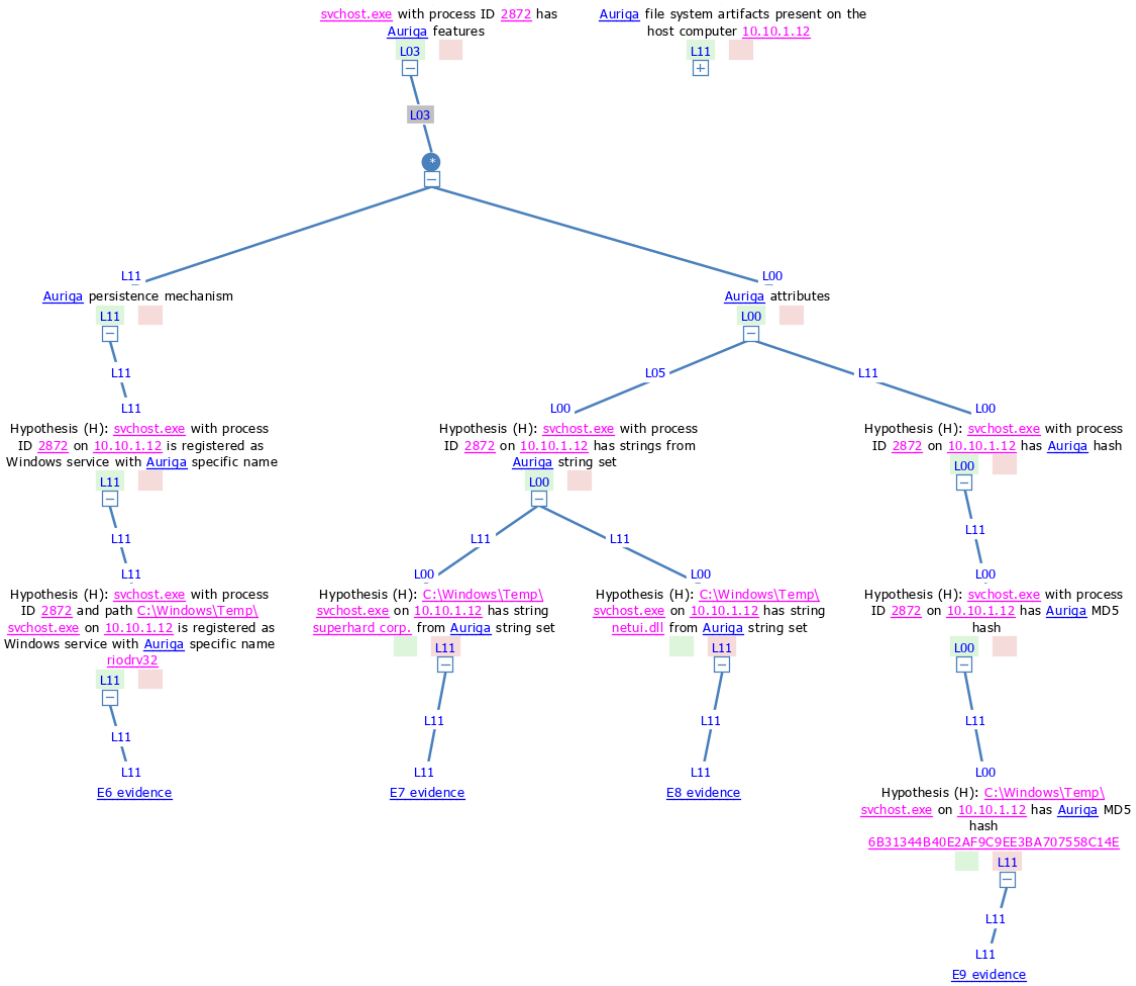

Figure 40 Fragment of the analysis of the Auriga variant from Table 2

Notice also the sum of the probabilities of "Auriga intrusion" and "APT1 intrusion" is over 100%. This is because these two hypotheses are not disjoint. Indeed, "APT1 intrusion" means any intrusion performed by the APT1 attacker group, using any of their malware tools, including Auriga.

The last row in Table 3 shows the duration of each experiment. The run time for the development and evaluation of the reasoning trees is 1 to 3 seconds. Most of the time is spent by waiting for the Collection Manager to return the results requested by the collection agents.

Table 3 Auriga experiment results

Experiment	With the Auriga intrusion used in training	With an intrusion by a variant of Auriga	With a Bangat intrusion
APT1 intrusion	L08 (85-90%) very likely	L08 (85-90%) very likely	L06 (75-80%)
Auriga malware	L08 (85-90%) very likely	L08 (85-90%) very likely	L06 (75-80%)
Duration	132 seconds	174 seconds	185 seconds

External dependencies, such as GRR, Elasticsearch, and VirusTotal, take time to perform searches and checks and some Collection Manager searches or checks require multiple calls to external agents. While these systems (particularly GRR and Elasticsearch) have been optimized to return results quickly, the time required to perform Auriga analysis was primarily made up of Collection Manager tasks.

5.3. Bangat Experiment

In the second experiment, detection of the Bangat intrusion from the first experiment was modeled, along with an extended APT1 ontology with the representation of Bangat malware as follows:

- Modeling began with the knowledge base generated by the last Auriga experiment (Bangat intrusion with the Auriga representation and the rules learned from the Auriga modeling).
- Within this knowledge base, the ontological representation of Bangat was developed.
- Analysis of a Bangat intrusion was performed using the ontological representation of Bangat, and the rules learned from the Auriga modeling.
- The generated analysis was refined and extended to accurately and completely analyze the Bangat malware intrusion.
- New rules needed to analyze Bangat were learned.

After learning the new rules, the CAAPT knowledge base was extended with six context-independent hypothesis patterns, five hypothesis analysis rules, three collection tasks, and three collection rules. As expected, teaching CAAPT to detect Bangat required only an incremental extension of the modeling, once the system had already been trained to detect Auriga.

After learning rules based on the new modeling, CAAPT's detection capabilities were tested in three scenarios:

- with the Bangat intrusion used in training;

- with an intrusion by a variant of Bangat;
- with a Seasalt intrusion.

Table 4 shows an overview of the simulated forensic artifacts used in the Bangat experiment. The Bangat variant used for training was the same as the one used in the Auriga experiment. The variant of Bangat used in the second run had three main differences. The alert was triggered with a different domain, the data files used in the first Bangat scenario were not present, and a different temporary file matching the Bangat regular expression was present on the infected host.

Table 4 Bangat and Seasalt experiment artifacts

Artifact Type	Bangat (Training)	Bangat (Variant)	Seasalt
C2 Domain	canada.cnndaily.com	data.firefoxupdata.com	mo.businessconsults.net
Service Name	(svchost) iprip	(svchost) nwsapagent	(svchost) SaSaut
Service Display Name	Remote Access Auto Connection Manager	Remote Access Auto Connection Manager	System Authorization Service
Service Description	N/A	N/A	Authorization and authentication service for starting and accessing machines.
Service Binary	%SYSTEM32%\rasauto32.dll	%SYSTEM32%\rasauto32.dll	%SYSTEM32%\svc.dll
Temp File(s)	%TEMP%\~_MC_4~	%TEMP%\~_MC_6~	N/A
Data File(s)	%TEMP%\sam.sav	None	%TEMP%\sam.dat %USERPROFILE%\sam.sav
Executable File	%WINDIR%\temp\svchost.exe	%WINDIR%\temp\svchost.exe	%WINDIR%\temp\svchost.exe
Auxiliary Program	N/A	N/A	%USERPROFILE%\java.exe
Unique String(s)	superhard corp.	superhard corp.	fxftest upfileok ubuntuguru.strangled.net/postinfo.html
Library File(s)	%SYSTEM32%\rasauto32.dll	%SYSTEM32%\rasauto32.dll %SYSTEMROOT$\temp\svchost.exe	%SYSTEM32%\drivers\riodrv32.sys
Registry Key(s)	N/A	N/A	N/A
User Agent String	N/A	N/A	Mozilla/4.0 (compatible; MSIE 5.00; Windows 98) KSMM

Table 5 summarizes the results of this experiment. Notice in column 3 that CAAPT succeeded to detect the intrusion with the variant of Bangat with the same probability as the detection of the intrusion with Bangat training example. This result shows CAAPT's ability to detect attacks that use the same malware with changes to the program's configuration.

Table 5 Bangat experiment results

Experiment	With the Bangat intrusion used in training	With an intrusion by a variant of Bangat	With a Seasalt intrusion
APT1 intrusion	L08 (85-90%) very likely	L08 (85-90%) very likely	L08 (85-90%) very likely
Auriga malware	L06 (75-80%)	L06 (75-80%)	L01 (50-55%) barely likely
Bangat malware	L08 (85-90%) very likely	L08 (85-90%) very likely	L01 (50-55%) barely likely
Duration	327 seconds	285 seconds	275 seconds

I then tested detection of the next malware from the APT1 family, Seasalt, for which CAAPT was not trained. As shown in Table 4, Seasalt uses a specific name (SaSaut) to register itself as a Windows Service and has different unique strings associated with it. It also does not use the command shell technique shared by Auriga and Bangat. Seasalt added an auxiliary program, which the Seasalt Windows Service DLL starts, and the network protocol was changed so it can be detected using a unique HTTP User Agent String.

Notice in column 2 that the probability of "Auriga intrusion" is L06 (75-80%) and the probability of "Bangat intrusion" is L08 (85-90%). This is not a contradiction because these two hypotheses are not disjoint. The Bangat malware is an evolution of the Auriga malware and therefore it has many features in common with Auriga. When checking for

an intrusion with Auriga, the system looks for the presence of the features of the Auriga malware on the infected computer, but some of these features are also the features of Bangat, so it is possible that the computer is infected by both Auriga and Bangat. Therefore, Auriga intrusion with probability L06 (75-80%) covers the case where the Auriga intrusion is accompanied by a Bangat intrusion. Similarly, Bangat intrusion with probability L08 (85-90%) is based on the detected Bangat features on the host computer which also includes some Auriga features. Thus, this probability also covers the case when there is both a Bangat an Auriga and intrusion.

Notice also in column 4 of Table 5 that CAAPT detected that very likely (85-90%) there is an APT1 intrusion, but the probability of being Auriga or Bangat is only barely likely 50-55%. This result shows the theoretical model can both identify new attacks by the same attacker and distinguish between different malware tools used in the attacks.

5.4. Seasalt Experiment

In the third experiment CAAPT was trained to detect the Seasalt intrusion from the second experiment, and the APT1 ontology was extended with the representation of Seasalt as follows:

- Modeling began with the knowledge base generated by the last Bangat experiment (Seasalt intrusion with the Auriga and Bangat representations and the rules learned from the Auriga and Bangat modeling).
- Within this knowledge base the ontological representation of Seasalt was developed.

Table 6 Seasalt and Kurton experiment artifacts

Artifact Type	Seasalt (Training)	Seasalt (Variant)	Kurton
C2 Domain	mo.businessconsults.net	kl-hqun.gmailboxes.com	launch.todayusa.org
Service Name	(svchost) SaSaut	(svchost) SaSaut	(svchost) iprip
Service Display Name	System Authorization Service	System Authorization Service	Remote Access Auto Connection Manager
Service Description	Authorization and authentication service for starting and accessing machines.	Authorization and authentication service for starting and accessing machines.	N/A
Service Binary	%SYSTEM32%\svc.dll	%SYSTEMROOT%\svc.dll	%SYSTEM32%\svc.dll
Temp File(s)	N/A	N/A	N/A
Data File(s)	N/A	N/A	%SYSTEM32%\SvcHost.DLL.log
Executable File	N%WINDIR%\temp\svchost.exe	%WINDIR%\temp\svchost.exe	%WINDIR%\temp\svchost.exe
Auxiliary Program	%USERPROFILE%\java.exe	%USERPROFILE%\java.exe	N/A
Unique String(s)	fxftest upfileok ubuntuguru.strangled.net/postinfo.html	Same as Training Variant	"you specify service name not in Svchost\netsvcs, must be one of following:" "CreateService(%s) SUCCESS. Config it" "Exception Catched 0x%X" "MyTmpFile.Dat"
Library File(s)	N/A	N/A	N/A
Registry Key(s)	N/A	Not present	HKLM\SOFTWARE\Microsoft\DirectT\dwHighDateTime HKLM\SOFTWARE\Microsoft\DirectT\dwLowDateTime
User Agent String	Mozilla/4.0 (compatible; MSIE 5.00; Windows 98) KSMM	Not present	Mozilla/4.0 (compatible; MSIE 7.0;)

- Generation of the analysis of the Seasalt intrusion was then attempted using the ontological representation of Seasalt and the rules learned from Auriga and Bangat. However, because the ontological representation of Seasalt is significantly different from those of Auriga and Bangat, many of the rules were not applicable.
- A new modeling for Seasalt intrusion was then created.

- The new rules needed to analyze Seasalt were learned, reusing the applicable rules learned from Auriga and Bangat (for example the analysis of the persistence mechanism and of the Registry keys).

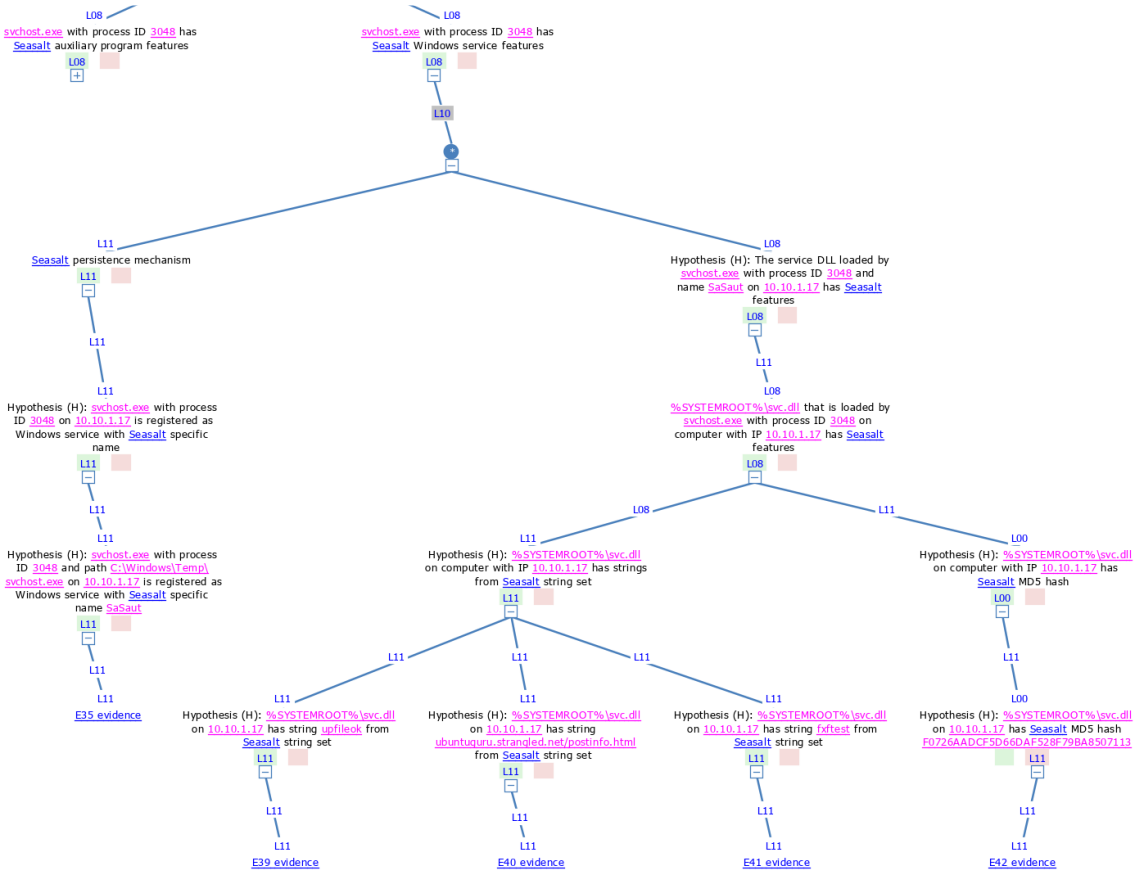

Figure 41 Fragment of the analysis of the Seasalt variant

Following rules learning, the CAAPT knowledge base was extended with 10 context-independent hypotheses patterns, seven hypotheses analysis rules, five collection tasks, and five collection rules. One new collection agent was also defined. This was a

larger extension of the system due to a more drastic change in Seasalt's features. After that, CAAPT's detection capabilities were tested in three scenarios:

- with the Seasalt intrusion used in training;
- with an intrusion by a variant of Seasalt;
- with a Kurton intrusion.

As shown in Table 6, the variant of Seasalt used in the second run showed CAAPT's ability to handle variance in its detection model. A different domain was used to trigger the BRO alert, the Windows Service DLL was in a different location than in the first run, the HTTP User Agent string was not present, and the Registry keys were not present.

Figure 41 shows a fragment of the reasoning of CAAPT when it analyzed the variant of Seasalt.

Table 7 summarizes the results of this experiment. Notice in column 3 that CAAPT succeeded to detect the intrusion with the variant of Seasalt with the same probability as the detection of the intrusion with the Seasalt training example.

Table 7 Seasalt experiment results

Experiment	With the Seasalt intrusion used in training	With an intrusion by a variant of Seasalt	With a Kurton intrusion
APT1 intrusion	L08 (85-90%) very likely	L08 (85-90%) very likely	L08 (85-90%) very likely
Auriga malware	L01 (50-55%) barely likely	L01 (50-55%) barely likely	L01 (50-55%) barely likely
Bangat malware	L01 (50-55%) barely likely	L01 (50-55%) barely likely	L03 (60-65%) likely
Seasalt malware	L08 (85-90%) very likely	L08 (85-90%) very likely	L00 (0-50%) lacking support
Duration	382 seconds	406 seconds	344 seconds

I then tested the detection of the next malware from the APT1 family, Kurton, for which CAAPT was not trained. Unlike Seasalt, which uses a single name for registering itself as a Windows Service, Kurton can be configured to use any name the user wants. However, threat intelligence (Mandiant, 2013) shows that the names *iprip*, *nwsapagent*, and *iprip32* were the most common names. Kurton also does not use an auxiliary program, but uses two different HTTP User Agent Strings, a data file located at %WINDIR%\System32\SvcHost.DLL.log, and unique Registry keys.

Notice in column 4 of Table 7 that CAAPT detected that very likely (85-90%) there is an APT1 intrusion, but the probability of being Auriga is only barely likely 50-55%, the probability of being Bangat is (60-65%) likely, and there is no evidential support that it is Seasalt. This experiment further confirmed CAAPT's ability to identify new attacks while distinguishing between specific malware programs.

5.5. Kurton Experiment

In the final simulated malware experiment I modeled detection of the Kurton intrusion from the third experiment, and extended the APT1 ontology with the representation of Kurton as follows:

- Modeling started with the knowledge base generated by the last Seasalt experiment (Kurton intrusion with the Auriga, Bangat, and Seasalt representations and the rules learned from their modeling).
- Within this knowledge base the ontological representation of Kurton was developed.

- Generation of the analysis of the Kurton intrusion was then attempted using the ontological representation of Kurton and the rules learned from the Auriga, Bangat, and Seasalt modeling. However, because the representation of Kurton combines some aspects of Auriga and Bangat (for example, the presence of data file artifacts) with some aspects of Seasalt (for example, the presence of network artifacts) while excluding other aspects of Auriga/Bangat/Seasalt, the top parts from any of the previous analyses were not applicable.
- A new modeling for Kurton was developed by combining the applicable parts from Auriga, Bangat, and Seasalt.
- New rules needed to analyze Kurton were learned, reusing the applicable rules learned from Auriga, Bangat, and Seasalt.

Following rules learning, the CAAPT knowledge base was extended with only one context-independent hypothesis pattern, and two hypotheses analysis rules. This again shows CAAPT's ability to adapt to learn new malware with only incremental changes. After learning the new rules, CAAPT's detection capabilities were tested in two scenarios:

- with the Kurton intrusion used in training;
- with an intrusion by a variant of Kurton.

Table 8 Kurton experiment artifacts

Artifact Type	Kurton (Training)	Kurton (Variant)
C2 Domain	launch.todayusa.org	ail.defenceonline.net
Service Name	(svchost) iprip	(svchost) nwsapagent
Service Display Name	Remote Access Auto Connection Manager	System Authorization Service
Service Description	N/A	Authorization and authentication service for starting and accessing machines.
Service Binary	%SYSTEM32%\svc.dll	%SYSTEM32%\nwsapagent.dll
Temp File(s)	N/A	N/A
Data File(s)	%SYSTEM32%\SvcHost.DLL.log	%SYSTEM32%\SvcHost.DLL.log
Executable File	%WINDIR%\temp\svchost.exe	%WINDIR%\temp\svchost.exe
Auxiliary Program	N/A	N/A
Unique String(s)	"you specify service name not in Svchost\netsvcs, must be one of following:" "CreateService(%s) SUCCESS. Config it" "Exception Catched 0x%X" "MyTmpFile.Dat"	CreateService(%s) SUCCESS. Config it Exception Catched 0x%X MyTmpFile.Dat
Library File(s)	N/A	N/A
Registry Key(s)	HKLM\SOFTWARE\Microsoft\DirectT\dwHighDateTime HKLM\SOFTWARE\Microsoft\DirectT\dwLowDateTime	HKLM\SOFTWARE\Microsoft\DirectT\dwHighDateTime HKLM\SOFTWARE\Microsoft\DirectT\dwLowDateTime
User Agent String	Mozilla/4.0 (compatible; MSIE 7.0;)	Mozilla/4.0 (compatible; MSIE8.0; Windows NT 5.1)

As shown in Table 8, the variant of Kurton used in the second run used a different name for the Windows Service and DLL filename and had a different unique HTTP User Agent string present.

Figure 42 shows a fragment of the reasoning of CAAPT when it analyzed the variant of Kurton.

Table 9 summarizes the results of this experiment. Notice in column 3 that CAAPT succeeded to detect the intrusion with the variant of Kurton with the same probability as the detection of the intrusion with the Kurton training example, L07 (80-85%). This is slightly lower than L08 (85-90%), the probability of being an APT1

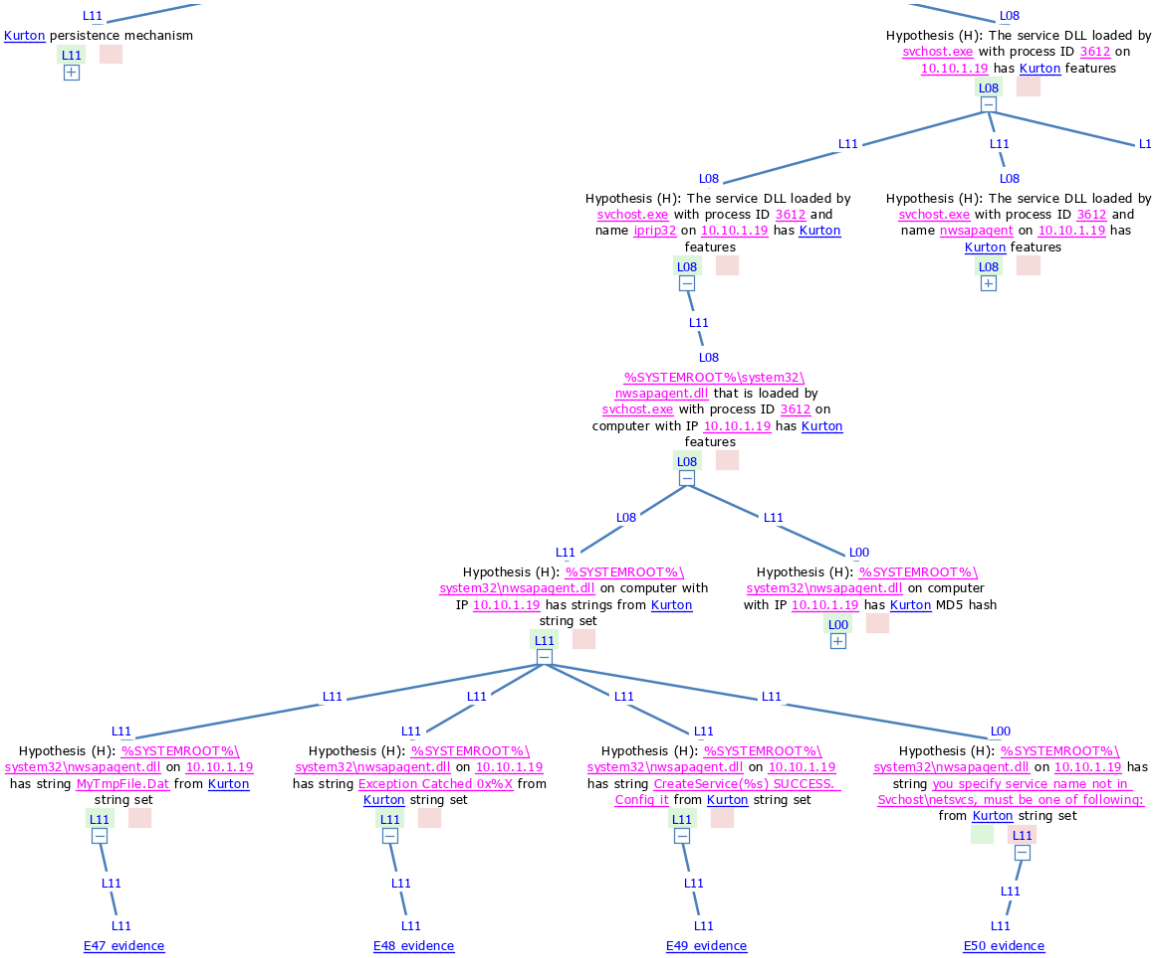

Figure 42 Fragment of the analysis of the Kurton variant

intrusion. Compared to the other analyzed malware, Kurton has fewer forensic indicators to examine. The largest subset of indicators are unique strings, which have less inferential force than other types of indicators. As such, without a matching hash value (which is normal for APT1 intrusions) the highest expected probability for detection of Kurton is L07.

Table 9 Kurton experiment results

Experiment	With the Kurton intrusion used in training	With an intrusion by a variant of Kurton
APT1 intrusion	L08 (85-90%) very likely	L08 (85-90%) very likely
Auriga malware	L01 (50-55%) barely likely	L01 (50-55%) barely likely
Bangat malware	L03 (60-65%) likely	L03 (60-65%) likely
Seasalt malware	L00 (0-50%) lacking support	L00 (0-50%) lacking support
Kurton malware	L07 (80-85%)	L07 (80-85%)
Duration	587 seconds	631 seconds

5.6. Detection of Real APT1 Malware

While the above experiments were conducted using attacks simulated by manually creating forensic artifacts, I also tested CAAPT's ability to detect attacks simulated by infecting VMs with real APT1 malware, specifically Bangat and Seasalt. This was done by reverse engineering the malware samples for each to determine how to install them on Windows 7 (they were originally developed for various earlier versions of Windows and Windows Server). Reverse engineering included both static and dynamic techniques to determine the malware samples' behavior.

In both cases, the malware is in the form of a Windows Service DLL. Installation of the service is done by running the installation function using the program *rundll32.exe*, which is designed to load a DLL and run a specified DLL export function. In the case of Bangat, the function is called *RundllInstall*, and the Seasalt function is called *Install*. Using this method, Seasalt installed and ran as expected. Getting Bangat to run was more challenging because it used the same service display name as an existing service. Because I was unable to manually change the name of the existing service, I had to manipulate the Bangat installation function as it was running. To do this I ran it in a

debugger, changing the display name in memory before the service registration function was executed. Doing this caused Bangat to install correctly. I then manually changed its display name to the original value using the Windows Registry.

Table 10 Real malware experiment artifacts

Artifact Type	Bangat	Seasalt
C2 Domain	att.infosupports.com	ubuntuguru.strangled.net
Service Name	(svchost) iprip	(svchost) Sasaut
Service Display Name	Remote Access Auto Connection Manager	System Authorization Service
Service Description	N/A	Authorization and authentication service for starting and accessing machines.
Service Binary	c:\temp\bangat1.dll	%USERPROFILE%\Documents\svc.dll
Temp File(s)	none	N/A
Data File(s)	N/A	N/A
Executable File	%WINDIR%\system32 \svchost.exe	%WINDIR%\temp\svchost.exe
Auxiliary Program	N/A	N/A
Unique String(s)	superhard corp. !b=z&7?cc,MQ>	fxftest upfileok ubuntuguru.strangled.net
Library File(s)	N/A	N/A
Registry Key(s)	none	none
User Agent String	N/A	None

Table 10 shows the artifacts present after installation and execution of Bangat and Seasalt. In both cases, only artifacts related to persistence and initial communication to the C2 servers were present. Data files, temporary files, and HTTP user agent strings only present themselves on the network during later stages of an infection, when the attacker has an interactive remote connection to the malware. I did not progress the

attack to later stages during this testing as reproducing this level of interactivity is very difficult and beyond the scope of the research.

Table 11 Real malware experiment results

Experiment	With real Bangat infection	With real Seasalt infection
APT1 intrusion	L00 (0-50%) lacking support	L00 (0-50%) lacking support
Auriga malware	L01 (50-55%) barely likely	L01 (50-55%) barely likely
Bangat malware	L05 (70-75%) likely	L01 (50-55%) barely likely
Seasalt malware	N/A	L08 (85-90%)
Duration	296 seconds	407 seconds

Table 11 shows the results of the experiment. In both cases, there was L01(50-50%) probability that there was an Auriga intrusion. This was due to the presence of the c:\Windows\Temp directory, which is common on Windows 7 computers but may not exist on Windows XP. In the Bangat experiment, the malware was detected with a L05 (70-75%) probability. While the probability is lower than that of the simulated experiments, the result is expected as there were fewer artifacts of the malware present as the attack had not progressed as far as in the simulated experiment. This result shows the trained models are robust and can still detect the malware with reasonably high probability even in the early stages of infection.

In the experiment using real Seasalt malware, it was detected with L08 (85-90%) probability, the same probability from the simulated experiment. Seasalt has fewer detectable artifacts created during later attack stages. Most of the modeled artifacts are either created during installation or present in the executable files. This means detection probability for early and late stage Seasalt intrusions are similar.

The experimental results of testing against real malware infections clearly show the learning agents in CAAPT are not only capable of detecting simulated malware infections but are also effective in detecting real APT1 malware infections.

5.7. Summary of Experimental Results

The above sections summarized the results of each experiment, separate from each other. In this section I consider all of these results together, in the context of the experimental goals I sought to achieve.

Overall, CAAPT learned 40 context-independent hypothesis patterns, two trigger rules, two indicator rules, 23 hypothesis analysis rules (some of them with large pattern trees that contain many context-depended hypotheses), 23 collection tasks, and 23 collection rules. Ten collection agents were also defined. They enabled CAAPT to detect intrusions from four families of the APT1 malware: Auriga, Bangat, Seasalt, and Kurton.

APT1, like many other attacker groups, practiced evolutionary development to adapt its malware to changes in network defense technology or simply to increase efficiency. These changes in the way malware presents itself on the network and on disk have made it difficult for signature-based intrusion detection tools to detect attacks because the attackers can change static information in their malware faster than defenders can adapt. However, the patterns of behavior change more slowly and with less variance. These characteristics of APT evolutionary development were successfully exploited by CAAPT, as shown by the experimental results.

5.7.1. Ability to Automatically Detect the Training Malware

In all cases, CAAPT was able to detect the malware it was trained on with high (L08 (85-90%) very likely) probability. This occurred even in the cases where real malware was used in the simulated attack. The only exception was for Kurton, where its smaller set of indicators with weaker inferential force yielded a detection probability of L07 (80-85%). This result, while expected, confirms the ability of agile cognitive agents to successfully orchestrate the incident response process in detection of known malware.

5.7.2. Ability to Detect Variants of the Training Malware

Next, after CAAPT was trained, based on one instance of the Auriga malware, it was able to also detect a variant of this malware. This was also the case with the other three malware programs considered (Bangat, Seassalt, and Kurton) and is a consequence of the learning method employed by CAAPT. Indeed, CAAPT generalizes a specific example and its explanation into a general rule that also cover similar examples that are likely to correspond to variants of the malware used in training. This important result shows the orchestration and automation conducted by cognitive agents can be generalized to variations in a malware program's configuration.

5.7.3. Some Ability to Detect Evolved Malware

Experimentation showed that CAAPT also succeeded in anticipating the evolutionary changes in the malware by learning patterns of IOCs in the form of hypothesis analysis rules. If one aspect of the malware's behavior changed and became undetectable by CAAPT, it was still detected with some probability based on the remaining observable evidence. For example, as shown in Table 3, after being trained to

detect Auriga and invoked to analyze a simulated intrusion using Bangat, CAAPT still reported an APT1 intrusion with probability 85-90%, but the probability of being Auriga was lower (75-80%). In the case of analyzing Seasalt after being trained on Auriga and Bangat, CAAPT still detected an APT1 intrusion with probability 85-90%, but the probability of being Auriga or Bangat was only 50-55% (see Table 5). A similar result was obtained in the case of analyzing Kurton after being trained on Auriga, Bangat, and Seasalt. CAAPT still detected an APT1 intrusion with probability 85-90%, but the probability of being Auriga was 50-55%, of being Bangat was 60-65%, and of being Seasalt was 0-50% (see Table 7).

These results showed two key novel features of CAAPT. First, it is capable of detecting new attacks by a sophisticated attacker even when they update their malware tools with new features and capabilities. The ability to detect previously unknown attacks is very difficult in cybersecurity and CAAPT's ability to do it, even when constrained to the scope of a single APT group, is a major contribution to security research. Second, this set of experiments showed CAAPT is capable of distinguishing between an APT group's malware tools once CAAPT has learned to detect multiple malware programs. This is important because sophisticated attackers use different tools in different parts of their attacker lifecycle. The ability to distinguish between malware tools means CAAPT can also autonomously determine what phase of the attacker lifecycle the attack is in, which in turn can tell network defenders how severe the detected intrusion is. It is a unique capability of this research.

5.7.4. Limited Incremental Training Needed to Detect a New Malware Family

These results are also explained by the training methodology which exploits the evolutionary development of APT1. Indeed, as discussed in the previous sections, modeling the detection of new malware is done based on the modeling of previous malware, and therefore shares many parts with the analyses of previous malware. This also significantly simplifies and speeds up the training of the agent. For example, as discussed previously, to train for Auriga detection, CAAPT had to learn 28 context-independent hypothesis patterns, two trigger rules, two indicator rules, 13 hypothesis analysis rules, 15 collection tasks, and 15 collection rules. Eight collection agents had also to be defined. Many of these were also applicable for the detection of Bangat intrusions. Therefore, to train for Bangat detection, a reduced number of knowledge elements needed to be learned: one context-independent hypothesis pattern, one hypotheses analysis rule, one collection task, and one collection rule. The same is true for the training to detect Seasalt and Kurton. The amount of knowledge elements that needed to be learned depended on the amount of change in the new malware. Notice, for example, that the two trigger rules and the two indicator rules learned for Auriga were also applicable to Bangat, Seasalt, and Kurton. Also, after defining eight collection agents to collect evidence for Auriga detection, only two more were needed to cover the collection needs for Bangat, Seasalt, and Kurton.

5.7.5. Efficient and High-Quality Analysis

While CAAPT coverage of malware detection is limited to APT1, and the increase in coverage will also increase the detection time, the times obtained in my

experiments are very small and support my hypothesis that a system like CAAPT will significantly speed–up the malware detection process. The total runtime to detect an intrusion increased from around 2 minutes, when CAAPT was checking for Auriga intrusions only, to around 10 minutes when CAAPT was checking for Auriga, Bangat, Seasalt, and Kurton intrusions (see Figure 43). However, the run time for the development and evaluation of the reasoning trees only increased from around 2 seconds, when CAAPT was checking for Auriga intrusions only, to around 6 seconds when CAAPT was checking for Auriga, Bangat, Seasalt, and Kurton intrusions.

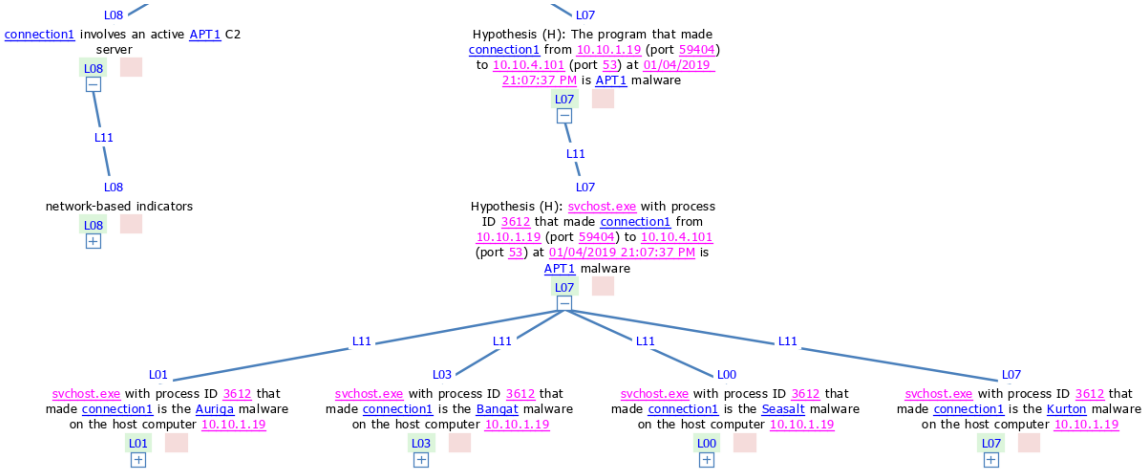

Figure 43 Analysis fragment for all studied APT1 malware

As discussed previously, most of the time is spent waiting for the Collection Manager to return the results requested by the collection agents. External dependencies, such as GRR, Elasticsearch, and VirusTotal, take time to perform searches and checks and some Collection Manager requests require multiple calls to external agents.

But time is only part one of advantages offered by a system like CAAPT. While professional CSOCs have processes to be followed by analysts to ensure consistent analytical quality, it is natural for analysts to take shortcuts when they believe evidence examined early in the process leads to an obvious answer. These analytical leaps can shorten analysis times but can also lead to errors. CAAPT, on the other hand, will follow its learned processes fully every time. This reduces error and provides consistent analytical results. As the number of evaluated hypotheses grows and processing times increase, it can be mitigated with additional computing power, shortening the amount of time required to exhaustively evaluate all generated competing hypotheses.

It is also possible to shorten the automated analysis time by restricting which analysis trees are used for which abductive triggers. For example, APT1 has 17 known malware programs used in the *initial compromise* phase of the attacker lifecycle and 27 malware programs used during the *gain foothold* phase (Mandiant, 2013). This limits the computational requirements for APT1 to 44 analysis trees. The APT1 hypotheses would likely be used for only abductive triggers generated from threat intelligence related to APT1. If CAAPT were trained to detect intrusions from multiple APT groups, the automated analyses conducted would be restricted to abductive triggers related to that APT group. This strategy places a cap on the computational requirements for CAAPT's autonomous analysis.

6. CONCLUSIONS

This section provides a summary of the major contributions of this research, status of the research with regard to the research questions described in Section 1.4, and possible future directions of the research.

6.1. Research Contributions

In this dissertation, I have described several novel contributions achieved in this research, including the following major contributions.

Ontology design for APT detection. First, I developed the ontological framework for describing the knowledge required to model the detection process for sophisticated threats, including APTs. For this research, I modeled knowledge of APT activity, the attacker lifecycle, the network environment, security alerts, and host-based IOCs used to detect malware. The ontological modeling is described in Section 3.2.1 and represents an advancement in modeling cybersecurity threat intelligence for use by cognitive agents to reason about sophisticated cyber threats.

Conceptual modeling of the automatic APT detection process. The second part of creating a theoretical model of APT detection was modeling the abductive, deductive, and inductive reasoning processes required to detect sophisticated threats. This included the modeling of all of the hypotheses required for detecting APT1 activity, including the four malware programs studied in this research (Section 3.2.2), the deductive process required to detect APT intrusions, decomposing the high-level hypotheses into sub-hypotheses until there was a requirement to search for specific IOCs

(Section 3.2.3), and the inductive process of using found evidence to derive conclusions as to the probability that a given hypothesis was true (Section 3.2.4). In total, this represents the first theoretical model of APT detection.

In addition to the fact that the APT detection model created in this research is novel, it is powerful for a few key reasons:

- The model uses the combined inferential force of weak indicators to synthesize, with high probability, conclusions about the presence of an APT attack. As such is robust, meaning it can detect APT intrusion activity even if some of the IOCs it searches for are missing.
- It is capable of detecting intrusions by an APT group when they change the configuration of their malware, which is unique in the security industry.
- It is capable of detecting APT intrusions even when they use new or evolved versions of malware. This predictive detection capability allows the cognitive agents, in collaboration with CSOC analysts, to cope with the evolution of sophisticated attackers' TTPs.

Automatic generation of abductive triggers from basic IDSs (e.g., BRO). While creation of abductive triggers from observations is a well-established technique in the field of cognitive AI, and is a core feature of Disciple, this research extended the capability of Disciple to automatically generate abductive triggers based on alerts from security devices. Specifically, the BRO IDS was used in this research to generate abductive triggers based on security alerts generated when a computer attempted to

contact a known APT1 domain. This contribution, described in detail in Section 3.2.2, required the theoretical model described above, along with custom software capable of identifying new security alerts from cyber threat intelligence and transforming them into messages understandable by the Trigger Agent. The final result of this contribution is a system which can automatically react to new security alerts and begin the abductive reasoning process. While this research focused specifically on the BRO IDS as a security sensor, both the theoretical model and the software developed during this study are broadly applicable, meaning any host-based or network-based detection system can be used to automatically generate abductive triggers.

Autonomous, hypothesis-driven search for evidence. Using APT1 as a case study, I created a theoretical model that, when implemented in Disciple, allowed cognitive agents to search for digital evidence in an autonomous manner. Furthermore, the hypothesis-driven search for evidence was done in an abstract manner, meaning the evidence requirements of the cognitive agents is decoupled from the specific search or collection agents used to satisfy the searches. This contribution is important because it allows for the cognitive agents to act in an autonomous manner to collect and reason about digital evidence in a CSOC environment (the key feature for security incident response orchestration), but also to collect much of the evidence in an on-demand fashion, reducing the data storage and network bandwidth requirements of the system.

Selection and integration of multiple, collaborative, search and collection agents working together to detect and investigate threats. In order to apply the theoretical model of APT detection to a real CSOC environment, a set of search and

collection agents was needed to satisfy the requirements of hypothesis-driven search for evidence. A major architectural contribution of this research was the identification of the types of data sources required for CSOC integration and selection of specific search and collection agents for use in the research. A contribution is also the deployment and configuration of these agents so the cognitive agents could use live data sources in their analyses. The search agent selection and architecture were described in Sections 3.3.1 and 3.3.2.

Development of the Collection Manager software for translating and optimizing abstract searches into searches executable by real collection agents. Lastly, this research led to development of the Collection Manager software, which is primarily responsible for translating abstract search requests from Disciple agents into concrete searches using real search and collection agents, and then translating the search results into evidence usable in the automatic analysis process. The Collection Manager is designed to provide an abstraction layer between the knowledge of the cognitive agents and the specific CSOC infrastructure. As described in Section 3.3.3, the architecture of the Collection Manager makes CAAPT easy to integrate with new CSOC environments, meaning the knowledge of the trained cognitive agents can be transplanted from CSOC to CSOC with minimal re-engineering of the system.

6.2. Status of Research Questions

As described in detail in Section 1.4, this study aimed to answer key research questions about improving the quality of intrusion detection for modern threats, supporting analysts in understanding and detecting both known and unknown attacks, and

increasing CSOC accuracy and efficiency. This section focuses on how well my research address each of these questions.

6.2.1. Intrusion Detection Improvements

The research questions around improving intrusion detection centered around whether or not cognitive agents using evidence-based reasoning could improve on the state of the art regarding two key aspects: 1) the ability to fuse data from multiple sources, both real-time and on-demand, to provide better threat detection than traditional IDSs and 2) using explicit logic to combine the strength of multiple weak IOCs into strong detections. This research answered both of these questions in the affirmative.

As discussed in depth in Section 3.3, CAAPT fuses data from a variety of passive and on-demand collection agents to detect intrusions. The research has demonstrated that combining data from internal data sources such as GRR, SYSMON, BRO, and Elasticsarch with data from external sources such as VirusTotal and WHOIS can yield detection capabilities more robust against evolving attacks than individual security tools can do on their own. The explicit logic described in Section 3.2 demonstrates this robust detection model where the combined inferential force of weak IOCs such as unique strings, filenames, and Registry keys can be used to detect, with high probability, sophisticated threat activity even when strong IOCs such as file hashes are not present. This key finding is critical to the ability to orchestrate security incident response using cognitive agents.

6.2.2. Analyst Support

The second category of research questions was whether or not CAAPT can support CSOC analysts in their intrusion analysis tasks by:

- being able to be trained to detect known threats based on CSOC expertise;
- supporting analysts in detecting previously unknown attacks through mixed-initiative reasoning;
- exhibiting flexible autonomy (strictly-guided, mixed-initiative, and full autonomy);

The theoretical model of threat detection described in Section 3.2 show how the system can be trained to detect known threats. A core design characteristic of the system is its ability to be trained by an expert CSOC analyst to detect threats using available threat intelligence. As the experimental results show how the system was trained to detect the Auriga malware used by the attacker group APT1. As was further shown in my experiments, the system is able to detect attacks by the same attacker group using previously unknown malware. This ability is rooted in a robust detection model which separates the probability of an attack being conducted by the attacker group (the left branch of the top-level analysis tree in Figure 19) and the probability that a specific malware program was used. As the experiments show, the system was able to detect not only variants of the same malware, but also new malware created as the group's malware arsenal evolved.

The mixed-initiative analysis process described in Section 3.4 shows how the system supports analysts in detecting previously unknown attacks. Once trained to detect

one or more attacks by an attack group, the system will autonomously detect the same or similar attacks. If the system encounters an attack it does not understand, the alert is sent to an analyst for additional training. Using previously learned reasoning rules, the system will suggest reasoning strategies to the analyst to guide them through the manual analysis process. These suggested reasoning strategies act as playbooks for analysis. This process also demonstrates flexible autonomy, where the cognitive agents are strictly-guided when being trained to detect new threats, use mixed-initiative reasoning when potentially new attacks are discovered which are similar to attacks the agents have been trained to detect, and fully autonomous when detecting attacks for which the agents have been trained to detect.

6.2.3. CSOC Performance

The final category of research questions involves whether security incident response orchestration using cognitive agents can improve CSOC performance. CSOCs generally use detection rate, false positive rate, and speed to determine the effectiveness and efficiency of the CSOC. The results of this research show CAAPT can help a CSOC improve all of these metrics.

Improvement in detection rates for sophisticated threats is a key contribution of this research. CSOCs often rely on high-quality IOCs from threat intelligence to detect threats, meaning someone must have previously discovered the threat, analyzed it, and published the IOCs to the security community in order for the threat to be detected. While CAAPT also relies on this same model for initial training, its ability to predict attacks by the same attacker group using new TTPs, as shown through experimentation

drastically improves detection rates for previously unknown attacks. CSOCs are also vulnerable to evolving threats. If an attacker group were to change one high-quality IOC in a new attack, many CSOCs would fail to detect it. Because CAAPT uses explicit logic to combine multiple weak IOCs into strong detection models, it can cope with changing attacker TTPs much better than traditional tools. An attacker would have to change their attack methodology much more radically to evade detection by CAAPT.

Improving false positive rates is another area where CAAPT can help improve CSOC effectiveness. False positive use cases are often CSOC-specific and fairly static, meaning they don't change frequently. While it was not shown through experimentation, the cognitive agents designed in this research can be trained to understand false positive use cases and apply them autonomously, reducing the overall false positive rate of the CSOC.

The cognitive agents developed for this research can also speed up CSOC operations. While the sample size in the experiments I performed is not large enough to show how CAAPT behaves when trained to detect dozens or hundreds of threats, initial testing shows it scales efficiently even using a single server to run the system. Adding additional computing power will allow the system to perform efficiently as the size of the knowledge base increases. More importantly, because the cognitive agents can automate repetitive CSOC work, it frees up valuable analysts to perform more impactful tasks such as threat hunting and intelligence analysis, improving the overall usage of CSOC resources.

6.3. Future Research

While the early results from this research are exciting, the possible future applications of the research are just as promising. This section provides a brief description of future directions I envision for this research.

Autonomously Investigating the Attacker Lifecycle. One of the key aspects of the theoretical model of sophisticated threat detection is modeling of the attacker lifecycle, when the system not only detects a threat, but also knows how far into the attacker lifecycle the attack has progressed. In future research, this context could be used to perform root cause analysis or to discover evidence of previous stages of the attack that may have been missed. This approach is described in more detail in Appendix A.

Improving Scalability of the Collection Manager. Because this was basic research, I was not able to explore how the system would perform at scale, on networks with tens of thousands of nodes, complex network topologies, and trained to detect dozens of threats. Collection Manager performance can be improved through caching of search to speed up responses, scheduling of searches of multiple machines to reduce network bandwidth and CPU usage, and load-balancing multiple Collection Manager servers to increase capacity.

Autonomous Threat Hunting. Detection of threat in near-real-time is just one aspect of the detection requirements of CSOCs. More mature organizations employ very experienced analysts to hunt for signs of stealthy threats on their network, leveraging intelligence on emerging threats or newly-discovered tactics. CAAPT could be trained to

leverage the same intelligence, and tactics repositories such as MITRE ATT&CK (attack.mitre.org) to autonomously hunt for threats on the network.

APPENDIX A – AUTONOMOUS INVESTIGATION OF THE ATTACKER LIFECYCLE

In previous applications of Disciple to other analytical domains, synthesis of a conclusion was the end of the analytical process. In future research, I can extend the analytical workflow by allowing autonomously synthesized conclusions to analytical scenarios to generate triggers for new analytical scenarios to evaluate. This chaining of autonomous analyses allows for emergent analysis to take place, including autonomous analysis of the attacker lifecycle.

The first step in this process is to model the attacker lifecycle in the knowledge base. Figure 12 showed how the concept of the attacker lifecycle and its phases can be modeled as an ontology fragment. In Figure 13, these concepts are extended by mapping APT1 activity to its own instance of the attacker lifecycle for the APT1 initial compromise and APT1 gain foothold phases.

For each phase of the attacker lifecycle, specific APT1 malware is associated to it. By further associating malware knowledge (Figure 14) with each piece of malware, I can further extend this ontology fragment until we have a detailed model of APT1's entire attacker lifecycle and can then use this knowledge to perform follow-on analysis when one phase of an attack is detected.

Generally speaking, the phases of the attacker lifecycle must progress in order. It is not uncommon for some phases to be skipped, but it is very rare for all of them to be skipped. This means if a later phase of the attacker lifecycle is detected, it increases the likelihood evidence of previous phases exist on the network. Analysis of previous phases

must be conducted, giving more weight to the evidence because of confirmed existence of a later attacker lifecycle phase. If the analysis scenarios are chained back to the reconnaissance phase, the root cause of the attack can be identified.

When a phase of an attack is detected it also means there may be evidence of later phases of the attack. Analysis of later phases should be run, using the likelihood of the detected phase as the maximum likelihood of the later phase. If this analysis is followed to the complete mission phase, it can be determined whether or not damage has occurred as a result of the attack. An overview of this process is shown in Figure 44.

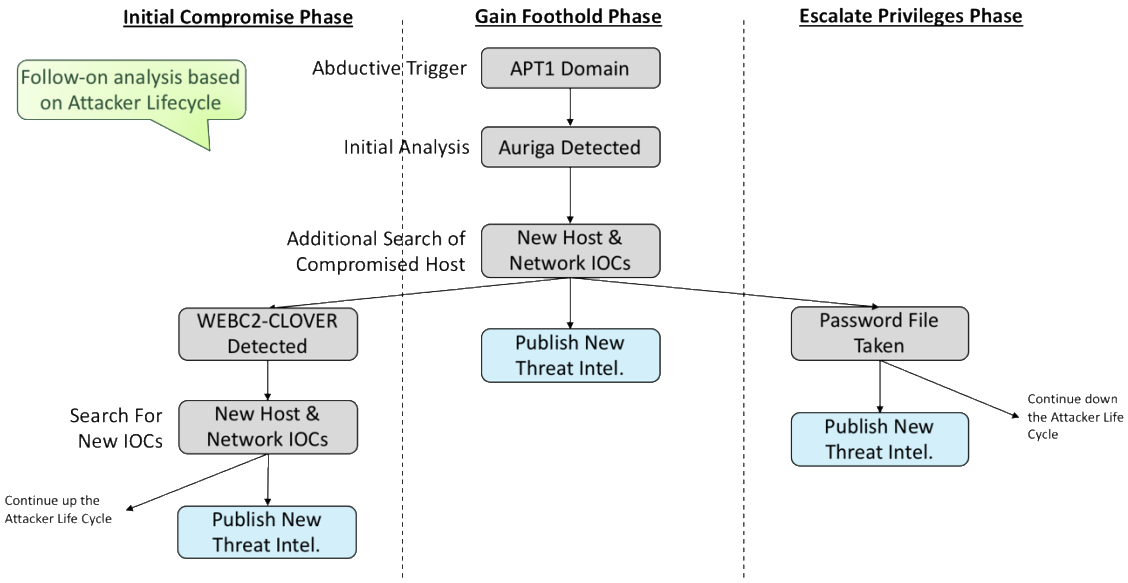

Figure 44 Example of autonomous attacker lifecycle analysis

A beneficial side effect of this analysis is the automatic identification of new threat intelligence. If the cognitive agents conduct analysis of earlier or later phases of

the attack and determine with high probability they occurred, any evidence which was found but did not match evidence already known as being associated with an APT group can now be associated with the APT group and published as threat intelligence. This means this new evidence can be used as primary indicators for detection of attacker activity using security tools such as BRO.

REFERENCES

Abdoli, F., & Kahani, M. (2008). Using Attacks Ontology in Distributed Intrusion Detection System. In T. Sobh (Ed.), *Advances in Computer and Information Sciences and Engineering* (pp. 153–158). Springer Netherlands.

Abdoli, F., & Kahani, M. (2009). Ontology-based distributed intrusion detection system. *Computer Conference, 2009. CSICC 2009. 14th International CSI*, 65–70. https://doi.org/10.1109/CSICC.2009.5349372

Ahmed, S. S. (2014). *Intrusion Alert Analysis Framework Using Semantic Correlation* (Thesis).

An Wang, J., Guo, M. M., & Camargo, J. (2010). An Ontological Approach to Computer System Security. *Inf. Sec. J.: A Global Perspective*, *19*(2), 61–73. https://doi.org/10.1080/19393550903404902

Ashri, R., Payne, T., Marvin, D., Surridge, M., & Taylor, S. (2004). Towards a semantic web security infrastructure. *Proc. of Semantic Web Services*, 49–64.

Aslam, T., Krsul, I., & Spafford, E. H. (1996). *Use of a taxonomy of security faults*.

Barnum, S. (2014). *Standardizing Cyber Threat Intelligence Information with the Structured Threat Information eXpression (STIX™)*. MITRE Corporation.

Beats. (2019). Beats: Data Shippers for Elasticsearch | Elastic. Retrieved April 21, 2019, from Beats website: https://www.elastic.co/products/beats

Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K. (2014). Network Anomaly Detection: Methods, Systems and Tools. *IEEE Communications Surveys Tutorials*, *16*(1), 303–336. https://doi.org/10.1109/SURV.2013.052213.00046

Blanco, C., Lasheras, J., Valencia-Garcia, R., Fernandez-Medina, E., Toval, A., & Piattini, M. (2008). A Systematic Review and Comparison of Security Ontologies. *Third International Conference on Availability, Reliability and Security, 2008. ARES 08*, 813–820. https://doi.org/10.1109/ARES.2008.33

Blanco, Carlos, Lasheras, J., Fernández-Medina, E., Valencia-García, R., & Toval, A. (2011). Basis for an integrated security ontology according to a systematic review of existing proposals. *Computer Standards & Interfaces*, *33*(4), 372–388. https://doi.org/10.1016/j.csi.2010.12.002

CarbonBlack. (2019). Carbon Black | Transforming Endpoint Security with Big Data Analytics. Retrieved April 21, 2019, from Carbon Black website: https://www.carbonblack.com/

Center for Strategic International Studies. (2014). *Net Losses: Estimating the Global Cost of Cybercrime*. Intel Security.

Chuvakin, A. (2018, March 6). The Best Starting Technology for Detection? Retrieved March 6, 2018, from Anton Chuvakin website: https://blogs.gartner.com/anton-chuvakin/2018/03/06/the-best-starting-technology-for-detection/

CNN. (2015). How Russians hacked the White House - CNN.com. Retrieved April 13, 2015, from CNN website: http://www.cnn.com/2015/04/07/politics/how-russians-hacked-the-wh/index.html

Cohen, L. J. (1977). *The Probable and the Provable*. Oxford: Clarendon Press.

Cohen, L. J. (1989). *An Introduction to the Philosophy of Induction and Probability*. Oxford: Clarendon Press.

Colace, F., De Santo, M., & Ferrandino, S. (2012). A Slow Intelligent Approach for the Improvement of Intrusion Detection and Prevention System. *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, 130–137. https://doi.org/10.1109/IMIS.2012.128

Conolly, J., Davidson, M., Richard, M., & Skorupka, C. (2012). *The Trusted Automated eXchange of Indicator Information (TAXII™)*. MITRE Corporation.

Cybersecurity Ventures. (2015). The Cybersecurity Market Report covers the business of cybersecurity, including market sizing and industry forecasts, spending, notable M&A and IPO activity, and more. Retrieved March 14, 2015, from Cybersecurity Ventures website: http://cybersecurityventures.com/cybersecurity-market-report/

CybOX. (2019). CybOX - About CybOX. Retrieved April 22, 2019, from CybOX website: https://cybox.mitre.org/about/

DARPA. (1999). DARPA Intrusion Detection Evaluation. Retrieved from DARPA Intrusion Detection Evaluation website: http://www.ll.mit.edu/ideval/index.html

Denker, G., Kagal, L., & Finin, T. (2005). Security in the Semantic Web using OWL. *Information Security Technical Report*, *10*(1), 51–58. https://doi.org/10.1016/j.istr.2004.11.002

Denker, G., Kagal, L., Finin, T., Paolucci, M., & Sycara, K. (2003). Security for DAML Web Services: Annotation and Matchmaking. In D. Fensel, K. Sycara, & J. Mylopoulos (Eds.), *The Semantic Web - ISWC 2003* (pp. 335–350). Springer Berlin Heidelberg.

Depren, O., Topallar, M., Anarim, E., & Ciliz, M. K. (2005). An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. *Expert Systems with Applications*, *29*(4), 713–722. https://doi.org/10.1016/j.eswa.2005.05.002

Eckmann, S. T., Vigna, G., & Kemmerer, R. A. (2002). STATL: An attack language for state-based intrusion detection. *Journal of Computer Security*, *10*(1), 71–103.

Elasticsearch. (2015). Elasticsearch: RESTful, Distributed Search & Analytics | Elastic. Retrieved September 5, 2015, from Elasticsearch website: https://www.elastic.co/products/elasticsearch

EnCase. (2017). EnCase Endpoint Investigator - Remote Digital Investigation Solution. Retrieved July 20, 2017, from EnCase website: https://www.guidancesoftware.com/encase-endpoint-investigator

Fenz, S., & Ekelhart, A. (2009). Formalizing Information Security Knowledge. *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, 183–194. https://doi.org/10.1145/1533057.1533084

Filebeat. (2018). Filebeat. Retrieved July 4, 2018, from Filebeat website: https://www.elastic.co/products/beats/filebeat

FireEye. (2015). Threat Intelligence – Malware Analysis. Retrieved September 5, 2015, from FireEye website: https://www.fireeye.com/products/dynamic-threat-intelligence.html

García-Crespo, Á., Gómez-Berbís, J. M., Colomo-Palacios, R., & Alor-Hernández, G. (2011). SecurOntology: A semantic web access control framework. *Computer Standards & Interfaces*, *33*(1), 42–49. https://doi.org/10.1016/j.csi.2009.10.003

Gomes, H., Zúquete, A., & Dias, G. P. (2009). *An overview of security ontologies*.

GRR. (2019). *GRR Rapid Response: remote live forensics for incident response: google/grr* [Python, GRR]. GRR.

Gruber, T. R. (1993). A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, (5.2), 199–220.

Gupta, A., Kuppili, P., Akella, A., & Barford, P. (2009). An empirical study of malware evolution. *Communication Systems and Networks and Workshops, 2009. COMSNETS 2009. First International*, 1–10. https://doi.org/10.1109/COMSNETS.2009.4808876

Hansman, S., & Hunt, R. (2005). A taxonomy of network and computer attacks. *Computers & Security*, *24*(1), 31–43. https://doi.org/10.1016/j.cose.2004.06.011

Hochberg, J., Jackson, K., Stallings, C., Mcclary, J. F., Dubois, D., & Ford, J. (1993). NADIR: An Automated System for Detecting Network Intrusion and Misuse. *Comput. Secur.*, *12*(3), 235–248. https://doi.org/10.1016/0167-4048(93)90110-Q

Hogland, G., & Butler, J. (2005). *Rootkits: Subverting the Windows Kernel*. Addison-Wesley Professional.

Hung, S.-S., & Shing-Min Liu, D. (2008). A User-oriented Ontology-based Approach for Network Intrusion Detection. *Comput. Stand. Interfaces*, *30*(1–2), 78–88. https://doi.org/10.1016/j.csi.2007.07.008

Hutchins, E. M., Cloppert, M. J., & Amin, R. M. (2011). Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, *1*, 80.

Iannacone, M., Bohn, S., Nakamura, G., Gerth, J., Huffer, K., Bridges, R., … Goodall, J. (2015). Developing an Ontology for Cyber Security Knowledge Graphs. *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, 12:1–12:4. https://doi.org/10.1145/2746266.2746278

Igure, V., & Williams, R. (2008). Taxonomies of attacks and vulnerabilities in computer systems. *IEEE Communications Surveys Tutorials*, *10*(1), 6–19. https://doi.org/10.1109/COMST.2008.4483667

Isaza, G. A., Castillo, A. G., & Duque, N. D. (2009). An Intrusion Detection and Prevention Model Based on Intelligent Multi-Agent Systems, Signatures and Reaction Rules Ontologies. In Y. Demazeau, J. Pavón, J. M. Corchado, & J. Bajo (Eds.), *7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009)* (pp. 237–245). Springer Berlin Heidelberg.

Isaza, G., Castillo, A., López, M., & Castillo, L. (2009). Towards Ontology-Based Intelligent Model for Intrusion Detection and Prevention. In Á. Herrero, P. Gastaldo, R. Zunino, & E. Corchado (Eds.), *Computational Intelligence in Security for Information Systems* (pp. 109–116). Springer Berlin Heidelberg.

iSIGHT Partners. (2015). ThreatScape® Threat Intelligence Platform. Retrieved September 5, 2015, from iSIGHT Partners website: http://www.isightpartners.com/products/threatscape/

Kagal, L., Finin, T., Paolucci, M., Srinivasan, N., Sycara, K., & Denker, G. (2004). Authorization and privacy for semantic Web services. *IEEE Intelligent Systems*, *19*(4), 50–56. https://doi.org/10.1109/MIS.2004.23

Kemmerer, R. A., & Vigna, G. (2002). Intrusion Detection: A Brief History and Overview (Supplement to Computer Magazine). *Computer*, *35*(4), 27–30.

Killourhy, K. S., Maxion, R. A., & Tan, K. M. C. (2004). A defense-centric taxonomy based on attack manifestations. *2004 International Conference on Dependable Systems and Networks*, 102–111. https://doi.org/10.1109/DSN.2004.1311881

Krügel, C., & Toth, T. (2002). Distributed Pattern Detection for Intrusion Detection.pdf. *Proceedings of the Network and Distributed System Security Symposium*. Presented at the Proceedings of the Network and Distributed System Security Symposium.

Landwehr, C. E., Bull, A. R., McDermott, J. P., & Choi, W. S. (1994). A Taxonomy of Computer Program Security Flaws. *ACM Comput. Surv.*, *26*(3), 211–254. https://doi.org/10.1145/185403.185412

Li, W., & Tian, S. (2010). An ontology-based intrusion alerts correlation system. *Expert Systems with Applications*, *37*(10), 7138–7146. https://doi.org/10.1016/j.eswa.2010.03.068

Ligh, M. H., Case, A., Levy, J., & Walters, A. (2014). *The Art of Memory Forensics* (1st ed.). Wiley.

Lin, J.-L., Wang, X. S., & Jajodia, S. (1998). Abstraction-based misuse detection: high-level specifications and adaptable strategies. *11th IEEE Computer Security Foundations Workshop, 1998. Proceedings*, 190–201. https://doi.org/10.1109/CSFW.1998.683169

Lindqvist, U., & Jonsson, E. (1997). How to systematically classify computer security intrusions. , *1997 IEEE Symposium on Security and Privacy, 1997. Proceedings*, 154–163. https://doi.org/10.1109/SECPRI.1997.601330

Lindqvist, U., & Porras, P. A. (1999). Detecting computer and network misuse through the production-based expert system toolset (P-BEST). *Proceedings of the 1999 IEEE Symposium on Security and Privacy, 1999*, 146–161. https://doi.org/10.1109/SECPRI.1999.766911

Mandiant. (2013). *APT1 - Exposing One of China's Cyber Espionage Units*.

Martimiano, A. F. M., & Moreira, E. S. (2005). An owl-based security incident ontology. *Proceedings of the Eighth International Protege Conference*, 43–44.

Martimiano, L. A. F., & dos Santos Moreira, E. (2006). The Evaluation Process of a Computer Security Incident Ontology. *WONTO*.

Martimiano, L. A., & Moreira, E. S. (2005). Using ontologies to assist security management. *Proceedings of the 8th International Protégé Conference*.

Martinez, C. A., Echeverri, G. I., & Sanz, A. G. C. (2010). Malware detection based on Cloud Computing integrating Intrusion Ontology representation. *2010 IEEE Latin-American Conference on Communications (LATINCOM)*, 1–6. https://doi.org/10.1109/LATINCOM.2010.5641013

McHugh, J. (2000). The 1998 Lincoln Laboratory IDS Evaluation. In H. Debar, L. Mé, & S. F. Wu (Eds.), *Recent Advances in Intrusion Detection* (pp. 145–161). Springer Berlin Heidelberg.

Meckl, S., Tecuci, G., Boicu, M., & Marcu, D. (2015). Towards an Operational Semantic Theory of Cyber Defense Against Advanced Persistent Threats. *Proceedings of the 10th International Conference on Semantic Technologies for Intelligence, Defense, and Security (STIDS)*, pp. 58–65, Fairfax, VA, 18-20 November. http://lac.gmu.edu/publications/2015/APT-LAC.pdf

Meckl, S., Tecuci, G., Marcu, D., & Boicu, M. (2018). Integrating Collaborative Cognitive Assistants Into Cybersecurity Operations Centers. *Proceedings of the 2018 AAAI Fall Symposium "Adversary-Aware Learning Techniques and Trends in Cybersecurity"*, October 18-20, Arlington, VA, Technical Report, AAAI Press, Palo Alto, CA, CEUR Workshop Proceedings, Vol.2269, pp.28-35, http://ceur-ws.org/Vol-2269/.

Meckl, S., Tecuci, G., Marcu, D., Boicu, M., & Zaman, A. B. (2017). Collaborative Cognitive Assistants for Advanced Persistent Threat Detection. *Proceedings of the 2017 AAAI Fall Symposium "Cognitive Assistance in Government and Public Sector Applications,"* pp.171-178, Arlington, VA, 9-11 November, Technical Reports FS-17-01-FS-17-05, AAAI Press, Palo Alto, CA.. http://lac.gmu.edu/publications/2017/Cyber-in%20vol%202017.pdf

Meier, M. (2004). A model for the semantics of attack signatures in misuse detection systems. In *Information Security* (pp. 158–169). Springer.

Meier, M., Bischof, N., & Holz, T. (2002). SHEDEL—A Simple Hierarchical Event Description Language for Specifying Attack Signatures. In *Security in the Information Society* (pp. 559–571). Springer.

Mila. (2013, March 3). Mandiant APT1 samples categorized by malware families. Retrieved April 5, 2015, from contagio website: http://contagiodump.blogspot.com/2013/03/mandiant-apt1-samples-categorized-by.html

Mitra, P., Pan, C.-C., Liu, P., & Atluri, V. (2006). Privacy-preserving Semantic Interoperation and Access Control of Heterogeneous Databases. *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, 66–77. https://doi.org/10.1145/1128817.1128831

MITRE ATT&CK™. (2019). Retrieved October 11, 2019, from https://attack.mitre.org/

More, S., Matthews, M., Joshi, A., & Finin, T. (2012). A Knowledge-Based Approach to Intrusion Detection Modeling. *2012 IEEE Symposium on Security and Privacy Workshops (SPW)*, 75–81. https://doi.org/10.1109/SPW.2012.26

MuleSoft. (2016, December 7). What is REST API Design? Retrieved July 4, 2018, from MuleSoft website: https://www.mulesoft.com/resources/api/what-is-rest-api-design

Naldurg, P., Sen, K., & Thati, P. (2004). A Temporal Logic Based Framework for Intrusion Detection. In D. de Frutos-Escrig & M. Núñez (Eds.), *Formal Techniques for Networked and Distributed Systems – FORTE 2004* (pp. 359–376). Springer Berlin Heidelberg.

Northcutt, S., Beale, J., Baker, A., Esler, J., & Kohlenberg, T. (2007). *Snort: IDS and IPS toolkit*. Syngress Press.

Obrst, L., Chase, P., & Markeloff, R. (2012). Developing an Ontology of the Cyber Security Domain. *Proceedings of the Seventh International Conference on Semantic Technologies for Intelligence, Defense, and Security*, *966*. George Mason University: CEUR.

O'Gorman, B., Wueest, C., O'Brien, D., Cleary, G., Lau, H., Power, J.-P., … Wallace, S. (2019). *Internet Security Threat Report*. *24*, 61.

OpenIOC. (2019). OpenIOC: Back to the Basics « OpenIOC: Back to the Basics. Retrieved April 22, 2019, from OpenIOC website: https://www.fireeye.com/blog/threat-research/2013/10/openioc-basics.html

Packetbeat. (2019). Packetbeat: Network Analytics Using Elasticsearch | Elastic. Retrieved April 21, 2019, from Packetbeat website: https://www.elastic.co/products/beats/packetbeat

Pagliery, J. (2015, August 5). The inside story of the biggest hack in history. Retrieved June 19, 2018, from CNNMoney website: http://money.cnn.com/2015/08/05/technology/aramco-hack/index.html

Paxson, V. (1999). Bro: a system for detecting network intruders in real-time. *Computer Networks*, *31*(23–24), 2435–2463. https://doi.org/10.1016/S1389-1286(99)00112-7

Peddabachigari, S., Abraham, A., Grosan, C., & Thomas, J. (2007). Modeling intrusion detection system using hybrid intelligent systems. *Journal of Network and Computer Applications*, *30*(1), 114–132. https://doi.org/10.1016/j.jnca.2005.06.003

Ponemon Institute. (2017, June). *2017 Cost of Data Breach Study*. Ponemon Institute.

Raskin, V., Hempelmann, C. F., Triezenberg, K. E., & Nirenburg, S. (2001). Ontology in Information Security: A Useful Theoretical Foundation and Methodological Tool.

Proceedings of the 2001 Workshop on New Security Paradigms, 53–59. https://doi.org/10.1145/508171.508180

S. Staniford-chen, S. C. (1998). *GrIDS--A GRAPH BASED INTRUSION DETECTION SYSTEM FOR LARGE NETWORKS*.

Saad, S., & Traore, I. (2010). Method ontology for intelligent network forensics analysis. *2010 Eighth Annual International Conference on Privacy Security and Trust (PST)*, 7–14. https://doi.org/10.1109/PST.2010.5593235

Saad, S., Traore, I., & Brocardo, M. L. (2014). Context-aware intrusion alerts verification approach. *2014 10th International Conference on Information Assurance and Security (IAS)*, 53–59. https://doi.org/10.1109/ISIAS.2014.7064620

Saad, Sherif, & Traore, I. (2010). Ontology-based Intelligent Network-Forensics Investigation. *SEDE*, 313–319.

Saad, Sherif, & Traore, I. (2011). A semantic analysis approach to manage ids alerts flooding. *Information Assurance and Security (IAS), 2011 7th International Conference On*, 156–161. IEEE.

Sabahi, F., & Movaghar, A. (2008). Intrusion Detection: A Survey. *2008 Third International Conference on Systems and Networks Communications*, 23–26. https://doi.org/10.1109/ICSNC.2008.44

Salahi, A., & Ansarinia, M. (2013). Predicting Network Attacks Using Ontology-Driven Inference. *ArXiv Preprint ArXiv:1304.0913*.

Schumacher, M. (2003). Toward a Security Core Ontology. In *Lecture Notes in Computer Science*: *Vol. 2754*. *Security Engineering with Patterns* (pp. 87–96). Springer Berlin Heidelberg.

Snapp, S. R., Brentano, J., Dias, G. V., Goan, T. L., Heberlein, L. T., Ho, C.-L., … others. (1991). DIDS (distributed intrusion detection system)-motivation, architecture, and an early prototype. *Proceedings of the 14th National Computer Security Conference*, *1*, 167–176. Citeseer.

Splunk. (2015). Operational Intelligence, Log Management, Application Management, Enterprise Security and Compliance. Retrieved September 5, 2015, from Splunk website: http://www.splunk.com/

Stewart, J. (2003a, July 8). Sobig.a and the Spam You Received Today. Retrieved May 15, 2015, from joestewart.org website: http://www.joestewart.org/sobig.html

Stewart, J. (2003b, July 8). Sobig.e - Evolution of the Worm. Retrieved May 15, 2015, from joestewart.org website: http://www.joestewart.org/sobig-e.html

Symantec. (2015). DeepSight Intelligence. Retrieved from DeepSight Intelligence website: http://www.symantec.com/deepsight-products/

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A Detailed Analysis of the KDD CUP 99 Data Set. *Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications*, 53–58. Piscataway, NJ, USA: IEEE Press.

Tecuci, G., Schum, D. A., Boicu, M., & Marcu, D. (2011). *Introduction to Intelligence Analysis: A Hands-on Approach with TIACRITIS*. Learning Agents Center, George Mason University.

Tecuci, G., Schum, D. A., Marcu, D., & Boicu, M. (2014). Computational approach and cognitive assistant for evidence-based reasoning in intelligence analysis. *International Journal of Intelligent Defence Support Systems*, *5*(2), 146–172. https://doi.org/10.1504/IJIDSS.2014.059976

Tecuci, G., Schum, D. A., Marcu, D., & Boicu, M. (2015). COGENT: Cognitive Agent for Cogent Analysis. *Proceedings of the 2015 AAAI Fall Symposium*. Presented at the "Cognitive Assistance in Government and Public Sector Applications," Arlington, VA. http://lac.gmu.edu/publications/2015/Cogent-overview.pdf

Tecuci, G., Marcu, D., Boicu, M., & Schum, D. A. (2016a). *Knowledge Engineering: Building Cognitive Assistants for Evidence-based Reasoning*. Cambridge University Press. https://www.cambridge.org/core/books/knowledge-engineering/B5D4AAED35FCEE759B79F9CF6A66FE06

Tecuci, G., Schum, D. A., Marcu, D., & Boicu, M. (2016b). *Intelligence Analysis as Discovery of Evidence, Hypotheses, and Arguments: Connecting the Dots*, Cambridge University Press. https://doi.org/10.1017/CBO9781316388488

Tecuci G., Kaiser L., Marcu D., Uttamsingh C., Boicu M. (2018a). Evidence-based Reasoning in Intelligence Analysis: Structured Methodology and System, *Computing in Science and Engineering*, 20(6) 9-21, November/December.

Tecuci G., Meckl S., Marcu D., Boicu M. (2018b). Evidence-based Detection of Advanced Persistent Threats, Special Issue on Evidence-based Reasoning and Applications, *Computing in Science and Engineering*, 20(6) 54-65, November/December.

Tecuci G., Meckl S., Marcu D., Boicu M. (2019a). Instructable Cognitive Agents for Autonomous Evidence-Based Reasoning. *Advances in Cognitive Systems*, Vol. 8, 2019. Also in *Proceedings of the Seventh Annual Conference on Advances in Cognitive*

Systems, Technical Report Number COLAB^2-TR-4, pp.183-204, August 2-5, 2019, Massachusetts Institute of Technology, Cambridge, MA.

Tecuci G., Marcu D., Boicu M., Meckl S., Uttamsingh C. (2019b). Toward a Computational Theory of Evidence-Based Reasoning for Instructable Cognitive Agents, *Proceedings of the 2019 AAAI Fall Symposium "Artificial Intelligence in Government and Public Sector,"* Arlington, VA, November 7-9. http://lac.gmu.edu/publications/2019/Tecuci-EBR-2019.pdf

Tsoumas, B., & Gritzalis, D. (2006). Towards an Ontology-based Security Management. *20th International Conference on Advanced Information Networking and Applications, 2006. AINA 2006*, *1*, 985–992. https://doi.org/10.1109/AINA.2006.329

Undercoffer, J., Joshi, A., Finin, T., & Pinkston, J. (2003). Using DAML+OIL to classify intrusive behaviours. *The Knowledge Engineering Review*, *18*(03), 221–241. https://doi.org/DOI:10.1017/S0269888904000049

Undercoffer, J., Joshi, A., & Pinkston, J. (2003). Modeling Computer Attacks: An Ontology for Intrusion Detection. In G. Vigna, C. Kruegel, & E. Jonsson (Eds.), *Recent Advances in Intrusion Detection* (pp. 113–135). Springer Berlin Heidelberg.

Undercoffer, J., Pinkston, J., Joshi, A., & Finin, T. (2004). A target-centric ontology for intrusion detection. *18th International Joint Conference on Artificial Intelligence*, 9–15.

USA TODAY. (2015). Timeline: North Korea and the Sony Pictures hack. Retrieved April 13, 2015, from USA TODAY website: http://www.usatoday.com/story/news/nation-now/2014/12/18/sony-hack-timeline-interview-north-korea/20601645/

Verizon. (2015). Learn from Verizon's 2014 Data Breach Investigations Report. Retrieved March 9, 2015, from Verizon Enterprise Solutions website: http://www.verizonenterprise.com/DBIR/2014/

Vigna, G., Eckmann, S., & Kemmerer, R. (2000). Attack Languages. *Proceedings of the IEEE Information Survivability Workshop Vol 366*. Presented at the Proceedings of the IEEE Information Survivability Workshop.

Volatility. (2015). volatilityfoundation/volatility. Retrieved May 15, 2015, from Volatility website: https://github.com/volatilityfoundation/volatility

Vorobiev, A., Han, J., & Bekmamedova, N. (2008). An Ontology Framework for Managing Security Attacks and Defences in Component Based Software Systems. *19th Australian Conference on Software Engineering, 2008. ASWEC 2008*, 552–561. https://doi.org/10.1109/ASWEC.2008.4483245

Weber, S., Karger, P. A., & Paradkar, A. (2005). A Software Flaw Taxonomy: Aiming Tools at Security. *Proceedings of the 2005 Workshop on Software Engineering for Secure Systems—Building Trustworthy Applications*, 1–7. https://doi.org/10.1145/1082983.1083209

Winlogbeat. (2019). Winlogbeat: Analyze Windows Event Logs | Elastic. Retrieved April 21, 2019, from Winlogbeat website: https://www.elastic.co/products/beats/winlogbeat

Yeung, D.-Y., & Ding, Y. (2003). Host-based intrusion detection using dynamic and static behavioral models. *Pattern Recognition*, *36*(1), 229–243. https://doi.org/10.1016/S0031-3203(02)00026-2

Zadeh, L. A. (1983). The role of fuzzy logic in the management of uncertainty in expert systems. *Fuzzy Sets and Systems*, *11*(1–3), 197–198. https://doi.org/10.1016/S0165-0114(83)80081-5

Zevin, S. (2009). *Standards for security categorization of federal information and information systems*. DIANE Publishing.

Zimmerman, C. (2014). *Ten Strategies of a World-Class Cybersecurity Operations Center*. MITRE Corporation.

BIOGRAPHY

Steven W. Meckl graduated from Owosso High School, Owosso, Michigan, in 1993. He received his Bachelor of Science in Engineering from the University of Michigan in 1998 and a Master of Science in Information Security and Assurance from George Mason University in 2010. Steve began his career as a software engineer, developing security software with a focus on cryptography for companies such as Symantec. From 2005 to 2015, he served as a Special Agent with the Federal Bureau of Investigation (FBI) where he spent his time investigating state-sponsored intrusions into critical US infrastructure. While at FBI Headquarters at the end of his tenure, Steve created and led Cyber Division's technical operations program and led FBI Cyber Division's incident response team. Steve currently serves as Senior Director of Global Operations for Symantec's Cyber Security Services business, leading a global team of incident responders, intelligence analysts, security incident analysts and security engineers who are responsible for monitoring the networks of Fortune 2000 companies.