Malware Vectors:
A Technique for Discovering Defense Logics

A Thesis
Submitted to the Faculty
in partial fulfillment of the requirements for the
degree of

Doctor of Philosophy

by

Gabriel Fortunato Stocco

Thayer School of Engineering
Dartmouth College
Hanover, New Hampshire

21 May 2014

Examining Committee:

Chairman ____________________
Dr. George Cybenko

Member ____________________
Dr. Eugene Santos

Member ____________________
Dr. Mark Borsuk

Member ____________________
Dr. Peter Chin

____
F. Jon Kull
Dean of Graduate Studies

ABSTRACT

Organizations face Cyber attacks of increasing sophistication. However, detection measures have not kept up with the pace of advancement in attack design. Common detection systems use detection rules or heuristics based on behaviors of known previous attacks and often crafted manually. The result is a defensive system which is both too sensitive, resulting in many false positives, and not sensitive enough, missing detection of new attacks.

Building upon our work developing the Covertness Capability Calculus, we propose Malware Vectors, a technique for the discovery of defense logic via remote probing. Malware Vectors proposes a technique for building malware by discovering obserables which can be generated without triggering detection. Malware Vectors generates probes to establish a vector of acceptable observable values that the attack may generate without triggering detection. We test attacks against an unknown defense logic and show that it is trivial to discover a covert way to carry out an attack. We extend this simulation to randomly generated defense logics and find that without a change in underlying strategy defenders cannot improve their position significantly. Further, we find that discovery of full logic can be efficiently achieved using only Membership Queries in most cases. Finally, we propose some techniques that a defender could implement to attempt to defend against the Malware Vectors technique.

Acknowledgements

In memory of

Timothy Roy Stocco and Alfredo Fortunato.

With special thanks to

George Cybenko, for your advice and guidance,

Aubrey Zerbach, for your "which" hunts and encouragement,

Mimi Fortunato, for your support.

Additional Acknowledgements in no particular order

Nils Sandell for saving me from zombies, Patrick Sweeney for being a great friend, Mark Borsuk, Eugene Santos, Peter Chin, Steve Kuhn, Phil Alford for high efficiency 1-2-3-4-5-6, Henry Danaher, Finn Danaher, Kate Farris, Louise Cullen for accepting payment for editing in Diet Coke, Al Fortunato for always being there, Evan Fortunato, Michael Fortunato, Gregory and Tiffany Frazier, Tuxedo and Chevy, Kristen Gustavson, Stephanie and Andy Turner, Eric Hansen for giving me a chance. And many other friends, family, fellow students and coworkers who I cannot list here in full.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

As business processes and proprietary information migrate to networked computing systems and even offsite systems, the security of these systems is increasingly critical as the increase in the value of these targets has not gone unnoticed. Today, organizations face increasingly sophisticated Cyber attacks[1]. In response, defenders have turned to systems that perform a task analogous to anti-virus systems - detecting behaviors that match signatures of known bad behaviors.

In this thesis, we propose a technique we call “Malware Vectors” which performs a side-channel[2] attack on logical information leaked by defensive logic based detection systems. We find that these defender systems tend to provide only a trivial amount of protection, at the cost of significant false certainty and high false positive rates. We further find that if we were able to establish a bound between the complexity of operations of “normal” behavior and “illicit” behavior, it would be possible to craft a significantly more effective set of rules that limit false positives.

We take as a given that there is a defender system which operates on a set of logic programmed by the system operators. We propose that these systems can be modeled by a

[1]The use of information technology to infiltrate or disrupt computer systems [1].

[2]A side-channel attack is an attack on the security of a system which relies on a secondary effect of behavior of the system to subvert the primary function of its security implementation [2]. Side-channel attacks are most commonly discussed in cryptography contexts. For example, it is possible to recover the encryption key of a DES cipher by observing how long the cipher takes to run [3].

boolean expression. We will examine how much learning can be done with respect to the underlying logic without access to such logic in addition to how much learning we actually need to do in order to evade such defender logics.

1.1 Outline

Chapter 1 will introduce you to the problem and the real world context in which it exists. In Chapter 2, we will provide terminology information and examples for real world scenarios in greater depth. Additionally, we provide examples of real world defense logics as logical expressions appropriate for our technique. We continue in Chapter 3 with some of our earlier work in on similar problems in this field to provide some contextualization for the work which laid the groundwork for this technique. Chapter 4 explores the analytic underpinnings of this technique, which inform our simulated results. Finally, Chapter 5 contains the description of our simulations and simulated results in depth.

1.2 Landscape

This work was driven by group work on “Covertness Calculus”, a technique we developed to quantify the covertness of a particular piece of malware with respect to a known set of defensive systems. We define covertness of a particular piece of software as the probability that the software will not cause the monitoring software of the system or network on which it is running to issue an alert. Figure 1.1 provides a layout of how this technique works. To quantify covertness we need a model for a defender’s alert logic, a model for the behavior of the malware in question and finally a model to calculate covertness based on the two parameters.

The Covertness Calculus, further discussed in Section 3.3, models a defender’s defensive systems as a logical block diagram. In the diagram, each block represent’s a defender’s

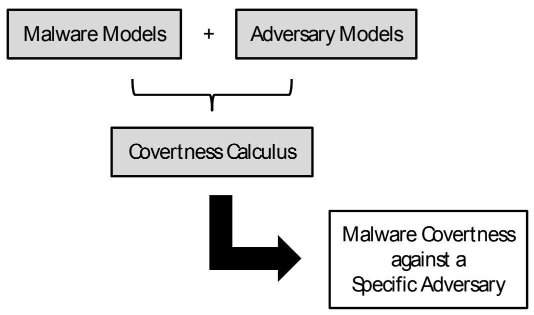


Figure 1.1: *The capability calculus takes as input the defender model and the behavior model of a malware sample and calculates a covertness score.*

sensor which is probabilistically triggered based on observed behavior of a piece of software. By first calculating the probability of each sensor triggering on the behavior of specified malware, and subsequently the reliability of the diagram as a whole, we can quantify the covertness ot the malware.

Defenders can use the Covertness Calculus, as is, to test the capability of their systems to detect known malware, and perhaps to test the efficacy against predicted future attacks. Of course, given the capability calculus model, if an attacker were to possess a model for the defense logic, they would be able to craft particularly covert malware. “Malware Vectors” builds upon the Covertness Calculus by proposing such a technique to disover defense logic for the purpose of covertness testing.

1.2.1 Modern Real World Framework

This section aims to place this work in context with current high-level discussions in the networked computer security, or Cyber domain. In these discussions a few models are used. In particular, we explore the “Cyber Kill Chain”, a technique for qualifying attacker actions.

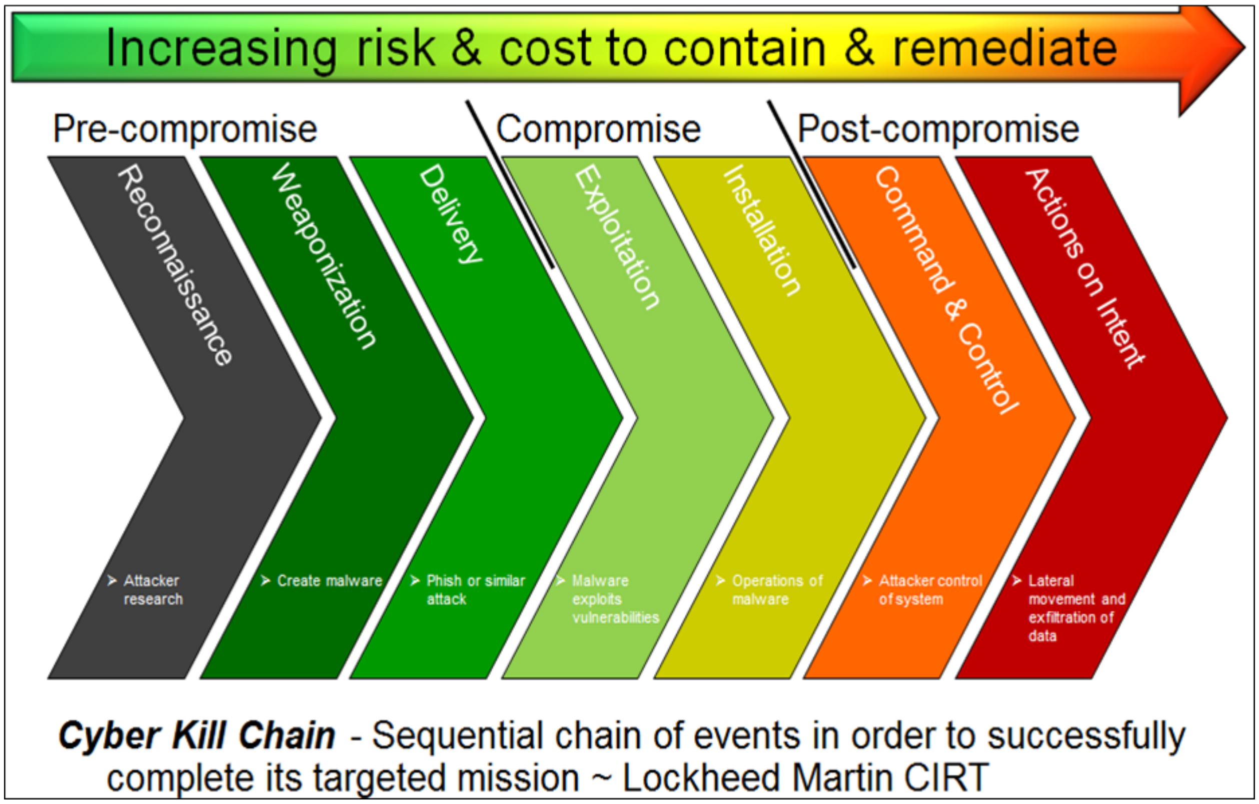

Increasing risk & cost to contain & remediate
Pre-compromise
Compromise
Post-compromise
Reconnaissance
Weaponization
Delivery
Exploitation
Installation
Command & Control
Actions on Intent
Attacker research
Create malware
Phish or similar attack
Malware exploits vulnerabilities
Operations of malware
Attacker control of system
Lateral movement and exfiltration of data
Cyber Kill Chain - Sequential chain of events in order to successfully complete its targeted mission ~ Lockheed Martin CIRT

Figure 1.2: *The Lockheed Martin Cyber Kill Chain [5].*

Cyber Kill Chain

The Cyber Kill Chain is a qualitative categorization technique used for describing the general path that attackers often take to compromise a network. A diagram of the original Cyber Kill chain model as envisioned by Lockheed Martin can be seen in Figure 1.2 [4]. The technique is used to evaluate the responses available to a defender in order to stop the attack. There are a number of different kill chains which have been developed, but most of them are pretty much equivalent. Generally, they have steps similar to: “Reconnaissance”, “Infiltration”, “Propogation”, “Exfiltration”.

The Cyber Kill Chain is used by organizations for two main purposes, first, to analyze weaknesses in existing defenses. Via this analysis, defenders are able to get some qualification as well as quantification of how well their defenses can work against theoretical attacks. This analysis can drive decisions about where to invest resources in order to improve defenses. [6].

Secondly, organizations use the kill chain in triage planning when they suspect they have been seriously compromised [7]. In triage, the use of the Kill Chain allows a defender to quickly eliminate the options that won't work[3] and limit the options to those that attempt to restrict the attacker from making a complete cycle to the exfiltration step[4].

Malware Vectors logically operates primarily at the Reconnaissance level, gathering information about a defender before the formal attack is launched, however, each probe actually generates observables arbitrarily according to what the probe is designed to test. Since each probe generates a set of chosen observables to test whether they are detected by the defender's logic, in fact the observed actions of these probes can span the whole of the kill chain. Conversely, the most traditional Attacker Reconnaissance techniques are network penetration testing and exploit testing against other systems on the network.

1.3 Real World Context

Historically, Cyber attacks have tended to be relatively unfocused. For example, the Conficker worm, detected in 2008, spread to upwards of 9 million machines in 200 countries [8]. The motivation for these attacks ranged from curiosity (in the case of worms), to financial motivations (in the cases of harddrive ransoms and spamnets) to political (in the case of many DDOS attacks).

Recently though, Cyber attacks have become a highly effective component of economic and nation-state espionage. This new generation of attacks is targeted and relatively precise. One such attack is the Stuxnet virus. Discovered in 2010, Stuxnet was specially crafted to attack specific centrifuges controlled by a specific model of centrifuge control system running a specific version of firmware on an air-gapped[5] intranet [10]. This attack has been

[3]For example: those that only affect steps on the Kill Chain that the attacker has already passed.

[4]This of course assumes that exfiltration is the objective of the attacker.

[5]A system or network not directly connected to the outside internet at any time [9].

widely considered to be a turning point for Cyber security incidents involving nation states [11][12].

In a more recent example, Neiman Marcus, a luxury department store based in the United States, experienced a compromise of its point of sale terminals over a period of three and a half months. Over that period of time, Neiman Marcus's Cyber security systems generated over 60,000 alerts relating to this attack. Regardless, system operators overlooked these alerts, because they were being generated about an executable named similarly to the standard point of sale system. Because of this oversight, attackers obtained a large quantity of credit card details. It has further been reported that these 60,000 alerts comprised only 1% of the alerts generated in this time period by Neiman Marcus's security systems [13]. It often is the case that even when a system is able to detect a brazen attack, the rate of false positives causes operators to not trust the output of the system in the first place. Rendering it, perhaps, worse than useless.

1.4 State of the Art Defense Techniques

Given this reality, it would be reasonable to infer that current defense techniques are largely ineffective. The current state-of-the-art in these defenses relies on a set of rule-based logics crafted, often by hand, to detect activities similar to attacks that are already widely known. These rules are implemented via an application such as ArcSight, Splunk or FireEye, which gather data from many different sensors[6], evaluate the "Observables"[7] in the data and trigger alerts when rules based on event counts or statistical properties thereof are violated.

Commonly used defensive systems are rule-based detection logic programs called an "Intrusion Detection System" (IDS) or "Security Information and Event Management"

[6]A few examples of sensors: a network activity monitor, a disk usage monitor,

[7]An observable is a single data point. For example, accessing a file on a machine or sending a packet are observables. When those actions are actually observed by a sensor we call them "events".

```
alert icmp any any -> any any (msg:"ICMP Packet"; sid:1;)
```

Figure 1.3: *Snort ICMP Packet Rule*

(SIEM) system. Some examples of such systems are Snort [14], a NIDS or "Network Intrusion Detection System" and ArcSight [15], a SIEM. These systems collate reports of events occuring on network appliances or host systems and compare statistical properties or raw counts of events to rules chosen by the system operator to generate alerts. These alerts can be programmed to either take immediate action or to prompt a human operator to review the alerts and take appropriate action. The rules are derived from vulnerabilities and attacks that are already known. Adding new rules usually requires some measure of manual configuration [16].

1.4.1 Configuration Example: Snort

When managing end-client machines, a network administrator may desire to monitor any ICMP "pings"[8] to that machine. Generally user machines do not need to allow any unsolicited inbound traffic, and responding to pings thus provides information leakage about the state of the machine, potentially leaving it more vulnerable. Figure 1.3 shows a Snort rule which raises a passive alert[9] whenever a ping packet is detected.

Explaining Figure 1.3 in more depth, this rule will trigger an "alert" whenever an "icmp" (or ping) packet is sent from "any any" (anywhere) to "any any" (anywhere) and print the message "ICMP Packet". To add this rule, a defender would add the rule to the Snort configuration file on their network monitoring machine, and then watch the log file that would be generated for any alerts. We discuss more complex rule creation in Section 2.4.1.

[8] A "ping" is a message a computer can send to another computer to ask "Can you hear me?". A response to a ping affirmatively confirms that there is a machine at the address the ping was sent to.

[9] An operator would have to notice and respond to the alert.

1.5 State of the Art Attack Techniques

For the purposes of discussing an "Attacker" from hereon, we assume that we are discussing a relatively skilled adversary with non-negligible resources. Such an attacker will be interested in systems that are actively monitored and reasonably modern in terms of their security infrastructure.

Though it is relatively low-tech, the state of the art in machine compromise is spearphishing. In a spearphishing attack an attacker will gather some general personal information about a worker or group of workers in the organization whose systems they wish to compromise. For example, they may gather the official title and full names of all of the secretaries. The attacker will then craft an email or social media message that that employee is likely to open, and attach a malicious executable to that email.

From the staging area on this first employee's machine, the malware will attempt to scout the network and then contact a central Command and Control Server. The next step in such an attack would often be for the Command and Control server to send new instructions to the software running inside the defender's network instructing it on how to proceed with the attack.

For this work we assume the ability to gain a foothold on some machine inside the defender's network, and thus we do not address the intial infection vector (such as phishing).
I

1.6 The "Malware Vectors" Technique

We propose the use of "Malware Vectors", a technique to discover and evade defense logic systems. In this technique, an attacker evaluates its set of priorities and based on those priorities, enumerates "capabilities" that would allow it to realize its objectives. Each capability is comprised of a set of actions taken on a defender network. Each action generates observables, the observables being assigned a value based on the range of possible values.

Each of these capabilities provides some utility to the attacker based on the aforementioned sets of priorities. Through the use of probes that generate known observables and observation of when probes are detected, the attacker is able to simultaneously learn information about the defense logic and thus discover a way to design a covert attack.

This probing technique remains covert, as the probes are designed not to reveal the attacker's actual objectives. Any alarms that are triggered appear to the defense to be working as intended, and should not spur the defense to change its logic. Undetected probes are by definition covert, as the defense logic does not detect them.

1.6.1 Real World Analogs

A kinetic analog to this technique is the ongoing defense testing performed by the United States and Russia, beginning during the Cold War and continuing with regular frequency to this day. Each country will, on occasion, probe the air defenses of the other by flying military planes near the other's airspace to observe the response they provoke. NORAD has stated that in 2009 alone it detected and intercepted Russian bombers sixteen times [17].

Similar techniques have also been shown in popular culture. In the 2001 Film "Bandits" [18], two men aim to rob a bank. In the planning stages of the robbery, they test the bank's defenses by intentionally triggering the alarm from outside. They then measure the response time of the police, and know exactly how long they will have to rob the bank and get away.

Antivirus definitions are very similar to modern cyber defense systems. Antivirus programs work by checking data on the user's harddrive and applications in running memory against a list of signatures of known bad activities.

1.7 Results

We find both analytic and empirical results in our exploration of this technique. First, in our analytic analysis we find that for any Defense representable as a Monotone K-DNF[10], with M terms on N variables, an Attack representable as an A variable conjunction takes at most

$$\left(1+\frac{1}{\binom{A}{K}}\right)\times M+\frac{1}{\binom{A}{K}}\times\left(\binom{N}{K}-M\right)$$

probes to discover the full defense logic. Additionally the expected number of probes to find a single vulnerability (missed detected conjunction) of size A, given a K-DNF defense logic with M terms is:

$$\frac{M}{\binom{A}{K}}$$

Secondly, we find empirically in Chapter 5 that a carefully crafted Defense Logic against a specific attack type is incapable of preventing an Attacker who uses 8 probes to evade the defenses. Additionally, we find more broadly for a large number of generated defenses that it is relatively trivial for an attacker to evade them. Even with up to 2000 Terms in a 3-DNF (which only has 4060 possible unique terms) the defender is unable to, on average, ensure that the attacker must use 40 probes to find a vulnerability.

[10]A logical expression composes entirely of disjuncted conjunctions of no more than K variables which are not negated.

Chapter 2

Background

In this Chapter, we review past research efforts in modeling Cyber scenarios and introduce the background material required to understand the contributions of this thesis. We begin with an Overview of Logical Expressions.

Logical Expressions are used extensively in the main contribution of Chapter 4 and 5. We continue with a quick primer on Game Theory and Attack Graphs, important for Chapter 3. We then discuss in further depth the background information on real world cyber problems and the models currently used to express them. Finally, we will introduce the more sophisticated techniques we build will upon in Chapters 4 and 5.

2.1 Overview of Logical Expressions

Logical Expressions consist of one of more Logical Operators and Logical Variables. Logical Variables, and subsequently Logical Expressions, can only evaluate to values "True" or "False", represented by 1 or 0 respectively. Each Operator operates at most on the two subexpressions adjacent to it. Logical expressions follow the order of operations rule with respect to parentheses. Additionally, the NOT Operator has precedence over other Operators; Operators otherwise are evaluated left to right. In Table 2.1, we introduce the Logical Operators we will use in this paper.

Logical Operator	Symbol	Meaning
AND	$\wedge$	Evaluates to True $\iff$ both subexpressions are True.
OR	$\vee$	Evaluates to True if either subexpression is True.
XOR	$\oplus$	Evaluates to True $\iff$ exactly one subexpression is True.
NOT	$\neg$	Negates the value of the subexpression to the right of the operator.

Table 2.1: *The Logical Operators we will use in this paper.*

There are a number of formal constructions of Logical Expressions that can allow for a measure of simplification in analysis of those expressions. Commonly used is the Disjunctive Normal Form (DNF). A DNF expression is a series of disjunctions (subexpressions OR'd together) with each subexpression consisting only of conjunctions (variables AND'd together). For example, this expression of the variables A, B, C and D is written in DNF:

$(A \wedge B) \vee (C \wedge D)$.

Formally, each conjunction of a DNF is a "term". For example, the expression above has two terms, $A \wedge B$ and $C \wedge D$. We also use a shorthand notation for terms, which implies conjunctions. For example, the above statement can also be written as:

$AB \vee CD$

In Chapter 5 we use an additional shorthand for logical expressions where each variable can take on additional values. For example, a_4 would represent the variable a taking on the value 4. This notation with lower case variables and a subscript is used to specify observables generated by an attacker.

For this notation it is also important to know the value space in which the variables are operating. We do this by considering the values to exist in a base, S. Thus, variables in base S are restricted to the natural numbers between 0 and $S - 1$.

To allow for defender logic that sets a ceiling on "acceptable" values, and detects events occuring with frequncy equaling or exceeding that ceiling, we use a shorthand consisting of uppercase variables with a subscript. This shorthand is equivalent to an expression that covers the range of values for the variables of a subexpression. Table 2.2 provides a few examples of how to expand these shorthand expressions.

Shorthand	Base	Expansion
A_V	S	$a_v \vee a_{v+1} ... \vee a_{S-1}$
A_2	4	$a_2 \vee a_3$
$A_3 \wedge B_2$	5	$(a_3 \wedge b_2) \vee (a_3 \wedge b_3) \vee (a_3 \wedge b_4) ... (a_4 \wedge b_4)$

Table 2.2: *The expansion of shorthand defense expressions.*

2.1.1 Implicants

An important concept for the work in Chapter 4 is that of the Implicant. A Logical Conjunction Term B is an Implicant of another Expression F if $B \implies F$. Put another way, B is an implicant of F if B is a "covering" term or a superset of F. For example, AB is an implicant for A, B and AB, because if AB is true, all of those terms must also be true [19]. Generally we are not interested in the set of implicants containing the term itself, so the set would be stated: $AB \implies \{A, B\}$. We introduce the term "capturing" for the reverse property. In this case either A or B captures AB because $AB \implies \{A, B\}$.

For a more complex example we find the implicants for $(A \wedge B) \vee (C \wedge D \wedge E)$. First we split the expression into its conjunction terms, AB and CDE. Next we would find the implicants for each term using Truth Tables. Since the implicants for AB are known, we show those for CDE in 2.3.

Value of C	Value of D	Value of E	Value of Term $C \wedge D \wedge E$
1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	0
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

Table 2.3: *The Truth Table for the first term $A \wedge B$.*

From the Truth Table 2.3 we find that CDE is an implicant for $\{C, D, E\}$. Additionally, every combination of those 1 length implicants is also an implicant. Since the only implicant of length 3 will be the term itself we only need to consider implicants of length 2. By counting there will be $\binom{3}{2} = 3$ additional implicants of length 2. These are: $\{CD, CE, DE\}$. Thus, the full set of implicants is: $CDE \implies \{C, D, E, CD, CE, DE\}$.

Formally, only conjunction terms can have implicants, however, we can draw some inferences about the nature of the set of implicants of the DNF expression based on its value. Since the expression is in DNF form, if any of its terms is true, the overall expression will also be true. Thus, if the full expression is true it must be the case that at least all of the members of a set of implicants for one of its terms must be true.

2.1.2 Monotone Expressions

For much of the work in Chapters 4 and 5 we focus on Monotone DNF Expressions. A Monotone expression is formally defined thusly: "[An expression] is monotonic if, for every combination of inputs, switching one of the inputs from false to true can only cause the output to switch from false to true and not from true to false" [20]. Put more simply, a monotone expression is an expression that has no negations. For example, AB is monotone, while $(\neg A) \wedge B$ is not because for the assignment of values $A = False, B = True$ if we switch A to True, the expression becomes False.

2.2 Overview of Game Theory

Game Theory is "the study of conflict and cooperation between intelligent rational decision-makers" [21]. Game Theory provides a technique for modeling adversarial scenarios. In this section we discuss the nature of information in games, and some basic background on how players make decisions in the formalized structure of Game Theory.

2.2.1 Nature of Information in Games

Game theoretic models provide various constraints on the information available to the players. Of particular interest is the contrast between imperfect and perfect information games as well as complete and incomplete information games.

In a perfect information game players are aware of all information constructing the current state of the game. Conversely in an imperfect information game, players may be unaware of some information about the state of the game.

Similarly, but distinctly, in a game of complete information each player is aware of the full structure of the game and the payoff functions of all players. In Chapter 3.1 we discuss an incomplete information game (each player is not aware of the utility functions of the other players), with hidden information (each player has a piece of secret information, making this an imperfect information game). The analysis of this multi-player game explores a technique for estimating future adversary actions based on an assumption of a continuous use of a single utility function by the adversary, and on observations from play.

2.2.2 Utility Functions

Players in games are rational and thus must have an empirical process for decision making. This process makes use of a function called a “Utility Function” which considers the costs and outcomes of an action. Players in games will aim to always maximize their expected utility. The utility of an action taken by a player is a function of the set of actions taken by all players.

Utility Functions are derived from the actual outcomes received and the costs incurred by the players, pursuant to adjustment via a “Risk Profile”, or a player’s attitude or tolerance for risk. A “Risk Profile” is a function representing the appetite of a player to risk. A “Risk Profile” transforms actual losses or gains to the perceived value of the loss or gain experienced by the player.

The simplest “Risk Profile” is a linear one, in which all gains and losses are multiplied

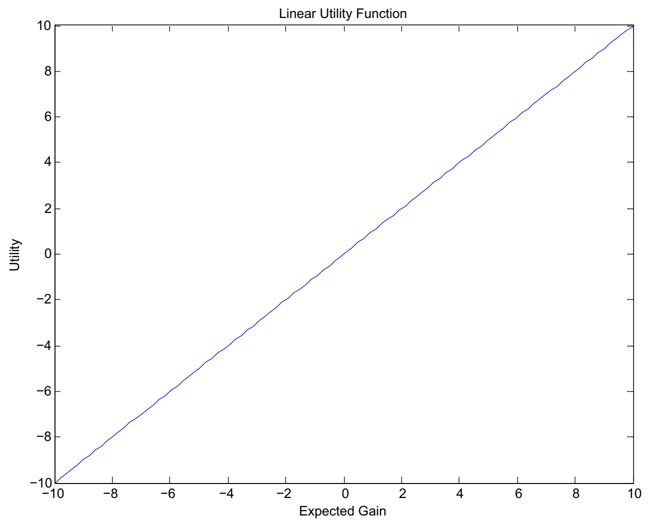


Figure 2.1: *The Linear Utility Function*

by the same constant C, as seen in Figure 2.1.

2.3 Overview of Attack Graphs

An attack graph is a finite graph in which directed edges represent steps in an attack on a defender's systems. The attacker wishes to traverse the graph from the start to their objective. The graph is weighted, and the weight on each edge represents the cost to an attacker to transition between two states. In Section 3.2, we discuss the analytical results regarding attack graphs which preceded this work.

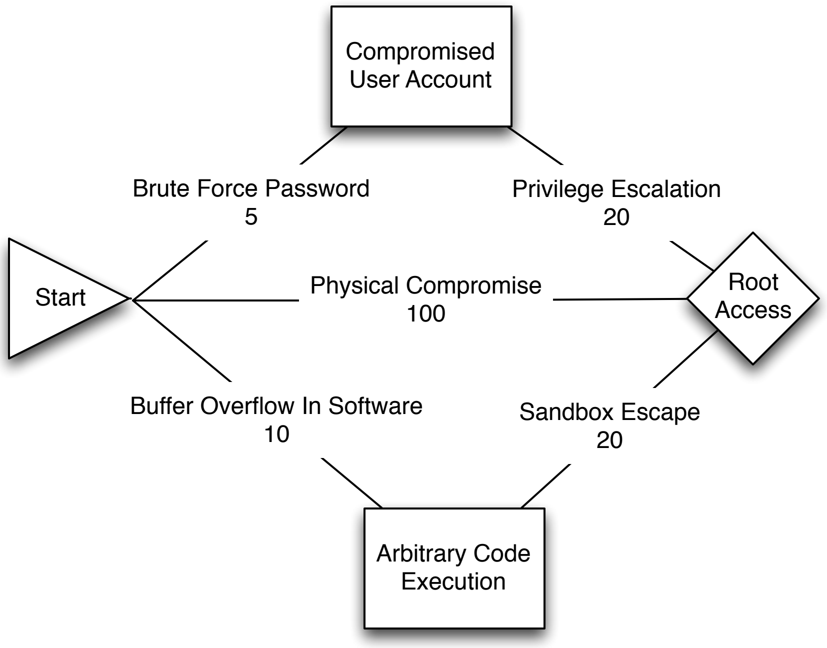


Figure 2.2: *An example of an attack graph.*

Figure 2.2 is an example of an attack graph where an attacker wishes to gain root[1] access to a system on the defender's network. Each edge represents an action the attacker can take, where its weight is the cost of traversing that edge. Each node represents a state the attacker can be in. In this graph, the attacker wishes to traverse the graph from "Start" to "Root Access". The attacker can take a number of paths, by either exploiting vulnerable software running on the system, compromising a user account, or performing a physical compromise of the machine. An attacker interested in minimizing their cost to compromise would take the path through the "Compromised User Account". A defender interested in increasing the difficulty of compromise could implement a password attempt limiter to drastically increase the cost of such an attack.

[1]"Root" access, also called "administrator" or "superuser" access refers to unrestricted access on a system, including the ability to change system files, executibles and configurations.

2.4 Current State of Cyber - Real World

In real world environments, an actor, who we will call the Defender, wants to complete a mission using the network and computing resources they own. In order to complete this mission they will generally require that computer systems must be networked or connected in some way.

Adversaries, who we will call Attackers, wish to use the resources of the Defender to execute their own mission, without the knowledge of the Defender. Some possible mission objectives include the exfiltration of proprietary information residing on the Defender's systems, or the use of the Defender's system as a intermediary system to attack a third party's systems.

Defenders have a variety of techniques they use to protect their systems against the attempts of Attackers to compromise them. These techniques span everything from auditing the software running on each system on the network to scanning network traffic for suspicious behaviors using an Intrusion Detection System or IDS. In particular this work focuses primarily on the functions of IDS and other related systems.

2.4.1 IDS

An Intrustion Detection System is a system that collates logs from a number of sources and compares events in those logs to numerical or statistical rules that trigger alerts when their parameters are violated. Snort is an example of a Network IDS (NIDS), while ArcSight is a Security Information and Event Management system, a more generic IDS.

IDS systems work by comparing their set of alert rules to properties of events on the network or systems under observation. The funtionality of these systems is thus defined by their alert rules. The vulnerabilities that exist in these systems are a function of shortcomings of these rules. These rules are almost entirely based on previous, known, vulnerabilities. In order to increase the efficacy of these retrospective rules, they are often written with

```
rate_filter
    gen_id 0, sig_id 1,
    track by_src,
    count 100, seconds 1,
    new_action drop, timeout 10
```

Figure 2.3: *This Snort filter will begin to drop all packets from an IP address when it has detected 100 ping packets from that address in 1 second [24].*

the objective to caputure the vulnerability, not the exploit[22]. For NIDS there are techniques similar to Malware Vectors, such as in Vigna[23]. Vigna proposes using random mutations exploits to widen a narrowly defined rule to attempt to capture the vulnerability rather than the exploit. However, unlike Malware Vectors, Vigna's "Mutant Exploits" are generated randomly and do not attempt to address optimal actions an attacker may take. Snort is open source, and comes with a set of default commonsense rules, which the community has deemed to be appropriate to detect known attacks.

Snort Examples

In this section we provide a few sample rules from Snort documentation, and show how they can be represented by a Monotone DNF expression. In addition, we show the interface to these two programs (Snort and ArcSight) in order to give a sense to which tools operators crafting rules would have access to. We discuss some of the practical shortcomings of these systems by using examples of times they have failed to provide the protection defenders assumed they would provide.

In Figure 1.3, we showed a very simple Snort rule for detecting ICMP traffic. If we were concerned about ping flood attacks, we could add a Snort "rate-filter" to detect any case where in excess of 100 pings per second are detected, as shown in Figure 2.3.

This rate filter would drop all traffic from any IP that sent more than 100 pings in a second. This could be represented by the trivial Monotone DNF, where A_i represents a ping rate per second of $50 * i$ and $S \geq 3$: A_2.

In a slightly more complex example, we assume the defender has a machine running a

```
alert tcp any any <> 10.0.0.1 80 (pcre:"/4\d{3}(\s|-)?\d{4}(\zs|-)?\d
    {4}(\s|-)?\d{4}/"; \
  msg:"VISA card number detected in clear text";content:"visa";nocase;
     sid:9000000;rev:1;)
    }
```

Figure 2.4: *This Snort rule will issue an alert message whenever it detects a network packet headed to the server at 10.0.0.1 on port 80 (unencrypted HTTP) which contains a Visa card number. The string following "pcre" is a Perl regular expression which captures a at least all Visa card numbers [25].*

website on the IP "10.0.0.1". This website receives credit card numbers over HTTPS (port 443). In order for the defender to maintain their relationship with their credit card processor they may be required to ensure that these credit card numbers are not transmitted over unencrypted HTTP (port 80). The defender could use the rule in Figure 2.4, which detects Visa card numbers sent in plaintext, to insure that their system is configured correctly.

ArcSight Examples

ArcSight is a proprietary software suite sold by HP. There are some public resources which discuss creating ArcSight rules, but significantly less than that for Snort rules. ArcSight Logger is a database baked application and provides two interfaces for checking queries against the event databases. The first interface executes queries similar to a Structured Query Language[2] database queries and allows interactive viewing of results. These rules are called "Searches". A system operator can save Searches and run them again later, but generally Searches are ephemeral. For example, Figure 2.5 contains a search which finds DHCPACK[3] events.

The second interface generates "Reports". A Report is generated via the results of an SQL query directly to the ArcSight database. This allows for more powerful queries

[2]SQL databases are perhaps the mainstay of the majority of most database systems on the web. Consistent with their name, these databases have a structured language for queries, very similar to a programming language.

[3]The DHCPACK packet is the final packet in the DHCP protocol for Dynamic IP assignment. The existence of this packet confirms that the server and client have agreed on an IP for the client - and contains the duration that the client will have that IP for.

```
applicationProtocol="DHCP" AND deviceAction="DHCPACK" AND
   destinationAddress = "130.64.205.133"
```

Figure 2.5: *This ArcSight Search will find all DHCP ACK packets sent to "130.64.205.133" [26].*

than the search. For example, the report in Figure 2.6 would return the frequency of login attempts separated by failed and successful and grouped by the IP which sent them.

The results of Reports can be configured to show up as Alerts. Alerts can either perform actions automatically (similar to the automatic IP blocking behavior for Snort we explored in Figure 2.3) or prompt a human operator to review the alert and possibly take action. Figure 2.8 is one of the ArcSight interfaces on which alerts can be viewed, as shown in ArcSight's marketing materials.

2.5 Cyber Security - Academic Literature

Cyber Security scenarios are often modeled as games. In particular these game theoretic models may assume a priori payoffs and perfect information. We discuss the shortcomings of many of these models in Section 3.1.

Recent work tends to focus on the potential vulnerabilities in "smart grid" networking. The smart grid connects small computers called "meters" which measure electricity consumption for households and businesses and relay that information either to one another or directly to the electrical utility. Compromises of these systems could theoretically spread rapidly, as the systems are largely homogenous. Additionally, compromises of these systems can have real world consequences. The United States Department of Homeland Security studied this problem and found that, at least under some conditions, it is possible to remotely cause a power generator to self-destruct [29].

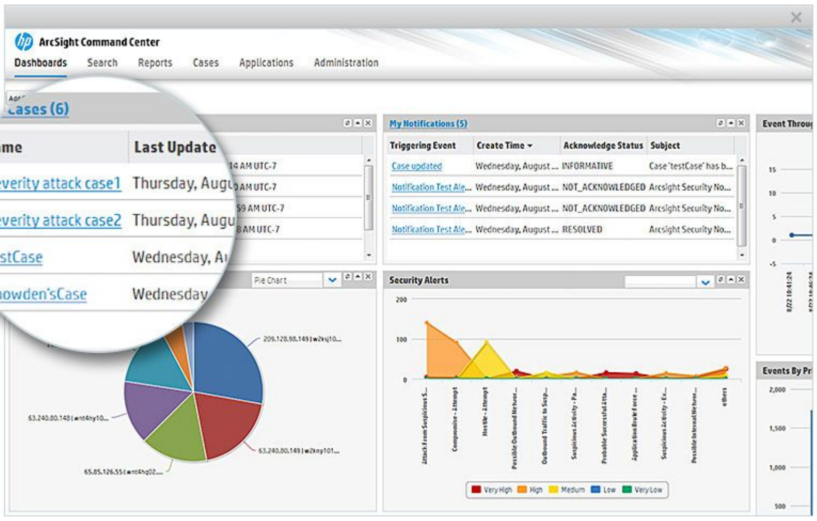
Ericsson [30] discusses techniques for designing secure system networks for the control

```
SELECT
   events.arc_sourceAddress,
   events.arc_sourceHostName,
   sum(IF (events.arc_categoryOutcome = '/Failure', 1, 0))
      as totalFailures,
   sum(IF (events.arc_categoryOutcome = '/Success', 1, 0))
      as totalSuccess,
   sum(IF (events.arc_categoryBehavior = '/Authentication/
      Verify', 1, 0)) as totalAttempts,
   count(DISTINCT events.arc_destinationAddress) as
      totalTargets
FROM
   events
WHERE
   events.arc_categoryBehavior = '/Authentication/Verify'
   <%campusWhereSnippet%>
GROUP BY
    events.arc_sourceAddress
ORDER BY
    <%authResult%> DESC
```

Figure 2.6: *This ArcSight Report [27], reports the total number of successful and unsuccessful login attempts on a per source basis, as well as the number of machines which were logged into. It is further restricted to IPs specified by the "campusWhereSnippet", specfied separately in the ArcSight Configuration. An example of such a snippet is included in Figure 2.7.*

```
    AND (
   events.arc_sourceAddress LIKE \'130.64.\%\'
   OR events.arc_sourceAddress LIKE '10.%'
   OR events.arc_sourceAddress LIKE '192.168.%'
   OR events.arc_sourceAddress LIKE '172.16.%'
   OR events.arc_sourceAddress LIKE '172.17.%'
   OR events.arc_sourceAddress LIKE '172.18.%'
)
```

Figure 2.7: *This snippet provides an example of how you can specify multiple addresses to restrict an ArcSight behavior to. The % symbols are a wildcard that will match any thing as long as the first part matches [27].*



Name	Last Update
severity attack case1	Thursday, Augu
severity attack case2	Thursday, Augu
testCase	Wednesday, A
snowden'sCase	Wednesday

Triggering Event	Create Time	Acknowledge Status	Subject
Case updated	Wednesday, August ...	INFORMATIVE	Case 'testCase' has b...
Notification Test Ale...	Wednesday, August ...	NOT_ACKNOWLEDGED	Arcsight Security No...
Notification Test Ale...	Wednesday, August ...	NOT_ACKNOWLEDGED	Arcsight Security No...
Notification Test Ale...	Wednesday, August ...	RESOLVED	Arcsight Security No...

Figure 2.8: *This is an image provided by HP, the manufacturer's of ArcSight, showing the interface an operator would look at to actively manage alerts [28].*

of the SCADA[4] systems which control these "smart grid" components. The authors propose that machines be logically separated by the tasks that they perform and the trustworthiness of each component. The trustworthiness of a component is based on the portions of the network that component is allowed to operate in. Figure 2.9 is the "Information Security Domain Model" used by Ericsson to model risk of various components prior to deciding how to monitor them.

2.6 Current State of Boolean Logic Discovery

2.6.1 Queries and Concept Learning

A common technique for discovering an unknown boolean function from iterated test vectors is called "Querying". The process of querying involves using a black box which knows the Expresison under test, called an "Oracle" which gives you information about the relationship between the Query and the Expression based on the type of query performed. There are multiple kinds of queries, of particular interest to this field of study are two

[4]SCADA stands for supervisory control and data acquisition. SCADA systems tend to consist of a centralized monitoring and control system which communicates with sensors and devices.

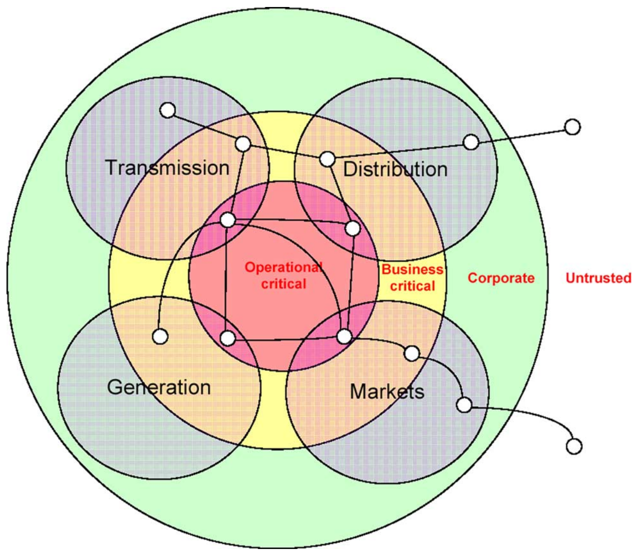


Figure 2.9: *An example of an Information Security Domain Model for Smart grids as envisioned by Ericsson [30]. The nodes represent different classes of systems. The systems are segregated into different networks, signified by the concentric circles. There is also a logical ordering on the nodes labeling what tasks they perform. For example the node second from the bottom-left is a Operation Critical Generation node. One such system in this class would be the control system for a nuclear power plant.*

kinds, namely Membership and Equivalence queries.

Definition Let U be the set of all logical expressions of a known set of variables V. Let L be the particular logical expression we wish to learn.

Membership: The input is an element $x \in U$ and the output is yes if $x \in L$, and no if $x \notin L$ [31].

Equivalence: The input is a set l and the output is yes if $L = l$, and no if $L \ = l$. If the answer is no, an element $x \in L \oplus l$[5], is returned [31].

For our investigation, we are limited to only membership queries, since the attacker will have no mechanism to test if their notion of the defender's logic is equivalent to the actual logic of the defender.

From Angluin [31] we find that the formulas we are using are a subset of DNF called Monotone DNF formulas. Angluin proves theorem 2.6.1.

[5]Recall $\oplus$ is the logical operator XOR or Exclusive Or, defined in Table 2.1

Theorem 2.6.1. *There is an algorithm that exactly identifies every monotone DNF formula ϕ_* over n variables that uses equivalence and membership queries and runs in time polynomial in n and the number of terms of ϕ_*.*

Unfortunately, in this scenario we don't have the luxury of running equivalence queries and are restricted to only membership queries. The literature is more limited on techniques which only deal with membership queries. Jackson [32] and Bergadano [33] each discuss a membership-query only algorithm for Probably Approximately Correctly, or PAC, learning DNFs. These papers do not address the topic of monotone DNF and thus the bound they find (Jackson finds $O(\frac{NM^8}{\epsilon^{12+c}})$ to a precision $1 - \epsilon$) is significantly higher than even the naive factorial algorithm ($O(\binom{N}{K})$) for exactly learning monotone K-DNF which are of sizes within the scope of these models. This is discussed in more detail in 4.1.2

Jackson, Lee et al. [34] discuss a PAC algorithm for learning monotone DNF with high probability. This algorithm is excellent in the limited cases it addresses, but it requires that for a K-DNF that there are approximately e^K terms. It seems that the authors restrict the number of terms in order to restrict the incidence of a single term being an implicant for multiple queries. This is a similar concern that we discuss in Chapter 4. Here we are unable to make the same assumptions.

2.7 Binary Integer Programming

We use Binary Integer Programming to validate that our search of the assignments to variables produces the optimal strategy in Chapter 5.

Linear programming in general is "the problem of maximizing or minimizing a linear function subject to linear constraints" [35]. Binary Integer Programming is a special case of Linear Programming in which variables can only be assigned the values 0 or 1. Whereas Linear Programming finds results in the real numbers we are interested in finding only assignments in $\{0, 1\}$.

Formally, Binary Integer Programming solves the problem:

$$Maximize \sum_{j=1}^{n} c_j x_j,$$

subject to the constraints:

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, i \in \{1...m\}$$

$$x_j \geq 0, j \in \{1...n\}$$

$$x_j, x_j \in \{0,1\} \forall j$$

This technique accepts weights c on n variables x and given constraints on which of those variables can be true finds the optimal set of variables to set to true.

For example, there is a common joke about college: You can have good grades, sleep, or friends, but you must choose two. If we assign weights to each of these priorities, we can use this technique to decide which two to choose.

Assume the variables are represented by x_1, x_2, x_3 respectively, with the weights represented by the same indices. We can set our priorities:

$$c = [10, 6, 7]$$

We then need to incorporate the constraint “choose two”. To do so we set $b_1 = 2$ and $a_{i,j} = 1, \forall j$. We can then solve using Matlab’s “bintprog” function[6].

```
bintprog([-10,-6,-7],[1,1,1],[2])
```

And we find:

```
ans = 1 0 1
```

By referring back to the order in which we assigned variables, we should disregard

[6]Note, this MATLAB function actually minimizes the function, so we must negate the values of the objective function in order to find the maximum.

sleep and acquire grades and friends.

Chapter 3

Results

3.1 Previous Work: Exploiting Adversary's Risk Profiles[36]

In 2012 we presented a paper titled "Exploiting Adversary's Risk Profiles in Imperfect Information Security Games" at the GameSec conference. This paper discussed a technique for observing the behavior of adversaries and adapting our behavior to improve outcomes. Specifially, we investigated a side-channel attack on information leaked by adversary actions to improve our estimate of the secret information they hold. We discovered that this technique allows a player to dramatically improve their play, increasing their probability to win seven-fold. This work was where we began investigating side-channel information attacks in adversarial decision problem scenarios.

3.1.1 Abstract

At present much of the research which proposes to provide solutions to Imperfect Information Non-Cooperative games provides superficial analysis that requires a priori knowledge of the game to be played. We propose that High Card, a simple Multiplayer Imperfect Information Adversarial game, provides a more robust model for such games, and further, that these games may model situations of real world security and international interest. We

have formulated two such real world models, and have created a modeling bot which, when facing adversaries with equal or better performing risk profiles, achieves a 7-fold increase in win performance.

3.1.2 Introduction

The field of research for imperfect information non-cooperative games can be said to comprise a number of areas of such research. To begin, we discuss two such areas - Poker research, and normal form simultaneous move games, in order to lay the groundwork for where our research fits into the field. Poker research, such as in Billings [37][38], Papp [39], focuses on creating AI or bot programs which play a game of poker - generally Texas Hold'em - based on some expert knowledge of the game of poker. These bots will sometimes perform in-depth opponent modeling, but generally their opponent models are rather simple in nature. However, even in the strongest case, this research focuses solely on the game of poker, and does not suggest that the results can be used as a model for other games or real world scenarios, such as security or international interest applications.

The second pertinent area of research is research that involves studies of specific normal form synchronous move games, such as Lye et al. [40] and Jiang et al. [41] [42], which propose to solve a broader problem for which these games serve as a model. Unfortunately, these normal form games results are only theoretical as they require that we know a priori the payoffs for the game. The games are 'solved' to find the Nash Equilibrium. However, often real world players don't play according to "optimal" strategies as they are too difficult to compute, even, in many cases, in a theoretical sense. In addition, these results do not provide any additional knowledge about the nature of non-cooperative game theory as a field.

We propose that High Card, a multi-player adaptation of von Neumann's betting game (similar to a single betting round in Poker) can serve as a model for security applications. High Card does not require that we assume a priori knowledge of payoffs, and further, it allows for multiplayer play. We will show that we have created a bot that creates opponent models based on past opponent play in order to estimate the secret information that adversaries have, and exploit those opponents to win additional resources.

3.1.3 High Card

Poker is a rather complex game to model. Much of the complexity of the game adds significant requirement for expert knowledge, but does not provide a benefit for use in modeling real world scenarios. As such, it is not uncommon to study a simplified version of the poker game, which preserves the basic elements of the game. Borel and von Neumann each simplify poker to a two-player zero-sum game, where instead of a hand of cards, each player is dealt a hand $X \in [1, S]$ [43]. High Card is essentially a multi-player adaptation of such simplified poker models. In particular, the simplifications we have made to the poker game are a subset of the simplifications made for the von Neumann poker model, namely that this is a multiplayer, unlimited bet sequence game rather than a two-player limited bet sequence game [44]. These simplifications drastically reduce the complexity. For example, in a game of Texas Hold'em with n opponents there are $H = \prod_{k=1}^{n} \binom{52-2k}{2} \div k!$ possible starting hands. For 6 opponents, this works out to over 1 quadrillion combinations of starting hands. In High Card, by contrast, for the same 6 opponent game there are only 13 billion starting hand combinations. In addition, Texas Hold'em has additional added complexities by the nature of its multiple betting rounds, whereas High Card only has one.

Game Parameters

Upon starting the game, a player is selected at random to be the "Dealer" and each player is issued a set, equal amount of chips. The player to the dealer's right is the "Big Blind", which also refers to the size of the ante the player in the Big Blind seat makes. The player to the right of the Big Blind is the "Small Blind", and antes an amount of chips less than or equal to the amount that the Big Blind wagers, traditionally half of the Big Blind wager.

Any chips wagered are immediately added to the “Pot", which the winning player receives in its entirety.

A Round of Play

At the beginning of each round the Blinds post their antes simultaneously. Betting then begins with the Dealer and proceeds to the left. Play continues until only M players remain. M is determined in advance.

Player take actions in turn, left around the table. When it is their turn a player may:

- Fold: abandon any claim to the Pot.
- Call: wager an amount of chips such that their total wager is equal to the largest wager made thusfar.
- Raise: wager an amount of chips larger than the largest wager so far by at least the Big Blind.
- Check: call a current additional wager of zero chips.

As a simple example, we have a game with a Big Blind of size 20, and a Small Blind of size 10, and each player starting with 50 chips. Players A, B and C sit around a table, B to A’s left, and C to B’s left. Player A puts in the Small Blind, Player B puts in the Big Blind. Player C chooses to Call the current wager of size 20, as set by B’s Big Blind. Player A then chooses to Raise the wager to 50, requiring that A put in 40 additional chips. Player B decides to Call Player A’s wager of 50, putting in 30 additional chips. Player C folds. There are now 120 chips in the Pot, and whichever of Player A or B has more chips will receive all 120 chips. The other player will be left with 0, and C will remain with 30.

Sidepots

If a player wagers an amount of chips larger than the amount of chips possessed by any other player who has not folded in the current round, a sidepot is created. If there are a number of players with chips smaller than the amount of chips wagered by a player, then more than one sidepot may need to be created. Each player is automatically a party to any sidepots which they are able to participate in. Once a sidepot is created, any raises will go into that sidepot, or if a new sidepot is required, into the new sidepot.

For example, Player A has 70 chips, Player B 50 and Player C 20. Player A wagers 70 chips. This puts 20 chips - because C only has 20 chips - into the Main Pot which all players are a party. 30 chips go into Side Pot 1 - because B only has 50 chips, 20 of which will go into the Main Pot should B call - to which Players A and B are Parties. The remaining 20 chips into Side Pot 2 which only Player A is a party to, because no other player has enough chips to participate in Side Pot 2. While it would not be realistic that A would wager chips into a Pot that only A could win, it does not effect the results in any way, as A is guaranteed to receive those chips back. Players must meet the bet in any sidepot to which they are a party in order to not fold, and Players may only win chips from Pots to which they are a party. Assume that then from the previous example, both B and C call Player A's wager. If Player A has a 30, Player B a 40 and Player C a 50, Player C will win 60 (20 + 20 + 20) chips from the Main Pot, Player B will win 60 (30 + 30) chips from Side Pot 1 and Player A will win (by default) his own 20 chips from Side Pot 2.

Round Resolution

When each player who has not folded, or caused the last raise or ante, has called or checked in succession since the last raise or ante, the betting is over. At that point the player with the best card wins the entirety of any pots to which they are a party. At the end of each round the Dealer, Big Blind and Small Blind positions pass one player to the left. Note, unlike in Poker where there are multiple rounds of betting in each hand, in High Card each hand only has one round of betting.

Rules in a Nutshell

At the start of each hand two players are forced to make an ante, and each player is dealt a card. Betting proceeds until all players have matched the same bet or folded out of the hand. The player with the highest card wins all the chips wagered. Which players are forced to make the ante changes, and play continues with a new hand until a predetermined number of players remain.

3.1.4 Examples

In this section in order to demonstrate the wider potential use of the model simulation, we propose a number of possible situations that could feasibly be modeled by High Card.

Diplomacy

We find that there is existing literature in which Poker is used as a metaphor for diplomatic relations. In particular, in Smith et al., no-limit poker in particular is used as a metaphor for North Korean - United States relations [45] and in Freeman it is used as a metaphor for Cold War nuclear disarmament discussions [46].

Such comparisons are apt because diplomatic relations constantly involve quarrels between countries often with outside actors intervening occasionally escalating to very high stakes. Focusing on these quarrels in particular, we propose that these relations can be modeled using a game of High Card. In particular, during such quarrels, it may be the case that a number of countries which possess damaging information about the other countries will be willing to risk some of their own credibility or resources in order to attempt to extract resources or credibility from the other countries. The end goal of course is that by doing so they will increase their own power in future negotiations.

We propose there exists a model for diplomatic relations in which, at each time period t two diplomatic adversaries have a small disagreement which forces them to risk some credibility. We assume that other actors may become involved in these diplomatic relations, each in turn deciding if they want to wager some credibility. During each actor's turn they may take one of three actions - matching the amount of credibility wagered by an opponent, withdrawing from the confrontation thus leaving behind any credibility that they have wagered, or raising the stakes and so by wagering additional credibility.

At the end of wagering the agent with the best secret information is able to gain diplomatic leverage over the other agents and by doing so use that information to take the credibility that was wagered on the quarrel.

A particular example of an ongoing high stakes diplomatic quarrel is the interaction between the United States and Pakistan. Though for many years the United States has been using drones to bomb non-military targets in Pakistan, it generally did not send actual US personnel into Pakistan. That changed with the assassination of Osama bin Laden. This action has caused a diplomatic quarrel between the two countries, with other countries intervening, and each country holding secret information about the other. It is suspected that the United States holds damaging information about Pakistan related to Osama bin Ladin's undetected life in the suburbs [47], and Pakistan arrested five people who they claim are CIA informants [48].

Even better, the 1960 U-2 incident between the United States and the Soviet Union can be seen as a two-player betting game in action. In this example, the United States triggers the diplomatic incident, sending a spy plane into Soviet airspace, the Soviets then shoot the plane down. The United States, bets that it can win the confrontation, and puts additional credibility on the line, stating that the plane is a NASA weather plane which mistakenly drifted into Soviet airspace. Unbeknownst to the United States of course, the Soviets had captured the plane mostly intact and the pilot alive. At this point they reveal their secret information, and claim the credibility that the U.S. had staked upon the incident, embarrassing President Eisenhower at multiparty talks with Great Britain and France [49].

Computer Security

In a model of Computer Security as a game of High Card each play is akin to a country competing against other countries to protect their own resources, and attempting to obtain the opponent resources. Each hand is akin to the 'big blind' being the defender, and the 'small blind' launching an initial cyber attack against their systems. We assume then that there is public information about these attacks, or that the attacks are frequent enough so as to be constant, and thus the players always able to participate. This is a reasonable assumption based on multiple statements from government officials, as well as actual security incidents that cyber attacks are a real, constant, ongoing threat [50][51].

The game proceeds as a normal game of High Card with each player who is not the Big Blind being presented the opportunity to participate in the attack. Whoever has the most sophisticated attack (best secret information) is then able to obtain resources. If they are defending, the resources they gain can be considered to be techniques that the attackers can no longer use. For the attackers, the resources are information about attack techniques the other attackers have used, as well as whatever resources (data, operational security) the defender was attempting to protect.

3.1.5 Simulation

We created a simulation of a High Card game using bot players whose actions were determined by various utility functions. At the beginning of each round the players are allocated an equal number of chips, here 1000, and seated at random positions around a table. The Big Blind was set to 20, and the Small Blind 10. We ran simulations with M set to either 1 or 2, for 6 player games. Each round uses a shuffled complete 52 card deck drawn without replacement.

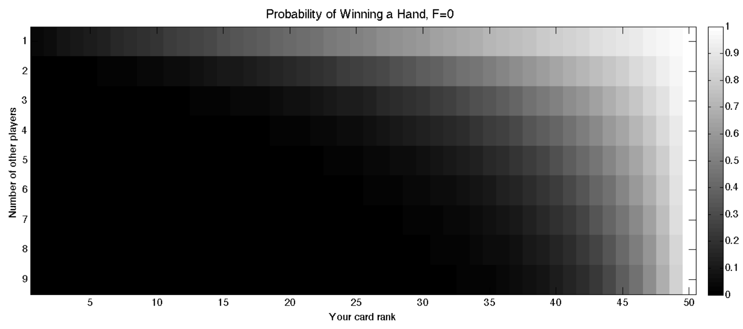


Figure 3.1: A heatmap for probability of winning, given your card rank and the number of players in the game. As shown, as N increases at a fixed C, the probability of winning decreases, but your potential profit could still be increasing.

Probability of Winning

In High Card the probability of a player winning a game of equals the probability that every other player at the table was dealt a card of a lower value than the card that the player was dealt.

$$P_{win} = \frac{\binom{S-C}{0}\binom{C-1}{N}}{\binom{S}{N}} = \frac{\binom{C-1}{N}}{\binom{S}{N}}$$

Given the total number of cards remaining, S, the rank of your card, C, and the number of other players in the game, N, we propose that it is a reasonable assumption that the players who have folded at the time a player is taking an action had cards of a rank lower than the rank of that player's card. Call the number of players who have folded F, thus:

$$P_{win} = \frac{\binom{C-1-F}{N-F}}{\binom{S-F}{N-F}}$$

A plot of the probability of winning given the number of other players in the game, and the card you have assuming F=0 can be seen in Fig. 3.1.

Utility Functions

We define $U_i(P, B, W)$ as the expected utility of player i betting on pot P with size of the bet to be made B, and the estimated probability to win W.

These bots use a number of different utility functions including:

- linear: $U(P,B,W) = W*P-(1-W)*B$ This utility function is the most obvious as there is no change value of a chip you expect to lose or a chip you expect to gain. In short, with this utility function you would be willing to wager exactly 50 chips for a 50% shot at 100 chips.
- superlinear: $U(P,B,W) = W*\frac{P^{1+\rho}}{1+\rho}-(1-W)*\frac{B^{1+\rho}}{1+\rho}$ This utility function will overvalue large payoffs and bets. Thus you should be willing to bet more than 50 chips for a 50% shot at 100 chips.
- sublinear: $U(P,B,W) = W*\frac{P^{1-\rho}}{1-\rho}-(1-W)*\frac{B^{1-\rho}}{1-\rho}$ This utility function undervalues large payoffs and bets. Thus you would only be willing to bet less than 50 chips for a 50% shot at 100 chips.
- prospect [52]: $U(P,B,W) = \frac{(W*P)^{1-\rho_a}}{1-\rho_a}-\frac{((1-W)*B)^{1-\rho_b}}{1-\rho_b}$ Prospect Utility is a social science theory which proposes that human actors will value possible gains differently than potential losses. In particular they will be very adverse to losing large amounts of money, whereas gains of large amounts of money are not that different than small amounts of money.
- cumulative prospect [53]: $U(P,B,W) = \frac{(f(W)*P)^{1-\rho_a}}{1-\rho_a}-\frac{(f(1-W)*B)^{1-\rho_b}}{1-\rho_b}$ Cumulative Prospect is a modification to prospect theory to also account for the reality that humans will generally perform poorly when attempting to estimate the actual frequency that events will occur at even when knowing the probability that those events should occur at.

We determined that the Prospect Utility bot was the best choice for a number of reasons. In particular, the prospect utility bot was desirable due to its background as a model for actual human behavior as it relates to decision making under risk [52]. This is desirable in order to demonstrate that our results can be applied to real world games, in particular for quickly creating models of opponent behavior based on samples of real world data, estimating their risk curves, simulating possible outcomes, and in the process refining your own strategy.

In testing even weak versions of the prospect utility bot defeated the linear bot, as well as the super and sublinear bots for many different ρ values. The results of simulating 5 bots types against one another can be seen in Table 3.1.

Table 3.1: Win rates of the 5 Different bot Types 1-game Iterations with 5 players per game. The ProspectUtility bot was by far the strongest.

BotName	ρ_a	ρ_b	$WinRate_{M=1}$
PBot	0.8	0.1	0.69749
ExpBot	0.2		0.01709
LogBot	0.4		0.02312
LinearBot			0.01910
CumulativePBot	0.8	0.1	0.24322

Prospect Theory Utility Tuning

After evaluating the alternative algorithms and selecting the prospect utility bot, we tested different versions of the bot using various parameters varying from 0.1 to 0.9 for each ρ_a and ρ_b. To prevent restricting the field of bots to only a single utility function we determined that the top 10 bots had roughly $0.6 \leq \rho_a \leq 0.9$ and $0.1 \leq \rho_b \leq 0.3$ and at most a 20% deviation in win percentages. We mitigated any restriction to too narrow a field of bots by drawing the field of bot candidates uniformly on ρ_a and ρ_b over those ranges, in units of 0.1. Fig. 3.2 shows an example of a prospect utility curve. It appears that being strongly risk averse in specific is a strong strategy. That is undervaluing possible gains and overvaluing possible losses, rather than just under or overvaluing gains or losses using the same function as the super or sublinear utility functions do, provides a much stronger strategy in this game.

In particular, the no-limit nature leads the linear or superlinear bots to bet too aggressively, thus allowing them to push around the more conservative players for small amounts, ultimately causing them to be eliminated when a more conservative player gets the card they were waiting for. Since the game is multiplayer and the players have finite resources

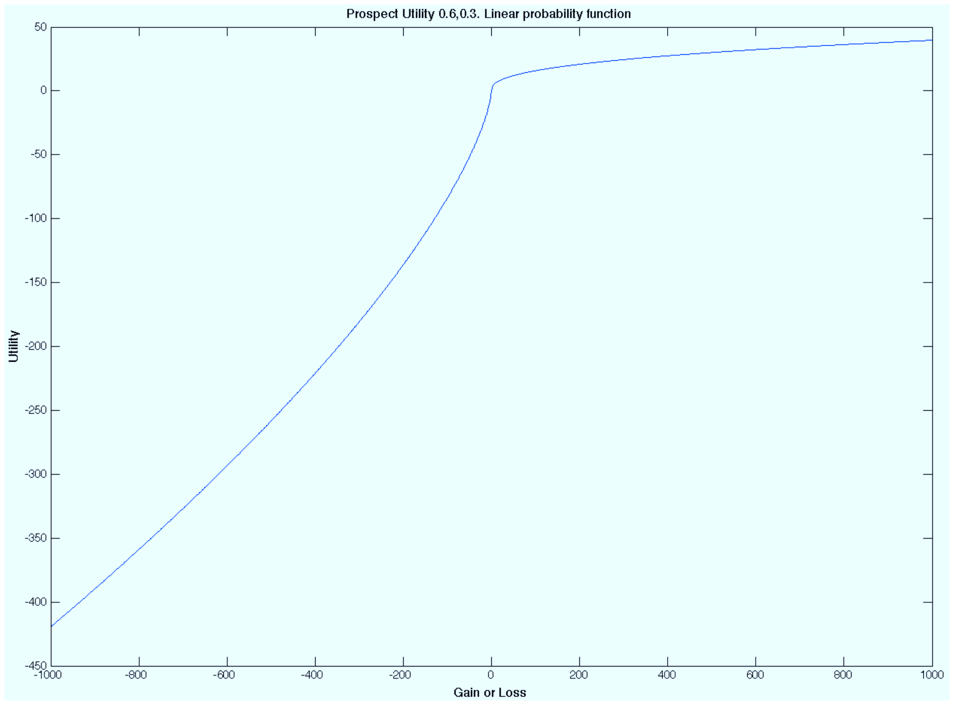


Figure 3.2: The utility curve for potential gains (greater than zero) and losses (less than zero) used by a prospect utility bot with parameters 0.6,0.3. As you can see the prospect utility bots are highly risk-adverse.

winning the overall game is much more important than winning the hand. We find that this too is applicable to real world scenarios. Actors in the games we have proposed would be loathe to wager all of their resources and risk being unable to participate in future rounds unless their success was all but guaranteed or they were on the verge of being eliminated anyway.

Non-Modeling Bots

The bots that do not model behavior have their actions determined by a utility function generated when the bots are added to the game at the start of the round . These bots choose an action by maximizing the size of bet which returns positive utility based on their utility function:

$$\max_B (U(P,B,W) > 0)$$

We choose to do this rather than simply maximizing the utility of the players because simple maximization of utility would imply that the bots always wish to make the smallest bet possible which would allow them to win the pot. However, since there are other players in the game this is not a situation like a lottery, in which the player is presented with the option to spend an amount, B, to be guaranteed the chance, W, to win the pot, P. Instead, they must play against other players. Each bot, by maximizing the bet they will make are able to force out other players who may be more risk averse, or, if other players choose to match the larger bet, they are able to take more resources from those players should they win. Since the expected utility in such situations is still positive, it should be preferable in such a situation to actually make the largest bet for which the player expects a positive utility, rather than the smallest. The desire of the bots to make these larger bets is tempered by the fact that the bots are strongly risk averse, and they will rarely make huge bets. In this case there will often be actual back and forth play, whereas when they are always trying to minimize their bet, the game itself is not very interactive, with most bots very rarely participating if they are at all risk averse. A strategy similar to this is used by professional poker players called value-sizing [54] [55].

Modeling Bot

In order to compete against the fixed strategy prospect utility bots we created a bot that would modify its approximation of the chances of winning a hand of High Card based on observed past behaviors of the players which it is playing against. This bot began each round with no information stored. As each hand was played, and a players hand was revealed at the end of a round (due to competing for winning the round), the bot stored the information of that player's play, indexed on the parameters at the time the action was taken, e.g. the pot size, required bet, number of players in the game, and the number of players who folded, and the card that the player had when they played their action in response to the action parameters.

$$W = \prod_{i=0}^{N} P(C_{mod} > C_i)$$

$$P(C_{mod} > C_i) = \begin{cases} \frac{\binom{C_{mod}-1}{1}}{\binom{S}{1}} & \text{if } max_i = \emptyset \\ 1 & \text{if } C_{mod} = C_i \text{ or } C_{mod} > max_i \\ \frac{C_{mod}-min_i}{max_i-min_i-1} & \text{if } C_{mod} < max_i \text{ and } C_{mod} > min_i \\ 0 & \text{if } C_{mod} < min_i \end{cases}$$

Figure 3.3: Probability Estimation Function for the Modeling Bot

Our bot played with the suboptimal, but highly conservative Prospect Utility strategy with parameters 0.6, 0.3. However, unlike the other bots who evaluate their chance of winning based only on the hypergeometric estimate of their chance of winning based only the card they possess; our modeling bot estimated its chance to win based on the historical play data it had for players who had previously played the same action which they had played this round.

The probability that the modeling bot (MB) will win is, as shown in Fig. 3.3, the product of the estimated probability that MB will beat each opponent, assuming that its probability of beating itself is 100%. If MB sees that there is an opponent who it has data for who only has shown higher cards in the same situations, it has no chance of winning. If MB has some information, MB estimates its probability as being a function of the range of those cards, as if they were uniformly distributed. If MB has no information, MB uses an approximation to the hypergeometric distribution for the remaining players. For example, assuming that the bot has a card ranked 40 and that there are two players for whom MB had no information for, MB would then use the equation in Fig. 3.4. It can treat these quantities as equal because for the size of the sample set we are drawing from and the number of items selected, selecting without replacement doesn't strongly effect the probabilities.

$$\left(\frac{\binom{40}{1}}{\binom{51}{1}}\right)^2 = 0.6151 \approx \frac{\binom{40}{2}}{\binom{51}{2}} = 0.6118$$

Figure 3.4: The hypergeometric can be approximated by calculating the hypergeometric for selecting one item and raising it to the power of the number of items you wish to select.

Results

Our simulation results consist of 2 sets of 1000 1-game tournaments. In these tournaments the modeling bot only has the information about hands played in the current game when modeling opponents. As depicted in the results in Table 3.2, the modeling bot achieves roughly a sevenfold increase in performance for both M=1 and M=2. The average number of hands played per round was 289 with M=2 and 615 for M=1.

Table 3.2: Win rates of the 12 Different bot Types and our ModelBot, 1-game Iterations with 6 players per game.

BotName	ρ_a	ρ_b	$WinRate_{M=2}$	$WinRate_{M=1}$
ModelBot	0.6	0.3	0.39139	0.12510
PBot	0.6	0.1	0.22746	0.14930
PBot	0.6	0.2	0.13444	0.04859
PBot	0.6	0.3	0.05305	0.01813
PBot	0.7	0.1	0.52542	0.35881
PBot	0.7	0.2	0.30512	0.16637
PBot	0.7	0.3	0.16531	0.04928
PBot	0.8	0.1	0.63265	0.44704
PBot	0.8	0.2	0.43141	0.25092
PBot	0.8	0.3	0.20155	0.07869
PBot	0.9	0.1	0.34362	0.13656
PBot	0.9	0.2	0.17043	0.03340
PBot	0.9	0.3	0.03937	0.00511

3.1.6 Conclusions

Compared to the baseline Prospect Utility bot with $\rho_a = 0.6, \rho_b = 0.3$ our modeling bot saw an improvement in play ranging from a factor of 6.9 to 7.5, while using a modeling scheme without heuristic inference, or attempting to wholesale recreate the risk curves of opponents. These results are promising given that the bot had minimal observational inputs, yet still performed quite well.

Additionally, we have proposed that there exists a set of games, which hold practical interest for modeling real world scenarios. Further, we propose that no-limit High Card presents a simple model that can be used to simulate these highly complex scenarios, allowing for preloading of adversary information, in order to simulate possible outcomes from previous observations.

3.2 Attack Graph Games and their Asymptotic Equilibria

In 2013 we worked on a project to analyze attack graphs. The content of this section is from an as yet unpublished paper titled "Attack Graph Games and their Asymptotic Equilibria", authored by George Cybenko and Gabriel Stocco.

This section presents an attack graph optimization problem that is suitable for modeling certain adversarial cyber attack/defend scenarios. The problem formulation is based on representing an attack as a finite directed graph in which the directed edges represent transitions between states in an attack and edge weights represent the estimated cost to an attacker for traversing the edge. An attacker strives to traverse the graph from a specified start node to a specified end node using the shortest directed path between those nodes. On the other hand, the defender seeks to allocate defensive measures in such a way as to maximize the attacker's minimal cost attack path. We study the role that minimal cut sets play in hardening the attack graph and prove that minimal cut sets are optimal defensive investments in the limit even though they may not play a role initially.

3.2.1 Abstract

Computer networks face a number of threats. One particular threat is an adversary gaining access to unauthorized resources by means of exploits. Attack graphs can be used both by a defender to analyze the security of their networks such as in Wang, Noel and Jajodia [56]. Automatic attack graph generation tools have been proposed and can be used by both attackers and defenders, with techniques for creating such analysis tools described in Sheyner et. al [57].

In our formalization of this adversarial situation, we assume that both the attacker and defender, in this prsent case, have access to and knowlendge of the same weighted attack graph. In addition to the actual structure of the graph, the costs to traverse edges must be accurately estimated. This problem has a number of proposed solutions, such as the QuERIES Methodology [58], designed for quantitation of security investment decisions on computer networks.

Given a weighted attack graph, $G = (V, E)$, with edge weights u_j and $\sum_j u_j = T$, an attacker starting at the source s wishes to traverse the network from s to the target t using the minimal cost path. The cost of a path is the sum of the edge weights along the path. The defender has an investment budget R which is invested to increase the weights of edges. If the defender invests x_j in defending edge j, then the net increase in edge j's weight is $\gamma_j x_j$ so that the defender's cost of traversing edge j is $u_j + gamma_j x_j$. The defender's goal is to maximize the minimum cost path subject to an overall investment budget, $\sum_j x_j \leq R$.

Israeli and Wood [59] have shown that when the decisions are binary (that is, invest in an edge or not with fixed costs for the attacker and defender when the investment is made) the resulting problem is NP-Hard. Fulkerson and Harding [60] have shown that the problem of maximizing the minimal cost path can be reduced to a maxflow problem when defender investments and attacker costs are linear and real-valued. Golden [61] uses a similar approach to Fulkerson and Harding to model a scenario where a certain path's cost must be increased by a set amount by modeling the problem as a minimum cost flow

problem. However, these previous works have not performed an asymptotic analysis of the relationship between cut sets and the allocation solution.

We prove that as R becomes large, the maximal minimal path cost grows as $\frac{R}{|C|}$ where C is the minimum cut and $|C|$ is the cardinality of the minimum cut.

To formulate this, let M be the path-edge incidence matrix so that $m_{ij} = 1$ if edge j is on path i and $m_{ij} = 0$ otherwise. The matrix M has a row for every directed path between the start node 1 and the end node n say. Let u be the vector of original weights on the edges in G and γ be the vector of multipliers for defender investments in those edges. That is, for a unit investment of defense in edge j, the resulting increase in cost for the attacker is γ_j.

The problem of maximizing the minimal cost path is the linear programming optimization problem:

Max Min Path Problem (M2P2)

$$Maximize\ z$$

subject to

$$M(u + \Gamma x) \geq z \geq 0$$

$$\mathbf{1}x = \sum_j x_j \leq R$$

$$x_j \geq 0$$

where Γ is the diagonal matrix with the γ_j along the diagonal and $\mathbf{1}$ is the row vector of 1's. Here R be the defender's investment total.

3.2.2 Results

We first consider the case where all $\gamma_j = 1$ and then generalize. Let $T = \mathbf{1}u$ be the existing cost total for all edges before any investment and $|C|$ be the cardinality of the minimum cut set.

Case I: $\gamma_j = 1$

Theorem: Let $\hat{z}$ be the optimal value for the M2P2 problem above. As the defender's investment $R \to \infty$, the minimal maximal cost path, $\hat{z}$, for the attacker grows like $a + \frac{R}{|C|}$, where α is a constant. (This means that, within a constant, the attacker's minimal cost path eventually grows like the defender's investment budget divided by the cardinality of the minimal cut set, $|C|$.

> **Proof**: For an edge j, recall that u_j is the initial cost of traversing that edge, x_j is the additional investment made by the defender and γ_j is the edge weight (cost) multiplier.
>
> Suppose that the cost of the max min path, $\hat{z}$, satisfies
>
> $$\hat{z} \geq \frac{R + T + \epsilon}{|C|}.$$
>
> Menger's Theorem [62, 63] states that the size of the minimum (unweighted) cut between any two nodes X and Y is equal to the maximum number of pairwise edge-independent paths from X to Y. Edge-independent paths are defined to be paths with no common edges.
>
> Accordingly, there exist $|C|$ independent paths between the start and end states in G. On those $|C|$ paths, the total weight is at least
>
> $$|C| * \frac{R + T + \epsilon}{|C|} = R + T + \epsilon > R + T$$
>
> because $\hat{z}$ is the cost of the minimal path. But this is a contradiction because $R+T$ is the total weight of all the edges in the graph. Therefore, because $\epsilon > 0$

was arbitrary,

$$\hat{z} \leq \frac{R+T}{|C|}.$$

On the other hand, if the defender invests exactly $\frac{R}{|C|}$ in each min cut edge then the maximum minimal cost path must have cost $z \geq \frac{R}{|C|}$ because every path from start to end must include at least one edge from the cut set. Therefore, for any R,

$$\frac{R}{|C|} \leq \hat{z} \leq \frac{R+T+\epsilon}{|C|} = \frac{R}{|C|} + \alpha.$$

Because the feasible region for the linear program $M2P2$ is a finite polytope, the optimal value will be achieved on an extreme point of the polytope and therefore for large R max min path $= \frac{R}{|C|} + \alpha$.

This proof can also be extended to integer and rational weights.

3.3 Covertness Calculus

The work covered by this chapter has been submitted as a book chapter in a book titled "Cyber Warfare" to be published soon.

Quantification of the capabilities of kinetic weapons has long been of interest, with techniques formalized for such assessment, for example the Joint Munitions Effectiveness Manuals [64]. In the Cyber domain the stage has been set with qualitative assessment of the important attributes of Cyber munitions [65], but little work has been done on quantification of such attributes.

The Covertness Calculus is a technique for quantifying the covertness of a particular munition against a particular adversary. As discussed in Section 1.2, Figure 3.5 is an overview of the components that are needed for this technique. Malware Vectors is a technique for modeling an adversary. In this section we will discuss the other two components, the modeling of Malware, and the Calculus itself.

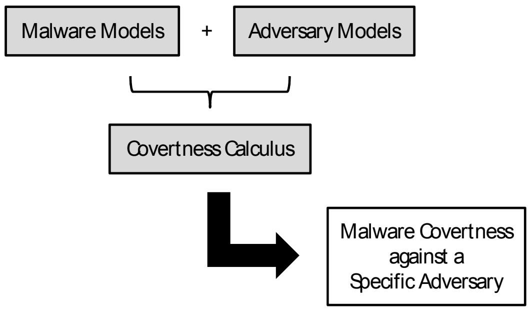


Figure 3.5: *The overall concept takes models of malware and adversary behavior, and produces a covertness measure against a specific adversary's defense architecture.*

3.3.1 Malware Modeling

We propose a methodology for modeling malware based on its observable attributes. Malware, like all software, produces measurable effects while it is runnning. For example, the malware uses some portion of the CPU of the system it is running on as well as some portion of RAM. Based on its purpose and the objectives of the creator it may also use space on the disk or capacity of the network connection of the system. All of these activities are observable using system or network monitoring utilities. The model of malware then is numerical properties of the observables which the malware generates. To find these values, it would generally be required to run the malware and measure its observable properties.

3.3.2 The Covertness Calculus Block Diagram Model

We model covertness as a capability calculus combining a model for the malware used by the attacker and the detection capabilities of the defender. The covertness of a particular piece of malware is the probability that it is not detected by the defender. We assume that both the attacker and the defender are able to measure the observables generated by the attacker's malware.

The defender is aware of and able to modify their detector rules at will, however, given the complexity of these systems and the great cost of downtime caused by false positives, defenders will generally tune their systems to only catch egregious violations of the rules.

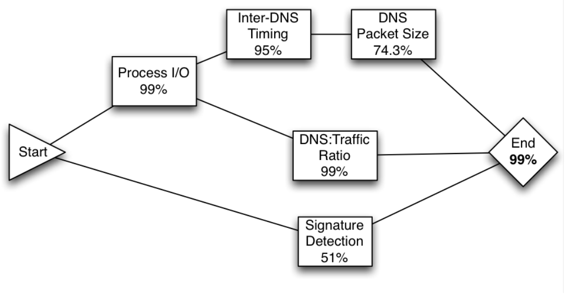


Figure 3.6: *A block diagram example with calculated probabilities for each logic node being triggered.*

As a result, we suggest that there is almost always a way for an attacker to accomplish their operational objectives with a low probability of detection. More to the point, among all sets of possible observables that an attacking piece of malware may exhibit when accomplishing a mission, there is some combination of those observables that minimizes the probability of detection. The key for the attacker, then, is to discover a way through the carefully constructed house of cards that minimizes the probability of detection. The defender, for his part, may realize that it will take automated techniques to model how an attacker would avoid his ruleset in order to improve it.

The covertness calculus is implemented as a reliability block diagram of the adversary logic which interacts with the observables generated by the malware. Each block's probability of success is equal to the probability that the block is triggered by the malware under test. Thus the probability of the circuit as a whole being reliable is the probability that the malware is detected. The covertness of the malware, is then $1 - P_{detect}$.

In the example presented in Figure 3.6, the malware is very likely to be detected (99%), and thus is not covert. Given that this is the case, an intelligent attacker will tune the observables generated by their malware so that it will be able to operate for a longer period of time before being detected on the defender's network.

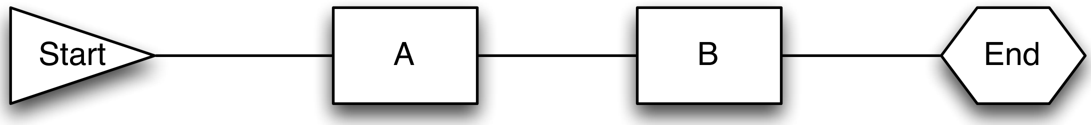


Figure 3.7: *An example block diagram containing two elements in series.*

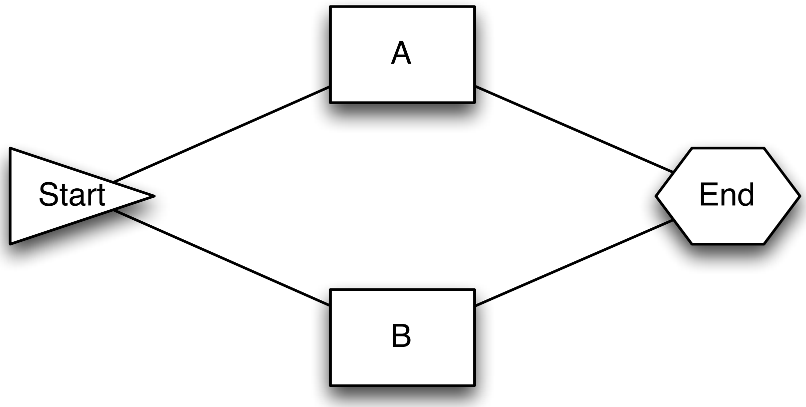


Figure 3.8: *An example block diagram containing two elements in parallel.*

Bounds on Covertness

In reliability theory the probability that two components, A and B, with reliabilities $P_A and P_B$ in series, such as in Figure 4.2, fail is $1-(P_A * P_B)$. If either component fails the system will fail. Alternately, components can be set up in parallel, where every component must fail for the system as a whole to fail. For example, in Figure 3.8, both C and D would have to fail. The probability of failure for the parallel system is $(1-P_A)(1-P_B)$.

———-

These two scenarios provide bounds on the potential covertness of software against defensive systems. If the defense logic is configured in serial, each rule must trigger in order to trigger an alert. That being the case in order to remain covert against such a system, an attack need only not trigger any one component. On the other hand, the lowest covertness is that of an attack against a fully parallelized defense logic. In such a scenario if any sensor triggers an alert would be raised, and the attack would not be covert.

However, real world designs would clearly incorporate both these techniques. Components in a series would reduce incidence of false positive, for example confirming that a suspected attack has a second property of interest before issuing an alert. Parallel design would be used as well, in order to increase the number of potential attack scenarios the defender could defend against, and increase the true positive rate.

Chapter 4

Analytic Analysis

The attacker has two objectives: the first of which is to find a single counter example to the defender's logic and the second is to fully cover the defender's logic. We investigate the effect that the properties of the defender's rules have upon these two objectives, and formulate how the attacker can efficiently accomplish them.

4.1 Assumptions

We assume that the defender's logic is equivalent to a monotone K-DNF. This means that the defender can have any number of logical terms in its detection logic, but each term must be a conjunction of at most three variables.

Additionally, we assume that a 3-DNF represents a single unique capability to the attacker; that is, the attacker would not design an attack with fewer than three observables. We choose 3-DNF because 2-DNF was too simple and since the size of the problems grows factorially we want the minimum value which allows us to extrapolate for other values. Further, we assume that all capabilities are equally valuable to the attacker, and that the attacker has a neutral appetite for risk.

Definition Let A be the number of boolean variables in a monotone conjunction term under test. Let N be the total number of boolean variables. Let K be the number of boolean variables in each term of a monotone DNF expression for which we want to find counterexamples. Unless otherwise stated, random selection means selection from a uniform distribution.

Addressing the first objective, the attacker will want to maximize its utility function. Remember that the attacker values all capabilities equally; thus, this utility function can be expressed generally as

$$U(A) = \binom{A}{K} - C$$

The attacker's cost will vary with the number of probes sent. We propose that the attacker would desire to modulate this downside by looking not just at the utility gained from a potentially undetected probe, but also the probability that a probe would go undetected before it is sent. We work towards such an expression which we find in Lemma 4.1.2 to be

$$P_M \leq \prod_{i=0}^{M-1} (1 - \frac{\binom{A}{K}}{\binom{N}{K} - i})$$

Using this expression we are able to simulate an attacker who varies the size of their attack vectors based on a moving estimate of P_M, seen in Chapter 5.

Lemma 4.1.1. *The probability, P_1, that a randomly selected A length conjunction of boolean variables is not captured by a single randomly-selected K length conjunction is: $P_1 = 1 - \frac{\binom{A}{K}}{\binom{N}{K}}$ where $A \leq K$.*

Proof. There are $\binom{A}{K}$ ways to select any K items out of A items without replacement. Thus, there are the same number terms of K variables which capture a term of A variables. In the space of N variables there are $\binom{N}{K}$ possible terms of K variables. It follows that the probability that a randomly selected term of length A is covered by a term of length K is

$$\frac{\binom{A}{K}}{\binom{N}{K}}$$

Trivially, the probability of the negation is

$$P_1 = 1 - \frac{\binom{A}{K}}{\binom{N}{K}}$$

$\square$

Lemma 4.1.2. *The probability,P_M, that a randomly selected term of A variables is not a subset of an M term K-DNF expression is bounded by*

$$P_M \leq \prod_{i=0}^{M-1} (1 - \frac{\binom{A}{K}}{\binom{N}{K} - i})$$

Proof. From Lemma 4.1.1 the probability that any given term of A variables is not a captured by a random term of K variables is $\frac{\binom{A}{K}}{\binom{N}{K}}$. Since we define the K-DNF expression to be comprised of unique terms, we select without replacement from those terms. Thus the probability P_m that the mth K variable term is not a superset, given that the previous $m-1$ terms were also not supersets is

$$1 - \frac{\binom{A}{K}}{\binom{N}{K} - m - 1}$$

Generally, the probability that M terms will not be a superset is bounded by

$$P_M \leq \prod_{i=0}^{M-1} (1 - \frac{\binom{A}{K}}{\binom{N}{K} - i})$$

$\square$

Given the expression from Lemma 4.1.2, the attacker can estimate the likelihood of an expression of arbitrary length A being a subset of the defender logic given an estimate of the number of rules ($\hat{M}$) that the defender has. Given this estimate, N and K, the attacker can find the expected value of a probe of size A is the probability that that probes isn't detected multiplied by the value of that probe.

$$U(M) = P_M * \binom{A}{K}$$

The attacker does not know M. Further, recall that the attacker is interested in a single counterexample to the defense logic and will stop immediately when a probe is not detected. Further, recall that the attacker is concerned with finding an optimal solution as a function of the number of probes required to converge as well as the actual utility provided by that solution. In the interest of ensuring that the attacker converges on a solution quickly, we can update the attacker's estimate of $\hat{M}$ to optimal size of the next probe.

This technique works regardless of the value of C, even if C varies with the number of probes sent so long as C does not vary A. Specifically, at each iteration, the optimal action taken by the attacker will be the largest expected utility action, without having to concern ourselves with the actual value of C. We can additionally justify ignoring C by application of the "Sunk Cost Fallacy" which states that at any time t one should never consider the costs already spent and irretrievable, only the current costs which can be expended, and the actions that can be taken.

We discuss using such a technique to modify the attacker estimate of M, $\hat{M}$ in our large scale simulations in Section 5.4.

The attacker may also be interested in another bound on M: the fewest number of expressions which would capture all terms of size A.

Lemma 4.1.3. *Any $\binom{N}{K} - \binom{A}{K} + 1$ term K-DNF expression is a superset of all A-DNF expressions.*

Proof. Taking P_M from Lemma 4.1.2 and setting it to 0, we solve for $i = M - 1$

$$1 = \frac{\binom{A}{K}}{\binom{N}{K} - i}$$

$$i = \binom{N}{K} - \binom{A}{K}$$

Thus any $\binom{N}{K} - \binom{A}{K} + 1$ length, K-DNF of randomly selected unique terms is a superset of all A-DNF expressions. $\square$

Finally, either party may be interested on the bound on the size of the optimal set of K-DNFs which cover all A-DNFs. We were able to find empirically that this value is bounded by:

$$\left\lceil \frac{\binom{N}{K} - \binom{A}{K} + 1}{\binom{A-1}{K-1}} \right\rceil \leq length \leq \left\lfloor \frac{\binom{N}{K} - \binom{A}{K}}{\binom{A-1}{K-1}} \right\rfloor$$

Lemma 4.1.4. *The shortest length K-DNF that is a superset of all A variable conjunctions is at most $\left\lceil \frac{\binom{N}{K}-\binom{A}{K}+1}{\binom{A-1}{K-1}} \right\rceil$ terms and at least $\left\lfloor \frac{\binom{N}{K}-\binom{A}{K}}{\binom{A-1}{K-1}} \right\rfloor$ terms.*

Discussion

We calculated the minimum size fully covering set for a number of values $2 \leq K \leq 7, 3 \leq A \leq 8$. We found that this pattern held for those values. To approach this problem from another perspective, we can view this as a graph theory problem. Specifically, we looked at the case $K = 2, A = 3$. Graph Theoretically this problem then becomes "What is the fewest number of edges on a graph on degree N where no triangle can be created without including one of those edges."

For example Figure 4.1 is an optimal solution to $N = 5, K = 2, A = 3$.

Finding a solution to this problem is non-trivial.For example, we cannot reliably find a solution using a greedy algorithm. For example for $N = 5, K = 2, A = 3$ a greedy algorithm requires one additional edge to fully cover all triangles as shown in Figure 4.2.

Specifically for cases where $K = 2, A = 3$ we find a solution which is a lower bound than upper bound found in Lemma 4.1.3. Specifically we divide the N nodes into 2 groups, one of size 2 and one of size $N - 2$. We then fully connect each group. To form a triangle one must naturally select three nodes. There are four options for ways to select nodes, but

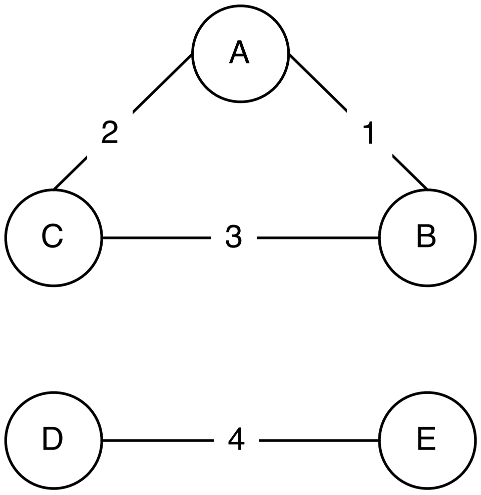


Figure 4.1: *An optimal (minimal edge) solution for* $N = 5, K = 2, A = 3$.

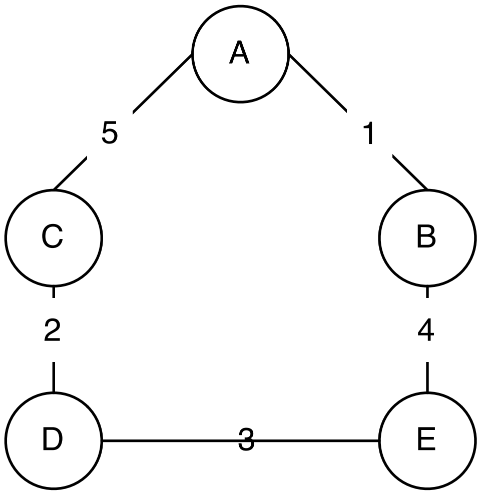


Figure 4.2: *A non-optimal solution for* $N = 5, K = 2, A = 3$.

because the first group only has two nodes, only three of the four are actually possible to occur. In the first case, all three nodes are selected from the larger group. Since each of these nodes are connected to the other, the triangle they form is certainly covered by the chosen edges. In the second case, two are selected from the larger group and one from the smaller. In this case, again, the two nodes from the larger group are connected, and thus the triangle formed is covered by an edge. In the final possible case, we select one node from the larger group and both of the nodes from the smaller. In this case, of course the two nodes in the smaller are fully connected and thus the triangle is covered by an edge.

The converse problem, specifically for $K = 2, A = 3$ has been proven by Mantel's Theorem, which states that any n-vertex triangle free graph can have at most $\frac{N^2}{4}$ edges [66]. For example, for $N = 5$ as above, $\frac{N^2}{4} = \frac{25}{4} = 6.25$. Since we have a total number of $\binom{N}{K} = \binom{5}{2} = 10$ possible edges, increasing to 7 edges would require creating a triangle, and thus $10 - 6 = 4$ edges is the smallest set of edges which could be created which must be included in any triangle.

Unfortunately we were unable to extend this technique to work on larger values of K or A, though a technique using a modification to Turan's Theory[67] may be able to provide insight. Turan's Theory is a generalization of Mantel's Theory for arbitrary A with $K = 2$. While we do not examine that here, it could be a future extension of this work. However, the values from graph theoretic analysis were consistent with the brute force testing derived values we found. Those values can be seen in Table 4.1

4.1.1 Implications for Learning a Single Counterexample

We assume that our attacker has a tradeoff between the value of the counterexample they have and the number of probes that it takes them to find such a counterexample. Further, we made the assumption that our attacker values each expression linearly in the number of K-DNF implicants it has. Recall from Lemma 4.1.2, we can calculate the expected probability of an A-DNF term being captured by M K-DNF terms. Consequently, the attacker utility

N	K	A	Actual Min	Hypothesis Max
6	1	1	6	6
6	1	2	5	5
6	1	3	4	4
6	1	4	3	3
6	1	5	2	2
6	1	6	1	1
6	2	2	15	15
6	2	3	6	7
6	2	4	3	4
6	2	5	2	2
6	2	6	1	1
6	3	3	20	20
6	3	4	6	6
6	3	5	2	2
6	3	6	1	1
6	4	4	15	15
6	4	5	3	3
6	4	6	1	1
6	5	5	6	6
6	5	6	1	1
6	6	6	1	1
7	2	3	9	10
7	2	4	5	6
7	2	5	3	3
7	2	6	2	2
7	2	7	1	1
7	3	4	*	11
7	3	5	5	5
7	3	6	2	2
7	3	7	1	1
7	4	5	7	8
7	4	6	2	3
7	4	7	1	1
7	5	6	4	4
7	5	7	1	1

Table 4.1: *The actual minimal values discovered via brute force test and the hypothesized ceiling on the size of the minimal set. This table continues in Table 4.2. Values marked with a "*" were not computed due to runtime.*

N	K	A	Actual Min	Hypothesis Max
5	1	1	5	5
5	1	2	4	4
5	1	3	3	3
5	1	4	2	2
5	1	5	1	1
5	2	2	10	10
5	2	3	4	4
5	2	4	2	2
5	2	5	1	1
5	3	3	10	10
5	3	4	3	3
5	3	5	1	1
5	4	4	5	5
5	4	5	1	1
5	5	5	1	1

Table 4.2: *The actual minimal values discovered via brute force test and the hypothesized ceiling on the size of the minimal set. Continued from table 4.1*

function is a function of the number of terms in the defender logic.

$$U(M) = \binom{A}{K} * P(M)$$

In this situation the attacker is unaware of the exact number of terms in the defender logic, thus the attacker must have some estimate for $P(\hat{M})$. We discuss some techniques for updating $P(\hat{M})$ to aid the attacker in quickly converging on a solution, and the implications of various choices for $P(\hat{M})$ in Section 5.4.

4.1.2 Implications for learning the Full Logic

There is a significant body of work on learning K-DNF expressions using Membership and Equivalence queries. Of particular interest, Jackson [32] finds that K-DNFs can be learned in polynomial time, specifically $O(\frac{NM^8}{\epsilon^{12+c}})$ to a precision $1-\epsilon$ using only membership queries. Evaluating this for values that we are likely to encounter in the scope of this project, $N = 30, M = 1000, \epsilon = \frac{1}{20}$, we find the runtime to be $1.2 * 10^{44}$. Unfortunately, while we could expect that this would grow polynomially in N, the base level at which it starts is high enough to be untenable.

Alternately, since we examine only the subset of monotone K-DNF, we know that brute force membership queries would give 100% accuracy and would be in all cases $O(\binom{N}{K})$, simply enumerating the terms of size K, which for the same values would be 4,060.

We could improve the lower bound at the cost of the upper bound by sampling with DNFs with more variables than K. Remember, for each $A > K$ there are $\binom{A}{K}$ K-DNFs that are an implicant of the A length term. Thus if we test with $\frac{\binom{N}{K}}{\binom{A}{K}}$ A-DNFs each chosen to not share any implicants with the others, whether or not those terms are members or not informs us whether any of the implicants of the term are in the defender logic. For any A length term that is a member, we would need to check the $\binom{A}{K}$ potential K length implicants. For the remaining A length terms we know that none of the implicants are in the logic. This give us runtime in the number of membership queries

$$\left(1 + \frac{1}{\binom{A}{K}}\right) \times M + \frac{1}{\binom{A}{K}} \times \left(\binom{N}{K} - M\right)$$

Note however that finding the set of such A-DNFs of minimal length can be computationally difficult.

Chapter 5

A Real World Approximation of the Vulnerability Discovery Algorithm

5.1 Description of the Problem

A defender has static defense logic comprised of rules in an IDS. These rules can be represented by a Monotone DNF expression. This expression covers a subset of all of the possible logical expressions of combinations of observables. If we assume that these expressions are boolean functions with n variables, or distinct observables, there are 2^n possible combinations of truth values. However, each of these combinations can be either true or false at any particular assignment of values in our boolean function so there are in fact 2^{2^n} possible boolean functions the defender could choose to detect[1]

We here attempt to find the optimal counter-example to the defender logic. Formally, the assignment of values to variables that is not covered by the defender logic while providing the most utility to the attacker of any such assignments. Though this gives the attacker a lower fidelity in learning the defender's overall logic, it dramatically reduces the complexity of the problem back down to 2^n.

[1]Remember that the Defender cannot simply "detect" everything, since legitimate activity also occurs on their network, so they must choose to detect some subset of the available assignments.

5.2 Technique

The attacker identifies which capabilities they will require and sets priorities based on how desirable each set of capabilities will be in terms of completing the mission. The attacker will then choose the set of capabilities which are most likely to generate observables which may be out of line with normal activity on the defender's systems. These capabilities will be the focus of the probe testing to come.

The attacker launches a series of probes to the defender's network. Each probe is designed only as an attempt to gauge the defenses of the defender. The probes contain no proprietary logic or payload. They simply attempt to arrive on the defender's systems and execute, thus generating observables at a known level. The attacker then measures whether the probe was able to execute or not, i.e. if it has been detected. If the probe was detected, the attacker is able to add a logical expression covering the probe's observables to the logic that the attacker knows the defender's network will detect.

Under the assumption that the defender has a static detection strategy, once the attacker finds a probe combination that does not trigger the defender's defensive logic, they have found a covert way to carry out their mission.

5.3 Experiment

We were unable to find a set of rules used by an ArcSight installation. Instead, we instead enlisted the help of Maj. Patrick Sweeney to develop a set of defender logics that we could then attempt to compromise.

We first generated sets of observables from a system running regular network browsing activity, and then from the same machine running DNS fluxing malware.

DNS fluxing is a technique employed by malware that uses a shared random seed on the distributed malware and on the command and control server. Thus, whenever the bot is not able to contact the command and control center, it will look to the random number

Rank	Metric Name	Relevant
1	Free System Page Table Entries	No
2	Connections Established	Yes
3	System Driver Resident Bytes	No
4	% Registry Quota in Use	No
5	Working Set Peak Memory Usage	Possible
6	Virtual Bytes Peak	Possible
7	Process Count	No
8	Disk Free Space	No
9	Disk Free Megabytes	No
10	Cache Bytes	Yes

Table 5.1: *Metrics found to have most dissimilar distributions in perfmon captures for web browsing vs. domain fluxing.*

generator to generate a new domain name. The bots will then send DNS requests for these domain names and if the command and control infrastructure is still running, it will be registering new domain names ahead of the bots.

5.3.1 Defender Logic Creation

The trouble is, of course, that these techniques are generally very noisy and look nothing like normal web traffic. It is very unusual for a system to generate lots of DNS requests and even more unusual for the ratio of DNS requests to HTTP traffic to be skewed toward DNS requests.

With the data, described in 5.3, we used the Mann-Whitney test to find the data distributions that were most dissimilar. From these numbers we trimmed those that did not change during the course of a single experiment ($\sigma \leq 10^{-4}$), and ranked them by dissimilarity, selecting the top ten metrics which are listed in Table 5.1.

In addition to the metrics from PerfMon, a built-in Windows monitoring program, (see Table 5.1) a number of metrics based on raw network traffic were developed.

We then developed a defender logic using the following metrics: Connections per Second, DNS Queries per Second, DNS replies not found per second, Datagrams sent/received

Metric Name
DNS Queries per Second
DNS replies “not found” per Second
σ of inter-DNS query timing

Table 5.2: *Network Capture Metrics found to be most dissimilar in network captures for web browsing vs. domain fluxing.*

Logical Symbol	Metric Name
A	DNS Queries per TCP Connection
B	DNS Replies “not found” per DNS Query
C	DNS Packets per Packet
D	$\frac{1}{\sigma}$ of inter-DNS query timing

Table 5.3: *Final Defender Design Metrics.*

and standard deviation (σ) of inter-DNS timing. As a function of these 5 variable respectively from A to E we developed the defender logic

$$ABDvACDvABEvACE$$

We simplified these five metrics to four more meaningful ones by removing B and turning A and C into ratios. These 4 variables are listed in table 5.3

As a function of these four, the expression above becomes:

$$AC \vee ABC \vee AD \vee ACD$$

Which simplifies to:

$$AC \vee AD$$

We decided on a value range from zero to ten. Ten was considered to be similar to the actual values of the domain fluxing malware. The defender values were then set as a function of the range from zero (none) to ten (domain fluxing) slightly above where the web browsing activity fit in the range. Giving us:

$$A_3C_2 \vee A_3D_4$$

5.3.2 Simulation

Given this defender logic, we ran two attack simulations against it. In the first, the attacker does not consider the cost of producing probes and only bases his utility on the utility of the derived covert malware.

In the second experiment the attacker considers the tradeoff between having to create more probes and the potential payoff from a sucessful undetected probe by discounting the potential utility from a set of capabilities by a function of their value. The logic here is that the larger the value of the observables generated by a capability, the more likely they are to be detected.

We first approached the problem by converting the attacker's requirements to a binary integer linear programming problem. To do so the attacker converts his integer constraints to binary constraints by converting each variable into $\lceil log_2(S-1) \rceil$ binary variables.

Binary Integer Programming Example

For example, with $S = 4$ and variables $\{A, B\}$, we define the attacker's utility profile:

Each rule implies all rules that capture it. So we subtract out the utilities of any of its

A Value	B Value	Utility
3	3	10
2	3	5
3	2	5
1	1	1

Table 5.4: *Example Attacker Utility Profile for Binary Integer Programming Example.*

Implicants[2]

Now we have our vector of $c = [1, 4, 4, 1]$. Next we need our matrix for the coefficients of the inequalities, Z[3] We set our coefficients $Z_{n,j} = 2^{n-1} \forall j$.

Finally, we add the attacker constraints based on their knowledge of the defense design that the defender has. For example if we know that the defender has the rule A_3B_2 we add the constraints: $Z * [1, 1, 1, 1,] \leq 3, Z * [1, 1, 1, 0] \leq 2$ to ensure that we cannot make the assignment of values A_3B_3 or A_3B_2 both of which would be covered by the defense logic we already know. The optimal next probe here would then be A_2B_3 with a Utility of 5. The attacker would send this probe and continue to iterate until a probe went undetected.

Brute Force

This problem can also be solved using a brute force method that generates all the possible values for the assignments of values and tests them against the defender logic. If an assignment of values would not be detected we find the capabilities that are covered by that assignment and the corresponding utility. Remember from Section 2.1 that a base 2 expression has 2^n possible assignments of n values. Here, we have base 10, and thus 10^n, or specfically 10^4 possible assignments of values. Since we have restricted that the attacker must have all components in his attack, our total search space of is 10^4. This approach dominates an approach considering all combinations of desires of the attacker,

[2]This requires the assumption Utility is monotonically increasing with increased capabilities.

[3]Normally this is denoted A but we already use A as a logical variable in this example.

A Value	B Value	Utility
3	3	1
2	3	4
3	2	4
1	1	1

Table 5.5: *Example Attacker Utility Profile for Binary Integer Programming Example, Step 2.*

which would run in 2^c where c is the number of capabilities specified in the attacker's utility function, keeping in mind that to represent the full expression using a binary program c grows logarithmically with S.

5.3.3 Results

To run the simulation we had to establish priorities for the attacker. A naive implementation may provide linear utility for each capability in its dimension, therefore, a value of ten would provide ten utility, and a value of five would provide five utility. Thus: $a_5b_4c_3d_2$ would provide fourteen utility.

Using this technique, we found the solution: $a_2b_9c_9d_9$ which fulfilled the requirements of having every element active and not being covered by the defense logic. Converging on this solution took 103 probes.

Upon closer observation this solution belies an overlooked externality. The attacker would have to keep a very minimal DNS per connection ratio (A) but would be able to maximize the DNS requests per packet rate (C). This in effect means that C is contrained by A. To reflect this, we allow the attacker to specify additional constraints on realized utility. For example, the attacker could specify that:

a_xc_y

provides $2 * min(x, y)$ utility, restricted to the smaller value provided by A or C. Given this constraint, the above solution would actually have the value equivalent to:

$a_2b_9c_2d_9$

which is non-optimal if we consider the underlying realities. The next iteration reduces the total number of utility values by pairing capabilities. So rather than forty potential capabilities, we can reduce this to ten without significant meaningful loss in fidelity.

Looking at the four parameters under test here, it is clear that A and C are intrinsically tied with respect to the objectives of the attacker.

Further we judge that the key component of DNS fluxing is to keep the fluxing DNS

A	B	C	D	Utility
9		9		20
7		7		15
4		4		10
2		2		5
1		1		1
	3		2	10
	2		3	10
	3		3	5
	4		4	5

Table 5.6: *Attacker utility values vs capabilities. Note that the utility for each capability is also provided for any capability that dominates it. So a_2c_2 also provides the utility of a_1c_1. An empty entry in a cell indicates that we don't care about the value of that parameter.*

request per time rate high enough to mantain availability. Common DOS attacks against DNS fluxing involve offline evaluation of the algorithm that the malware uses and preregistering the domains in order to hijack the command and control channel. As long as this rate, which incorporates both B and D, is high enough, it should provide some financial disincentive to such an attack. We judge that for b_x and d_y, $x * y \geq 6$, the rate of fluxing DNS requests is high enough. Significantly higher rates do not provide much more utility; thus, there is a drop off in marginal gain.

With these parameters, we find a solution of $a_2b_4c_2d_4$ with only eight iterations.

A further modification would consider the cost of creating probes or of having probes detected. If we allow some relaxation of the assumption that the defender keeps their detection strategy static, then it is conceivable that they could begin to modify their defenses in reaction to a large number of detected probes. We theorize that as the sum of squares of the levels of parameters increases, the probability of detection increases in a manner similar to logarithmically toward the 100% asymptote.

In particular, we posit that this relationship approximates $\sqrt{(}x/MAX)$ where MAX is the maximum possible sum of squares, and x is the sum of squares of the particular expression.

With this modification we find the solution $a_3b_2c_3d_2$ with only two iterations. While this solution has a slightly lower gross utility, depending on the cost of a probe the net utility may yet be higher.

5.4 Simulation

Having validated our approach using the single handcrafted example above, we set out to see how design decisions by a defender would effect the efficacy of these attacks. To do so, we generated 1000 defenders each with M-term randomly generated 3-DNF expressions on 30 variables, where M varied from $500 \rightarrow 2000$ in increments of 250. We investigated in particular what the effect that the number of terms in the defender logic had on attacker gain. We were also interested in how much the attacker could negate this effect, and varied the value of $\hat{M}$ on the same range.

5.4.1 Malware Vectors Algorithm

The Malware Vectors algorithm used for this experiment is provided below. This albgorithm attempts to maximize the payoff to the attacker by keeping a continually updating estimate that any particular probe will be captured, allowing for testing of optimally sized probes.

- The utility for an attack of size A is $\binom{A}{K}$

  1. The attacker calculates P_M with their estimate for M and each possible value of $K \leq A \leq N$
  2. The attacker chooses an A which provides the largest expected utility
  3. The attacker generates random combinations of A variables until such a combination isn't captured by the known defense logic.
  4. The attacker tests that against the defense logic.

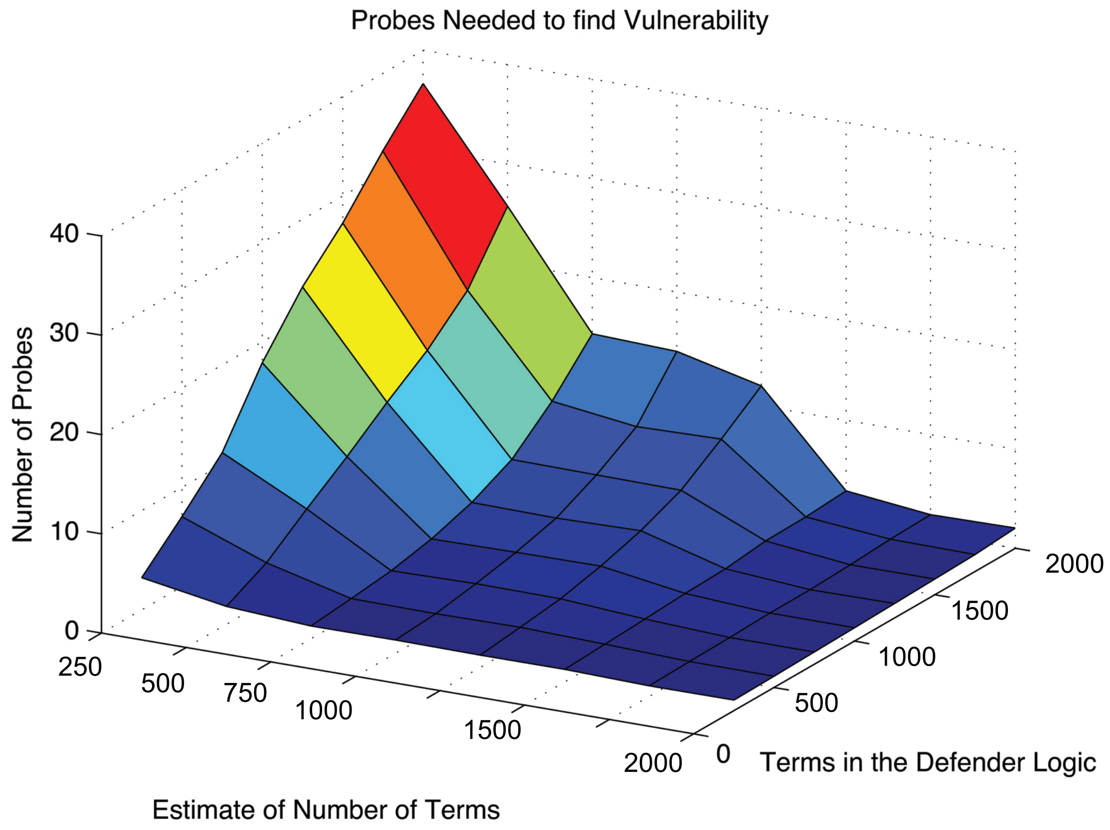


Figure 5.1: *Number of Probes vs. Defender Logic Size vs. Estimate of Logic Size*

5. If the attack is successful, we are done.

6. If not, increase our estimate for M by 1 and goto 1.

The runtime of this algorithm in the worse case is $O(2\binom{N}{K})$ in the worst case, when the defender has nearly all logical statements and the attacker assumes they have 0. This situation is unlikely however, as it would preclude the defender from obtaining any positive utility on their network to triggers alarms based on all the activity. Further we are assuming that the defender is using blacklists not whitelists, and thus the majority of the space will be open, with only known bad areas triggering alerts. As such, in practice, the run time is much better.

If the attacker underestimates M by a large margin, he will expend a large amount of additional resources testing probes. However, a lower estimate for M also allows the attacker more chances to test larger samples. Optimization of the technique used to modify

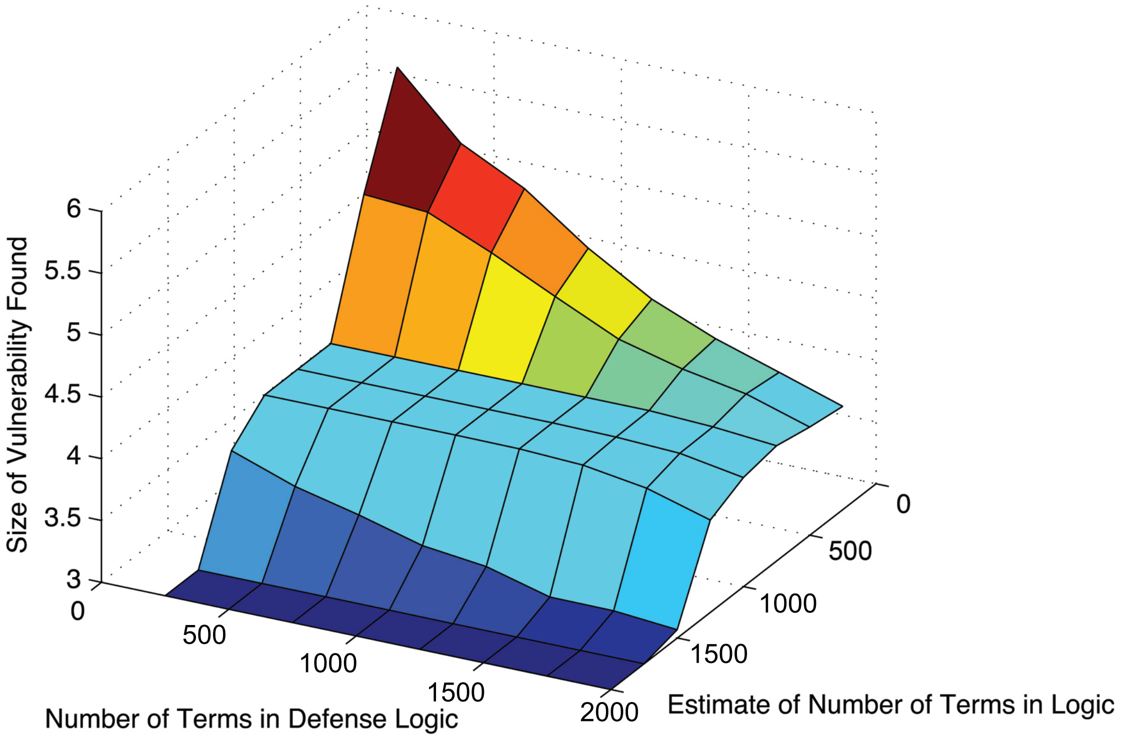


Figure 5.2: *Vulnerability Size vs. Defender Logic Size vs. Estimate of Logic Size.*

$\hat{PM}$ could be performed based on the particular costs experienced by the attacker. In this case, we attempted only to modify $\hat{PM}$ when we gained actual knowledge that the estimate for M must shift.

5.4.2 Relation to SAT Problem

The generation of random simulated K-DNF formula has been a topic of discussion with respect to the satisfiability (SAT) problem. The SAT problem attempts to, given a boolean formula, find an assignment of variables to that formula which satisfies the formula, or simply to find if the formula is satisfiable or not. Mitchell et. al[68] discuss particular sets of boolean functions which based upon their ratio of clauses to variables are easier or harder to satisfy. In particular, the authors discover that at a ratio of slightly over 4 clauses per variable the probability of a formula drops below 50% and begins to quickly approach 0%. In our work, we guarantee that the formulas are satisfiable since they are DNF and Monotone. Consequently, we do not expect to experience the same relationship between clause number and variable size. Malware Vectors could be described as trying to solve a blind non-satisfiability problem, and we expect that as the number of unique DNF clauses increases the probability of not being satisfied follows the statistical measures we establish in Chapter 4.

Chapter 6

Conclusion

We have shown that it is trivial to discover a counter-example to the defender's logic, given the assumption that the defender cares about false positves.

Further we prove that it is easy to find a bound on the number of statements that the defender has in their defense expression, irrespective of the number of variables, the number of terms that they have and the size of those terms given that the terms are of the same size and that the size is known to the attacker.

Finally, we have found a bound on the number of expressions required to learn the underlying expression in full. Under the assumption that the defender desires to avoid false positives, this bound is always a smaller number of statements to test than the full exploration of the factorial space of the K-DNFs.

This work demonstrates that the current model of security by obscurity vis a vis the security of these rules is insufficient. A key problem that is not addressed in this field is that what these defense systems attempt to do is derive intent from actions. Unfortunately, that process is very difficult to carry out.

One stop-gap defense against this technique is to bring the model of moving target defense to the defense logic used by a system. First, we must allow the assumption that attacks that are detectable generally persist for a long enough time to be detectable at more than one moment. Given this assumption, we can devise a technique that would change the detection rules used at any timepoint in a randomized manner. This being the case, it would be possible to trick an adversary into believing that a probe is safe, when in fact it is not. Any attacks that are more persistent than a probe would still be detected, since the detection rule would come back periodically.

6.1 Potential Future Work

We forsee that there are a number of avenues for potential future work that builds upon our findings. In terms of analytic work, there is room to validate our hypothesis about the minimal size set of K-DNF terms. While the range we found experimentally is small, there is certainly room to find the exact formula for such examples. The graph theoretic approach seems to be promising here, particularly Turan's theorem is a generalized version of Mantel's theorem for K_{r+1} size clique free graphs. Mantel's theorem validated our experimental results for small values, and thus it is possible that the generalized Turan's theorem could provide the basis for an analytic solution. Additionally, while we do not propose such an algorithm here, it should be possible to discover the actual optimal sets, and not just calculate their sizes. Secondly, further analytic work could expand the full logic learning algorithm which uses A length terms to accept terms of arbitrary length to allow to contributions toward learning the full logic while using the Malware Vectors algorithm with variable A.

On the experimental side, it would be interesting to see how the simulated results worked with higher fidelity captures of real defense logic and attack objective functions. A first step towards this may be to derive some logical expressions to represent real defense systems based on an obtained set of rules. Attacker values could be derived via value elicitation or other methods. Finally, these experiments could be expanded to test a real world dynamic malware generation algortihm competing against a maintained defensive system, to validate that the experimental results are indeed a model applicable to the real world systems we modeled here.

Appendix A

Licenses

A.1 Written Content License

A.2 Creative Commons Attribution NonCommercial Share-Alike 4.0 International

A.3 Software License

Appendix B

Source Code

```
function [v1,design,iters] = mwDesignSim(MAXVAL,sets,vals,
   defense)
%Max is the maximum value any entry can have.  These are
   integers. Min 0.
%Sets is the list of design criteria desired by the attacker
   .  One row per.
%Vals is the corresponding values for the attackers design
   criteria.
%Defense is the defender's actual detection logic.
    n=size(sets,2);
    t1 = ones(1,n)*(MAXVAL-1);
    design = zeros(0,n);
    iters = 1;
    disj=isDisjunct(t1,defense);
    if(disj)
        v1=t1;
        %If the defender doesn’t detect an attack with
           everything turned on
        %We can just use that.
    else
        %We add the detected design to the logic matrix
        design = [design;t1];
        for i=1:(MAXVAL^n)
            iters=iters+1;

            t1 = generateMWBinaryProgram( design, sets, vals
               , MAXVAL );
            t1
            if(isDisjunct(t1,defense))
                v1=t1;
                design;
                return;
            else
                design = reducer([design;t1])
            end
        end
    end
end
```

```
function [ret]=newcombnk(N,K)
%enumerate the combinations of N choose K.
%10x faster than the matlab builtin
cur=N(1:K);
mx=N(size(N,2));
```

```
pos=K;
ret=zeros(nchoosek(N(size(N,2)),K),K);
ret(1,:)=cur;
for i=2:size(ret,1)
    if(cur(pos)<(mx-(K-pos)))
        cur(pos)=cur(pos)+1;
    else
        while(cur(pos)==(mx-(K-pos)))
            pos=pos-1;
        end
        min=cur(pos);
        start=pos;
        for j=1:K-start+1
            cur(pos)=min+j;
            pos=pos+1;
        end
        pos=K;
    end
    ret(i,:)=cur;
end
```

```
function [ v1 ] = generateMWProbeConcious( design, sets,
   vals, MAXVAL, requirements, cost)
%GENERATEMWPROBECONCIOUS Summary of this function goes here
%   Detailed explanation goes here
    bits=size(sets,1);
    highVal=2^bits;
    options = sparse(highVal,size(sets,2));
    values = sparse(highVal,1);
    parfor i=1:2^bits-1
        selected=decodeBits(i,2,bits);
        options(i,:)=max(sets(selected>0,:),[],1);
        if(isDisjunct(options(i,:),design) && (sum(options(i
           ,:)>=requirements) >= sum(requirements)))
            for j=1:size(sets,1)
                if(sum(options(i,:)>=sets(j,:))==size(sets
                   ,2))
                    values(i)=values(i)+vals(j);
                end
            end
            values(i)=values(i)/cost(options(i,:));
        else
            values(i)=0;
            options(i,:)=0;
        end
    end

    [~,I]=max(values);
    v1=options(I,:);
end
```

```
function [ v1 ] = generateMWBinaryProgram( design, sets,
   vals, MAXVAL )
%Generate the optimal boolean vector given the design
   constraints
```

```
%(prohibited) and the preferences (sets) which have values (
   vals), where
%each value can have maximum value MAXVAL-1 (MAXVAL is
   really number of
%possible values it can take on.
    TwoPow = floor(log2(MAXVAL)+1);
    VarEnd=size(sets,2)*TwoPow;
    f=[zeros(1,VarEnd),-1*vals'];
    A=zeros(0,VarEnd+size(sets,1));

    %for each rule
    for i=1:size(sets,1)
        cur = sets(i,:);
        %for each element of each rule
        %Add a constraint for the value and every potential
           value greater
        %since they are represented in binary
        for j=1:size(cur,2)
                RowRep = decodeBits(cur(j),2,TwoPow);
                firstOne = find(RowRep,1,'first');
                if(isempty(firstOne))
                    %Nothing to add here, all zeros
                    continue;
                end
                RowRep(1:firstOne)=1;
                for k=1:size(RowRep,2)
                    NewRep(k)=-1*RowRep(k)*(2^(TwoPow-k));
```

```
            end
            RowRep = NewRep;
            MakingARow=[];
            if(j>1)
                MakingARow=[MakingARow,zeros(1,TwoPow*(j
                   -1))];
            end
            MakingARow=[MakingARow,RowRep];
            if(j<size(cur,2))
                MakingARow=[MakingARow,zeros(1,(size(cur
                   ,2)-j)*TwoPow)];
            end
            if(i>1)
                MakingARow=[MakingARow,zeros(1,i-1)];
            end
            if(max(abs(MakingARow))>0)
                MakingARow=[MakingARow,cur(j)];
            else
                continue
            end
            if(i<size(sets,1))
                MakingARow=[MakingARow,zeros(1,size(sets
                   ,1)-i)];
            end
            A=[A;MakingARow];
        end
    end
```

```
desStart=size(A,1);
%Start of design requirements
%For each design rule
%a better  way to do this
%shouldnt go through every number, just every bit
   greater than
%so like 3,4,8,16
powers=[];
for i=1:(TwoPow)
    powers(TwoPow+1-i)=2^(i-1);
end
for i=1:size(design,1)
    cur=design(i,:);
    tmp=cur;
    idxes = find(tmp);
    %start at the end and work our way backwards
    lastIdx = size(idxes,2);
    done=0;
    %add the first thing itself
    toAdd = [];
    for k=1:size(tmp,2)
        toAdd=[toAdd,decodeBits(tmp(k),2,TwoPow).*powers
           ];
    end
    A=[A;toAdd,zeros(1,size(A,2)-size(toAdd,2))];
    while(~done)
        next=0;
        idx=lastIdx;
        while(~next)
            if(tmp(idxes(idx))<MAXVAL)
                %increment the current index, if you cant, set it to
                %its value from cur, and continue to the next index
                %and so on for each index until you can increment
                %something, setting each previous to its minimum from cur
                if(mod(tmp(idxes(idx)),2)==0)
                    if(2*tmp(idxes(idx))<MAXVAL)
                        tmp(idxes(idx))=tmp(idxes(idx))*2;
                        break;
                    end
                end
                tmp(idxes(idx))=tmp(idxes(idx))+1;
                break;
            else
                if(idx==1)
                    done=1;
                    break;
                end
                tmp(idxes(idx))=cur(idxes(idx));
                idx=idx-1;
                continue;
            end
        end
        toAdd = [];
        for k=1:size(tmp,2)
            toAdd=[toAdd,decodeBits(tmp(k),2,TwoPow).*
               powers];
        end
        A=[A;toAdd,zeros(1,size(A,2)-size(toAdd,2))];
    end
end
b=[zeros(1,desStart),sum(A(desStart+1:size(A,1),:),2)
   '-1];
%set the maximum values
for i=1:size(sets,2)
    row=[];
    if(i>1)
        row=[zeros(1,size(sets,2)*i)];
    end
    row=[row,powers];
    if(i<size(sets,2))
        row=[row,zeros(1,size(sets,2)*(size(sets,2)-i))
           ];
    end
    row=[row,zeros(1,size(A,2)-size(row,2))];
    A=[A;row];
end
```

```
    b=[b,MAXVAL.*ones(1,size(sets,2))];
    [t,fval,exitflag,output]=bintprog(f,A,b)
    v1=zeros(1,size(design,2));
    %the variables in t are useless, instead use the set
       values to see which priorities are
    %active.
    fvalSubSelect = t(size(sets,2)*TwoPow+1:size(t,1));
    v1 = max(sets(fvalSubSelect>0,:),[],1);
end
```

```
function [ v1 ] = generateMW( design, sets, vals, MAXVAL )
%Generate the one step back most optimal malware based on
   the priorities
% and the Values
    included=ones(size(sets,1),1);
    v1=max(sets(included>0,:),[],1)
    for i=1:size(sets,1)
        if(isDisjunct(v1,design))
            included(i)=0;
        end
    end
end
```

```
function [ v1 ] = generateMW( design, sets, vals, MAXVAL )
%Generate the one step back most optimal malware based on
   the priorities
% and the Values
    included=ones(size(sets,1),1);
    v1=max(sets(included>0,:),[],1)
    for i=1:size(sets,1)
        if(isDisjunct(v1,design))
            included(i)=0;
        end
    end
end
```

```
%Design Restrictions:
%can’t be captured by
% 1 1 1 1
% 1 1 1 0

%values [2;3;4] corresponding with abilities
%abilities  1 1 0 1
%           1 0 1 1
%           1 1 0 0

A =[-1    -1     0    -1     3     0     0;
    -1     0    -1    -1     0     3     0;
    -1    -1     0     0     0     0     2;
     1     1     1     1     0     0     0;
     1     1     1     0     0     0     0;];
 f=[0     0     0     0    -2    -3    -4];
 b=[0     0     0     3     2];

[t,fval,exitflag,output]=bintprog(f,A,b);

```

```
%Design restrictions
%can't be captured by
%1 1 1
%3 2 0

%values [20;50;10;5] corresponding with
% 0 3 3
% 2 2 2
% 1 1 0
% 0 1 1

A=[0     0       -2      -1      0       0       3       0
                  0       0;
0       0       0       0       -2      -1      3       0
                  0       0;
-2      0       0       0       0       0       0       2
                  0       0;
0       0       -2      0       0       0       0       2
                  0       0;
0       0       0       0       -2      0       0       2
                  0       0;
0       -1      0       0       0       0       0       0
                  1       0;
0       0       0       -1      0       0       0       0
                  1       0;
0       0       0       -1      0       0       0       0
                  0       1;
0        0        0        0        0        -1       0        0
            0        1;
1        0        1        0        1        0        0        0
            0        0;
1        0        0        1        1        0        0        0
            0        0;
1        0        0        1        0        1        0        0
            0        0;
1        0        1        0        0        1        0        0
            0        0;
0        1        1        0        1        0        0        0
            0        0;
0        1        1        0        0        1        0        0
            0        0;
0        1        0        1        1        0        0        0
            0        0;
0        1        0        1        0        1        0        0
            0        0;
1        1        1        0        0        0        0        0
            0        0];

f=[0 0 0 0 0 0 -20 -50 -10 -5];
b=[0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2];
```

```
function [ P ] = convertCBDtoDNF( A )
%CONVERTCBDTODNF Converts a CBD matrix to DNF
%   Converts a CBD formatted probability transition matrix
   to Disjunctive
```

```
% Normal Form.
    A=ceil(A);%fix all probability values to connectivity
    P = paths(1,[],A,size(A,2));%find all the paths
       individually
    P=P(:,2:size(P,2)-1);%chop off the start state and the
       absorbing state
end
```

```
function bits = decodeBits(state,base,elements)
    bits = zeros(1,elements);
    if(state>(base^elements))
        bits= -1*ones(1,elements);
        return;
    end
    for j=elements-1:-1:0
        bits(elements-j)=floor(state/(base^j));
        state=mod(state,base^j);
    end
end
```

```
function [ strategyIters ] = defenseGen( userBehaviors,
   userValues, malwareBehaviors, malwareCosts, attackerTypes
   , typeCosts)
%DEFENSEGEN Summary of this function goes here
%   Detailed explanation goes here


end
```

```
function val = encodeBits(state,S,n)
%Returns the value representing the state passed in the
   space with base S
% and number of values n
    val=0;
    for j=n:-1:1
        val=val+state(n-j+1)*S^(j-1);
    end
end
```

```
function [ disjunct, rule ] = isDisjunct( A,B )
%returns 0 if the sets of DNF expressions are not disjunct,
   1 if they are
%also can return the rule which captured an expression from
   set A.
%Only will detect if B captures A, not if A captures B.

    %find any expressions in B which are supersets of A, (i.
       e. have no
    %entries other than the entries in A)
    I=find(sum(B(:,A==0),2)==0,1);
    %If there are none, A is disjunct from B
    disjunct=isempty(I);
    %return the Rule in B that detected A
    rule=I;

end
```

```
function [ subset, rule ] = isAnySubset( A,B )
%returns 0 if the sets of DNF expressions are not disjunct,
   1 if they are
%also can return the rule which captured an expression from
   set A.
%Only will detect if B captures A, not if A captures B.

%Basically this is isDisjunct for A and B both matrices



    %find any expressions in B which are supersets of A, (i.
       e. have no
    %entries other than the entries in A)
    I=find(sum(B(:,A==0),2)==0,1);
    %If there are none, A is disjunct from B
    disjunct=isempty(I);
    %return the Rule in B that detected A
    rule=I;

end
```

```
function [ disjunct, rule ] = isDisjunct( A,B )
%returns 0 if the sets of DNF expressions are not disjunct,
   1 if they are
%also can return the rule which captured an expression from
   set A.
%Only will detect if B captures A, not if A captures B.
%     disjunct=1;
```

```
    I=find(sum(B(:,A==0),2)==0,1);
    disjunct=isempty(I);
    rule=I;
%     for j=1:size(B,1)
%         for i=1:size(A,1)
%             if(length(find(A(i,:)>=B(j,:)))==size(A,2))
%                 disjunct=0;
%                 rule=[A(i,:);B(j,:)];
%                 return
%             end
%         end
%     end
end
```

```
function [ P ] = paths( next, built, A, target)
%PATHS Recursively find paths
%   P is a list of singular paths which connect next to the target of
%   A, assuming that the last column of A is an absorbing state.
    cols = size(A,2);
    if (isempty(built))
        built = zeros(1,cols);
        built(next)=1;
    end
    if (next == target)
        P=built;
        P(1,next)=1;
    else
        [~,opts,~] = find(A(next,:))
        P=zeros(0,cols);
        for i = 1:size(opts,2)
            if(built(opts(i))~=1 && next~=opts(i)) %no loops
                built(next)=1;
                P=[P;paths(opts(i),built,A,target)];
            end
        end
    end
end
```

```
function [ canReduce,v3 ] = reduce( v1,v2 )
%REDUCE When possible, reduce two logical expressions to one
%   reduce(v1,v2) simplifies two logical statements in
    Dinjunctive Normal
%   Form to a single statement when possible.  The
    statements must be
%   enetered as row vectors of 1’s 0’s and -1’s.  For
    example:
%   reduce([1 0 -1],[1 0 0]), would reduce to: [1 0 0], and
    is equivilent
%   to reducing (ac’) + (a) to (a).
%
%   It returns [canReduce, v3], the boolean of if the
    statements could be
%   reduced, and if they could, the new logical statement.
    canReduce=false;
```

```
v3=[];
C=abs(v1+v2);
zers = length(find(v1==0));
if(isempty(find(C==1)))
    if(length(find(C==0))==zers+1)
        [~,j,~]=find(C==0);
        v3=v1;
        v3(j)=0;
        canReduce=true;
    end
else
    %the places where one has a zero and the other has a
        value
    [~,j,~]=find(C==1);
    if(v1(j(1))==0)
        t1=v1;
    else
        t1=v2;
    end
    for i=2:length(j)
        if(t1(j(i))~=0)
            return;
        end
    end
    v3=t1;
    canReduce=true;
end
```

```

end

function A = reducer(A)
    while(true)
        [x,y]=size(A);
        B=zeros(0,y);
        cant=ones(1,x);
        for i=1:x
            for j=i+1:x
                if(i~=j)
                    [can,new] = nStateReduce(A(i,:),A(j,:));
                    if(can)
                        cant(i)=0;
                        cant(j)=0;
                        B=[B;new];
                    end
                end
            end
            if(cant(i))
                B=[B;A(i,:)];
            end
        end
        if(sum(cant)==length(cant))
            break;
        end
        A=unique(B,'rows');
    end
```

```
end
```

```
function [ firstsArr,sizesArr ] = surfVals( samples,Range,N,
   K)
    %Runs advancedAttackSimulator with the appropriate
       arguments and
    %generates two arrays to be used with surf() to make a 3
       D plot
    firstsArr=zeros(size(Range,2));
    sizesArr=zeros(size(Range,2));
    for i=1:size(Range,2)
        for j=1:size(Range,2)

            [firsts,~,sizes,~]=advancedAttackSimulator(
               samples,Range(i),N,K,Range(j),[]);
            firstsArr(i,j)=mean(firsts);
            sizesArr(i,j)=mean(sizes);
            [i,j]
        end
    end
end
```

```
function [firsts,alls,sizes,eff]=advancedAttackSimulator(
   trials,statements,sensors,K,Yestimate,doAlls)
    %logic=rand(sensors,statements,K);
    if(statements>nchoosek(sensors,K))
        fprintf('Error:␣defense␣statements␣>␣nchoosek(
           sensors,K).');
        firsts=-1;
        alls=-1;
        sizes=-1;
        return
    elseif(statements==nchoosek(sensors,K))
        fprintf('Error:␣Complete␣space␣coverage␣by␣defense.'
          );
        firsts=-1;
        alls=-1;
        sizes=-1;
        return
    end
    firsts=zeros(1,trials);
    alls=zeros(1,trials);
    sizes=zeros(1,trials);
    eff=zeros(1,trials);
%     combs=combnk(1:sensors,K);
    for j=1:trials
        internalY=Yestimate;
        defense=zeros(1,sensors);
        defense(randperm(sensors,K))=1;
        while(size(defense,1)<statements)
            candidate=zeros(1,sensors);
            candidate(randperm(sensors,K))=1;
            while(~isDisjunct(candidate,defense))
                candidate=zeros(1,sensors);
                candidate(randperm(sensors,K))=1;
            end
            defense=[defense;candidate];
        end
%          logics=combs(randperm(size(combs,1),statements),:)
   ;
%          for i=1:statements
%              defense(i,logics(i,:))=1;
%          end
        v1=ones(1,sensors);
        design=[];
        antiDesign=[];
        capturedProbes=zeros(1,sensors);
        firstVulnCount=0;
        while(~isDisjunct(v1,defense))
            firstVulnCount=firstVulnCount+1;
            design=[design;v1];
            capturedProbes(size(v1,2))=capturedProbes(size(
               v1,2))+1;
            [v1,internalY]=generateMWIterativeSim(design,
               sensors,K,internalY,capturedProbes);
        end
        antiDesign=[antiDesign;v1];
        detections=0;
        TestsReq=1000;
        N=sensors;
        %not sure how to Optimize this based on the A's
        A=K;
        if(doAlls)
            for i=1:TestsReq
                %generate an A length permutation of the
                   numbers 1:N
                row = zeros(1,N);
                row(randperm(N,A))=1;
                if(~isDisjunct(row,defense))
                    detections=detections+1;
                    design=[design;row];
                else
                    antiDesign=[antiDesign;row];
                end
            end
            design=reducer2State(design,K);
            %calculate our mu for the binomial
            %wald method
            p=1-(detections)/(TestsReq);
            pmax=min(p+2*sqrt(.25/TestsReq),1);
            pmin=max(p-2*sqrt(.25/TestsReq),0);
            testP=1;
            yMin=0;

            numer=nchoosek(A,K);
            denomBase=nchoosek(N,K);
            yMax=0;
            yMid=0;
            while(pmin<testP)
                denom=max(denomBase-yMid,numer);
                testP=testP*(1-numer/denom);
                %deceptively named, this is the Y associated
                    with the min P
                yMin=yMin+1;
                if(yMax==0 && pmax>testP)
                    yMax=yMin-1;
                elseif(yMid==0 && p>testP)
                    yMid=yMin-1;
                end
            end
            %make sure we have captured every part of the
               design with a 3DNF
            while(max(sum(design,2))>K && i<nchoosek(N,K))
                I=find(sum(design,2)>K,1);
                %replaces all the >3s with 3s

                J=find(design(I,:));
                idxes = randperm(sum(design(I,:),2),3);
                row=zeros(1,N);
                row(J(idxes))=1;
                while(isDisjunct(row,defense))
                    antiDesign=[antiDesign;row];
                    i=i+1;
                    flag=0;
                    while(~isDisjunct(row,[design;antiDesign]) && size([design;antiDesign],1)<nchoosek(N,K))
                        idxes = randperm(sum(design(I,:),2),3);
                        row=zeros(1,N);
                        row(J(idxes))=1;
                        flag=flag+1;
                        if(flag>500)
                            breakpoint
                        end
                    end
                end
                design(I,:)=0;
                %see if this row captures any other rows
    %             [T,toRemove]=isDisjunct(row,design);
    %             while(~T)
    %                 design(toRemove,:)=-1;
    %                 [T,toRemove]=isDisjunct(row,design);
    %             end
                toRemove=find(sum(design(:,row~=0),2)>=K);

                design(I,:)=row;
                design(toRemove,:)=[];
    %             added=added+1;
                %remove all empty rows
                i=i+1;
                %row=zeros(1,N);
                %row(randperm(N,K))=1;
                %generate new rows until we generate a unique one, shouldn’t be
                %many loops ever unless the number of defense statements
                %approaches the number of potential statements, which it never
                %should because of false positive concerns
                %while(ismember(row,design,’rows’))
                %    row=zeros(1,N);
                %    row(randperm(N,K))=1;
                %end
                %if(~isDisjunct(row,defense))
                %if(ismember(row,defense,’rows’))
                %    design=reducer([design;row]);
                %    added=added+1;
                %end
                %i=i+1;
            end
            added=size(design,1);
            %add up to the small side of the confidence interval
            while(added<yMax && i<nchoosek(N,K))
                row=zeros(1,N);
                row(randperm(N,K))=1;
```

```
                %generate new rows until we generate a unique one, shouldn’t be
                %many loops ever
                while(~isDisjunct(row,[design;antiDesign]))
                    row=zeros(1,N);
                    row(randperm(N,K))=1;
                end
                %if(~isDisjunct(row,defense))
                %if we know they are KDNF
                if(~isDisjunct(row,defense))
                    design=[design;row];
                    added=added+1;
                else
                    antiDesign=[antiDesign;row];
                end
                i=i+1;
            end
            %design=reducer(design);
    %         size(design,1)
    %         size(defense,1)
            alls(j)=i;
        else
            alls(j)=nchoosek(sensors,K);
        end
        firsts(j)=firstVulnCount;
        sizes(j)=sum(v1);
        eff(j)=alls(j)/nchoosek(N,K);
        end
end
```

Bibliography

[1] O. E. Dictionary, "cyber-attack." [Online]. Available: http://www.oed.com/view/Entry/250879?redirectedFrom=Cyber+attack#eid212385738

[2] A. Ivanov, "Side-channel attacks," 2005.

[3] A. Hevia and M. Kiwi, "Strength of two data encryption standard implementations under timing attacks," in *LATIN'98: Theoretical Informatics*. Springer, 1998, pp. 192–205.

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

[5] I. Blog, "It security cyber kill chain."

[6] L. Myers, "The practicality of the cyber kill chain approach to security," October 2013. [Online]. Available: http://www.csoonline.com/article/2134037/strategic-planning-erm/the-practicality-of-the-cyber-kill-chain-approach-to-security.html

[7] "Hp attack life cycle use case methodology," March 2014. [Online]. Available: http://h20195.www2.hp.com/V2/GetPDF.aspx%2F4AA4-9490ENW.pdf

[8] B. News, "Clock ticking on worm attack code," January 2009. [Online]. Available: http://news.bbc.co.uk/2/hi/technology/7832652.stm

[9] R. Shirey, "Internet security glossary, version 2," Internet Engineering Task Force, RFC 4949, August 2007. [Online]. Available: https://tools.ietf.org/html/rfc4949

[10] *The Real Story of Stuxnet*, 2013.

[11] "Stuxnet marked a turning point for the entire automation industry." [Online]. Available: https://s3.amazonaws.com/s3.documentcloud.org/documents/356902/siemens-statement.pdf

[12] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *Security & Privacy, IEEE*, vol. 9, no. 3, pp. 49–51, 2011.

[13] B. Elgin, "Neiman marcus hackers set off 60,000 alerts while bagging credit card data."

[14] "Snort ids." [Online]. Available: http://www.snort.org/

[15] HP, "Arcsight security information and event management." [Online]. Available: http://www8.hp.com/us/en/software-solutions/siem-arcsight/

[16] ——, "Arcsight threatdetector." [Online]. Available: http://www8.hp.com/us/en/software-solutions/software.html?compURI=1418958

[17] C. News, "Norad downplays russian bomber interception." [Online]. Available: http://www.cbc.ca/news/politics/norad-downplays-russian-bomber-interception-1.929222

[18] "Bandits," Film, October 2001. [Online]. Available: http://www.imdb.com/title/tt0219965/

[19] "Implicant." [Online]. Available: https://en.wikipedia.org/wiki/Implicant

[20] "Dedekind number," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Dedekind_number

[21] R. B. Myerson, *Game theory*. Harvard university press, 2013.

[22] finchy, "What-is-a-snort-rule.md," GitHub. [Online]. Available: https://github.com/vrtadmin/snort-faq/blob/master/Rules/What-is-a-Snort-rule.md

[23] G. Vigna, W. Robertson, and D. Balzarotti, "Testing network-based intrusion detection signatures using mutant exploits," in *Proceedings of the 11th ACM conference on Computer and communications security*. ACM, 2004, pp. 21–30.

[24] "Snort: Rule snort ping flood," 2013. [Online]. Available: http://seclists.org/snort/2013/q1/897

[25] J. McMillan, "Using snort to detect clear text credit card numbers," November 2009. [Online]. Available: https://www.sans.org/security-resources/idfaq/snort-detect-credit-card-numbers.php

[26] B. Walther, "Arcsight logger - searching," March 2012. [Online]. Available: https://wikis.uit.tufts.edu/confluence/display/exchange2010/ArcSight+Logger+-+Searching

[27] E. M. Gustavson, "Arcsight logger - reports and queries," May 2011. [Online]. Available: https://wikis.uit.tufts.edu/confluence/pages/viewpage.action?pageId=44800827

[28] HP, "Early notice of potential attacks." [Online]. Available: http://www.www8-hp.com/us/en/images/ArchSight1_tcm245_1622582_tcm245_1622575_tcm245-1622582.jpg

[29] B. Schneier, "Staged attack causes generator to self-destruct," October 2007. [Online]. Available: https://www.schneier.com/blog/archives/2007/10/staged_attack_c.html

[30] G. Ericsson, "Cyber security and power system communication;essential parts of a smart grid infrastructure," *Power Delivery, IEEE Transactions on*, vol. 25, no. 3, pp. 1501–1507, July 2010.

[31] D. Angluin, "Queries and concept learning," *Machine learning*, vol. 2, no. 4, pp. 319–342, 1988.

[32] J. Jackson, "An efficient membership-query algorithm for learning dnf with respect to the uniform distribution," in *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*. IEEE, 1994, pp. 42–53.

[33] F. Bergadano, D. Catalano, and S. Varricchio, "Learning sat-k-dnf formulas from membership queries," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM, 1996, pp. 126–130.

[34] J. C. Jackson, H. K. Lee, R. A. Servedio, and A. Wan, "Learning random monotone dnf," in *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2008, pp. 483–497.

[35] T. Ferguson, "Linear programming: A concise introduction," *UCLA [online] http://www. math. ucla. edu/˜ tom/LP. pdf*, 1998.

[36] G. F. Stocco and G. Cybenko, "Exploiting adversary's risk profiles in imperfect information security games," in *Decision and Game Theory for Security*. Springer, 2011, pp. 22–33.

[37] D. Billings, "Algorithms and assessment in computer poker," Ph.D. dissertation, University of Alberta, 2006.

[38] D. Billings, D. Papp, J. Schaeffer, and D. Szafron, "Opponent modeling in poker," in *AAAI/IAAI*, 1998, pp. 493–499.

[39] D. R. Papp, *Dealing with imperfect information in poker.* University of Alberta, 1999.

[40] W. Jiang, B.-x. Fang, H.-l. Zhang, Z.-h. Tian, and X.-f. Song, "Optimal network security strengthening using attack-defense game model," in *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*. IEEE, 2009, pp. 475–480.

[41] W. Jiang, H.-L. Zhang, Z.-H. Tian, and X.-f. Song, "A game theoretic method for decision and analysis of the optimal active defense strategy," in *Computational Intelligence and Security, 2007 International Conference on*. IEEE, 2007, pp. 819–823.

[42] K.-w. Lye and J. M. Wing, "Game strategies in network security," *International Journal of Information Security*, vol. 4, no. 1-2, pp. 71–86, 2005.

[43] C. Ferguson and T. S. Ferguson, "On the borel and von neumann poker models," *Game theory and applications*, vol. 9, pp. 17–32, 2003.

[44] J. Von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior (60th Anniversary Commemorative Edition)*. Princeton university press, 2007.

[45] W. R. Smith and D. Lai, "United states vs. north korea in no-limit poker: Alligator blood or dead money?" *The Korean Journal of Defense Analysis*, vol. 17, no. 2, pp. 111–126, 2005.

[46] S. Freeman, "The highest stakes poker game ever played: Ronald reagan, mikhail gorbachev, and the reykjavik summit of 1986," Ph.D. dissertation, Vanderbilt University. Dept. of History, 2010.

[47] (2011, May). [Online]. Available: http://www.spiegel.de/international/world/us-pakistani-relations-a-forced-marriage-plagued-by-ever-deepening-distrust-a-761190.html

[48] (2011, 6). [Online]. Available: http://tribune.com.pk/story/192590/haqqani-defends-decision-to-detain-cia-informants/

[49] [Online]. Available: http://www.mtholyoke.edu/acad/intrel/u2.htm

[50] [Online]. Available: http://techland.time.com/2011/05/16/uk-government-under-constant-cyber-attack/

[51] [Online]. Available: http://www.cbsnews.com/stories/2011/06/01/national/main20067895.shtml

[52] D. Kahneman and A. Tversky, "Prospect theory: An analysis of decision under risk," *Econometrica: Journal of the Econometric Society*, pp. 263–291, 1979.

[53] A. Tversky and D. Kahneman, "Advances in prospect theory: Cumulative representation of uncertainty," *Journal of Risk and uncertainty*, vol. 5, no. 4, pp. 297–323, 1992.

[54] "Value (poker)." [Online]. Available: https://en.wikipedia.org/wiki/Value_(poker)

[55] F. T. River. [Online]. Available: http://poker-strategy.flopturnriver.com/5-Biggest-Leaks-Of-Losing-NL-Player-3.php

[56] L. Wang, S. Noel, and S. Jajodia, "Minimum-cost network hardening using attack graphs," *Computer Communications*, vol. 29, no. 18, pp. 3812–3824, 2006.

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

[58] L. Carin, G. Cybenko, and J. Hughes, "Cybersecurity strategies: The queries methodology," *Computer*, vol. 41, no. 8, pp. 20–26, 2008.

[59] E. Israeli and R. K. Wood, "Shortest-path network interdiction," *Networks*, vol. 40, no. 2, pp. 97–111, 2002. [Online]. Available: http://dx.doi.org/10.1002/net.10039

[60] D. R. Fulkerson and G. C. Harding, "Maximizing the minimum source-sink path subject to a budget constraint," *Mathematical Programming*, vol. 13, pp. 116–118, 1977, 10.1007/BF01584329. [Online]. Available: http://dx.doi.org/10.1007/BF01584329

[61] B. Golden, "A problem in network interdiction," *Naval Research Logistics Quarterly*, vol. 25, no. 4, pp. 711–713, 1978.

[62] T. Loring, "A proof of menger's theorem," 2005. [Online]. Available: http://www.math.unm.edu/~loring/links/graph_s05/Menger.pdf

[63] D. West, *Introduction to Graph Theory*. Prentice Hall (Upper Saddle River, NJ), 1996.

[64] U. Army, "Joint technical coordinating group for munitions effectiveness program office." [Online]. Available: http://web.amsaa.army.mil/JTCGMEPO.html

[65] G. Cybenko and J. Syverson, "Quantitative foundations for information operations," 2007.

[66] P. Erdos, A. W. Goodman, and L. Pósa, "The representation of a graph by set intersections," *Canad. J. Math*, vol. 18, pp. 106–112, 1966.

[67] P. Turán, "On an extremal problem in graph theory," *Mat. Fiz. Lapok*, vol. 48, no. 436-452, p. 137, 1941.

[68] D. Mitchell, B. Selman, and H. Levesque, "Hard and easy distributions of sat problems," in *AAAI*, vol. 92. Citeseer, 1992, pp. 459–465.