STANFORD UNIVERSITY

Cyber Risk Management

AI-Generated Warnings of Threats

by

Isaac J. Faber

A DISSERTATION SUBMITTED TO THE
MANAGEMENT SCIENCE AND ENGINEERING DEPARTMENT
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

in the
Department of Management Science and Engineering

June 2019

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Elisabeth Pate-Cornell, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Ross Shachter

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Herbert Lin,

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

Abstract

Abstract: This research presents a warning systems model in which early-stage cyber threat signals are generated using machine learning and artificial intelligence (AI) techniques. Cybersecurity is most often, in practice, reactive. Based on the manual forensics of machine-generated data by humans, security efforts only begin after a loss has taken place. The current security paradigm can be significantly improved. Cyber-threat behaviors can be modeled as a set of discrete, observable steps called a 'kill chain.' Data produced from observing early kill chain steps can support the automation of manual defensive responses before an attack causes losses. However, early AI-based approaches to cybersecurity have been sensitive to exploitation and overly burdensome false positive rates resulting in low adoption rates and low trust from human experts. To address this problem, this research presents a collaborative decision paradigm with machines making low-impact/high-confidence decisions based on human risk preferences and uncertainty thresholds. Human experts only evaluate signals generated by the AI when decisions exceed these thresholds. This approach unifies core concepts from the disciplines of decision analysis and machine learning by creating a super-agent. An early warning system using these techniques has the potential to avoid more severe downstream consequences by disrupting threats at the beginning of the kill chain.

Keywords: cybersecurity, machine learning, artificial intelligence, risk analysis, decision analysis, early warning systems

Acknowledgements

This dissertation was developed over an extended three year period of learning and transition. It is made possible only by the excellent assistance of my advisor and mentor Professor Elisabeth Paté-Cornell. Also, I owe a debt to the faculty of the Management Science and Engineering Department in the School of Engineering at Stanford University for building my intellectual foundation. Specifically, Professor Ronald Howard, Associate Professor Ross Schachter, Herb Lin, Lee Ross, and the faculty of the Decision and Risk Analysis Group, who helped sharpen my knowledge in preparation for this effort. Additionally, I would not have been able to complete this task without support from family. Specifically my beautiful wife Kerry, to whom I owe everything, and my four childern Jaycee Mae, Julianne, Austin, and Annaliegh.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Opening Remarks

This thesis presents a cybersecurity risk management model with early-stage cyber-threat warnings generated using machine learning techniques. This approach leverages thresholds to balance between decisions by artificial intelligence (AI) and expert-human intervention. Decision thresholds are set using the concepts of impact and uncertainty. Decisions with significant organizational impact are defined by the organization a priori. In kind, as the AI learns there will be areas in the decision space with high degrees of uncertainty (from the AIs perspective). The source of this uncertainty can be the system (aleatory) or the AI itself (epistemic). Warnings are generated using impact and uncertainty thresholds which signal the expert-human to become more directly involved.

The concept of a 'gate-set' provides a framework defining a clear set of access control decisions which influence both the security and the usefulness of a defended system. The defender has control over a series of gates (the gate-set) which they can either leave open or closed. The defender starts from the point of limited (or no) information. The goal of the system's defender is to find an optimal gate-set policy as quickly as possible.

The fundamental cybersecurity problem is that organizations must defend their information systems from threats[1]. The motivation behind defending systems is that organizations rely on them for operations and to store confidential data. Threats are third-party entities who want to exploit the system and data for purposes at odds with the owner of the system. If threats are successful, organizations realize disruptions in operations or loss of valuable information. This potential loss is an organizational risk than needs to be managed.

[1]Cyber-threats are actors that intentionally seek to inflict damage on, or steal from, a defended information system.

However, it can be difficult for defenders to effectively protect systems due, in part, to the dynamic nature of threats and the large amount of security data produced by defensive activity. Furthermore, the data generated by these same devices contain a low signal-to-noise ratio. For this problem, the use of automated security decisions through the application of AI-based technologies may be a solution. An AI-based machine (a robot) can scan quickly through large amounts of data and then pass consequential, complicated, or low confidence decisions to a human expert by exception.

This research shows how to improve security decisions, through intelligent access control, by leveraging the integration of machine learning and expert analysis. The super-agent concept, proposed in Chapter 3, is a unified entity in which a collaborative and co-operative decision-making environment is shared between expert-humans and AI-based robots. This idea has far-reaching implications beyond cybersecurity. The super-agent exists in systems where an AI-based robot and an expert-human have joint authority over the same sequential decision. Each decision has different characteristics of potential impacts and uncertainties. The super-agent must have a method to determine who among the robot and expert-human should perform the individual decision based on the risk attitude and preferences.

AI-based based techniques have become integrated with many aspects of daily life, and therefore essential and unexplored considerations arise. Within cybersecurity settings, this research also examines the problem of decision making in allocating a task to a robot versus a human. This compromise is essential when a massive amount of complex data requires both machine and human intervention. The concern with a robot is that it may lace confidence or may take a costly organizational risk if left unsupervised. These concerns have imposed limitations on the use of AI-based technologies in practical applications. However, these technologies are not commonly used for operations and integrating them into current practice will have challenges.

This paper focuses on active defense, as shown in Figure 2.1. Active defense is, in current practice, reactive and manual [1]. Security events are often identified only after substantial organizational damage has occurred [2]. In addition, attacks and defensive responses can be complicated and opaque to non-experts. There is a large technical disparity between system owners and defenders. Owing, in part to this disparity, system owners may feel compelled to invest heavily in security practices. However, the security paradigm based on the manual forensics of machine-generated data by expert-humans can be improved. In recent years, the increase in computing power and data availability for expert analysis have made it possible to integrate AI with traditional security practices.

The cybersecurity landscape has evolved rapidly to become a critical sphere of modern life. Human society relies on information systems more than ever before with some nations adopting sophisticated cyber capabilities. Global spending on cybersecurity is at an all-time high for industries and governments[2]. Figure 1.1 shows the trend of growth in spending on internal and external spending by private organizations worldwide [3].

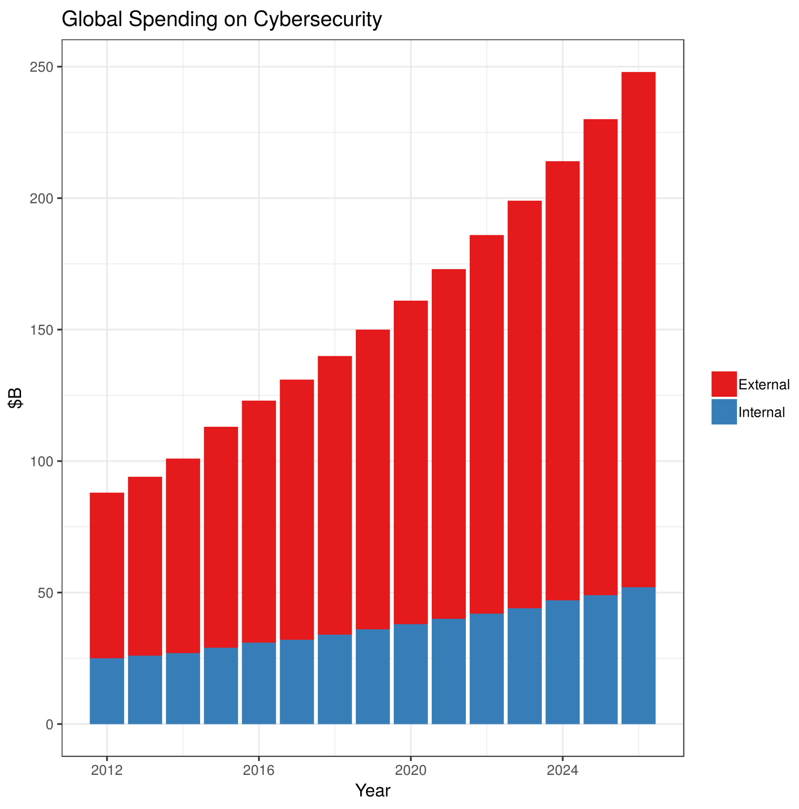


FIGURE 1.1: Global Cybersecurity Spending.

With an increase in cybersecurity spending, the critical question to be asked is 'how will the cybersecurity evolve?' New methods of analysis are needed to ensure the best resource allocation. In this context, preemptive action is an important topic of interest[3]. Response to large-scale incidents is costly [2][4]. Therefore, it is reasonable to ask: Can a decision maker identify and adapt to threats before they fully present themselves? Anticipating the future remains elusive in all domains. However, if actions can be taken based on the attributes available during the early stages of a complex cyber-attack, it might be possible to adapt and avoid some damages. Therefore, there is a clear need for a warning system to improve defensive practices.

[2]Spending is a combination of internal (Security personnel) and external (hardware and software solutions for the organization. Recent trends have seen dramatic growth in spending on external products.

[3]Preemptive action in cybersecurity is changing or 'hardening' a system before organizational damage is realized. Damage can manifest in several ways such as loss of data or loss of productivity.

[4]Incidents are cyber security events that result in some loss or disruption of an information system.

Various actors are continually probing devices connected to the internet.[5] Some of this behavior is innocent; some is intended to gather information. However, much of the probing is explicitly done to find weaknesses within the cyberspace. In Chapter 4, the nature of this probing is explored using a real-world dataset collected for his research. It is essential to point out that all cyber-attacks start with some variation of this first step. Network defenders might ignore probing as they get overwhelmed by large amounts of data being exchanged with frequent attempts. This type of early-stage threat behavior is rarely explored and is considered routine activity by defenders [4]. Tasks such as evaluating probing activity can now be done, in part, by an AI based machine. This enables early warnings to be generated in such contexts which can feed a risk management process.

The model presented in Chapter 3 fits into the interface between cybersecurity, warning systems, and machine learning. The modern understanding of cyber systems has changed over the past few years to encompass a broad range of actors, intents, and capabilities. Traditional approaches are not enough to understand the real-world defense of networked systems. However, warning systems are well understood and are effectively deployed in many high-risk systems, such as fire and natural disasters, which allow for the prevention of significant loss. While traditional warning systems are not easily applied to cybersecurity, they do provide a philosophical foundation as a starting point. Because sophisticated attacks take place in stages, the information related to the detection and identification of these threats at early stages could be used to stop the progression of an attack[6]. The task of generating cyber threat warnings can be done using machine learning techniques.

Warning systems can be built using machine learning models based on threat actor behavior techniques. Actors within an information system exhibit behavior that manifests as a digital signature in data recorded by security devices[7]. The patterns within this digital signature can be used to probabilistically categorize activities and actors into useful classifications that feed a risk management processes. Patterns are then interpreted as signals and applied within a warning system.

Cybersecurity requires a detection technique that encompasses the distribution of threat actors thus enabling early-stage detection. Migration to behavior-based detection methods need to be developed on a robust foundation. Machine learning based behavioral

[5]Devices are internet connected general purpose computing systems such as desktops, laptops, and smartphones.

[6]Attack stages are required because of the complexity of 'hacking' a system. The more complex the information system, the more steps are needed for an attack to be successful

[7]Signature in this context is a term that refers to a statistical relationship that can be attributed to a given actor, threat or non-threat. It is also useful as a signal for warning a system owner.

detection techniques can suffer from high false positive rates. However, recent developments have produced models that can map complex multi-featured data while offering hyper-parameters for regularization to avoid over-fitting. These techniques, when applied in a warning system, create the foundation for cyber risk management.

These more advanced machine learning techniques can learn over time and, within prescribed limits, automate a subset of defensive actions on behalf of the owner of the system. This research will apply and extend these techniques using the super-agent concept. The outcome of this work also applies to other settings where AI and expert analysts work cooperatively.

1.2 Motivation

General-purpose computing devices connected to a network have become widely popular. They have impacted every aspect of modern life from industrial control systems to high powered gaming computers. The current approaches to secure devices and networks are compartmentalized. The hacks of high-profile corporations using the advanced Stuxnet virus have demonstrated that cyber-attacks are complex and sophisticated processes that exploit weaknesses across extensive information networks [5]. While cybersecurity is a significant consideration in most information systems, it is often assumed as the purview of a handful of devices or modules. The reason to conduct cybersecurity is to protect the operations and information within the systems. However, there is a compromise between the allocation of finite resources and the maximum level of protection that can be achieved.

Devices (such as host computers) needing security from external threats are protected by a set of installed clients and upstream network traffic monitoring system. This approach requires sampling from a large amount of information and relies heavily on signatures already known. On the other hand, internal security approaches tend to adopt cryptography-based technologies, such as secure socket layer (SSL), Internet Protocol Security (IPsec), and Advanced Encryption Standard (AES).[8] However, cryptographic techniques developed to protect information can often be used to exploit vulnerabilities and circumvent the external protection devices. The study of attack graphs has shown that exploitation attempts can bypass traditional security practices and devices [6]. For example, moderately complex enterprise networks that contain thousands of endpoints create a polynomial-sized growth in attack surface making it almost impossible to monitor a large set of network paths thoroughly. For this reason, security protocols direct

[8]Secure Socket Layer (SSL) is a standard to encrypt communication between a server and a browser. IPsec is a method to encrypt communication across TCP/IP networks. Advanced Encryption Standard (AES) is a standard for encrypting data.

network traffic through a small number of ‘choke points’ where visibility is possible. These choke points are called security stacks.

An essential part of cybersecurity is understanding the risk associated with unsecured network devices and with those not performing their intended function. For example, when a database sends information to an unintended recipient without the owner’s consent. While this may not seem like a significant cybersecurity threat, its complexity is exposed when realizing that information systems are not, at their core, intended to perform specific tasks. Information systems, such as databases, are general-purpose systems that are provided software commands to perform specific functions. Unlike the physical domain, where the physical and control planes are the same; information system devices, for example, intended to route traffic or perform specific services can be manipulated to perform other tasks without the user’s knowledge. In this way, the control plane of general-purpose devices is abstracted from the user and therefore creates a security risk.

The primary reason to conduct cybersecurity is the presence of ubiquitous threat of actors with a desire to manipulate the control plane of general-purpose information systems. This manipulation is typically at odds with the purposes of the system owners. A typical exploit is the co-option of the systems intended purpose. These may manifest as a loss or theft of data or a change in system behavior like the Stuxnet virus. The complex nature of such an exploit may prompt specific techniques to exhibit consistent behavior which can be recorded within the network devices and the security stack, creating an effective digital fingerprint.

The characteristics of this digital fingerprint can be leveraged to understand intricate behavioral patterns produced from various actors.[9] The idea of behavior-based network security has been studied to some extent. Some work has focused on general descriptions of behaviors, [7] while others have shown implementations based on single data streams [8]. A few authors have taken a broad survey of the topic. [9, 10] However, only a few of these methods are in use in real-world applications. In practice, most security data are used for forensic purposes or limited exploration using heuristic-based security models.

Most security models and commercial products focus on individual components of cybersecurity, such as network devices, including firewalls, intrusion detection/prevention or endpoint add-ons, such as antivirus or host intrusion detection systems. The growth in external security systems spending has revealed this preference (shown in Figure 1.1). Nearly every large organization deploys a suite of capabilities for cybersecurity from

[9]The nature of cyber actor (human) behavior over time becomes inherently non-linear and highly complex within heterogeneous data.

multiple sources. These different capabilities are, typically, incompatible and require manual configuration to exchange information among each other.

Current approaches aggregate the records from all security devices in a single database called a Security Information and Event Manager (SIEM) to detect and manage responses to cyber-attacks. The database then becomes the central reference from which cybersecurity operators gain an understanding of various attacks against their defended system. The SIEM enables an efficient manual analysis of the aggregate information. However, it does not provide a method for understanding the dynamic and stochastic nature of cyber-threats without significant human intervention. This inadequacy results in cyber-attacks being undetected for months or in some cases never detected at all.

Cybersecurity analysis mostly remains a manual process partly due to the limited literature available to understand the complexity of this multi-faceted domain. Many advanced topics, such as deep packet inspection, cryptography, signature-based methods, and behavioral analysis have been explored from the perspective of a single point in a network, and not as a holistic system. In practice, cybersecurity involves overseeing and combining data from multiple aspects within a network using limited resources both in terms of financial and human capital (analysts). This research explores the integration of AI-generated signals in early warning systems with more traditional security components that include data aggregation and manual analysis.

Common risk and decision modeling approaches are the least studied topics within cyber systems. The lack of decision modeling techniques extends to system characterization and descriptions. This is partly due to the continually evolving nature of the discipline. The concept of securing networked information systems has a floating vernacular that has shifted in the past half-decade. The topic has expanded to encompass an area that is increasing in scope. This expansion has created gaps in the existing modelling literature.

The body of knowledge of cybersecurity historically fell under the umbrella terms 'network security' or 'software security'. Network security is a discipline that includes general network defense and information (or data) security. Some topics include cryptography, internet-security protocols, user authentication, firewalls, and intrusion protection [11, 12]. On the other hand, software security methodologies focus on the defense against malicious activity through better software design [13]. Indeed, the concept of software security is well documented and accepted with security practices dating back many decades. While these approaches are fundamental, they do not capture the entire cyber domain[10].

[10]The cyber domain references the concept that information systems are an operational space and not just a tool. Actors can move, interact, and influence in this space.

Within the last few years, several machine learning techniques have become popular in the literature. In particular, deep learning algorithms have been developed using large amounts of data and high computing power enabled by specialized processors called Graphical Processing Units (GPUs). The models have accomplished impressive results in areas such as natural language processing and computer vision, making technologies like self-driving cars possible. These models are associated with the term ‘artificial intelligence.’ Several new applications of deep learning (and related methods) are being explored in cybersecurity. However, no standard approach has been adopted, either by academia or the corporate world.

1.3 Outline of thesis

This thesis is divided into five chapters as follows: Chapter one introduces the topics in cybersecurity, artificial intelligence, machine learning, and related terminologies.

Chapter two consists of a literature review of cybersecurity with details of the current state of the art leading to a discussion with an academic, industrial, and governmental interests. Existing modeling techniques are reviewed along with the current security practices with an emphasis on ‘security stack’. The expanding definition of cybersecurity is explored in conjunction with the general decision-making frameworks in other domains. Also, warning systems are reviewed and discussed with implications for cyber systems. Finally, machine learning techniques are discussed as they apply to security practices. A crucial contribution of this review is to incorporate risk and decision modeling techniques not currently reported within the security literature.

Chapter three provides detail on the model and methodology. The model is divided into sub-sections. The first sub-section provides a graphical explanation and justification for each component of the model. The second sub-section is the quantitative application which includes the formulation of actor behaviors, utility impacts, candidate machine learning techniques, and decision thresholds. Most importantly, a decision paradigm is presented within the context of intelligent access control using gate-set. The super-agent must try and find an optimal gate-set policy at the earliest.

Chapter four presents an overview of the chosen application areas. The model is applied to one contrived and one real-world problem. The test consists of a set of globally distributed honeypots which produces a stream of high-quality data on early-stage threat behavior. This chapter shows how such information can be leveraged to inform the proposed model. The dataset presented in this section was prospectively collected for this research.

Chapter five discusses findings, expands on some of the topics from the previous two chapters, and provides potential options for future work.

Chapter 2

Background

2.1 Cyber Risk

2.1.1 Domain

The scope of cybersecurity has significantly increased in recent years. The term cyber is used as a synonym for defensive as well as offensive activities (like hacking or exploitation[1]). Cyber hacking has far-reaching strategic implications for organizations and governments. Figure 2.1 illustrates the relationship between different activities within the current cyber domain.

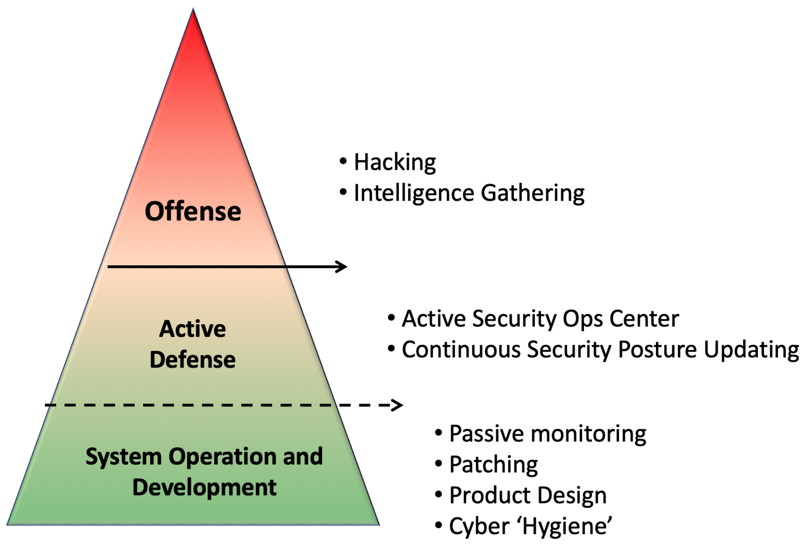


FIGURE 2.1: Cyber Domain

[1]Hacking is a term used to refer to an action intended to result in system exploitation or change in authority over the systems control plane.

The cyber domain consists of three categories with some degree of overlap. These categories are briefly described as follows:

- Offense: Activities involving exploitation or attempted exploitation of unowned systems.
- Active Defense: Activities involving defending a system with a dedicated work force and tools which control and update the defensive posture.
- System Operation and Development: Activities involving defensive security practices by non-security participants, including but not limited to, password change, token management, software design, and network design.

This research focuses on active defense, as shown in Figure 2.1. The cyber defense is considered as a layered mechanism based on principles similar to those of Open Systems Interconnection (OSI) architecture. The OSI architecture is commonly used to connect information technology devices. The range of activities, from offensive to system operations, are complemented by a suite of tools designed for specific applications. Each of these areas requires operators with a particular skill set.[2] A view of the OSI model is shown in Figure 2.2 with relevant defensive activities and tools.

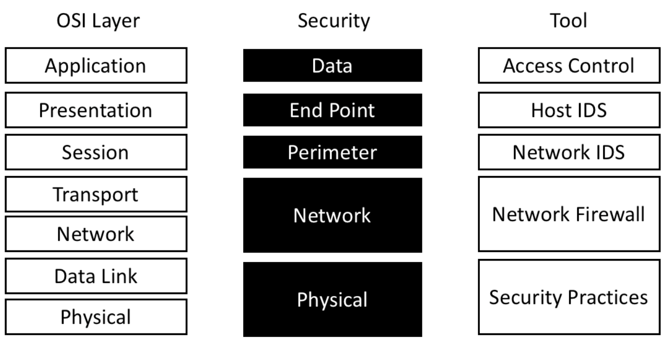


FIGURE 2.2: Security vs Network OSI model.

Capital investment is needed to train experts and to develop tools to build effective defenders. However, offensive entities often need much fewer resources to be successful at exploitation. This asymmetry leads to a problematic trade-off for network defenders.

[2]Typically, cybersecurity operators are trained in a specific application, such as operating systems, network security or application security.

For defenders pursuing a perfectly secure system results in exponentially increasing costs. This research is motivated, in part, on the compromises that defenders have to make in this regard. Instead of perfect defense they need to find the optimal security policy based on its specific organization's preferences using the current state of knowledge.

2.1.2 Evolution

It has become clear in recent years that nation-states and private actors have been participating in activities outside the bounds of the traditional network security. This participation may not be just as a defender of their networks but also as an active offensive entity with various other interests. The recent alleged attacks by Russia against Georgia [14] and North Korea's actions against critical infrastructure in South Korea [15] demonstrate this point. 'Cybersecurity' is used as an expansion of the term 'network security' as adopted by the policy makers within the United States. This term has changed to include sophisticated and persistent malicious entities. Also, the term cyber warfare has become popular due to its recognition within the media and government as an essential element of national security. Figure 2.3 shows the changes in the relative popularity of these terms based on the results of Google search over the last few years.

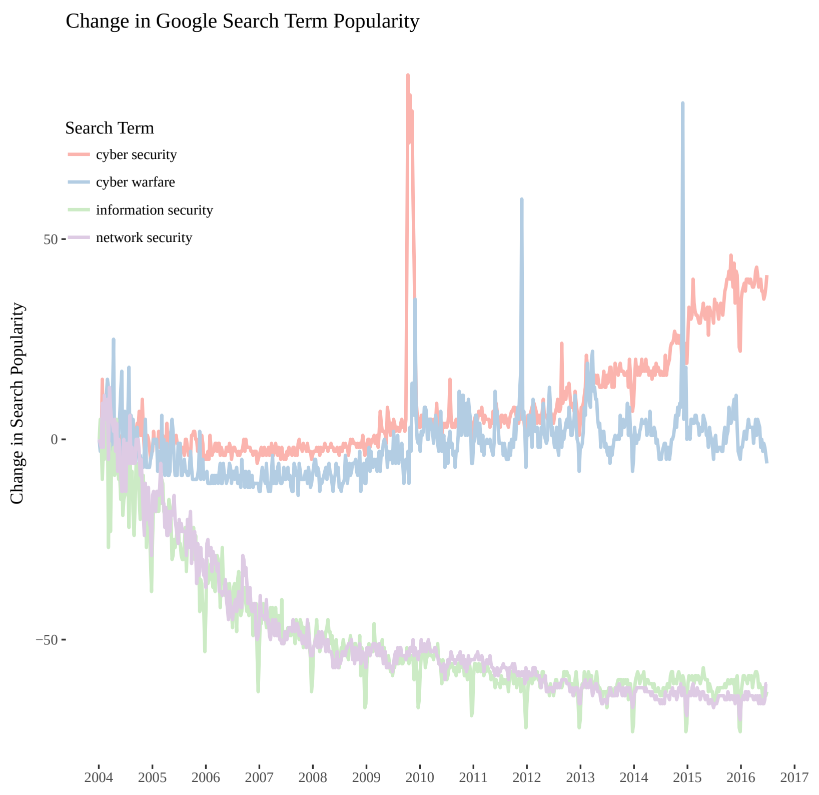


FIGURE 2.3: Google Search Term Popularity.

In 2009, the spike in popularity of the search term ‘cybersecurity’ coincided with the White House’s initial declaration in October as the ‘cybersecurity awareness month.[3] This awareness gave credibility to the descriptive term ‘cyber’ as it concerns general information security. Earlier during this year, the United States government chartered the creation of the United States Cyber Command (USCYBERCOM). Its mission states:

> USCYBERCOM plans, coordinates, integrates, synchronizes and conducts activities to: direct the operations and defense of specified Department of Defense information networks and; prepare to, and when directed, conduct full-spectrum military cyberspace operations in order to enable actions in all domains, ensure US/Allied freedom of action in cyberspace and deny the same to our adversaries.

This mission statement is a legal charter to conduct defensive and offensive cyber operations when asked by the United States government to do so. Cyber capabilities are no longer limited to intelligence purposes. Expansive military doctrinal descriptions are detailed in the United States Department of Defense joint publication JP 3-12. Other nation-states are producing similar publications for policy, planning, and guidance.

The United States is not the only nation operational in this domain. Currently, over 60 countries have commissioned similar military units [16]. Noticeable cyber warfare units are present in China, Iran, Israel, and North Korea. Also, active cyber elements that include activists, criminals or state-sponsored non-government groups have become commonplace. The international hacker community known as Anonymous has taken credit for a broad range of offensive activity ranging from defacing websites to directing attacks [17].

Russia is a significant player in the cyber domain, and is known to have attempted to influence the United States presidential election in 2016 [18]. However, Russias cyber capabilities are less formalized compared to those in other countries. In Russia, cyber activities are conducted through intermediaries instead of directly funded state-sponsored entities. For example, the Russian Business Network (RBN) is a far-reaching criminal organization known to operate a large contingent of hackers [19]. Russian officials are believed to use RBNs capabilities to conduct cyber efforts.

The evolution in cyber capabilities has led to an increase in corporate spending (Figure 1.1). Since private entities are not allowed to conduct offensive operations, they must rely on the responsive actions of law enforcement or other government agencies. This

[3]Cybersecurity Awareness Month was an annual effort led by the Department of Homeland Security: https://www.dhs.gov/national-cyber-security-awareness-month

limits private organization spending to be focused, almost exclusivity on active defense. An example from 2014 includes the hack of Sony Corp by North Korea over the release of a distasteful movie [20]. As a result, Sony corporation suffered an estimated loss of $100 million. However, they could not respond directly to North Korea and had to accept the incident as a cost of doing business over the internet. Defenders, like Sony, need better ways to design active defensive operations within the limited budget. If entities cannot respond to such cyber hacks, they must have the best defense in place. Answering difficult security questions require models to enable testing without sampling the broad set of internet data or without compromising production systems.

2.1.3 Modeling Techniques

While there are a growing number of cybersecurity based modeling techniques discussed in the literature, only a few have been applied to the real-world resource-constrained environments with a focus on risk and decision making. The changing landscape has a significant impact on the body of knowledge surrounding modeling and analysis of complex cyber systems. Many approaches are available to model modern information networks accurately. However, the transition from traditional network security to cybersecurity demands additional complex interactions within these systems.

Cyber systems are challenging to secure [21]. However, models could help in this context. Cyber models are abstract representations of the real-world counterpart. The scope and complexity of the modern internet can make it impractical to conduct experiments or observe activity on live data. For this reason, modeling approaches are needed to quickly and adaptively test new concepts. Three types of cybersecurity-based models are physical, graphical, and mathematical.

2.1.3.1 Physical Models

A physical model of a cyber system is a scaled down representation of an information network useful for conducting tests within a smaller scope. The National Cyber Range of the United States is on such example [22]. The range acts as a test bed for a broad spectrum of cyber capabilities. This facility is the largest of its type in the country. Also, other smaller labs exist for training and testing of advanced defensive and offensive techniques. For example, some national organizations hold hacking/counter-hacking competitions based on physical models of cyber systems. The most popular form of these competitions is a game called Capture the Flag [23]. Another approach in physical models uses small scale kinetic systems. For example, The SANS Institute (officially the Escal Institute of Advanced Technologies) is the most prevalent cybersecurity curriculum builder that offers a course specialized in protecting and hacking at miniature city Cyber City. Figure 2.4 shows a layout of the mock city built using Supervisory Control and Data Acquisition (SCADA) infrastructure, which is physically manipulable.

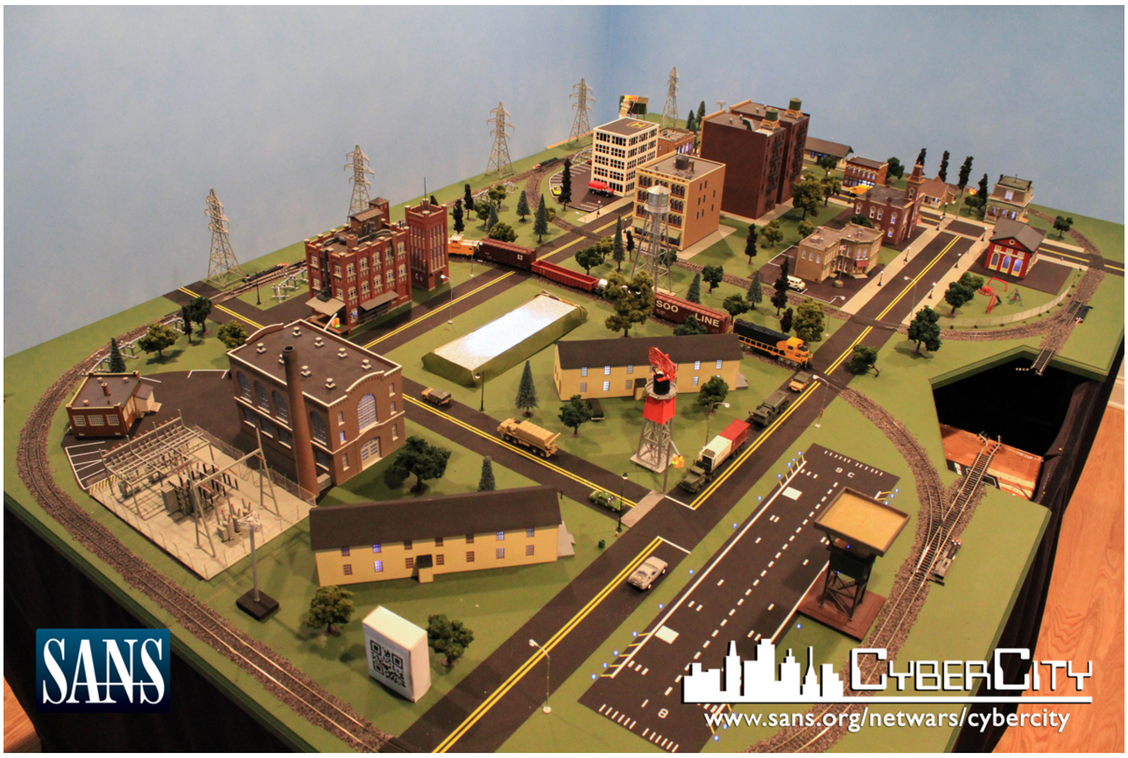


Figure 2.4: Cyber City Model by the SANS Institute

Models and the related courses aim to show the impact of cybersecurity on real-world components. Cyber defenders also use this type of cyber modeling techniques to evaluate and secure systems. However, the physical systems are challenging to construct and customize, and hence it is unpractical to build large scale physical replica of defended systems. Therefore, newer modeling approaches are required.

2.1.3.2 Graphical Models

A graphical model of a cyber system is commonly used by system defenders to represent the general architecture. For example, Figure 2.5 is a basic graphical representation of a defended network and some of its components. However, these models are limited to physical topology or generalized ontologies.

A report on cyber-physical systems highlighted the drawbacks of graphical techniques [24]. For example, UML and SysML provide a graphical foundation and basic interface definitions but do not model at the semantics (language) level. Also, their models do not include the fundamental uncertainties associated with the cyber domain. This level

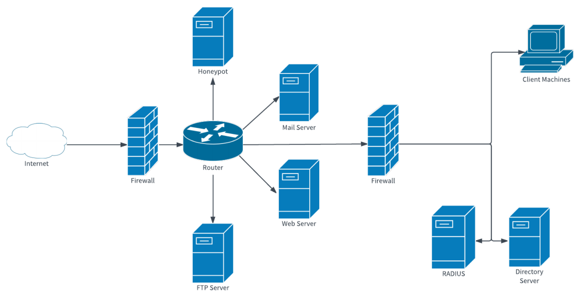


FIGURE 2.5: Example Network Security Diagram (Source: Lucid Chart)

of interaction creates security gaps where malicious activities may occur and therefore is a cause of concern. Graphical models are relatively easier to create and understand. They provide a quick and comprehensive view of systems useful for planning. One such application is the generation of attack-graphs. A security exercise includes a Red Team generating a graphical depiction of attack methods for the target system. The graphical depiction has to find routes via hopping from one physical system to another until a specific component is compromised. This graphical depiction is, therefore, considered much simpler than a comprehensive view. An example is shown in Figure 2.6.

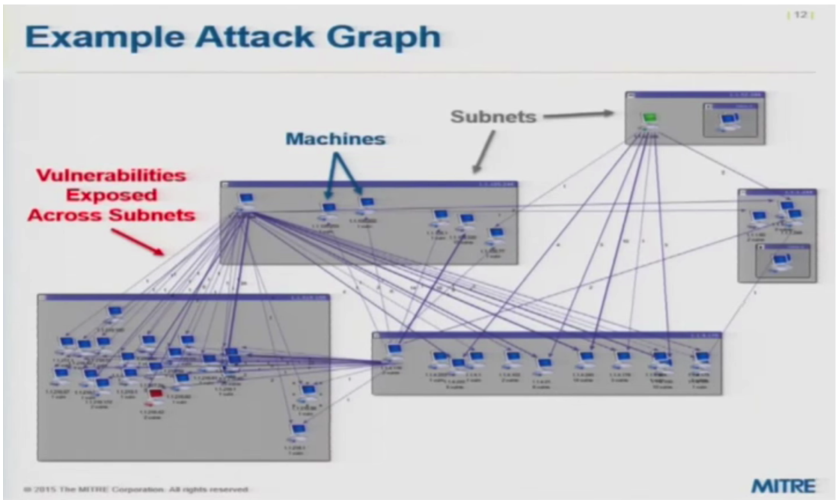

Example Attack Graph
12
Subnets
Machines
Vulnerabilities Exposed Across Subnets
© 2015 The MITRE Corporation. All rights reserved.
MITRE

FIGURE 2.6: Example Attack Graph

The use of attack-graphs is standard practice for defenders and Red Teams. However, they can result in a lack of total risk comprehension by the defenders. Even though attack-graphs do show vulnerable systems, the graphs are generated manually without much of the system context. Moreover, these techniques are deterministic. A defender must know (in advance) the traffic paths, the vulnerabilities of the system, and the value of the defended information to create a useful graphical model. Also, these methods must have a vulnerability metric, such as the Common Vulnerability Scoring System (CVSS), to judge between systems [25]. Therefore, the quantitative formulation of these models based on assumptions may result in limits of system understanding.

2.1.3.3 Mathematical Models

The final category of models uses mathematics to represent information, interconnections, and semantics in an abstract quantitative form. However, the techniques addressed in the previous sections typically focus on a specific location on a network and often do not take holistic approaches. Significant research on graphical models has been reported in the literature [26, 27, 27]. Furthermore, new studies using advanced algebra for agent-based models have shown promising results [28].

Automatically generating attack-graphs is the most popular method of modeling cybersecurity systems using mathematics. In this context, the term graph refers to a representation of a network as nodes and edges rather than a visual network topology. However, the method of generating an automated attack-graph takes a long time to scale as most of the information required by the system defenders is unavailable in real-time. Many real-world systems are just too large to maintain a real-time map of potential attack paths.

However, attack-graphs are useful in building small scale models. Sheyner et al., 2002, demonstrated the efficiency of attack-graphs at a small scale [29]. A threat has a discrete set of uncertain steps to accomplish before it can compromise a defended host. This idea can be modeled using Markovian methods. Abraham et al., presented a model based on Markov Decision Process where paths are represented as different states with uncertain transitions culminating in an absorbing state (a successful attack) [30–32].

However, attack-graphs are useful in building small scale models. Sheyner et al., 2002, demonstrated the efficiency of attack-graphs at a small scale[29]. A threat has a discrete set of uncertain steps to accomplish before it can compromise a defended host. This idea can be modeled using Markovian methods. Abraham et al., presented a model based on Markov Decision Process where paths are represented as different states with uncertain transitions culminating in an absorbing state (a successful attack) [30–32].

Attack-graphs have been quantitatively analyzed using probabilistic methods and graph theory. Li and Vaughn [33] present models using cluster analysis. The concept of Attack Countermeasure Trees (ACT) is given by Roy et al. [34], and the advanced metric analysis is provided by Wang and Idika [35, 36].

Analysis of attack-graphs requires a priori knowledge of each component in the networks vulnerability status by the defender (or the attacker), which is unrealistic. Alternatively, attack-graphs can be viewed as a comprehensive set of stages instead of a set of devices. In later sections, the concept of kill chains is explored as an extension to the idea of generalized attack-graphs.

Accurate models are needed to incorporate the current state of cybersecurity with complex interactions and actors. These models should also include actionable parameters for risks and decisions. Techniques based on graphical and mathematical models applied to risks and decisions are not well documented in the literature. Detecting malicious activities and finding security gaps are the two main topics of interest for the cyber systems modeling techniques.

In general, the of modeling is to allow efficient detection of malicious or threatening activities by the defended systems. The next section discusses the tools and techniques currently used by cybersecurity defenders.

2.2 Threat Detection

Historically, defenders detect threats by deploying sensors at different stages of an attack. Such activity is referred to as intrusion detection[4]. Each sensor can either be deployed on a host device or used to tap network traffic. The complex nature of a threat requires data from multiple sources to carry out an investigation. The generation and collection of data have a cost-benefit trade-off. The cost of handling large datasets is traded against the benefit of a safer system.

The organizational cost of threat detection is based on the size of the defensive human workforce, and the cost of the sensors deployed. The array of sensors within an organization is referred to as a security stack. The following section discusses the commonly used security stack components.

2.2.1 Security Stack

The security stack is the collection of installed clients on the host, physical, and virtual devices that an organization deploys to defend their information system. The host-based clients often referred to as anti-virus software, monitor activity within a single computer or server. Network systems are deployed at a specific point to maximize visibility and situational awareness of the defenders. Network devices are connected in series with traffic flowing in from one side and flowing out from the other side. Security devices

[4]Threats and intrusions are not the same thing. Threats are a broad category of network actors, and intrusions are events. Furthermore, labeling of some security devices as 'intrusion detection' may be misleading

are commonly placed at the boundary of a defended system that routes inbound and outbound traffic through a set of monitored devices. Figure 2.7 is a photograph of a Joint Regional Security Stack (JRSS) deployed within the United States Department of Defense.

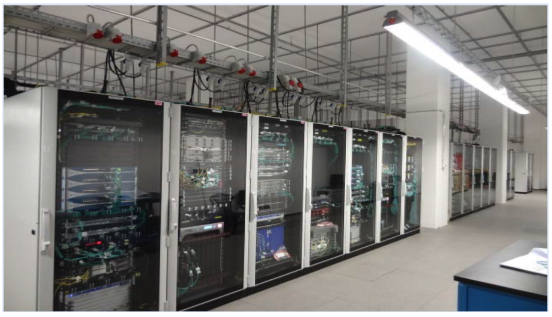


Figure 2.7: Joint Regional Security Stack deployed by the United States Department of Defense

The number of devices within a security stack can vary depending on the organization. For example, the JRSS figure depicts a network stack with 20 racks, and each of which can hold up to 48 different devices. The size of the deployed stack is governed by the engineering requirements based on performance and redundancy. However, there is a standard set of tools that most organizations deploy.

2.2.1.1 Intrusion Detection Systems

A tool that sniffs network traffic is commonly used in a security stack. A network sensor device that aids network sniffing is called an intrusion detection system (IDS). The IDS monitors the network traffic for all incoming and outgoing system activities directed through the stack and searches for any known signatures. This method has been a critical part of network security for the last two decades [37]. Typically, the IDS is deployed as part of a security stack along with the firewall or other devices, which are discussed later in this chapter.

The IDS operates, almost exclusively, on known signatures. The packets within network traffic are inspected via pre-programmed string-matching operations against known malicious indicators. Locally developed, open sourced, and vendor procured indicators are used to program an IDS. The most commonly used network IDS is called Snort, an open source system that offers free and paid signatures [38]. Some IDS devices directly operate on host computers and are referred to as antivirus agents or host-based security systems.

2.2.1.2 Firewalls

Similar to an IDS, a firewall can be used to monitor incoming and outgoing traffic on a network. Firewalls limit unwanted traffic based on rule sets for the intended devices. IDS activity informs firewall rule sets, which are most often deployed at the edges of a network. These devices have been in use since 1988 [39]. Nowadays, firewalls are often depicted as the first line of defense for any organization, as shown in Figure 2.8.

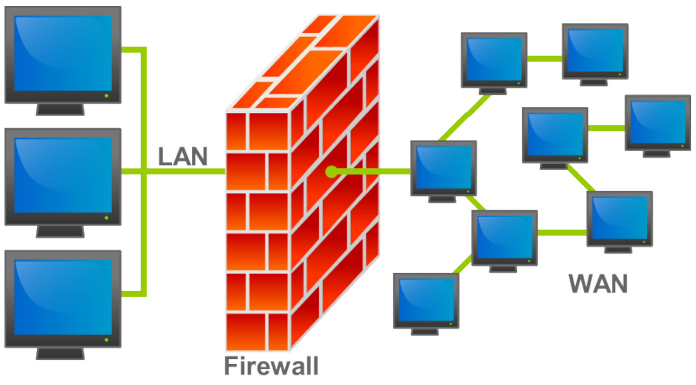


FIGURE 2.8: Common Firewall Depiction: Protecting the LAN from the WAN

The use of a firewall is not necessarily specific to network security. Many organizations use them to control the content available to system users. For example, the 'Chinese firewall' is a set of devices used to filter network traffic for people within China [40]. At a smaller scale, some organizations also filter content for their employees for performance-related reasons.

2.2.1.3 Network Proxies

A network proxy is another commonly used device in a security stack. The proxy masks the devices that communicate outside the network by providing a single outward facing point. Therefore, the external entities have visibility of the proxy and not the devices behind it. Historically, these devices are not only intended to provide security but also to manage the limited number of publicly available internet protocol (IP) addresses. Nowadays, it has become more challenging to collect host device information because the host information, such as operating system and software versions, can be used to map vulnerability databases to generate attack-graphs. A proxy makes this process difficult as it obscures the devices with which an external entity is trying to connect.

2.2.2 Data Aggregation and Correlation

Single security stack devices rely on known signatures with visibility of a single location, either network or host, which is a significant disadvantage. Previous studies have aimed at combining data from multiple sensors to build a clearer picture of threat activity [41]. However, the complexity of some attacks can easily overpower the abilities of IDS, firewall, and proxy collection. Such situations would require manual analysis of a broad range of data over a long period.

A Security Information and Event Manager (SIEM) is used to execute this approach. An alert from an IDS initiates an ’incident response’, also known as manual analysis. The human component in the loop has always been challenging to scale, and newer approaches have been developed to standardize the problem of incident response.

Incident response improves the security of a defended system by investigating and taking action. The primary mode of opertaion is to adjust the networks access posture. Cybersecurity defenders can allow or restrict any specific activity. The following section discusses the tools and techniques used in this context.

2.3 Access Control

Typically, threat detection is followed by a response. In cybersecurity, defensive responses often manifest as a change in access to the information system. Two extreme approaches used for access control are blacklisting and whitelisting. In signature-based sensor technology, when a given sequence of information (or rule) is detected, a sensor alert is triggered. Blacklisting is the practice of rejecting something if the sensor rule is met. Conversely, whitelisting is the practice of rejecting everything that does not match a rule. These approaches can also be taken with other features of cyber entities like domain names, IP addresses, files or any characteristic that has encoded information.

2.3.1 Blacklisting

Blacklisting requires maintaining a list of signatures to be identified as a threat based on some historical precedent. This list restricts access to the defended system. Anything outside this list is allowed. However, the size of these lists can be large. As an example, a blacklist used in chapter four contains twelve million IP addresses. Checking and monitoring network traffic or file usage against such a list can impact system performance. Moreover, blacklists can become stale because threats may quickly alter themselves if their signatures are blocked. Therefore, the rate at which a blacklist change is a critical factor for the defender.

An example of blacklisting includes an anti-virus software applied to system files which only checks the presence of a hash[5] (a unique file identifier) on the list of known malicious files [42]. Updating anti-virus software often makes this list of known signatures longer. However, cybersecurity applications, like domain name filtering, are not updated often enough to be effective against sophisticated attackers [43]. This is because attackers can quickly change their characteristics, like domain name, to take advantage of latency in updating the blacklists. The lists of bad actor signatures can soon grow to make the string matching approach ineffective. A longer list leads to a slower string matching because each operation may require scanning the entire blacklist again. This can also be true of general characteristics. For example, firewalls can restrict networks from communicating directly with external geographical locations (or entities) which the defender considers of high risk. Such activity restrictions are commonly referred to as blocking.

The blacklisting approach has evolved in recent years. Updating the blacklist often results in significant improvements in risk exposure. Therefore, the efficacy of a blacklist is sensitive to the threat dynamics [44]. For this reason, blacklists are excellent candidates for predictive techniques. Prakash [45] gives an example of these techniques applied to phishing, and an application to spam is provided by Ramachandran [46]. The latter presents a concept of behavior detection that will be explored in the following sections.

2.3.2 Whitelisting

On the other extreme, the whitelisting technique determines access based on a set of known non-threat signatures. All signatures are restricted except those on the whitelist. In the most sensitive systems, whitelisting is a security standard but challenging to execute in practice. For example, Supervisory Control and Data Acquisition (SCADA) devices deployed to manage critical infrastructure are often configured to communicate via whitelists. Figure 2.9 shows a local control system networked via a set of SCADA connections in a power generation facility.

The SCADA devices directly control the power generation equipment and represent a significant risk if compromised. However, these systems often require connections to external entities for maintenance, troubleshooting, software updates, and consolidated monitoring.

[5]A hash is a signature of a file generated from the files creation process allowing it to be tracked. As the file changes, the hash changes with it.

Figure 2.9: Example SCADA Control Room

The use of whitelisting reduces the value of a system and is not considered suitable for entities that interact with the general population. For example, an e-commerce website would not know the signatures of all future customers and hence cannot deploy a useful whitelisting security method.

In practice, organizations use a combination of blacklisting and whitelisting techniques. For example, administrative access ports and tools are protected using a whitelist of known administrators, and on the other hand, public ports for HTTP traffic are guarded using a blacklist.

An organizational approach to access control depends on the sensitivity of the protected information and the value of the services provided to the users. Systems that place greater importance on protecting information will have more restrictive access than those seeking ubiquitous availability. A simple example could be the difference between the access control of a nuclear power station versus the Google website. The former is whitelist protected, and the latter is blacklist protected.

In this research, the access control policies are generated by combining the whitelisting and blacklisting approaches into an intelligent paradigm. To make this leap, a framework for threat behavior is explored. The following section discusses the concept of kill chains and their relation to threat identification and access control.

2.4 Cyber Kill Chain

A popular approach to standardize incident response and to study events is by using conventional methods of exploitation as a framework. One such technique called the 'cyber kill chain' has been developed by Lockheed Martin Corporation [38]. It is based on a modeling approach that categorizes malicious activities, from an attacker's perspective, into seven stages as shown in Table 2.1 with example signals:[6]

Table 2.1: Cyber Kill Chain Stages (j)

j	Step	Example Signal Type	Lead Time
1	Recon	Boundary access behavior	Weeks
2	Weaponize	Rate of vulnerabilities of systems	Days
3	Deliver	Virus detection on IDS systems	Minutes
4	Exploit	Anomaly behavior	None
5	Control	Improper access control use	None
6	Execute	Anomaly of data transmission	None
7	Maintain	Patterns in external communication events	None

The kill chain aims to dismantle an attack by disrupting it at an early stage. There are only a few steps that allow the mitigation of all damages, as shown in Table 2.1. Some amount of damage would have been incurred within the final four stages, which will require remediation. Figure 2.10 shows an example of a targeted malware attack with timing expectations. A defender can avoid damages if an attack is broken within the first three steps of the kill chain. A defender will attempt to limit the scope of impact after the exploit is complete.

The MITRE Corporation, a United States Federally Funded Research and Development Corporation (FFRDC), has improved the Lockheed's initial modeling approach by expanding steps 5-to-7 into nine additional categories. Figure 2.11 shows the MITRE Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) matrix.

The matrix in Figure 2.11 reveals that sophisticated actors can easily evade single or multiple homogeneous data stream monitoring approaches. An enormous amount of heterogeneous data must be evaluated across hosts, servers, and security devices to detect moderately advances threats. Additionally, methods of detection must include external intelligence. This external information has precedence with the legacy IDS signature adoption but adds more robust profiles of actor behavior.

In cyber systems, the motivation for behavior modeling is rooted in the complexity of attack graphs and their target networks. Sharing signatures of IDS devices may be

[6]Lead Time is the period between a signal is observed and a damage event is realized. These stages are based on Lockheed Martins kill chain.

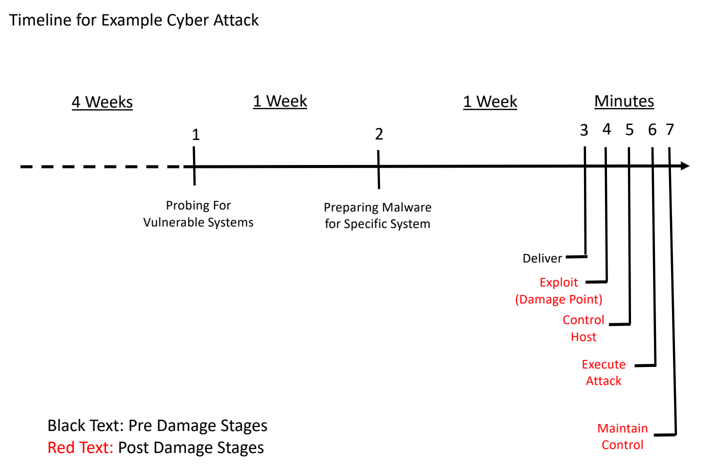
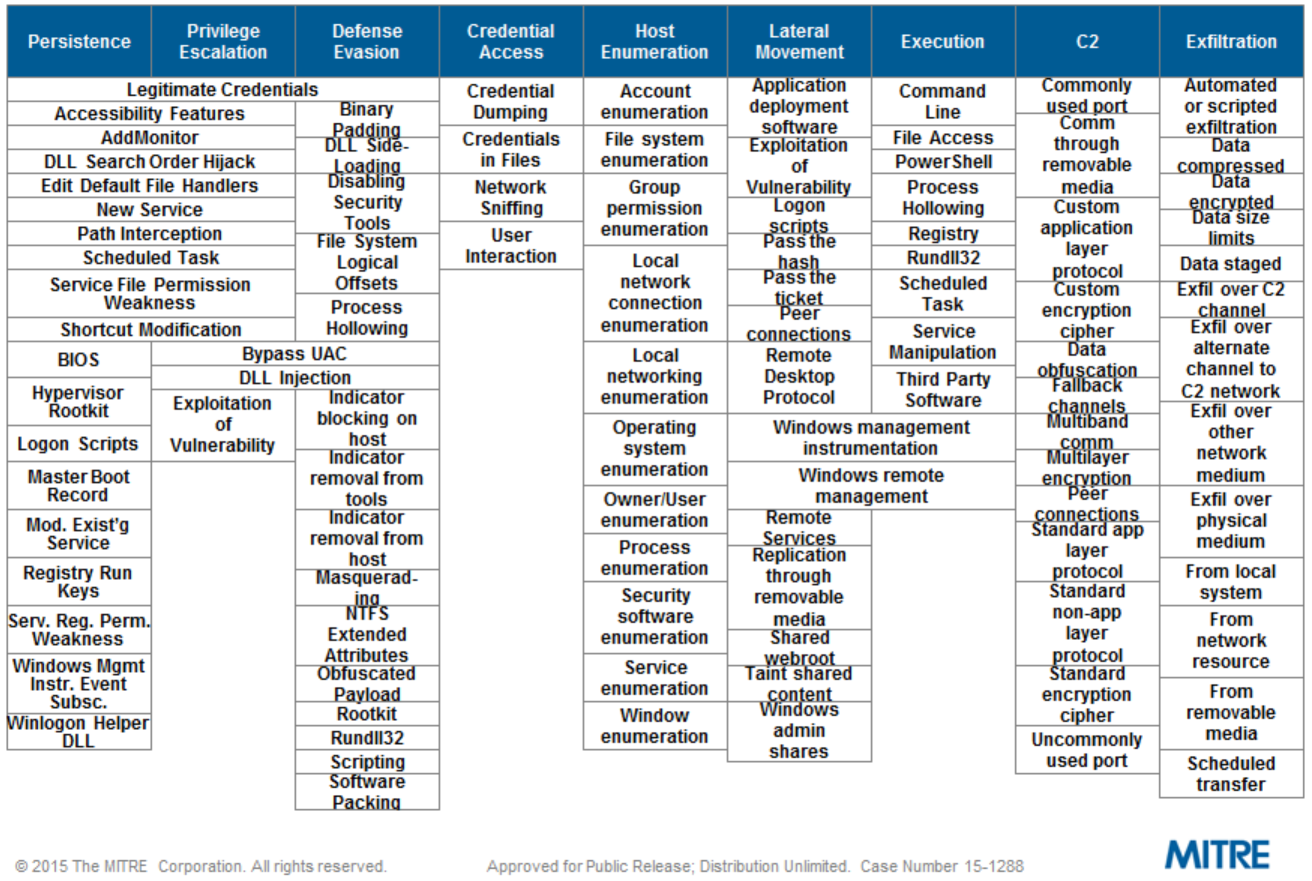

Timeline for Example Cyber Attack
4 Weeks
1 Week
1 Week
Minutes
1
2
3
4
5
6
7
Probing For Vulnerable Systems
Preparing Malware for Specific System
Deliver
Exploit (Damage Point)
Control Host
Execute Attack
Maintain Control
Black Text: Pre Damage Stages
Red Text: Post Damage Stages

FIGURE 2.10: Example Kill Chain for Malware Attack.

simple, but sharing behavioral information is not straightforward. Current approaches use a set of characteristics for identifying known threats. Features like IP addresses, domain names, email addresses or file hashes are standard methods. This information sharing approach results in an abrupt growth in threat repositories since these features are easily changeable by actors. Few freely available sources of information routinely update over ten million threat indicators within their databases [47].

The current techniques used for threat identification are still mostly based on the historical signature approach. Defenders find it challenging to capture a threat even when signals have been triggered at the early stages of the kill chain. Behavior patterns are not adequately exploited, in part due to a large amount of information needed to obtain them across the kill chain. Much of the analysis is done manually and gathered in a case file. Some work has been done to standardize these case files, such as the Structured Threat Information eXpression (STIX) and the Trusted Automated Exchange of Indicator Information (TAXII). However, few attempts have been made to automate this process.

The advent of new big data capabilities helps make statistical and machine learning techniques more readily available for threat detection identification. The use of such tools could aid in the automation of threat detection and response. However, these statistical methods rely on historical data. Limited research has been done on homogeneous data types [48] [49]. However, a high false positive rate of these quantitative methods in advanced threat detection may have limited their use in real-world applications. Robust



Persistence	Privilege Escalation	Defense Evasion	Credential Access	Host Enumeration	Lateral Movement	Execution	C2	Exfiltration
Legitimate Credentials	Legitimate Credentials	Legitimate Credentials	Credential Dumping	Account enumeration	Application deployment software	Command Line	Commonly used port	Automated or scripted exfiltration
Accessibility Features	Accessibility Features	Binary Padding	Credentials in Files	File system enumeration	Exploitation of Vulnerability	File Access	Comm through removable media	Data compressed
AddMonitor	AddMonitor	DLL Side-Loading	Network Sniffing	Group permission enumeration	Logon scripts	PowerShell	Custom application layer protocol	Data encrypted
DLL Search Order Hijack	DLL Search Order Hijack	Disabling Security Tools	User Interaction	Local network connection enumeration	Pass the hash	Process Hollowing	Custom encryption cipher	Data size limits
Edit Default File Handlers	Edit Default File Handlers	File System Logical Offsets		Local networking enumeration	Pass the ticket	Registry	Data obfuscation	Data staged
New Service	New Service	Process Hollowing		Operating system enumeration	Peer connections	Rundll32	Fallback channels	Exfil over C2 channel
Path Interception	Path Interception	Bypass UAC		Owner/User enumeration	Remote Desktop Protocol	Scheduled Task	Multiband comm	Exfil over alternate channel to C2 network
Scheduled Task	Scheduled Task	DLL Injection		Process enumeration	Windows management instrumentation	Service Manipulation	Multilayer encryption	Exfil over other network medium
Service File Permission Weakness	Service File Permission Weakness	Indicator blocking on host		Security software enumeration	Windows remote management	Third Party Software	Peer connections	Exfil over physical medium
Shortcut Modification	Shortcut Modification	Indicator removal from tools		Service enumeration	Remote Services	Windows management instrumentation	Standard app layer protocol	From local system
BIOS	Bypass UAC	Indicator removal from host		Window enumeration	Replication through removable media	Windows remote management	Standard non-app layer protocol	From network resource
Hypervisor Rootkit	DLL Injection	Masquerading			Shared webroot		Standard encryption cipher	From removable media
Logon Scripts	Exploitation of Vulnerability	NTFS Extended Attributes			Taint shared content		Uncommonly used port	Scheduled transfer
Master Boot Record		Obfuscated Payload			Windows admin shares			
Mod. Exist'g Service		Rootkit						
Registry Run Keys		Rundll32						
Serv. Reg. Perm. Weakness		Scripting						
Windows Mgmt Instr. Event Subsc.		Software Packing						
Winlogon Helper DLL								

© 2015 The MITRE Corporation. All rights reserved. Approved for Public Release; Distribution Unlimited. Case Number 15-1288

MITRE

Figure 2.11: MITRE Kill Chain Extension w/ data sources

model-based detection methods are required to reduce the false positive rates and to improve the automated threat detection across heterogeneous data streams.

This research aims to model the risks and decisions in complex cyber systems using the technology of big data and parallel computing to build behavior identification techniques. Previous research in this area mainly focused on specific domains. The following section discusses warning systems for risk analysis and its application in threat detection.

2.5 Early Warning Systems

The sequential nature of the kill chain motivates the application of early warning systems in cybersecurity. Each step of an attack produces a unique signal. Cyber threat actors use a multi-stage attack approach. On the other hand, defenders collect behavioral information using security stack technologies. Therefore, it is possible to find early-stage behavior patterns within the routinely collected data.

Early behaviors can serve as signals of more serious events that may occur further along the kill chain. Part of this research focuses on using reconnaissance data to signal risk levels for defenders. In a warning system, these signals are generated, processed, and routed to a decision maker before severe consequences have incurred. Applications of warning systems in other areas include health [50], finance [51], and natural disasters [52]. The research on warning systems primarily focuses on saving human lives. However, it can also be applied to the analysis of the cost versus benefit problem. Studies such as [53] and [54] demonstrate the connection between signals and probabilistic risk analysis, albeit with some limitations.

In warning systems, signals are generated using statistical predictions based on historical data. By definition, predictions have a likelihood of being incorrect. Also, predictive methods might ignore key systems features. This creates the conditions for so-called 'black swans,' where decision makers may be caught off guard by an unexpected outcome [55]. In a broader context, signal monitoring based on statistical analysis is susceptible to overlook the impact of rare events or perfect storms [56]. It is therefore essential to consider the entire system and its inherent flaws when designing a warning system. The errors associated with false positives and true negatives determine the quality of any system. The warning systems approach used in this thesis will build on the work done by Pate et al., 1986 [57]. Figure 2.12 demonstrates the distinction between true and false alerts derived from their work.

Early warning systems are not optimal, and hence a question arises: How sensitive should the signal producing device be tuned? This question can also be explored from

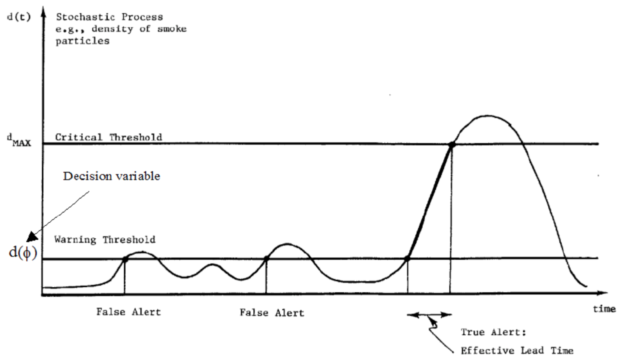


FIGURE 2.12: Warning System Tuning Example ("Warning Systems in Risk Management" M. E. Paté-Cornell 1986).

the cost/benefit perspective. Issuing too many false alerts impacts the cost, as well as, the human interpretation of the warning signals. Other considerations include, how much lead time does the warning signal provides to the defender, and what actions can be taken when a signal is detected. The technique developed in this research aims to find an optimal warning threshold to inform the access control policy, based on risk tolerances.

The following set of considerations are explained in Chapter 3.

- A model of cyber-attack behavior during the early stages
- A decision model for defenders given the dynamics of the sensors sensitivity
- A memory model for defenders given the memory of past alerts

There are two critical considerations to find optimal thresholds for a warning system. The first consideration is the link between response time and error rates. The second is the memory in the response level, often called the 'cry wolf' effect. Quantified costs and benefits also serve as critical inputs to this type of model.

For a given system, tuning the warning thresholds for a known process results in an optimal setting as shown by [58] and [57]. An overview of Paté-Cornell's approach is described in Appendix A.

The knowledge that signals in warning systems arrive at a given rate can be used to optimize costs based on the impact of events. These events are discrete observations in the warning system that may cluster to form behavior patterns. They are critical for calculating costs and benefits as a consequence of changes in thresholds. The set of variables to consider in a warning system is described in table 3.5.

TABLE 2.2: Example Cyber Warning System Variables

Events	Description
E	Cyber Threat
S	Signal of Threat
A_t	Alert - True
C_f	Alert - False (type 1 error). Can be caused in two ways; either by the process meeting the threshold or by an irrelevant occurrence
N_t	No Alert - False (type 2 error)
N	Number of compromised entities without the warning system
W	Warnings $(A_t + A_f)$
irrelevant occurrence	An event that triggers the alarm but is not part of the inherent process (i.e. sensor alerts because of unscheduled penetration testing)

The dynamics of an example discrete model are represented in Figure 2.13.

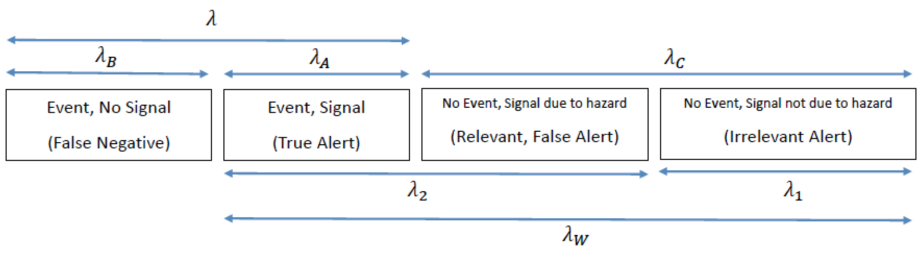


FIGURE 2.13: Warning System Rates in Context (Derived from "Warning Systems in Risk Management" M. E. Pat-Cornell 1986)

The characteristics of warning systems allow simple evaluation of the systems quality and cost/benefit analysis. The variables allow for the assessment of uncertain signals.

Furthermore, signal creation techniques can be guided using this framework, which addresses an important question: how to implement the signal generating algorithm in cybersecurity?

The characteristics of warning systems allow simple evaluation of the systems quality and cost/benefit analysis. The variables allow for the assessment of uncertain signals. Furthermore, signal creation techniques can be guided using this framework, which addresses an important question: how to implement the signal generating algorithm in cybersecurity?

2.6 Machine Learning for Cybersecurity

Security-focused machine learning is the concept of teaching a system to identify and signal a threat or attack without the direct intervention of a human analyst. As a term, it is extensive in application. Mitchell defines machine learning as

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at a task in T, as measured by P, improves with experience E [59].

This definition implies that a machine learns from and improves with experience. In this view, actor behavior is not just a system baselining characteristic but an interplay of many factors that can be used to learn to classify. With machine learning different behaviors (from differing data sources) can serve as the experience for learning the task of threat identification. Some of these approaches can be applied directly in the intrusion detection system [60]. However, the broader interpretation has a wide range of applications across the cybersecurity domain [61]. These are not limited to but include identification of botnets [62], malware [63] and phishing [64]. Each of which has a complex behavior profile that can require heterogeneous data sources for proper detection.

The techniques discussed here fall into the category of supervised machine learning. Supervised learning is used when historical data has both observed inputs and outputs. The model inputs are called features and the output variable of the host is the response. The goal of a learning algorithm is to train a model that best fits the inputs to the outputs. For example, linear regression can be used as a basic supervised learning model.

Reinforcement learning is another important category of modeling. These techniques allow learning by machines through action and environment response instead of just historical data. In this context the machine experiments (explores the environment) as the learn mechanism. One classic application is active learning with multi-armed bandit models [65].

The use of machine learning for cybersecurity has a significant history. The highest profile example is in spam filtering. Spam is the colloquial term for unsolicited bulk email which became a substantial problem in the early days of the internet [66]. This application uses the concept of classification; an email is spam or not spam. This core idea can be easily extended to general security; an entity is a threat or not a threat. A set of candidate supervised classification algorithms are discussed in detail in Chapter 3.

When using machine learning there are many considerations. The following example illustrates them and using data derived from Jacobs [67] for exploring how supervised machine learning would be applied to a general use case of identifying a compromised host (malware). Data presented assumes access to normalized system-level information about a host computers processor and memory resource usage. Also, historical information would have been collected about whether these hosts is infected. Figure 2.14 shows this information as a dot plot.

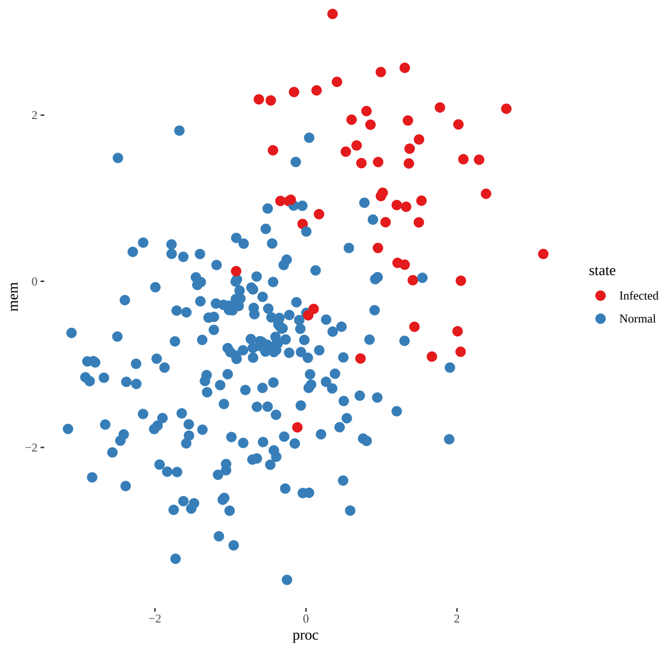


Figure 2.14: Observed Security Data Example (Jacobs and Rudis 2014)

A quick survey of Figure 2.14 shows that there is a natural separation of infected and healthy hosts. However, in order to train a machine to identify the same a learning algorithm must be used. The generic process that machine learning applications take are:

1. Break data into training and testing portions
2. Select learning model
3. Train algorithm (an instance of a model type)
4. Test the performance of the algorithm
5. Repeat until satisfied with accuracy

The process of testing is referred to as validation and is distinct from other modeling approaches in different disciplines. For example, econometrics seek to understand the characteristics of an entire data set [68].

The approach to validation can be done as one segmentation, a sliding, or a growing window [69]. For this example, a single segmentation is used. The code can be found in Appendix B. The results from this test approach are visualized in Figure 2.15.

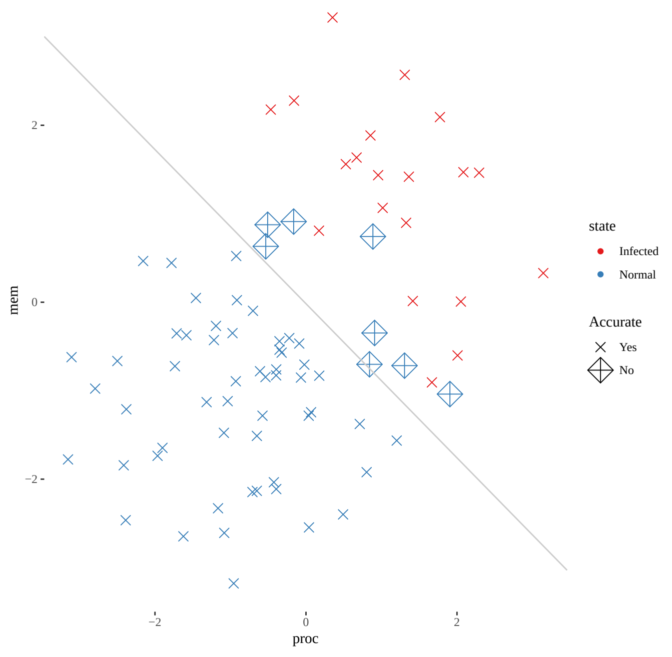


FIGURE 2.15: Machine Learning Performance on Test Data

Figure 2.15 shows a basic learning method of using a hyperplane to best separate the classes. However, there is some error. Both type I and type II errors are present as a consequence of the type of model. This simple example illustrates how machine learning techniques have the same characteristics as early warning systems. The fundamental error associated with classification is the same as that of signals. This makes machine learning a useful tool in the effort to deploy an early warning system for cybersecurity.

2.6.1 Feature Engineering

The application of machine learning to cybersecurity has grown significantly in the past few years. The advent of modern deep learning (artificial neural network) techniques (discussed in Chapter 3) has resulted in a growing body of literature. The primary concern when building machine learning methodologies is data collection and feature engineering.

As mentioned in the previous section, features are raw inputs to the model and response variables are used for training. For example, in computer vision tasks, features are the colors of individual pixels in an image and the response is a label of what the picture contains (like a cat). In cybersecurity, the features are derived from the data produced by the security stack and the response label is provided by forensic expert analysis.

The range of applications for machine learning is quite broad. However, this research is concerned explicitly with network threat or attack identification. The availability of public datasets for this application is limited. The most popular canonical datasets used for publication are the NSL-KDD and the DARPA Intrusion Detection dataset, both of which have considerable shortcomings [70] [71]. The NSL-KDD dataset is just a version of the DARPA dataset. One of the problems with these datasets is that they are developed from contrived environments and may not reflect information available to a network defender. For example, the IDS device produces individual alerts with a single timestamp.

The NSL-KDD data set is presented as a consolidation of a set of network activity. These consolidations are often referred to as flows or connections as they are derived from a collection of packet level data. The dataset is labeled with, so-called, anomalies which represent different types of attacks. While the set of 41 features in the data is robust it is not easily replicated in real-world settings. Also, using this large of feature set impacts the performance of the algorithm and the online performance in practice. There have been several papers that specifically deal with the feature dimensionality reduction using various methods. For example, Javaid and Yousefi-Azar use an autoencoder [72] [73]. Chae uses a subset of features [74]. Another approach is given by Eid that implements Principal Component Analysis [75]. Each of these methods uses a two-step approach. First, reduce the feature set then perform model training.

Table 2.3: Basic Features of Individual TCP Connections

feature name	description	type
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of wrong fragments	continuous
urgent	number of urgent packets	continuous

Table 2.4: Content Features Within a Connection Suggested by Domain Knowledge

hline feature name	description	type
hot	number of hot indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of compromised conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if su root command attempted; 0 otherwise	discrete
num_root	number of root accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_hot_login	1 if the login belongs to the hot list; 0 otherwise	discrete
is_guest_login	1 if the login is a guestlogin; 0 otherwise	discrete

The problem with feature dimensionality reduction is that machine learning models become less explainable. If the feature set used to train a model is a latent representation of actual features, then using a detection algorithm for updating security policy becomes difficult. Even if an algorithm performs well when detecting attacks or threats it may not be useful for informing specific response actions. One important contribution of this research is to explore the deployment of models which directly inform access control policy decisions. In order to do this efficiently, features must be persevered for use in decision applications.

The selection of a feature set impacts the performance of the model, availability of data, and practical application. The following tables are the set of features which the literature uses and are derived from the NSL-KDD dataset. The origin of this dataset was a competition and these descriptions were given to the contestants.

Each of the feature tables represents feature categories for network threat detection found in the literature. Table 2.3 is populated with example packet level features, this is the

TABLE 2.5: Traffic Features Computed Using a Two-Second Time Window

feature name	description	type
count	number of connections in the past two seconds	continuous
serror_rate	% of connections that have SYN errors	continuous
rerror_rate	% of connections that have REJ errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections in the past two seconds	continuous
srv_serror_rate	% of connections that have SYN errors	continuous
srv_rerror_rate	% of connections that have REJ errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous

lowest threshold of network cyber-data that can be collected. As packets are aggregated, more sophisticated analysis can be performed. For example, the flows mentioned earlier are session level information between two hosts.

Tables 2.5 and 2.4 are examples of features that can be inferred from individual sessions. However, the feature set does not contain identifying characteristics like actual IP addresses and ports. In practice, access control is done by blocking the characteristic features. As session data are aggregated, even more, robust analysis can be conducted. Patterns can be observed using behavioral based host characteristics like temporal, diurnal, geospatial. The following section discusses the use of behavioral based threat detection.

2.6.2 Behavior-Based Threat Detection

The application of behavior-based detection techniques is a paradigm shift from the traditional implementation of IDS systems. Behavior is the manifestation of a specific actors characteristics over time contained within security-related data. This includes a range of data view points from packets to multi-session activity. In this view, actors can be identified, not by conducting attacks, but by behaving in a particular fashion. Specific behaviors are detailed in Chapter 3.

For example, one temporal behavior for a given IP address can be represented as the time between connection attempts [76]. This can then be recorded and evaluated independently of the actual IP address. Independent evaluation of this type is of a mathematical nature and often requires non-traditional kinds of technology to conduct [61]. Because behavior can be represented as an algorithm, its use can, in some cases, be much less computationally expensive than signature-based methods.

Behavior-based detection is one solution to the limitations of black and whitelisting. The nature of detecting behavior is inherently uncertain and results in error rates for both type I and II. However, the definition of behavior is not universally accepted. One common approach similar to behavior-detection techniques is anomaly detection. This approach model's normal system behavior and then uses an algorithm to identify outliers [77]. Like the IDS, anomaly detection is typically restricted to a single data stream, such as traffic rates or resource usage [78]. Anomalies are identified through a process similar to statistical hypothesis testing [79]. However, for this research, there is a broader interpretation of behavior-based techniques that enables the use of robust machine learning methods [63].

The types of machine learning applications in behavioral detection are broad and include nearly every popular technique. Both supervised and unsupervised methods have been explored [67]. A small sample of recent methods includes regression, deep learning, clustering and Bayesian networks [80] [81]. Many of these techniques are also patented like [82] and [83]. However, these methods are primarily a classification of attacks as they present themselves. The literature is less robust when viewing machine learning techniques in the context of identifying specific actors. This research is a unique contribution to the approach of using machine learning techniques to generate warnings to identify a threat actor during the early stages of the kill chain.

2.7 Background Summary

The current state of practice in cybersecurity is to collect sensor data from the security stack and external information (like blacklists) and then build a security policy through a set of heuristics or manual decisions. Figure 2.16 is a diagram of the way this flow currently works for the cyber defender.

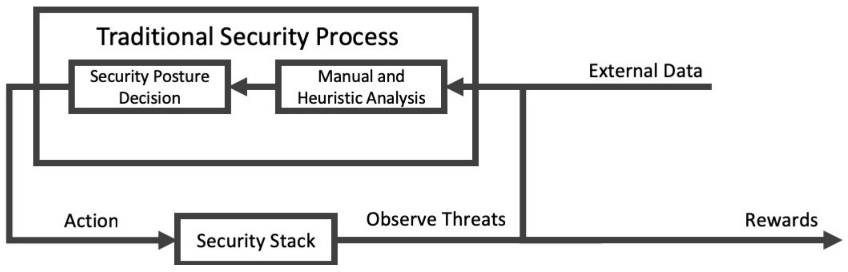

Traditional Security Process
Security Posture Decision
Manual and Heuristic Analysis
External Data
Action
Security Stack
Observe Threats
Rewards

FIGURE 2.16: Current Cybersecurity Workflow

This process is an active loop where defenders continuously react to new information and update the organization's security policies. As discussed in previous sections there are numerous efforts to improve parts of this process. However, there is not a common approach to integrating machine learning techniques, expert analysis, and decisions making. The next chapter formulates this system and proposes a method for intelligent access control. This approach leverages the knowledge of the expert cyber analyst, the speed of artificial intelligence, and the practical implementation of decisions about the security posture. This model is primarily useful for the management of cyber risk.

Chapter 3

General Model

The modeling and decision support techniques proposed here are built from the perspective of the system defender. This model framework is a tool for the management of cyber risk through the balance of system restrictiveness to usefulness. Methodologies are proposed for risk assessment, risk reporting, and risk control using feedback from machine learning processes.

Further, the modeling scope is defined as a single organization with a specific risk preference conducting active defense. The quantitative purpose of the model is to assist in the identification and deployment an optimal defensive policy. A policy consists of a set of decisions which maximize the defender's risk adjusted expected utility. In practice, optimal decisions strive to allow legitimate actors to use the resources of the system and block threat actors from the same.

Consider the example of defending an e-commerce website. Defenders recognize that each website visitor (actor) represents either a potential customer or a threat. The goal would be to allow as many customers to visit the site while blocking the threats. The risk assessment would involve understanding the impact of threats. In a perfect information setting the defender could block all threats and allow all legitimate customers. However, in practice, there is a degree of uncertainty of who is legitimate and who are threats. The defender must develop a defensive posture that balances this uncertainty allowing as many customers as possible while blocking likely threats. A poor defender could be one of two things: allow too many threats or do not allow enough customers. Also, both customers and threats can change over time giving the defender an inherently dynamic role. Risk control and reporting are the result of evaluating and implementing a defensive policy based on the historical data and available expert knowledge.

To address the uncertain and dynamic circumstances coupled with a high volume of data this model takes the unique view of the defender as a super-agent. A super-agent consists of two decision makers; an artificial intelligence (AI) based robot and an expert-human counterpart. The robot is pre-programmed with a reward (utility) function, decision thresholds, prior probabilities for various threats, and a risk preference. As the robot observes events, it updates (improves) the threat probabilities. Importantly, the robot is capable of reading large volumes of data from a central database (security repository) which is designed to enable the calculation of likelihoods. This database aggregates all of the defenders real-time and historical information from all available network security sensors and external data sources. The robot, using its learned understanding of the data, is given autonomy to make decisions below pre-established thresholds (based on utility and uncertainty) and elevates signals to the expert by exception.

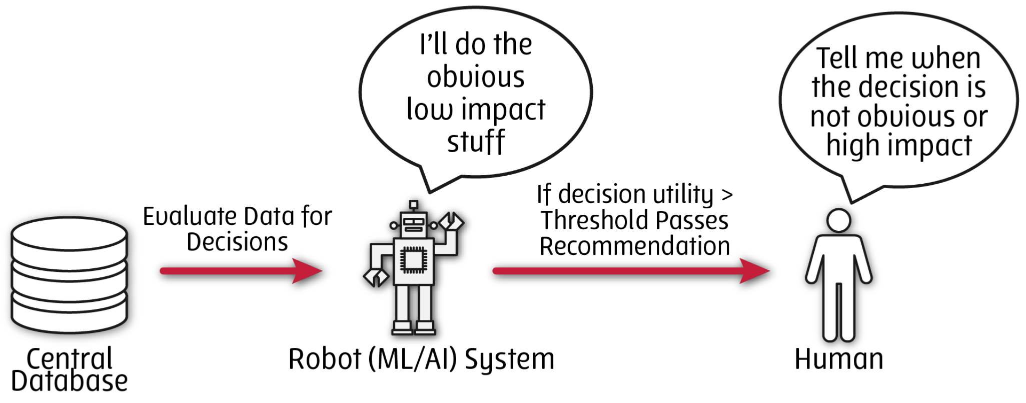

I'll do the obvious low impact stuff
Tell me when the decision is not obvious or high impact
Evaluate Data for Decisions
If decision utility > Threshold Passes Recommendation
Central Database
Robot (ML/AI) System
Human

Figure 3.1: Robot and Human Constitute a Super-Agent Decision Maker

The area of robot (AI) and human cooperation for a specific task accomplishment is an essential field of research on its own. In general, the human must know what thresholds govern the robot's decisions and what specific functions it is designated to perform. Also, the robot needs to know when to pass decisions to the human. The combination of these two agents creates a more capable 'super-agent.' The super-agent is preferred in complex but data-rich environments where human judgment can be essential, but the pace of decisions exceeds manual analysis capabilities. This concept is explored in more detail later in this chapter.

A super-agent system defender has a challenging role in that he/she has a static position which, for performance reasons, is difficult to alter over short time horizons. Again, consider the e-commerce website defender. They cannot easily deploy extra security devices or capabilities (like a new firewall) in the short term without risking an impact on the customer's experience while using the website.

The static position consists of defending an information system using installed sensors and devices that are tuned to detect or block intrusions (Intrusion-Detection-System and firewall). The general model assumes that defensive static security space (decision space) is fixed. However, the dynamics of strategic decisions are briefly discussed in later sections.

Sensors can be adjusted, by the human, in their sensitivity to collect information with a tradeoff between information over-load and situational awareness. A robot can assist to process this information and make below-threshold decisions as it learns to identify threats according to the priors and likelihood functions of the system (i.e., Bayesian updating). When predefined thresholds are met, the AI will pass the decision (with context) to the human for more in-depth investigation.

The tactical decision space for the super-agent consists of a series of blocking actions. Consider the kill chain discussed in Chapter 2. At each stage in an attack, the defender has a set of gates that they can open or close, preventing or allowing the activity. In this modeling context, a gate allows specific behaviors, so that closing a gate means preventing a particular behavior. This behavior is then blocked for both threats and legitimate users. For example, closing a gate might be: block network traffic on port 80 from the country of Nigeria. This would prevent Nigerian criminals from accessing the website but would do the same for Nigerian customers.

The uncertainty is that if select gates are closed, it may block legitimate network activity, resulting in a cost to the defender's organization (similar to the e-commerce defender example earlier). The defender's goal is to find the optimal gate policy (the set of gates to be opened and closed) that have the highest expected benefit to cost (utility). Figure 3.2 is a simple view of the defenders decision space over a time interval governed by the kill chain which illustrates the trade-off between blocking threats vice legitimate activity. Threats and legitimate activity can behave similarly and sometimes attacks and legitimate access traverse a similar set of gates.

Figure 3.2 demonstrates that the combination of gate states has polynomial growth (2^n where n is the number of gates which can be either open or closed). While more gates give the defender larger decision base, it also makes finding an optimal policy more computationally challenging. Each gate has some probabilistic characteristic for threat activity within a single step in the kill chain. However, threats and legitimate actors traverse a set of gates which leads to the consideration of joint probability distributions that grow with the gate states and kill chain stages. Another important factor is that the earlier in the kill chain a threat is stopped, the lower the chance of downstream damage. Optimal policies should take action at early stages when signals with sufficient confidence occur. Imperfect signals (or alerts) for attacks at each gate are generated by

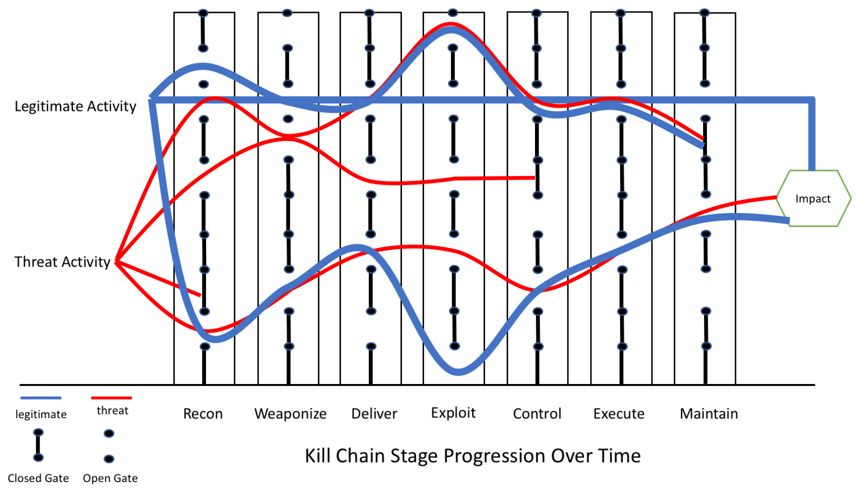


FIGURE 3.2: Cybersecurity Defender Decisions as Gates

learning from historical data, real-time context, and threat intelligence. The higher the amount of quality information the robot can learn from the more advantageous it will be. However, collecting massive amounts of data comes at a cost to the organization.

Machines which produce security-related alerts can, and do in some cases, collect every piece of information that traverses the communication channel. This extreme and expensive method is referred to as full packet capture (PCAP). On the other end of the spectrum is the deployment of simple heuristics that only alert the super-agent when known threats are present. Picking a middle ground that balances between cost and capability guides the decision on how much security information to collect. As the price of storage has dropped and technologies to store and retrieve large amounts of data have become a more commonplace, many organizations collect more data than they process [84].

Technologies designed to store large amounts of data are colloquially referred to as big data systems [85]. Such systems operate on parallel computation schemes, where information is spread across multiple independent smaller systems that behave like a single large machine. Within these (big data) environments complex logic can be executed across significant sizes of data. The advent and adoption of big data systems allow for the exploration of advanced decision support techniques in information-rich environments like those found in cybersecurity. Such systems are ideal environments for the central database depicted in Figure 3.1. High-quality signals can be produced from a larger pool of data than historically possible with manual analysis and heuristics. While big data systems may not be required for smaller organizations; they are necessary for most moderate cybersecurity tasks that seek to deploy AI technologies.

Modern AI systems generally perform well in data-rich environments, such as cybersecurity. AI models leverage the large amount of historical information created by the security devices (like Intrusion-Detection-Systems), learning how to identify threats, possibly even those that may be difficult for a human to observe and identify. The following sub-sections assume that super-agents have access to large-scale storage systems which make such applications possible. The following sections will present a framework for decisions in cybersecurity beginning with a detailed description of the defender's typical system characteristics, decisions, and posture.

3.1 Dependencies and Relevance

The general model is a utility optimization based on a set of decisions and related uncertainties. The model is only concerned with tactical intra-time period decisions. The premise is that the decision maker (super-agent) must learn the optimal policy (set of decisions) over time through a trial and error scheme (explore vs. exploit). The challenge comes from the dynamic and uncertain qualities of the system. However, there are circumstantial conditions that can assist in managing the system, which is governed, in part, by strategic choices. The strategic decisions of interest include:

- Central database design
- Defensive technology stack
- Risk tolerance thresholds

The strategic setting allows for the execution of tactical decision policies which will be made by the super-agent at each time period. The following sections explore these tactical decisions and related uncertainties. The model puts the decision space into the context of the super-agent with responsibility delineation between the robot and the human [86]. Defensive decisions are made by the robot through explicitly programmed heuristics, risk attitude, and AI at below-threshold utility and confidence levels. If pre-programmed alert thresholds are exceeded, the decisions are elevated to a human (with context).

Warnings are generated during the robot's evaluation of the central database. The robot will generate these warnings using machine learning methods. These methods include active and supervised learning techniques, which are explored in depth later.

Evaluating and quantifying the quality of the alerts through the super-agents belief about component uncertainties allows for the exploration of decision space for use within an optimization model. In general, the earlier the warning is generated, the better the decision policy and the lower the probability of successful attack and organizational damage.

3.1.1 Decisions

3.1.1.1 Robot and Human Decisions

The model's decision maker consists of a super-agent that has both an AI and a human component. The concept of human-machine interaction is not new. For example, the discipline of cybernetics is concerned with control-systems involving humans, machines, and interfaces [87]. The communication between agents (machines and operators) impacts a system in a closed loop. For example, when an automobile and driver interact the two become one, de facto, entity. The automobiles interface is designed such that the human can conduct effective operations consistent with their desired outcomes, like driving to the local store.

However, the interaction in a system between artificial intelligence and human is not as well understood. With the super-agent, the two actors (robot and human) are not complementary as in the classical cybernetics paradigm. They are making the same decisions but at different thresholds where each actors strength is leveraged. The robot is better at fast-paced low ambiguity decisions, and the human is better at slower paced more complicated situations. There are obvious implications for risks and benefits with such an approach. One example where this problem is acute is fledgling self-driving (autonomous) car technology. Unlike the cybernetics example with just a human actor and automobile, there is a third entity. This third entity is an AI (robot) that operates the vehicle in place of the human at specific thresholds.

The basic concept of an autonomous vehicle is that it should be able to complete a transportation task with no human intervention. However, in practice, perfect safety and performance are not feasible in many circumstances as the technology improvement is not yet fully mature. This means that at the current state, the AI conducting the driving must, at specific points in time, give control over to a human for correction. Many such systems gauge the quality of the AI by its autonomy value or the percentage of the time the AI is in full control of the system [88]. In fact, most of these autonomous systems learn by watching the human during an intervention. The AI and the human in this system are making the same decisions but for circumstances which they are each best suited. With low information the human drives more often, and with enough information the AI drives more often. Information gathering continues until the limit of full autonomy is reached. There is also a consideration for the social perception of risk from self-driving cars (or similar technology). Even with excellent safety records, many of AI to human transitions could be mandated from external entities (such as laws, regulations, or policies).

The inherent characteristic of these systems is that the AI can only be trusted to make a subset of the decisions and when it sees a circumstance out of its decision thresholds, an expert-human must temporarily take over control. This prompts some important considerations for how the AI should be making decisions as well as how to pass the controls to the human. For example, it is an active area of research to ensure that autonomous AI technologies incorporate ethics [89] and some level of moral reasoning to avoid undesirable social biases [90]. This research also argues that AI should integrate external decision utilities and risk attitudes of the AI owner.

This model takes the approach that the AI should explicitly be an extension of the human decision maker. As the machine is learning, it must have reward functions and architecture that are identical with (not just complementary to) those of the human. Because the AI first evaluates all information and data, it must know the thresholds at which it will call on the human for each decision a priori. The super-agent ultimately reserves authority to the human, so that all decisions are consistent with their (the humans) preferences. The robot is making the same decisions as the human up to a point, like the self-driving car. It is a requirement for this model that the AI will follow these guidelines when making a decision:

- AI only makes decisions consistent with the human's risk attitude
- AI will only make decisions above (or below) a specified confidence threshold
- AI will only make decisions above (or below) a given utility threshold
- If any threshold is not satisfied, the AI will pass the decision recommendation to the human for final adjudication

The limitations of the robot are those of the programs which direct its behavior. Such programs may not be capable of fully understanding situations which require more context was trained with. The super-agent benefits from the speed and accuracy of the robot as well as the discrimination and external knowledge of the human. This type of super-agent will present itself in many domains that choose to incorporate AI into systems involving human beings or so called 'human in the loop.' Within the cybersecurity context, this paradigm is useful as a large amount of low level/threat decisions can be done by the robot, and only use expert-human intervention when pre-determined thresholds are reached. This will allow for the optimization of utility and efficient management of risk for a determined level of security.

3.1.1.2 Strategic Defensive Posture

Organizations will make decisions concerning the strategic defensive posture based on their thresholds of acceptable risk. These decisions must also be balanced against the resources they are willing to commit. Decisions manifest as purchasing decisions for risk management technologies. In Chapter 2 the topic of the security stack was discussed in detail. The stack represents capabilities provided by strategic defensive posture decisions. For example; Intrusion-Detection-Systems, firewalls, and proxies are common devices found in a stack. Also, the central repository depicted in Figure 3.1 is also the result of a strategic investment. Also, 'how much' and 'what type' of data is collected from the stack are enabled by the choice in database technology. Devices type decisions for a security stack are made infrequently. Often these choices represent a significant cost to the defended organization.

In general, the more significant the investment in the security stack and central repository the greater the decision space and data will be available to the super-agent. Also, the more capabilities, the greater flexibility in tactical decisions over a given time period. For example, the website defender discussed earlier could react much faster to threats if equipped with a repository that enabled custom real-time alerting as opposed to a simple database that only allowed for searching. However, the value of the websites protected information would need to outweigh the lifetime cost of the additional device. Each new capability has requirements for integration, training, and maintenance.

Often security is not a priority for small organizations. For example, most small businesses do not invest in basic security practices [91] and may not have a security stack at all. The cost for security technologies may not be justified by the value of the defended system or the risk preference of the organization. However, as organizations increase in value, their defensive posture expands to ensure that sensitive information and operations are adequately protected.

The strategic posture of a defender defines the decision space for responding to events. The more resilient the defensive system, the greater the number of available actions after or during an attack. At the most basic level a defensive system typically has three key features;

- Sense: Identify threats through signatures
- Block: Stop threat activity by breaking the kill chain
- Mitigation: Respond to damage done after an attack

With these three components, a defender can identify an attack, know where it is coming from, stop the threat from accessing a system, and respond when events take place. The blocking components represent the **gate set** defined in Figure 3.2 referenced earlier. These features can be at the network layer through an Intrusion-Detection-System and firewall or on a host using anti-virus or similar technology. From this primary starting point, different posture approaches include additional capabilities. The following is a list of possible additions to sensing, blocking, and mitigation.

- Sink Hole: Redirecting specific traffic to an isolated area
- Honeypot: Collecting good and bad traffic from a node intended to attract threat actors
- Infrastructure Migration: The ability to modify the defender's network topology
- Algorithm-Based Sensing: Detection through the use of statistical and mathematical techniques on sensor data
- Intelligence-Based Sensing: Collecting external information to build signatures and updated sensors, provided by human/expert analysis aggregation conducting offensive operations
- Response Actions: Capabilities that allow for some reaction to a perceived threat. For example, a 'hack back' if allowed by law

The more capabilities a defender has, the larger the decision space. However, the concept of a gate set is the same. For example, if a nation state is collecting intelligence, that information can be used to build honeypots to better observe known threat actors to understand and, ultimately, block the behavior. This depth of knowledge, provided by intelligence collection, may not be available to a corporation which would then need to rely on algorithm-based sensing. These restrictions for the private corporation are the result of legal constraints.

The internal policies and guidance complement the technical capabilities of an organizations security stack. For example, most organizations have policies in place to modify the systems that are under threat. This may include taking hosts 'off-line' or more advanced techniques of using virtualization to change the infrastructure to disrupt attacks as they are occurring. As with access controls, changes in infrastructure are driven by strategic decisions. An organization that heavily leverages virtualization has much greater flexibility. However, the machine that manages the virtualization (hypervisor) can become a single point of failure in the system. Decisions about when to change the infrastructure are typically made later in the kill chain when more severe steps have been identified. The reason for this is that the cost of these changes can be significant, often requiring human analysts to conduct a considerable amount of manual effort.

The policies that govern the use of technologies, like virtualization, impact the super-agent's decisions space in a similar way as the devices in the stack itself. Organizations will often have standard procedures which guide actions and responses of system defenders. Because these policy decisions, like device investments, are made infrequently the modeling presented in this research assumes they are static and give deference to the super-agent.

3.1.1.3 Defensive Response

The rules of cyber engagement can vary significantly depending on the timing and mission of the attacked organization. For example, legal considerations restrict private entities from engaging in offensive cyber operations [20]. However, nation-states can leverage advanced techniques to conduct improved defense which include intelligence gathering and deterrence.

In today's cybersecurity industry, private entities can purchase intelligence from third parties (if this information is collected legally). This is, however, done by exception and defensive response is often limited to internal mechanisms. With this consideration, private organizations are reduced to a decision on how much external access they should block, balancing the quality of services provided to users. For web-based companies, this can mean a significant trade between revenue generating activities and security. The more blocks in place, the costlier the false positives if they represent the loss of potential customers.

The decision space for response rules is governed by the type of organization and its sensitivity to loss of information or disruption of services. Some entities have a very low tolerance for any disturbances. One acute example is that of industrial control systems which operate critical infrastructures. In these types of systems, policies that use whitelisting techniques, discussed in chapter 2, can be common. However, with entities that depend on traffic for revenue, like online retailers, such extreme approaches, are not practical from a business standpoint.

With the proper optimal defensive posture, certain types of system access can be granted or denied. At the most basic level in a network, session-level data can be used to make this determination. For example, source/destination IP addresses or time of day can be used as decision support. With more advanced systems, session level data can be enriched by external data. These can include geospatial information or behavior information like multi-session temporal patterns. These characteristics then serve as a basis on which to adjust access control. Access controls can only be adjusted within the boundaries enabled by the strategic and policy decisions.

Access control decisions can be made in near real-time when defensive strategic posture decisions enable it. However, the posture changes at the pace of the organization's requirements and purchasing systems [92]. As discussed in the previous section, a defender grants themselves greater real-time decision space the more significant the investment in posture.

Defensive response decision can be thought of in the context of kill-chain lead time presented in the previous chapter. As warnings generated by the super-agent are observed, a gate policy is formulated to break the kill chain at its earliest stage. The earlier the chain is broken, the lower the impact on the defended organization. In a practical sense, the kill chain is broken by denying some type of activity (closing a previously open gate) on the system.

3.1.1.4 Offensive Response

For this research, offensive responses will be discussed but are not included in the modeling. Offensive activities (not represented on the influence diagram in Figure 3.4) are the most nascent of the actions that can be taken by organizations. Offensive responses are mostly limited to attacks or intelligence operations from nation-state entities. However, several private agencies in the United States have been legally authorized to participate in this domain more frequently. One significant example is Microsoft with their Digital Crimes Unit (DCU) [93]. The DCU is responsible for conducting internet-level interdiction against cybercriminal entities which include countering human trafficking, preventing software piracy and botnet 'takedowns.'

Entities that have the ability and legal framework to conduct offensive responses make up a small portion of organizations in the United States cyber domain. However, for deterrence purposes, these entities can often act on behalf of other less capable organizations. For example, after the Russian hacking attempt against the Democratic National Committee during the 2016 United States presidential election the federal government released sensitive information about their (Russian) infrastructure to the public [94].

While not a direct attack it was a defensive warning/response demonstrating the offensive activity taking place against them.

3.1.2 Uncertainties and Deterministic Functions

3.1.2.1 Cyber Actor

A cyber actor is an archetype of entities that interact with the defended system. The two base categories of actors are legitimate and threat. When conducting behavior-based threat detection (described in detail later), it is essential to identify how each actor operates. The taxonomy presented below, are example actor subcategories separated at the top level between legitimate and threat. Also, the subsets within these top-level categories have different behavior profiles. Listed in the table 3.1 are the possible actors specific to this research.

Table 3.1: Taxonomy of Cyber Actors External to the System

CATEGORY	Type	Description
Legitimate	Administrator	Maintains the defended system
	Developer	Builds and repairs the defended system
	User/Customer	Consumes system value
	Incidental	Interacts with the system accidentally or as part of an unrelated activity (ex. A user enters the wrong web address)
Threat	Competitors	Evaluates the system
	Data Collector	Scrapes information without the defenders consent
	Criminal Hackers	Attempts theft of valuable data, taking control of systems, or blackmailing
	Hacktivists	Disrupts activity for political or social motivation
	Foreign Nations	Disrupts activity, steals knowledge or collects intelligence
	Insider Threats	Disrupts activity, steals knowledge for private benefit

The primary consideration for actor classification is that the subcategories of threat behavior are often and intentionally, similar to legitimate actors. For example, to avoid detection, a sophisticated nation-state might only use techniques that are similar to a regular user. However, an unsophisticated criminal hacker might attempt brute force attacks to access parts of the defended network that are used by the system administrators or developers. Figure 3.3 shows example groupings of cyber actors with similar behavior profiles. The next section explores the set of possible behaviors of the cyber actors.

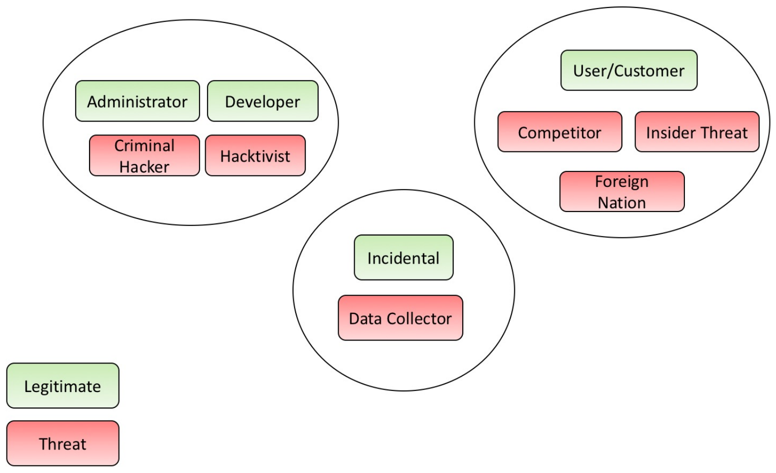


FIGURE 3.3: Example Grouping of Actors by Similar Behavior

These groupings are not a set rule but instead are intended to represent the difficulty of the classification task of distinguishing threat from legitimate actor. Each of the legitimate actors provides some value to the system, whereas a threat decreases it. The problem facing a cybersecurity defender is to ensure the system generates as much value as possible. This is done by allowing the behavior of legitimate actors while denying that of threats. The super-agent decision paradigm is made more difficult by the similarity of actors.

3.1.2.2 Cyber Actor Behavior Types

Each of the actors from the previous section has a set of behavior types associated with the interaction of the defended system. Behavior types are discrete categories, each of which has a quantifiable observation. The definition of behavior for this research is:

> Behavior: A quantifiable characteristic or combination of characteristics generated from a security device within the security stack that can be used to update the belief in the presence of an actor.

The definition is broad in comparison with traditional cybersecurity signatures which are distinct values. A behavior type has a distribution of values that can, potentially, be used for actor classification. Consider the website defender. One characteristic they can observe from all users is time-of-day. It may be the case that legitimate actors use the website during regular business hours and threats attempt to access it late at night. This behavior (specific time-of-day) can then be used to inform decisions about the gate set by the super-agent.

Data collected from security stack devices coupled with actor classification information (through analysis or intelligence) creates a behavior signature for a specific actor. Historical data collected about these behaviors can be used in machine learning methods. In this model, the features discussed in chapter 2 are synonymous with behavior type. For example, the defender observes a criminal hacker using port 22 from a source IP address located in Nigeria, or a customer using port 443 during regular business hours. These behavior types (ports, source IP location, time-of-day) have observable quantities. With enough collected data, accurate classification becomes possible.

Unlike the number of steps in the kill chain, there is a broad range of characteristics that can be classified as behavior. Behavior is an aggregate or combination of a set of observed behavior information (example types are shown in Table 3.2). In this view, an actor produces an algorithmic (behavior-based) signature. Such signatures are generated as outputs of machine learning models and become alerts in the warning system. The specific low-level behavior types represent the machine learning models feature set, which includes but are not limited to the examples shown in table 3.2.

Each of these behavior types can be used as independent variables to aid in the task of classification. Depending on the actor, the behavior will generally manifest differently. One example would be frequency. For a low-skill actor, the rate of individual signatures (i.e., how many times an IP address generates events) can be very high. However, more advanced actors might seek to avoid drawing attention to obvious spikes in activity. Threats that originate from inside the network (so-called insider-threat) have their own set of behaviors. The behavior of this type might be found in the pattern of life analysis, as discussed earlier with the website defender, where activity is happening after typical work hours. With all behavior types, the observations are mapped to actors with some degree of uncertainty.

Historical behaviors are realizations of probability distributions that have been observed and are associated with a known actor. By contrast, specific behaviors are observations for which the actor is unknown. In the AI construct, a machine will learn to categorize actors through the use of historical data. The robot's performance will be based on how well it matches specific behavior sets to unknown actors.

Table 3.2: Behavior Types for Cyber Actors

CATEGORY	EXAMPLE	DESCRIPTION
Signature	IP Address	network level
	MAC Address	physical level
	Geo-location	location based
Temporal	Inter-arrival time	time between connection attempts
	Time of day	patterns around work cycles
	Pattern of life	general patterns for actors (such as work hours or days)
Frequency	Top talkers	the highest frequency
	Infrequent visitors	seldom visits
Other	Amount of data sent/received	level of activity with the network
	Promiscuity of behavior	number of places visited on the network
	Type of interactions	using different network resources
	Protocols used	type of connections

3.1.2.3 Open Source Signatures and Threat Intelligence

The condition of general (global) cyber threat activity at a given point is fluid and based on several factors. Impacts on the defender of known and unknown software vulnerabilities, organizational positioning, and hacker capabilities all contribute to trends that affect the effectiveness of a defense. Patterns are found in observed behavior and specific target types across the open internet and related organizations. Research of external trends (information gathered from other organizations) can be applied to local security decisions by learning from and incorporating the lessons into the defender's posture. One example is the use of anti-virus signature updates. Most updates of this type occour with the discovery of a virus on another, but similar, organization's system. This discovery then benefits all organizations with the same anti-virus software.

With most organizations conducting cybersecurity, information is generated internally (threat intelligence) and collected externally (open source or shared signatures). Specifically, open-source security communities or custom-developed threat intelligence directly affect the defenders posture. These sources of data are quantifiable and actionable. These quantities, if available, are observed and available to support security decisions.

An industry has grown around these datasets including both free and community-curated, as well as paid services. Some of the most significant contributors to this type of open list are government agencies responsible for protecting private entities.

3.1.2.4 Cyber Threat

Table 3.3: Mathematical Notation for Cyber Threat Model

Notation	Description
n	number of total actors (of all types) arrived at a gate
k or x_k	number of observations of a specific actor type for a specific gate
p_k	probability of actor at a specific gate
$\hat{p}_k$	estimated probability of an actor at a specific gate

In the previous section, cyber actors are described using a top-level dichotomy of legitimate or threat. The uncertainty of a cyber-threat listed here is specifically relevant to observed and historical behaviors. In other words, this is the probability of a threat that is knowable and learned by the super-agent. Threat types can be modeled using a cardinal discretization and multinomial distribution for a given gate or behavior. When an actor arrives at the system and exhibits some particular behavior, the actor type is sampled (or realized) from this distribution. Learning the parameters of this distribution for each gate over time is the goal of the super-agent. In a dynamic system, this distribution can change based on actions and incentives by the defender or actor. For example, if a website defender blocks all user traffic late at night (closes a gate), the threat actors may change and become active during regular business hours (move to another gate). As the super-agent learns the distribution, there must be some allocation for this type of dynamics. However, for simplicity, the initial model formulation will assume static actor behavior.

For the static model, at a minimum, a binary threshold can be used (threat or not-threat). For a given mix of actors to arrive at a particular gate are governed a binomial distribution.

$$f(k, n, p) = \Pr(k; n, p) = \Pr(X = k) = \binom{n}{k} p_{threat}^{k} p_{notthreat}^{n-k} \tag{3.1}$$

Where p_{threat} and $p_{notthreat}$ are threat and not threat probabilities, respectively, and $p_t + p_{nt} = 1$. This is expanded to the multinomial distribution for m actors.

$$f(x_1, \ldots, x_k; n, p_1, \ldots, p_k) = \Pr(X_1 = x_1 \text{ and } \ldots \text{ and } X_k = x_k) \tag{3.2}$$

$$= \begin{cases} \dfrac{n!}{x_1! \cdots x_k!} p_1^{x_1} \times \cdots \times p_k^{x_k}, & \text{when } \sum_{i=1}^{k} x_i = n \\ \\ 0 & \text{otherwise,} \end{cases} \tag{3.3}$$

Here p_k is a specific actor type. These distribution parameters are unknown at the beginning of the super-agent's time and must be learned through experience. Threat actors modeled as discrete probabilities based on a gate set (behaviors) has practical advantages. For example, a cyber-criminal, that does not discriminate among the targets which they attempt to exploit, might have a much higher probing rate than a more sophisticated actor. Also, the cyber-criminal may operate on port 22 (an example of a known system administrator port) and the advanced threat on port 443 (a standard user port). These behaviors are then used to update the probability that given actor is a threat. Learning methods will be discussed in greater detail in later sections.

3.1.2.5 Successful Attack

A probabilistic representation of a successful attack is modeled using a decision framework such as a Markov Decision Process where stages are steps in the kill chain. The transition probabilities depend on the decision space and the threat type. Different probability thresholds can inform changes in the system and rewards if they are identified during earlier stages.

This research uses the early warning system framework to inform this probability during the first few stages of the kill chain. Also included are the dynamics and uncertainties about the lead time of each step and time to intrusion. These characteristics are reflected in the probabilities of transition. For an attack to have some impact, a threat must transition through the stages of a kill chain until an effect is observed, successful or not.

3.1.2.6 Organization Type

The type of organization being defended is relevant to the probabilities of different actors. For example, a public internet-facing entity is regularly attacked with low-level hacking attempts. By contrast, high-value organizations are targeted explicitly with more persistent and advanced techniques. Advanced actors that target high-end entities are often referred to as 'Advanced Persistent Threat' or APT [95]. These bookends of ability represent a spectrum of cyber threat capabilities. While there is a continuum of organization types Table 3.4 details the taxonomy used in this research.

Table 3.4: Defenders Organization Type Taxonomy

Defender Category	Type
Government	1st Tier Nation
	2nd Tier Nation
	Developing Nation
	Local or State Government
Private	For Profit Corporation - Large
	For Profit Corporation - Small
	Healthcare Organization
	Not-For Profit Corporation
	Religious Organization
	Community Based Organization

How an organization positions itself (its posture), the value of information and its sensitivity to threats play a role in the type and severity of attacks. 1st and 2nd tier nations may have the ability to conduct large-scale offensive and defensive operations. As such they, along with large corporations, have the greatest value to protect. Smaller organizations or local governments may not have enough value or resources to conduct a sophisticated defensive strategy. As a general rule of thumb, the larger the organization, the higher the value of the defended system and the more threat activity will be present.

The exception to this rule is that of a small entity protecting a significant value. This smaller entity may be targeted by sophisticated actors who discover this asymmetry. An example of this is small defense contractors that maintain databases containing expensive government-funded technology. In several high profile cases these organizatioins and have been specifically targeted. In the now well-known case of the F-35 fighter, Chinese hackers stole much of the plans for the airplanes advanced technologies from a small firm with poor security practices. This pattern has also been observed with personally identifiable information being stored in subcontractor databases on behalf of larger organizations. For example, healthcare records, credit reports, security clearances, and education transcripts have all been stolen from small organizations acting in such capacities.

3.1.3 Cost

For this model, the impact of a successful attack is a function of the utility of the protected network. Defensive decision policies are intended to safeguard organizational value. They also have the potential to impact it adversely. An extreme defensive policy results in a system so restrictive that it affects use by legitimate actors. On the other hand, a defensive policy that is too permissive can result in significant damage to the protected network.

These bookend policies, of extreme restriction or extreme permissiveness, are not desirable and do not represent a quality approach to defense. Some balance between them represents the optimal trade-off. The network defender must find the balance point where the network is providing required services but still denies enough threat activity as not overly to burden the organization.

The formulation of cost is a quantified as a utility function where a set of distributions govern the costs and benefits of threat and legitimate activities at given gates in the kill chain stages. This metric (cost) then becomes the primary variable of interest in the optimization technique which supports the super-agent policy decisions. The challenge in selecting a defensive policy is that, in many cases, impacts are not realized for many stages in the future. This implies that the super-agent may be making decisions in a stochastic and possibly time-discounted paradigm. The optimal policy for the network defender will be the set of actions that result in the highest expected value of its utility and minimal impact over the life of the system.

3.1.4 Diagram

The purpose of an influence diagram is to graphically represent the relevance and uncertainty of the systems components. The previous sections describe the model components and their justification. Figure 3.4 is the model depicted as an influence diagram.

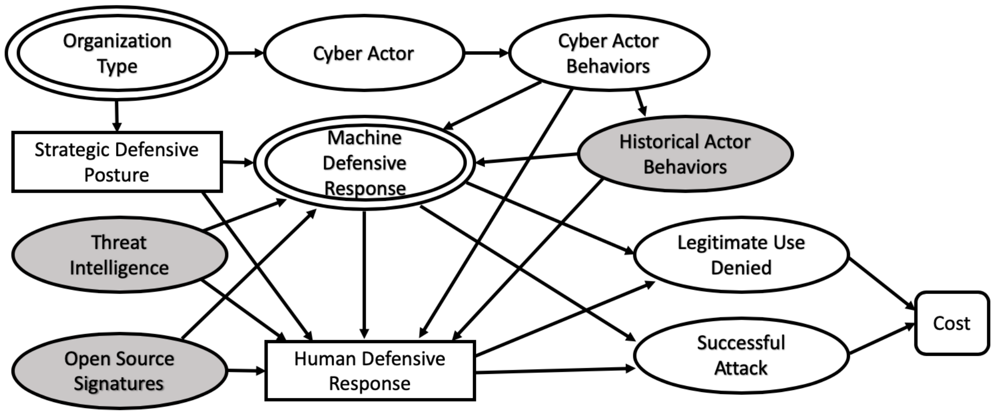


FIGURE 3.4: Influence Diagram for Cybersecurity Defender

The system modeled in Figure 3.4 is that of the risk of a cyber-attack at a single point in time and the related uncertainties from the defender's point of view. It includes decisions, random variables, and events. The following is a brief description of the nodes (which are explained in detail in the previous sections).

- Org Type: One of the organizations listed in Table 3.4
- Strategic Defensive Posture: The decision on technical defensive capabilities.
- Cyber Actor: One of the actors listed in Table 3.1
- Cyber Actor Behavior: A set of the behaviors listed in Table 3.2
- Threat Intelligence: Near term information gathered external to the defended system about threat actors
- Open Source Signatures: Near term open source information collected about threat actors
- Historical Behavior: Long term overserved behavior associated with specific actors on the defended system
- Specific Behavior: The current combination of attributes for the behaviors of an actor in real-time
- Cyber Threat: The short-term probability that the current actor is a threat
- Machine Defensive Response: The decision that the robot makes
- Human Defensive Response: The decision that the human makes given that the robot passes the decision
- Successful Attack: The uncertainty of a successful attack if the actor is a threat given the signals
- Legitimate Use Denied: The uncertainty of a denial of legitimate actor behavior
- Cost: realized by the defender

There are many facets to a defensive decision for cybersecurity. Some items not considered here include compliance, organizational public image, and industry trends. However, the model presented contains all the necessary components relevant to the super-agent. In general, a cyber actor arrives at the defended network and displays a set of behaviors. The uncertainty of cyber actors is influenced by the type of organization, and then the behaviors exhibit a quantifiable realized value. These realized behavior values, along with external and historical information influence the uncertainty of the actor being a threat. These behaviors are used by the robot, based on its pre-programmed functions and learning, to generate warnings. The likelihood of the actor being a threat is relevant to the super-agents defensive response (or defensive response policy). The defensive response then becomes relevant to the probability of a successful attack along with the likelihood of the actor being a threat, organization type, and actor. A successful attack and legitimate use being denied is then relevant to the cost or impact on the defended system.

3.2 Mathematical Model

In the previous sections the model is described contextually and graphically. Here the components are expressed quantitatively to allow for exploration and explicit programming of the super-agent. The following subsections begin with notation and then aggregate model components to construct a set of decision algorithms. The algorithms are the explicit programs that govern the robots activity.

3.2.1 Basic Notation

Table 3.5 explains the mathematical notation indexing and representation used for the remainder of this research.

TABLE 3.5: Mathematical Notation for Model

Notation	Description
s	step in a kill chain (also attack state)
n	index of step in a kill chain (also attack state)
a	index of action that a security super-agent can take to defend a system
i	index of cyber actor type (threat or not)
x	behavior
b	index of behavior type
o	index of defended organization type
$P_{i,a}$	Stochastic transition matrix for kill chain steps for a given actor and action
t	discrete time period, related to stages in the kill chain
g	index of gate at a given kill chain step
G_t	vector of gate states at time t also the decision policy
$p_{g,i}$	probability of actor i at gate g
$\hat{p}_{g,i}$	estimated probability of actor i at gate g
γ	risk preference t.
$u_t(g, y_t, r_t, \gamma)$	utility for gate and time.
$\hat{u}_t(g, i)$	estimated utility by actor, for each gate at time t.
$\hat{u}_t(g)$	estimated utility for each gate at time t.
f_g	multinomial distribution of actors at gate g
y_t	observation of actor at time t.
$\hat{y}_t$	machine learning estimate of actor at time t. The same as $\hat{p}_{g,i}$.
r_t	observation of reward at time t.
e_t	external information observed at time t.
D	human directed policy
ϕ	threshold for human intervention

3.2.2 Parameters

The model relies on a set of parameters which dictate the inputs and probability distributions. The following list describes these values and maps them to taxonomies discussed in earlier sections.

- There are a total S kill chain stages indexed by n governed by the taxonomy in Table 2.1
- There are a total of I actor types indexed by i governed by the taxonomy in Table 3.1
- There are a total B behavior types indexed by b governed by the taxonomy in Table 3.2
- There are a total O defended organization types indexed by o governed by the taxonomy in Table 3.4
- There are a total T actions that the super-agent can take indexed by a governed by the taxonomy listed in section 3.1.1.2

These parameters become the foundation for the stochastic components described in the following sections.

3.2.3 Machine Learning with a Known Gate Set

The topic of machine learning concerns itself with training a machine to perform a specific task. Often, the machine learns to recognize new signals, through the updating of probabilities. There are two cases of application for the super-agent: the machines make the decision or the machine alerts and guides the decision makers. The goal for the super-agent is to find the optimal gate policy (which gates to keep open and which to close).

In this section the case of a known gate set is assumed. The super-agent is not allowed to explore additional decisions beyond opening and closing specified gates. For example, a defender, due to security stack control restrictions, may only be able to control the blocking of IP addresses or ports. This then becomes a finite and knowable gate set for which the super-agent seeks an optimal policy.

Each gate has some unknown distribution (f_g and $u(g, i)$), of actors and utility. The scenario considered in this research is that at each time step an open gate has a single observation of an actor, with the type governed by a multinomial distribution, and reward. The utility distribution is continuous and unique to all combinations of actors and gates. It (the utility) is a function of actor, reward and risk preference.

$$f_g(p_{g,1}, p_{g,2} \dots p_{g,I}) \tag{3.4}$$

f_g has the pdf depicted in Equation 3.2 and has the following parameter constraints.

- Outcomes are Actor Types: $1, 2, \dots, I$
- Distribution of $(p_1, \dots, p_I)$, $\sum_{i=0}^{I} p_i = 1, \quad p_i \geq 0, \quad \forall_i = 1, \dots, I$

Because the distribution f_g is unknown, like $u(g, i)$, the parameters need to be learned. With machine learning the estimates are learned and improved over time with experience. There are several approaches to estimation and learning covered in the next section. The common consideration is that the estimation of the gate distribution parameters will be a function of historical behavior and actor observations as well as external information from open source and intelligence activities.

$$\hat{p}_{g,i,t} = f(y_1, \dots, y_{t-1}, x_{1,1}, \dots, x_{b,t}, e_1, \dots, e_t) \forall b, t \tag{3.5}$$

The behavior x has both historical values from the beginning to $t-1$ and current time values t. The historical values have observed outcomes (behaviors and classified actors y_1 to y_{t-1} which are useful for learning. The external information e_1 to e_t is also available from the beginning to current time. The estimation of these parameters then represents the robots understanding of the activity at a given gate and can be used as an input for the decision framework.

As mentioned above there is an unknown distribution of utility at each gate for each actor. The utility for a given and gate set are an unknown continuous distributions which are function of gate and actor. A vector of conditional utility distributions can be described as follows.

$$\mathbf{u}(g, i) = \begin{bmatrix} u(g,1) \\ u(g,2) \\ \dots \\ u(g,I) \end{bmatrix} \tag{3.6}$$

At a given gate when an actor arrives the realization is modeled as a one hot vector where $p_{g,i} = 1$ for the single actor and $p_{g,j} = 0$ where $j \neq i$. Similar to the multinomial distribution the $u_t(g, i)$ distribution needs to be learned by the robot. The estimation of the distribution of utility is done in the same way as f_g.

$$\hat{u}_(g, i) = f(u(y_{1,1}), \ldots, u(y_{t,I}))\forall g, t \tag{3.7}$$

The number of utility distributions per gate is equal to the number of actor types. Figure 3.5 is a representative decision tree for each gate.

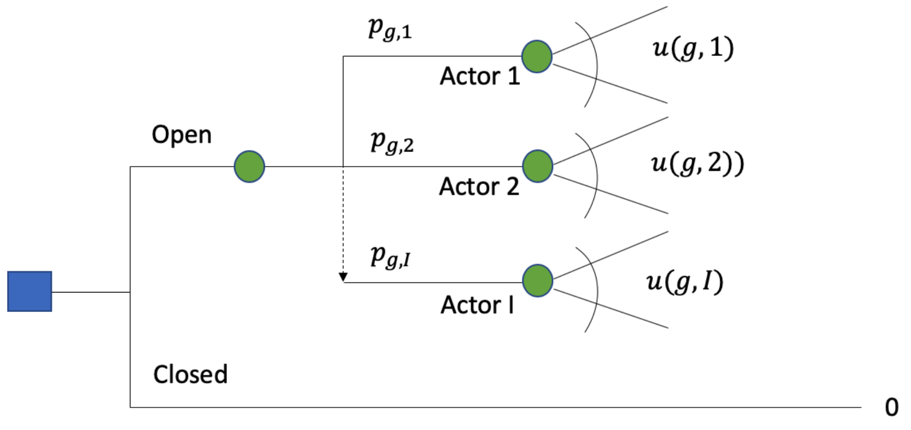


FIGURE 3.5: Decision Tree For Each Gate

Once the distribution parameters are estimated (or assessed as a prior) the expected utility for each gate is calculated. For the single threat actor (binary with legitimate as 0 and threat as 1) case, the expected utility at time t is given by:

$$E[\hat{u}_t(g)] = \hat{p}_{g,0}(\hat{u}_t(g, 0)) + (1 - \hat{p}_{g,1})(\hat{u}_t(g, 1)) \tag{3.8}$$

For the multi-actor case, the expected utility at time t where I is the number of actors in the system is given by:

$$E[\hat{u}_t(g)] = \sum_{z=0}^{I} \hat{p}_{g,z}(u_t(g, z) \tag{3.9}$$

These expectations are used as the basis for generating a gate policy. However, the performance of the selected policy depends primarily on the accuracy of the parameter estimates. The concern is that a robot with minimal historical data will have difficulty learning accurate estimates. The objective then becomes the speed of learning. The following sections discuss several methods that can be leveraged.

3.2.3.1 Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) is a classical statistical technique that seeks to find the parameters of a distribution that maximize the likelihood of observed data.

$$L(\theta) = f(x_1;\theta)f(x_2;\theta)\dots f(x_n;\theta) \tag{3.10}$$

The objective is to find the value of θ which maximizes the function $L(\theta)$ with the observed data x_n. This technique can be readily applied to the discrete distribution of actor types. Consider the binomial actor case (threat or legitimate).

$$f(x;p) = \begin{cases} p^x(1-p)^{1-x}, & \text{when } x = 0, 1 \\ 0 & \text{otherwise,} \end{cases} \tag{3.11}$$

It can be shown that maximizing the likelihood yields a simple estimation.

$$\begin{aligned} L(p) - p^{x_1}(1-p)^{1-x_1}p^{x_2}(1-p)^{1-x_2}\dots p^{x_n}(1-p)^{1-x_n} && (3.12) \\ = \prod_{i=1}^{n} p^{x_i}(1-p)^{1-x_i} && (3.13) \\ = p^{\sum_{i=1}^{n} x_i}(1-p)^{n-\sum_{i=1}^{n} x_i} && (3.14) \end{aligned}$$

Consider for this function the value $\hat{p}$ maximizes $L(p)$. It follows that $\hat{p}$ also maximizes $ln(L(p))$. Therefore.

$$lnL(p) = (\sum_{i=1}^{n} x_i)ln(p) + (n - \sum_{i=1}^{n} x_i)ln(1-p) \tag{3.15}$$

Then.

$$\frac{dlnL(p)}{dp} = \frac{(\sum_{i=1}^{n} x_i)}{p} - \frac{(n - \sum_{i=1}^{n} x_i)}{1-p} \tag{3.16}$$

Setting this equation to 0 and solving for p yields.

$$\hat{p} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{3.17}$$

This is generalized to the multinomial case with the same result, parameter estimates are the percentage of times an actor is historically observed. Similarly, if the utility distribution is considered to be Gaussian then MLE can also be applied. Consider the case when variance (σ) is known but the mean (μ) is not.

$$L(\mu) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} exp[\frac{-(x_i-\mu)^2}{2\sigma^2}] \tag{3.18}$$

Then it follows that $\hat{\mu}$ maximizes both $L(\mu)$ and $lnL(\mu)$ so.

$$lnL(\mu) = -(n-2)ln(2\pi\sigma^2) - (2\sigma^2)^{-1}\sum_{i=1}^{n}(x_i-\mu)^2 \tag{3.19}$$

The derivative of this equation is

$$\frac{dlnL(\mu)}{d\mu} = (\sigma^2)^{-1}\sum_{i=1}^{n}(x_i-\mu) \tag{3.20}$$

Setting this equation to 0 and solving yields the familiar estimator.

$$\hat{\mu} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{3.21}$$

The use of MLE allows for efficient and intuitive parameter estimation. In the fast-paced cybersecurity environment the speed and simplicity of this technique could be beneficial. However, the quality of the estimates depends on collecting a sufficient amount of data and a non-changing system. Also, this method only leverages historical observations of actor types per gate and does not use other behaviors or external information which could greatly improve the estimate quality and may allow for faster convergence towards knowledge of the true distributions.

3.2.3.2 Bayesian Conjugacy

The issue with the use of MLE is that it is difficult to incorporate external (expert) information and may require a large volume of data. An alternative is to use a Bayesian estimation technique which leverages conjugate distributions. The philosophy of the Bayesian method is that the super-agent starts with a set of prior distributions (beliefs about the distributions). As data is observed the priors are updated.

The benefit of conjugacy is that it can be computationally convenient to update a prior distribution with observed data. This is because the prior and the posterior have the same form. If a pair of distributions are conjugate there is a closed form updating calculation. Most distributions are not conjugate, however there are several convenient pairs which are useful for this application.

The estimation of parameters for the binomial actor case can be modeled as a Beta/Binomial conjugate pair. The beta distribution represents the prior distribution of belief about the probability of a given actor type at a specific gate. The binomial distribution 3.11 is the likelihood function populated by observed data. Consider a gate set with a set of priors that are beta-distributed with parameters $\alpha \in (\alpha_1, \ldots, \alpha_G)$ and $\beta \in (\beta_1, \ldots, \beta_G)$. The prior probability is a function of θ_g and takes the form.

$$p(\theta_g) = \frac{\Gamma(\alpha_g + \beta_g)}{\Gamma(\alpha_g)\Gamma(\beta_g)} \theta_g^{\alpha_g - 1}(1 - \theta_g)^{\beta_g - 1} \tag{3.22}$$

In this formulation the Γ denotes the gamma function. The functional form of the beta distribution is conjugate with the binomial so that information collected from the later can be used to directly update the former. The updating scheme is quite simple; for each data point if a threat is observed add one to the alpha parameter else add one to the beta parameter

$$(\alpha_g, \beta_g) \leftarrow \begin{cases} (\alpha_g, \beta_g), & \text{if } y_t \neq g \\ (\alpha_g, \beta_g) + (y_t, 1 - y_t) & \text{if} y_t = g \end{cases} \tag{3.23}$$

The beta/binomial conjugacy can be generalized to the multinomial case using the Dirichlet distribution as a prior (this is discussed in a later section). Similar to the MLE technique, if the utility distribution is assumed to be Gaussian with a known variance (σ^2) a similar updating scheme can be used.

$$(\mu, \sigma^2) \leftarrow (\frac{1}{\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}} \left(\frac{\mu_0}{\sigma_0^2} + \frac{\sum_{i=1}^{n} u(y_t)}{\sigma^2} \right), \left(\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2} \right)^{-1}) \tag{3.24}$$

In this formulation $u(g, i)$ are the observed utility values for a given actor and gate. The usefulness of this technique is that the super-agent can start with a prior distribution allowing for an initial gate policy in line with their belief. In addition, the prior distributions can be updated using external information through the dynamics of parameter adjustments (like adding value to an α_g parameter based on some intelligence report).

3.2.3.3 Metalog Updating

In the previous sections two well-known methods of statistical estimation where discussed, MLE and Bayesian techniques. The major drawback with both of these methods is that a distribution shape must be assumed a priori. Often distributional assumptions are practical and valid while other times they are simply a convenience. For example, in the Bayesian updating scheme the distribution for utility is assumed to be Gaussian and the variance is known. However, in practice these two assumptions are difficult, if not impossible, to validate.

This research presents a novel concept of using a modern class of probability distribution to update beliefs. The metalog is a highly flexible continuous distribution that takes its shape directly from observed data without opinion [96]. A detailed review of these distributions and their modification for this research is contained in Appendix D.

The metalog distribution takes the shape of the data that it represents. However, the process of generating a distribution requires some user input. In particular, the amount of flexibility a distribution has is determined by the number of terms [1]. Also, the number of terms must be less than the number of available data point observations.

Each term has two components, an a_i value and an $f(y_i)$ value where y_i is a cumulative probability value for a given data point quantile. The $f(y_i)$ values are determined by the characteristics of the metalog terms as defined in [96]. The distribution is built using an optimization algorithm that finds the best fit a_i values (real numbers) to a vector of observed data x. Once the distribution is built the a_i values become constant parameters. For example, equation 3.25 is a four term metalog quantile distribution.

$$M(y) = a_1 + a_2 ln(\frac{y}{1-y_1}) + a_3(y-0.5)ln(\frac{y}{1-y}) + a_4(y-0.5) \tag{3.25}$$

Figure 3.6 shows the flexibility of the metalog on an example multimodal data set as terms increase. While the underlying data is the same, the distribution gives flexibility to represent the informational nuances of the raw data with more terms.

[1]metalog terms are similar to the concept of degrees of freedom. The more terms the more flexible the distribution to fit the data.

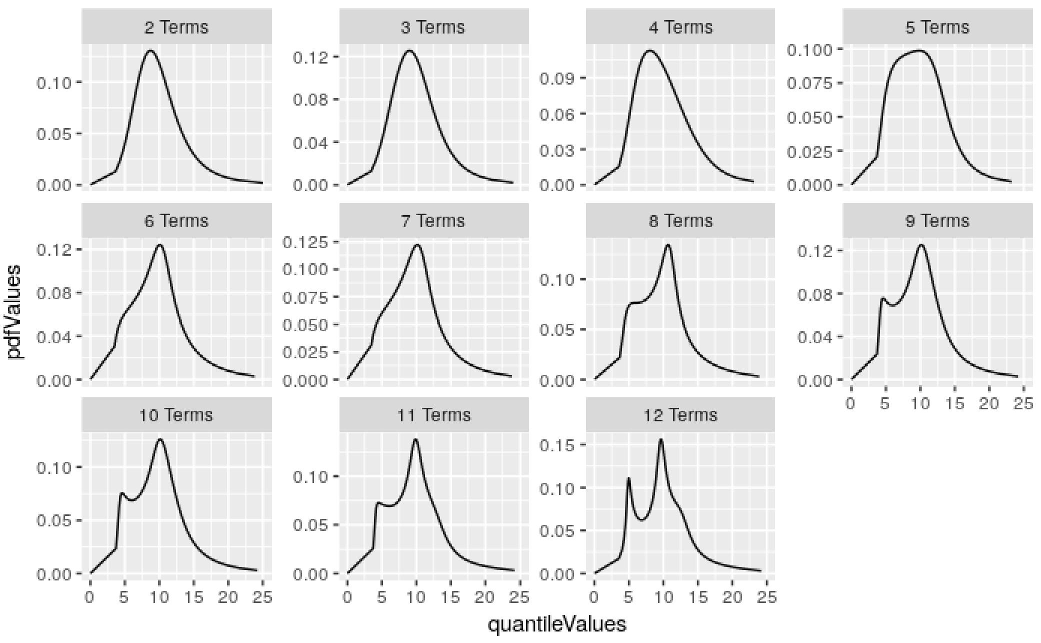


FIGURE 3.6: Example Metalog Distribution Demonstrating Term Flexibility

The updating scheme for this research is to rebuild the metalog distribution with every newly observed data point. As actors arrive at a gate the type and utility are observed then appended to the other historical observations. This approach is more intuitive than the Bayesian approach of needing to modify parameters of an assumed distribution. Also, it requires much less data than the MLE approach to build robust distributional shapes. In the dynamic setting where the underlying system characteristics are changing the metalogs potential is the greatest. When building a distribution older information can be left off or replaced at the super-agents discretion.

One concern is that the flexibility of the metalog invites overfitting. Specific applications may be more sensitive than others to noisy observations. In addition, the continuous distribution may not be useful for estimating discrete probabilities the way the conjugate pair Beta/Binomial are. In the section supervised machine learning techniques are presented. In such contexts the gate set is not defined a priori and must also be learned by the super-agent.

3.2.4 Machine Learning with an Unknown or Partially Known Gate Set

In the previous section it was assumed that the super-agent had a predefined and static gate set from which to sample and update beliefs. In practice, this can be a reasonable assumption. Defenders work on updating access control lists, as discussed in Chapter

2. However, these techniques have limitations in many settings due to the number of possible gates.

For example, it is reasonable to monitor a predefined gate set of activity from individual countries. This set would only contain a few hundred entities (195 countries) to collect and update data. However, it is not as feasible to monitor and update a gate set that is significantly larger. For example, consider a set that includes each IP address and port combinations. There are 256^4 IPv4 addresses and 65534 ports. This combination results in a gate set of size 2.18×10^{14}. This is, obviously, not manageable in a practical setting. Not only is the number of gates enormous, many would have sparse activity with few observations from which to build accurate estimates.

In order to address the scale problem this research proposes the use of supervised machine learning algorithms. These techniques can use observed behaviors (IP address and ports) as inputs and return an actor type probability. In addition, techniques for encoding high-dimensional features can be leveraged to reduce complexity. Once the machine has learned an accurate model from historical data. The model can be explored to determine a gate set. This exploration takes the form of sensitivity analysis on input features.

There are a large number of supervised learning classification techniques. Methods have technical differences but are conceptually the same. The robot uses new information to update probabilistic beliefs that determine a gate set policy. The generation of actor type probabilities is known as classification.

There is a rapidly growing body of knowledge for general-purpose machine learning classification methods [97]. These techniques fall into a particular, so-called, category of supervised learning referenced earlier. As such, classification techniques learn from reliable historically recorded data that contain observations with the dependent and independent variables. Specifically, this research is concerned with training for a classification task using early-stage events for prediction and parameter estimation purposes.

Supervised learning borrows its name from the concept of learning using known, reliable historical mappings. That is an observed feature set $\mathbf{x}$ is mapped to an observed dependent variable y (sometimes called x, y pairs). The quality of the machine learning technique is judged by how well the algorithm can map these pairs. This mapping is typically measured by a, so called, cost function. The cost is some measure of distance (mean square error, absolute distance, p-norm, etc.) between $\hat{y}$ and y, the machines prediction and the actual outcome.

Figure 3.7 is a graphical depiction of the supervised learning construct using example nodes and edges representing features and mappings. The generalized paradigm is that the behaviors are fed into an algorithm as features which produces a prediction.

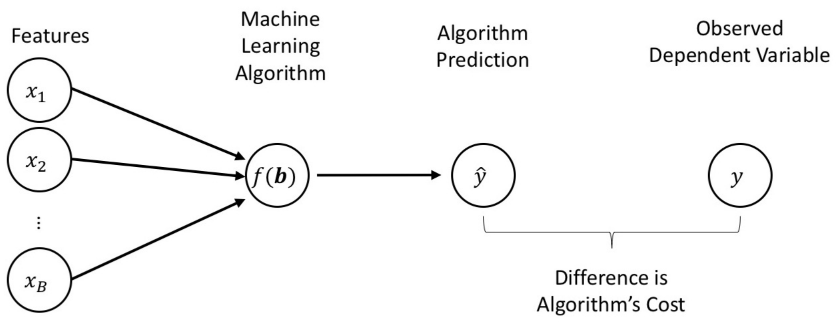

Features
Machine Learning Algorithm
Algorithm Prediction
Observed Dependent Variable
x_1
x_2
x_B
$f(\boldsymbol{b})$
$\hat{y}$
y
Difference is Algorithm's Cost

Figure 3.7: Basic Supervised Learning Paradigm

That prediction is then compared to the observed value. The total difference across all observations represents the resulting cost and model quality.

Data collected from a network and host devices can give indications of behavior for a specific cyber actor (legitimate or threat). However, unlike the number of steps in the kill chain, there is a broad range of characteristics that can be classified as behavior. If behavior can be broadly defined an actor can produce a digital signature in many ways that are, for practical purposes, limitless (re the combinatorics issue discussed earlier). Such signatures become features for a machine learning model and include but are not limited to the examples shown in table 3.2.

The important characteristic of supervised learning is that models learn over time using historical data which contain both features (behaviors in this case) and dependent responses (actor types). In the context of the gate-set the super-agent is creating a policy which can be improved by incorporating actor specific behavior information in addition to the gate itself. For example, consider a website defender who has defined their gate-set as opening and closing inbound ports. With MLE and the Bayesian techniques discussed earlier the super-agent is simply observing the distribution of actors that arrive at each port. However, the probability estimate for each actor can be improved if other behaviors, in addition to the port, are used to classify. Other information could include IP address, time of day, and frequency of connection attempts.

Each of these behavior types can be expanded in a broad range of observations to be used as independent behavior variables. Depending on the actor, the behavior will manifest differently. One example would be frequency. For a low-level actor, the rate of unique events (how many times an IP address is seen) can be very high. However, more advanced actors would avoid drawing attention from apparent spikes in activity. Threats that originate from inside the network (so-called insider-threats) have their own set of behaviors. Insider-threats might have differentiation found in pattern-of-life

Table 3.6: Notation for Supervised Learning Models

Notation	Description
i	Index of actor types
x_b, X	Observed behavior value (indexed at b)
$\mathbf{w}$	Vector of weights applied to behaviors
B	Number of behavior observations
J	Cost function
h	Index of historical observation

activity occurring after typical work hours. With all behavior types, the observations are mapped to actors with a degree of uncertainty. Table 3.6 gives the notation to model the probability that a behavior observation categorizes a given actor type.

Cyber actors a have a behavior x_b at kill chain stage s and are classified by actor type y_i. For example, a Russian hacker (actor y_i) would have a specific session arrival rate (behavior x_b) for reconnaissance activities (stage s). This formulation assumes that the behavior is observable by the defender (from the security stack) and recorded as historical data. All of the supervised learning techniques will use this formulation and the paradigm presented in Figure 3.7.

3.2.4.1 Behavior Sensitivity for Gate Set Policies

As a learning technique is trained against historical data the prediction accuracy improves and, at some threshold, justifies greater automation. How the model is used can have different manifestations. For example, the most basic case is that the model is used for alerting. Every actor that arrives at the network has their behaviors passed through the algorithm and receives a score. If that score is above some threshold an alert is generated for the humans consideration and further investigation. However, alerting is passive defense and does not allow for interdiction of the actors before they arrive. If the algorithm is of sufficient quality, it should lend itself to analysis which produces a gate set decision policy of similar quality.

Backing out a gate set policy from a trained algorithm is done using techniques borrowed from sensitivity analysis. A simple approach is to examine the sensitivity of $\hat{y}$ to a given individual behavior x_b or sets of behaviors. The expression of sensitivities is a differential equation representing prediction change with respect to behavior. For example $\frac{d\hat{y}}{dx_b}$ in the single feature case or $\frac{d\hat{y}}{d\mathbf{x}_b}$ in the multi-feature case.

These differential equations take various forms depending on the chosen supervised learning method. In later sections each technique will explore the formulation of input sensitivities. Then sensitivities can be optimized in a similar way as the known gate set methods. A goal for a supervised learning technique would be to have convex continuous sensitivity functions that allow for convenient optimization using well known methods. However, this may not always be possible. For example, use of some of the more advanced deep learning techniques (discussed in later sections) can make developing these sensitivity functions challenging and do not guarantee convexity or continuity.

3.2.4.2 Cost Functions

In order to train a supervised machine learning algorithm some method of best fit to historical data is used. The type of fitting method is called a cost function (a measure of error between model predictions and observed data). The machine requires the use of an accumulation of reliable historical data to perform this task. Finding the algorithm which produces an optimal (typically minimal) cost implies that a machine can potentially help wade through noisier early stage data to provide an identification (classification) before significant business-related costs are realized. One critical point is that a machine learning cost function is not the same as the cost referred to in the early warning system. The latter is a reference to financial cost or value, and the machine learning cost is specific to the model training.

Cost function can take many forms. However, they all attempt to perform a similar role. Optimization techniques play an important role because the goal of the machine learning method is to minimize total cost between a model prediction and realized values. Ideal attributes of a cost function are convexity and simplicity. Similar to the discussion in Appendix D regarding the metalog distribution fitting using ordinary least squares (OLS). Every supervised learning model will require a similar approach.

Because one of the applications of interest is to identify actor types. Cost functions that produce estimates of classification are needed. In the most basic case classification is binary, threat or not-threat. The traditional logistic cross entropy cost for the binary classification case, $J(\theta)$, is given by the following function.

$$J(\theta) = -\sum_{h=1}^{H}(y_i log(\theta(\mathbf{x_b}) + (1-y_i)log(1-\theta(\mathbf{x_b}))) \tag{3.26}$$

This function sums across an observed data set. Where $\theta(x) = P(y|\mathbf{x}_b)$ and y_i is the true classification (0 or 1) of an observation and there are H historical observations. While this cost function is known to be convex and simple it does not have a closed form solution like OLS. In addition, this particular formulation is only useful in the binary classification setting. However, a minor modification to this formulation can handle larger sets of classification categories. With a vector of probabilities ($\hat{\mathbf{y}}$) is then used to calculate cost in a similar manner as Equation 3.26.

$$J(\theta) = -\sum_{h=1}^{H} [\sigma(\theta, \hat{\mathbf{y}})\mathbf{y}_h] \tag{3.27}$$

With a cost function there are H historical observations and $\mathbf{y}_h$ is a $1 \times I$ one hot vector. The one hot vector contains a 1 for the observed class and entries of 0 everywhere else. The σ function returns a vector of probabilities. Both cost functions presented here are known to be convex but do not have closed form solutions.

The approach to minimizing these cost functions is done through an iterative technique; gradient descent. Due to convexity the logistic cross entropy cost functions in 3.26 and 3.27 ensure iterative techniques converge to a global minimum.

Gradient descent is a first order iterative optimization algorithm which finds the minimum of a convex function. The approach is for the algorithm to pick an arbitrary starting point then take gradual steps towards the optimal point via the steepest gradient. The general form is given in equation 3.28

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n),\ n \geq 0 \tag{3.28}$$

The $F(x)$ function is the cost function with respect to features (behaviors). The iteration steps are repeated until some predefined threshold of error (difference between x_{n+1} and x_n) is reached. In the gradient descent approach γ_n is often called the learning rate as it governs how far each step can take at each iteration. If the learning rate is too large the technique can oscillate and never converge, if it is too small it may not converge in a reasonable amount of time.

There are variations of gradient descent used in the machine learning literature and practice. These include momentum, stochastic gradient descent, and fast gradient method. Each has its benefits for specific use cases. For this research the classical gradient descent method will be used unless otherwise specified.

Another consideration is how often the model should be updated. One major drawback of using supervised techniques is that they can be computationally expensive to re-train when new information is collected. The cost function can be evaluated every time new information is available, or it can be done periodically based on the preferences of the super-agent. The following sections expand on the concept of supervised learning and cost function by presenting candidate classification algorithms.

3.2.4.3 Logistic and Softmax Regression

Consider that the binary actor type is a discrete variable which can take on either a value of 0 or 1. Logistic (sigmoid) function can be used to train a classification system. The logistic regression approach has the following set of assumptions:

- The dependent variable is either binary or ordinal
- $P(y_i) = 1$, is the desired outcome if there is no error in the signal (positive classification)
- Error terms (differences in observed and predicted values) are independent
- Linearity of independent factors and log odds, the sigmoid function (from eq. 3.31) is derived from the odds
- Stationarity of the observed system
- Large sample sizes

The primary assumption that drives the formulation is log odds. The feature (behaviors) set is considered to be a set of independent factors that are expressed as a linear combination.

$$z = w_0 + w_1x_1 + \ldots + w_Bx_B \tag{3.29}$$

Then the log odds are expressed as 3.30

$$ln\left(\frac{p(x)}{1-p(x)}\right) = z \tag{3.30}$$

The logistic regression takes the form given in equation 3.31 from the manipulation of 3.30. The learning process produces a set of best fit coefficients (w_b) and by convention $x_0 = 1$.

$$\theta(z) = \theta(y|\mathbf{x}) = \frac{1}{1+e^{-z}} \tag{3.31}$$

The function returns a value between 0 and 1 ($\theta(y_i|\mathbf{x})$) which is then interpreted as a probabilistic statement ($\hat{p}(y_i|\mathbf{x})$) about the likelihood that an observation is a given classification. The graphical form of this expression can be seen in Figure 3.8.

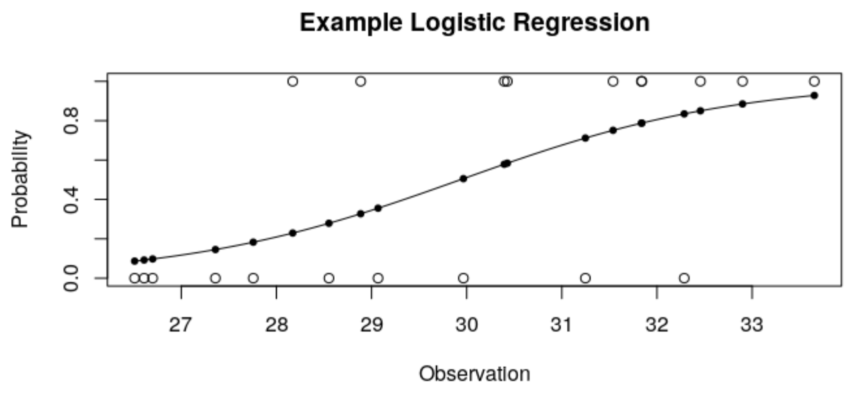


FIGURE 3.8: Flow of Logic for Mathematical Model

The intuition of this approach is that output from the sigmoid function can be interpreted as a probability. For classification a threshold on the function is be selected. This threshold is typically set at the median (0.5). For example, any observation receiving a probability of higher or lower than 0.5 is either positively (1) or negatively (0) classified. However, as formulated in 3.31 it is only useful for the binary classification.

In order to grow beyond the binary case, the sigmoid can be expanded to the softmax function. A softmax function is used to represent a categorical distribution over all possible actors. Instead of a machine learning algorithm directly producing $\hat{y}$ it produces a $I \times 1$ vector of probabilities as a function of $\hat{y}$ as shown in Equation 3.32.

$$\sigma(\mathbf{z}_i) = \frac{e^{z_j}}{\sum_{z=1}^{K} e^{\hat{z}_i}} for i = 1, \ldots, I \tag{3.32}$$

This expression can then be used in the multi-class cost function from equation 3.27. The important difference in this technique is that instead of a single z value with one linear combination there is a $\mathbf{z}$ with I rows for each actor type. This creates a $\mathbf{W}$ weight matrix of size $[B \times I]$. Logistic and softmax regressions represent the most popular methods of classification in machine learning applications. They are not as flexible as other techniques; however, they are well understood and simple to implement.

Both functions also lend themselves to straight forward interpretation for sensitivity analysis and gradient descent. Taking first order conditions of the binary cross entropy cost and applying the chain rule with the sigmoid function gives the logistic regressions expression for sensitivity.

$$\frac{d\theta(z)}{dz} = \frac{1}{(1+e^{-z})} - \frac{1}{(1+e^{-z})^2} = \theta(z)(1-\theta(z)) \tag{3.33}$$

$$\frac{dz}{dx_b} = w_b \tag{3.34}$$

$$\frac{d\theta(z)}{dx_b} = w_b\theta(z)(1-\theta(z)) \tag{3.35}$$

In a similar manner the cost function with respect to values can be expressed as:

$$\frac{d(J)}{dx_b} = w_b(\theta(z) - y_h) \tag{3.36}$$

In kind this process can also be done for the softmax regression.

$$\frac{d\sigma(z_i)}{dz_j} = \sigma(z_i)(1-\sigma(z_i)) \text{if} i = j \tag{3.37}$$

$$\frac{d\sigma(z_i)}{dz_j} = -\sigma(z_i)\sigma(z_j) \text{if} i \neq j \tag{3.38}$$

$$\frac{dz_i}{dx_{b,i}} = w_{b,i} \tag{3.39}$$

$$\frac{d\sigma(z_i)}{dx_{b,i}} = -w_{b,i}\sigma(z_i)\sigma(z_j) \tag{3.40}$$

With these expressions each of the behaviors can be explored for their impact on the sensitivity of the estimated likelihood of actor types.

3.2.4.4 Random Forests

Another classic approach to classification in supervised learning are tree methods. Often referred to as classification and regression trees (CART). These approaches use simple node and branch structures to create surprisingly robust predictors [98]. The method proposed here is the random forest. As the name implies it is a method of creating a set of randomly generated classification trees (forest) and then averaging the results. Random forests are known to be robust against overfitting, perform well compared to other advanced classification models, and are not sensitive to hyper-parameter tuning [99].

Each classification tree is grown assuming some B features (behaviors in the super-agents case) and H data points. A random tree is generated using a subset of the features s_B

and a bootstrap of the historical data s_H. The basic algorithm for random forests, where T is the number of random trees to generate, is as follows:

TABLE 3.7: Random Forest Algorithm

Algorithm: Random Forest

```
for $t = 1, 2, ...T$ do
  Draw $s_H$ bootstrap samples (with replacement)
  Uniformly Select $s_B$ features from $B$
  Grow a Classification tree with parameters $s_B$ and $s_H$
  Record Trees performance and out of bag (OOB) error
end for $t$
Average Results for Classification Prediction
```

This meta technique is simple. A set of sub predictors are generated and then aggregated (arithmetically averaged in this case). Approaches to prediction of this type are referred to as ensemble classifiers as they are a balance of multiple sub-models. In particular, random forests fall into an ensemble method called bagging where sub-models are independent. This is compared to boosting where sub-models are fed each others results.

The key mechanism of the random forest is the generation of the classification tree. Like the forest the tree is also a simple structure with some important characteristics. Each node in the tree represents some split in the data on a given feature that provides useful information for the classifier. The node then has a set of branches (binary) which either terminate (leaf) or give way to another node which will make another split. For example, the website defender discussed earlier may have a decision tree with a root node of time-of-day and a child node of standard or non-standard port. This scenario is depicted in Figure 3.9.

In this example, the after-hours traffic on non-standard ports has a greater likelihood of being a threat. The prediction is made using a simple proportion of cases. In addition, without error checking the majority proportion serves as a nave estimate of the probability given the feature conditions used to arrive at the leaf (prediction) node. As the feature set grows so does the possible size of the tree and its split points. These constitute the two primary choices when building a tree; how big should it be and how should the split point be decided.

The size of the tree is typically based on the quality of information from the error rate at a given node. The basic approach used in this research is that a tree stops growth when no split decreases the trees error fitting of the bootstrapped data. The tree algorithm works from the top down searching for the best split at each node given the current subset of data (data is subset at each node based on the split condition). As the tree

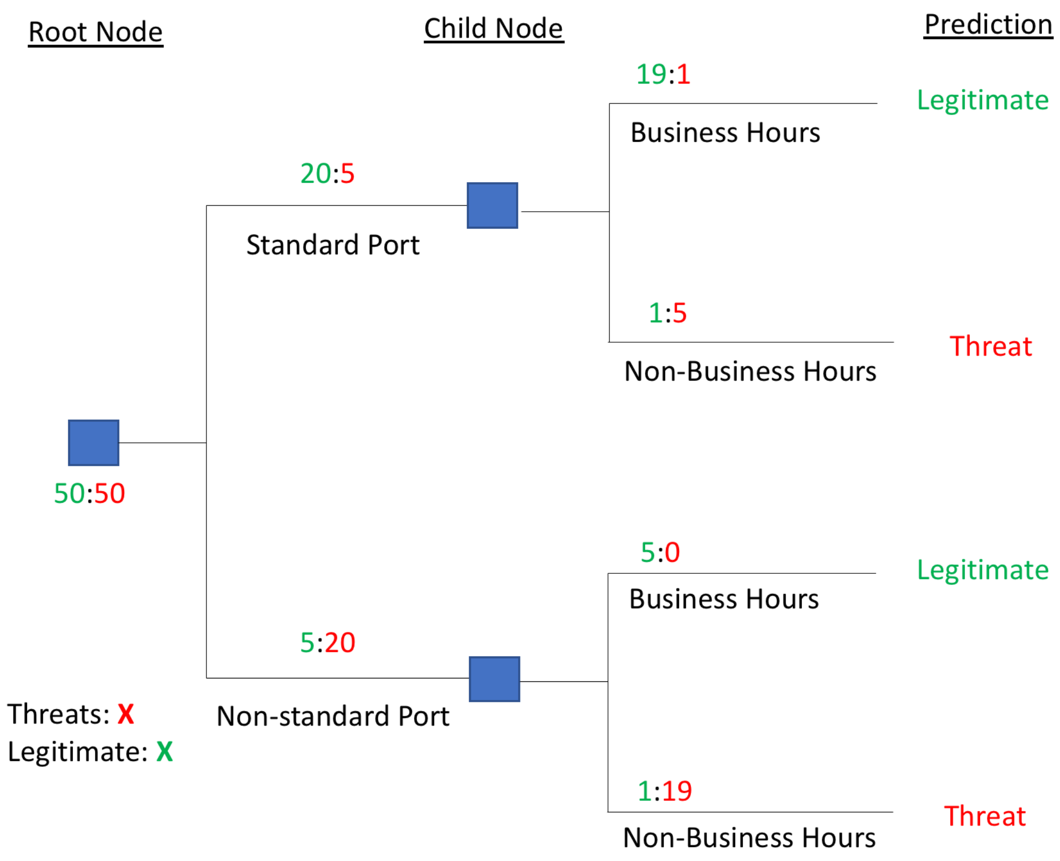


FIGURE 3.9: Example Classification Tree

grows, naturally, the data thins thereby providing finer detail of the classifier with more of the feature set contributing against fewer observations. When the tree is done it represents a series of simple feature conditions that can be used to make a prediction. Consider the following algorithm where ϵ is the difference in error at each node and f is the number of features

TABLE 3.8: Decision Tree Algorithm

Algorithm: Classification Tree

```
do while $\epsilon > 0$
  Calculate split error for all unused features $B$
  Select the feature with the lowest split
  Calculate $\epsilon$ as diff in error of tree without split and with split
```

Unlike logistic regression the random forest algorithm does not lend itself to an analytical sensitivity analysis. Finding an optimal policy with this technique requires searching across the gate set using iterative methods. However, due to the nature of the forest the most consequential behaviors are represented, on average, at the beginning (top) of the tree. Average tree structures then become sources of information for the most important behaviors. However, random forests can be explored based on the average impact on cost when specific features are included (or not included). This approach is often referred to as important feature analysis and will be demonstrated with an application in Chapter 4.

3.2.4.5 *k* Nearest Neighbors

The *k* Nearest Neighbor (kNN) classification algorithm is among the most straight forward. The basic concept is that an observation is most likely to be classified the same as others that have similar feature characteristics. Unlike logistic regression but similar to random forests kNN is a non-parametric technique. The method is considered to be instanced-based or lazy learning because the computation is done upon classification as opposed to model generation. As the name applies the algorithm has one significant parameter k, the number of nearest points on which to base a classification estimate. Figure 3.10 is an example depiction of a 5NN model.

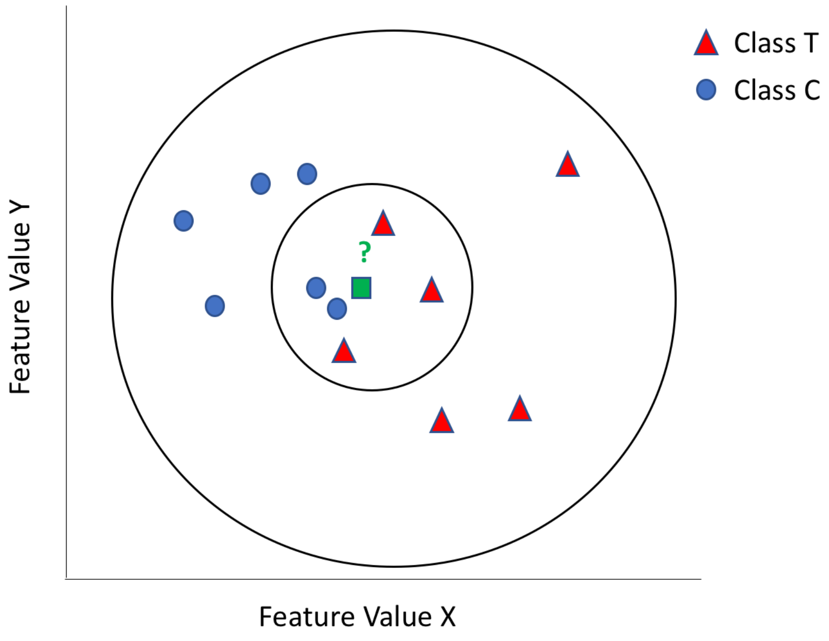


FIGURE 3.10: Example kNN Model where $k = 5$

Figure 3.10 shows that the closest five neighbors are considered for classification of the green square. In this example case the classification prediction would be T. However, if k was reduced to 1 the classification would change to C. This approach assumes homophily of some degree where like classes will congregate with other similar featured class type.

Finding the amount of similarity is part of the model tuning. The class prediction is done by a simple majority of the neighboring classes if there is no weighting. With weighting some distance-based discounting can be applied as w_i with the following conditions:

$$\sum_{i=1}^{k} w_i = 1 \forall w \geq 0 \tag{3.41}$$

The challenge in the technique is finding the neighbors. For each observation to be classified a pairwise distance from all other points must be considered. There are several ways to measure the distance between two feature vectors, this research will use Euclidian. The Euclidian distance is given in equation 3.42.

$$||\mathbf{x_{b,1}} - \mathbf{x_{b,2}}|| = \sqrt{(\mathbf{x_{b,1}} - \mathbf{x_{b,2}})(\mathbf{x_{b,1}} - \mathbf{x_{b,2}})} \tag{3.42}$$

This technique is sensitive to the number of features used in the learning process. Most application of kNN with a significant feature size (greater than 10) use some type of dimensionality reduction like principal component analysis or linear discriminate analysis. Similar to the random forest, because there is no general model for kNN the behavior space needs to be exhaustively searched for classification sensitivity useful for generating a gate policy.

3.2.4.6 Neural Networks

Artificial Neural Networks (ANN) have been rebranded in recent years to deep learning models. The motivation behind this change is that computation advances (specifically in parallel microchip design in graphical processing units or GPUs) have made training ANNs with many hidden, and often nuanced, hidden layers feasible. This research will take advantage of modern deep learning architecture and the related computational advances.

The concept of a deep learning model is similar to the boosting technique briefly discussed in the random forest section. A model is not a straight forward calculation against inputs and weights like logistic or OLS regression. Instead inputs are passed through a series of functions that are fed into each other until reaching an output layer that conducts the actual classification task. The method of nesting regressions on top of each other provides the deep learning algorithm an large amount of flexibility. Figure 3.11 is an example of a network architecture for a network with two hidden layers.

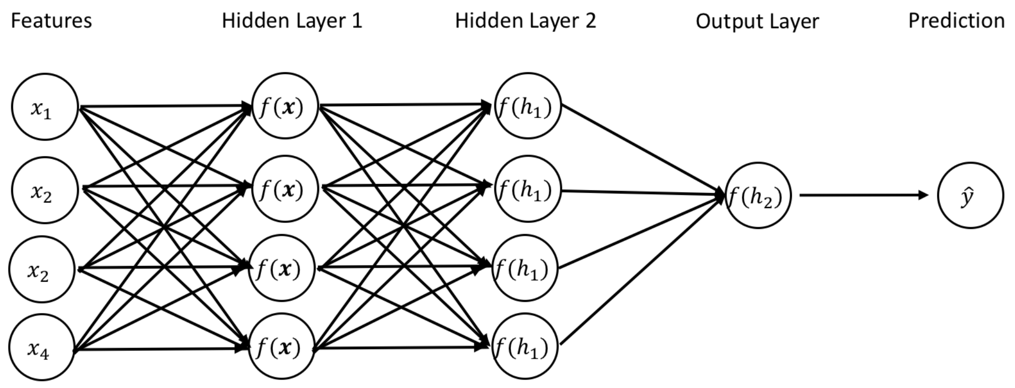


FIGURE 3.11: Example of Deep Learning Network Architecture

In this architecture the first layer features (behaviors) are inputs to a set of functions $f(h_1)$. Each feature input is represented by a linear combination with weights in the same way as logistic and softmax regression. For example, the input to the first node (top node) on the first hidden layer is given by.

$$z_1^2 = w_{0,1}^1 x_0^1 + w_{1,1}^1 x^1 + w_{2,1}^1 x_2^1 + w_{3,1}^1 x_3^1 + w_{4,1}^1 x_4^1 \tag{3.43}$$

In this expression the subscripts determine the input and layer notation $z_{\text{node}}^{\text{layer}}$. The weights have an extra index because they are independent between each network layer and is referenced as $w_{\text{node,connected node}}^{\text{layer}}$. This means that each layer has an input of vector the same length as $\mathbf{x}$ $(1 \times n_x)$multiplied by a weight matrix $\mathbf{W}$ of size n_x, n_{x+1} which results in an input vector to the next layer $\mathbf{z}$ of size $1 \times n_{x+1}$.

The vector of inputs to a layer are of the same length as the layer itself. At each node in the layer the input is passed through some pre-determined activation function $(f(x))$. There are many choices for activation functions. For this research activation functions will be sigmoid function from equation 3.31. This choice has the convenience of straight forward differentiation as seen when constructing sensitivity for logistic regression and cost functions and convexity.

As values pass through the network they are eventually connected to a final output layer which process a probabilistic classification estimate $\hat{y}_t$. This last layer is the same logistic or softmax formulation from equations 3.31 and 3.32 for binary or multi-class classification. The great benefit of deep learning algorithms is that they are able to construct highly non-linear relationships between features. Training complex network architecture can be both time and resource consuming. However, once they are trained, implementing them in practice is very efficient. Often deployment consists of a simple matrix multiplication exercise.

For sensitivity of a deep learning model the same approach used in logistic and softmax regression can be leveraged by extending equation 3.33 and 3.37 through other steps of application of the chain rule. For example, the sensitivity of a single layer in a deep learning algorithm used for binary classification would be given by 3.44

$$\frac{d\theta(z^3)}{dz^3} = \frac{1}{(1+e^{-z^3})} - \frac{1}{(1+e^{-z^3})^2} = \theta(z^3)(1-\theta(z^3)) \tag{3.44}$$

$$\frac{dz^3}{d\mathbf{x^2}} = \mathbf{w_3} \tag{3.45}$$

$$\frac{d\theta(z^3)}{d\mathbf{x^2}} = \mathbf{w_3}\theta(z^3)(1-\theta(z^3)) \tag{3.46}$$

This process continues depending on how deep the network is, combining the results to create a closed form analytical sensitivity expression. The use of deep learning models has become very popular in recent years, as discussed in Chapter 2. Various architectures are compared for performance in the application presented in Chapter 4. However, all of the architectures are built on the same or similar principles presented here.

The major drawback for the super-agent of using this technique is that the algorithms can be time intensive to train and then re-train as more data is collected. If the defended system is dynamic, then heavily relying on deep learning models may not be practical. In addition, there is a growing body of work on general adversarial networks or GANs. The basic application of a GAN is to leverage the non-linear nature of the deep learning model in order to create synthetic behaviors that fool the classifier. If a security deep learning model becomes known, training a GAN is a simple and straight forward way to circumvent it. Work has been done in this area for malware detection [100] [101]. Any deep learning model developed for access control would have similar challenges to address.

3.2.5 Learning Methods Summary

A connection can be made between the similarities in the formulation of classic warning systems and machine learning. The learning method will result in an imperfect prediction system that can be used to directly inform the warning system parameters discussed in Chapter 2. When the model is trained, it will have false positive and false negative error rates as measured against a test set of data. This information is useful as an input to a warning system and decision threshold.

How useful a technique is depending on the specific application of interest to the super-agent. The two categories of techniques, known and unknown gate sets, have different approaches and assumptions. If the gate set is known the super-agent wants to learn the distribution of actors and utilities as fast as possible. However, if the gate set is unknown the super-agent also wants to find the gate set as fast as possible. Because of these differences the pros and cons of each type of technique needs to be addressed separately

For circumstances where the super-agent has a known gate set Table 3.9 summarizes the benefits and drawback of each technique.

Table 3.9: Pros and Cons of Known Gate Set Learning Techniques

Technique	Small Data	Easy to Update	Estimate Actor Type	Estimate Utility	Well Known
MLE	No	Yes	Yes	Yes	Yes
Bayesian Updating	Yes	Yes	Yes	No	Yes
Metalog Updating	Yes	Yes	Yes	Yes	No

Table 3.9 shows a set of important considerations for each technique. The first concern is whether technique can be used reliably with a small data set. MLE requires as many as thirty data points before a confident estimate is generated. Whereas Bayesian and Metalog updating can provide information, through the use of priors, with a single data point. For updating, each technique is suitable to easily integrate new information. All of the techniques are useful for estimating an actor type probability distribution. However, for estimating utility, the classic gaussian Bayesian technique requires a known variance, which makes this approach less robust. And finally, both MLE and the Bayesian techniques are well known and the Metalog is a novel concept introduced through this research.

The considerations for an unknown gate set are different, in that these techniques are only concerned with actor classification and will rely on the known gate set methods to supply utility estimates. Table 3.10 gives a detailed list of concerns for each technique.

Table 3.10: Pros and Cons of Unknown Gate Set Learning Techniques

Technique	Easy to Train	Easy to Update	Easy Analysis	Sensitivity
Logistic/Softmax	Yes	Yes	Yes	
Random Forest	Yes	No	No	
K Nearest Neighbor	NA	No	No	
Neural Networks	No	Yes	Yes	

With both the Logistic/Softmax and Random Forest techniques, training is a simple and fast procedure that can be done efficiently even with large amounts of data. However, kNN does not require training and the Neural Networks can be intensive when building the first algorithm. When updating with new information both Logistic/Softmax and Neural Networks can start learning from the previous weighting values making updating efficient. However, with both Random Forests and kNN the models must be rebuilt completely when new information is introduced. In addition, Logistic/Softmax and Neural Networks have closed form expressions for sensitivity whereas the other method requires manual exploration.

The performance of each technique will be explored in Chapter 4 in greater detail using a real-world use case. Now the concern turns to the practical question of what to do with the information produced from the machine learning techniques. The following section explores the decision-making framework that leverages the estimates generated by the various learning processes.

3.2.6 Decision Property Formulation

The super-agent learns the probabilities over time so that it can make the best gate-set policy decision. Figure 3.12 graphically represents the active learning paradigm for the robot.

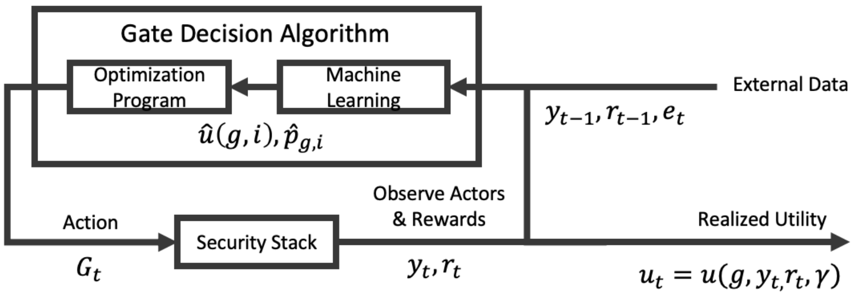


FIGURE 3.12: Flow of Logic for Mathematical Model

The previous sections covered how the robot learns. Each of the learning techniques have strengths and draw backs, however, they perform the same basic function. They all produce probabilistic estimates for actor type and utility. With the estimates the robot can create an optimal policy with its current state of information, at that point in time. In addition, decisions which exceed thresholds can be passed off to the human. The resulting policy is then evaluated over a single period where the security stack collects more information which then results in a new set of estimates.

The primary consideration of the robot in building a policy is which decisions should be passed to the human. This research proposes the use of thresholds as an initial mechanism to enable this determination. The following sections discusses the use and generation of specific thresholds.

3.2.6.1 Human Intervention Thresholds

In Chapter 2 the concept of up crossing thresholds was discussed in the context of early warning systems. A similar approach is taken here. The threshold for the super-agent is a point, beyond which, the robot cannot be trusted to act. The reason for the lack of trust is that certain decisions fall into one of two categories. First, the robot is not confident in its recommendation. Second, is that the decision caries a substantial value impact for the organization. These two categories are not mutually exclusive. For example, if decisions are low impact and low confidence then the robot can be given latitude to implement its recommendation and vice versa.

- $\lambda_p = Var[\hat{p}_{g,i}]$ variance of the parameter estimate for the actor distribution
- $\lambda_u = Var[E[\hat{u}(g,i)]]$ variance of the expectation estimate for utility distribution
- $\lambda_a = \hat{p}_{g,i}$ of the prior of actor distribution
- \$$\lambda_q = F^{-1}(p)$ quantile value from some p of the prior of the gate utility distribution where F^{-1} is the inverse CDF of $u(g,i)$
- \$$\lambda_e = E[\hat{u}(g,i)]$ expected value of the prior of the gate utility distribution

Human analysts have contextual understanding of the system and external demands that the machine will not understand. For this reason, a low confidence decision by the machine may be much easier for the human to understand. In addition, a high impact decision is not customarily understood by the robot. This lack of understanding also implies some lack of accountability. Appropriate thresholds ensure that an organization is conscious of consequential actions. Figure 3.13 is a graphical depiction of threshold limit for an organization based on the combination of confidence and impact.

As Figure 3.13 implies a threshold Φ is represented as a linear combination of confidence and impact. This research proposes that confidence is represented a discrete probability (Expectation of $\hat{p}$) and impact is a point on the distribution of utility (quantile point $u(g,i)$). Utility is evaluated as a function of risk attitude as explored by Howard [102]. Where utility is the result of interpreting monetary values through a u-curve which is a function parameterized by the decision makers risk preference. Equation 3.47 is the functional form used in this research.

$$u(g,i,\gamma,r) = 1 - e^{-\gamma r} \tag{3.47}$$

For the utility function we assume constant risk aversion and γ represents the risk aversion coefficient with the following characteristics.

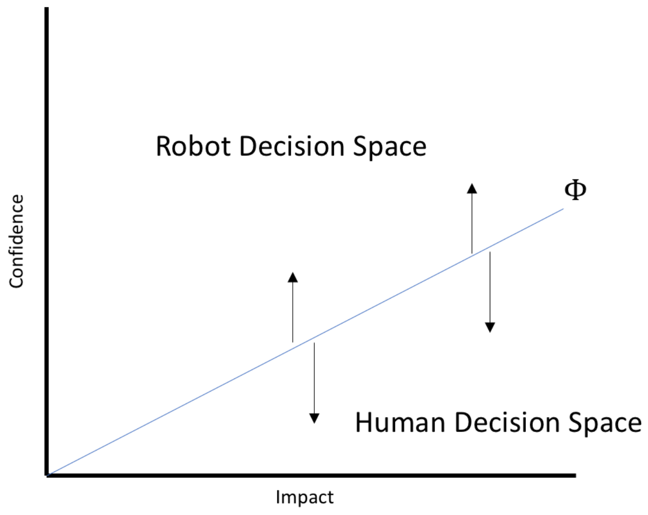


Figure 3.13: Example Threshold Limit

- $\gamma < 0$: Risk Preferring
- $\gamma = 0$: Risk Neutral
- $\gamma > 0$: Risk Averse

Both confidence and utility distributions are outputs of the learning models discussed in previous sections. However, the choice of the point used to determine impact for the distribution is subjective. One straight forward way is to assess impact is just use the expected value of utility, however, this ignores variation. Instead, an alternative is to choose a point on the lower tail. There is a similar concept in finance called value at risk or VaR. Which, typically, uses the lower 95% ($p = 0.95$) return to express the variation and impact. However, VaR is quite controversial as it ignores tail risk and asymmetries. The benefit of the super-agent concept is that the threshold can be set based on the natural constraints of the human.

Setting the thresholds must be part of the organizations preference. Obviously, the human is the limiting factor. The fewer decision the human executes, the more time they can spend researching the best course of action. However, if the human is mostly idle, they may regress in understanding of the system deferring too much responsibility to the robot. Setting a decision threshold so that the humans time is optimally used is on alternative.

The cost of the robot and the humans time is well known by the organization and gives a simple trade-off decision for the organization. The number of humans in the super-agent can be scaled by risk preference. As the robots performance (confidence) improves, over time, this may lead to a lower demand on the humans time. In order for the robot to learn it must be fed with a consistent stream of new information. The following section discusses methods of trading between collecting information to learn and protecting the system.

3.2.6.2 Explore and Exploit

The decision cycle in Figure 3.12 has some fundamental characteristics. The first is that initial decisions may not have any historical or external data from which to develop reliable estimates. This implies that an important performance characteristic of an algorithm is how fast it can learn from sparse or small amounts of data. There is an opposite problem in that there is may be an enormous amount of historical data. Which requires algorithms that can take full advantage of the information in the entire corpus.

A trade-off in the decision cycle is between exploring and exploiting. The more data that is collected the better all of the learning models will perform. However, collecting data means exposing the defended network to risks. The concept of exploiting is following decision guidance from all known information (for example, choosing the gate policy based on the current maximum expected utility). However, early data collection might be biased or misleading due to pure chance. This means that in order to find the globally optimal policy the robot must take risks in order to explore other alternatives.

The concept of explore and exploit are built into the active learning (known and unknown gate set) models discussed in the earlier sections. For example, Thompson sampling using the Beta/Binomial method makes determinations based on realizations from estimated distributions. However, for the unknown gate set models it is not as clear how to balance this trade-off. For example, using neural networks produces a probabilistic estimate across actor types. This can be used for maximum likelihood estimates. This must then be combined with some type of sampling technique in order to find the optimal policy. Some work has been done to express the weighting coefficients in supervised techniques as probability distributions for this reason [103].

There are three techniques that can be used to conduct the explore/exploit trade-off; epsilon-greedy, upper confidence bound, and Thompson sampling. Each of these methods have their own formulation and application. Epsilon-greedy is a method of consistently exploring with some probability ϵ or doing choosing the best option with existing information (exploiting). Given mean estimates $(\hat{\mu_{1,b}}, \ldots, \hat{\mu_{I,b}})$ a gate is opened at random with rate ϵ or closed with rate defined by Equation 3.48.

$$p_b(closed) = 1 - \epsilon \tag{3.48}$$

If ϵ is constant there is a proportional upper bound of performance. However, there are method to anneal ϵ with time that can improve performance. The second method of Upper Confidence Bound or UCB picks alternatives based on a pre-determined confidence bound. Figure 3.14 shows an example of bounds in a UCB setting.

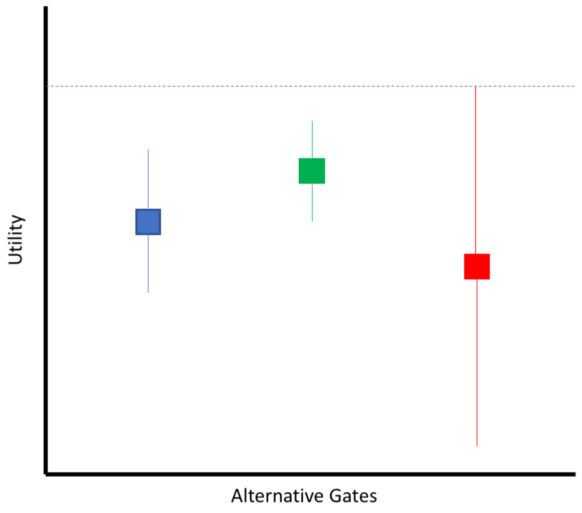


FIGURE 3.14: Example Upper Confidence Bound

With UCB the method of exploring is to greedily select the alternatives with the most preferable bound. The bounds for alternatives are defined by the UCB1 convention shown in equation 3.49. With UCB the t_x is the number of times an alternative has been tried (gate opened) and $2\ln(t)$ is a factor that ensure alternatives are tried on some periodic basis.

$$\text{argmax}(\mathbf{x}) = E[\hat{(}\mu_x)] + \gamma\sqrt{\frac{2\ln(t)}{t_x}} \tag{3.49}$$

The final method used for exploration, Thompson sampling, is a technique that takes advantage of Bayesian updating. Instead of exploring at a set rate or with a bound it is done by sampling from a prior distribution. Consider the Bayesian updating method to estimate the multinomial actor distribution discussed in the previous section. At each time period Thompson sampling will sample from all gate prior distributions and then use those realizations to build a gate policy. Over time the priors will update to posteriors narrowing the variance of the parameter estimates. In many settings Thompson sampling is known to be superior to both epsilon-greedy and UCB [104]. Using the explore and exploit trade-off, at each time period the super-agent must decide on a gate policy. The following section details the decision-making process.

3.2.7 Decision Making Methodology for Risk Management

The method proposed here is a Markov Decision Process or MDP. This technique is advantageous in decision scenarios where part of the system is random, but actions are under the control of an agent. With each time step and set of actions the agent learns about the system. There are several well-known methods to find an optimal set of actions (policy) using recursive exploration. With estimates from the learning models there are several approaches to determine an optimal policy (with a given state of knowledge) depending on the number of steps in a kill chain. For multi-step kill chains the following state action transition matrix describes the conditions from which the super-agent needs to make decisions.

Table 3.11: Notation for Markov Decision Process

Value	Description
β_i	Probability of not being targeted by a given actor
δ_i	Probability of being targeted by a given actor
γ_i	Probability of a given actor advancing one step
$\lambda_{n,i}$	Probability of an actor staying in the same kill chain step
τ	Probability of an actor being removed from the system

The transition matrix for a kill chain would take the following form where $\delta_i = 1 - \beta_i$ $\gamma_{n,i} = 1 - \lambda_{n,i}$. For a given action $a = G_t$ then the transitions are defined as:

$$P_{n+1,i,a} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ \vdots \\ N \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & \dots & N \end{array} \\ \begin{pmatrix} \beta & \delta & 0 & 0 & \dots & 0 \\ \tau & \lambda_{1,i} & \gamma_{1,i} & 0 & \dots & 0 \\ \tau & 0 & \lambda_{2,i} & \gamma_{2,i} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \tau & 0 & 0 & 0 & \dots & \lambda_{N,i} \end{pmatrix} \end{array} \tag{3.50}$$

The following sections describe the decision formulation for two cases: one step and multistep kill chains.

3.2.7.1 Single Step Kill Chain

A single step kill chain view can be convenient for both practical and computational reasons. Often stages in a kill change can be consolidated and viewed as combinations in their entirety. In addition, it may be the case that the super-agent only has security responsibility for a single step in the chain. For example, boundary protection is often separated, in practice, from onsite physical security teams.

The optimization program in this model can take several forms depending on the setting. The purpose of the optimization is to return a vector of open and closed gates represented as a binary column vector. This vector represents the policy of the super-agent:

$$G_t = \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ m \end{bmatrix} \tag{3.51}$$

The optimization program takes the general formulation.

$$\begin{aligned} \underset{G_t}{\text{maximize}} \quad & \mathbf{G_t^T U_t} \\ \text{subject to} \quad & G_{g,t} = 1 \forall g \in S \\ & \sum G_{g,t} = x \\ & G_{g,t} \in \{0,1\} \end{aligned} \tag{3.52}$$

The goal of the model is to find G_t^*. The difficulty in finding the optimal gate policy is that there is a tremendous combination of possibilities and the actor and utility functions are stochastic. However, because the distribution of actors at each gate is multinomial distributed one alternative is to use an online active learning algorithm with a Dirichlet-Multinomial updating scheme that is an extension of the Beta/Binomial bandit problem. The Dirichlet distribution is conjugate with the Multinomial and has the following update scheme.

- Parameters: $(\alpha_1, \ldots, \alpha_M)$
- PDF: $\alpha p_1^{\alpha_1 - 1} p_2^{\alpha_2 - 1} \ldots p_M^{\alpha_M - 1}$
- Update: $\alpha_M \leftarrow \alpha_M + 1$ if outcome M observed

$$\text{Prior } P_{g,i,t} \sim \text{Dirichlet } \big(\alpha_{g,i,t}(1), \alpha_{g,i,t}(2), \ldots, \alpha_{g,i,t}(|(I+1)|)\big)$$

The algorithm 3.12 can be leveraged to learn the reward and threat probabilities. This algorithm is adapted from the work done by Russo et al. [104].

This method will allow for efficient learning over time with sampled observations. The condition of general cyber threat activity (e_t) at a given point is fluid based on external

Table 3.12: Single Stage Kill Chain Algorithm

Algorithm: Active Learning Single Stage Cybersecurity

```
for $t = 1, 2, ....$ do
  for $g = 1, 2, ..., m$ do
    Sample $\theta_g \leftarrow dirchlet(\alpha_{g,i}) \forall i$
  end for $g$
  $G_t \leftarrow$ Optimization Program$(\theta_g)$
  Submit $G_{g,t} > \phi_g$ and $R_{g,t} > \phi_r$ to human for evaluation
  observe $R_t$
  Update: $\alpha_{g,i} \leftarrow \alpha_{i,g} + 1$ if outcome $T$ observed
  Update: $\alpha_{g,i} \leftarrow \alpha_{i,g} + 1$ if outcome $e_t$ observed
end for $t$
```

factors. Impacts from known and unknown software vulnerabilities, organizational positioning and hacker capabilities all contribute to more significant trends. These trends can be applied to local security decisions by learning from and incorporating external data into uncertainties. These values, if available, are observed at the time of security decisions.

3.2.7.2 Multi Step Kill Chain

In the multistep paradigm, the same methodology of Dirichlet-Multinomial can be leveraged. However, instead of using a simple optimization scheme, each period requires solving a Markov Decision Process. The notation in table 3.11 in included for this formulation.

In a similar fashion rewards are defined across the kill chain steps according to an action policy as defined earlier. The algorithm 3.13 for learning the optimal gate policy becomes:

Table 3.13: Multi-Stage Kill Chain Algorithm

Algorithm: Active Learning MDP Cybersecurity

```
for $t = 1, 2, ....$ do
  for $g = 1, 2, ..., m$ do
    Sample $\theta_g \leftarrow dirchlet(\alpha_{g,i}) \forall i$
  end for $g$
  $G_t \leftarrow$ Solve MDP$(\theta_g)$, observe $R_t$
  Submit $G_{g,t} > \phi_g$ and $R_{g,t} > \phi_r$ to human for evaluation
  Update: $\alpha_{g,i} \leftarrow \alpha_{i,g} + 1$ if outcome $T$ observed
  Update: $\alpha_{g,i} \leftarrow \alpha_{i,g} + 1$ if outcome $e_t$ observed
end for $t$
```

One consideration of this approach is how often the model is updated. The reward function can be evaluated every time new information is available, or it can be done periodically based on the preferences of system owners. The first few steps of an attack that a defender takes are used to classify threats.

In the next chapter this model will be explored using two applications. Both the known and unknown gate set approaches are used. The known gate set is a constructed example and the unknown setting is done using a real world dataset.

Chapter 4

Application

The model presented in Chapter 3 has a few structural considerations. While the kill chain is discretized into seven steps, at no point during these steps is a threat actor assumed to be positively identified. Indeed, many threats flow through the steps with the sole intent of never being recognized. The most significant consideration is that threat classification is done independently of the kill chain. Steps in the chain are attributed to a threat after forensic analysis is done. The following sections will discuss two applications of the general model with independent validation of threats. The exact implementation of these models can be found in Appendix H.

4.1 Known Gate Set Example

This section has a constructed use case to explore the properties and considerations of the known gate set model. Consider a cyber defender with a behavior set consisting of two source ports $(s1, s2)$, two destination ports $(d1, d2)$, two destination IP addresses $(i1, i2)$, and two times of day (work hours and non-work hours). Figure 4.1 is a graphical depiction of these behaviors.

In this case, the defender has 16 combinations of behavior attributes which constitute the gate set. Table 4.1 is the actual (but unknown to the super-agent) distribution of actors and utility at each of the gates. For simplicity, actors are drawn from the binary setting, being only threat or legitimate.

With perfect information of the distributions at each gate, the optimal policy is to keep eight of the gates open and the other eight closed with a total expected policy value of 5.2. As discussed in Chapter 3 these distributions are unknown to the super-agent.

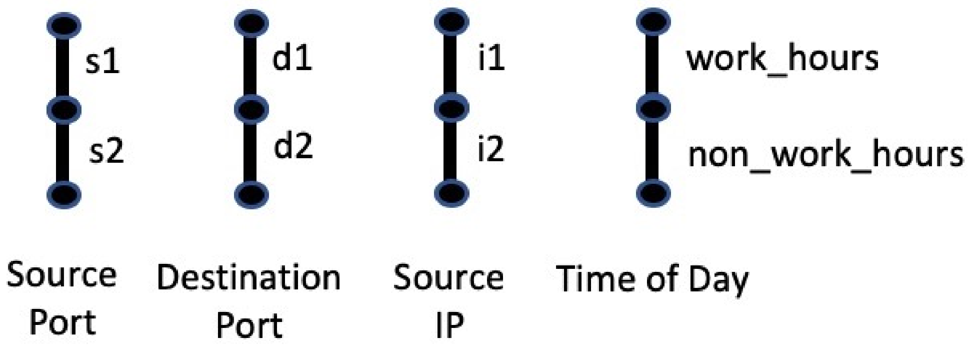


FIGURE 4.1: Simple Use Case Behavior Set

TABLE 4.1: Simple Use Case Data

gate	threat	non-threat	threat utility	non-threat utility	$E[u(g)]$
s1, d1, i1, w	0.1	0.9	~N(-2,1)	~N(2,1)	8
s1, d1, i1, nw	0.3	0.7	~N(-2,1)	~N(2,1)	4
s1, d1, i2, w	0.6	0.4	~N(-2,1)	~N(2,1)	-2
s1, d1, i2, nw	0.6	0.4	~N(-2,1)	~N(2,1)	-2
s1, d2, i1, w	0.1	0.9	~N(-2,1)	~N(2,1)	8
s1, d2, i1, nw	0.2	0.8	~N(-2,1)	~N(2,1)	6
s1, d2, i2, w	0.5	0.5	~N(-2,1)	~N(2,1)	0
s1, d2, i2, nw	0.7	0.3	~N(-2,1)	~N(2,1)	-4
s2, d1, i1, w	0.1	0.9	~N(-2,1)	~N(2,1)	8
s2, d1, i1, nw	0.3	0.7	~N(-2,1)	~N(2,1)	4
s2, d1, i2, w	0.6	0.4	~N(-2,1)	~N(2,1)	-2
s2, d1, i2, nw	0.6	0.4	~N(-2,1)	~N(2,1)	-2
s2, d2, i1, w	0.1	0.9	~N(-2,1)	~N(2,1)	8
s2, d2, i1, nw	0.2	0.8	~N(-2,1)	~N(2,1)	6
s2, d2, i2, w	0.5	0.5	~N(-2,1)	~N(2,1)	0
s2, d2, i2, nw	0.7	0.3	~N(-2,1)	~N(2,1)	-4

Each of these distributions are multi-modal. Sorting the gates by expectation, Figure 4.2 depicts the shape.

In order to learn the distributions, the super-agent must engage in an explore-exploit methodology. The goal is to find the optimal gate policy but also to explore the negative expectation gates as little as possible. Further, this scenario assumes the following additional conditions:

- super-agent is risk neutral ($\gamma = 0$)
- threshold for human decision by confidence (λ_u or $Var[E[\hat{u}(g)]]$) on actor type where the standard deviation is greater 0.05

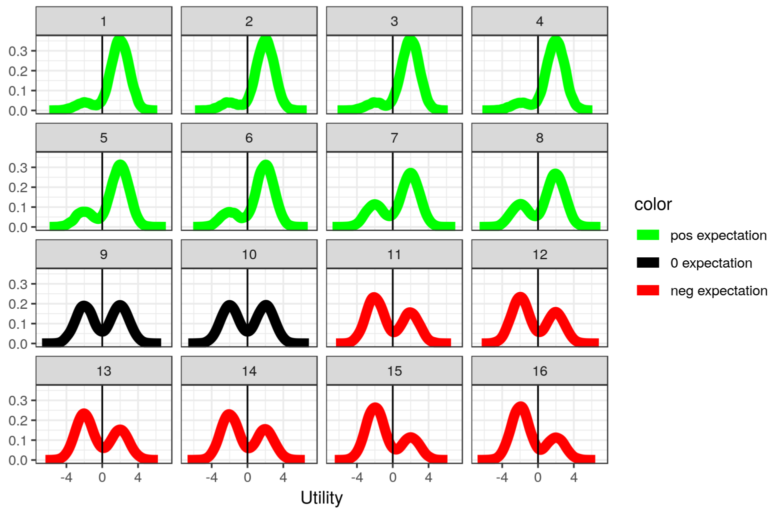


FIGURE 4.2: Distributions of Gates

For convenience, this use case assumes that the expert-human makes the same decision as the robot. However, in practice, this may not always be true.

4.1.1 Gate Set Decisions and Observations

In this simple example, it is trivial to enumerate the gate set (as is done in Table 4.1). Also, an assumption is made that if a gate is open during one time period (iteration), then one observation is made. An observation at a gate consists of collecting data on the actor type and realized utility. Using the three methods proposed in Chapter 3, Figure 4.3 shows the average performance of each technique.

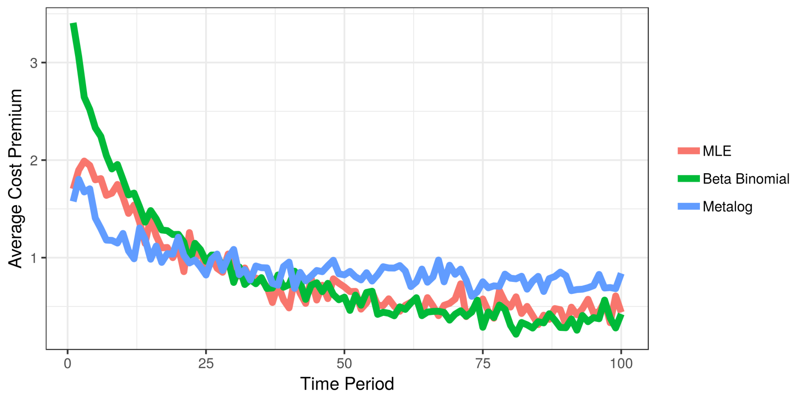


FIGURE 4.3: Simple Use Case Behavior Set

Each method is judged based on the average regret. Regret is defined as the difference between average realized utility (over thousands of simulations) and the expected utility of the optimal policy. The goal for an algorithm is to reach a minimal regret bound as quickly as possible. As Figure 4.3 shows, the metalog utility estimation outperforms the other methods in terms of speed of learning. However, as the lower bound of regret is reached it is surpassed by beta-binomial and MLE in the long run.

The reason is that the Bayesian annealing in the metalog updating narrows much faster than does the beta-binomial. This means that the metalog learning technique will decide on a policy after a short exploration phase and then will mostly keep the same decision. On the other hand, the beta-binomial will continue to explore much longer. Figure 4.4 shows an example of each technique for a single simulation iteration and a horizon of one hundred time periods.

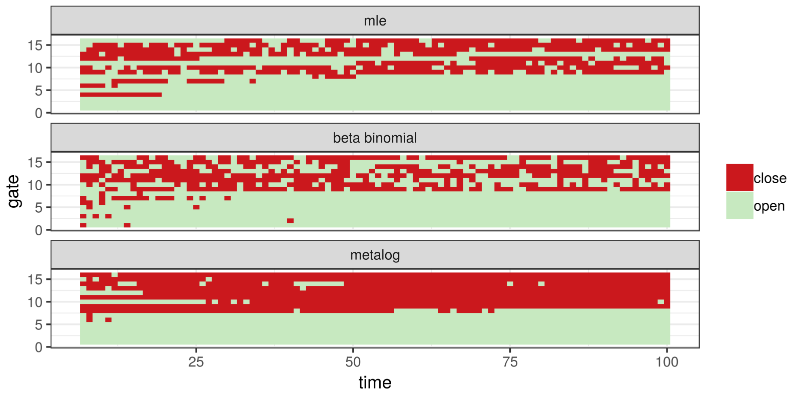


Figure 4.4: Sample Gate Positions for a Single Simulation

Each method has a different characteristic. The metalog does a short exploration phase then mostly stays with the same decision. Whereas both MLE and beta-binomial tend towards a lot more exploring. For a practical application, the quick confidence of the metalog technique might be preferred.

4.1.2 Risk Tolerance and Decision Trade-off

Using the utility function approach to machine learning allows for exploring how the humans risk tolerance and attitude impact the decision space. For example Figure 4.5 shows how the beta-binomial gate decisions change based on differing values of gamma (the parameter in equation 3.23).

The insight from this visualization is that the robot opens more gates as the human is more risk seeking and closes them with risk aversion. This single adjustment on the machines understanding of the data creates a significant impact on the decision outcomes. One way to think about this is that more risk averse entities, like governments, would tend to close most of their gates. On the other hand, those who are more risk

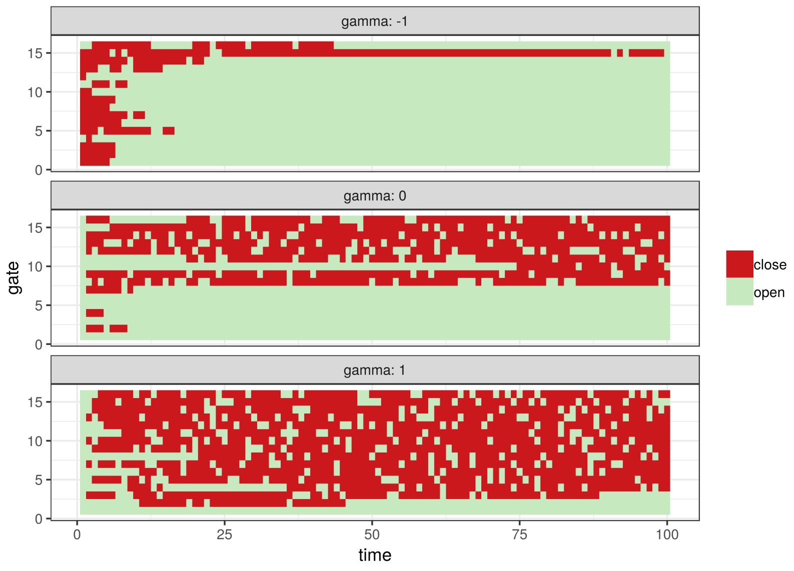


FIGURE 4.5: Simple Use Case Behavior Set

seeking would tend to open them. This single factor explains, quantitively, the observed behavior in many organizations.

Another benefit of this model is the collection of the Value of Information (VoI) for each gate. Often cybersecurity defenders have limited forensic resources, so it is important to understand where the best place to apply efforts is to reduce uncertainty. Figure 4.6 shows the VoI of a subsection of the gates.

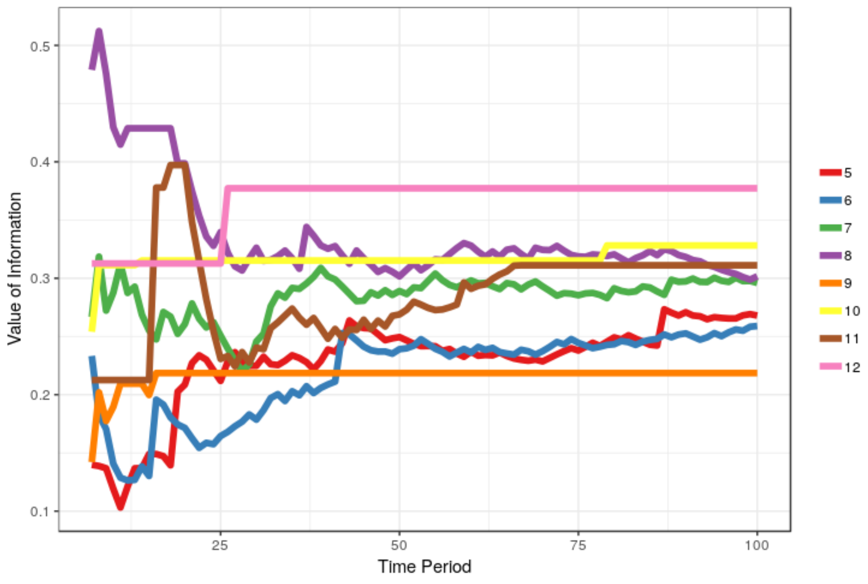


FIGURE 4.6: Value of Information Per Gate

As the robot learns, the VoI for each gate is updated based on its current state of information. At some point these values stabilize either because the door has been closed or because enough information is collected to have an accurate estimate. Through the VoI at each point in time, the super-agent has a method which provides the best place is to dedicate available resources. This is not currently a feature of any existing machine learning model.

In addition to the contributions of threat risk and VoI there is also information about uncertainty, both epistemic and aleatory. As the robot learns the utility distributions at each gate, the epistemic uncertainty is reduced and the aleatory uncertainty is better estimated. Figure 4.7 shows an example of these values changing as the robot learns. In this case epistemic uncertainty is derived from the variance of the beta-distribution prior. The aleatory uncertainty is measured by the variance of the distribution of utility (using a lower bound quantile).

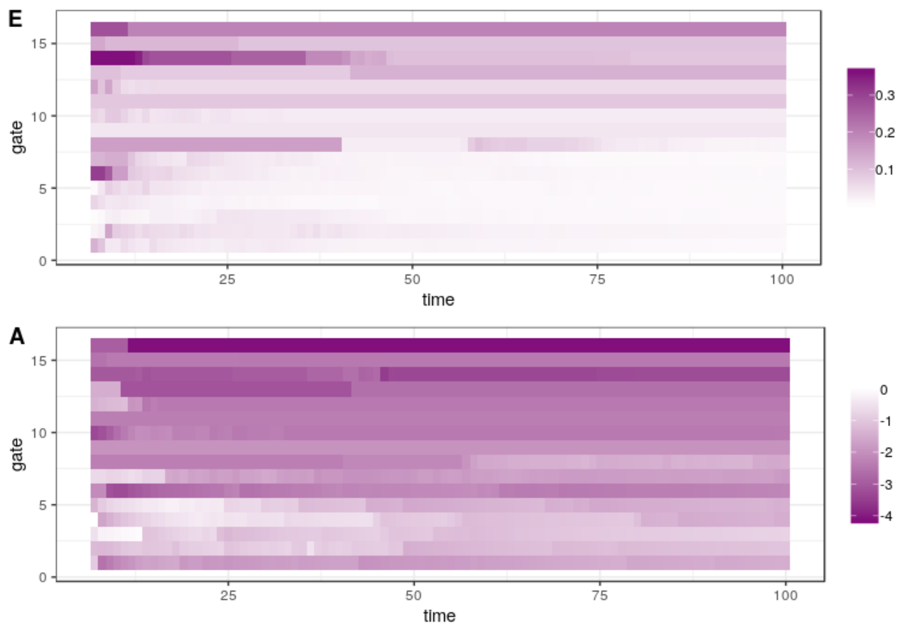


FIGURE 4.7: Epistemic and Aleatory Uncertainty Changes over Time

The epistemic uncertainty is reduced until the robot decides to keep a gate closed, at which point no more information is needed to make the decision. However, gates which are kept open, almost completely reduce epistemic uncertainty to zero. With Aleatory uncertainty, after some initial variation, during the explore phase the values stabilize and become consistent.

Using the information of risk and uncertainty, the decision threshold can be explored between the robot and the human. Figure 4.8 shows an example run that demonstrates the behavior of expert and robot decisions during the machine learning process.

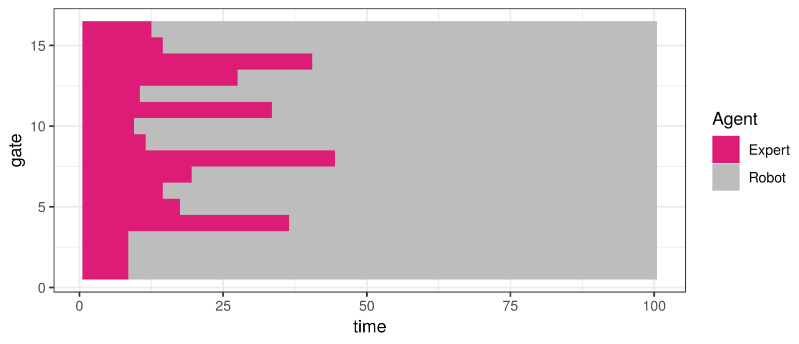


FIGURE 4.8: Decision Threshold Example for Robot and Expert

As the epistemic uncertainty is reduced, the robot makes decisions more frequently. This is a significant result from the model. The human, a priori, does not decide this distinction. It is the robot that realizes the uncertainty is too high for a confident decision.

The expert analysts in the super-agent can bring better contextualization to the decision where the robot is unsure about the quality of the decision. However, over time as the robot learns about the system, it can automate the majority of decisions. In the case that the system changes, the epistemic uncertainty may increase again which would drive more human involvement.

While this example setting of a small known gate set is trivial, there are many applications where it can be used with few modifications. However, many real-world data collection efforts result in gate sets that are too large to collect data using the methods discussed here. The following section expands these concepts to a real-world setting where the enumeration of the gate set is not feasible. In this case, the techniques of supervised learning discussed in Chapter 3 will be implemented.

4.2 Unknown Gate Set

In practice, threat categorization is done using a reliable database that has information useful for this task. Most classification is done through an aggregate list of indicators, such as the blacklists mentioned in Chapter 2. This research uses an open source blacklist aggregator by CriticalStack [47].

An aggregator service combines and vets a portfolio of threat reporting services. Open source lists are characteristically given as a set of indicators such as IP addresses, domain names, email addresses or file-hashes. However, more advanced classification lists can include behavior profiles. This is required when advanced actors work across or through multiple low-level indicators (multiple IP addresses for example). This list provides a subscription service that can be updated at various intervals. During the period of collection for this research almost 10 million indicators where collected. An in-depth exploration of this dataset is included in Appendix E. Table 4.2 gives a summary breakdown of types collected.

Table 4.2: Blacklist Collected Data Summary

Indicator Type	Quantity
IP Address	8,774,471
Domain Name	291,126
File Hash	43,934
URL	248,501

Each of these indicators has been validated by a third party as a threat source and is useful for classification purposes. There is some potential for bias in using an open source list. More advanced threats may operate in such a way that their indicators do not appear on these lists. Additionally, blacklists are compiled by organizations for internal use only and may not be shared in the open. The most obvious example of this is nation-states who use intelligence methods to aggregate lists. This list has been collected over an 18-month period from April 2016 to October 2017. Figure 4.9 shows the frequency of collection quantities.

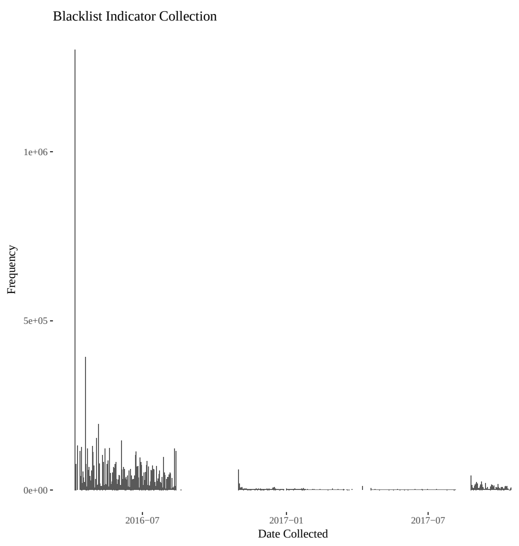


Figure 4.9: Blacklist Unique Indicator Collection Frequency

Early in the collection period when many indicators where being collected for the first time, the rate of new additions to the blacklist was high. However, as the list became more mature, fewer unique indicators are observed. This implies that many threats continue to use the same indicators. It is likely that low-level threats re-use indicators that still work for their purposes, where advanced actors may not behave the same way.

Table 4.3: Data Type by Kill Chain Stage (see Table 1 for reference)

j	Step	Data Type
1	Recon	network flow, boundary device logs
2	Weaponize	exploit database, vulnerability database
3	Deliver	IDS logs, host/server logs
4	Exploit	network flow, host/server logs, credential database
5	Control	host/server logs, credential database
6	Execute	network flow, host/server logs, credential database
7	Maintain	network flow, host/server logs, credential database

In addition to threat classification, data must be gathered for various steps in the kill chain. Figure 2.11 from the MITRE corporation gives an overview of the type of information that would need to be collected for forensic purposes to identify threat activity. Table 4.3 provides an example, by kill chain step, of what type of data needs to be obtained for forensic or classification methods.

These basic data types are generated by the security stack and are required to observe behavior characteristics over time. The following section discusses, in detail, an experiment which presents data collected during potential real-wold reconnaissance activities. This data is then used to classifiy actors.

4.2.1 Honeypot Experiment

This experiment will use data collected from real-world reconnaissance activity in order to learn the behavior characteristics of threats. This approach is motivated by the early warning system construct. To stay ahead of a threat in the kill chain, identification during reconnaissance has the potential to have the most significant value. This step, by definition, is the first touch of a defended system by a threat. However, the amount of data created by security stack boundary devices can grow large. Historically, it has been difficult for defenders to use this data for identification due to the size and inherent complexity. These considerations make this use case an ideal first application for machine learning and early warning systems.

The data for this experiment is collected through a set of deployed static sensors that behave like a public network entry point. This data has many of the same features that are found in the blacklists discussed earlier. A detailed exploration of the data can be found in Appendix F. These sensors are not configured or designed to attract a specific type of traffic, they are passive and collect metadata about each interaction with entities that initiate connection attempts. These types of sensors are often referred to as 'honeypots' as they can appear as easy targets for threats. For example, one piece of

Table 4.4: Honeypot Location, Provider and Type

Location	Cloud Service Providor	Type
Virginia, USA	Amazon Web Services	Snort / Cowrie
California, USA	Amazon Web Services	Snort / Cowrie
Frankfurt, DE	Amazon Web Services	Snort / Cowrie
Seoul, South Korea	Amazon Web Services	Snort / Cowrie
London	Digital Ocean	Snort / Cowrie
Toronto, Canada	Digital Ocean	Snort / Cowrie
Brazil	Azure	Snort / Cowrie
South East Asia	Azure	Snort / Cowrie

information collected is the username and password used during a login attempt. Figure 4.10 is a word cloud of the most commonly occouring values.

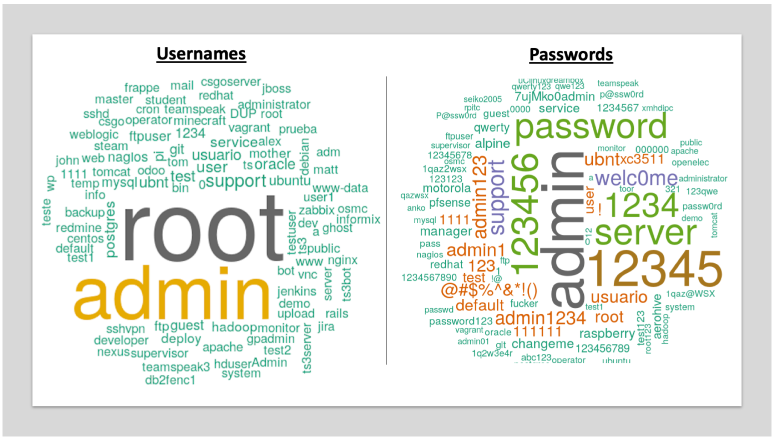


Figure 4.10: Common Usernames and Passwords Used on the Honeypots

Over the course of this experiment 18 sensors collected over 600,000 uninitiated reconnaissance events. The two types of sensors, Snort and Cowrie, receive different kinds of data. Snort sensors are a type of IDS that collects generic connection attempts.[1] Cowrie sensors are environment emulators which collect data, like username and password, on login attempts for a single port (22). Table 4.4 gives some information about each type of honeypot deployed. Given the discrete nature of the collection location, technique and sensor types there is likely some bias in the observed data. One possibility is that more advanced actors may not be present in the sample set as they would avoid common detection techniques employed in the experiment.

Figure 4.11 represents the aggregation of collection for honeypot indicators. The time horizon for collecting these events was from March to October 2017. Where there were some gaps of accumulation due to maintenance on the devices. This practice was consistent as the sensor deployment ramped up.

[1]Generic connection attempts are gathered using the base signatures that are distributed with the Snort community deployment.

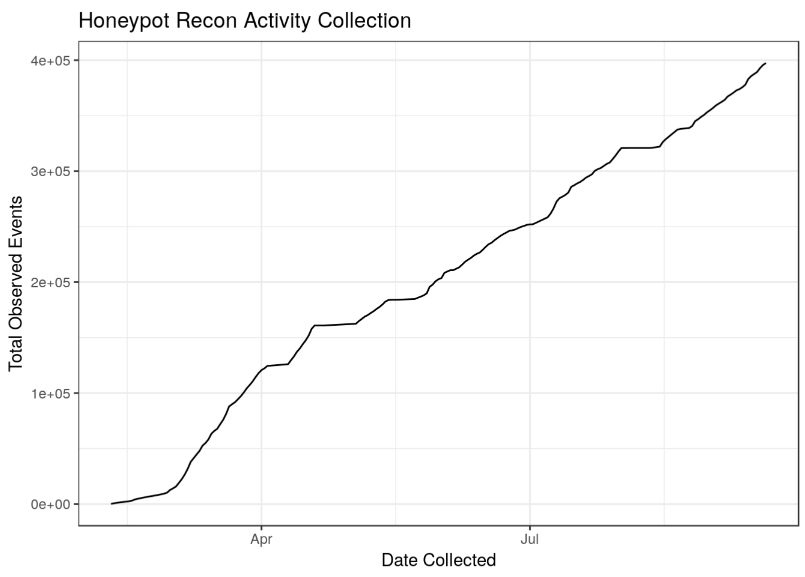


FIGURE 4.11: Honeypot Indicator Collection

TABLE 4.5: Honeypot Data Summary

Factor	Quantity
Unique IP Addresses	40,450
Number on Blacklist	13,630

4.2.2 Results

For this basic model, a few assumptions are required. First, individual actors can be identified by unique IP addresses. Second, sufficient behavioral characteristics can be gathered from the collected data. Third, for classification, threat and non-threat behavior are stationary. The honeypot collected data, when compared to the open source blacklist, has the characteristics in table 4.5.

Using this information, supervised machine learning techniques can be explored. A first exploratory example feature would be total frequency. This will answer the question, 'does higher (or less) frequency correlate to higher (or less) likelihood of being on the blacklist?' Figure 4.12 shows an ordered plot of all unique IP address by total frequency over the collection period.

A cursory review of Figure 4.12 shows that the higher the frequency of appearances within the collected honeypot data, the higher the proportion of classified threats. The data demonstrates a power law relationship with one IP address accounting for 5% of all observations. Also, over half of the other addresses are observed just one time. Figure 4.13 shows the proportion of threats being classified as the total observed frequency increases.

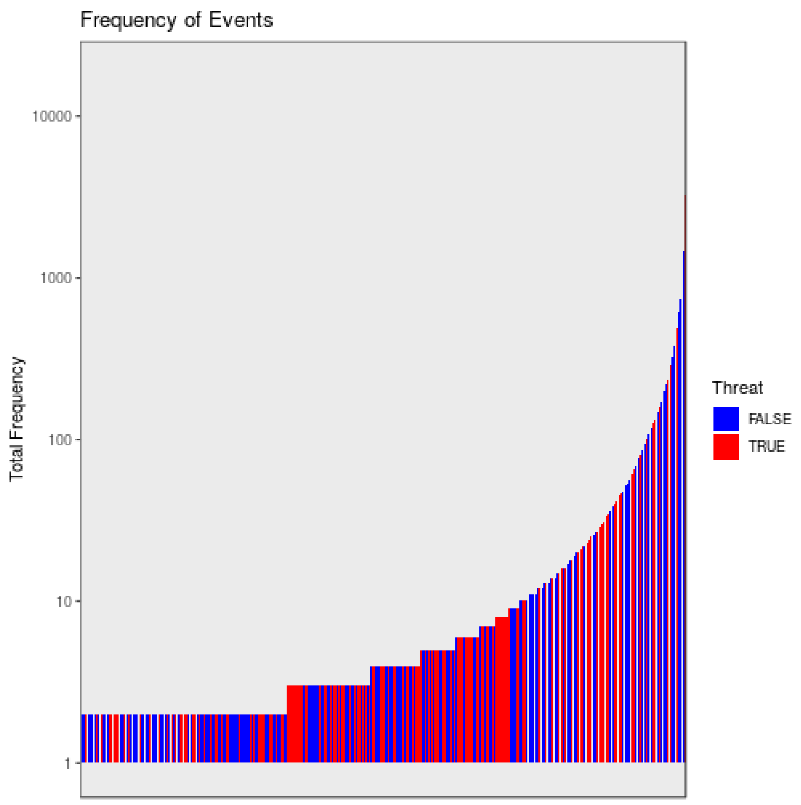


Figure 4.12: Honeypot IP Address by Total Observations (True is classified a threat)

As discussed in Chapter 3 the number of gates can be substantial (for example, containing all IP address, and port combinations). In order to manage the situation, a set of supervised learning models are trained using blacklist labeled historical behavioral data. The set of learning models used for classification and utility estimation for this research include.

- Logistic Regression
- k Nearest Neighbors
- Random Forest
- Neural Networks

The most dynamic of the candidate models is the neural network. Initial work on feature set encoding and training can be found in Appendix G. The feature engineering analysis can also be found in Appendix F. In all, there are 28 behavioral characteristics used during the supervised learning process. Table 4.6 is a list of feature categories used.

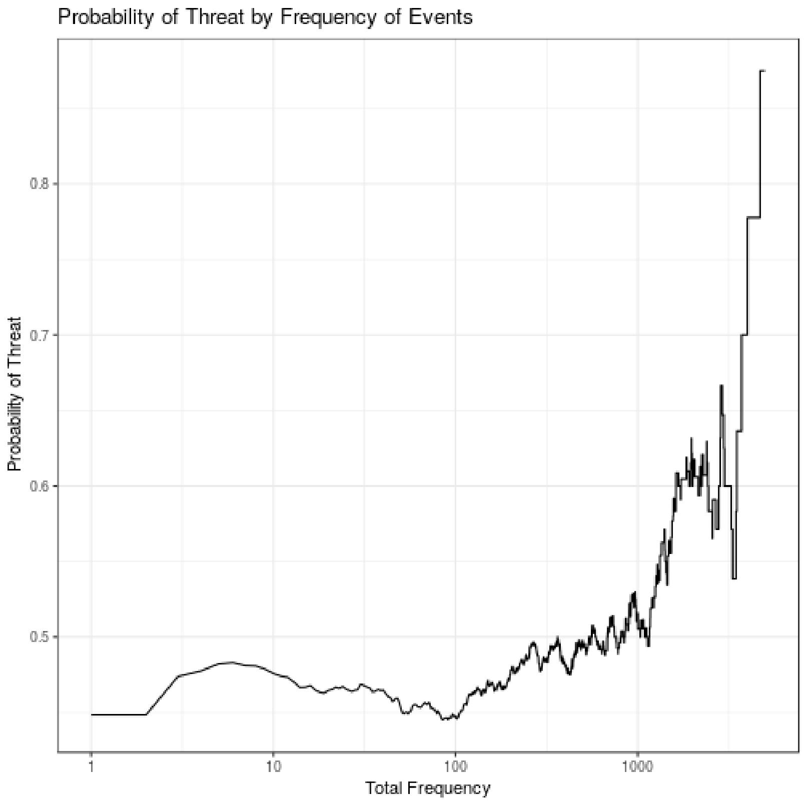


Figure 4.13: Portion of Classified Threats with Greater Frequencies

Table 4.6: Example Feature Categories

Category	Example	Description
Signature	IP Address	network level
	Geo-location	location based
Temporal	Inter-arrival time	time between sessions
	Time of Day	patterns of work cycles
	Pattern of life	general patterns for actors
Frequency	Top talkers	high frequency
	Infrequent visitors	seldom visits
Other	Packet header	technical connection information
	Amount of data sent/received	level of activity with the network
	Promiscuity	number of unique behaviors used

Table 4.7: Supervised Learning Model Accuracy

Model	Accuracy
logistic regression	0.86
random forest - 1000 trees	0.86
neural network 3 hidden layers	0.86
10 nearest neighbors	0.90
8 nearest neighbors	0.92
6 nearest neighbors	0.90
4 nearest neighbors	$\tilde{0}$.99
2 nearest neighbors	$\tilde{0}$.99

With these features as inputs Table 4.7 shows the accuracy of the best performing models after training.

The regression, neural network (shallow) and random forest models all perform well with an accuracy of 86%. However, the best performing models are the nearest neighbors. The lower the k value in the kNN models the better the accuracy. This implies that actor types do tend to be similar as measured by behavior. Exploring this result further the 28 features are reduced into 2 principal components and shown in Figure 4.14.

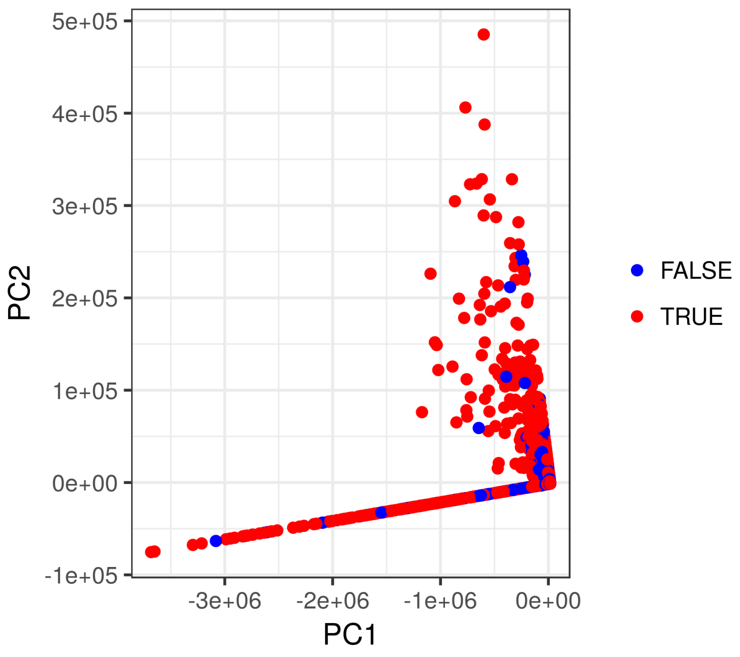


Figure 4.14: Principal Component Analysis of Supervised Learning Feature Set

Figure 4.14 shows that non-threats (non-blacklisted) IP addresses tend to all collect in one area. Those that are different tend to be threats. This supports both the traditional anomaly approaches discussed in Chapter 2 and the behavior techniques presented in this research. The next step in the model is to develop a gate set with the trained models. The following section presents a method using the random forest.

4.2.2.1 Unknown Gate Set Feature Importance

Using the concept of importance from the random forest model, global features can be evaluated by their sensitivity to cost. Figure 4.15 is a graph of the most important features.

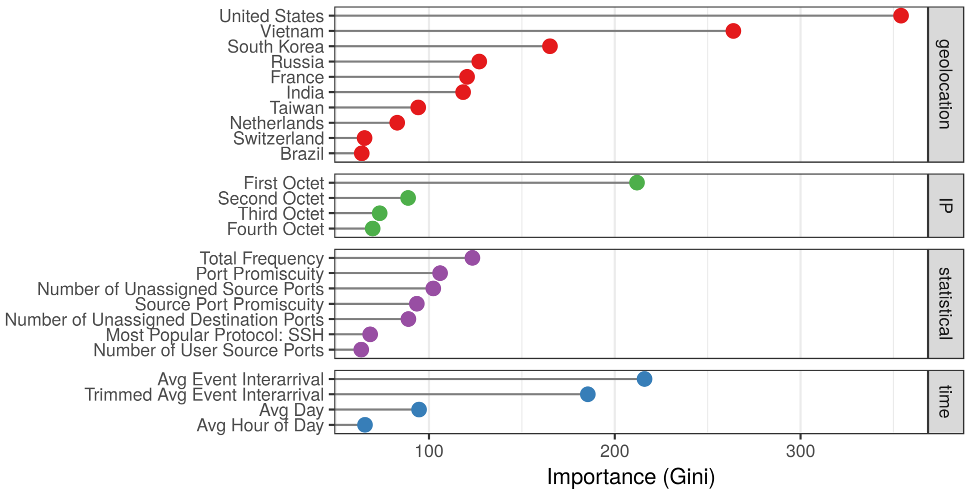


FIGURE 4.15: Important Features for Random Forest Model

There are several interesting outcomes from the important feature analysis. The IP address characteristics are among the most influential. Geolocation and octet values have the most significant impacts on classification. Other statistically derived behaviors are also influential. Temporal and communication port characteristics, along with frequency and promiscuity, are imperative. Using this information, one possible gate set could be formulated as:

- Block all activity from countries: United States, Russia, Taiwan, Korea, India, France, Netherlands, Switzerland, Brazil
- Block any other behavior with: Frequency greater than 100, Interarrival less than 0.05 days, Any Port Promiscuity greater than 5, and Communicating from an unassigned port

4.2.2.2 Uncertainty

Similar to the known gate set model the concept of uncertainty can be leveraged to motivate expert-human involvement by the super-agent. However, unlike the Bayesian methods used earlier, epistemic uncertainty is not easily calculated from the supervised learning techniques. The approach used here is to evaluate the distribution of probabilistic predictions from the candidate learning models. The metalog distribution is used for this purpose owing to its flexibility.

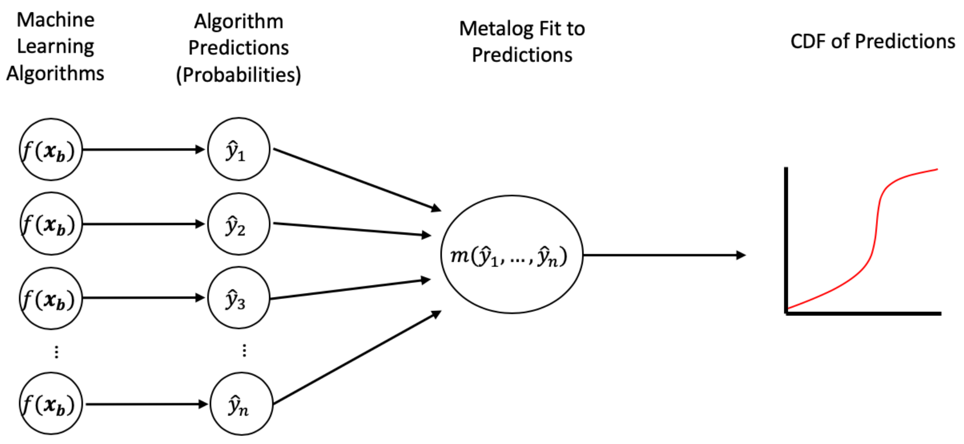

Machine Learning Algorithms
Algorithm Predictions (Probabilities)
Metalog Fit to Predictions
CDF of Predictions
$f(x_b)$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
$\hat{y}_n$
$m(\hat{y}_1, \ldots, \hat{y}_n)$

FIGURE 4.16: Epistemic Uncertainty Evaluation

For each event, predictions are gathered from each of the learners and then used to make a distribution. The wider the distribution the more uncertain the super-agent is about the decision. Consider the geolocation features from the previous section. Figure 4.17 shows the epistemic uncertainty dynamics with different countries of origin.

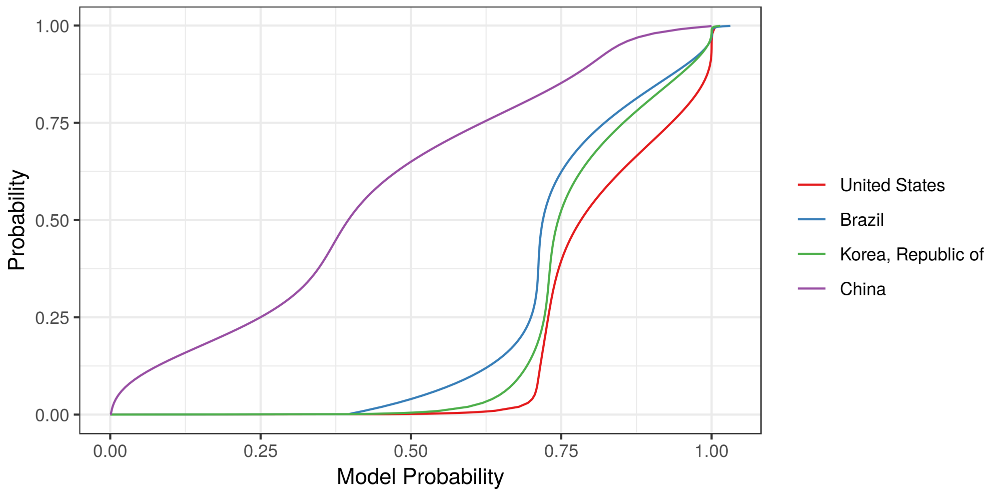


FIGURE 4.17: Epistemic Uncertainty Example with Geolocation

For expert-human involvement decisions concerning IP addresses orginating from the United States could be safely automated (done by the robot). However, decisions about Chinese IP addresses have much more uncertainty and would be much better candidates for human intervention.

The outcomes from the models presented in this chapter demonstrate signifigant insight and value. Both the known and unknown gate set methods yield practical and important contributions. The next chapter discusses conclusions and future work.

Chapter 5

Conclusion and Future Work

The cybersecurity domain is inherently complex. The activities and actors are obscured by a technical layer of hardware and software. Even so, all facets of modern life are impacted by services provided by information systems. Securing these systems from threats is a substantial challenge owing, in part, to the complexity. In order to better manage the problem, this research presents and demonstrates a model that is intuitive and practical. The gate-set concept is flexible and robust in its application for defenders. It is useful as a mental-model, valid for communication, and independent evaluation of security policies. This novel approach gives system defenders an effective tool to conduct probabilistic risk management. Unique contributions generated for this research include:

1. Formulation of the defensive cybersecurity problem as a behavior 'gate-set' optimization
2. Developed AI and human super-agent concept using cooperative decisions
3. Large gate set policy generation using feature sensitivity
4. Novel metalog applications (sampling and uncertainty estimation)
5. **rmetalog** and **pymetalog** software packages
6. Novel cyber behavior dataset using real-world honey pots and an open source blacklist

Risk in cybersecurity is generated when an organization relies on an information system for operations or the storage of organizational/trade/industry secrets. Threats are constantly trying to compromise systems in order to gain control or steal the secrets. In this view, risk is best represented as the presence and behavior of threats against a defended system. The gate-set concept provides a framework to quantify threat impact based on their unique behavior profiles. The fundamental contribution of this research is a methodology which evaluates actor behavior in order to recommend an optimal security policy. However, there are several considerations a defender must deal with in order to develop this policy.

With the gate-set concept, the cybersecurity environment can be improved through the employment of artificially intelligent (AI) technologies. Large datasets generated by security stacks are best evaluated using AI. This research provides a complete framework for deploying such technologies in a practical setting. Intelligent access control leverages the strengths of both AI and expert-human decision makers in order to create optimal gate set policies.

In addition, this research highlights many improvements to be made in cybersecurity research. The most critical is more and better-quality data from which AI algorithms can learn. Currently, the shortfalls of the existing models originate a lack of quality data. In order to address the data problem and demonstrate the value of the model proposed in Chapter 3, this research uses a self-generated, non-canonical dataset. The dataset is created through a globally deployed set of honeypots collecting real-world reconnaissance attempts. Reconnaissance is the first step in the kill-chain, making this data relevant for early threat detection and response. Additionally, events are classified using an open source blacklist. The honeypot collection and labeling apparatus mirror the typical real-world conditions of a system defender.

However, using a self-generated dataset which contains only honeypot events labeled with an imperfect blacklist may lead to irregularities and performance concerns. This is approach, despite its faults, is an improvement on the existing canonical datasets which may not represent reality. For example, in Chapter 2 the NSL-KDD dataset, the most used cyber dataset in academic literature, is discussed. This dataset (NSL-KDD) was developed artificially to identify attacks (not specific actors) and has well known biases, making it unrealistic and impractical for real-world use. In contrast, the honeypot data contains rich true-to-life behavioral information about actors. Instead of looking for active attacks the model uses this data to ask, 'how do specific actors behave?' This view is both practical and useful for informing a defensive (gate-set) policy.

Real-world data collection involves the problem of dimensionality. The gate set grows too large and becomes unmanageable. Supervised learning is used to address this issue. The proposed approach is that behaviors are used as model features and used to train a classification (or regression) model. The model then becomes the knowledge center to build a gate set policy through the exploration of feature sensitivity. The sensitivity of inputs give a direct connection to actor behavior, which then inform the gate-set policy.

With the supervised learning approach, feature engineering is required to both manage dimensionality and improve accuracy within the constraints of computational performance. Encoding of behaviors, such as source and destination ports, is once such method. Due to the large amount of ports (over 65 thousand), without encoding, entries would be large, sparse, and one-hot vectors. While improving learning performance this approach also reduces the decision space. The balance between encoding and sparse vectors for supervised learning in cybersecurity is an important area with little work done. Future iterations of models should look at various combinations of encoding methods.

Labeling the data also can be significantly improved. For example, there are other known indicators of potentially threatening behavior for IP addresses other than being placed on a blacklist. For example, an IP address that tries to login to the administrative port can safely be assumed to be a threat without ever showing up on a blacklist. This additional labeling could improve the real-world results of the current models. It is possible that the model presented in Chapter 4 predicts and detects threatening behavior from IP addresses listed as threats and the same behavior from unlisted IP addresses also as threats. However, in its current form, these are counted as false positives. Better labeling has the potential to address these type I and II error problems.

The concepts presented in this research represent the foundation for future work. Specifically, a full probabilistic model using signals, events, countermeasures, and their effectiveness. The use of classification techniques in machine learning can be expanded significantly to include a more robust feature set and actor classification. Additionally, the metalog probability distribution should be evaluated for usefulness in as an improvement over the sigmoid function used in standard logit regression and deep learning. There are a great many steps needed to complete a comprehensive warning system. The basic model presented here does not evaluate cost-benefit decisions for strategic considerations. In addition, Chapter 4 presents two single stage models. Multi-stage models are described in Chapter 3 but not implemented. Expanding the application to include reinforcement learning and a POMDP learning method are important extensions.

This research presents a framework to use as the foundation for applying machine learning and artificial intelligence in a cooperative environment. The super-agent concept is a fundamental field of research that will be explored in much greater depth.

Appendix A

Early Warning Systems Overview

The general application of generic warning systems is given here. This review is derived from work by Paté-Cornell [57].

Table A.1: Early Warning System Signal Thresholds

Thresholds	Description
ϕ	Warning Threshold that triggers an alert to a decision maker
ϕ^*	Optimal Threshold that provides the maximum amount of benefits

Events are discrete observations in the warning system that govern the behavior. They are critical for calculating costs and benefits of changes in thresholds. Rates are the pace at which given events take place over a specified time horizon.

Table A.2: Early Warning System Rates

Rate	Description
λ	rate of events
λ_w	rate of warnings
λ_t	rate of true alerts
λ_f	rate of false alerts
λ_1	rate of irrelevant alerts
λ_2	rate of alerts where the hazard threshold is exceeded

The cost and benefit of a system are evaluated in the context of rates.

Probabilities of events have an important impact on the system behavior.

With this notation, basic computations can be established for context. Rates have the following relationships.

Table A.3: Early Warning System Costs

Cost&Benefit	Description
C_w	Cost of warning system
C_t	Cost of response with high response rate (following a true alert)
C_f	Cost of response with low response rate (following a false alert)
α	portion of entities saved given a true alert

Table A.4: Early Warning System Probabilities

Probability	Description
$p(E\|S)$	probability of event given signal
$p(S\|E)$	probability of signal given event
p_t	probability of signal being true
p_f	probability of signal being false

$$\lambda_w = \lambda_t + \lambda_f \tag{A.1}$$

$$= \lambda_1 + \lambda_2 \tag{A.2}$$

In addition, the rate of true signals can be expressed.

$$\lambda_{A_t} = \lambda_{A_f} p(E_n|S_n) \tag{A.3}$$

$$= (\lambda_1 + \lambda_2) p(E_n|S_n) \tag{A.4}$$

$$= \lambda p(S_n|E_n) \tag{A.5}$$

$$given \tag{A.6}$$

$$\lambda_w p(E|S) = \lambda p(S|E) \tag{A.7}$$

$$\lambda_w = \lambda \tfrac{p(S|E)}{p(E|S)} \tag{A.8}$$

In the same way, false alerts can be evaluated.

$$\lambda_{A_f} = \lambda_w - \lambda_{A_t} = (\lambda_1 + \lambda_2) - \lambda p(S_n|E_n) \tag{A.9}$$

Benefits and costs can be calculated in a memoryless context using the following straight forward method.

Benefit is the expected number of entities saved per time unit.

$$EV(\delta L) = \alpha \lambda_t N_t = \alpha N_t \lambda p(S|E) \tag{A.10}$$

Cost is the expected value of the costs of all alerts per time unit

$$EV(C) = C_w \lambda_w = C_w(\lambda_1 + \lambda_2) = C_w(\lambda \frac{p(S|E)}{p(E|S)}) \tag{A.11}$$

In more advanced constructs there is a memory with systems where the probability of response (α) changes based on previous system outcomes. This is modeled using a Markov process. For this section, we will only consider a one-step memory technique that will be expanded more in the chapter 3.

Appendix B

Machine Learning Background Example Code

```
#split into train and test sets
n <- nrow(newtest2)

test.size <- as.integer(n/3)
testset <- sample(n, test.size)
test <- newtest2[testset, ]

train <- newtest2[-testset, ]
#train the model
fit.train<-glm(Var2 ~ Freq, data = train,family=binomial())
summary(fit.train)
#for testing portion
test_values<- 1/(1 + exp(0.4656336 - 0.0009246 * test$Freq))

pred_simple <- prediction(test_values, test$Var2)
perf_simple <- performance(pred_simple, "tpr", "fpr")
a <- plot(perf_simple, col = "blue",xlim=c(0,1),ylim=c(0,1))
abline(a=0,b=1,col='red')
```

Appendix C

Machine Learning Application Model Code and Output

The following code is used to generate the plots used for the example machine learning problem in chapert 2. First step is to take the sample data set and separates it into a training and testing set at 2/3 and 1/3 respectively.

```
set.seed(123)
n <- nrow(memproc)

test.size <- as.integer(n/3)
testset <- sample(n, test.size)
test <- memproc[testset, ]

train <- memproc[-testset, ]
```

For this example, a simple comparison to average values for the two variables will be used.

```

inf <- colMeans(train[train$state=="Infected", c("proc", "mem")])
nrm <- colMeans(train[train$state=="Normal", c("proc", "mem")])




  predict.infect <- function(data) {

  proc <- as.numeric(data[['proc']])
  mem <- as.numeric(data[['mem']])

  inf.a <- inf['proc'] - proc
  inf.b <- inf['mem'] - mem

  inf.dist <- sqrt(inf.a^2 + inf.b^2)
  nrm.a <- nrm['proc'] - proc
  nrm.b <- nrm['mem'] - mem
  nrm.dist <- sqrt(nrm.a^2 + nrm.b^2)

  ifelse(inf.dist<nrm.dist,"Infected", "Normal")
}
```

With this function, the algorithm can be used to generate predictions from the test set. An important point here is that this example did not include a, so-called, cost function. This is a tool that advanced methods of learning use to improve. This concept will be discussed in further detail in Chapter 3 when exploring modeling techniques.

```
prd<-apply(test,1,predict.infect)
```

Output from initial ML training approach.

```
## glm(formula = Threat ~ Freq, family = binomial(), data = train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.7917   -0.9875   -0.9875    1.3792    1.3800
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.4656336  0.0150126 -31.016  < 2e-16 ***
## Freq         0.0009246  0.0002012   4.595 4.33e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 25586  on 19150  degrees of freedom
## Residual deviance: 25556  on 19149  degrees of freedom
## AIC: 25560
##
## Number of Fisher Scoring iterations: 4
```

Appendix D

Metalog Review and Modification

D.1 Review

The metalog distribution is a family of modern data driven continuous probability distributions. Using empirical observations as quantiles a metalog distribution is fit to the natural shape of the data without opinion. These distributions, after fit, are simple closed form expressions that can be efficiently leveraged for exploration, simulation, or general probabilistic analysis.

The metalog distributions in their current from are presented by Keelin in 2016 [96]. However, they are inspired by earlier work by Keelin and Powley, on quantile parameterized distributions [105]. The general concept is that a distribution should be granted the flexibility to represent data in its natural state. This view is quite different than the classical distribution families that are constructed for interpretability, ease of use, mathematical elegance or historical precedent. The need for these distributions comes from the inherent inflexibility of currently popular methods. For example, the normal distribution is limited to fitting the first two moments of empirical data, beyond this the skewness and kurtosis are fixed (0 and 3 respectively).

Even in the early days of probabilistic analysis empirical datasets demonstrated substantial differences in character than could be captured by inflexible methods. One famous example is the students t-distribution that was developed by Gosset because the tail thickness of the gaussian distribution couldnt fit the data gathered [106]. It was an empirical reason that motivated the new distribution not theory. This became a common theme with an ever-expanding universe of distributions constructed for specific use cases. This expanding set of specific distributions continues today. The implicit goal is a probability distribution that is flexible enough to fit any valid continuous dataset.

The metalog distribution completes this journey by providing an un-opinionated distribution that is parameterized directly by the data itself. The user has only to select how much flexibility to give the distribution, through the use of increasing terms. As terms increase so does the distributions shape flexibility.

The metalog distribution takes the shape of the data that it represents. However, the process of generating a distribution requires some user input. In particular, the amount of flexibility a distribution has is determined by the number of terms, which is a value selected a priori. Also, the number of terms must be less than the number of available data point observations.

Each term has two components, an a_i value and an $f(y_i)$ value where y_i is a cumulative probability value for a given data point quantile. The $f(y_i)$ values are determined by the characteristics of the metalog terms as defined in [96]. The distribution is built using an optimization algorithm that finds the best fit a_i values (real numbers) to a vector of observed data x. Once the distribution is built the a_i values become constant parameters. For example, equation D.1 is a four term metalog quantile distribution.

$$M(y) = a_1 + a_2 ln(\frac{y}{1-y_1}) + a_3(y-0.5)ln(\frac{y}{1-y}) + a_4(y-0.5)\ldots \tag{D.1}$$

Figure D.1 shows the flexibility of the metalog on an example multimodal data set as terms increase. While the underlying data is the same, the distribution gives signifigant flexibility to map the informational nuances of the raw data with more terms.

The updating scheme for this research is to rebuild the metalog distribution with every newly observed data point. As actors arrive at a gate the type and utility are observed then appended to the other historical observations. This approach is more intuitive than the Bayesian approach of needing to modify parameters of an assumed distribution. Also, it requires much less data than the MLE approach to build robust distributional shapes. In the dynamic setting where the underlying system characteristics are changing the metalogs potential is the most powerful. Older, stale information can simply be left off or replaced with new information at the super-agents discretion when building a distribution.

One concern is that the flexibility of the metalog brings the drawback of potential overfitting. Specific applications may be more sensitive than others to noisy observations. In addition, the continuous distribution may not be ideal for estimating discrete probabilities the way the Beta/Binomial conjugacy setting are. The trade-offs are explored in the applications discussed in chapter 4.

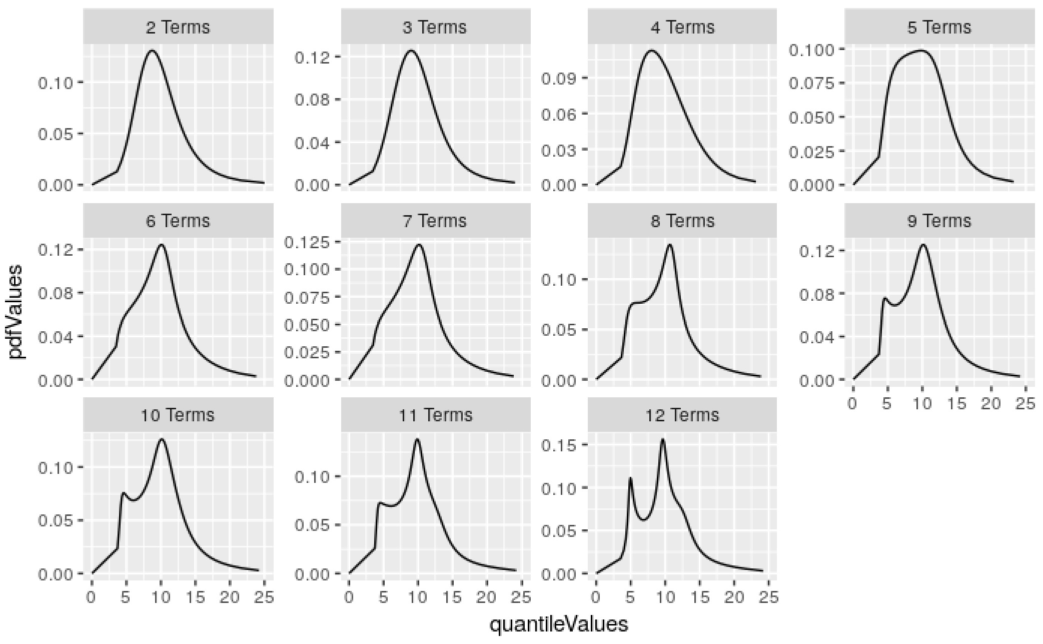


FIGURE D.1: Example Metalog Distribution Demonstrating Term Flexibility

D.2 Modification for Greater Robustness

The most significant consideration when using the metalog distribution is the consideration of feasibility. There are differing considerations between the relationship of terms and the number of available data points. Because the a_i value as linear in the their relationship to the $f(y_i)$ and x values (functional and empirical values) the OLS technique is conveniently applied.

$$\mathbf{x} = \mathbf{aY} \tag{D.2}$$

If the number of available data points is equal to the number of terms, a simple closed form solution to the a vector can be calculated.

$$\mathbf{a} = \mathbf{Y}_n^{-1}\mathbf{x} \tag{D.3}$$

This approach is possible because there are n equations and n unknowns in this system. This type of closed form approach is useful in a small data setting or where expert solicitation returns a minimal set of information (for example, asking for high, medium, and low values). However, when the number of data points exceed the number of terms, a distribution is referred to as being fit in extended form (extended metalog). This circumstance requires a more general fitting method. Fortunately, the linear properties which allowed for equation D.3 also allow for other well-known methods to be used in this circumstance. The proposed method of fitting by Keelin is ordinary least squares (OLS). With OLS solving for the best fit **a** is simply stated in a closed form way.

$$\mathbf{a} = [\mathbf{Y^T Y}]^{-1}\mathbf{Y^T x} \tag{D.4}$$

However, neither the method given in equation D.3 nor D.4 are is constrained by the feasibility conditions. Keelin presents the sufficient conditions required for a metalog to be valid [96]. However, in application he only uses the feasibility condition to check the validity of a distribution after it is created. The basic constraint is that the PDF $(m_n(y))$ must be positive for all values of y and n is the specified number of terms, implying that the CDF is strictly increasing.

$$m_n(y) = [f'(y)\mathbf{a}]^{-1} > 0 \forall y \in \{0, 1\} \tag{D.5}$$

Equation D.5 is derived by taking the first derivative of equation 3.25 with respect to y and inverting the result. Ignoring this constraint often results in distributions that are not valid. Furthermore, it is not clear what conditions lead to infeasibility. This makes the metalog unstable for use in several application areas which require consistent robust distributions; these may include automated environments, reliability calculations and other probabilistic analysis. For example, consider the following simple example of three data points and three probabilities.

$$\begin{aligned} x &= \{10, 20, 40, 100\} \\ y &= \{0.1, 0.5, 0.9, 0.99\} \end{aligned} \tag{D.6}$$

This simple application, when using methods D.3 or D.4 results in an infeasible distribution. Consider Figure D.2 the fitted distribution results in a pdf that has negative values about the mean.

Inspection of the CDF in Figure D.3 give some indication as to why this is happening.

The CDF folds back on itself with respect to the quantile values. This can happen because the cumulative probability values are not a function of the quantile values. Indeed, Figures D.2 and D.3 are not produced functionally but, instead out of convenience. Both the quantile values in equation 3.25 and the probabilities in equation D.5 are a function of cumulative values y. This is a different form than is typically found with most traditional probability distributions, which are most often a function of some quantile value.

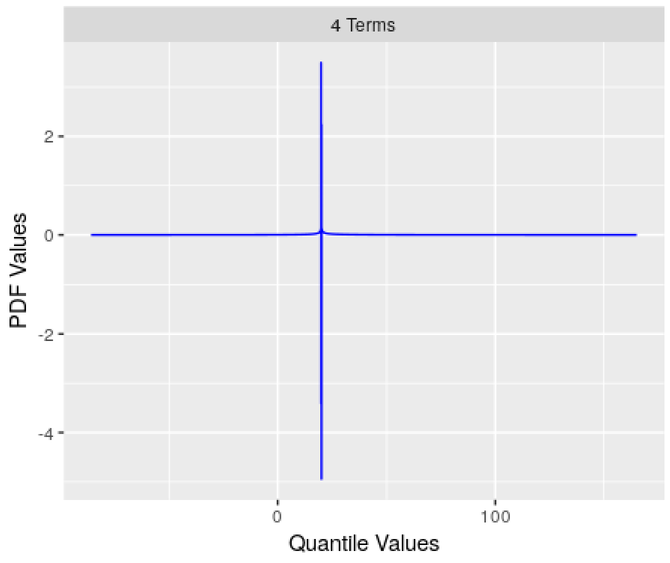


FIGURE D.2: Example Metalog Distribution PDF

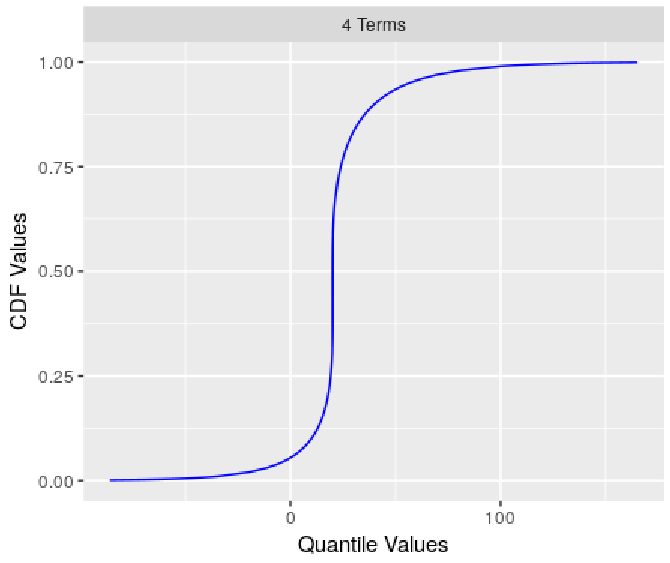


FIGURE D.3: Example Metalog Distribution CDF

The metalog form has a major convivence for sampling but a drawback in lending itself to having simple methods of ensuring feasibility and stability.

The reason the constraint in equation D.5 cannot be directly included within the closed form methods like OLS optimization routine is that it must be valid for all values of y between 0 and 1. However, the values of y are determined as probabilities from the empirically collected data, which are finite. Because the optimization has the **a** vector as the decision variables and the y values as parameters there is no simple method to expand the optimization. This research proposes a method to partially address this shortcoming.

First, the optimization fitting scheme is reformulated from OLS to a linear program (LP) minimizing the absolute value differences where z is the number of data points.

$$\begin{aligned} \underset{a}{\text{minimize}} \quad & \sum_{i=1}^{z} |M_n(y_i) - x_i| \\ \text{subject to} \quad & M_n(y) = \mathbf{aY} \\ & y_i \in \{0,1\} \forall i \end{aligned} \tag{D.7}$$

This type of program is simple to solve using well known methods. In addition, linear constraints can be included resulting in a manageable optimization program. In order to address the feasibility requirement, the program is expanded to include a grid of g values with some error (ϵ) are required to be greater than or equal to 0.

$$\begin{aligned} m_n(y) = [f'(y)\mathbf{a}]^{-1} >= 0 + \epsilon \\ \forall y \in g \end{aligned} \tag{D.8}$$

The selection of the step size in the grid is an important trade-off. The larger the step size the better the performance of the LP. However, the smaller the step size the less likely the resulting distribution is to be infeasible. Using the linear program method the above example can now be solved with a resulting feasible distribution.

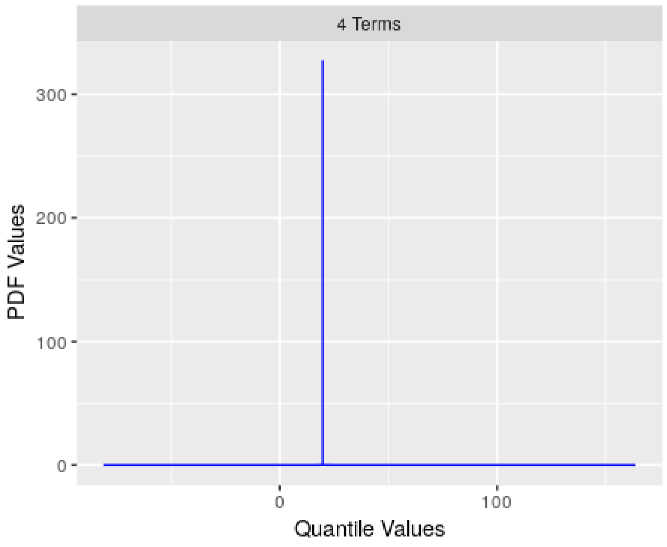


FIGURE D.4: Example Feasible Metalog Distribution PDF with LP

Figure D.4 shows the resulting distribution using this more robust method. While the form is not a traditional smooth shape, it does meet the conditions required of a probability distribution.

This method is much more robust to infeasibility than the methods originally proposed by Keelin but there is a draw back. With a large amount of data points collected the can quickly become difficult to compute. The reason for this is that empirical values become constraints in the optimization, which exponentially grow computation time. For example, a million data point distribution (not uncommon in many settings) would become intractable without significant computation capabilities.

In order to address this issue, the methodology used in this research is to create a representative vector of the continuous empirical values. This is a similar approach that Keelin is proposing with the generalized metalog formulation (as opposed to the extended form). In the general form a representative vector of ten probabilities and ten terms is used exclusively. The application in this research will allow greater flexibility than the general form at the cost of computation time.

The representative vector here is restricted to a maximal length of 1000 elements (in addition to some over representation in the tails). Each of these elements are evenly spaced probabilities between 0 and 1. This representative vector is then used for the OLS and LP techniques discussed above. The algorithm for developing a metalog distribution then becomes.

TABLE D.1: Metalog Fitting Algorithm

Algorithm: Metalog Fitting

```
for $n = 1, 2, ....$ do
  if(length($x$)>100
    Build representative vector x
  Run OLS
  if(OLS is valid)
    return OLS result
  else
    Run Linear Program
    Return Linear Program result
end for $n$
```

The algorithm D.1 checks to see the length of the data and if to long for computation builds a representative vector. While this technique is more robust than the pure OLS method presented by Keelin, there remain several circumstances where a distribution can be infeasible due to the in-exact grid method. However, these conditions are much less common and make the distributions useful in a greater number of real-world use cases. This methodology is implemented in an R package which is discussed in the following section.

D.3 Overview of R Package

This research has developed an implementation of the metalog distribution into an R package. The development version of this package is available at this link.

To install the package from this repository use the following:

```
library(devtools)
install_github('isaacfab/RMetalog')
```

Once the package is loaded you start with a data set of continuous observations. For this repository, we will load the library and use an example of fish size measurements from the Pacific Northwest. This data set is illustrative to demonstrate the flexibility of the metalog distribution as it is bi-modal. The data is installed with the package.

```
library(RMetalog)

data("fishSize")
summary(fishSize)
```

The base function for the package to create distributions is:

```
r_metalog()
```

This function takes several inputs:

- x - vector of numeric data
- term_limit - integer between 3 and 30, specifying the number of metalog distributions, with respective terms, terms to build (default 13)
- bounds - numeric vector specifying lower or upper bounds, none required if the distribution is unbounded
- boundedness - character string specifying unbounded, semi-bounded upper, semi-bounded lower or bounded; accepts values u, su, sl and b (default 'u')
- term_lower_bound - (Optional) the smallest term to generate, used to minimize computation must be less than term_limit (default is 2)

- step_len - (Optional) size of steps to summarize the distribution (between 0.001 and 0.01, which is between approx 1000 and 100 summarized points). This is only used if the data vector length is greater than 100.
- probs - (Optional) probability quantiles, same length as x

Here is an example of a lower bounded distribution build.

```
my_metalog <- r_metalog(fishSize$FishSize,
                        term_limit = 9,
                        term_lower_bound = 2,
                        bounds=c(0,60),
                        boundedness = 'b',
                        step_len = .01)
```

The function returns an object of class rmetalog and list. You can get a summary of the distributions using summary.

```
r_metalog_summary(my_metalog)
```

The summary shows if there is a valid distribution for a given term and what method was used to fit the data. There are currently only two methods traditional least squares (OLS) and a more robust constrained linear program. The package attempts to use OLS and if it returns an invalid distribution (described in the literature referenced above) tries the linear program. You can also plot a quick visual comparison of the distributions by term.

```
r_metalog_plot(my_metalog)
```

As the pdf plot shows with a higher number of terms, the bi-modal nature of the distribution is revealed which can then be leveraged for further analysis. Once the distributions are built, you can create ‘n‘ samples by selecting a term.

```
s<-r_metalog_sample(my_metalog,n=1000,term=9)
hist(s)
```

You can also retrieve quantile values with a probability in a similar way.

```
r_metalog_quantile(my_metalog,y=c(0.25,0.5,0.75),term=9)
```

Appendix E

Exploratory Data Analysis of Collected IP Blacklist

This research uses publicly available real-world threat feeds to curate a useful blacklist of IP addresses. The following sections are an exploratory analysis of the raw data collected during this process. This list is intended to be representative of a real-world list that cybersecurity professionals would use to implement a defensive posture policy.

Exploratory Data Analysis of Collected IP Blacklist

Overview

This research uses publicly available real-world threat feeds to curate a useful blacklist of IP addresses. The following sections are an exploratory analysis of the raw data collected during this process. This list is intended to be representative of a real-world list that cybersecurity professionals would use to implement a defensive posture policy.

```
# load the blacklist data
library(tidyverse)
library(rgeolocate)

# this is quite a big files so you will need more memory
# than the free tier provides
load("~/analysis/data/historical.RData")
```

Blacklists are fluid documents in cybersecurity. They represent the current state of known threat indicators that may need to be acted upon. Indicators, such as IP addresses, domain names, and file hashes are added and removed regularly. For this research, a non-canonical (for exploratory and example use only) blacklist is collected from open source threat intel feeds collected by the Critical Stack Intel Market Place. It contains real-world observations used by cybersecurity practitioners daily. However, this compiled list represents a periodic snapshot of the state of all such lists. The list used here includes the date on which in the indicator was collected from the Critical Stack service (a proxy for when the indicator was placed on the list). Updates were performed daily using an automated script. This list can be useful to see what do the internets bad actors use as their last point of observable reference. It is not a complete record of threats but some meaningful subset useful for analysis.

How an indicator is put on the list is different based on the source that adds it. In general, some type of activity was observed by the indicator at some point in time. Some of the blacklist data sources (feeds) are passively collected, like the Modern Honey Network, where others are a product of more formal analysis. There is a collection of security professionals that make indicator data available to the broader community. Sometimes activity is flagged that is the result of legitimate activity resulting in false positives. However, this research will assume that this is a low freqency exception and can be safely ignored. This list has four variables:

```
nrow(historical)
#> [1] 9358041
head(historical)
#> fields_indicator indicator_type Date
#> 1 adserving.cpminventory.com Intel::DOMAIN
2016-04-06
#> 2 188.2.98.161 Intel::ADDR 2016-04-06
#> 3 retromoderators.hansonthebeach.com Intel::DOMAIN
2016-04-06
#> 4 65.219.249.178 Intel::ADDR 2016-04-06
#> 5 117.218.48.31 Intel::ADDR 2016-04-06
#> 6 117.247.211.228 Intel::ADDR 2016-04-06
#> meta.source
#> 1 from http://hosts-file.net/ad_servers.txt via
intel.criticalstack.com
#> 2 from https://www.dan.me.uk/torlist/ via
intel.criticalstack.com
```

```
#> 3 from http://talosintel.com/angler-exposed/ via
intel.criticalstack.com
#> 4 from http://nullsecure.org/threatfeed/master.txt
via intel.criticalstack.com
#> 5 from
http://wget-mirrors.uceprotect.net/rbldnsd-all/dnsbl-1.uceprotect.net.gz
via intel.criticalstack.com
#> 6 from
http://wget-mirrors.uceprotect.net/rbldnsd-all/dnsbl-1.uceprotect.net.gz
via intel.criticalstack.com
summary(historical$Date)
#> Min.   1st Qu.   Median Mean 3rd Qu.
#> "2016-04-06" "2016-04-20" "2016-05-28" "2016-07-04"
"2016-07-14"
#> Max.
#> "2017-10-15"
```

There are 9.3 million unique records on the collected blacklist. Other lists contain much larger samples of addresses. For example, FireHOL IP maintains a running list of 600 million. The list considered here was collected over a 16 month period from April 2016 to October 2017 from a selection of reputable sub-feeds. The fields_type variable specifies which type of indicator and the fields_indicator is the actual value. The counts of types are shown in this table.

```
table(historical$indicator_type)
#>
#> Intel::ADDR Intel::DOMAIN Intel::EMAIL
Intel::FILE_HASH
#> 8774471 291126 1 43934
#> Intel::FILE_NAME Intel::URL
#> 8 248501
```

The majority of the entries are IP addresses. This research will focus on these as a point of reference. The following subset forms the indicator list of interest.

```
ip_historical <- historical %>% filter(indicator_type ==
    "Intel::ADDR")
rm(historical)  #to save some memory!
```

As this blacklist is produced from an aggregator it is useful to see how many sources are used to produce it.

```
sources <- as.data.frame(table(ip_historical$meta.source))
sources <- sources %>% arrange(desc(Freq))
head(sources)
#> Var1
#> 1 from
http://wget-mirrors.uceprotect.net/rbldnsd-all/dnsbl-1.uceprotect.net.gz
via intel.criticalstack.com
#> 2 from http://lists.blocklist.de/lists/all.txt via
intel.criticalstack.com
#> 3 from
http://wget-mirrors.uceprotect.net/rbldnsd-all/ips.backscatterer.org.gz
via intel.criticalstack.com
#> 4 from http://labs.snort.org/feeds/ip-filter.blf
via intel.criticalstack.com
#> 5 from https://www.packetmail.net/iprep.txt via
intel.criticalstack.com
```

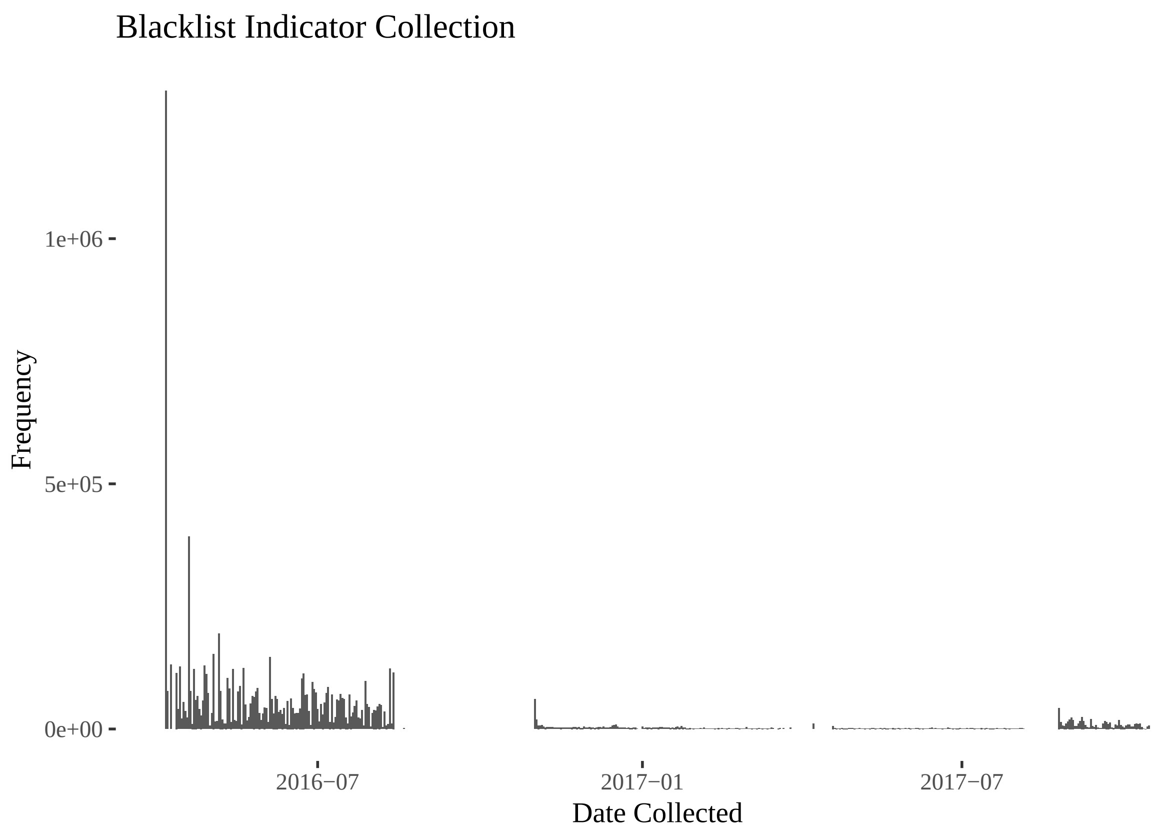
```
#> 6 from http://blocklist.greensnow.co/greensnow.txt
via intel.criticalstack.com
#> Freq
#> 1 6217929
#> 2 906904
#> 3 463645
#> 4 458677
#> 5 152279
#> 6 126941
```

As the frequency list shows the majority of the IP addresses come from uceprotect.new who collects information on IP address that engage in spam-related activity. However, there are also lists contributed from snort (an open sources intrusion detection system) and passive collection systems like packetmail, totaling over 1000 unique sources. With this collection, a broad range of suspicious activities can be used to justify placement of an indicator on the list.

Temporal Analysis

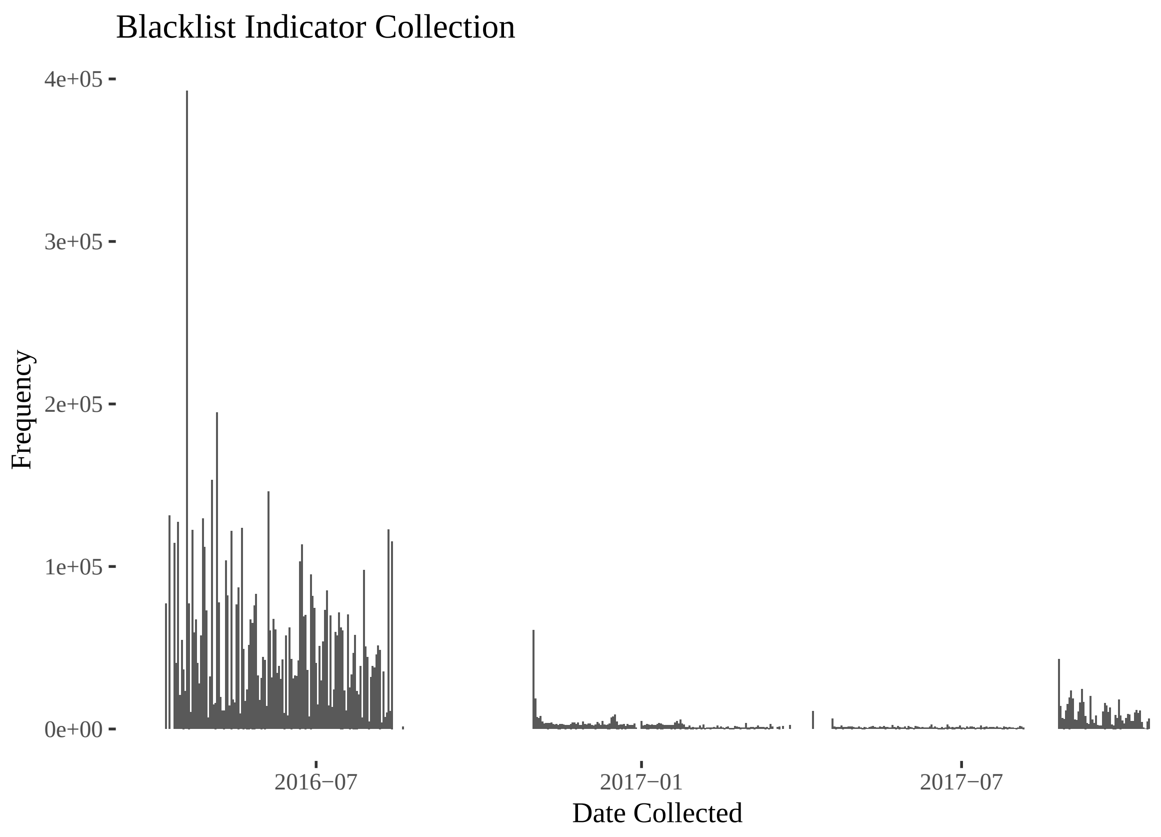
As this list represents a running view into recognized threats and it can be explored for basic insights. One question of interest; What is the pace of unique indicators being added over time?

```r
load("~/analysis/data/historicalBlackListDates.RData")
library(ggthemes)
ggplot(hist.date, aes(x = Var1, y = Freq)) + geom_bar(stat = "identity") +
    ylab("Frequency") + xlab("Date Collected") + ggtitle("Blacklist Indicator Collection") +
    theme_tufte()
```



The first date of collection (April 4th, 2016) gathered the current list at that time. This results in a large peak on the first day meaning that a signifigant majority are historically maintained. With a simple adjustment, the unique add values can be seen more clearly.

```
hist.date <- filter(hist.date, Var1 != "2016-04-06")

ggplot(hist.date, aes(x = Var1, y = Freq)) + geom_bar(stat = "identity") +
    ylab("Frequency") + xlab("Date Collected") + ggtitle("Blacklist Indicator Collection") +
    theme_tufte()
```

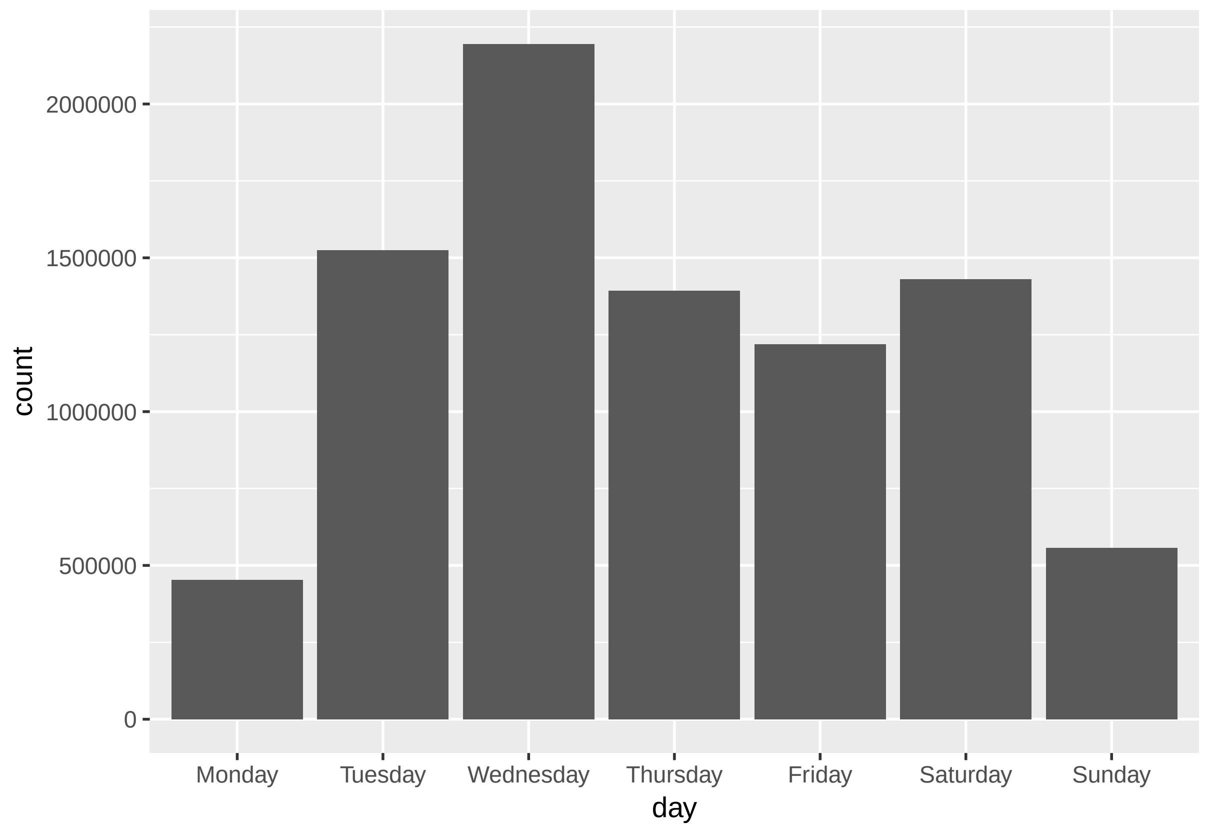


In the first few months of collection, the list added new values consistently. However, as collection went along the new values decreased. This indicates that common threats tend to use and reuse the same or similar indicators over time. However, there is a noticeable increase in late 2017. One other point of interest is that the gaps in plots represent times when low numbers of indicators are collected or the feed aggregator was under maintenance. This represents some uncertainty for when indicators are actually added to the list and might impact the use of the data for analysis. At the very least this points towards some latency in adding indicators.

```r
summary(hist.date$Var1)
#> Min.   1st Qu.   Median Mean 3rd Qu.
#> "2016-04-07" "2016-07-25" "2017-01-28" "2017-01-16"
"2017-06-11"
#> Max.
#> "2017-10-15"
# number of days where no indicators where collected
as.numeric(as.Date("2017-10-15") - as.Date("2016-04-06")) -
    nrow(hist.date)
#> [1] 130
```

Each of the contributing sources update their information on differing intervals.

```r
ip_historical$day <- factor(weekdays(ip_historical$Date),
    levels = c("Monday", "Tuesday", "Wednesday", "Thursday",
        "Friday", "Saturday", "Sunday"))
ggplot(ip_historical, aes(x = day)) + geom_bar()
```



The days in which the indicators are added are done mostly mid-week (Tuesday, Wednesday and Thursday). Both Monday and Sunday are the least frequent and Saturday is relatively common. This is another indication of a set weekly cadence for each of the contributing lists. For a cyber threat actor, a new indicator would have, at least, a week until it makes its way onto this public list.

Geospatial Analysis

Another question of interest is the geolocation of the IP address on the list. Location information can be explored using an IP enriching data set. This research uses the IP2Location provided data set to conduct this comparison.

```
geo_ip <- read.csv("~/analysis/data/IP2LOCATION-LITE-DB1.CSV",
    header = FALSE)
colnames(geo_ip) <- c("begin", "end", "country_code", "country")
head(geo_ip)
rm(geo_ip)
```

This data comes in IP address integer format. However, the blacklist is detailed using the traditional format (i.e. 0.0.0.0). To geocode the rgeoloate package is used comparing the list against the binary formated data provieded by IP2Location.

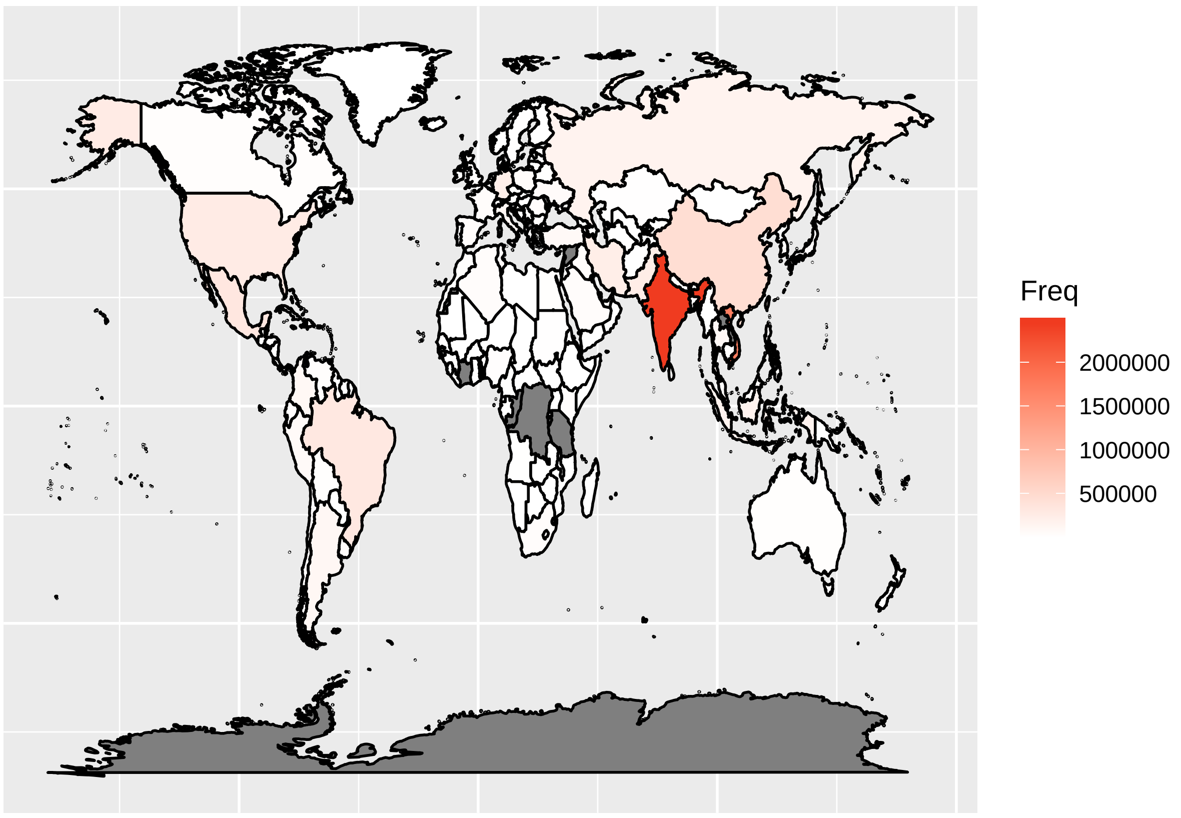
```
# use this function to geolocate but it takes a long
# time so we just load the previously saved results
f <- "~/analysis/data/IP2LOCATION-LITE-DB3.BIN"
temp <- ip2location(ip_historical$fields_indicator, file = f)
```

```r
ip_historical <- cbind(ip_historical, temp)
rm(temp)
```

```r
load("~/analysis/data/ip_historical.RData")

countryFreq <- as.data.frame(table(ip_historical$country_name))
countryFreq <- countryFreq %>% arrange(desc(Freq))
head(countryFreq)
#>            Var1    Freq
#> 1         India 2440614
#> 2      Viet Nam 1704408
#> 3         China  454555
#> 4        Mexico  321406
#> 5        Brazil  311262
#> 6 United States  268789
```

With this adjusted data the frequency of appearance by country is available for analysis.

```r
world_data <- map_data("world")

# some of the important countries are miss labeled!
countryFreq <- countryFreq %>% mutate(Var1 = as.character(Var1)) %>%
    mutate(Var1 = if_else(Var1 == "United States", "USA",
        if_else(Var1 == "United Kingdom", "UK", if_else(Var1 ==
            "Taiwan, Province of China", "Taiwan", if_else(Var1 ==
            "Russian Federation", "Russia", if_else(Var1 ==
            "Iran, Islamic Republic of", "Iran", if_else(Var1 ==
            "Korea, Republic of", "South Korea", if_else(Var1 ==
            "Korea, Democratic People's Republic of", "North Korea",
            if_else(Var1 == "Venezuela, Bolivarian Republic of",
                "Venezuela", if_else(Var1 == "Bolivia, Plurinational State of",
                  "Bolivia", if_else(Var1 == "Viet Nam",
                    "Vietnam", Var1)))))))))))


cyber_map <- merge(world_data, countryFreq, by.x = "region",
    by.y = "Var1", all.x = TRUE)

cyber_map <- cyber_map %>% arrange(group, order)

ggplot(cyber_map, aes(x = long, y = lat, group = group,
    fill = Freq)) + geom_polygon(colour = "black") + scale_fill_gradient2(low = "white",
    high = "#f03b20") + xlab("") + ylab("") + theme(axis.title.x = element_blank(),
    axis.text.x = element_blank(), axis.ticks.x = element_blank(),
    axis.title.y = element_blank(), axis.text.y = element_blank(),
    axis.ticks.y = element_blank())
```



With some discrepancies in the names during the geolocation it is better to use the country codes and an interactive map. The following will produce an interactive map using the googleviz package.

```r
library(googleVis)
op <- options(gvis.plot.tag = "chart")
countryCode <- as.data.frame(table(ip_historical$country_code))
Geo = gvisGeoChart(countryCode, locationvar = "Var1", colorvar = "Freq",
    options = list(projection = "kavrayskiy-vii"))
plot(Geo)
```

The largest contributor to blacklist IP addresses is India and Vietnam (a surprising result). Also, other countries of interest with high frequencies include China, The United States, Iran, Brazil, and Russia. Every country in the world has some number of entries on the list (including North Korea and the Vatican). The quality of the list in aggregating common threats is apparent. However, even with such an extensive list it is not clear that advanced threat actors would be caught.

Advanced Actors

To test the advanced threat hypothesis data from the United States operation GRIZZLY STEPPE is used. Part of this operation is a publicly released set of analysis done in classified settings which uncover malicious Russian activity. Part of the data release includes a set of IP addresses that are reported as belonging to Russian entities. The following is a summary.

```r
russian_cyber <- read_csv(file = "~/analysis/data/JAR-16-20296A.csv")
head(russian_cyber)
#> # A tibble: 6 x 8
#> INDICATOR_VALUE TYPE COMMENT ROLE ATTACK_PHASE
OBSERVED_DATE HANDLING
#> <chr> <chr> <chr> <chr> <chr> <lgl> <chr>
#> 1 efax[.]pfdregi~ URL <NA> URL ~ <NA> NA TLP:WHI~
#> 2 private[.]dire~ FQDN <NA> C2 C2 NA TLP:WHI~
#> 3 www[.]cderlear~ FQDN <NA> C2 C2 NA TLP:WHI~
#> 4 ritsoperrol[.]~ FQDN <NA> <NA> <NA> NA TLP:WHI~
#> 5 littjohnwilhap~ FQDN <NA> <NA> <NA> NA TLP:WHI~
#> 6 wilcarobbe[.]c~ FQDN <NA> <NA> <NA> NA TLP:WHI~
#> # ... with 1 more variable: DESCRIPTION <chr>
paste("There are", nrow(russian_cyber), "known Russian indicators on this list")
#> [1] "There are 911 known Russian indicators on this
list"
```

This data set includes the indicator value and some contextual information about what the indicators represent (i.e., command and control-C2 nodes). Of the 911 indicators the type frequency is broken down as:

```
str(russian_cyber)
#> Classes 'spec_tbl_df', 'tbl_df', 'tbl' and
'data.frame': 911 obs. of 8 variables:
#> $ INDICATOR_VALUE: chr
"efax[.]pfdregistry[.]net/eFax/37486[.]ZIP"
"private[.]directinvesting[.]com"
"www[.]cderlearn[.]com" "ritsoperrol[.]ru" ...
#> $ TYPE : chr "URL" "FQDN" "FQDN" "FQDN" ...
#> $ COMMENT : chr NA NA NA NA ...
#> $ ROLE : chr "URL WATCHLIST" "C2" "C2" NA ...
#> $ ATTACK_PHASE : chr NA "C2" "C2" NA ...
#> $ OBSERVED_DATE : logi NA NA NA NA NA NA ...
#> $ HANDLING : chr "TLP:WHITE" "TLP:WHITE"
"TLP:WHITE" "TLP:WHITE" ...
#> $ DESCRIPTION : chr "It is recommended that network
administrators review traffic to/from the URL address
to determine possible malicious activity." "The Remote
Access Tool malware
\"8F154D23AC2071D7F179959AABA37AD5\" attempts to use
this C2." "The Remote Access Tool malware
\"AE7E3E531494B201FBF6021066DDD188\" attempts to use
this C2." "It is recommended that network
administrators review traffic to/from the domain to
determine possible malicious activity." ...
#> - attr(*, "spec")=
#> .. cols(
#> ..  INDICATOR_VALUE = col_character(),
#> ..  TYPE = col_character(),
#> ..  COMMENT = col_character(),
#> ..  ROLE = col_character(),
#> ..  ATTACK_PHASE = col_character(),
#> ..  OBSERVED_DATE = col_logical(),
#> ..  HANDLING = col_character(),
#> ..  DESCRIPTION = col_character()
#> .. )
table(russian_cyber$TYPE)
#>
```

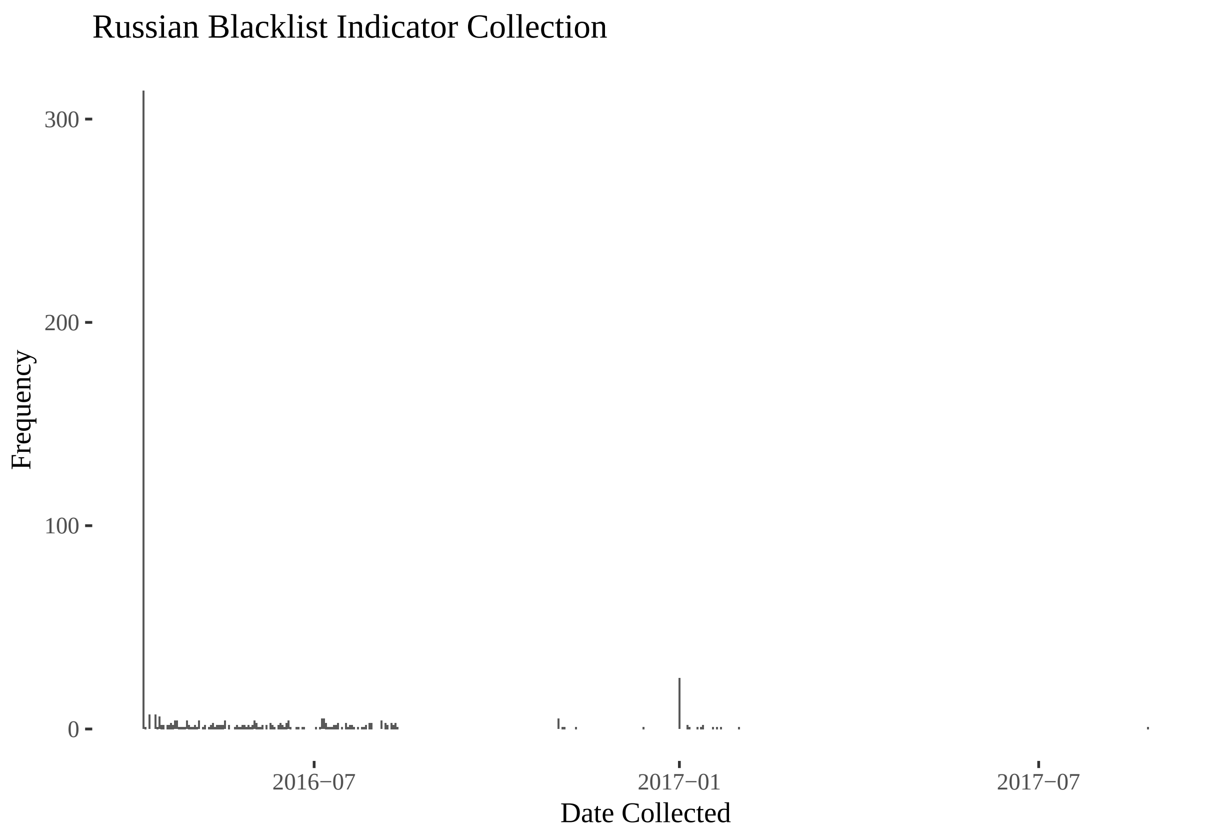
```
#>     FQDN IPV4ADDR       MD5       URL
#>       10      876        24         1
table(russian_cyber$ATTACK_PHASE)
#>
#>  C2
#> 213
table(russian_cyber$ROLE)
#>
#> C2 FILE HASH WATCHLIST IP_WATCHLIST
#> 2 24 876
#> URL WATCHLIST
#> 1
```

For this research, the concern is with how many of the 876 known advanced Russian activity IP addresses are on the blacklist.

```
ip_russian <- russian_cyber %>% filter(TYPE == "IPV4ADDR")
# clean up the brackets
ip_russian$INDICATOR_VALUE <- gsub("\\[", "", ip_russian$INDICATOR_VALUE)
ip_russian$INDICATOR_VALUE <- gsub("\\]", "", ip_russian$INDICATOR_VALUE)

table(ip_historical$fields_indicator %in% ip_russian$INDICATOR_VALUE)
#>
#>   FALSE    TRUE
#> 8773907     564
```

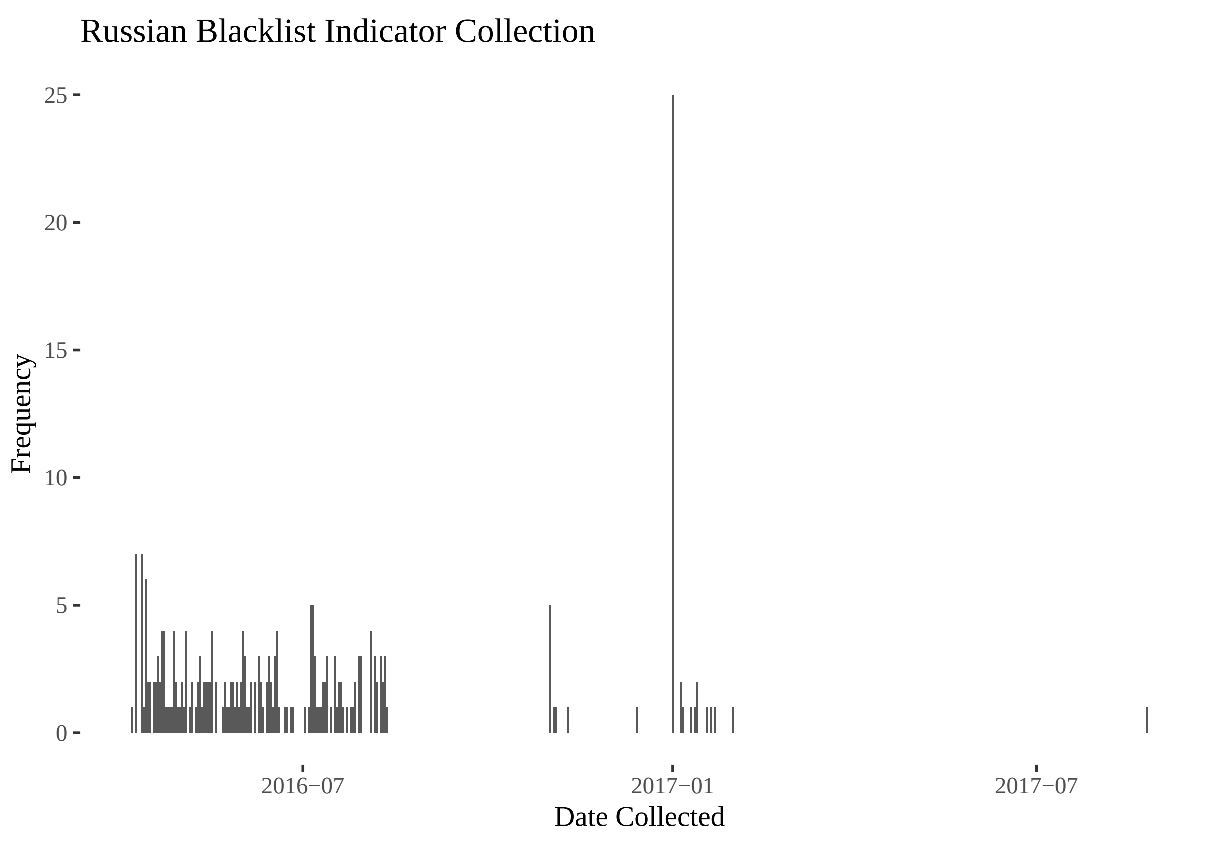
For this publicly collect blacklist `{r} paste0(round(564/876,4) * 100,'%')` percent are present. This is an impressive value as it demonstrates that publicly available information can be assumed to contain information about a significant distribution of actors from spamming criminals to nation states. However, advanced actors may not use the same IP addresses all the time or those from their own country.

```
# subset by russian IPs
russian_blacklist <- ip_historical[which(ip_historical$fields_indicator %in%
    ip_russian$INDICATOR_VALUE), ]
# frequency table
russian_date <- as.data.frame(table(russian_blacklist$Date))

russian_date <- russian_date %>% mutate(Var1 = as.Date(Var1,
    origin = "1970-01-01"))

summary(russian_date$Var1)
#> Min.  1st Qu.  Median Mean 3rd Qu.
#> "2016-04-06" "2016-05-10" "2016-06-13" "2016-07-06"
"2016-07-23"
#> Max.
#> "2017-08-25"
t <- "Russian Blacklist Indicator Collection"
ggplot(russian_date, aes(x = Var1, y = Freq)) + geom_bar(stat = "identity") +
    ylab("Frequency") + xlab("Date Collected") + ggtitle(t) +
    theme_tufte()
```



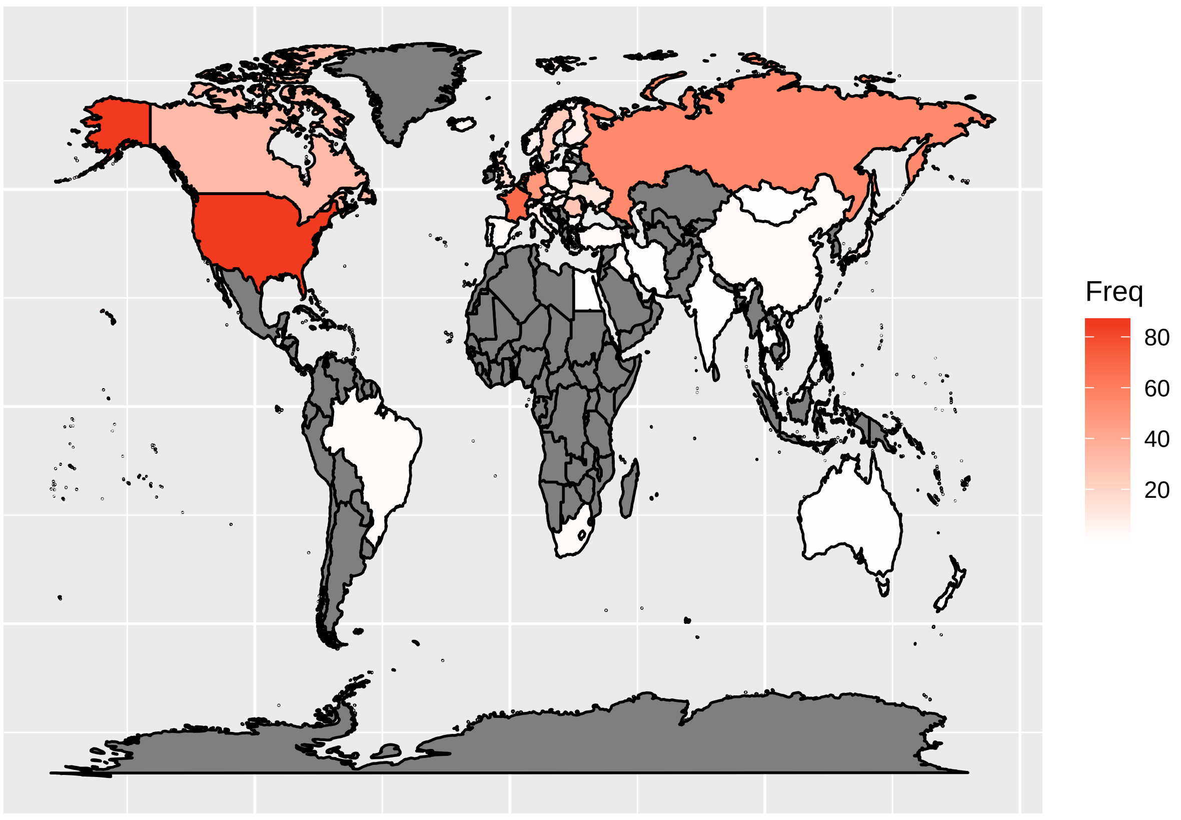
Like the general application, most of the indicators used by the Russians are part of the historical list (over 300). Adjusting for this data gives the frequency of unique additions.

```r
russian_date <- russian_date %>% filter(Var1 != "2016-04-06")
f <- "Russian Blacklist Indicator Collection"
ggplot(russian_date, aes(x = Var1, y = Freq)) + geom_bar(stat = "identity") +
    ylab("Frequency") + xlab("Date Collected") + ggtitle(f) +
    theme_tufte()
```



The date when this list was published was December 2016. The timing of the DNC hack (the event that motivated this GRIZZLY STEPPE'S publishing) was in the spring of the same year. The activity here shows that in the spring and the summer the frequency of which the IP address was added to the blacklist was consistent with the alleged nefarious activity. Shortly after the release the number collected drops to zero (save one outlier). The geolocation for these Russian indicators looks as follows.

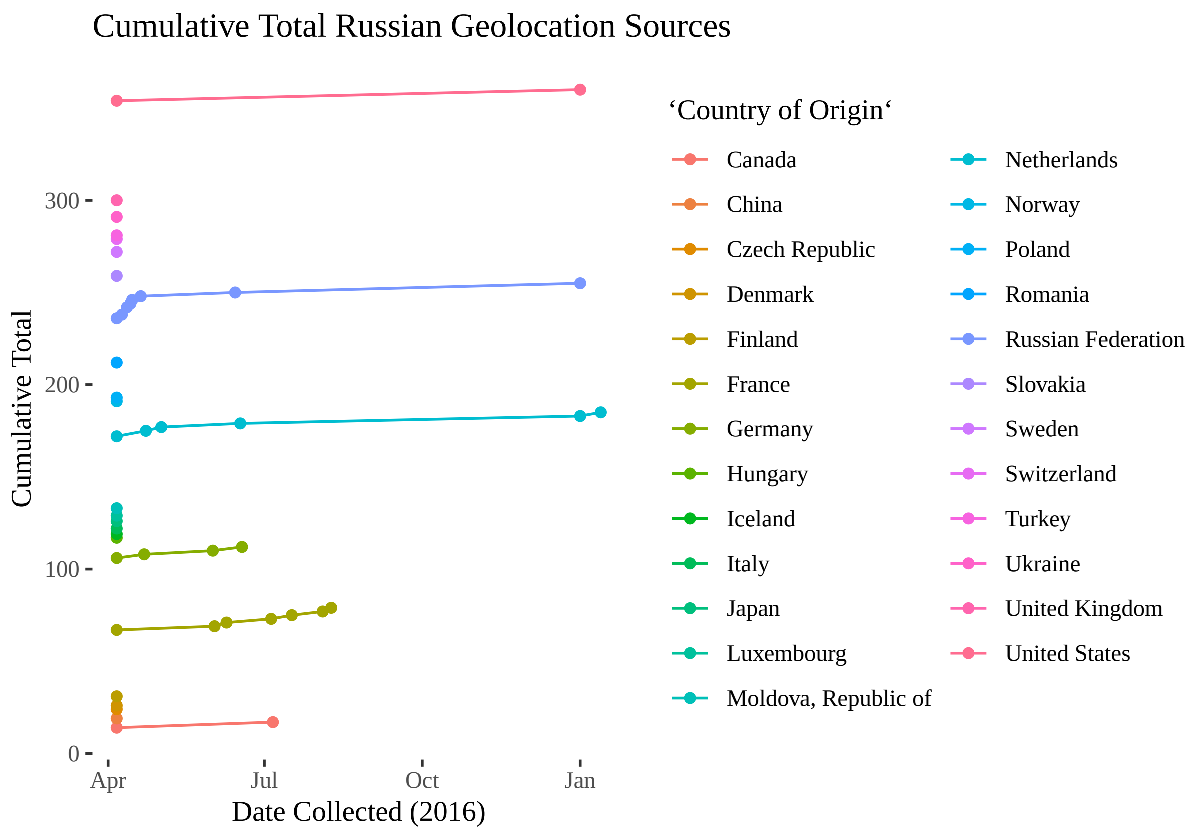
```r
russian_freq <- as.data.frame(table(russian_blacklist$country_name)) %>%
    mutate(Var1 = as.character(Var1)) %>% mutate(Var1 = if_else(Var1 ==
    "United States", "USA", if_else(Var1 == "United Kingdom",
    "UK", if_else(Var1 == "Taiwan, Province of China", "Taiwan",
        if_else(Var1 == "Russian Federation", "Russia",
            if_else(Var1 == "Iran, Islamic Republic of",
                "Iran", if_else(Var1 == "Korea, Republic of",
                  "South Korea", if_else(Var1 == "Korea, Democratic People's Republic of",
                    "North Korea", if_else(Var1 == "Venezuela, Bolivarian Republic of",
                      "Venezuela", if_else(Var1 == "Bolivia, Plurinational State of",
                        "Bolivia", if_else(Var1 == "Viet Nam",
                          "Vietnam", Var1)))))))))))) %>%
    arrange(desc(Freq))

head(russian_freq)
#>          Var1 Freq
#> 1         USA   85
#> 2 Netherlands   74
#> 3      France   68
#> 4      Russia   55
#> 5     Germany   49
#> 6      Canada   32

cyber_map <- merge(world_data, russian_freq, by.x = "region",
    by.y = "Var1", all.x = TRUE)

cyber_map <- cyber_map %>% arrange(group, order)

ggplot(cyber_map, aes(x = long, y = lat, group = group,
    fill = Freq)) + geom_polygon(colour = "black") + scale_fill_gradient2(low = "white",
    high = "#f03b20") + xlab("") + ylab("") + theme(axis.title.x = element_blank(),
    axis.text.x = element_blank(), axis.ticks.x = element_blank(),
    axis.title.y = element_blank(), axis.text.y = element_blank(),
    axis.ticks.y = element_blank())
```



Interestingly of the top 5 countries from with IP addresses are used Russia is included. However, the most significant amount comes from within the United States and Europe (Netherlands, France, and Germany). One final question would be to compare the geospatial data with the temporal data. The following shows the cumulative Russian blacklist IPs by time and by country of origin added to the list.

```
russian_cum <- as.data.frame(table(russian_blacklist$Date,
    russian_blacklist$country_name)) %>% filter(Freq > 1) %>%
    mutate(Var1 = as.Date(Var1))

colnames(russian_cum) <- c("Date", "Country of Origin",
    "Freq")
```

```r
ggplot(data = russian_cum, aes(x = Date, y = cumsum(Freq),
    colour = `Country of Origin`)) + geom_line() + geom_point() +
    ylab("Cumulative Total") + xlab("Date Collected (2016)") +
    ggtitle("Cumulative Total Russian Geolocation Sources") +
    theme_tufte()
```



This additional view shows that while the largest amount of IP addresses originates from the United States. Most of the new additions during the time of the DNC hack are from Russia. This may indicate some expediency on the part of the Russian actors to exploit a known vulnerability using less covert final hop IP addresses.

Conclusion

The blacklist here, while not canonical, is useful for understanding the current state of cybersecurity by indicator. There are a number of useful metrics that can be gleaned from this exploration.

- A broad range of actors are represented
- The list is representative of current practice
- Most indicators are historically bad and a relative few are added periodically

For this research, the blacklist explored here will be used to classify IP addresses for use in supervised learning processes.

Appendix F

Exploratory Data Analysis of Collected Honeypot Data

Exploratory Data Analysis of Collected Honeypot Data

Isaac J. Faber

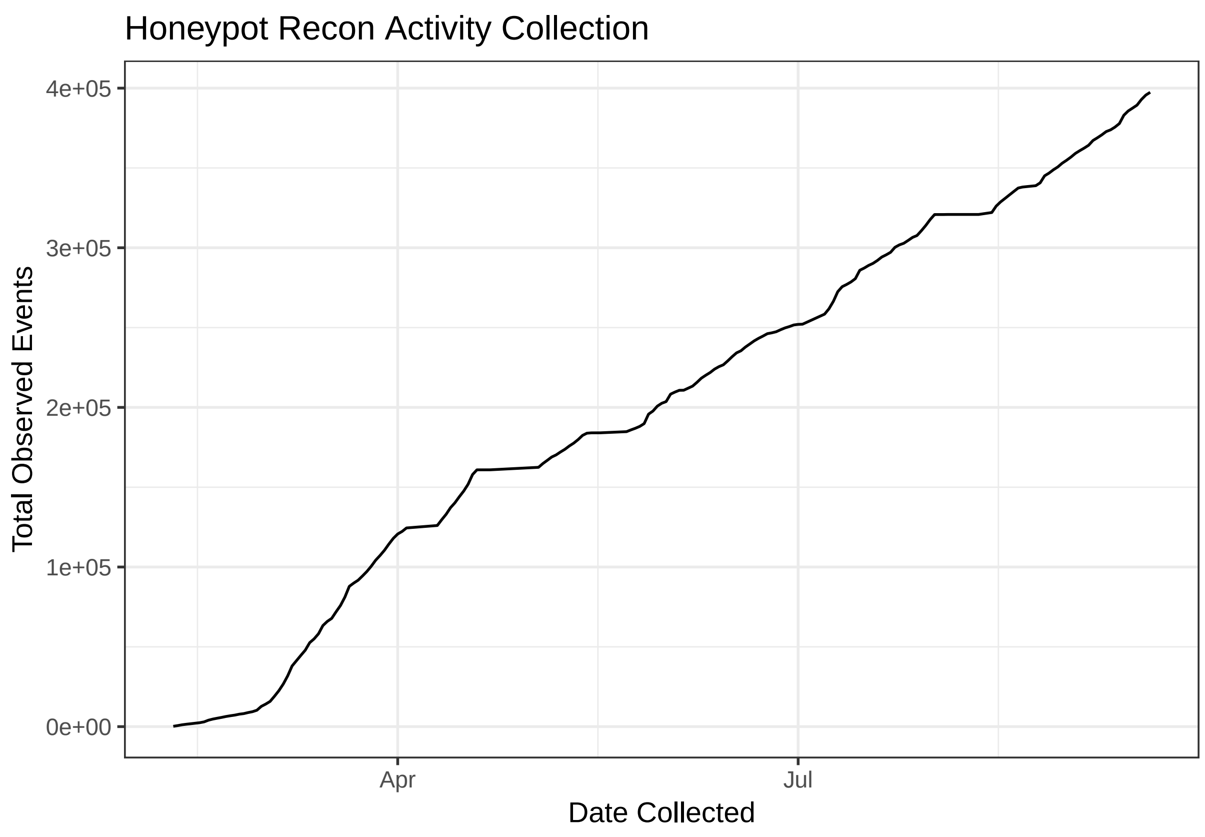
Overview

For this research a set of 8 real-world honeypots where deployed as public facing servers on three different cloud infrastructure providers, AWS, Digital Ocean, and Azure. These honeypots where completely passive and only collected data on uninitiated connection attempts. The honeypots where deployed using the Modern Honey Network - MHN. Each honeypot had two clients installed to collect data; snort and cowrie.

Snort is a popular intrusion detection system that is useful for collecting packet and session level information from specific network ports. The snort rules used for collection are default values provided by the MHN. Cowrie is a medium interaction honeypot that is useful for collecting specific types of activity. For example, saving the username and password that are being attempted. None of the honeypots had custom DNS beyond what was provided by default from the cloud provider. The cowrie program only monitored activity on port 22 (the admin port).

Because these honeypots are passive the data collected from them can be considered a proxy for the general background noise of the internet. The purpose of this research is to extract behavioral information from various actor types; legitimate and threat. This raw data contains behavioral information that is then categorized using the collected blacklist data discussed in another Annex. The actual deployment timeline of the systems was staggered over a few months. The following sections explore the data collected during this effort.

```r
library(tidyverse)
library(ggthemes)
# raw honeypot data
load("~/analysis/data/honey.RData")

# number of records collected
print(nrow(honeypot.data))
#> [1] 665140

# a modified verison of data for basic temporal
# evaluation
load("~/analysis/data/honeyDate.RData")

# add a feature for the known russian IP addresses
russian_cyber <- read_csv(file = "~/analysis/data/JAR-16-20296A.csv")
ip_russian <- russian_cyber %>% filter(TYPE == "IPV4ADDR")
rm(russian_cyber)

ip_russian$INDICATOR_VALUE <- gsub("\\[", "", ip_russian$INDICATOR_VALUE)
ip_russian$INDICATOR_VALUE <- gsub("\\]", "", ip_russian$INDICATOR_VALUE)

honeypot.data$russian_actor <- (honeypot.data$source_ip %in%
    ip_russian$INDICATOR_VALUE)
rm(ip_russian)

ggplot(honey.date, aes(x = Var1, y = cumsum(Freq))) + geom_line(stat = "identity") +
    ylab("Total Observed Events") + xlab("Date Collected") +
    ggtitle("Honeypot Recon Activity Collection") + theme_bw()
```



```
summary(honey.date$Var1)
#> Min.   1st Qu.   Median Mean 3rd Qu.
#> "2017-02-09" "2017-03-27" "2017-06-05" "2017-05-31"
"2017-07-26"
#> Max.
#> "2017-09-19"

rm(honey.date)

# convert to R dates and save as character string
honeypot.data$timestampProper <- as.POSIXct(as.numeric(as.character(honeypot.data$timestamp)),
    origin = "1970-01-01")

honeypot.data$timestampChar <- as.character(honeypot.data$timestampProper)
honeypot.data$Date_actual <- as.Date(honeypot.data$timestampProper)

# difference between actual date observed and the date
# it was seen on the blacklist

honeypot.data$blacklist_diff <- honeypot.data$Date_actual -
    honeypot.data$Date

# extact hours from the string and save as scaled
honeypot.data$hour <- as.numeric(substr(honeypot.data$timestampChar,
    12, 13))
honeypot.data$hourScaled <- scale(honeypot.data$hour)
```

There are 660 thousand events recorded in the collected data. The information collected has a number of useful features for analysis. The collection of these events is more consistent than the blacklist. There are gaps in the collected information owing to maintenance of the honeypot collection system. This data was collected from February 2017 to September 2017.

```
# structure of dataframe
str(honeypot.data)
#> 'data.frame': 665140 obs. of 74 variables:
#> $ source_ip : Factor w/ 40450 levels
"218.92.236.53",..: 1 2 3 3 3 4 4 4 4 4 ...
#> $ X_id..oid : Factor w/ 660088 levels
"589bad9111c2547609dd0b02",..: 1 2 179511 2049 3 30249
1970 95592 15384 3934 ...
#> $ destination_ip : Factor w/ 8 levels
"172.31.14.41",..: 1 1 8 4 1 1 2 1 2 2 ...
#> $ protocol : Factor w/ 4 levels
"TCP","UDP","ICMP",..: 1 2 1 1 1 1 1 1 1 1 ...
#> $ hpfeed_id..oid : Factor w/ 660069 levels
"589bad8411c2547609dd0b01",..: 1 2 179492 2033 3 30233
1954 95573 15368 3918 ...
#> $ timestamp : Factor w/ 660069 levels
"1486597507.89",..: 1 2 179492 2033 3 30233 1954 95573
15368 3918 ...
#> $ snort.priority : Factor w/ 3 levels "2","3","1":
1 1 1 1 1 1 1 1 1 1 ...
#> $ snort.header : Factor w/ 272 levels
"1:2010935:2",..: 1 2 1 1 1 4 4 4 4 4 ...
#> $ snort.classification : Factor w/ 9 levels
"3","4","30","14",..: 1 2 1 1 1 3 3 3 3 3 ...
#> $ snort.signature : Factor w/ 271 levels "ET POLICY
Suspicious inbound to MSSQL port 1433",..: 1 2 1 1 1 4
4 4 4 4 ...
#> $ source_port : Factor w/ 59895 levels
"4663","5300",..: 1 2 10578 1580 3 3965 13 13 1973
2873 ...
#> $ honeypot : Factor w/ 2 levels "snort","cowrie": 1
1 1 1 1 1 1 1 1 1 ...
#> $ identifier : Factor w/ 16 levels
"b9caa5cc-ee55-11e6-aedd-5e6cb417f095",..: 1 1 15 4 1
1 2 1 2 2 ...
#> $ sensor : Factor w/ 8 levels
"b9caa5cc-ee55-11e6-aedd-5e6cb417f095",..: 1 1 8 4 1 1
2 1 2 2 ...
#> $ destination_port : Factor w/ 473 levels
"1433","5060",..: 1 2 1 1 1 286 126 87 95 168 ...
#> $ session_ssh.version : Factor w/ 189 levels
"SSH-2.0-libssh2_1.4.3",..: NA NA NA NA NA NA NA NA NA
NA ...
#> $ auth_attempts.login : Factor w/ 14636 levels
"root","admin",..: NA NA NA NA NA NA NA NA NA NA ...
#> $ auth_attempts.password : Factor w/ 25103 levels
"1234567","dreambox",..: NA NA NA NA NA NA NA NA NA NA
...
```

```
#> $ auth_attempts.login1 : Factor w/ 2453 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login2 : Factor w/ 2433 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login3 : Factor w/ 1432 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login4 : Factor w/ 150 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login5 : Factor w/ 146 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login6 : Factor w/ 143 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login7 : Factor w/ 141 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login8 : Factor w/ 141 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login9 : Factor w/ 141 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login10 : Factor w/ 141 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login11 : Factor w/ 141 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login12 : Factor w/ 141 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login13 : Factor w/ 141 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login14 : Factor w/ 141 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login15 : Factor w/ 141 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login16 : Factor w/ 141 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login17 : Factor w/ 141 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login18 : Factor w/ 141 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
```

```
...
#> $ auth_attempts.login19 : Factor w/ 140 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login20 : Factor w/ 141 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.login21 : Factor w/ 134 levels
"admin","usuario",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.password1 : Factor w/ 3679 levels
"1234","1111",..: NA NA NA NA NA NA NA NA NA NA ...
#> $ auth_attempts.password2 : Factor w/ 3712 levels
"12345","111111",..: NA NA NA NA NA NA NA NA NA NA ...
#> $ auth_attempts.password3 : Factor w/ 2199 levels
"admin123","123123",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.password4 : Factor w/ 493 levels
"1234","123321",..: NA NA NA NA NA NA NA NA NA NA ...
#> $ auth_attempts.password5 : Factor w/ 497 levels
"admin1","1234",..: NA NA NA NA NA NA NA NA NA NA ...
#> $ auth_attempts.password6 : Factor w/ 377 levels
"admin1234","admin",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.password7 : Factor w/ 372 levels
"admin","password",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.password8 : Factor w/ 377 levels
"admin1","admin",..: NA NA NA NA NA NA NA NA NA NA ...
#> $ auth_attempts.password9 : Factor w/ 368 levels
"admin123","1234",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.password10: Factor w/ 373 levels
"admin123","12345",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.password11: Factor w/ 361 levels
"admin","admin1",..: NA NA NA NA NA NA NA NA NA NA ...
#> $ auth_attempts.password12: Factor w/ 368 levels
"admin1234","1234",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.password13: Factor w/ 362 levels
"1234","12345",..: NA NA NA NA NA NA NA NA NA NA ...
#> $ auth_attempts.password14: Factor w/ 371 levels
"admin","password",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.password15: Factor w/ 370 levels
"12345","admin1234",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.password16: Factor w/ 361 levels
"admin1234","1234",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.password17: Factor w/ 357 levels
"admin123","admin1234",..: NA NA NA NA NA NA NA NA NA
NA ...
```

```
#> $ auth_attempts.password18: Factor w/ 355 levels
"admin","admin123",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ auth_attempts.password19: Factor w/ 350 levels
"admin123","password",..: NA NA NA NA NA NA NA NA NA
NA ...
#> $ auth_attempts.password20: Factor w/ 349 levels
"admin","1234",..: NA NA NA NA NA NA NA NA NA NA ...
#> $ auth_attempts.password21: Factor w/ 350 levels
"12345","password",..: NA NA NA NA NA NA NA NA NA NA
...
#> $ threat : logi TRUE FALSE TRUE TRUE TRUE TRUE ...
#> $ russian_actor : logi FALSE FALSE FALSE FALSE
FALSE FALSE ...
#> $ timestampProper : POSIXct, format: "2017-02-08
15:45:07" "2017-02-08 15:59:02" ...
#> $ timestampChar : chr "2017-02-08 15:45:07"
"2017-02-08 15:59:02" "2017-05-13 00:58:53"
"2017-02-17 09:59:49" ...
#> $ Date_actual : Date, format: "2017-02-08"
"2017-02-08" ...
#> $ blacklist_diff : 'difftime' num 265 NA 138 53 ...
#> ..- attr(*, "units")= chr "days"
#> $ hour : num 15 15 0 9 16 10 4 12 18 21 ...
#> $ hourScaled : num [1:665140, 1] 0.513 0.513 -1.643
-0.35 0.656 ...
#> ..- attr(*, "scaled:center")= num 11.4
#> ..- attr(*, "scaled:scale")= num 6.96
#> $ indicator_type : chr "Intel::ADDR" NA
"Intel::ADDR" "Intel::ADDR" ...
#> $ Date : Date, format: "2016-05-19" NA ...
#> $ meta.source : chr "from
http://www.binarydefense.com/banlist.txt via
intel.criticalstack.com" NA "from
http://blocklist.greensnow.co/greensnow.txt via
intel.criticalstack.com" "from
http://blocklist.greensnow.co/greensnow.txt via
intel.criticalstack.com" ...
#> $ country_code : chr "CN" NA "CN" "CN" ...
#> $ country_name : chr "China" NA "China" "China" ...
#> $ day : chr "Thursday" NA "Monday" "Monday" ...

# breakdown of event by honeypot program and threat (if
# the source_ip is on the blacklist)
table(honeypot.data$honeypot, honeypot.data$threat)
#>
#>           FALSE   TRUE
#>   snort   13062  24141
#>   cowrie 320107 307830
```

The number of events collected by cowrie greatly outnumber those gathered by snort. This is an essential feature of the data set; in a practical setting any actor that attempts to log in to the port which cowrie is monitoring (22) can reasonably be considered a threat even if they are not on a blacklist. From a behavioral perspective, this is a causal statement that allows for the modification of a security policy with no further analysis.

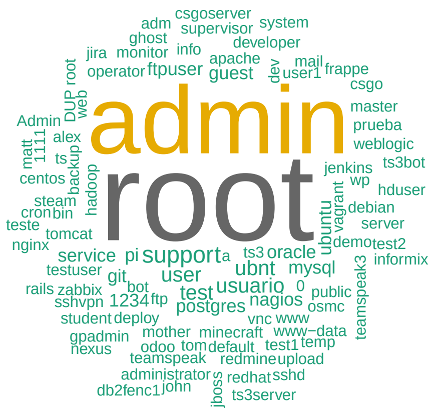
Interestingly, only half of the cowrie source IP address where contained on the collected blacklist. The blacklist collection stopped at a similar time as the on the honeypots, so it is likely that many of these addresses would eventually be placed on a blacklist with a continued collection.

One interesting behavior data point that is collected by cowrie is the username and passwords attempted.

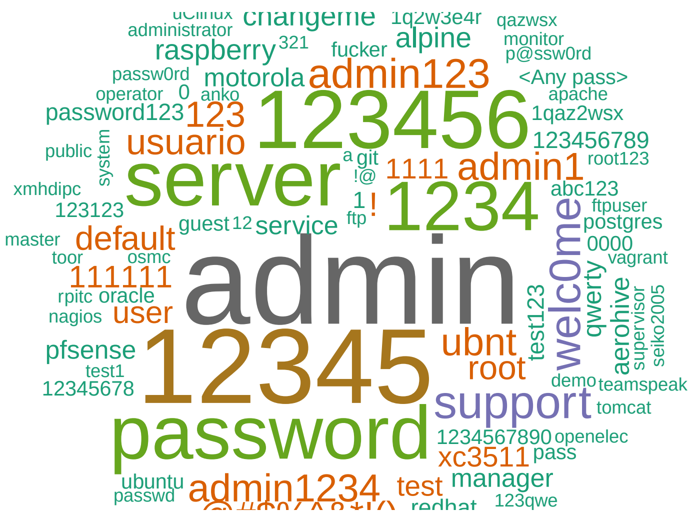
```r
login_temp <- honeypot.data[, grep("login", colnames(honeypot.data))]
password_temp <- honeypot.data[, grep("password", colnames(honeypot.data))]

login_values <- as.character(login_temp[, 1])
password_values <- as.character(password_temp[, 1])

for (i in 2:ncol(login_temp)) {
    login_values <- c(login_values, as.character(login_temp[,
        i]))
    password_values <- c(password_values, as.character(password_temp[,
        i]))
    login_values <- na.omit(login_values)
    password_values <- na.omit(password_values)
}
rm(login_temp)
rm(password_temp)
# number of unique usernames attempted
length(unique(login_values))
#> [1] 14993
# number of unique passwords attempted
length(unique(password_values))
#> [1] 27054
```

There is a large number of both unique usernames and passwords attempted. A simple word cloud can help visualize these values based on frequency.

```r
library(wordcloud)
#> Loading required package: RColorBrewer

un_words <- as.data.frame(table(login_values))
pw_words <- as.data.frame(table(password_values))

wordcloud(un_words$login_values, un_words$Freq, random.order = FALSE,
    max.words = 100, min.freq = 10, colors = brewer.pal(8,
        "Dark2"))
```



```r
wordcloud(pw_words$password_values, pw_words$Freq, random.order = FALSE,
    max.words = 100, min.freq = 10, colors = brewer.pal(8,
        "Dark2"))
```



```r
rm(un_words, pw_words)
```

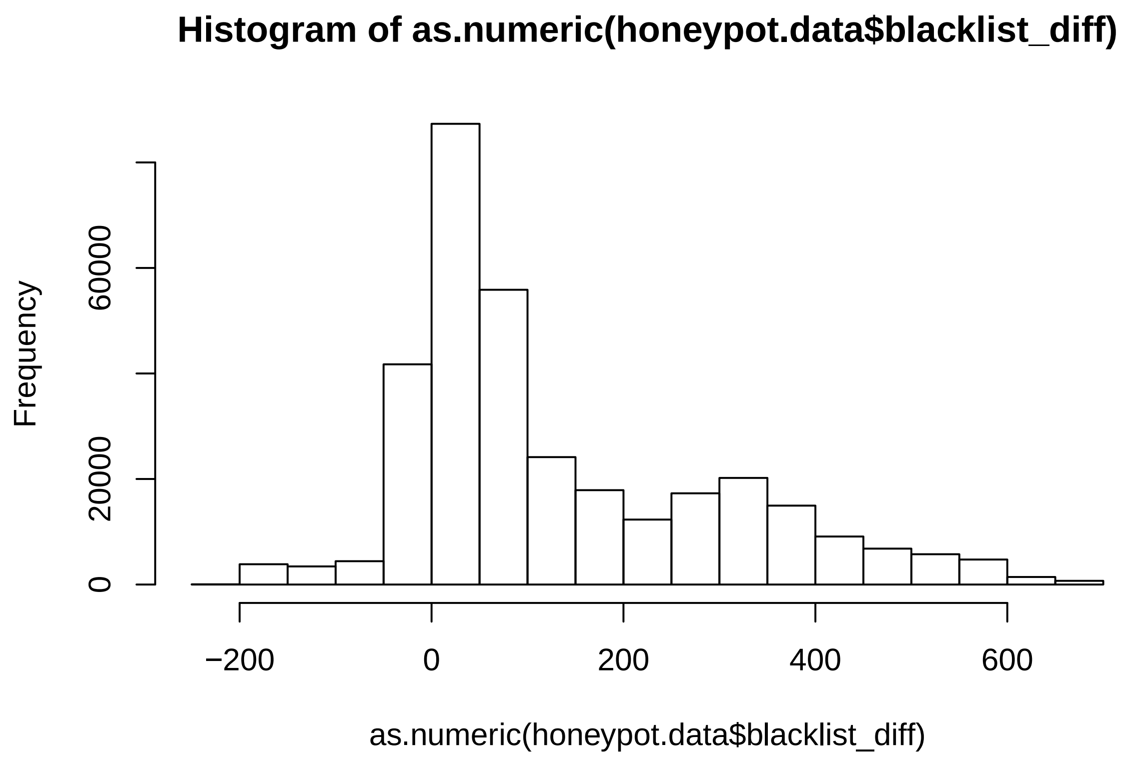
The word clouds show that a basic set of authentication attempts are reused quite often. However, more sophisticated combinations are present as well. Many of the combinations are looking for publicly facing 'Internet of Things' devices (like raspberry pi).

From here using both the snort and cowrie data (even assuming that cowrie data can be handled using heuristics), some basic behavioral patterns can be explored.

Temporal Analysis

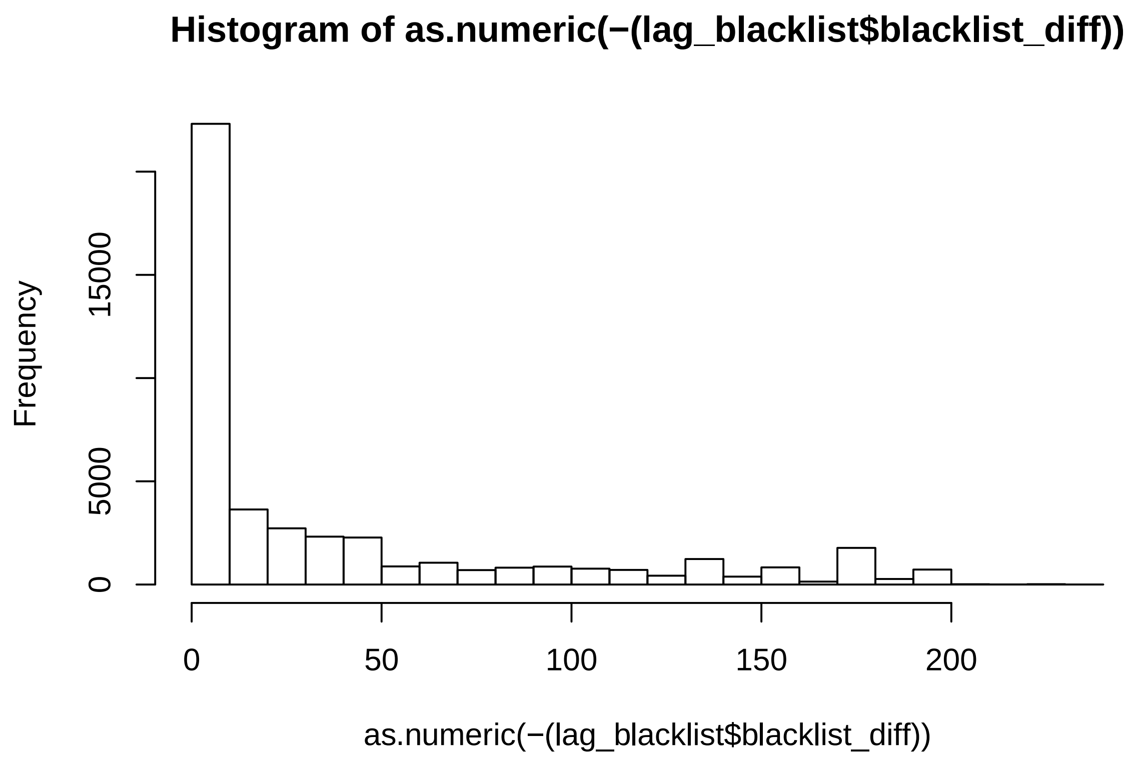
There are two primary dates and times contained in the dataset; date of honeypot observation and date placed on the blacklist. The variable blacklist.diff is the difference between these two dates.

```r
hist(as.numeric(honeypot.data$blacklist_diff))
```



Most of the blacklisted IP address seen by the honeypots were on the list well before being observed. Many of these blacklisted addresses where identified as much as two years before being collected by the honeypot. This indicates that some threat actors reuse the same indicators without consideration of consequences. However, a smaller amount was placed on the list after being observed.

```
lag_blacklist <- honeypot.data %>% filter(blacklist_diff <
    0)
# number seen by the honey pot before being placed on
# the blacklist
nrow(lag_blacklist)
#> [1] 44889

# breakdown of these indicators by honeypot type
table(lag_blacklist$honeypot)
#>
#>  snort cowrie
#>   5613  39276

hist(as.numeric(-(lag_blacklist$blacklist_diff)))
```



As we can see from the histogram the number of lag days until placed on a blacklist was quite short (typically done within the week or month). However, some smaller portion of the times it took several months.

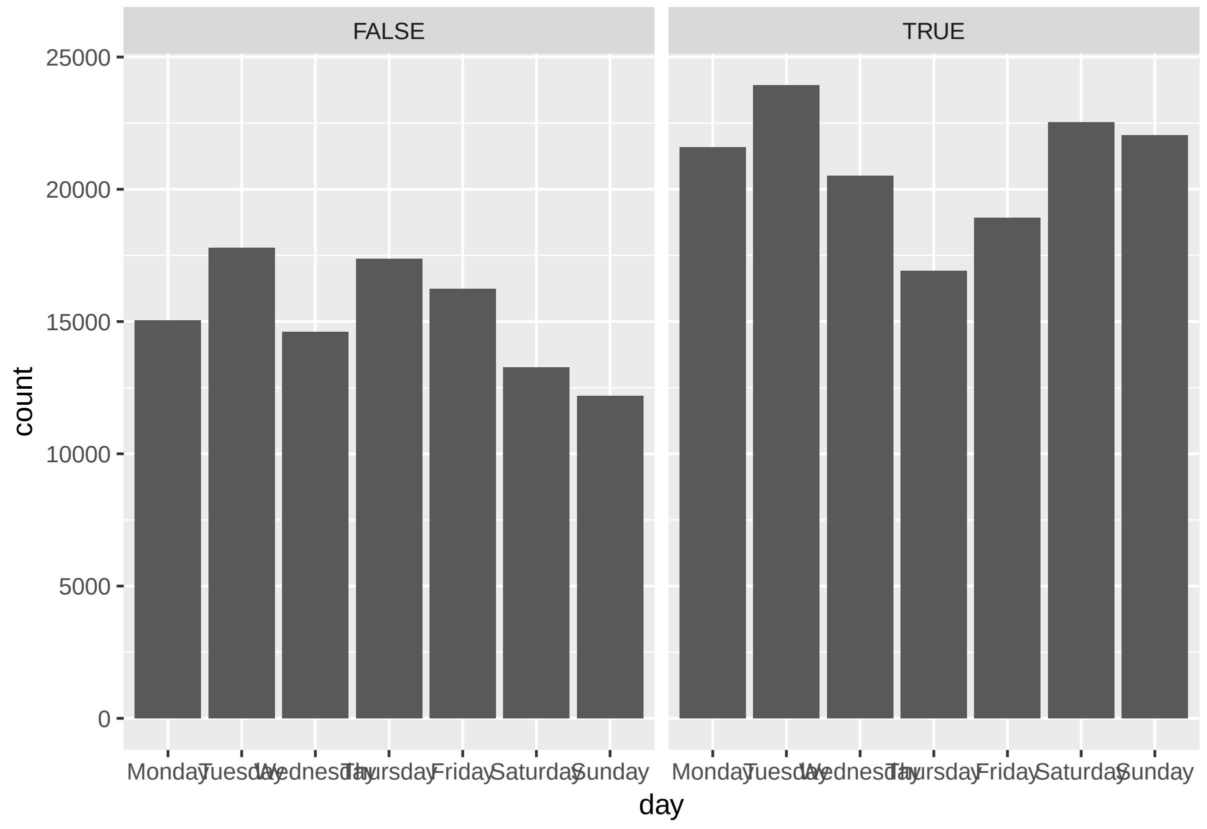
Earlier it was discussed that some of the unlabeled events might eventually be placed on a blacklist. This is evidence that data collected from the last few months by the honeypot may not have had enough time to be placed on such list. So for machine learning purposes, the last few months should be withheld or a significant bias could be introduced and could negatively impact performance.

```
honeypot.data_date_trim <- honeypot.data %>% filter(Date_actual <
    "2017-07-01")
rm(honeypot.data)

table(honeypot.data_date_trim$russian_actor, honeypot.data_date_trim$honeypot)
#>
#>          snort cowrie
#>   FALSE  35345 217493
#>   TRUE      50     68
```

This adjustment has reduced (but not removed) the issues concerned with unlabeled blacklist threats.

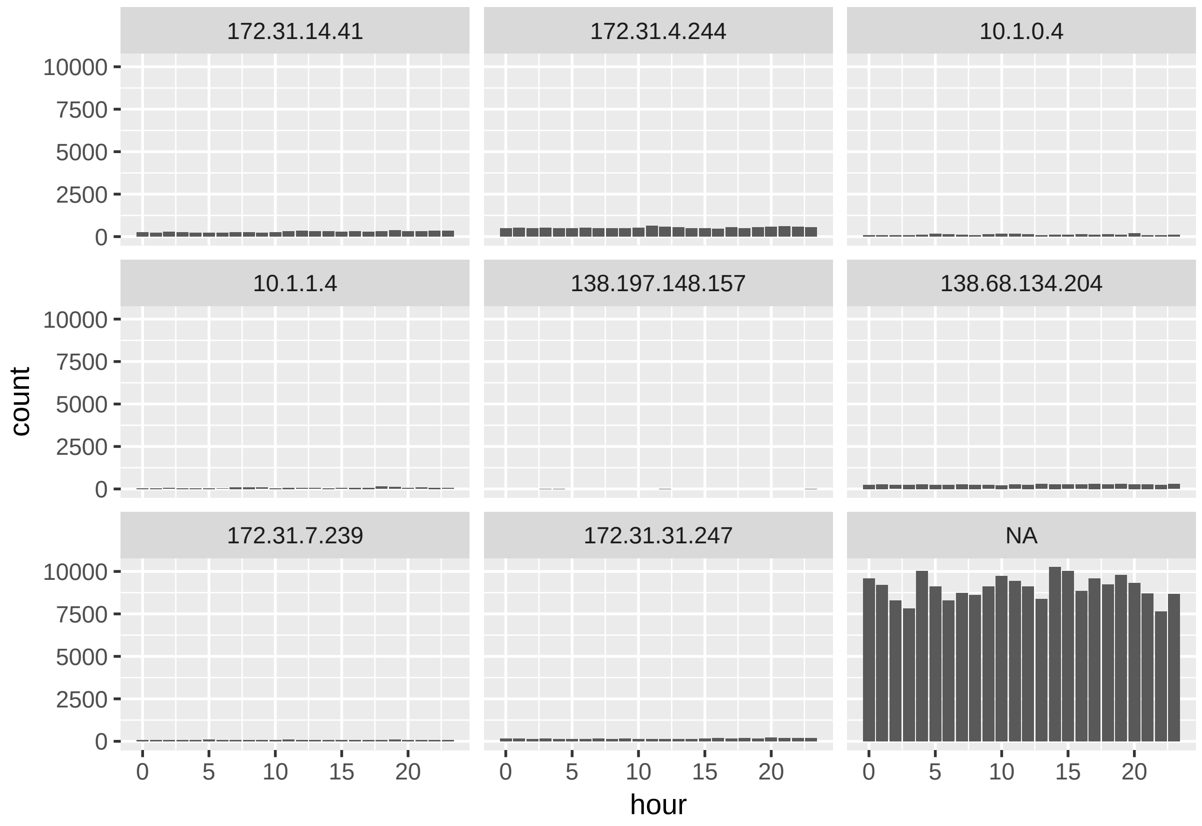
Another interesting question about honeypot activity during specific days can be explored.

```
honeypot.data_date_trim$day <- factor(weekdays(honeypot.data_date_trim$Date_actual),
    levels = c("Monday", "Tuesday", "Wednesday", "Thursday",
        "Friday", "Saturday", "Sunday"))

ggplot(honeypot.data_date_trim, aes(x = day)) + geom_bar() +
    facet_wrap(. ~ threat)
```



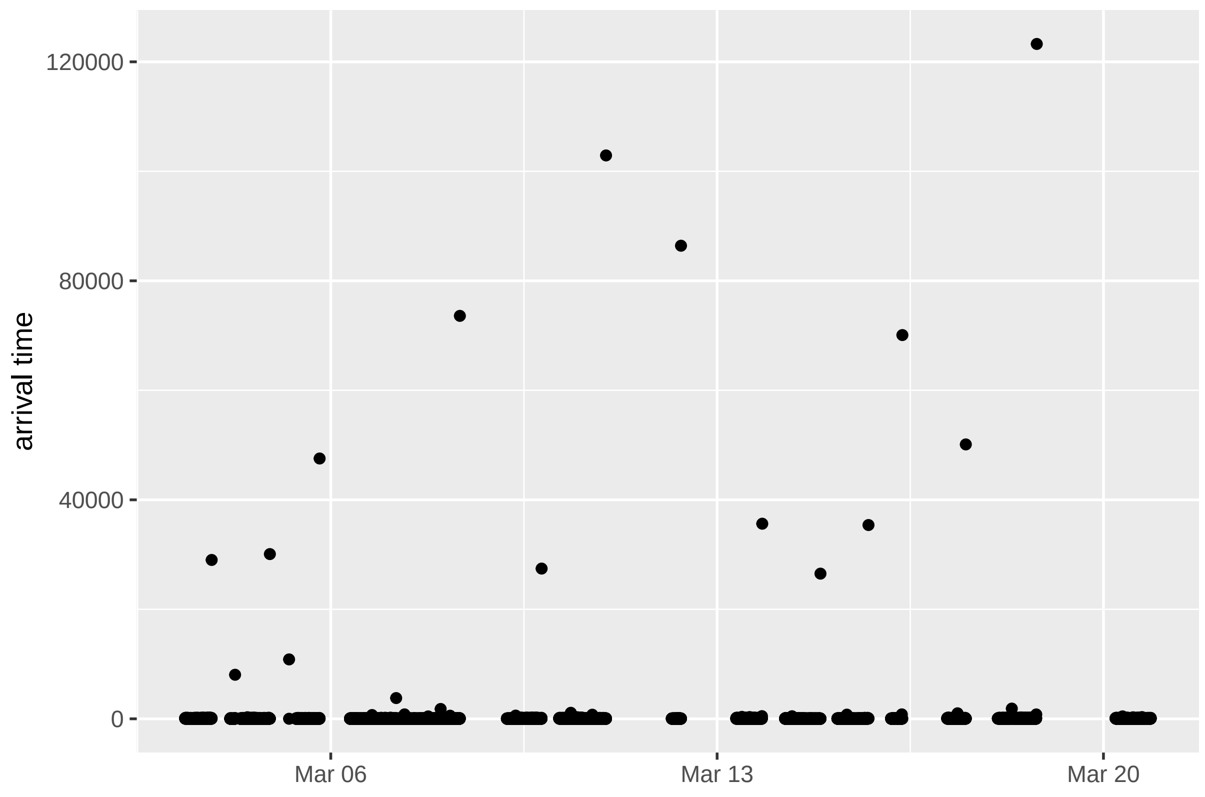
Across all actors, there does not seem to be much of an impact on which day of the week is used. However, those events categorized as threats seem to be more active in the early weekdays compared to the non-threats. We can do the same thing with an hour of the day.

```
ggplot(honeypot.data_date_trim, aes(x = hour)) + geom_bar() +
    facet_wrap(. ~ destination_ip)
```

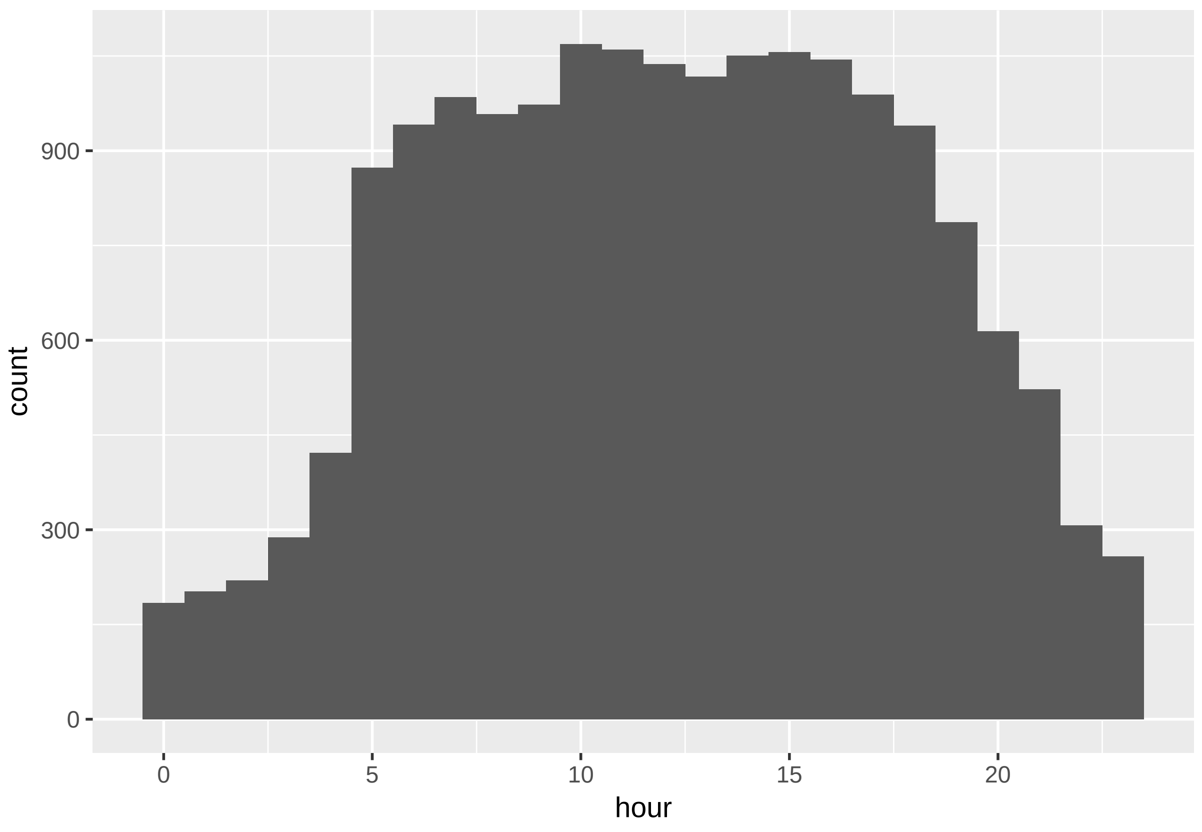


Each of the honeypots has little specific targeting time in the hour of the day across all actors. In general, this is the case with all values. As each threat or activity accrues at each honeypot distribution value tends to uniform. However, for individual IP addresses, this is not always the case.

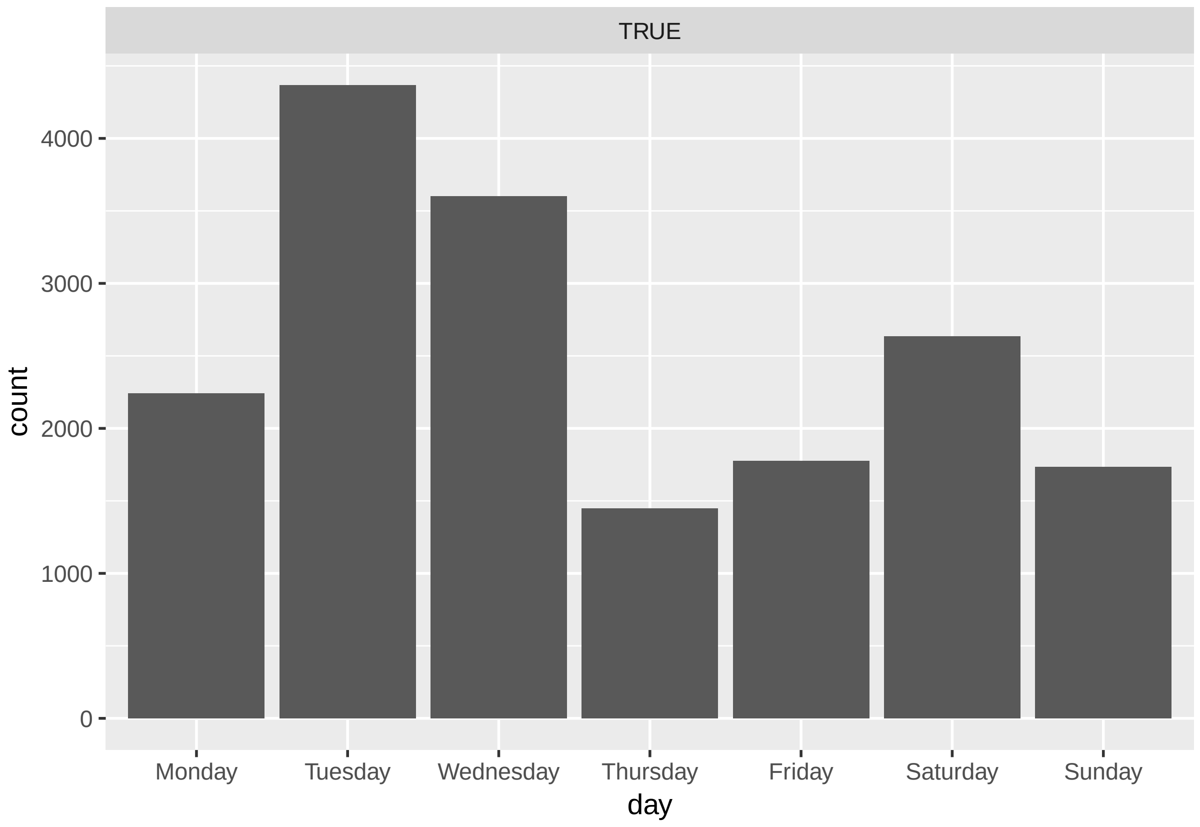
```r
# this is a very active blacklisted IP address from
# China
china_ip <- honeypot.data_date_trim %>% filter(source_ip ==
    "122.194.229.7") %>% arrange((timestampProper)) %>%
    mutate(timediff = as.numeric(c(diff(timestampProper),
        0))) %>% filter(timediff > 0)

# interarrival time
ggplot(china_ip, aes(x = timestampProper, y = timediff)) +
    geom_point() + xlab("") + ylab("arrival time")
```



```r
# hourly time
ggplot(china_ip, aes(x = hour)) + geom_histogram(bins = 24)
```



```
# day of week
ggplot(china_ip, aes(x = day)) + geom_bar() + facet_wrap(. ~
    threat)
```



With this individual IP address from China, we can see a distinct pattern in both times of day (middle of the workday) and day of the week (more active in the early week). The first plot shows that the interarrival time between events also contains a distinct pattern. This implies that distinct features patterns might not be systemically apparent but they are with individual IP addresses. The presence of distinct feature behavior with unique IP addresses will be used later during the supervised learning process.

Geospatial Analysis

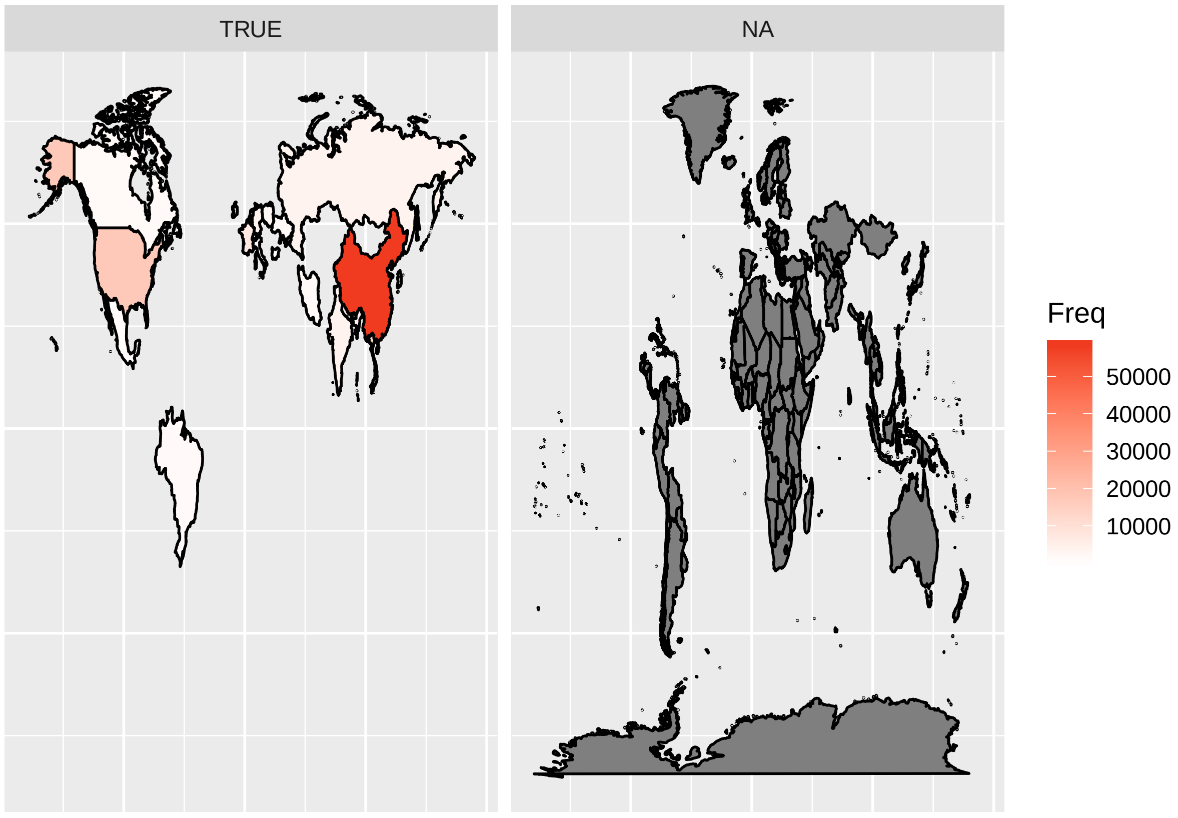
The temporal aspect of individual IP addresses contains some identifying information. Also, geographic information can be explored similarly. The IP addresses need to be geolocated first.

```r
library(rgeolocate)
f <- "~/analysis/data/IP2LOCATION-LITE-DB5.BIN"
temp <- ip2location(honeypot.data_date_trim$source_ip, file = f,
    fields = c("region", "city", "lat", "long"))

honeypot.data_date_trim <- cbind(honeypot.data_date_trim,
    temp)
rm(temp)

countryFreq <- as.data.frame(table(honeypot.data_date_trim$country_name,
    honeypot.data_date_trim$threat))

countryFreq <- countryFreq %>% arrange(desc(Var2, Freq)) %>%
    filter(Freq > 1000)
```

```
colnames(countryFreq) <- c("Country", "Threat", "Freq")
countryFreq
#>                       Country Threat  Freq
#> 1                      Brazil   TRUE  1497
#> 2                    Bulgaria   TRUE  1062
#> 3                      Canada   TRUE  1785
#> 4                       China   TRUE 58184
#> 5                      France   TRUE  7209
#> 6                     Germany   TRUE  4016
#> 7                       India   TRUE  3947
#> 8  Iran, Islamic Republic of   TRUE  1724
#> 9                     Ireland   TRUE  8565
#> 10                      Italy   TRUE  1683
#> 11         Korea, Republic of   TRUE  1983
#> 12                     Mexico   TRUE  1025
#> 13                Netherlands   TRUE  7106
#> 14                     Poland   TRUE  2698
#> 15         Russian Federation   TRUE  3853
#> 16                Switzerland   TRUE  1187
#> 17                    Ukraine   TRUE  1753
#> 18              United States   TRUE 17442
#> 19                   Viet Nam   TRUE 12108
```

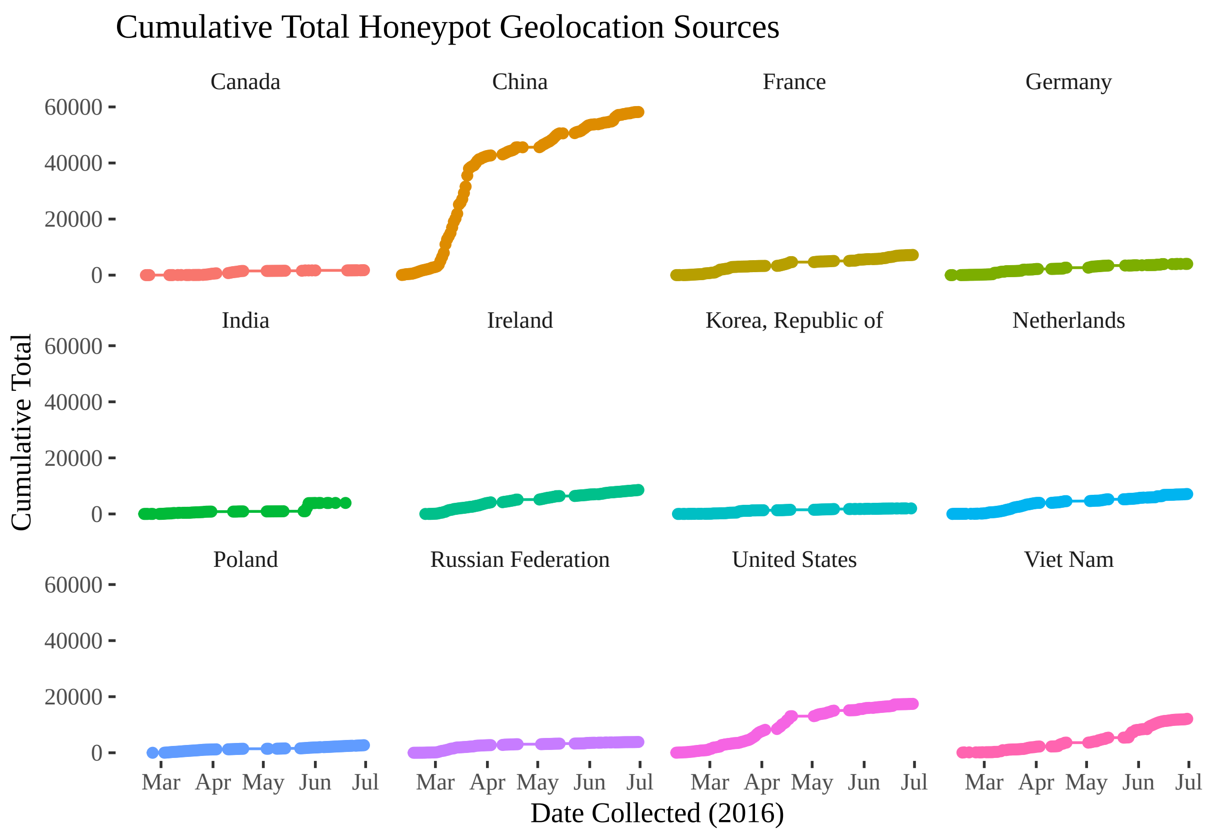
The top contributors to the list are familiar and consistent with the top addresses on the blacklist. The largest population of collected events originates from China. The largest collection of non-threat events originates from the United States. It is not clear why this might be the case, perhaps blacklists are less likely to add IP address from the United States for policy or technical reasons.

```
world_data <- map_data("world")

# some of the important countries are miss labeled!
countryFreq <- countryFreq %>% mutate(Country = as.character(Country)) %>%
    mutate(Country = if_else(Country == "United States",
        "USA", if_else(Country == "United Kingdom", "UK",
            if_else(Country == "Taiwan, Province of China",
                "Taiwan", if_else(Country == "Russian Federation",
                  "Russia", if_else(Country == "Iran, Islamic Republic of",
                    "Iran", if_else(Country == "Korea, Republic of",
                      "South Korea", if_else(Country ==
                        "Korea, Democratic People's Republic of",
                        "North Korea", if_else(Country ==
                          "Venezuela, Bolivarian Republic of",
                          "Venezuela", if_else(Country ==
                            "Bolivia, Plurinational State of",
                            "Bolivia", if_else(Country ==
                              "Viet Nam", "Vietnam", Country))))))))))


cyber_map <- merge(world_data, countryFreq, by.x = "region",
    by.y = "Country", all.x = TRUE)

cyber_map <- cyber_map %>% arrange(group, order)

ggplot(cyber_map, aes(x = long, y = lat, group = group,
    fill = Freq)) + geom_polygon(colour = "black") + scale_fill_gradient2(low = "white",
    high = "#f03b20") + xlab("") + ylab("") + theme(axis.title.x = element_blank(),
    axis.text.x = element_blank(), axis.ticks.x = element_blank(),
    axis.title.y = element_blank(), axis.text.y = element_blank(),
    axis.ticks.y = element_blank()) + facet_wrap(. ~ Threat)
```



Another interesting view is how the countries event total grows over time.

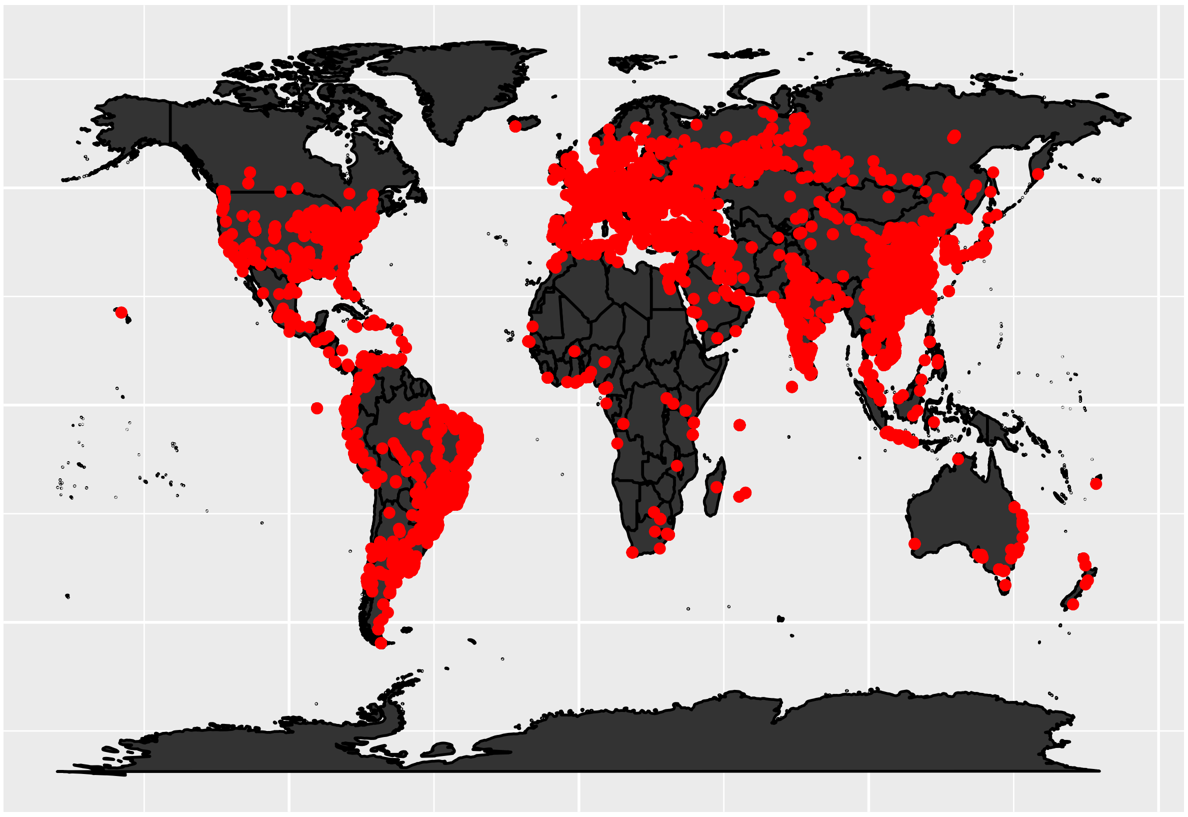
```
countryFreq <- as.data.frame(table(honeypot.data_date_trim$country_name))
countryFreq <- countryFreq %>% arrange(desc(Freq))

honey_cum <- as.data.frame(table(honeypot.data_date_trim$Date_actual,
    honeypot.data_date_trim$country_name)) %>% filter(Freq >
    1 & (Var2 %in% countryFreq$Var1[1:12])) %>% mutate(Var1 = as.Date(Var1))

colnames(honey_cum) <- c("Date", "Country", "Freq")

honey_cum <- honey_cum %>% group_by(Country) %>% mutate(Cumsum = cumsum(Freq))

ggplot(data = honey_cum, aes(x = Date, y = Cumsum, color = Country)) +
    geom_line() + geom_point() + ylab("Cumulative Total") +
    xlab("Date Collected (2016)") + facet_wrap(. ~ Country) +
    ggtitle("Cumulative Total Honeypot Geolocation Sources") +
    theme_tufte() + theme(legend.position = "none")
```



```
rm(honey_cum)
rm(countryFreq)
```

Many of the locations have large amounts of activity in very short periods of time, sometimes just one day. However, the general trend is to add events consistently from countries over time. One major exception is China. The IP address explored earlier accounted for nearly all of the entire countries collect events over the course of a single month. The geolocation does have region and city level data which can add some context.

```
ggplot(cyber_map, aes(x = long, y = lat, group = group)) +
    geom_polygon(colour = "black") + geom_point(data = honeypot.data_date_trim,
    aes(long, lat, group = country_name), colour = "red") +
    xlab("") + ylab("") + theme(axis.title.x = element_blank(),
    axis.text.x = element_blank(), axis.ticks.x = element_blank(),
    axis.title.y = element_blank(), axis.text.y = element_blank(),
    axis.ticks.y = element_blank())
#> Warning: Removed 7010 rows containing missing values (geom_point).
```



```
rm(cyber_map)
rm(world_data)
```

Additional Feature Analysis

In addition to time and geography, there are other essential features that the honeypot collected — specifically the communication ports and protocols.

```
# number of unique source port (port communicated from)

length(unique(honeypot.data_date_trim$source_port))
#> [1] 52497

# frequency of desintation port (por communicated to)

table(honeypot.data_date_trim$destination_port)
#>
#> 1433 5060 2455 47808 23 3306 22 502 8086 523
#> 9130 8121 3 150 845 3640 223853 144 7 155
#> 1883 53 137 3390 8883 177 27017 7547 161 111
#> 145 298 15 11 143 132 109 92 841 349
#> 513 3389 2222 1911 6379 993 5358 123 1434 465
#> 136 201 149 135 60 21 208 191 167 21
#> 17 9191 1900 995 8443 4911 3394 25 7001 69
#> 137 4 295 27 29 5 3 36 10 3
#> 443 1080 8118 54138 9000 3398 5001 8000 143 3392
#> 156 34 4 4 48 2 1 24 22 4
#> 84 2375 5555 11211 1521 5432 1099 7777 9100 50802
#> 3 20 6 52 148 177 35 6 5 4
#> 8081 80 5038 3391 6888 3402 9002 9999 8334 3400
#> 102 158 3 5 1 3 2 9 4 2
#> 3397 48899 3393 8080 1471 3395 9943 5901 8888 15
#> 2 6 3 81 6 2 2 23 12 6
#> 5006 587 1025 5071 8009 4899 3128 9060 19 3387
#> 7 14 5 4 30 45 18 1 12 2
#> 21 2323 12345 110 81 5800 5632 8123 3384 3382
#> 28 83 4 24 56 42 35 7 1 2
#> 3386 8089 389 4022 8088 5070 3380 3388 7007 2083
#> 4 4 16 3 7 5 3 6 1 9
#> 3385 17000 20547 9051 3790 2376 8004 1060 8006 5000
#> 1 2 10 3 4 5 1 3 1 8
#> 7778 8087 2060 53413 5094 5357 3000 8010 6789 3412
#> 1 9 2 3 3 3 3 5 2 2
#> 311 9200 5683 3408 3414 3060 3405 8098 8139 2123
#> 3 56 5 3 3 3 2 4 4 2
#> 8140 10000 4060 9595 3299 195 3411 2628 2181 554
#> 4 4 2 2 5 2 1 4 9 2
#> 5080 5073 6060 135 5065 3409 5076 14147 5084 5091
#> 3 4 3 4 3 2 3 4 2 2
#> 5050 44818 1029 5162 9418 41300 1777 515 5904 21379
#> 2 9 2 2 5 16 5 1 3 3
#> 8008 7060 3410 4070 4063 5269 3415 3403 8091 8093
#> 6 2 3 4 5 3 2 2 2 1
#> 30718 3413 5672 37777 8069 7010 26 8126 8060 51106
#> 3 3 4 18 3 1 6 2 7 3
#> 3374 6667 3376 666 5009 7071 8998 4443 3377 27015
#> 1 4 1 7 1 1 8 7 1 4
#> 3370 3784 8095 8333 9042 3378 873 1991 444 5900
#> 1 1 1 3 2 1 5 5 6 62
#> 5007 9001 25105 445 8099 1604 8834 1234 8097 3381
#> 6 2 5 43 2 7 3 1 1 3
#> 8453 32400 902 1023 7657 3407 4800 20 3689 32764
#> 1 1 2 2 2 1 4 2 5 4
#> 79 2400 5353 8800 25565 55362 5601 5903 4848 9600
#> 1 2 6 1 2 1 7 4 5 8
#> 3780 4567 3310 10001 70 5560 636 3406 1200 3268
#> 4 3 7 14 6 2 8 1 9 3
#> 3269 50100 3460 5984 1038 88 82 20000 4040 789
#> 4 3 2 6 2 7 7 12 1 4
#> 20001 8554 3401 9064 500 3038 13579 6881 6664 1962
#> 1 3 1 13 3 2 2 2 2 6
#> 104 6666 1818 129 37 5905 623 8090 2332 626
#> 4 6 1 5 2 2 4 2 1 4
#> 5351 45554 3404 33389 58260 1010 29081 17185 4000
520
#> 2 4 2 2 2 2 1 5 6 6
#> 5577 5907 5938 9080 5908 119 9550 7474 3001 5909
#> 1 1 7 5 1 4 1 7 4 1
#> 5902 2433 4444 3300 49 771 55553 21025 7779 9098
#> 5 4 2 2 3 5 6 3 2 1
#> 50000 33899 8649 5911 5815 37397 139 2152 60088 13
#> 2 1 5 1 1 4 2 3 2 1
#> 3749 26165 2067 23424 3129 2000 990 55554 3542 3541
#> 4 2 4 2 1 3 1 1 3 1
#> 7548 61439 9981 3190 5008 3399 50070 9160 3396
33434
#> 2 3 7 1 1 1 6 6 1 6
#> 5986 113 5 2081 5090 5072 5062 21320 2087 5074
#> 1 1 1 1 1 2 2 2 2 2
#> 5078 5082 5092 5064 23023 5066 5068 5075 5085 5095
#> 2 2 2 2 3 2 2 2 2 1
#> 5222 5521 10200 8083 2086 62078 2404 16010 4369
4123
#> 1 2 1 4 2 5 6 5 2 1
#> 30005 16993 10022 9797 9833 1400 1177 8084 5912
1213
#> 1 4 1 4 2 3 2 3 1 1
#> 18245 8085 8989 8462 8642 992 9443 28017 808 40000
#> 1 4 1 1 1 1 6 3 2 1
#> 5806 5920 179 10554 64738 4840 3309 9151 1311 83
#> 1 1 1 2 2 1 1 2 2 1
#> 9050 6000 102 2480 16992 9009 3383 22022 55443 6969
#> 3 3 13 2 1 3 1 1 1 3
#> 1467 8181 30301 3301 5096 3302 8909 24809 8082
30401
#> 1 2 3 1 1 6 1 1 2 3
#> 175 3702 30022 11 13389 8889 631 20143 9869 30501
#> 1 1 1 1 1 1 1 1 2 3
#> 30001 8112 34002 37015 4730 1741 53274 12738 19009
9003
#> 4 1 2 1 2 1 1 1 1 2
#> 4685 5813 3440 3351 3358 3361 3435 3372 8180 5914
#> 1 1 0 0 0 0 0 0 0 0
#> 4333 5811 7859 35923 53281 4782 7454 7256 2012 7465
#> 0 0 0 0 0 0 0 0 0 0
#> 7319 7441 7057 7430 7309 7339 7144 7284 7000 1000
#> 0 0 0 0 0 0 0 0 0 0
#> 11300 27016 3201
#> 0 0 0
```

A similar breakdown of the protocols used to communicate.

```
table(honeypot.data_date_trim$protocol)
#>
#>    TCP    UDP   ICMP    ssh
#>  24162  11118    115 217561
```

The majority of the events are ssh targeting port 22. For threat behavior this makes sense. They are attempting to use the default admin port to gain access to the system. This set of information (country -> protocol -> source_port -> destination_port) can be considered a set of behavior. For example, consider the IP address “182.18.22.136” from China.

```r
library(riverplot)

example_ip <- honeypot.data_date_trim %>% filter(source_ip ==
    "182.18.22.136") %>% select(country_name, source_port,
    protocol, destination_port) %>% mutate_at(vars(1:4),
    as.character)

nodes <- data.frame(ID = c(unique(example_ip$country_name),
    unique(example_ip$protocol), unique(example_ip$source_port),
    unique(example_ip$destination_port)), x = c(rep(1, length(unique(example_ip$country_name))),
    rep(2, length(unique(example_ip$protocol))), rep(3,
        length(unique(example_ip$source_port))), rep(4,
        length(unique(example_ip$destination_port)))), col = c(rep("red",
    length(unique(example_ip$country_name))), rep("yellow",
    length(unique(example_ip$protocol))), rep("blue", length(unique(example_ip$source_port))),
    rep("orange", length(unique(example_ip$destination_port)))),
    stringsAsFactors = FALSE)

country_to_protocol <- as.data.frame(table(example_ip$country_name,
    example_ip$protocol)) %>% filter(Freq > 0 & !is.na(Var1) &
    !is.na(Var2)) %>% arrange(Var1, Var2) %>% mutate_at(vars(1:2),
    as.character)

protocol_to_source_port <- as.data.frame(table(example_ip$protocol,
    example_ip$source_port)) %>% filter(Freq > 0 & !is.na(Var1) &
    !is.na(Var2)) %>% mutate(Var2 = as.numeric(as.character(Var2))) %>%
    arrange(Var1, Var2) %>% mutate_at(vars(1:2), as.character)

source_port_to_dest_port <- as.data.frame(table(example_ip$source_port,
    example_ip$destination_port)) %>% filter(Freq > 0 &
    !is.na(Var1) & !is.na(Var2)) %>% mutate_at(vars(1:2),
    as.character) %>% mutate_at(vars(1:2), as.numeric) %>%
    arrange(Var1, Var2) %>% mutate_at(vars(1:2), as.character)

edges <- rbind(country_to_protocol, protocol_to_source_port,
    source_port_to_dest_port) %>% set_names(c("N1", "N2",
    "Value")) %>% mutate(ID = paste0(N1, "_", N2))

rm(country_to_protocol, protocol_to_source_port, source_port_to_dest_port)

r <- makeRiver(nodes, edges)
plot(r)

rm(edges, nodes, r, example_ip)
```

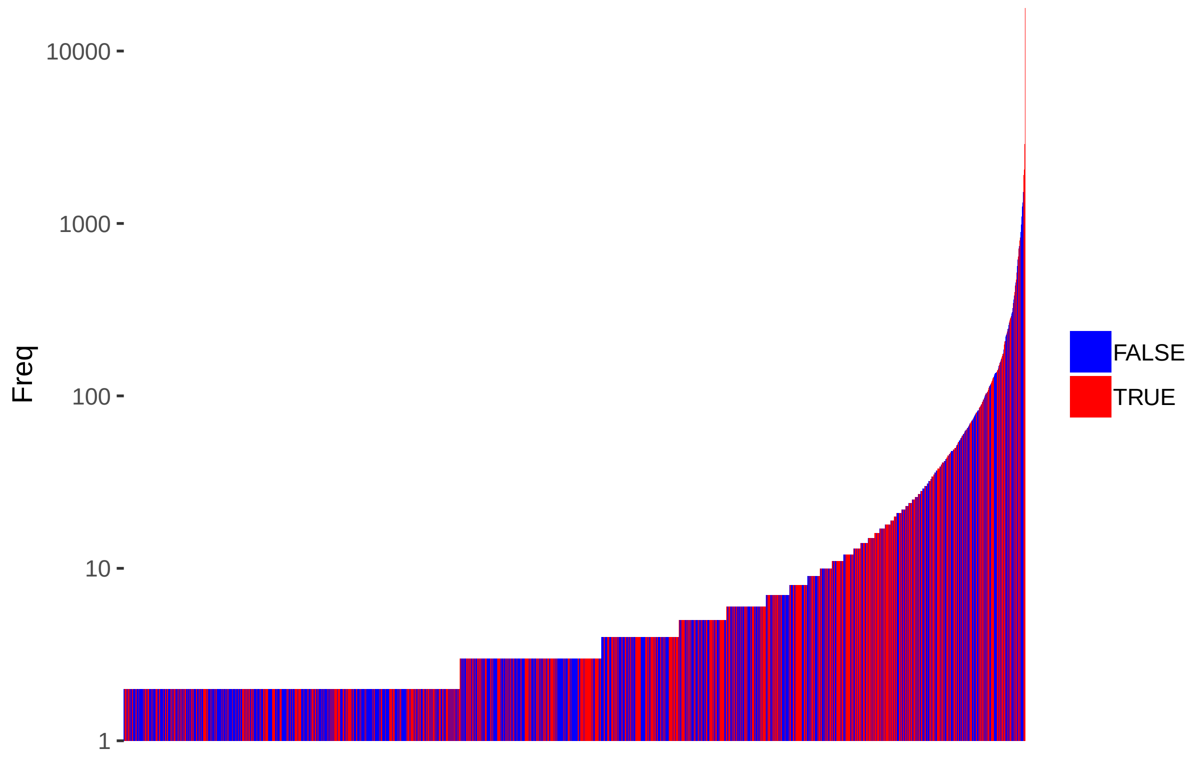
Sometimes the flow of information for a single IP address to large to plot this way. Consider the Chinese IP address used in the previous section.

```r
# number of total events
nrow(china_ip)
#> [1] 17797

# the number of source port used by this IP address
# during it's activity
```

```
length(unique(china_ip$source_port))
#> [1] 15167

# number of protocols uses
table(as.character(china_ip$protocol))
#>
#>   ssh   TCP
#> 17794     3

# breakdown of destination ports
table(as.character(china_ip$destination_port))
#>
#>    22
#> 17797
rm(china_ip)
```

From this one IP address, the vast majority of the events originate from different ports. This is 'port promiscuity' and can be considered unique behavior. However, the connections are very consistent using a single protocol and destination port. Of course, the frequency of the events is suspicious on its own. If we look at the total frequency of and IP addresses events as a behavior, it also contains some identifying trends.

```
# total freqency of IP addresess

threats <- honeypot.data_date_trim %>% filter(threat ==
    TRUE)

source_ip_freq <- as.data.frame(table(honeypot.data_date_trim$source_ip),
    stringsAsFactors = FALSE) %>% mutate(Var2 = (Var1 %in%
    threats$source_ip)) %>% arrange(desc(Freq)) %>% filter(Freq >
    1)

rm(threats)

ggplot(source_ip_freq, aes(x = reorder(Var1, Freq), y = Freq,
    fill = Var2)) + geom_bar(stat = "identity") + scale_y_log10() +
    scale_fill_manual(values = c("blue", "red")) + theme(axis.title.x = element_blank(),
    axis.text.x = element_blank(), axis.ticks.x = element_blank(),
    legend.title = element_blank())
```

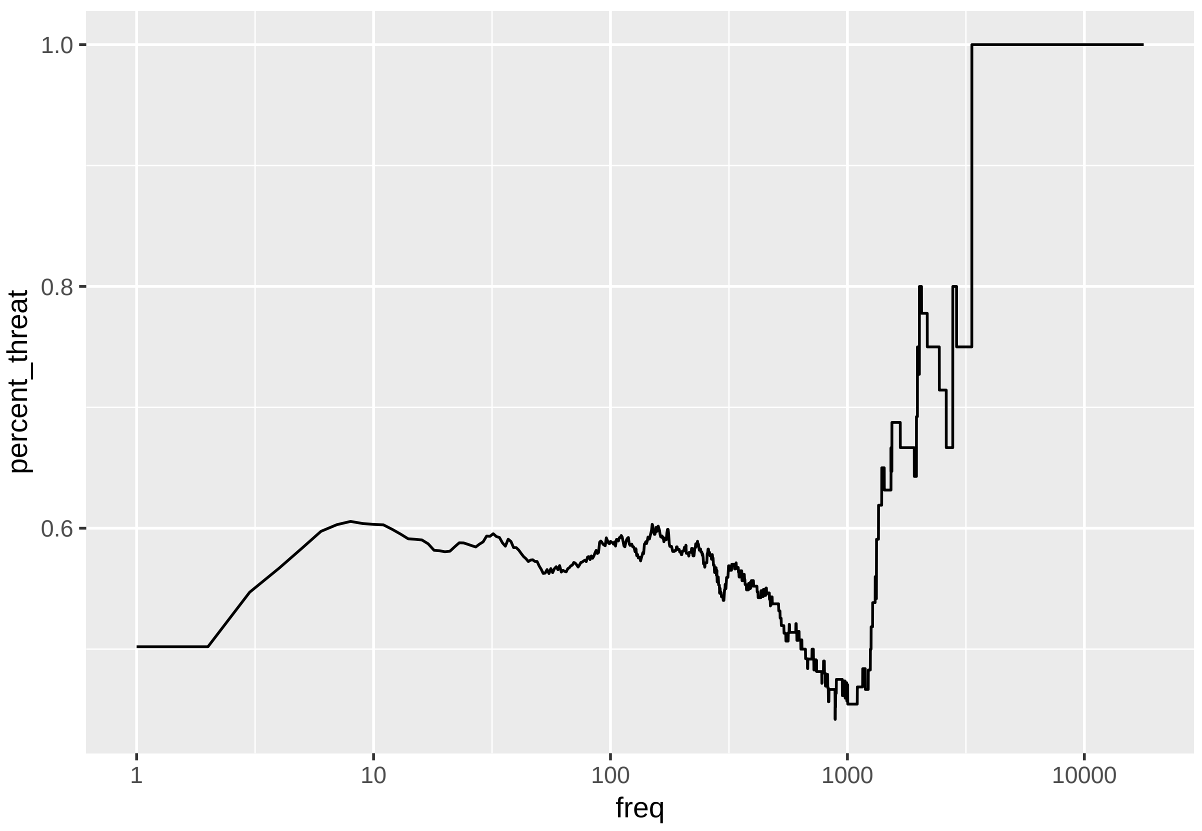


```
rm(source_ip_freq)
```

A quick overview of the frequency compared to threat category shows that the higher the frequency, the more often the IP address is blacklisted. Another quantifiable way to look at this is to see the percentage of blacklisted IP addresses increaeses with the frequency.

```
threats <- honeypot.data_date_trim %>% filter(threat ==
    TRUE)

source_ip_freq <- as.data.frame(table(honeypot.data_date_trim$source_ip),
    stringsAsFactors = FALSE) %>% mutate(Var2 = (Var1 %in%
    threats$source_ip)) %>% arrange(desc(Freq)) %>% filter(Freq >
    1)

rm(threats)

percent_threat_by_freq <- data.frame()
for (i in 1:max(source_ip_freq$Freq)) {
    temp <- source_ip_freq %>% filter(Freq >= i)
    p <- sum(as.numeric(temp$Var2))/nrow(temp)
    temp <- data.frame(freq = i, percent_threat = p)
    percent_threat_by_freq <- rbind(percent_threat_by_freq,
        temp)
}
```

```
ggplot(percent_threat_by_freq, aes(x = freq, y = percent_threat)) +
    geom_line() + scale_x_log10()
```



```
rm(percent_threat_by_freq)
```

Frequencies between 1 and 1000 appear on the blacklist about 50% of the time. However, over 100 the percentage climbs quickly. This is a natural result which points to another reliable behavioral indicator.

Advanced Actors

Similar to the blacklist, one question that might be considered is the range of actors which can be reasonably expected to be collected using this metodology. Using the data from operation GRIZZLY STEPPE the honeypot data can be checked to see if it gathered information about this level of actor.

```
table(honeypot.data_date_trim$threat, honeypot.data_date_trim$russian_actor)
#>
#>          FALSE   TRUE
#>   FALSE 106502      0
#>   TRUE  146336    118
```

Of the quarter-million events in the trimmed set 118 of them belong to Russian actors, and all of these are captured on the blacklist. So it is reasonable that, while low frequency, even advanced actors are present in data collected from public facing security devices such as is presented here.

Conclusion

Similar to the blacklist this data set cannot be used in a canonical, but it is quite useful to represent the messiness inherent in a real-world setting. The labeling of threats is incomplete but is passable to build algorithms against. While the behavioral information collected in this setting is limited, there remains a reasonable set to use as initial features in a supervised machine learning setting. These include

- IP address - geolocation information: country, city, lat, long
- Port information - source and destination: unique types and promiscuity
- Temporal information - the time of day, week, month and year
- Frequency - the number of events, interarrival times between events
- threat - blacklist occurrence
- honeypot type - snort or cowrie
- attempted login - username and password

This information exists in every cybersecurity setting and can serve as a basis for initial feature selection when building automated learning systems.

Appendix G

Deeplearning Architecture Exploration

To build a manageable feature set, this research makes use of custom developed encodings (similar to NLP modeling) using expert input. The input layer of the model X will include three categories of encoded features: time encoding, port encoding, and IP encoding. These encodings come from the 600,000 network events of the honeypot data and are done by unique IP address. The time encoding includes 2 parameters:

- the average of the hour of the timestamp (GMT)
- the standard deviation of the hour of the timestamp (GMT)

The average hour of the day, t_i will be a discrete input from 1 to 24 corresponding to the i^{th} hour of the day in Greenwich Mean Time.

The next subset of parameters comes from the port encoding of the data. There are six port encoding parameters. The first is a binary variable indicating whether the protocol request is regular (1) or irregular (0). The next input parameter is the source port of the network event. The following three port-related parameters are a one-hot vector used to classify the destination and source port:

- common database ports
- standard network ports (i.e. 22, 80, 443, etc.)
- the number of unique ports

Finally, the IP encoding includes 3 input parameters:

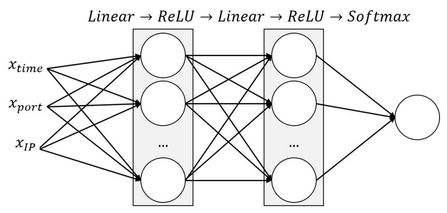


FIGURE G.1: The network architecture involves two hidden layers that are linear with ReLU activation functions and a sigmoid activation layer.

- total frequency of requests
- Geo-code: latitude
- Geo-code: longitude

Using these encodings avoids the pitfalls of large one hot vectors. For example, the dataset contains 473 destination ports. Using a one-hot vector to represent all of those would result in 472 parameters with value 0 (the destination ports not asked by the training example), and 1 with value 1 (the destination port requested by the training example). A large number of zeros input into the neural network will result in a sparse network. Therefore, by using creative encodings, avoids a sparse network that is unable to learn efficiently.

G.1 Network Architecture

A feedforward network is implemented here. The network in its original form (Figure G.1) has three layers and a softmax output. The hidden layers are linear with ReLU activation functions. This is a generic network architecture that allows for rapid initial implementation of the model and then to be able to focus more on iterations for improving performance. Because the output to classifies as 0 (not a threat) or 1 (threat), the softmax function with two outputs is the best for the last activation function. In the fully connected network, there are 478 corresponding neurons per hidden layer and 1 neuron for the activation layer. The resulting size of the parameters of the model is as follows: $W^{[1]} = [478, 9], b^{[1]} = [478, 1], W^{[2]} = [478, 478], b^{[2]} = [478, 1], W^{[3]} = [478, 2], b^{[3]} = [2, 1]$. This means that the model will handle 458,403 different parameters. This is a manageable number of parameters and is easily reducible with changes in network architecture as needed. However, using this network architecture still produces both bias and variance.

Three additional network architectures are explored to address the issues of bias and variance. To reduce bias in the model, a bigger network is used (Figure G.2). The bigger

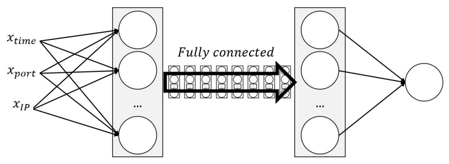


FIGURE G.2: The network architecture involves ten hidden layers that are linear with ReLU activation functions and a sigmoid activation layer.

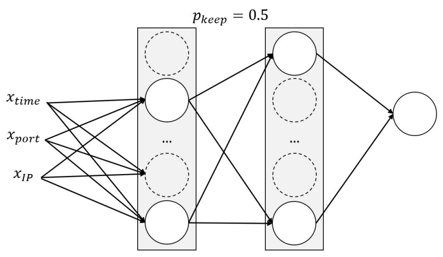


FIGURE G.3: The network architecture involves two hidden layers that are linear with ReLU activation functions and a sigmoid activation layer with dropout implemented in each hidden layer (keep probability = 0.5).

network contains 11 fully-connected layers instead of 3 which will avoid underfitting the model. The resulting size of the parameters of the bigger network are as follows: $W^{[1]} = [12, 9], b^{[1]} = [12, 1], W^{[2]} = [12, 12], b^{[2]} = [12, 1], W^{[3]} = [12, 12], b^{[3]} = [12, 1], W^{[4]} = [12, 12], b^{[4]} = [12, 1], W^{[5]} = [12, 12], b^{[5]} = [12, 1], W^{[6]} = [12, 12], b^{[6]} = [12, 1], W^{[7]} = [12, 12], b^{[7]} = [12, 1], W^{[8]} = [12, 12], b^{[8]} = [12, 1], W^{[9]} = [12, 12], b^{[9]} = [12, 1], W^{[10]} = [12, 12], b^{[10]} = [12, 1], W^{[11]} = [2, 12], b^{[11]} = [2, 1]$. Much like the original network, each of these layers is feed-forward. Yet, the number of connections between layers is much smaller in order to reduce computation time. Therefore, the total number of variables is only 1,550.

To reduce variance in the model, regularization is introduced in the form of dropout (Figure G.3). Dropout is used because of the relatively large layer sizes. Dropout effectively shrinks the network and spreads out the weights through the model. The first implementation of the dropout model is on the three-layer network. The amount of regularization effect from dropout is dependent on the probability of keeping a neuron in the network. As the probability of maintaining a neuron increases, the regularizing impact decreases. However, as the probability of maintaining a neuron in the network goes up, the amount of training set error decreases. To balance the benefits of regularization with the adverse effects of training set error, the probability of keeping a neuron in our dropout model is 0.5.

Finally, a combined 11 layer network and dropout is explored in order to combine the benefits of reduced bias and reduced variance (Figure G.4). It is expected that the

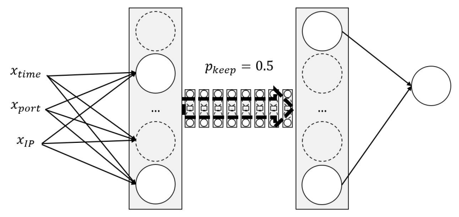


FIGURE G.4: The network architecture involves ten hidden layers that are linear with ReLU activation functions and a sigmoid activation layer with dropout implemented in each hidden layer (keep probability = 0.5).

benefits will be subadditive because the larger network will increase variance and the regularizing effect of dropout will increase bias.

G.2 Loss Function

The networks all use use a standard cross-entropy loss function: $\mathcal{L}\{y, \hat{y}\} = -[y \log \hat{y} + (1-y) \log(1-\hat{y})]$. This loss function is ideal for the threat/non-threat binary classification of this model. Therefore, the cost function for this model is $J(\theta) = \frac{1}{m} \sum \mathcal{L}\{y, \hat{y}\}$ as seen in chapter 3. An AdamOptimizer is deployed for efficient gradient descent which helps to minimize costs over the cross entropy loss function. The AdamOptimizer will use the standard hyperparamter values of: $\beta_1 = 0.9, \beta_2 = 0.999, and \epsilon = 10^{-8}$. This is a logical choice for this problem because it is easy to configure with only the hyperparamter value, α, needed for training. It combines the advantages of gradient descent and RMSProp and is best equipped to handle sparse gradients.

G.3 Hyperparameters

The choice of hyperparameters - learning rate, epochs, and minibatch size - is crucially important, as hyperparameters affect all subsequent parameters. After testing over a wide range of values, the best values for this early application where

- Learning rate = 0.001
- Epochs (time steps)= 1500
- Minibatch Size = 32

This resulted in the best model performance. In testing these values, learning rates from 0.1 to 0.00001 where explored as well as epochs from 200 to 2000. The minibatch size is varied extensively in multiples of 2 from 32 to 1,024. Over the experiments, it was found that a minibatch size of 32 performed the best and did not significantly impact the run time of the model.

G.4 Measuring Error

Model performance is measured by exploring error rates. By comparing the results to the blacklist dataset, error of the training set is easily calculated for both the development set, and the test set. Using these measurements of error, decisions on architecture for the levels of variance and bias from the models will be straightforward.

For cybersecurity applications, false positives are especially resource-intensive errors. Therefore, a focus will be on minimizing false positive results to improve precision and recall related to the decision thresholds for alerting the expert analyst. Due to this concern with false positives, the value of precision will be of more significant interest than recall or F1 score to measure error. However, accuracy, precision, recall (sensitivity), F1 score, and specificity will all be collected for each model variant.

$$
\begin{aligned}
Accuracy &= \frac{TruePos + TrueNeg}{TurePos + FalsePos + TrueNeg + FalseNeg} \\
Precision &= \frac{TruePos}{TruePos + FalsePos} \\
Recall &= \frac{TruePos}{TruePos + FalseNeg} \\
F1 &= \frac{2 * Precision * Recall}{Precision + Recall}
\end{aligned}
$$

G.5 Results

G.5.1 Performance Metrics

The model is run using mini-batch gradient descent with and Adam optimizer using TensorFlow in python. Table G.1 shows the variability among different network architectures. Overall, deep learning methods outperformed traditional machine learning methods, like logistic regression, in all measures. Accuracy was highest with the original 3 layer network, the enlarged 11 layer network, and the 11 layer network with dropout. Precision was the weakest performance measure for all of the different model structures. Precision is the proportion of threat detections that are true threats rather than false positives. This indicates that more than half of the threats detected by the model are not actually threats. The 11 layer network performs the best of any network in precision. Recall and F1 score are less important measures for this application, as we aim to minimize false positives. The recall represents the proportion of actual threats that were detected. In this case, all of the network architectures performed relatively well. As expected, the F1 scores fell in between the precision and recall scores for all network architectures.

Table G.1: Test Model Results

	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.59	0.40	0.59	0.46
3 Layers	0.64	0.44	0.64	0.51
Dropout	0.63	0.42	0.63	0.49
11 Layers	0.64	0.47	0.62	0.53
11 Layer (Dropout)	0.64	0.45	0.64	0.53

G.5.2 Costs

The cost improvement for all models is generally consistent. The costs improve over epochs but level out quickly with only marginal improvement after epoch 100. As expected with minibatch implementation, the cost experiences some perturbations over time. These are minor when compared to the general trend of the model costs. These perturbations are due to the stochastic nature of a minibatch implementation. The original network architecture and the larger network architecture cost improvements (Figures G.5 and G.6) are not dramatic, but they do show measurable improvement over time. This indicates that the model is learning from the dataset, albeit slowly. As the network depth increases, the overall cost computed also decreases. This reflects the overall improved performance as the network grows. The dropout network did not improve costs after the first iteration (Figure G.5). This is likely because the model did not have much variance. Therefore, regularization did not improve the costs associated with the model.

G.6 Conclusions

The initial performance of these models against the collected data is not strong. The maximum accuracy is still under 65%. However, the models do show signs of promise; particularly, in that they universally outperform traditional machine learning methods, such as logistic regression. Of the network architectures explored, the 11 layer network

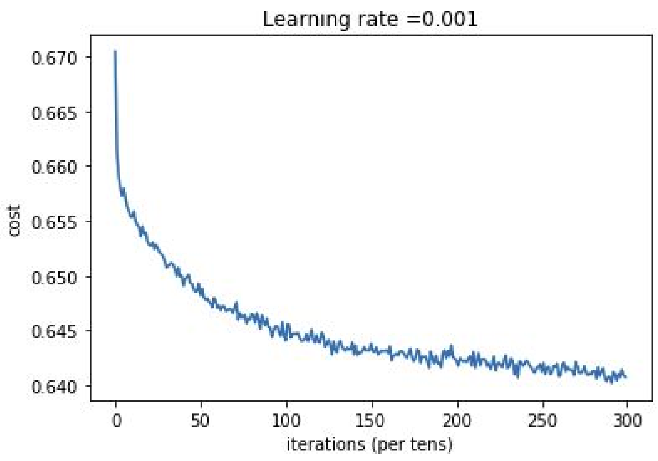


FIGURE G.5: The cost function of the three layer network over 1500 epochs.

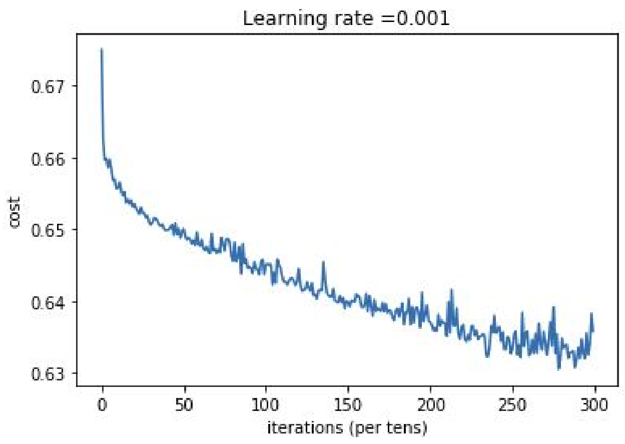


FIGURE G.6: The cost function of the eleven layer network over 1500 epochs.

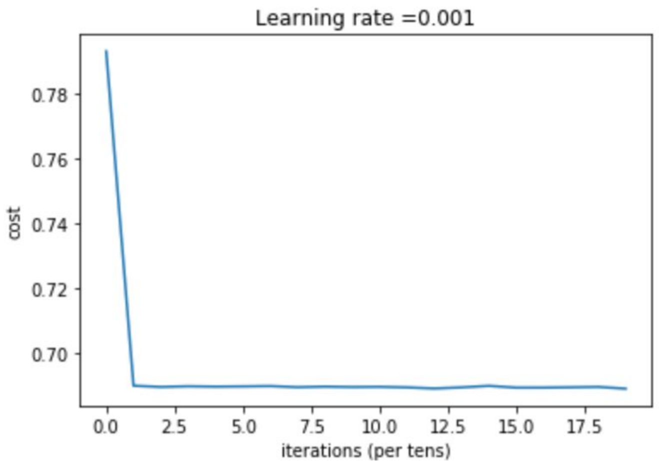


FIGURE G.7: The cost function of the three layer network with dropout over 100 epochs.

without dropout performed the best. This indicates that the model inherently has more problems with bias than with variance.

The reason for this poor performance is likely due to the dataset itself. The individual events (log data from the honeypot sensors) are noisy and may not contain enough information, especially with the encoding. While encoding allows us to avoid scarce networks, it requires a high level of expertise and will inevitably influence the model results.

The high bias observed in the model is likely also caused in part by the data itself. Collecting raw internet data will always introduce bias at some level. The collection of raw and biased internet data is, therefore, the cause of some of the performance degradation.

Appendix H

Machine Learning Implementation of Known and Unknown Gate Set

Machine Learning Implementation of Known and Unknown Gate Set

This code can be forklifted from the MatrixDS project link here:

https://community.platform.matrixds.com/community/project/5a6fb4b57a031800010a0bce/files

Known Gate Set Implementation

Supporting functions

```r
# Active Learning Suport Functions Thompson Sampling
# with Beta Binomial and Gaussian MLE with epsilon
# greedy exploration Thompson Sampling

# a utility function

utility_function <- function(x, gamma) {

    if (gamma == 0) {
        return(x)
    }
    x <- (1 - exp(x * -gamma))
    if (gamma == 0) {
        return(x)
    }
    return(x)
}

value_from_utility <- function(x, gamma) {
    if (gamma == 0) {
        return(x)
    }
    # this is a bit hacky but works for edge sampling issues
    # with the t distribution
    u <- (1 - x)

    if (u < 0) {
        u <- 1e-06
    }

    x <- -(log(u)/gamma)
    return(x)
}

singleTimeBetaBinomialThompsonSample <- function(threatProbVector,
    actorRewards, gatePositionThompson, betaParams, gamma) {
    myList <- list()
    for (i in 1:length(threatProbVector)) {
        # first sample from the priors
        b <- rbeta(1, betaParams[i, 1], betaParams[i, 2])
        t <- rnorm(1, betaParams[i, 3], betaParams[i, 4])
        nt <- rnorm(1, betaParams[i, 5], betaParams[i, 6])

        expectation <- (1 - b) * (t) + b * (nt)
        # print(expectation)
        expectation <- value_from_utility(expectation, gamma)
        # print(expectation) next set the gate policy
        if (expectation > 0) {
            gatePositionThompson[i, 1] <- 1
        } else {
            # close gate and set period reward to zero
            gatePositionThompson[i, 1] <- 0
            gatePositionThompson[i, 2] <- 0
        }
        # next sample from the dist to see if there is an actual
        # threat

        if (gatePositionThompson[i, 1] == 1) {
            # type door sample from distribution
            x <- runif(1)
            # add data point to data frame value for number of
            # observations
            betaParams[i, 7] <- betaParams[i, 7] + 1

            if (x < threatProbVector[i]) {
                gatePositionThompson[i, 2] <- utility_function(rnorm(1,
                  actorRewards$threat_mean[i], actorRewards$threat_sd[i]),
                  gamma)
                # do the bayesian updating here
                m <- ((betaParams[i, 8] * betaParams[i,
                  3]) + betaParams[i, 4]^2 * gatePositionThompson[i,
                  2])/(betaParams[i, 4]^2 + betaParams[i,
                  8])
                v <- 1/((1/(betaParams[i, 4]^2)) + (1/(betaParams[i,
                  8])))

                betaParams[i, 3] <- m
                betaParams[i, 4] <- sqrt(v)
                betaParams$beta[i] <- betaParams$beta[i] +
                  1

            } else {
                gatePositionThompson[i, 2] <- utility_function(rnorm(1,
                  actorRewards$nonthreat_mean[i], actorRewards$nonthreat_sd[i]),
                  gamma)

                # do the bayesian updating here
                m <- ((betaParams[i, 8] * betaParams[i,
                  5]) + betaParams[i, 6]^2 * gatePositionThompson[i,
                  2])/(betaParams[i, 6]^2 + betaParams[i,
                  8])
                v <- 1/((1/(betaParams[i, 6]^2)) + (1/(betaParams[i,
                  8])))
                betaParams[i, 5] <- m
                betaParams[i, 6] <- sqrt(v)

                betaParams$alpha[i] <- betaParams$alpha[i] +
                  1
            }
        }

    }  #close the for loop
    myList$gatePositionThompson <- gatePositionThompson
    myList$betaParams <- betaParams

    return(myList)
}  #close the function

singleTimeMLESample <- function(threatProbVector, actorRewards,
    gatePositionMLE, mleParams, epsilon, gamma) {
    myList <- list()
    for (i in 1:length(threatProbVector)) {
        # first sample from the priors
        epsilonGreedySample <- runif(1)
        # next set the gate policy
        expectation <- (mleParams[i, 1] * (mleParams[i,
            2]) + (1 - mleParams[i, 1]) * (mleParams[i,
            3]))
        # print(paste0('MLE:',expectation))
        expectation <- value_from_utility(expectation, gamma)
        # print(paste0('MLE:',expectation))

        if (epsilonGreedySample <= epsilon) {
            gatePositionMLE[i, 1] <- 1
        } else if (expectation > 0) {
            gatePositionMLE[i, 1] <- 1
        } else {
            gatePositionMLE[i, 1] <- 0
            gatePositionMLE[i, 2] <- 0
        }

        # next sample from the dist to see if there is an actual
        # threat
        x <- runif(1)
        if (gatePositionMLE[i, 1] == 1) {
            # x1 type door add data point to data frame value for
            # number of observations
            mleParams[i, 4] <- mleParams[i, 4] + 1

            if (x < threatProbVector[i]) {
                gatePositionMLE[i, 2] <- utility_function(rnorm(1,
                  actorRewards$threat_mean[i], actorRewards$threat_sd[i]),
                  gamma)
                # mle estimate update
                mleParams[i, 5] <- mleParams[i, 5] + 1
                mleParams[i, 2] <- (mleParams[i, 2] * ((mleParams[i,
                  5] - 1)/mleParams[i, 5]) + gatePositionMLE[i,
                  2] * (1/mleParams[i, 5]))
                mleParams[i, 7] <- (mleParams[i, 7] * ((mleParams[i,
                  5] - 1)/mleParams[i, 5]) + (gatePositionMLE[i,
                  2] - mleParams[i, 2])^2/mleParams[i, 5])
            } else {
                gatePositionMLE[i, 2] <- utility_function(rnorm(1,
                  actorRewards$nonthreat_mean[i], actorRewards$nonthreat_sd[i]),
                  gamma)

                mleParams[i, 6] <- mleParams[i, 6] + 1
                mleParams[i, 3] <- (mleParams[i, 3] * ((mleParams[i,
                  6] - 1)/mleParams[i, 6]) + gatePositionMLE[i,
                  2] * (1/mleParams[i, 6]))
                mleParams[i, 8] <- (mleParams[i, 8] * ((mleParams[i,
                  6] - 1)/mleParams[i, 6]) + (gatePositionMLE[i,
                  2] - mleParams[i, 3])^2/mleParams[i, 6])
            }
            mleParams[i, 1] <- mleParams[i, 5]/(mleParams[i,
                5] + mleParams[i, 6])
        }

    }  #close the for loop
    myList$gatePositionMLE <- gatePositionMLE
    myList$mleParams <- mleParams

    return(myList)
}  #close the function

singleTimeMetalogSample <- function(threatProbVector, actorRewards,
    gatePositionMetalog, metalogParams, gamma) {
    myList <- list()
    for (i in 1:length(threatProbVector)) {
        # default to open gate
        s <- 1

        # default to zero
        t_sample <- 0

        # stored data for a given gate
        metalog_gate_raw_data <- metalogParams$data[metalogParams$data$gate ==
            i, ]
        n_obs <- nrow(metalog_gate_raw_data)
        if (n_obs < 9) {
            n_terms <- 6
        } else {
            n_terms <- 6
        }

        # if there are more than 4 observations for a gate
        # create a metalog object and sample from it
        if (nrow(metalog_gate_raw_data) == 6) {
            my_metalog <- metalog(metalog_gate_raw_data$obs,
                term_limit = n_terms, save_data = TRUE)
            # fix this sampling
            t_sample <- rt(1, my_metalog$params$bayes$v[2])

            s <- my_metalog$params$bayes$q_bar[2] + sqrt(my_metalog$params$bayes$sig[2,
                2]) * t_sample

            s <- value_from_utility(s, gamma)

            # save the metalog as a prior
            metalogParams$priors[[i]] <- my_metalog
            # if there are not 4 observations open the gate to
            # require exploration until enough data to fit a
            # distribution is collected
        }
        if (n_obs > 6) {
            # fix this sampling

            t_sample <- rt(1, metalogParams$priors[[i]]$params$bayes$v[2])

            s <- metalogParams$priors[[i]]$params$bayes$q_bar[2] +
                sqrt(metalogParams$priors[[i]]$params$bayes$sig[2,
                  2]) * t_sample
            s <- value_from_utility(s, gamma)
        }

        # next set the gate policy print(paste('gate:',i,'
        # s:',s))
        if (s > 0) {
            gatePositionMetalog[i, 1] <- 1
        } else {
            gatePositionMetalog[i, 1] <- 0
            gatePositionMetalog[i, 2] <- 0
        }

        if (gatePositionMetalog[i, 1] == 1) {
            # next sample from the dist to see if there is an actual
            # threat
            x <- runif(1)
            if (x < threatProbVector[i]) {
                gatePositionMetalog[i, 2] <- utility_function(rnorm(1,
                  actorRewards$threat_mean[i], actorRewards$threat_sd[i]),
                  gamma)

            } else {
                gatePositionMetalog[i, 2] <- utility_function(rnorm(1,
                  actorRewards$nonthreat_mean[i], actorRewards$nonthreat_sd[i]),
                  gamma)
            }
            # add the observation to the dataset
            temp <- data.frame(gate = i, obs = gatePositionMetalog[i,
                2])
            metalogParams$data <- rbind(metalogParams$data,
                temp)

            # if the gate was opened update with the new observation
            if (n_obs > 5) {
                metalog_gate_raw_data <- metalogParams$data[metalogParams$data$gate ==
                  i, ]
                metalogParams$priors[[i]] <- metalog(metalog_gate_raw_data$obs,
                  term_limit = n_terms, save_data = TRUE)
                gatePositionMetalog$q10[i] <- qmetalog(metalogParams$priors[[i]],
                  0.1)
                gatePositionMetalog$q50[i] <- qmetalog(metalogParams$priors[[i]],
                  0.5)
                gatePositionMetalog$q90[i] <- qmetalog(metalogParams$priors[[i]],
                  0.9)
            }
        }

    }  #close the for loop
    myList$gatePositionMetalog <- gatePositionMetalog
    myList$metalogParams <- metalogParams

    return(myList)
}  #close the function
```

Model implementation

```
# Active Learning Test Script for Example Gate Policy
# Use Case MLE, Beta/Binomial Thompson Sampling, and
# Metalog

library(progress)
library(ggplot2)
library(rmetalog)
library(tidyverse)
library(cowplot)

source("/home/rstudio/analysis/active_learning/activeLearningSupport.R")

# this is a vector of probabilites in a binomial setting
# for likelihood of a threat
threatProbVector <- c(0.1, 0.3, 0.6, 0.6, 0.1, 0.2, 0.5,
    0.7, 0.1, 0.3, 0.6, 0.6, 0.1, 0.2, 0.5, 0.7)

threatProbVector <- sort(threatProbVector)

# number of gates in the example
gateSize <- length(threatProbVector)

# a dataframe with threat and nonthreat rewards and
# standard deviations
actorRewards <- data.frame(threat_mean = rep(-1, gateSize),
    threat_sd = rep(1, gateSize), nonthreat_mean = rep(1,
        gateSize), nonthreat_sd = rep(1, gateSize))
```

```r
# express risk attitude as rho
gamma_vec <- c(-1, 0, 1)

# number or loops or iterations for averaging
l <- length(gamma_vec)

# number of epochs per iteration (time periods)
e <- 100

# some parameters for the epsilon greedy applications
# (used with MLE)
epsilon <- 0.2

# a data frame to capture average rewards over the
# example horizon
avgRewards <- data.frame(period = seq(1, e, 1), avgRewardsMLE = rep(0,
    e), avgRewardsThompson = rep(0, e), avgRewardsMetalog = rep(0,
    e))

# a progress bar to keep track during a run
pbY <- progress::progress_bar$new(total = (e * length(gamma_vec)))
set.seed(123)
avgGammaMle <- data.frame()
avgGammaMetalog <- data.frame()


for (z in 1:l) {
    gamma <- gamma_vec[z]

    # these Params object are for holding the distribution
    # current state during an epoch
    mleParams <- data.frame(prob_threat = rep(0.5, gateSize),
        threat_mean = rep(1, gateSize), nonthreat_mean = rep(1,
            gateSize), n = rep(0, gateSize), n_threat = rep(0,
            gateSize), n_nonthreat = rep(0, gateSize), threat_var = rep(1,
            gateSize), nonthreat_var = rep(1, gateSize))

    betaParams <- data.frame(alpha = rep(1, gateSize), beta = rep(1,
        gateSize), threat_mean = rep(-1, gateSize), threat_sd = rep(1,
        gateSize), nonthreat_mean = rep(1, gateSize), nonthreat_sd = rep(1,
        gateSize), n = rep(0, gateSize), known_var = rep(1,
        gateSize))

    metalogParams <- list()
    metalogParams$data <- data.frame(gate = c(), obs = c())
    metalogParams$priors <- list()

    # these object are to track the position of the gate in
    # a specific epoch
    gatePositionMLE <- data.frame(position = rep(0, gateSize),
        rewards = rep(0, gateSize))
    gatePositionThompson <- data.frame(position = rep(0,
        gateSize), rewards = rep(0, gateSize))
```

```
gatePositionMetalog <- data.frame(position = rep(0,
    gateSize), rewards = rep(0, gateSize), q10 = rep(0,
    gateSize), q50 = rep(0, gateSize), q90 = rep(0,
    gateSize))

# collect the rewards in each epoch for each strategy
periodRewards <- data.frame(period = seq(1, e, 1), rewardsMLE = rep(0,
    e), openGateMLE = rep(0, e), rewardsThompson = rep(0,
    e), openGateThompson = rep(0, e), rewardsMetalog = rep(0,
    e), openGateMetalog = rep(0, e))

avgBetaParams <- data.frame()
avgMleParams <- data.frame()
avgMetalogParams <- data.frame()


for (t in 1:e) {
    pbY$tick()
    ##### Thompson Sampling with Beta/Binomial ######
    myList <- singleTimeBetaBinomialThompsonSample(threatProbVector,
        actorRewards, gatePositionThompson, betaParams,
        0)

    gatePositionThompson <- myList$gatePositionThompson
    betaParams <- myList$betaParams
    # collect some data about the the beta/binomial/normal
    # case
    tempBetaParams <- data.frame(time = rep(t, gateSize),
        gate = seq(1, gateSize, 1), alpha = betaParams$alpha,
        beta = betaParams$beta, threat_mean = betaParams$threat_mean,
        threat_sd = betaParams$threat_sd, nonthreat_mean = betaParams$nonthreat_mean,
        nonthreat_sd = betaParams$nonthreat_sd, n = betaParams$n)
    tempBetaParams <- cbind(tempBetaParams, gatePositionThompson)
    avgBetaParams <- rbind(avgBetaParams, tempBetaParams)
    periodRewards$rewardsThompson[t] <- sum(gatePositionThompson[,
        2])
    periodRewards$openGateThompson[t] <- sum(gatePositionThompson[,
        1])
    ##### MLE with Epsilon Greedy ######
    myList <- singleTimeMLESample(threatProbVector,
        actorRewards, gatePositionMLE, mleParams, epsilon,
        gamma)

    gatePositionMLE <- myList$gatePositionMLE
    mleParams <- myList$mleParams

    tempMleParams <- data.frame(time = rep(t, gateSize),
        gate = seq(1, gateSize, 1), prob_threat = mleParams$prob_threat,
        threat_mean = mleParams$threat_mean, nonthreat_mean = mleParams$nonthreat_mean,
        gamma = gamma)
    tempMleParams <- cbind(tempMleParams, gatePositionMLE)
    avgMleParams <- rbind(avgMleParams, tempMleParams)
    periodRewards$rewardsMLE[t] <- sum(gatePositionMLE[,
        2])
    periodRewards$openGateMLE[t] <- sum(gatePositionMLE[,
        1])

    ####### Thompson Samling with Metalog #####
    myList <- singleTimeMetalogSample(threatProbVector,
        actorRewards, gatePositionMetalog, metalogParams,
        gamma)

    gatePositionMetalog <- myList$gatePositionMetalog
    metalogParams <- myList$metalogParams

    # metalog params
    gate_median <- rep(1, gateSize)
    gate_median_uncertainty <- rep(1, gateSize)
    gate_lower_quantile <- rep(1, gateSize)

    if (t > 6) {
        gate_median <- c()
        gate_median_uncertainty <- c()
        gate_lower_quantile <- c()
        for (y in 1:gateSize) {
            gate_median <- c(gate_median, metalogParams$priors[[y]]$params$bayes$q_bar[2])

            gate_median_uncertainty <- c(gate_median_uncertainty,
              sqrt(metalogParams$priors[[y]]$params$bayes$sig[2,
                2]))

            gate_lower_quantile <- c(gate_lower_quantile,
              qmetalog(metalogParams$priors[[y]], y = c(0.05),
                term = 6))
        }
    }
    tempMetalogParams <- data.frame(time = rep(t, gateSize),
        gate = seq(1, gateSize, 1), gate_median = gate_median,
        gate_median_uncertainty = gate_median_uncertainty,
        gate_lower_quantile = gate_lower_quantile, gamma = gamma)

    tempMetalogParams <- cbind(tempMetalogParams, gatePositionMetalog)
    avgMetalogParams <- rbind(avgMetalogParams, tempMetalogParams)
    periodRewards$rewardsMetalog[t] <- sum(gatePositionMetalog[,
        2])
    periodRewards$openGateMetalog[t] <- sum(gatePositionMetalog[,
        1])

    # ############### #just clean up the work space
    rm(tempMetalogParams, tempMleParams, tempBetaParams,
        gate_lower_quantile, gate_median, gate_median_uncertainty)
    # #update reward values


}  #close episode loop
```

```
    # collect rewards for averaging after each epoch
    avgRewards$avgRewardsMLE <- ((avgRewards$avgRewardsMLE *
        (z - 1) + periodRewards$rewardsMLE)/z)
    avgGammaMle <- rbind(avgGammaMle, avgMleParams)
    avgGammaMetalog <- rbind(avgGammaMetalog, avgMetalogParams)

    avgRewards$avgRewardsThompson <- ((avgRewards$avgRewardsThompson *
        (z - 1) + periodRewards$rewardsThompson)/z)

    avgRewards$avgRewardsMetalog <- ((avgRewards$avgRewardsMetalog *
        (z - 1) + periodRewards$rewardsMetalog)/z)
}  #close loop for averaging
rm(metalogParams, mleParams, betaParams, myList, pbY, actorRewards,
    gatePositionMetalog, gatePositionMLE, gatePositionThompson,
    e, l, epsilon, gamma_vec, gateSize, threatProbVector)

### Exploring a single episode ######

gate_position_metalog <- avgMetalogParams %>% mutate(position = if_else(position ==
    1, "open", "close")) %>% select(time, gate, position) %>%
    mutate(model = "metalog")

gate_position_bb <- avgBetaParams %>% mutate(position = if_else(position ==
    1, "open", "close")) %>% select(time, gate, position) %>%
    mutate(model = "beta_binomial")

gate_position <- avgMleParams %>% mutate(position = if_else(position ==
    1, "open", "close")) %>% select(time, gate, position) %>%
    mutate(model = "mle")

gate_position <- rbind(gate_position_metalog, gate_position_bb,
    gate_position)
rm(gate_position_bb, gate_position_metalog)

ggplot((gate_position %>% filter(time > 6)), aes(x = time,
    y = gate, fill = as.factor(position))) + geom_tile() +
    scale_fill_manual(values = c("#cb181d", "#c7e9c0")) +
    theme_bw() + theme(legend.title = element_blank()) +
    facet_wrap(~model, ncol = 1)

rm(gate_position)

# Uncertainty Exploration

p1 <- ggplot((avgMetalogParams %>% filter(time > 6)), aes(x = time,
    y = gate, fill = gate_median_uncertainty)) + geom_tile() +
    scale_fill_gradient2(low = "#e0ecf4", high = "#810f7c") +
    theme_bw() + theme(legend.title = element_blank())

p2 <- ggplot((avgMetalogParams %>% filter(time > 6)), aes(x = time,
    y = gate, fill = gate_lower_quantile)) + geom_tile() +
    scale_fill_gradient2(low = "#810f7c", high = "#e0ecf4") +
    theme_bw() + theme(legend.title = element_blank())
```

```r
plot_grid(p1, p2, labels = c("E", "A"), ncol = 1)

gate_position_gamma <- avgGammaMle %>% mutate(position = if_else(position ==
    1, "open", "close")) %>% select(time, gate, position,
    gamma) %>% mutate(model = "mle")

ggplot(gate_position_gamma, aes(x = time, y = gate, fill = as.factor(position))) +
    geom_tile() + scale_fill_manual(values = c("#cb181d",
    "#c7e9c0")) + theme_bw() + theme(legend.title = element_blank()) +
    facet_wrap(~gamma, ncol = 1)

# plot for expert human thresholds
avgBetaParams$var <- avgBetaParams$alpha * avgBetaParams$beta/((avgBetaParams$alpha +
    avgBetaParams$beta)^2 * (avgBetaParams$alpha + avgBetaParams$beta +
    1))

avgBetaParams$sd <- sqrt(avgBetaParams$var)
avgBetaParams$Decision_Threshold <- "Robot"
avgBetaParams$Decision_Threshold[avgBetaParams$sd > 0.15] <- "Expert"
avgBetaParams$Decision_Threshold <- as.factor(avgBetaParams$Decision_Threshold)
ggplot(avgBetaParams, aes(x = time, y = gate, fill = Decision_Threshold)) +
    geom_tile() + scale_fill_brewer(palette = "Set1") +
    theme_bw()

voi_metalog <- avgMetalogParams %>% mutate(q10 = if_else(q10 >
    0, q10, 0)) %>% mutate(q10 = if_else(q50 > 0, q50, 0)) %>%
    mutate(q10 = if_else(q90 > 0, q90, 0)) %>% mutate(voi = ((0.1 *
    q10 + 0.8 * q50 + 0.1 * q90) - gate_median)) %>% filter(time >
    6) %>% filter(gate < 13 & gate > 4)

ggplot(voi_metalog, aes(x = time, y = voi, color = as.factor(gate))) +
    geom_line(size = 2) + scale_color_brewer(palette = "Set1") +
    xlab("Time Period") + ylab("Value of Information") +
    theme_bw() + theme(legend.title = element_blank())


#### active learning plot####
load("~/analysis/data/first_2000.RData")

# calculate regret
thompson_rewards <- data.frame(period = avgRewards$period,
    reward = (5.2 - avgRewards$avgRewardsThompson), model = "Thompson Beta_Binomial")

mle_rewards <- data.frame(period = avgRewards$period, reward = (5.2 -
    avgRewards$avgRewardsMLE), model = "MLE")

metalog_rewards <- data.frame(period = avgRewards$period,
    reward = (5.2 - avgRewards$avgRewardsMetalog), model = "Metalog")

totalRewards <- rbind(thompson_rewards, mle_rewards, metalog_rewards)
totalRewards <- total_rewards_first_2000 %>% mutate(model = ifelse(as.character(model) ==
    "Thompson Beta_Binomial", "Beta_Binomial", as.character(model))) %>%
    mutate(period = as.numeric(period))

total_rewards_first_2000 <- total_rewards_first_2000 %>%
    mutate(model = if_else(as.character(model) == "Thompson Beta_Binomial",
        "Beta Binomial", as.character(model))) %>% mutate(model = factor(model,
    levels = c("MLE", "Beta Binomial", "Metalog")))
ggplot(total_rewards_first_2000, aes(x = period, y = reward2,
    colour = model)) + geom_line() + xlab("Time Period") +
    ylab("Average Cost Premium") + theme_bw()
```

Unknown Gate Set

```r
# Large Gate Sets using supervised learning I'm going to
# crush this :)


library(rmetalog)
library(parsnip)
library(tidyverse)
library(progress)
library(tidymodels)

# pull in the honeypot data from the EDA
load("~/analysis/data/honey_munged.RData")

ports <- read.csv("~/analysis/data/service-names-port-numbers.csv")

ports <- ports[grep("-", ports$Port.Number, invert = TRUE),
    ] %>% mutate(Port.Number = as.numeric(Port.Number))

ports_assigned <- ports %>% filter(Description != "Unassigned")

# only keep the blacklisted IPs that are identified
# later
honeypot <- honeypot.data_date_trim %>% select(source_ip,
    protocol, source_port, destination_port, threat, hour,
    Date_actual, day, country_name, region, city, timestampProper) %>%
    mutate_at(vars(1:5), as.character) %>% mutate_at(vars(3,
    4), as.numeric) %>% mutate(day_of_month = as.numeric(format(Date_actual,
    "%d"))) %>% select(-Date_actual) %>% mutate(source_ip_keep = source_ip) %>%
    separate(source_ip, into = c("first_oct", "second_oct",
        "third_oct", "fourth_oct"), extra = "drop") %>%
    mutate_at(vars(1:4), as.numeric) %>% mutate(threat = as.logical(threat)) %>%
    mutate(destination_port_assigned = (destination_port %in%
        ports_assigned$Port.Number))
# some variables removed for formating see repo for
# complete code
str(honeypot)

# remove the original data
rm(honeypot.data_date_trim, ports, ports_assigned)
```

```r
save(honeypot, file = "~/analysis/data/honeypot_munged_more.RData")

ips <- unique(honeypot$source_ip_keep)

pbY <- progress::progress_bar$new(total = (length(ips)))
analysis_data <- data.frame()
for (i in ips) {
    pbY$tick()

    temp_honeypot <- honeypot %>% filter(source_ip_keep ==
        i)

    # protocol
    mpp <- as.data.frame(table(temp_honeypot$protocol)) %>%
        arrange(Freq)
    mpp <- as.character(mpp$Var1[1])

    # source ports
    sp_df <- as.data.frame(table(temp_honeypot$source_port_type))
    num_system_s <- nrow(sp_df[sp_df$Var1 == "system", ])
    num_user_s <- nrow(sp_df[sp_df$Var1 == "user", ])
    num_dynamic_s <- nrow(sp_df[sp_df$Var1 == "dynamic",
        ])
    num_unknown_s <- nrow(sp_df[sp_df$Var1 == "unknown",
        ])
    num_unassigned_s <- nrow(temp_honeypot[temp_honeypot$source_port_assigned ==
        FALSE, ])
    spp <- length(unique(temp_honeypot$source_port))
    if (nrow(sp_df) == 0)
        spp <- 0
    if (length(num_system_s) == 0)
        num_system_s <- 0
    if (length(num_user_s) == 0)
        num_user_s <- 0
    if (length(num_dynamic_s) == 0)
        num_dynamic_s <- 0
    if (length(num_unknown_s) == 0)
        num_unknown_s <- 0
    if (length(num_unassigned_s) == 0)
        num_unassigned_s <- 0

    # destination ports
    dp_df <- as.data.frame(table(temp_honeypot$destination_port_type))
    num_system_d <- nrow(dp_df[dp_df$Var1 == "system", ])
    num_user_d <- nrow(dp_df[dp_df$Var1 == "user", ])
    num_dynamic_d <- nrow(dp_df[dp_df$Var1 == "dynamic",
        ])
    num_unknown_d <- nrow(dp_df[dp_df$Var1 == "unknown",
        ])
    num_unassigned_d <- nrow(temp_honeypot[temp_honeypot$destination_port_assigned ==
        FALSE, ])
    if (length(num_system_d) == 0)
        num_system_d <- 0
    if (length(num_user_d) == 0)
        num_user_d <- 0
    if (length(num_dynamic_d) == 0)
        num_dynamic_d <- 0
    if (length(num_unknown_d) == 0)
        num_unknown_d <- 0
    if (length(num_unassigned_d) == 0)
        num_unassigned_d <- 0

    mdp <- as.data.frame(table(temp_honeypot$destination_port)) %>%
        arrange(Freq)
    dp1 <- as.character(mdp$Var[1])
    dp2 <- as.character(mdp$Var[2])
    dpp <- length(unique(temp_honeypot$destination_port))
    if (length(dp1) == 0)
        dp1 <- "unknown"
    if (is.na(dp1))
        dp1 <- "unknwon"
    if (length(dp2) == 0)
        dp2 <- "unknown"
    if (is.na(dp2))
        dp2 <- "unknown"
    if (length(dp1) == 0 & length(dp2) == 0)
        dpp <- 0

    # calculate the summary statistics
    tf <- nrow(temp_honeypot)
    ah <- mean(temp_honeypot$hour)
    ad <- mean(temp_honeypot$day_of_month)
    # this is a bit hacky
    awd <- mean(as.numeric(temp_honeypot$day))

    # inter arrival times
    ir <- 0
    tir <- 0

    if (tf > 1) {
        temp_honeypot <- temp_honeypot %>% arrange(timestampProper) %>%
            mutate(ir = as.numeric(timestampProper - lag(timestampProper,
                default = 0)))
        temp_honeypot$ir[1] <- 0
        ir <- mean(temp_honeypot$ir)
        tir <- mean(temp_honeypot$ir, trim = 0.1)
    }

    temp <- data.frame(first_oct = temp_honeypot$first_oct[1],
        second_oct = temp_honeypot$second_oct[1], third_oct = temp_honeypot$third_oct[1],
        fourth_oct = temp_honeypot$fourth_oct[1], country_name = temp_honeypot$country_name[1],
        region = temp_honeypot$region[1], threat = temp_honeypot$threat[1],
        most_pop_protocol = mpp, num_system_s = num_system_s,
        num_user_s = num_user_s, num_dynamic_s = num_dynamic_s,
        num_unknown_s = num_unknown_s, num_unassigned_s = num_unassigned_s,
        source_port_promisc = spp, num_system_d = num_system_d,
        num_user_d = num_user_d, num_dynamic_d = num_dynamic_d,
        num_unknown_d = num_unknown_d, num_unassigned_d = num_unassigned_d,
        frist_pop_dest_port = dp1, second_pop_dest_port = dp2,
        source_port_promisc = spp, dest_port_promisc = dpp,
        total_freq = tf, avg_hour = ah, avg_weekday = awd,
        avg_day = ad, interrarival = ir, trimmed_interarrival = tir)

    analysis_data <- rbind(analysis_data, temp)
    rm(mpp, dp1, dp2, spp, dpp, tf, ah, awd, ad, ir, tir,
        mdp, temp_honeypot, i, temp, num_system_s, num_user_s,
        num_dynamic_s, num_unknown_s, num_unassigned_s,
        num_system_d, num_user_d, num_dynamic_d, num_unknown_d,
        num_unassigned_d)
}

rm(ips, honeypot)

## start the predicitive modeling component here ######
## clean up the data
analysis_temp <- analysis_data
analysis_data <- analysis_data %>% mutate_at(vars(5:8, 20,
    21), as.factor) %>% select(-region)
# the region is a bit too specific so it causes some
# model issues replacing missing data with the highest
# frequency observation
for (i in 1:ncol(analysis_data)) {
    temp <- as.data.frame(table(analysis_data[, i])) %>%
        arrange(desc(Freq))
    analysis_data[, i] <- analysis_data[, i] %>% replace_na(temp$Var1[1])
    if (class(analysis_data[, i]) == "factor") {
        analysis_data[, i] <- droplevels(analysis_data[,
            i])
    }

}

# split the data for training and testing
set.seed(12)
data_split <- initial_split(analysis_data, strata = "threat",
    p = 0.975)

analysis_train <- training(data_split)
analysis_test <- testing(data_split)

# logistic regression

log_reg <- logistic_reg() %>% set_engine(engine = "glm") %>%
    fit(threat ~ ., data = analysis_data)

# knn
library(kknn)

# 4 nearest
```

```
knn_10 <- nearest_neighbor(mode = "classification", neighbors = 10) %>%
    set_engine(engine = "kknn") %>% fit(threat ~ ., data = analysis_data)

knn_8 <- nearest_neighbor(mode = "classification", neighbors = 8) %>%
    set_engine(engine = "kknn") %>% fit(threat ~ ., data = analysis_data)

knn_6 <- nearest_neighbor(mode = "classification", neighbors = 10) %>%
    set_engine(engine = "kknn") %>% fit(threat ~ ., data = analysis_data)

knn_4 <- nearest_neighbor(mode = "classification", neighbors = 4) %>%
    set_engine(engine = "kknn") %>% fit(threat ~ ., data = analysis_data)

knn_2 <- nearest_neighbor(mode = "classification", neighbors = 2) %>%
    set_engine(engine = "kknn") %>% fit(threat ~ ., data = analysis_data)

# random forest
library(randomForest)

rf_100 <- rand_forest(mode = "classification", trees = 100) %>%
    set_engine(engine = "randomForest") %>% fit(threat ~
    ., data = analysis_data)

rf_250 <- rand_forest(mode = "classification", trees = 250) %>%
    set_engine(engine = "randomForest") %>% fit(threat ~
    ., data = analysis_data)

rf_500 <- rand_forest(mode = "classification", trees = 500) %>%
    set_engine(engine = "randomForest") %>% fit(threat ~
    ., data = analysis_data)

rf_750 <- rand_forest(mode = "classification", trees = 750) %>%
    set_engine(engine = "randomForest") %>% fit(threat ~
    ., data = analysis_data)

rf_1000 <- rand_forest(mode = "classification", trees = 1000) %>%
    set_engine(engine = "randomForest") %>% fit(threat ~
    ., data = analysis_data)

# shallow network
library(nnet)

nn_1 <- mlp(mode = "classification", hidden_units = 1, epochs = 500) %>%
    set_engine(engine = "nnet") %>% fit(threat ~ ., data = analysis_data)

nn_2 <- mlp(mode = "classification", hidden_units = 2, epochs = 500) %>%
    set_engine(engine = "nnet") %>% fit(threat ~ ., data = analysis_data)

nn_3 <- mlp(mode = "classification", hidden_units = 3, epochs = 500) %>%
    set_engine(engine = "nnet") %>% fit(threat ~ ., data = analysis_data)

models <- list()
models$log_reg <- log_reg
models$knn_10 <- knn_10
```

```r
models$knn_8 <- knn_8
models$knn_6 <- knn_6
models$knn_4 <- knn_4
models$knn_2 <- knn_2
models$rf_100 <- rf_100
models$rf_250 <- rf_250
models$rf_500 <- rf_500
models$rf_750 <- rf_750
models$rf_1000 <- rf_1000
models$nn_1 <- nn_1
models$nn_2 <- nn_2
models$nn_3 <- nn_3
# clean up
rm(nn_1, nn_2, nn_3, rf_100, rf_1000, rf_250, rf_500, rf_750,
    knn_10, knn_2, knn_4, knn_6, knn_8, log_reg)

models_roc_auc <- data.frame()
models_predictions <- data.frame()
for (i in 1:length(models)) {
    model_result <- data.frame(threat = analysis_data$threat,
        preds = predict(models[[i]], new_data = analysis_data)[1],
        probs = predict(models[[i]], new_data = analysis_data,
            type = "prob")[, 2], model = names(models)[i])
    colnames(model_result) <- c("threat", "preds", "probs",
        "model")
    # collect the accuracy result <- model_result %>%
    # roc_auc(truth = threat, probs) result$model <-
    # names(models)[i] collect the predictions by model
    # models_roc_auc <- rbind(models_roc_auc,result)
    models_predictions <- rbind(models_predictions, model_result)
}

rm(result, data_split, model_result, i)

# pca plot for explanation of knn result
analysis_temp <- analysis_data %>% mutate_all(as.numeric)
pca <- prcomp(analysis_temp)

plot_pca_data <- cbind(as.data.frame(pca$x[, 1:2]), labels = analysis_data$threat)

ggplot(plot_pca_data, aes(x = PC1, y = PC2, colour = labels)) +
    scale_color_manual(values = c("blue", "red")) + geom_point() +
    theme_bw()
rm(pca, plot_pca_data)

#### explaining the features ######
library(lime)
library(caret)
# list the models
for (i in 1:length(models)) {
    print(i)
    print(class(models[[i]]))
}
```

```r
# lime has performance issues working on a large data
# set with many features... something to consider
# improving # Create an explainer object explainer <-
# lime(analysis_data, models[[7]]) # Explain new
# observation explanation <- explain(analysis_data,
# explainer, n_labels = 1, n_features = 2)

# can use important features from the random forest
# models

varImpPlot(models[[11]]$fit, type = 2)

imp_features <- varImp(models[[11]]$fit, type = 2)

imp_features_trim <- imp_features %>% mutate(model = rownames(imp_features)) %>%
    arrange(desc(Overall)) %>% slice(1:25)

#### uncertainty of predictions #####
unique_models <- unique(models_predictions$model)

for (i in 1:length(models)) {
    temp <- models_predictions %>% filter(model == unique_models[i])
    colnames(temp) <- c(paste0(unique_models[i], "_threat"),
        paste0(unique_models[i], "_preds"), paste0(unique_models[i],
            "_probs"), paste0(unique_models[i], "_model"))
    analysis_data <- cbind(analysis_data, temp)
}
rm(temp)
# use metalog to build the uncertainty plots
library(rmetalog)

locations <- c("United States", "Brazil", "Korea, Republic of",
    "China")

uncert_data <- data.frame()
for (locs in locations) {
    temp <- analysis_data %>% filter(country_name == locs &
        threat == TRUE) %>% select(contains("prob"))
    temp <- c(as.matrix(temp))
    temp <- temp[!temp %in% c(0, 1)]

    my_metalog <- metalog(temp, term_limit = 9, boundedness = "sl",
        bounds = 0)

    temp <- data.frame(probs = my_metalog$M$y, dens = my_metalog$M$M9,
        loc = locs)
    uncert_data <- rbind(uncert_data, temp)
    # print(plot(p,s,type='l',main = locs))
}

ggplot(uncert_data, aes(x = dens, y = probs, color = loc)) +
    geom_line() + ylab("Probability") + xlab("Model Probability") +
    scale_color_brewer(palette = "Set1", name = "") + theme_bw()
```

```r
### gate policy ######

locations <- c("United States", "Brazil", "Korea, Republic of",
    "Taiwan, Province of China", "Russian Federation", "France",
    "Netherlands", "Switzerland", "India")

for (loc in locations) {
    temp <- analysis_data %>% filter(country_name == loc)
    print(loc)
    print(table(temp$threat))
}

### evaluate thresholds for continuous variables ####
analysis_sub <- analysis_data %>% filter(!(country_name %in%
    locations))

threats <- analysis_sub %>% filter(threat == TRUE)

# first oct
first_oct_freq <- as.data.frame(table(analysis_sub$first_oct),
    stringsAsFactors = FALSE) %>% mutate(Var2 = (Var1 %in%
    threats$first_oct)) %>% arrange(desc(Freq))

rm(threats)

percent_threat_by_freq <- data.frame()
for (i in 1:max(first_oct_freq$Freq)) {
    temp <- first_oct_freq %>% filter(Freq >= i)
    p <- sum(as.numeric(temp$Var2))/nrow(temp)
    temp <- data.frame(freq = i, percent_threat = p)
    percent_threat_by_freq <- rbind(percent_threat_by_freq,
        temp)
}

ggplot(percent_threat_by_freq, aes(x = freq, y = percent_threat)) +
    geom_line() + scale_x_log10()

first_oct_threshold <- first_oct_freq %>% filter(Freq >=
    1000)

analysis_sub2 <- analysis_sub %>% filter(!(first_oct %in%
    first_oct_threshold$Var1))

table(analysis_sub$threat)
table(analysis_sub2$threat)

# interarrival
threats <- analysis_sub %>% filter(threat == TRUE)

first_oct_freq <- as.data.frame(table(analysis_sub$interrarival),
    stringsAsFactors = FALSE) %>% mutate(Var2 = (Var1 %in%
    threats$interrarival)) %>% arrange(desc(Freq))
```

```
rm(threats)

percent_threat_by_freq <- data.frame()
for (i in 1:max(analysis_sub$interrarival)) {
    temp <- analysis_sub %>% filter(interrarival >= i)
    p <- sum(as.numeric(temp$Var2))/nrow(temp)
    temp <- data.frame(freq = i, percent_threat = p)
    percent_threat_by_freq <- rbind(percent_threat_by_freq,
        temp)
}

ggplot(percent_threat_by_freq, aes(x = freq, y = percent_threat)) +
    geom_line() + scale_x_log10()

first_oct_threshold <- first_oct_freq %>% filter(Freq >=
    1000)

analysis_sub2 <- analysis_sub %>% filter(!(first_oct %in%
    first_oct_threshold$Var1))

table(analysis_sub$threat)
table(analysis_sub2$threat)
```

Bibliography

[1] Diana Kelley and Ron Moritz. Best practices for building a security operations center. *Information Systems Security*, 14(6):27–32, 2006.

[2] Sasha Romanosky, Lilian Ablon, Andreas Kuehn, and Therese Jones. Content analysis of cyber insurance policies: How do carriers write policies and price cyber risk? 2017.

[3] Cat Zakrzewski. The evolution of a cybersecurity firm. 2017. URL `https://www.wsj.com/articles/the-evolution-of-a-cybersecurity-firm-1494986640`.

[4] Sasha Romanosky. Examining the costs and causes of cyber incidents. *Journal of Cybersecurity*, 2(2):121–135, 2016.

[5] Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *Security & Privacy, IEEE*, 9(3):49–51, 2011.

[6] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *null*, pages 121–130. IEEE, 2006.

[7] Evan E Anderson and Joobin Choobineh. Enterprise information security strategies. *Computers & Security*, 27(1):22–29, 2008.

[8] Anna Sperotto, Gregor Schaffrath, Ramin Sadre, Cristian Morariu, Aiko Pras, and Burkhard Stiller. An overview of ip flow-based intrusion detection. *IEEE communications surveys & tutorials*, 12(3):343–356, 2010.

[9] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. A survey of intrusion detection techniques in cloud. *Journal of Network and Computer Applications*, 36(1):42–57, 2013.

[10] Pedro Garcia-Teodoro, J Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1):18–28, 2009.

[11] Charlie Kaufman, Radia Perlman, and Mike Speciner. *Network security: private communication in a public world*. Prentice Hall Press, 2002.

[12] William Stallings. *Cryptography and network security: principles and practices.* Pearson Education India, 2006.

[13] Gary McGraw. Software security. *IEEE Security & Privacy*, 2(2):80–83, 2004.

[14] Paulo Shakarian. The 2008 russian cyber campaign against georgia. *Military review*, 91(6):63, 2011.

[15] Richard A Clarke and Robert K Knake. *Cyber war.* HarperCollins, 2011.

[16] Jeffrey Carr. *Inside cyber warfare: Mapping the cyber underworld.* " O'Reilly Media, Inc.", 2011.

[17] Brian B Kelly. Investing in a centralized cybersecurity infrastructure: Why hacktivism can and should influence cybersecurity reform. *BUL Rev.*, 92:1663, 2012.

[18] Hal Berghel. Oh, what a tangled web: Russian hacking, fake news, and the 2016 us presidential election. *computer*, 50(9):87–91, 2017.

[19] Brian Krebs. Shadowy russian firm seen as conduit for cybercrime. 2007. URL http://www.washingtonpost.com/wpdyn/content/article/2007/10/12/AR2007101202461_pf.html.

[20] Stephan Haggard and Jon R Lindsay. North korea and the sony hack: exporting instability through cyberspace. 2015.

[21] Wenye Wang and Zhuo Lu. Cyber security in the smart grid: Survey and challenges. *Computer Networks*, 57(5):1344–1371, 2013.

[22] Jinendra Ranka. National cyber range. Technical report, DEFENSE ADVANCED RESEARCH PROJECTS AGENCY ARLINGTON VA STRATEGIC TECHNOLOGY OFFICE (STO), 2011.

[23] Joseph Werther, Michael Zhivich, Tim Leek, and Nickolai Zeldovich. Experiences in cyber security education: The mit lincoln laboratory capture-the-flag exercise. In *CSET*, 2011.

[24] Patricia Derler, Edward A Lee, and Alberto Sangiovanni Vincentelli. Modeling cyber–physical systems. *Proceedings of the IEEE*, 100(1):13–28, 2012.

[25] Peter Mell, Karen Scarfone, and Sasha Romanosky. Common vulnerability scoring system. *IEEE Security & Privacy*, 4(6), 2006.

[26] Peng Xie, Jason H Li, Xinming Ou, Peng Liu, and Renato Levy. Using bayesian networks for cyber security analysis. In *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, pages 211–220. IEEE, 2010.

[27] Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. Dynamic security risk management using bayesian attack graphs. *IEEE Transactions on Dependable and Secure Computing*, 9(1):61–74, 2012.

[28] Jason Jaskolka, Ridha Khedri, and Qinglei Zhang. Endowing concurrent kleene algebra with communication actions. In *International Conference on Relational and Algebraic Methods in Computer Science*, pages 19–36. Springer, 2014.

[29] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M Wing. Automated generation and analysis of attack graphs. In *null*, page 273. IEEE, 2002.

[30] Subil Abraham and Suku Nair. A novel architecture for predictive cybersecurity using non-homogenous markov models. In *Trustcom/BigDataSE/ISPA, 2015 IEEE*, volume 1, pages 774–781. IEEE, 2015.

[31] Subil Abraham and Suku Nair. Cyber security analytics: a stochastic model for security quantification using absorbing markov chains. *Journal of Communications*, 9(12):899–907, 2014.

[32] Subil Abraham and Suku Nair. Predictive cyber-security analytics framework: A non-homogenous markov model for security quantification. *arXiv preprint arXiv:1501.01901*, 2015.

[33] Wei Li and Rayford B Vaughn. Cluster security research involving the modeling of network exploitations using exploitation graphs. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 2, pages 26–26. IEEE, 2006.

[34] Arpan Roy, Dong Seong Kim, and Kishor S Trivedi. Attack countermeasure trees (act): towards unifying the constructs of attack and defense trees. *Security and Communication Networks*, 5(8):929–943, 2012.

[35] Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Toward measuring network security using attack graphs. In *Proceedings of the 2007 ACM workshop on Quality of protection*, pages 49–54. ACM, 2007.

[36] Nwokedi Idika and Bharat Bhargava. Extending attack graph-based security metrics and aggregating their application. *IEEE Transactions on Dependable and Secure Computing*, 9(1):75–85, 2012.

[37] Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, 1999.

[38] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238, 1999.

[39] Kenneth Ingham and Stephanie Forrest. A history and survey of network firewalls. *University of New Mexico, Tech. Rep*, 2002.

[40] Harsh Taneja and Angela Xiao Wu. Does the great firewall really isolate the chinese? integrating access blockage with cultural factors to explain web user behavior. *The Information Society*, 30(5):297–309, 2014.

[41] Tim Bass. Intrusion detection systems and multisensor data fusion. *Communications of the ACM*, 43(4):99–105, 2000.

[42] Jon Oberheide, Evan Cooke, and Farnam Jahanian. Rethinking antivirus: Executable analysis in the network cloud. In *HotSec*, 2007.

[43] Anirudh Ramachandran, David Dagon, and Nick Feamster. Can dns-based blacklists keep up with bots? In *CEAS*, 2006.

[44] Christian J Dietrich and Christian Rossow. Empirical research of ip blacklists. In *ISSE 2008 Securing Electronic Business Processes*, pages 163–171. Springer, 2009.

[45] Pawan Prakash, Manish Kumar, Ramana Rao Kompella, and Minaxi Gupta. Phishnet: predictive blacklisting to detect phishing attacks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–5. Citeseer, 2010.

[46] Anirudh Ramachandran, Nick Feamster, and Santosh Vempala. Filtering spam with behavioral blacklisting. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 342–351. ACM, 2007.

[47] Critical Stack. Critical stack open source threat feed. 2017. URL https://intel.criticalstack.com/.

[48] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 129–138. ACM, 2012.

[49] Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, and LiWu Chang. A novel anomaly detection scheme based on principal component classifier. Technical report, DTIC Document, 2003.

[50] Luke AJ ONeill. Immunitys early-warning system. *Scientific American*, 292(1): 38–45, 2005.

[51] Matthieu Bussiere and Marcel Fratzscher. Towards a new early warning system of financial crises. *journal of International Money and Finance*, 25(6):953–973, 2006.

[52] Reid Basher. Global early warning systems for natural hazards: systematic and people-centred. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 364(1845):2167–2182, 2006.

[53] Michelle M Baron and EB Pate-Cornell. Designing risk-management strategies for critical engineering systems. *IEEE Transactions on Engineering Management*, 46 (1):87–100, 1999.

[54] M Elisabeth Paté-Cornell. Learning from the piper alpha accident: A postmortem analysis of technical and organizational factors. *Risk Analysis*, 13(2):215–232, 1993.

[55] Nassim Nicholas Taleb. The black swan. *The Impact of the*, 2007.

[56] Elisabeth Paté-Cornell. On black swans and perfect storms: risk analysis and management when statistics are not enough. *Risk analysis*, 32(11):1823–1833, 2012.

[57] M Elisabeth Paté-Cornell. Warning systems in risk management. *Risk analysis*, 6 (2):223–234, 1986.

[58] ELISABETH PATÉ-CORNELL. On signals, response, and risk mitigation. *Accident Precursor Analysis and Management: Reducing Technological Risk through Diligence*, page 45, 2004.

[59] Tom M Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37): 870–877, 1997.

[60] Anup K Ghosh, Aaron Schwartzbard, and Michael Schatz. Learning program behavior profiles for intrusion detection. In *Workshop on Intrusion Detection and Network Monitoring*, volume 51462, pages 1–13, 1999.

[61] Sumeet Dua and Xian Du. *Data mining and machine learning in cybersecurity.* CRC press, 2016.

[62] Carl Livadas, Robert Walsh, David Lapsley, and W Timothy Strayer. Usilng machine learning technliques to identify botnet traffic. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 967–974. IEEE, 2006.

[63] Ivan Firdausi, Alva Erwin, Anto Satriyo Nugroho, et al. Analysis of machine learning techniques used in behavior-based malware detection. In *Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on*, pages 201–203. IEEE, 2010.

[64] Saeed Abu-Nimeh, Dario Nappa, Xinlei Wang, and Suku Nair. A comparison of machine learning techniques for phishing detection. In *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*, pages 60–69. ACM, 2007.

[65] Sébastien Bubeck, Nicolo Cesa-Bianchi, et al. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 5(1):1–122, 2012.

[66] Ion Androutsopoulos, John Koutsias, Konstantinos V Chandrinos, George Paliouras, and Constantine D Spyropoulos. An evaluation of naive bayesian anti-spam filtering. *arXiv preprint cs/0006013*, 2000.

[67] Jay Jacobs and Bob Rudis. *Data-Driven Security: Analysis, Visualization and Dashboards*. John Wiley & Sons, 2014.

[68] George G Judge, Rufus Carter Hill, William Griffiths, Helmut Lutkepohl, and Tsoung Chao Lee. Introduction to the theory and practice of econometrics. 1982.

[69] Luis Torgo. Data mining with r. *Learning with case studies. CRC, Boca Raton*, 2011.

[70] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–6. IEEE, 2009.

[71] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):262–294, 2000.

[72] Ahmad Javaid, Quamar Niyaz, Weiqing Sun, and Mansoor Alam. A deep learning approach for network intrusion detection system. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pages 21–26. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.

[73] Mahmood Yousefi-Azar, Vijay Varadharajan, Len Hamey, and Uday Tupakula. Autoencoder-based feature learning for cyber security applications. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 3854–3861. IEEE, 2017.

[74] Hee-su Chae, Byung-oh Jo, Sang-Hyun Choi, and Twae-kyung Park. Feature selection for intrusion detection using nsl-kdd. *Recent advances in computer science*, pages 184–187, 2013.

[75] F Eid Heba, Ashraf Darwish, Aboul Ella Hassanien, and Ajith Abraham. Principle components analysis and support vector machine based intrusion detection system. In *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, pages 363–367. IEEE, 2010.

[76] Maryam Feily, Alireza Shahrestani, and Sureswaran Ramadass. A survey of botnet and botnet detection. In *Emerging Security Information, Systems and Technologies, 2009. SECURWARE'09. Third International Conference on*, pages 268–273. IEEE, 2009.

[77] Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer networks*, 51 (12):3448–3470, 2007.

[78] W Timothy Strayer, David Lapsely, Robert Walsh, and Carl Livadas. Botnet detection based on network behavior. In *Botnet detection*, pages 1–24. Springer, 2008.

[79] Joseph Sexton, Curtis Storlie, Joshua Neil, and Alexander Kent. Intruder detection based on graph structured hypothesis testing. In *Resilient Control Systems (ISRCS), 2013 6th International Symposium on*, pages 86–91. IEEE, 2013.

[80] Maryam M Najafabadi, Flavio Villanustre, Taghi M Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. Deep learning applications and challenges in big data analytics. *Journal of Big Data*, 2(1):1, 2015.

[81] Raymond C Borges Hink, Justin M Beaver, Mark A Buckner, Tommy Morris, Uttam Adhikari, and Shengyi Pan. Machine learning for power system disturbance and cyber-attack discrimination. In *Resilient Control Systems (ISRCS), 2014 7th International Symposium on*, pages 1–8. IEEE, 2014.

[82] Douglas W Hill and James T Lynn. Adaptive system and method for responding to computer network security attacks, July 11 2000. US Patent 6,088,804.

[83] Theodora Heather Titonis, Nelson Roberto Manohar-Alers, and Christopher John Wysopal. Automated behavioral and static analysis using an instrumented sandbox and machine learning classification for mobile security, June 6 2017. US Patent 9,672,355.

[84] Tariq Mahmood and Uzma Afzal. Security analytics: Big data analytics for cybersecurity: A review of trends, techniques and tools. In *Information assurance (ncia), 2013 2nd national conference on*, pages 129–134. IEEE, 2013.

[85] Hsinchun Chen, Roger HL Chiang, and Veda C Storey. Business intelligence and analytics: From big data to big impact. *MIS quarterly*, 36(4), 2012.

[86] Dean M Murphy and M Elisabeth Paté-Cornell. The sam framework: Modeling the effects of management factors on human behavior in risk analysis. *Risk analysis*, 16(4):501–515, 1996.

[87] Norbert Wiener. *Cybernetics or Control and Communication in the Animal and the Machine*, volume 25. MIT press, 1961.

[88] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[89] Jean-François Bonnefon, Azim Shariff, and Iyad Rahwan. The social dilemma of autonomous vehicles. *Science*, 352(6293):1573–1576, 2016.

[90] Faisal Kamiran and Toon Calders. Classifying without discriminating. In *Computer, Control and Communication, 2009. IC4 2009. 2nd International Conference on*, pages 1–6. IEEE, 2009.

[91] Atul Gupta and Rex Hammond. Information systems security issues and decisions for small businesses: An empirical examination. *Information management & computer security*, 13(4):297–310, 2005.

[92] Lawrence A Gordon, Martin P Loeb, and Tashfeen Sohail. A framework for using insurance for cyber-risk management. *Communications of the ACM*, 46(3):81–85, 2003.

[93] Herbert B Dixon Jr. Human trafficking and the internet (and other technologies, too). *Judges J.*, 52:36, 2013.

[94] Scott Shackelford, Michael Sulmeyer, Amanda Craig, Ben Buchanan, and Brian Micic. From russia with love: Understanding the russian cyber threat to us critical infrastructure and what to do about it. 2017.

[95] Nikos Virvilis and Dimitris Gritzalis. The big four-what we did wrong in advanced persistent threat detection? In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, pages 248–254. IEEE, 2013.

[96] Thomas W Keelin. The metalog distributions. *Decision Analysis*, 13(4):243–277, 2016.

[97] Gordon V Cormack et al. Email spam filtering: A systematic review. *Foundations and Trends® in Information Retrieval*, 1(4):335–455, 2008.

[98] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[99] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[100] Weiwei Hu and Ying Tan. Generating adversarial malware examples for black-box attacks based on gan. *arXiv preprint arXiv:1702.05983*, 2017.

[101] Wei Yang, Deguang Kong, Tao Xie, and Carl A Gunter. Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, pages 288–302. ACM, 2017.

[102] Ronald A Howard. The foundations of decision analysis. *IEEE transactions on systems science and cybernetics*, 4(3):211–219, 1968.

[103] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[104] Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, and Ian Osband. A tutorial on thompson sampling. *arXiv preprint arXiv:1707.02038*, 2017.

[105] Thomas W Keelin and Bradford W Powley. Quantile-parameterized distributions. *Decision Analysis*, 8(3):206–219, 2011.

[106] Richard Mankiewicz. *The story of mathematics*. Cassell, 2000.