Framework For Modeling Attacker Capabilities with Deception

2019

Sharif Hassan
University of Central Florida

A FRAMEWORK FOR MODELING ATTACKER CAPABILITIES WITH DECEPTION

by

SHARIF HASSAN
B.S. University of Central Florida, 2000
M.S. Florida Institute of Technology, 2008

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term

2019

Major Professor: Ratan Kumar Guha

ABSTRACT

In this research we built a custom experimental range using opensource emulated and custom pure honeypots designed to detect or capture attacker activity. The focus is to test the effectiveness of a deception in its ability to evade detection coupled with attacker skill levels. The range consists of three zones accessible via virtual private networking. The first zone houses varying configurations of opensource emulated honeypots, custom built pure honeypots, and real SSH servers. The second zone acts as a point of presence for attackers. The third zone is for administration and monitoring. Using the range, both a control and participant-based experiment were conducted.

We conducted control experiments to baseline and empirically explore honeypot detectability amongst other systems through adversarial testing. We executed a series of tests such as network service sweep, enumeration scanning, and finally manual execution. We also selected participants to serve as cyber attackers against the experiment range of varying skills having unique tactics, techniques and procedures in attempting to detect the honeypots.

We have concluded the experiments and performed data analysis. We measure the anticipated threat by presenting the Attacker Bias Perception Profile model. Using this model, each participant is ranked based on their overall threat classification and impact. This model is applied to the results of the participants which helps align the threat to likelihood and impact of a honeypot being detected. The results indicate the pure honeypots are significantly difficult to detect. Emulated honeypots are grouped in different categories based on the detection and skills of the attackers. We developed a framework abstracting the deceptive process, the interaction

with system elements, the use of intelligence, and the relationship with attackers. The framework is illustrated by our experiment case studies and the attacker actions, the effects on the system, and impact to the success.

I dedicate this dissertation, my research, and all the efforts involved first to my loving wife Kimmy who without her I would not have been able to complete this work. I would also like to dedicate this to my daughters Kinsley and Layla for it was you both who gave me the motivation. Finally, I dedicate this work to Clover for always being by my side.

ACKNOWLEDGMENTS

I would first like to express my gratitude to my advisor Dr Ratan Guha for everything you have done. I want to thank my committee for taking their time to support my research. I would like to acknowledge the support and help by each participant in my research experimentation. I want to thank everyone who supported me at Lockheed Martin. I would like to thank my family and friends for their support and dedication. I want to especially thank my wife Kimmy for everything she has done to help and support me. Finally, and certainly not least, I want to thank God for giving me the ability to complete this work.

Committee Members:

Dr. Ratan K Guha

Dr. Mostafa Bassiouni

Dr. Mainak Chatterjee

Dr. Ronald DeMara

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

APT – Advanced and Persistent Threat

CND – Computer Network Defense

DC3- Department of Defense Cyber Crime Center

DDoS – Distributed Denial of Service

DIB – Defense Industrial Base

DNS – Domain Name Service

DoD – Department of Defense

DSIE - DoD Collective Sharing Environment

FTP – File Transport Protocol

GUI Graphical User Interface

IDD – Intelligence Driven Defense

IoC – Indicator of Compromise

IoT – Internet of Things

IP – Internet Protocol

IRB – Institute Review Board

NPS – Naval Postgraduate School

SCP – Secure Copy

SFTP Secure File Transport Protocol

SSH – Secure Shell

OSINT – Open Source Intelligence

TCP – Transport Control Protocol

TTP – Tactics, Techniques, and Procedures

VPN – Virtual Private Network

CHAPTER ONE: INTRODUCTION

All war is based on deception, a quote by Sun Tzu in the Art of War [1]. This quote tells a truth about how humankind has leveraged the art of deception as a tool for quite some time. War being just one example, deception can have many drivers such as hunting, where the prey is deceived to enter a trap. Merriam-Westers dictionary defines deception as the act of causing someone to accept as true or valid what is false or invalid. It further states the fact or condition of being deceived, or something that deceives [64]. Both the offensive and defensive sides use deception as either ways to attack or protect. Cyber is no different. Cyber attackers leverage deception to hoax users into opening malicious emails [3]. Defenders use deception to lure the attackers away or to attack and study their actions [4].

Cyber threat is typically followed by a defensive response. In some situations, the intelligence is already available, and the threat can be detected. Other times the threat is unknown and naturally become a reactive response. Threat can be characterized in diverse ways but is typically referred to as Advanced and Persistent threat (APT) [5]. In many ways the advanced portion may not be as sophisticated as first imagined. Discovering a simple zero-day vulnerability and developing an associated exploit is always the unknown threat that is feared. It gives the attacker the upper hand until eventually disclosed by means of correlating intelligence based on the attacker activities. From there patches are created, announcements are made, and the cycles completes. Being able to possibly trick the attacker to perform these unknown attacks on a system setup as a deception versus a true production system with critical data would be the ideal scenario.

Being able to deceive an attacker to thwart or study their tactics, techniques, and procedures (TTP) is a major benefit to a Cyber defender, especially if the attacker is unaware. As different deception approaches, such as honeypots, have been tried and tested, some succeed but many falls short or are not designed for the intended attacker [6]. Just recently in the 2017 Symantec Internet Security Threat Report, where Symantec created and leveraged honeypots extensively for Internet of Things (IoT) technology [94]. Honeypots can vary in complexity from basic network socket listeners to a fully interactive system or application. Many are opensource, but some are custom coded that leverage real network services where they are modified, or substantial log monitoring is added. A Cyber attacker detecting a system as a honeypot usually causes them to ignore or cease their TTPs. The quicker they detect, the less effective for capturing useful attacker activity.

A significant amount of research has been conducted into constructing and deploying honeypots as well in the analysis of acquired attacker TTPs but very little on experimental testing of honeypots to better understand their effectiveness. Literature shows that some attempts have been made to experiment and test for the effectiveness of deception systems. One such study focused on skilled participants attacking a defined environment with a pre-determined successful attack vector route and where deception was introduced in random or planned situations. Participants were measured on how they handled the deception and its impact on their success and resulting attack vector map [82]. Another experiment focused on testing the effectiveness of honeypots and their deceptive levels in real networks by varying their location and virtualization [81]. A third research effort developed software tools and specific metrics to measure the effectiveness of honeypots. They did this by comparing specific areas of honeypot

filesystems against real filesystems identifying the variances [2]. Where these experiments homed in on the effectiveness of using deception, they lacked in the understanding how diverse honeypot designs and deployments are effective regarding detectability, especially with different grades of attacker skills. Therefore, we attempt to close the gap further and our contribution is threefold. First, we further contribute by adding additional experimentation into the success and effectiveness of deception software. Secondly, we explore and compare the use of emulated opensource deceptive software, namely honeypots, along with the use of real "pure" systems configured with monitoring as deceptive honeypots to understand their effectiveness and what characteristics of honeypots are more susceptible to detection. Lastly, we corelate and observe the aptitude for diverse attacker levels to detect the presence of deceptive software.

This dissertation is strongly experiential, specifically on testing honeypot systems in a custom developed experiment range. In our study, the focus is Secure Shell (SSH) based honeypots on Ubuntu Linux 14.04.5. Shell based meaning command line used to login, interact, manage, and administrative a system. Our primary experiment is a participate based controlled study that captures the ability for each to detect and possibly identify, out of a series of services, which are honeypots or not and justify why. Some honeypot access will be leaked using provided open source intelligence (OSINT) and while other the default configuration or just the login prompt. Participants will mark on a score sheet if they believe the services are not, suspicious, or indeed a honeypot. Each participant will be asked to complete a survey to assess their skills and align them to a set of defined attacker profiles and bias model. The intention is to determine detectability primarily in emulated versus pure honeypot design effectiveness and to

correlate the attacker skills in the participant survey to the detection rate based on the scorecard results.

In addition to the primary participant-based study, a control experiment is conducted against the same experiment range to create a baseline model for various degrees of testing. These tests range from general basic network sweep, conducting active reconnaissance scanning, to performing manual testing. General basic network sweep attempts to discover any known honeypots. Active reconnaissance scanning leverages tools that are specifically configured to probe for anomalies. Third is manual hands on keyboard penetration testing leveraging more advanced capabilities such as intercepting and modifying network traffic, living off the land using binaries inside honeypots to see if they function appropriately and so forth.

Finally, we then use the results of these experiments to propose a deception framework that outlines deception to attacker relationship and what characteristics of honeypots are more susceptible to detection. We articulate the attackers and use work completed on a bias profile perception model that attempts to loosely categorize them into profiles representing the spectrum of skills, capabilities, impact, and threat. We use this model to map the success of identifying honeypots in various experiments to varying levels of attacker skills. We also explore the deceptive process in a deception system state framework that outline the deceptive process and attacker influence.

CHAPTER TWO: BACKGROUND

To effectively study defensive cyber deception, there first needs to be a basis of understanding. Subsequent chapters will leverage this background material as a foundation to the presented concepts, approach, and results. The Background information starts with a review of different secure shell server software packages. Next attacker types are discussed grouped in to specific profiles. We then cover the basics on Cyber intelligence and indicator research. Finally, a look into deception itself followed by the deceptive process reviewing three existing deception models.

Secure Shell Technologies

SSH, developed in 1995 by Tatu Ylönen from the Helsini University of Technology in Finland, is a widely used client-server service which allows for remote encrypted connectivity to a system via a command line interface [36]. SSH is typically leveraged to login into remote machines and execute commands but also has the capabilities to create network tunnels, conduct port forwarding, and support the transferring of data [34]. The extension of the protocol suite includes such utilities as secure file transport protocol (SFTP) and secure copy protocol (SCP) allowing for the transfer of data such as binaries. SSH is a successor to legacy file transport protocol (FTP) and Telnet which allowed for similar functionality minus the link encryption. There are a few versions of the SSH protocol. The original version by Ylönen was dubbed SSH-1 and later found to be vulnerable in 1998 and again in 2001 with various encryption related issues. In 2006, SSH-2 was released to help address many of the known issues with SSH-1.

This dissertation focused on deceptive based SSH honeypots to determine their effectiveness and detectability. As a part of this research and experimentation, the use and inclusion of "real" SSH services are leveraged as a way of comparison and confusion. Each one of those software packages are described in further detail below.

OpenSSH

OpenSSH is one of the most widely used SSH software packages for Linux today. Many of the Linux operating systems come pre-packaged with OpenSSH as a default method to connect and manage the system remotely. OpenSSH is a derivative of the original definitive version of SSH released by Ylönen as a free product. As licensing restrictions became a continued issue with further development of free SSH software, Björn Grönvall, at the time a professor at Stockholm Sweeden, started to re-look at the original free SSH and fix several "bugs", calling it OSSH or Open SSH. At this time a popular of Linux variant, OpenBSD 2.6, was set for release and had hopes to include the OSSH server by Grönvall. During this time, the OpenSSH project was maturing and eventually became the focus to eventually replace OSSH. Three months later OpenSSH version 1.2.12 became the main SSH server in the OpenBSD build [87]. Today at version 7.7, OpenSSH is available under the GNU Public License and can be installed on a wide range of operating systems.

Dropbear

A homegrown SSH server named Dropbear started its development in October 2002 with its initial release in April 2003. Self-claiming to be installed in millions of computers, Dropbear is a small footprint SSH server and client that can be executed on any POSIX-based platforms

such as Linux, Mac OS X, Free BSD, or Cygwin. Like the other SSH packages, it also supports SSH-2 and the use of SFTP and SCP as additional services within the protocol suite [89]. Dropbear is the core SSH components of the OpenWRT project focused on an opensource wireless router firmware [90]. Dropbear is currently at version 2018.76 and is distributed under a MIT-style license [89].

Defining Attackers Profiles

The attacker perhaps is the most variable aspect of the cyber defense equation. The degree of sophistication in which an attacker can carry out a specific TTP is a direct correlation to the threat they can deliver. Some attackers discover new or unknown vulnerabilities, vectors of attack in which they exploit their target or sell for a profit [20]. Other attackers merely use tools, scripts, and pre-canned exploits, which have been already created by higher skilled individuals [21]. It is this range that begs the questions on what sort of attacker may be up against a deployed deception.

Attackers are not created equal. Every attacker has its own set of skills, knowledge and understanding of technology and its way of being manipulated. There are several ways to describe how attackers, hackers, differ and what makes up their skill level and motivation. Especially in the industry and news related articles, several security professionals have attempted to classify hackers into groups such as crime, spamming, spies, hacktivist, and APT [38]. There has been research in even differentiating the name or title hacker. Studying the difference between a hacker and intruder and how the former looks for bugs or holes and the later takes advantage of those holes to gain unauthorized access [39]. Others have studied ways to detect

between a novice and advanced hacker based on validating conceptual expertise using a developed tool that can differentiate attackers based on their actions [42]. There is not much on the academia front that significantly hones in on defined profiles for types of attackers based on their bias, motivation skills, and more. Because of this, we created groups for this research to simplify the types of attackers decided by their bias meaning their purpose, the set of capabilities they possess, and partially professional intuition. The idea of an attacker bias is based on a concept found in a dissertation by M. H. Almeshekah [17]. Having a defined set of attacker profiles can help the industry and academic researcher alike benefit from a greater understanding of attacker differences.

Some attackers are simply driven by the challenge of success in which they can "hack" a system [40]. Others have a more direct reason and invest a significant amount of funds and resources to succeed [41]. Outlined below are the attacker profiles considered in this research. They are defined by their motivation, objectives, typical TTPs, and methods [40]. The profiles will be used to help represent the attacker spectrum and how they compare with one another by their threat impact and likelihood of success. As it will soon be made clear, the profiles are listed in order of least to greatest threat. There is no rigid placement for where a potential attacker or hacking group may best fit. It is based on following information gleamed from intelligence reports, their political/military backing, source of funding, and motivation.

Annoyance Threats

Representing the lower bound, individuals or script kiddies have been on the steady rise as exploits to circumvent technology continues to be automated, shared and published. Attackers

in this category are generally unskilled and utilize existing tools or scripts created by other more advanced attackers or industry security professionals. Attacks are broad based and typically unknown or irrelevant and considered "noise" but occasionally can be problematic. TTPs include the use of known worms/viruses, exploit kits, automated scanners, crawlers, SPAM, basic malware, spyware and adware [38 ,40]. A summary of the profile is:

- Scope: broad-based
- Duration: short-lived
- TTPs: leverage others tools and development, worms/viruses, exploit kits, automated scanners, crawlers, SPAM, basic malware, spyware and adware
- Motivation: thrill, bragging rights, learning

Cyber Crime

Cyber Crime is opportunistic short-lived broad-based attacks motivated primarily by financial gain identity theft, credit card fraud, extortion, and botnet recruitment. Typical TTPs include extortion malware techniques, phishing or SPAM emailing, and hosting malware on legitimate websites [38, 40, 71]. The recent "Wannacry" worm attack is an example of how encryption was used to block access to victim data. A ransom to the attackers using bitcoin was the only way to acquire the decryption key [43, 44]. WannaCry infected computers in over 150 countries using a leaked NSA Windows SMB vulnerability zero-day exploit released by the hacking group Shadowbrokers. It is believed that the North Korean nation sponsored hacking group Lazarus created the worm. Though their primary focus may be nation sponsored espionage, this purpose was to extract capitol and cause havoc. The damage by the worm was

limited due to a security researcher who quickly registered a domain name he found in the malware code that disrupted communications. Over two hundred thousand systems were impacted globally, but less than one hundred thousand dollars were collected by the attackers [61]. A summary of the profile is:

- Scope: broad-based
- Duration: short-lived
- TTPs: similar as annoyance profile plus extortion malware techniques, phishing or SPAM emailing, and hosting malware on legitimate websites.
- Motivation: money

Cyber Terrorism

Cyber terrorism, synonymously to how they act in physical warfare, conduct disruptive or destructive acts on behalf of or in support of a terrorist group or organization. Exploitation lasting from mid to long term, their approach is typically targeted with some degree of sophistication in support of their ideology. Typical TTPs include DDoS, known and custom exploitation kits, modified malware, and creative delivery approaches. In a paper by the United State Army War College Press, they explain how terrorists are becoming more cyber engaged for anonymous communications, to cause substantial amounts of damage with little cost versus traditional ground confrontations and invites a connection to a worldwide audience for propaganda purposes [45]. In December 2015, cyber terrorists perpetrated a successful attack against three energy distribution companies in the Ukraine and disrupted the electricity supply of over 230,000 people. This attack occurred during the Russian-Ukrainian war and has been

attributed to a Russian group known as Sandworm. The campaign required careful planning and reconnaissance and lead to months of work to return the control centers to a fully operational state [60]. A summary of the profile is:

- Scope: semi-targeted or targeted
- Duration: mid-long term
- TTPs: distributed denial of service (DDoS), known and custom exploitation using kits, modified malware, and creative delivery approaches.
- Motivation: money

Hacktivism

Hacktivism, stated simply, stands for "politically motivated hacking" [59]. Many skilled individuals join an organized group and put their hacking abilities to serve a cause with a political, ethical, religious, or retaliatory motive. Typically, short term, these attacks cause a significant amount of havoc and are in many times disruptive to operations by disclosing sensitive information for public viewing. Their techniques include DDoS, traditional hacking techniques, spear phishing and more [48, 71]. The group Anonymous is a known example of a hacktivist hacking group that is seen today performing such style attacks [47]. The ideology of this group is fluid and diversified. The group supports digital piracy and has targeted entertainment entities such as the Recording Industry Association of America and the Motion Picture Association of America with attacks in response to their support of copyright enforcement [58]. They have been associated with everything from charity activities, such as Operation Oklahoma for flash flood victims, to political activism, such as Operation

CyberPrivacy protesting Canadian anti-terror legislation, and many more [58]. A summary of the profile is:

- Scope: semi-targeted or targeted
- Duration: short-lived
- TTPs: DDoS, traditional hacking techniques, custom exploitation, crafty spear phishing and more
- Motivation: political, ethical, religious, or retaliatory

Insider Threat

A legitimate internal individual on the "trusted" side that has alternative motives with their access is a major concern to many organizations and governments. Insider attackers are typically planted, cohered, or disgruntle and have access to trusted systems or critical data not normally available to outsiders. Insiders abuse their approved credentials, privileges, and access to acquire data for exfiltration or to plant and install malware. Some have the necessary technical skills, but most are carrying out instructions from another advanced entity outside of the boundary (many times nation sponsored). The length of any given intrusion can either be quick or long term if successful and undetected. Insider attacks can also include physical sabotage in systems, security, or surveillance. A known insider threat example is the executive administrative assistance at Coca cola who used her access to steel and attempt to sell the formula for Coke to Pepsi [49]. Another example involves a Pennsylvania utility worker who was sentenced June 2017 for hacking into his former employer computer systems and disabling the remote water meter readers which the company manufactured. This attack disabled remote

metering and forced the dispatch of numerous technicians to physically read meters until the system could be restored [56]. A third example is a software engineer, for an unnamed financial services firm in New York, who was arrested in April for steeling proprietary code from his employer. Allegedly, the engineer used multiple methods to exfiltrate the data which included steganography to hide the code within pdf files masked as personal documents attached in emails and uploading of encrypted zip files to a third-party website [57]. A summary of the profile is:

- Scope: targeted
- Duration: short to long-lived
- TTPs: Similar as seen in the annoyance profile plus extortion malware techniques, phishing or SPAM emailing, and hosting malware on legitimate websites.
- Motivation: money, exfiltration of intellectual property for purposes of eliminating years of research and development, competitive economic and/or military advantage

Nation Sponsored

Representing the upper bound, nation-sponsored attackers are targeted, organized, and well-funded by a government entity or influential military to carry out cyber espionage and pilfering data. Commonly referred to as Advanced and Persistent Threat, these campaigns are usually medium to long term where exfiltration of intellectual property for purposes of eliminating years of research and development, competitive economic and/or military advantage. Typical TTPs include zero-day exploits, social engineering, phishing, watering holes, and focused perimeter-based attacks. Many of the APT reports by Manidant such as its first, APT1, describes how the Communist Party of China tasked the Chinese People Liberation Army to

perform such attacks [50, 51]. An example Nation sponsored is the Lazarus Group Cobra Guardians of Peace/Unit. This group is believed to be a North Korean government funded hacking team. They were responsible for the catastrophic hacking of Sony Pictures in 2014 which caused a significant amount of destruction to Sony data and their ability to operate [55, 46]. A joint Department of Homeland Security & FBI Alert was released on June 13th, 2017 warning of a distributed denial of service botnet infrastructure and other tools the group has been utilizing to target media, aerospace, critical infrastructure, and financial organizations [52]. Another example is a group called Shadow Brokers which is responsible for the creation of Stuxnet malware which leverage leaked NSA tools and targeted specific Siemens SCADA systems utilized by Iran in the enrichment of uranium in 2010 [53]. Eternal Blue, one of the SMB vulnerabilities released by the Shadow Brokers in the April 2017, was believed to be utilized by the Lazarus Group for WannaCry [54]. Another hacker group named OilRig group, which categorized as APT 24, targets primarily the middle eastern region with cyber espionage. This group is believed to be performing cyber espionage reconnaissance efforts to support Iranian nation-state interests focusing on finical, government, energy, chemical, and telecommunications [101]. A summary of the profile is:

- Scope: semi-targeted or targeted
- Duration: medium to long-lived
- TTPs: zero-day exploits, social engineering, weaponized phishing, watering holes, and focused perimeter-based attacks.
- Motivation: exfiltration of intellectual property for purposes of eliminating years of research and development, competitive economic and/or military advantage

Basics on Cyber Intelligence and Indicators

By design of the technology we use, cyber attackers leave evidence behind. This evidence can be produced at various stages of a cyber-attack and come in many forms such as logs, files, processes. These cyber clues are key in developing an intelligence foundation of adversary cyber TTPs. By capturing and sharing this information, it is possible to be further prepared in detecting or discovering and attacker [67]. Developing an indicator surrounds the notion of giving meaning or bounds in defining an indication that something exists with pre-defined criteria. An Indicator can be thought of as a clue or flag to a bigger activity or situation. Being able to identify something based on an indicator can help identify potential concerns or cause an action to take place [65, 67]. An indicator can be as simple as checking to see if a user logged into a system or an obfuscated string inside a file. With massive amounts of information being generated by the continued addition of IT devices, there is a vast big data problem [66]. Among this big data are the very clues that lead to the TTP of a cyber adversary.

There has been extensive research into using security intelligence in cyber defense. Intelligence Driven Defense (IDD) is how defense contractor Lockheed Martin refers to their approach in tackling the cyber attacker [65]. Their approach spawns form the ideas in their paper Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. In this paper they describe the IDD concept as a risk management strategy that addresses the threat component of risk by incorporating analysis of adversaries, their capabilities, objectives, doctrine and limitations. They furthermore discuss how this is a continuous process leveraging indicators to discover new adversary activity which thus creates additional indicators. They define indicators as either being atomic, computed, or

behavioral. Atomic meaning cannot be broken down any further an is at its lowest term. Computed meaning indicators which are derived from data created by a cyber-attack incident. Behavioral being collections of computed and atomic indicators outlining an event of the two combined [65, 72]. Referencing the Cyber Kill Chain in figure 1, if an adversary is using email to deliver a weaponized attachment, attributes about this adversary may be known such as the email address used [3]. This would cause their action to be blocked and thus no further penetration into the subsequent level of the chain. However, if indication of attacker activity was discovered in the later phases of the chain, then there would be additional clues in the preceding phases [65].

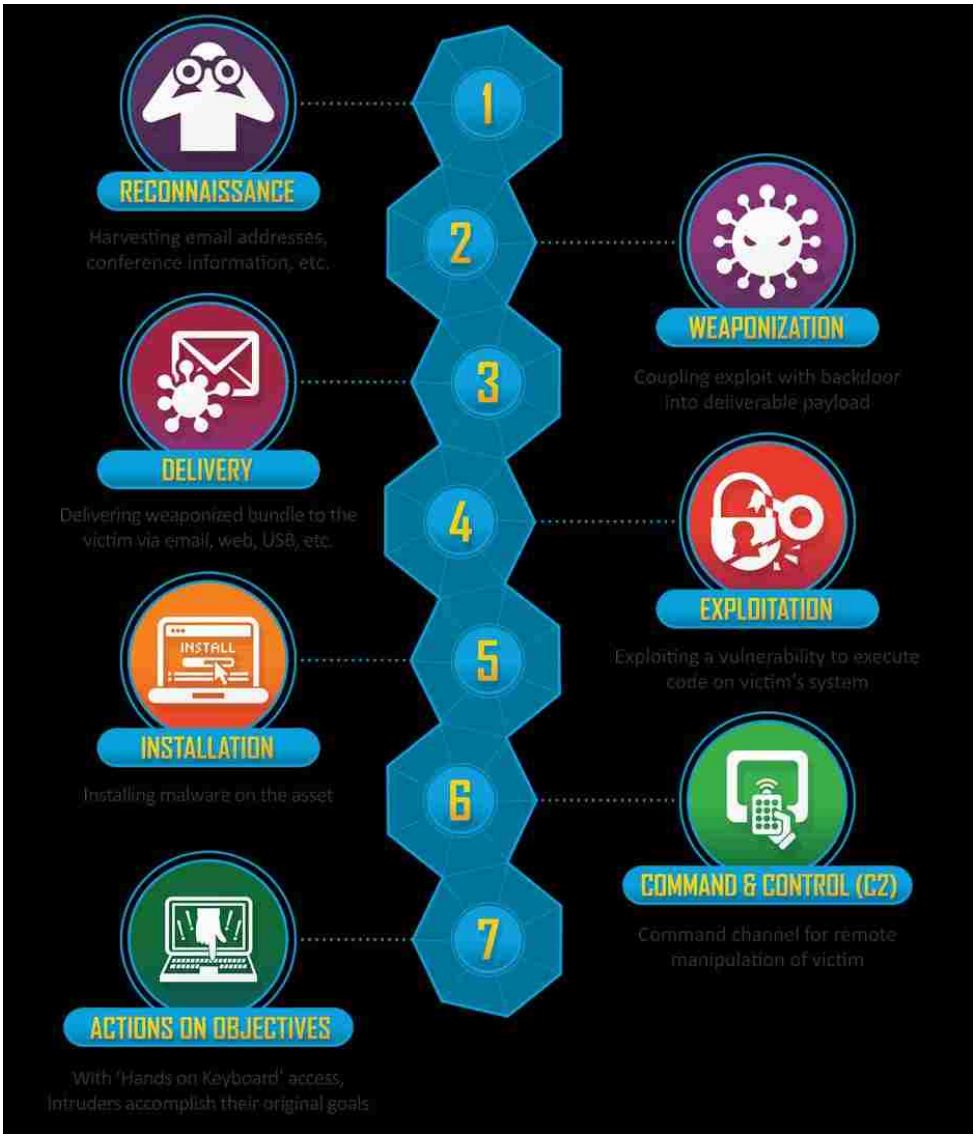


Figure 1: Lockheed Martin Cyber Kill Chain

Many projects such as Open IOC, industry companies such as FireEye, government funded groups such as the Defense Security Information Exchange have explored and shared in the idea of intelligence through indicators to detect. The Open IOC project (indicator of compromise) is a collection focusing on known TTPs in a computing environment [68]. The use

of open source indicators is a good place to gain intelligence and see how others are tackling the indicator challenge and sharing the actions attacker carried out. The DoD Defense Industrial Base (DIB) Collective Sharing Environment (DCISE) host out of the Department of Defense Cyber Crime Center (DC3) serves as a network for DoD contractors to share unclassified threat information to broaden the detection and warning capabilities [69].

Indicators can take on many forms and are primarily driven by indicating something by giving a clue to a larger context. Indicators allow for something to trigger, correlate, or flag on its existence. The indicator lifecycle begins with artifacts of data that is either generated because of an attackers TTPs or of due to the actions of a defender. This data or artifacts can indeed be something of value and a clue that sheds lights onto an activity. As figure 2 depicts, artifact data can be generally noise but upon being something of value it becomes discoverable where it can be found and identified or tied to a given fact. Once this artifact is created as an IOC is generally detectable and represents a smaller subset of the grander universe of data that has significant relevance to an event of interest [7].

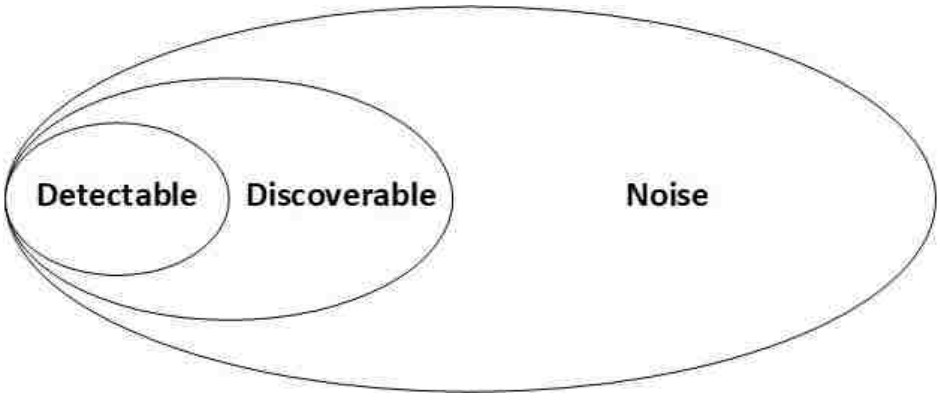


Figure 2: The universe of cyber data

Existing research in Cyber Deception

Deception comes in many forms and the distinction between what is and not deception can have blurry lines. For this Bell and Whaley proposed two simple concepts of dissimulation, hiding the truth and simulate, showing the false. Following this simple concept ensures that the intention is about deceiving versus another behavior. Dissimulate and simulate can be dissected in many ways but some defined techniques behind dissimulation include masking the real to blend in, repacking to appear as something else and dazzling to confuse. Some of the techniques behind simulate include inventing the false when it does not exist, mimicking the false by appearing to be like something else, and decoying to divert attention elsewhere. Using these basic techniques helps better define and develop deception for cyber defensive [12].

Cyber deception is frequently synonymous with nefarious activities led by attackers. It is the act of a cyber defender using deceit to study, detect or learn from the adversary. Security through obscurity has always been viewed as a negative practice in the IT security world [13]. Any additional security that can add a level of defense makes sense versus not having it at all (most situations).

Since several motives are behind cyber defensive deception and many definitions exists on the specifics, several models have been developed to help frame the cyber defense deception process. Each model brings its own unique approach to formulate a progression of sorts on how cyber defensive deception is played out and perceived. Some models have limitations and are only fixated on specific types of deception. Three existing models are described in which each brings its own unique approach to organizing and illustrating the deceptive process. With their approach there are both commonalities and difference in the attributes they describe. In addition,

they each have their own method of execution and their ability to decide the flow of events. Both the attributes and decidability are the focal point of understand the models and their process.

Basic Model

The Basic Deception-Process was created by Yuill and others [14,15] as a model for designing deception operations of Computer Network Defense (CND). There are three main stages and a decision gate that resembles a deterministic state diagram approach. Referring to figure 3, the first stage is the deception operation development which is further broken down into three sub categories of planning, building the deception, and preparing to engage target. *Planning* includes the deception goals, identifying potential targets, for the deception, or determining operation requirements. Next, *Building the Deception* emphases on the story line and the overall event schedule. The storyline is the pseudo outline, detailed as necessary, of the deception structure. Lastly, *Preparing to Engage the Target* is ensuring the readiness of the deception. Target is used to represent to interacting with the deception, namely attackers.

Once the Deception Operation Development stage is complete the model continues to the Deployment stage. In this stage the main objective is about deploying the deception "storyline" to production. For instance, if the deception involved a web application, the first stage would include the planning of the web application, building the web application, and preparing it to go live. This stage makes the web application available and visible to external targets [14].

Once a target is engaged the model steps to the Target Engaged phase which surrounds the idea of whether the target has been deceived. Procedurally, the deception story is received,

accepted, and the intended action was taken. Any resulting feedback, intelligence, which the target may have created is recorded and either alerted to or analyzed later. At some point during or after the target engagement, a decision gate determines if the deception should continue "as is", return to the Deception Operation Development stage, or terminate the deception all together [14, 15].

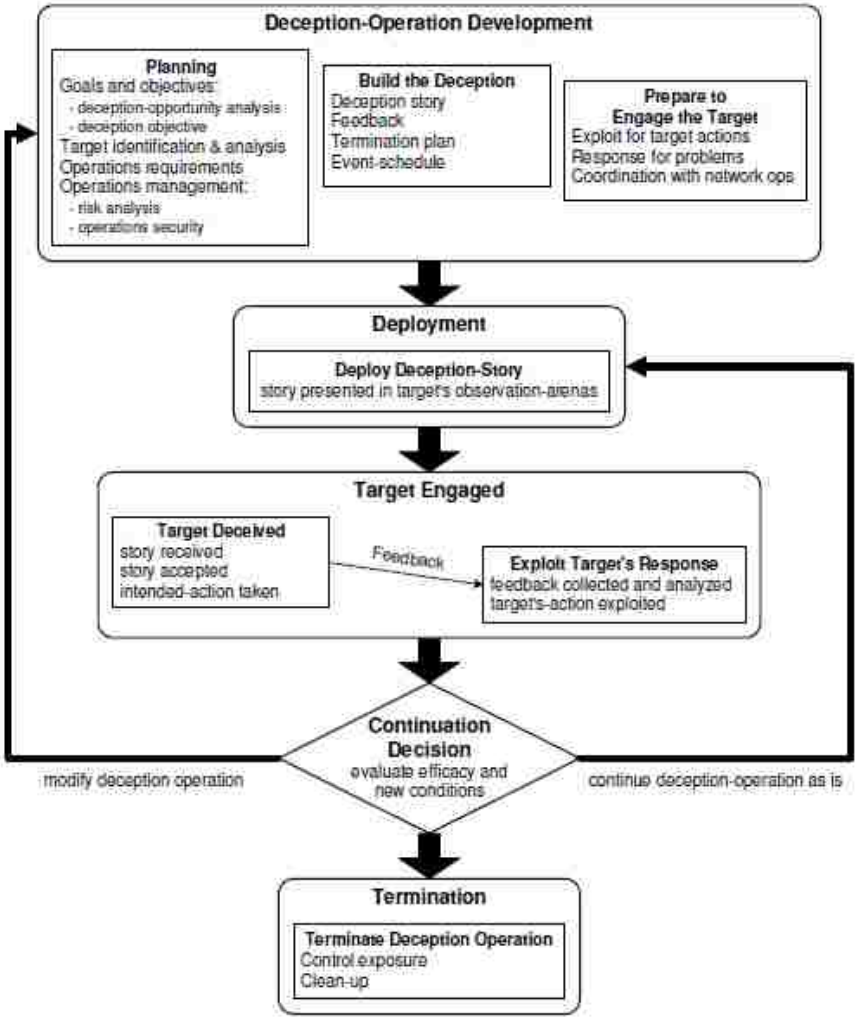


Figure 3: Basic Deception Process

The Incorporated Model

The incorporate model shown in figure 4 was presented in a paper titled Planning and Integrating Deception into Computer Security Defense by M. H. Almeshekah et al [16,17]. Besides being linear in nature, the model is comparable to the Basic model by having three main stages which include planning deception, implementing and integrating deception, and monitoring and evaluating deception. The first stage is broken down into six sub categories where each is concentrated at ultimately orchestrating the deception and how it can be incorporated into computer defense.

The first sub category of the planning deception stage is defining the strategic goal and what is the hopeful outcome of the defensive ruse. Ensuring the overall objectives are defined and the actual deception is plausible is key. This leads to the next sub category which emphasizes on how the attacker (synonymously also referred to as targets) will react to the desired deception. This can be a very important depending on what type of attackers are trying to be deceived. The degree of complexity in the deception can really be impactful for script kiddie style attacks but prove almost comical to nation state sponsored adversaries that exhibit APT campaign style opposing force attacks [5, 18]. This is where the third sub category plays in integral part in understanding the attacker biases. The attacker bias studied upfront helps to perceive possible preconceptions of attackers and the degrees of complexity they are capable of. Knowing or even planning this upfront can make the deception better aligned with the right target attackers. We focus on the importance of knowing the attacker and their bias in chapter 3.

The forth sub category of the planning stage concerns what is simulated or dissimulated for the deception to. This step is centered at identifying and analyzing the patterns and

characteristics of what and when to deceit. The Incorporation model research further explains various system components where deception may be applied into the functionality and/or state of the system. State being for example System Software and Services, System Activity, System weaknesses, System configuration and several others which are listed [16]. To put simply developing the storyline of the deception.

The fifth sub category is feedback channels and knowing what to look for when the deception occurs. To have successful feedback channels, it is critical to understand what to even look for within the system and the deployed deception. For example, if the deception involved hosting a web application that as purposely susceptible to a specific vulnerability and would require and advanced attacker to discover and exploit, then the scope is significantly small, and the feedback is thus narrow to that one thing. However, if the deception was designed to include an entire operating system being subject to compromise by attackers, then the feedback is much wider. This causes a higher amount of required intelligence to effectively monitor and recognize attacker activities. Monitoring is key to knowing if your deception is successful or not. This leads to the final sub category under planning which is the risk and countermeasures. Deceptions have risk and specific consequences if all does not go as planned. Knowing these risks and developing hypothetical countermeasures is an effective way to further ensue a successful deception [16].

Once the planning stage is complete the model moves to the Implementing and Integrating Deception stage. This is a straight forward step dedicated on implementing the deception that was planned and described in the storyline. Once deployed, the model moves to the next stage which is monitoring and evaluating the deception. Taking the identified feedback

channels from the planning stage and monitoring the deception for the identified activity to demine if the deception has been engaged upon and the activities of the attackers are what is expected. The final output is whether the attackers believed, suspected, or disbelieved the deception which result is the model halting [16].

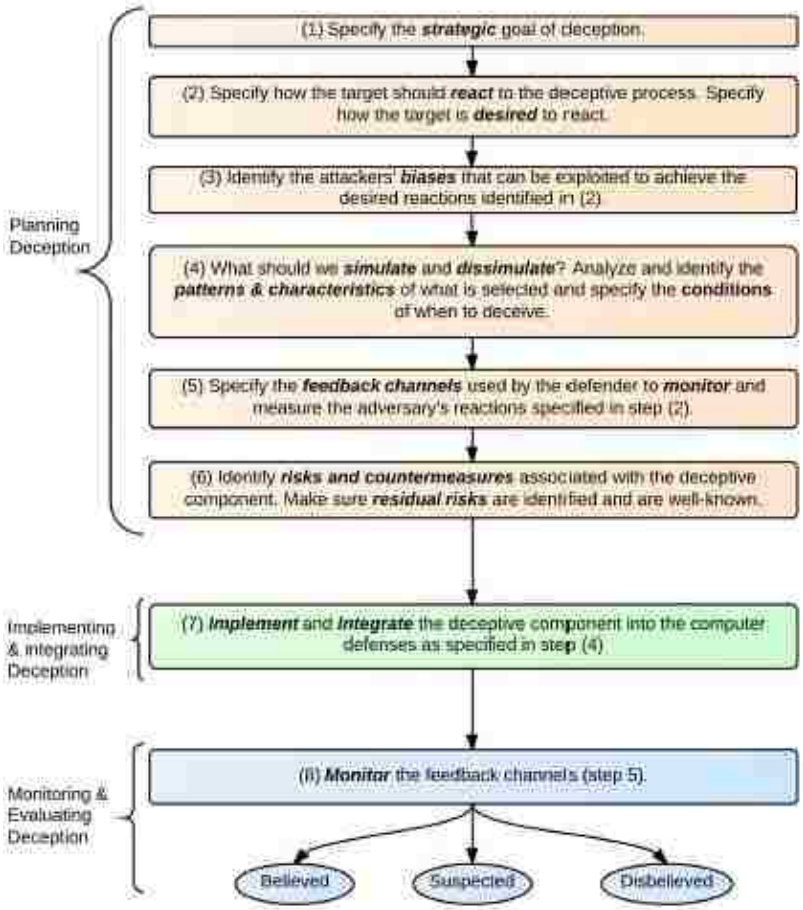


Figure 4: Deception Incorporation model. Process is linear in nature and starts in the Planning Deception section and end in the Monitoring & Evaluating Deception section with a ultimate believed, suspected, or disbelieved.

Cyber D&D Model

A third model is presented by K. Hecman et al [19] in a paper called the Spiral Cyber Denial and Deception lifecycle management process. This model, unlike the Basic and Incorporated models is cyclical in nature allowing for iterative progression of the deception until it has been tested and considered ready for production. Referring to figure 5, the process beings in the center of the circle inside the Plan Cyber D&D quadrant. This quadrant is very similar to the Basic and Incorporated models which is concentrating on planning the deception. While the approach in this first stage compared to the other models is similar, this model introduces some different elements. The cyber D&D TTPs training curricula, best practices, and metrics are examples of elements not seen in other models being reviewed. These additional elements are observables and gauges to the effectiveness of the deception being planned which can potentially generate useful metrics for later analysis [19].

Once the planning stage is completed the model spirals to the implement quadrant. This portion focuses on incorporating any necessary tools that are required, additional threat data which is very similar to the TTPs, and further concentration on managing the metrics collected. As the defined deception is further compiled it is then passed in the spiral to the next quadrant on deploying and executing. During this stage, additional fine tune modifications are considered to ensure the successfulness of the deception. Monitoring, observing, and collecting field data and further metrics are also a part of this stage. Once the deception is considered complete or has run its course, the final quadrant is post deployment analysis [19].

Post deployment analysis is studying the deception through the spiral lifecycle and assessing its effectiveness. The deception outcome is analyzed to determine if additional

improvements are needed in the planning and implementation stages and if the deception is still a prototype. Reviewing the feedback to the deception planning is also considered as another measurement to help make the determination on if additional prototypes are needed to get the deception in a desired state. The spiral continues around the circle through the four quadrants with the intention of the spiral tightening to a final state in the center of the circle within the deploy and execute stage [19].

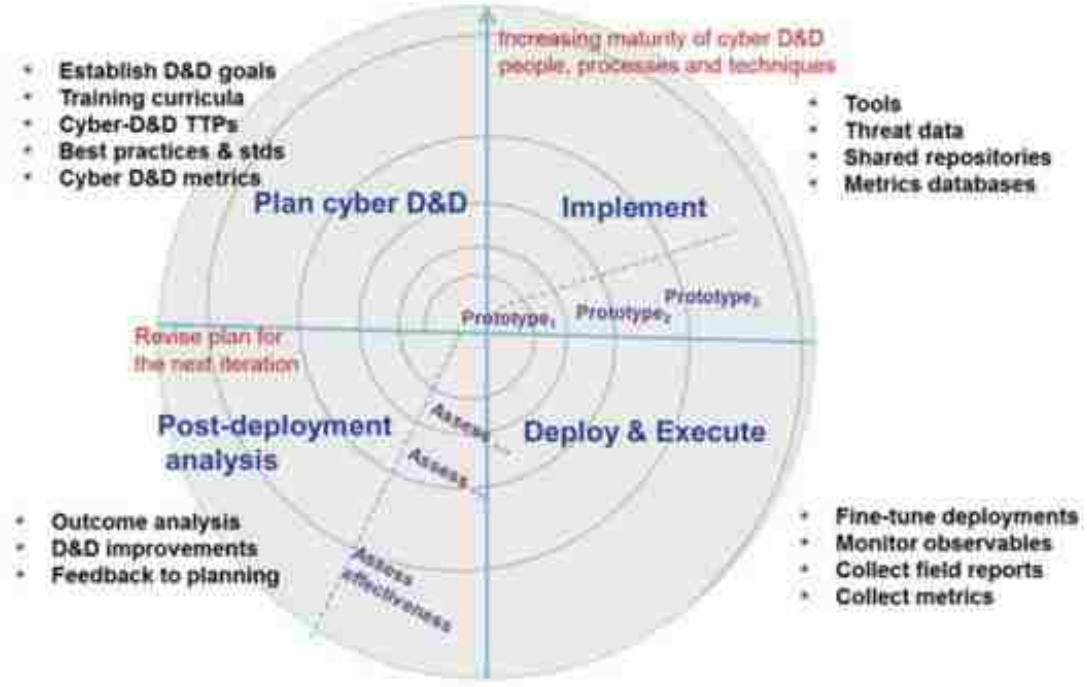


Figure 5: Spiral Cyber Denial and Deception lifecycle management process. Process starts in the Plan cyber D&D quadrant and spirals clockwise through the other quadrants until ending up in the Deploy and Execute quadrant after many possible iterations. The tightness of the spiral can dictate the number of possible iterations.

Common Tools and Utilities

Any number of tools or utilities could be used to conduct research, perform testing, enumerate information, or exploit a vulnerability. In our research, many tools and approaches

are leveraged for evaluating and detecting a deceptive system, namely honeypots. Some of the common tools which are used and mentioned in this research are highlighted in this section to give additional context to their purpose and usage.

Nmap

Nmap is an acclaimed open source network mapping tool that performs network discovery and security auditing. The tool has several options and can be leveraged for network inventory, service patch levels, monitoring system uptimes, and more. The tool leverages raw IP packets in unique ways to determine what hosts are available, their listening services, versions, operating systems, potential firewalls in use, and dozens of other characteristics. The tool also has an incorporated scripting engine which allows for the execution of code that exists as packages with the tool or can be custom created. Scripts for example can examine an IP and service and determine if its vulnerable using proof of concept exploit code. Nmap traditionally is used via the command line interface but comes in a graphical user interface (GUI) called Zenmap [77].

Metasploit

Another common tool known to many elite and script kiddie attackers alike is the Metasploit Framework written by H.D. Moore and is currently part of Rapid7. This tool focuses on developing, executing, discovering, enumerating and exploiting code against remote target systems. This is done by coded modules written in ruby that can typically be configured within the frameworks msfconsole utility. For example, if a remote system was vulnerable to a well-known exploit on a remote port, using msfconsole a pre-coded module most likely exists that can

be configured and executed to exploit the vulnerability. Metasploit is known for its shellcode creation, which is a small piece of code used as the payload in the exploitation of a software vulnerability [76]. The framework also includes packing capabilities to hide code within binary and obfuscation tools to help evade anti-virus. [62, 77].

CHAPTER THREE: DECEPTION AS A SOFTWARE

Many of the deceptive processes outlined in the background chapter are deployed as software solutions commonly referred to as honeypots. This chapter explores the background of honeypots in addition to specifics that will be used throughout this research. We will start by exploring the motivation of a honeypot. Next discuss the various attributes that make up the various kinds of honeypot interaction. We review past research and experiments done in testing honeypot systems. Finally, we will review some of the different SSH based opensource honeypots using within this research.

Honeypot Fundamentals

Honeypots have been in existence for quite some time introduced publicly in the 1990s and early 2000s. The Cockoo's Egg, a novel by Clifford Stoll, references using a honeypot to track a spy committing computer espionage [85]. In 1990 the BackOfficer Friendly, a simple to install, easy to configure, free honeypot developed by Marcus Ranum at Network Flight Recorder was designed to run on the Windows operating system and emulate several services such as to FTP, Telnet, SMTP, HTTP, and POP3 [83]. Founded in 1999, the Honeynet project is a leading international non-profit security research organization focused at studying the latest attacker TTPs and developing open source security tools to improve Internet security [33]. In 2002, a paper titled *The HoneyNet Project: Trapping the Hackers* by Lance Spitzner describes the evolution of the Honeynet project and how that has led to defining some of the first honeypot nomenclature such as the types, purpose, of what defines a honeypot [84, 86]. Today the use of

honeypots is every more expanding. As the cyber war escalates there has been an increased demand in acquiring new intelligence and alerting.

Honeypots are picked at, probed, and attacked just as any other IT computing device is subject to. The main difference, a honeypot is designed and developed to monitor, aiding in the detection of Cyber attacking or information systems misuse. The motivation can be: 1 - ways to deflect or distract the attacker to a false product system. 2 - To gain intelligence and study their TTPs in a research capacity, specifically unknown attacks. 3 – To simply be used as alerting mechanisms [24,25].

Honeypots can be categorized based on their design and deployment approach [86]. The design can differ depending on the objective of the honeypot, typically focusing on the motivation, data collection or alerting. The two main honeypot design types are emulated and pure. Emulated honeypots are coded software that mimics a real application, network socket, operating system, and other service. Many emulated honeypots have been developed over the years and are available opensource. Pure honeypots are completely based on production or real operating systems and applications with additional monitoring, tapping, or recording techniques to capture attacker activity [74]. Since pure honeypots are based on the real system or application they are typically custom built for each situation. However there does exist packaged pure honeypot solutions such as TELPOT which leverages a real Telnet services but adds in a proxy layer to intercept and record all data [35].

Using table 1 below, honeypots have many tentacles to their design which directly correlates to the interaction level or experience the attacker is presented with. For both emulated

and pure designs, the interaction levels can range between low and high with various possibilities in between. Low interaction typically only allows minimal interfacing with the honeypot such as a static web page or TCP socket [22, 23]. For low interaction, most of the backend parts of the honeypot system are considered isolated and is primarily focused on logging. High interaction honeypots are more sophisticated as they are meant to be more complex and engaging, providing an experience to the attacker with a system or application [21]. A high interaction honeypot can easily be setup as a low depending on its configuration. For example, restricting access to just the FTP service login prompt verses allowing the attacker to guess and be successful at logging in and interacting with the actual command prompt. The higher the interaction level, the more information is typically collected for studying attacker TTPS such as system and application logs. On the contrary, the higher the interaction level the increased probability the honeypot will be detected by the attacker. The lower the interaction level the less maintenance is required. Scalability is also easier the less intensive the deception solution tends to be.

Table 1: Honeypot design types and resulting components

	Emulated		Pure	
Interaction	Low	High	Low	High
Maintenance	Low	High	Low	High
Risk of Attack/detection	Lower	Higher	Lower	Higher
Intelligence gathered	Limited	Extensive	Limited	Extensive
Typical Code	Opensource		Custom Built	

The deployment approach can vary as honeypots have been developed in many types such as the traditional single host or service, deployment and connected to honey networks,

honey tokens, and honey bots [26]. The main driver behind the type of honeypot is indicative of its purpose. If the idea is to study attackers in the botnet arena, then honeypots would be developed to mimic either a botnet server or more so a client of an existing adversary botnet cluster. The idea of linking honeypots together in a network or mesh is not new. In 2006, a presentation by the Honeynet project organization call GDH1 – Global Distribution Honeynet was focused on deploying high interactive honeypots globally with a central location for data collection [32]. This same project was further expanded in 2007 with GDH2 named HonEeeBox which focused on Rapid Deployment of Many Distributed Low Interaction Malware Collectors [33]. The main intention was fast, cheap and the ability to create a massive malware sensor grid collecting data.

The ability to gain access to wide range of emulated honeypot software has become trivial with code repositories like GitHub [27]. The ability to have pre-configured containers "ready to go", such as Docker, essentially allows for a honeypot to be deployed in minutes [28]. The idea of multi-honeypot packages has also become popular. A common one is T-POT which contains a custom-built web interface to manage numerous docker deployed honeypots and its collected data [63]. This ease however includes the risk of honeypots being too simplistic, "rubber stamped" or perhaps known and detectable by the adversary. Some honeypots are indeed real systems, services, or application that instead have been rigged with a significant amount of monitoring and proper network isolation. With virtual technology becoming widely used in legitimate servers and services, the idea of a honeypot being hosted on virtual environments is not as suspicious as once before.

Opensource Honeypots Used in This Research

Below are the emulated shell-based honeypots which will be used in this research for various experiments and testing. Depending on how they are used, they can range from low to medium-high interaction. Low based on accessing only the login prompt and TCP socket. Medium-high based on logging in and interacting with the shell. In this research, it is the intention to use the honeypots as configured and downloaded for one variant. Then subsequent variants with changes made to its configuration with the motivation to further elude the attacker.

Figure 6 illustrates the linage for when each honeypot was conceived, last updated, and more importantly where it may have been forked from one another. The foundation for many of the common honeypots seen today stem from the original Kojoney honeypot. A fork was made to create kojoney2 which was then the basis of Xsweet. Equally Kippo was forked from kojoney and then resulted in the inspiration and foundation for Cowrie. Each bring their own characteristics but share common elements such as using python and more specifically the twisted packages which provides an event driven network engine for development in TCP.

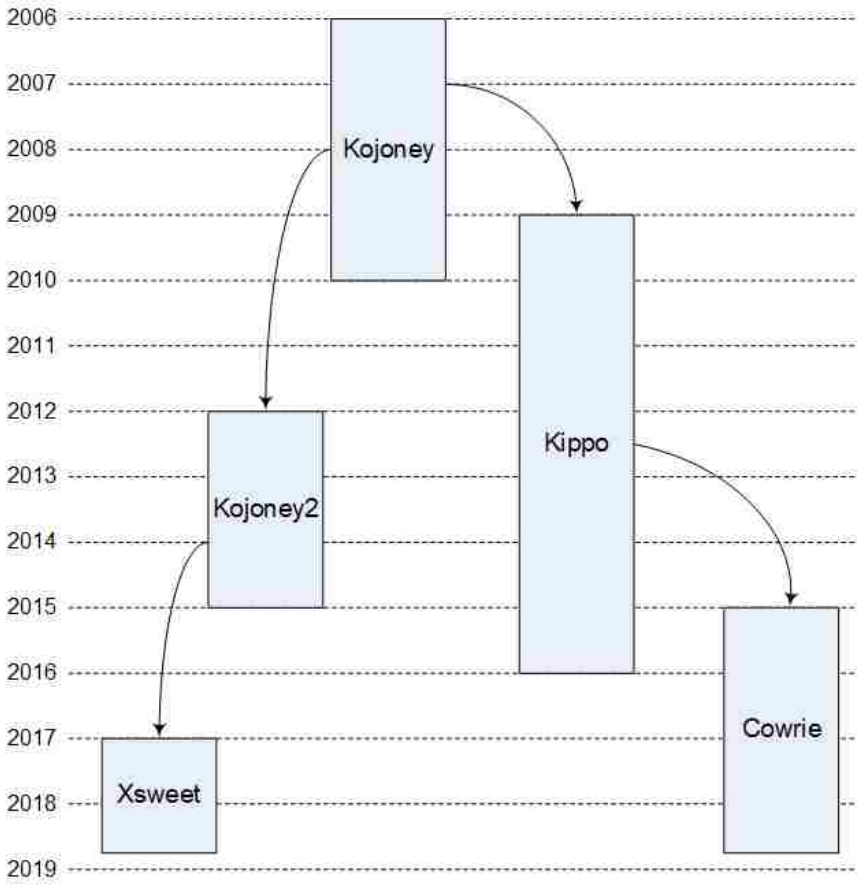


Figure 6: Lineage of opensource honeypots used in this research. Depicts Kojoney as the original honeypot with subsequent honeypots forked and related.

Kippo

Kippo is a medium interaction SSH honeypot based directly on the SSH honeypot named Kojoney [30]. Originating in circa 2009, Kippo is designed to log a wide range of attacker activities to include brute force login and full shell interaction. Some key features include a fake file system with the ability to add/remove files and the use of trickery in pretending to execute binaries, commands and responding to the user. The honeypot was developed in Python and Python Twisted and is somewhat maintained by community contributions. Kippo was last modified in September 2016 and is available as source binaries on GitHub in addition to several

custom configured Dockers containers that exist today [29]. It is suspected that the version of Kippo being used in this research is 0.9. This version has remained as the documented version since 2014.

Cowrie

Cowrie is a fork from the popular honeypot Kippo. A security researcher, Michael Oosterhof, took notice circa 2015 of a Kippo deployment that was not logging complete details of some of the discovered SSH attacks. As a result, logging was enhanced and additional capabilities such as proxy support, SFTP, SCP were added. This resulting branch was named Cowrie to distinguish it from the original software. Like its predecessors, Cowrie was also developed in Python and Python Twisted. Cowrie was last updated in August 2018 at the time of this dissertation writing and is currently at version 2018-6-29. The source binaries are available on GitHub in addition to the many custom configured Dockers containers [31].

Kojoney2

Kojoney2 is based of the original honeypot project Kojoney developed by Jose Antonio Coret [91, 92] and was last updated in December 2014. Derived from the Spanish word cojon combined with honey, Kojoney was developed by Justin Klein Keane in around 2012 to mainly improve many of the "bugs" discovered in its original predecessor and to also re-structure the configuration of the honeypot. Kojoney2 is written in Python and Python Twisted and primarily focuses on simulating a real SSH environment. Kojoney2. More attention was placed on how the honeypot responds to legitimate shell commands by expanding this capability over the

original Kojoney. Kojoney2 is currently at version is 2.1 and is available for download as source binaries on GitHub [92].

Xsweet

Xsweet was developed by Ousama AbouGhoush and in 2014 was a fork from the kojoney2 honeypot. Xsweet is based off Kojoney2 and shares a similar functionally and motive of logging attempts to gain access and providing shell functionality to the attacker. Xsweet pays more attention to an elaborate shell experience as the authors emphasis was to capture what the attacker did once inside. In the original motivation for this honeypot, it was deployed to a cloud hosting service where massive brute force login attempts were attempted against all ports and services. Xsweet is assumed to be at version 1.0, was last updated in February 2018, and is available for download as source binaries on GitHub [36].

SSH Honeypot Detection

Trying to detect a honeypot service is like looking for a noise abnormality when attempting to detect a submarine from the natural sounds of the ocean. Different techniques have been attempted and some successful in detecting the presence of a honeypot service. As each honeypot is developed and customized to its deployable requirements, certain characteristics can be evident that separate themselves from other services. For example, a real SSH server may respond differently in the networking TCP packets versus a Kippo honeypot. Others include the way the banner statement is portrayed, TCP fingerprint, simulated operating system version, login prompts and many more characteristics that can be possible indicators to authenticity.

Some but very little has been developed to aid in the detection of honeypot technology. Some commonly known cyber software tools have been enhanced with indicators that flag when responses about a service appears to match a honeypot characteristic. Nmap is a good example of where detection indicators were leveraged to help further strengthen the possibility to accurately fingerprint or detect a remote TCP service. Figure 7 are snippets of the Nmap services file that contains characteristics that equate to a honeypot software. For example, in the last snippet a specific way the simulated SQL server responds matches the recorded indicator thus flagging it as Dionaea honeypot MS-SQL server [77].

```
match ssh m|^SSH-2\.0-Twisted\r?\n| p/Kojoney SSH honeypot/
i/protocol 2.0/


match ssh m|^SSH-([\d.]+)-OpenSSH_([\w.]+)\r?\n.*aes256|s p/Kojoney
SSH honeypot/ i/Pretending to be $2; protocol $1/

match honeypot m|^503 Service Unavailable\r\n\r\n\0$| p/Network
Flight Recorder BackOfficer Friendly honeypot/

match honeypot m|^\r\nlogin: \0$| p/Network Flight Recorder
BackOfficer Friendly telnet honeypot/

match honeypot m|^\r\n[-\w_.]+ [\d.]+ - Unauthorized access
\x07prohibited under penalty of law\.\r\n\r\nlogin: \xff\xfc\x01|
p/Whiz Kid Technomagic Imaginary telnet honeypot/ o/Windows/
cpe:/o:microsoft:windows/a


match honeypot m|^Microsoft Windows XP \[Version [\d.]+\]\n\(C\)
Copyright 1985-\d+ Microsoft Corp\.\n\nC:\\>| p/honeyd cmdexe.pl/

match ms-sql-s
m|^\x04\x01\x00\x2b\x00\x00\x00\x00\x00\x00\x1a\x00\x06\x01\x00\x20
\x00\x01\x02\x00\x21\x00\x01\x03\x00\x22\x00\x00\x04\x00\x22\x00\x0
1\xff\x08\x00\x02\x10\x00\x00\x02\x00\x00| p/Dionaea honeypot MS-
SQL server/
```

Figure 7: Snippet from the Nmap service identification file

Metasploit is another tool that has honeypot detection capabilities added such as the module named detect_kippo written by Andrew Morris. Seen in figure 8, an extract of the ruby code sends unexpected data to the remote network SSH service and in its response checks for specific error messages. If those messages matches the indicators identified in the module "Protocol mismatch" and "bad packet length" then it returns as a Kippo honeypot. Figure 9 demonstrations the detect_kippo module being accessed within the Metasploit msfconsole [69].

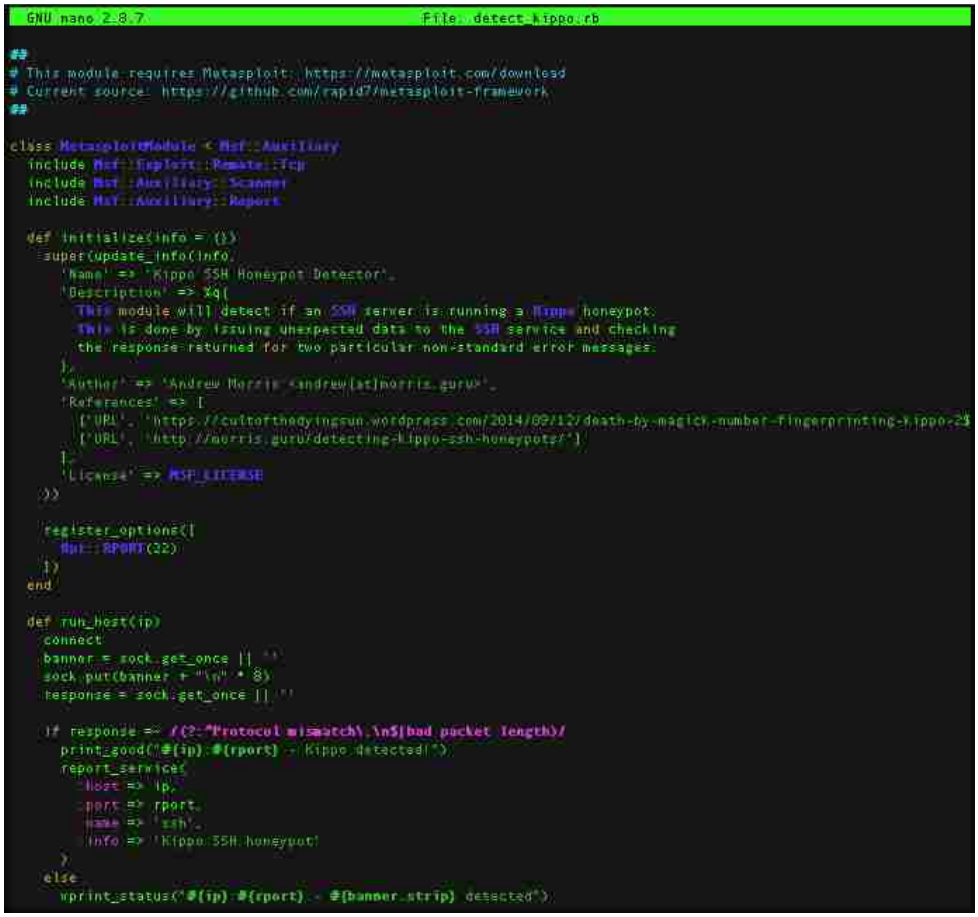


Figure 8: detect_kippo Metasploit framework module

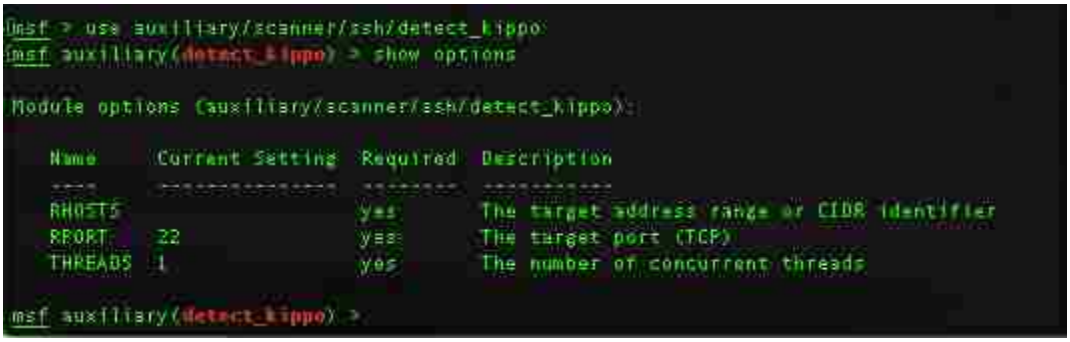


Figure 9: detect_kippo being accessed through Metasploit msfconsole.

In a recent research paper, dated around the time of this dissertation writing, titled “Bitter Harvest: Systematically Fingerprinting Low-and Medium-interaction Honeypots at Internet Scale”, the authors Alexander Vetterel and Richard Clayton focused on techniques to identify honeypots based on specific flaws within protocols that are subtly different from the systems they are impersonating [75]. The research focused on three protocols starting with SSH then telnet, and finally HTTP/Web. Looking more closely at the SSH protocol, the research looked for deviations in responses to a client version string SSH2_MSG_KEXINIT and secondly at the SSH2_MSG_KEXINIT packet.

For the version, they created 192 specific client version strings following the syntax of “SSH-protoversion-swversion SP comment crlf”. They compared several systems which included five-real OpenSSH servers, and an instance of TwistedConch, four Kippo honeypots, and four Cowrie honeypots. For the SSH2_MSG_KEXINIT packet, they focused on the algorithms defined in RFC 4250 where they generate thousands of custom variants using each algorithm, correct and incorrect packet padding, random bytes, and more. In total for both the algorithm packet and client version, they generated over 58k custom packets.

After sending over 157 million probes where each character is recorded from a server response they gain insight into normal SSH connection behavior. The find that the most common client version string is "SSH-2.2-OpenSSH \r\n" and for details in the SSH2_MSG_KEXINIT packets they discovered that "ecdh-sha2-nistp52" is the most common key-exchange algorithm, "ssh-dss" as the host key, "blowfish-cbc" as the encryption algorithm, "hmac-sha1 as the mac algorithm, and "zlib@openssh.com" as the compression algorithm. After conducting scans internet-wide based on what they learned from the smaller subset of systems, they indeed were able to identify a considerable number of Kippo and Cowrie honeypots as each responded differently to the custom packets. It was determined that most implementations of SSH services were OpenSSH 6.6 and 7.2. They were able to gleam that the SSH2_MSG_KEXINIT packets used random bytes for Kippo and Cowrie compared to OpenSSH which uses null. The abnormality could also be seen for the various encryption algorithms being supported by the honeypots versus the real SSH servers. By the completion of the research, Cowrie had already addressed the use of random characters in SSH2_MSG_KEXINIT by replacing it with null.

Past Research on Testing Honeypots

Historically honeypot research has focused twofold, being the novel new honeypot technology and very much on what sort of data the honeypots collects. As new honeypots are being conceived, developed, tweaked, deployed, and eventually attacked, the question is raised on how effective honeypots are, especially to varying attackers. With the ease of access and quick deployment capabilities, a Cyber defender could deploy a honeypot, visible to attackers,

with the hopes it assists them collecting data to better understand their enemy or to lure them away to something more enticing. The defender may have no idea if the honeypot software contains any coding errors or vulnerabilities thus allowing for undesired vectors of attack. Or how easy is it to detect that this software is deceptive in nature which results in altering the attacker TTPs. This section will review some of the different experimentation and testing experiments done by past researchers. These experiments give insight into what has been attempted, how it worked, results, and what to experiment that is different.

The first research experiment centered on providing an environment where attackers were asked to hack their way through a series of systems, penetrating as far as they can go to the end state. Deceptions where used in some cases during their journey to see how this changed the results, additional time it took, and different vectors they ventured to reach the end state. The second research experiment was testing the effectiveness of honeypots in different environments by determining if using virtualization effected the results. Both a residential internet connection and the network for the Naval Postgraduate School (NPS) were used as entry points. In summary, all three research of experiments presented a diverse approach to testing the effectiveness of cyber deceptions and honeypots.

Red Teaming Experiments with Deceptive Technologies

Fred Cohen et al in a publication named for this section heading developed and executed an experiment "designed to measure the effects of deception defenses on attacks against computer systems and networks", using the words from their own publication [82]. To understand the implications of technical deceptions in information protection, an experimental

assessment was conducted with specific deceptive methods done against human attackers. Several objectives were outlined:

- Improving the knowledge of the participants on how systems can be attacked and defended.
- Understanding how much it takes to tell the attacker before they finally area able to defeat the deception.
- Understanding the dynamics of the attacker group, specifically on how they were with success, failure, impact on workload with deceptions, coming up with ideas, stagey, and tactics.
- Understanding the impacts of initial access to deceptive defenses

Over four weeks, six different teams participated in the challenge where the use of deception was not communicated in the beginning of the experiment. The experiment was conducted five times each lasting four hours where six exercises were completed in each. Thus, the total number of experiments conducted was 30 where some had the deceptions turned on, turned off, or at random. Participants were academically skilled college students studying computer security at a national laboratory. The same experiments were also conducted with highly skilled ethical professional attackers that do this type of testing as a career.

Each experiment had known attack graphs followed by actual attack graphs for each group that participated. The use of these graphs helped provide a means for measuring the effectiveness of defenses by measuring the progress by attackers over time. The research discovered that when deception was enabled, attackers never got as far toward the truth as they

did when the deceptions were disabled. Meaning the deceptive techniques seemed to have worked and when enabled, the attacker would always go the route of the deception attack graph and were convinced they were heading down the right direction. When deception was turned on, the real system was less obvious while the deceptive systems were more obvious and available. If attacked and exploited, the attacker may think the game is complete. In some cases, teams would declare victory over the experiment when the deception kept them away from the real target.

The results indicated in summary that the network technology deception capabilities are very effective at what they are designed to do but can be significantly improved upon especially in its content. Another significant observation is that deception enables the attackers to waste time trolling down deceptive rabbit holes giving the defender more time to react, gain insight, collect data, and reducing defensive cost.

Testing Deceptive Honeypots

In a thesis by Aymen Yahyaoui for the Navy Postgraduate School, an experiment was conducted focused on testing the effectiveness of different honeypots and their deceptive levels in real world networks by varying their location and by adding in virtualization [81]. Objectives are outlined as follows:

- Measure the effectiveness of honeypots by comparing ingress network traffic to the honeypots to a legitimate website.
- The compare malicious and legitimate traffic distribution by country and region.
- To measure the effectiveness by varying the location and type of implementation.

The experiment took place at two locations. The first being a residential area, where a HTTP honeypot Glastopf was connected to the internet for a period of two months. The second location was within the NPS network utilizing the same Glastopf HTTP honeypot in conjunction with the Kippo SSH honeypot, also for a two-month duration. In addition to the two deployed honeypots on the NPS network, Glastopf and Kippo were also deployed on virtual machines. All traffic was captured and analyzed using SNORT intrusion detection.

The results for comparing ingress network traffic to the honeypots against ingress traffic to a legitimate website was not an easy task according to the researcher. The biggest hurdle was separating the legitimate network traffic from the malicious. The assumption was made that since the legitimate webserver was educational-institution which showed no value to real attackers. Results showed that the HTTP honeypot showed more attacks than the legitimate web server over the same period. To further make the HTTP honeypot more enticing (more deceptive) to an attacker against the NPS network, additional content was added such as links, static content from other websites, and more. This resulted in the number of attacks increasing, mostly automated scripts.

The Kippo SSH honeypots, being on both a real and virtual system, operated for more than four months and saw a significant difeent in the number of attack occurrences. One example, on the real system, Kippo saw 87197 attempts for the user root where on the virtual system Kippo saw 186214. This same increase was seen across all username and passwords attempted. The surprising result was the number of attempts conducted by each unique IP addresses was far less on the virtual system. Another interesting results for virtual versus reals

was the Glastopf HTTP honeypot also exhibited command-line attacking on the virtual system not seen on the real honeypot host.

The diversification of countries was also interesting part of the experimentation results. For the Glastopf HTPT honeypot, internet crawlers, code that is used to index search engines, was the most prevalent and not categorized as a specific country. France came in as the highest source folled by the Netherlands than the U.S. It was assumed that most would come from the Asian Pacific region, but spoofing is suspected. This did not match the comparison for the real webserver used in comparison of network traffic where China cover 60% of the total followed by the U.S. and Ukraine.

The ending results demonstrated that indeed the location changed the effectiveness of the honeypot. In testing the HTTP honeypot in both locations, it showed after a few weeks that the NPS honeypot had more attacker activity, as something on the NPS network is mostly likely more promising than in in a residential area.

Measuring the Effectiveness of Honeypot Counter-Counter deception

A third research by Neil Rowe developed software tools to assess the effectiveness of honeypots using metrics to calculate and summarizes a file system by different vectors [2]. Their focus was developing a way to test the effectiveness of a honeypot filesystem with determining how close it resembles a real file system. The research objectives are outlined as follows:

- "Design a honeypot to look like a normal computing system"
- "Given a computer system, decide whether it is a honeypot or not"

- "Design a honeypot to be maximally effective at fooling attackers into thinking its is a normal computer system"
- The compare malicious and legitimate traffic distribution by country and region.

The idea of counter-counter deception is key in this research with two strategies being possible such as seeking the unconscious clues and seeking the inconsistencies. Like intrusion detection with signature-checking versus anomaly, unconscious clues arise when speed the required of the deceiver is having trouble, such as a liar who fidgets. Other unconscious clues are the number of words used and the generality of being vague. Inconsistencies are difficult to track due to the complex nature of a deception and typically are in the form of inadvertent mistakes.

Throughout the experiment, several tools were developed to capture data in both real and honeypot filesystems such as NFDir. The most focused tool evaluated or compared filesystems from one another by conducting 36 unique metrics on each directory of a filesystem. Analyzing about 1500 common directories a minute, the content is subject to first order and second order statistical analysis. First order being the overall file and directory tree structure, names of directories and so forth. For example, the directory name summer2015 for an academic system is not abnormal. Second order statistics resulting from mathematical calculations such as median, mean. standard deviation, minimum and maximum, say for a filename. Based on their results and outcome of comparisons, it was proven successful at effectively providing a counter-couterdeception capability, that being it can detect based on the clues that a filesystem is perhaps a deception.

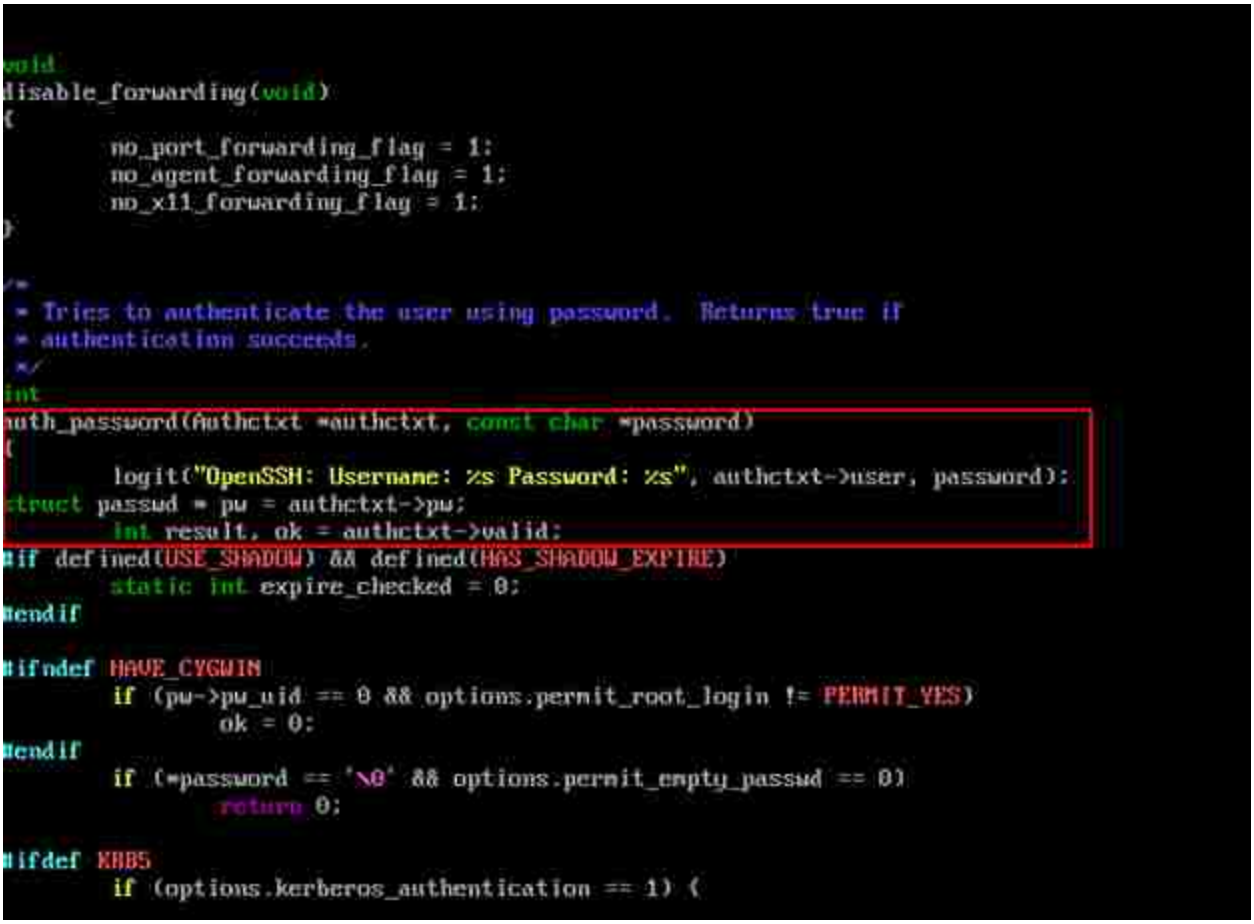
CHAPTER FOUR: CUSTOM PURE HONEYPOT AND EXPERIMENT RANGE BUILD

In this chapter we explore the customization that was necessary for conducting empirical experimental testing within this research. We first explore the buildout of a custom honeypot solution which leverages a real operating system. This pure honeypot has the main intention of differentiating itself from the normally easy to get opensource emulated honeypot solution we see today. The other major contribution is the custom build of an experiment range. This range is used to conduct experiments against the custom and identified opensource honeypot software, introducing an attacker point of presence, and monitoring activities to understand events that unfold during experiments.

Custom Pure Honeypot Buildout

As mentioned in the previous chapter, pure honeypots are built based on using a real filesystem versus an emulated approach found in most opensource solutions. As a part of the experiment range built for this research, a custom pure based honeypot is introduced alongside the use of emulated honeypots. By creating a pure honeypot, the idea is to have a system with the same motivation as an emulated solution but better blend in with real production systems. Several degrees of complexity can be applied in further masquerading the existence of honeypot behaviors and monitoring. In this research the amount of intricacy was not the primary emphasis but more so the use of a pure honeypot among many other emulated deployments as a delta. For a real filesystem to be leveraged, some configuration, customization, and code was introduced.

Using OpenSSH 6.2 on Ubuntu 14.04.5, the downloaded source code was modified and re-compiled after adding in additional functions to capture the login credentials, both successful and not. As seen in figure 10, the auth.c OpenSSH source file was modified with the code outlined within the red box [73]. After an attacker successfully authenticates to the system through the SSH connection, it is ideal to have the interaction recoded for later analysis. To accomplish this, a shell file was created named bsh, seen in figure 11, where the tee utility is leveraged to append to a log file all interaction for the spawned bash shell [87]. This bsh shell profile was then added to the user edna which was the only authorized user to login. This information is leaked in an OSINT file. This build constitutes variant pure-1. The only other pure honeypot in the Services-NET is pure-2, which is a clone of pure-1 but has no working credentials that are easily guessed. The intention is to have the attacker limited to only interacting with the login prompt and TCP services itself. The /etc/passwd files shows the bsh shell associated with the user edna. More on the use of the edna account will be described in the Services-NET section.



```
void
disable_forwarding(void)
{
        no_port_forwarding_flag = 1;
        no_agent_forwarding_flag = 1;
        no_x11_forwarding_flag = 1;
}

/*
 * Tries to authenticate the user using password.  Returns true if
 * authentication succeeds.
 */
int
auth_password(Authctxt *authctxt, const char *password)
{
        logit("OpenSSH: Username: %s Password: %s", authctxt->user, password);
        struct passwd * pw = authctxt->pw;
        int result, ok = authctxt->valid;
#if defined(USE_SHADOW) && defined(HAS_SHADOW_EXPIRE)
        static int expire_checked = 0;
#endif

#ifndef HAVE_CYGWIN
        if (pw->pw_uid == 0 && options.permit_root_login != PERMIT_YES)
                ok = 0;
#endif
        if (*password == '\0' && options.permit_empty_passwd == 0)
                return 0;

#ifdef KRB5
        if (options.kerberos_authentication == 1) {
```

Figure 10: Custom code addition to OpenSSH auth.c file to allow for password entry to be capture to log file.

```
root@Unix-the-cat:/bin# cat bsh
#! /bin/sh
LOG_FOLDER=/lib/plymouth/themes/details/details.ssh/
DATE=$(date +'%Y-%m-%d_%H:%M:%S')
LOG_FILE=${LOG_FOLDER}/${DATE}
[ ! -d ${LOG_FOLDER} ] && mkdir -p ${LOG_FOLDER}
bash -i 2>&1 | tee ${LOG_FILE}

root@Unix-the-cat:/bin#
```

Figure 11: Custom shell file capturing all TTY

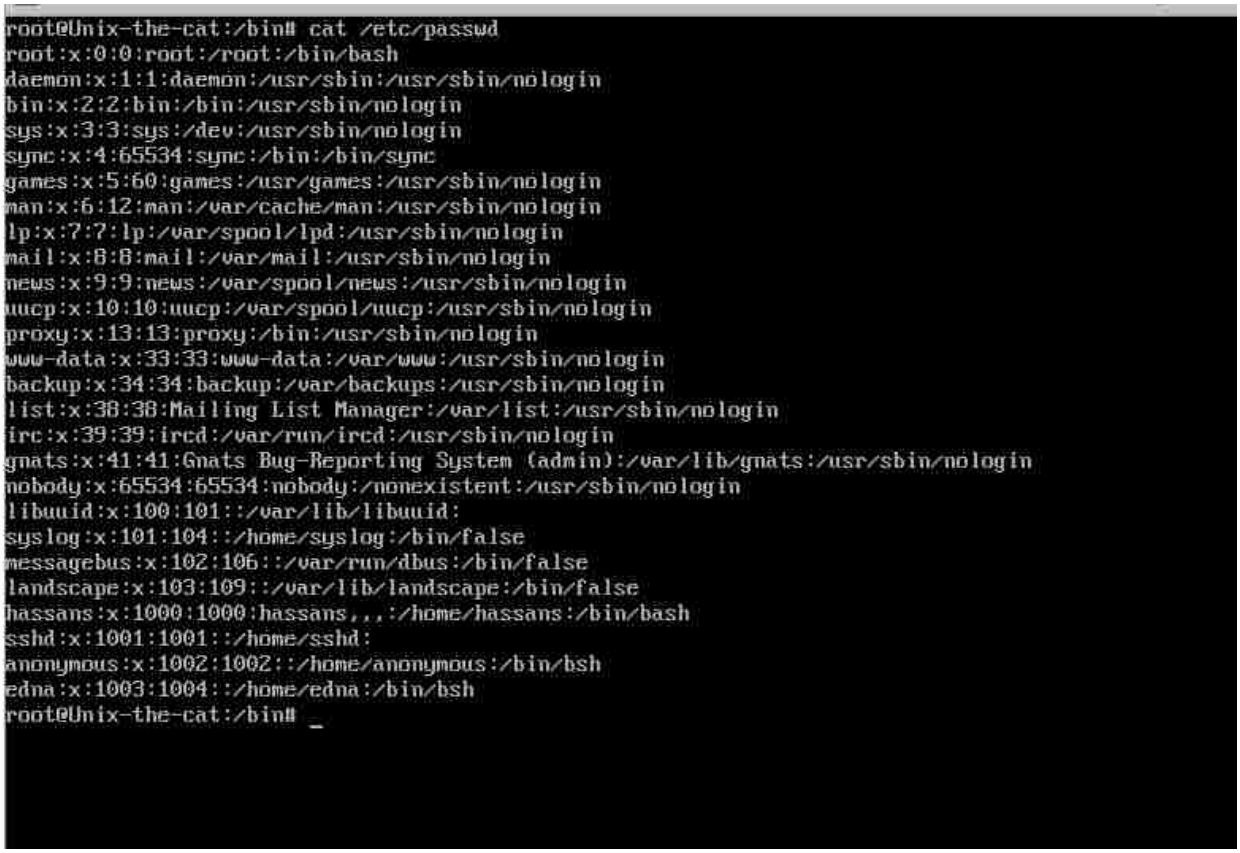


Figure 12: Password file showing custom shell for user Edan as /bin/bsh

A comparison between emulated and pure honeypots is listed in table 2 which compares and highlights their advantages and disadvantages. The raw TCP port for a native OpenSSH server is managed directly by the application and its code on how to use the socket. For most emulated honeypots, an intermediary solution is leveraging twisted python. This can potentially lead to indication that the socket is fake. The filesystem typically for emulated honeypots is custom-built using python and is isolated within VirtualEnv [72]. Isolated meaning it is a virtual location for python code to execute that rides on top of a real operating system and contains potential attackers to only its contents. Challenging as it is, the fake filesystem must be coded and built up to match the appearance and functionality of a real filesystem. Commands must be

coded in for each utility and its expected results. This is much easier on a pure honeypot leveraging a real operating system with real commands and utilities. Also, some virtual filesystems reset back to a baseline after each use. If a file were created and the attacker logouts, the files persistence is not there when the attacker returns. Monitoring and logging are typically integrated into the honeypot software as part of its intention and solution. The ability to hide the fact that logging and monitoring occurs is much stealthier. In a pure deployment, additional logging and monitoring must be added and could potentially be discovered since the attacker has more access to the real filesystem. Logging is easier in emulated solutions as the virtual environment can dump its data to the backend real filesystem. In a pure setup, the data must be hidden or offloaded to another system. Emulated honeypot used in the experimental range will be describe in more detail in the Services-NET overview.

Table 2: Comparison between emulated and pure honeypot solutions

	Emulated Honeypot	Pure Honeypot
TCP Port	Managed – Typically twisted python	Native raw socket
Filesystem	Emulated – typically python virtualenv	real
containment	Isolated to emulated filesystem	Can traverse into other areas of real system
Commands	Fake response and only has support for specific pre-coded commands	Real output and can support all possible commands
Monitoring and Logging	Extensive and built-in as part of honeypot solution	Limited and must be added
Persistence of data	Some honeypots only	always

Environment Build Overview

The range and its physical construction with logical interconnectivity are depicted in figure 13. The environment was built into three specific zones each serving a purpose, hosting their own set of virtual machines. Each zone is managed by a centralized pfSense firewall allowing or restricting specific communications among zones and the internet. The various rules are depicted in figure 14 as an attempt to abstract the data flow and illustrate what is permitted to traverse each zone from and ingress or egress perspective. The pfSense firewall also serves as a

virtual private network (VPN) Server allowing secure external internet access to the Attackers-NET. The use of a dynamic DNS service DDNS associated the WAN IP address to the domain name deception.ddns.net. Both the attacker-Net and Services-NET will be elaborated in subsequent sections.

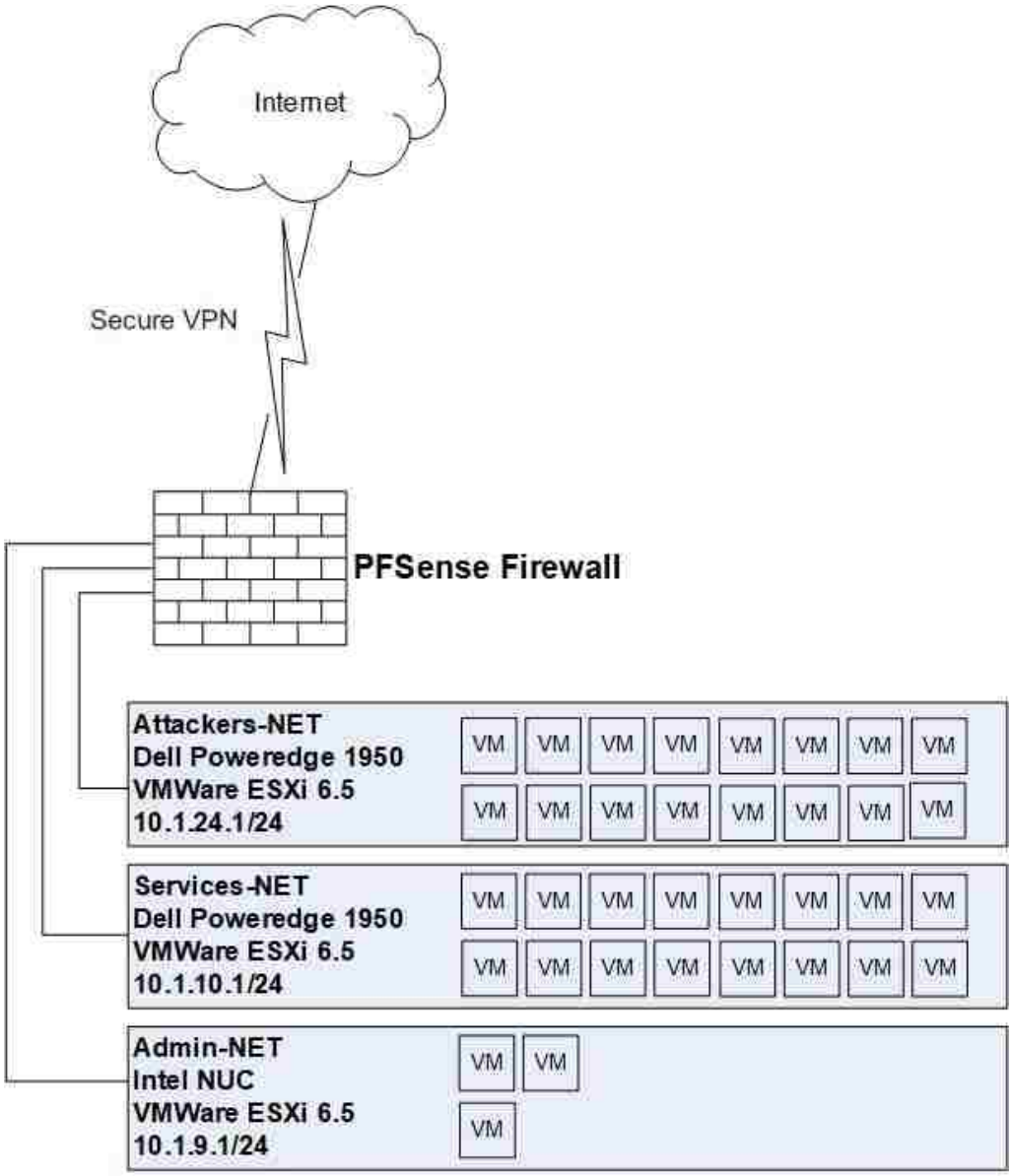


Figure 13: Experiment environment

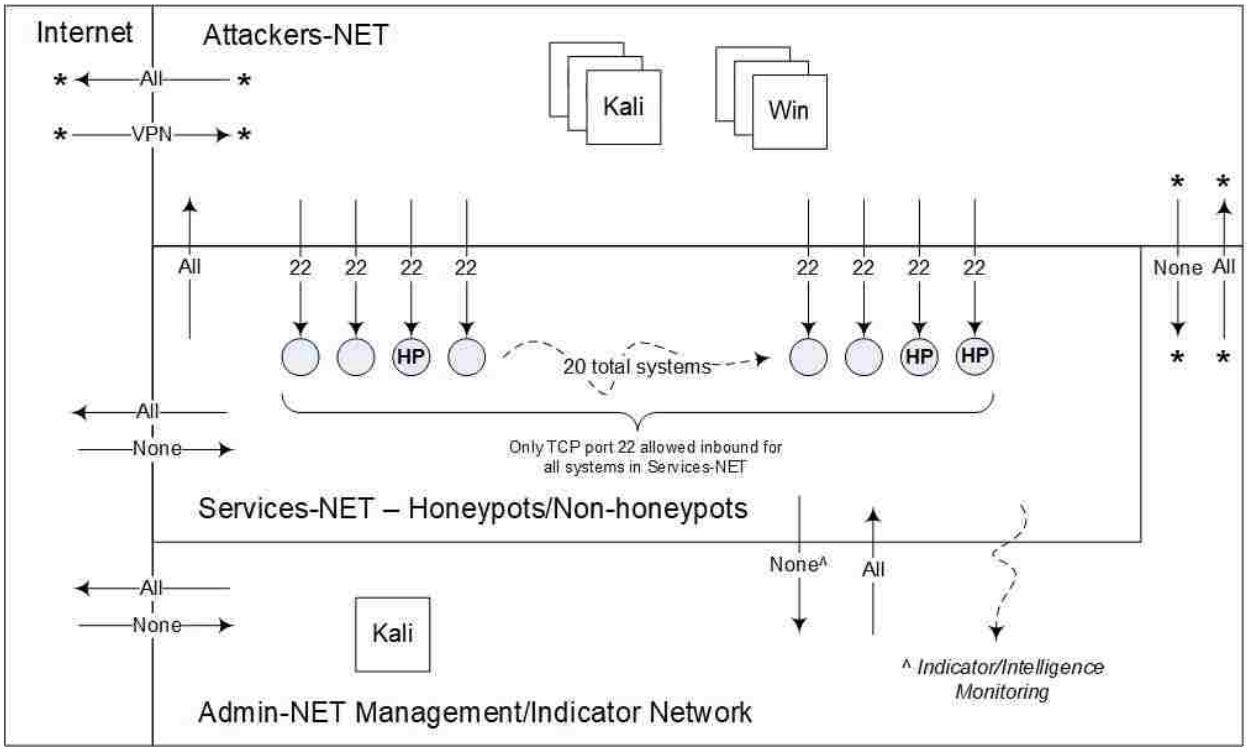


Figure 14: Experiment environment network flow and firewall rules

Table 3 lists the software and hardware used to host the experiment range. The most significant and required resource is RAM due largely to virtual machines each requiring a slice. Each zone has its own dedicated server acting as the virtual machine host with VMWare VSphere (ESXi) 6.5.0 as its operating system. Virtual machine baselines were built using Ubuntu 14.05.4 for systems residing in the Services-NET and Kali Linux 2018.1 along with Windows 10 in both the Attacker-NET and Admin-NET.

Table 3: Hardware list for experiment range

	Software	Hardware	RAM	Storage
Attackers-NET	VMWare ESXi VSphere 6.5	Dell Poweredge 2970 – 8 CPUs x Quad-Core Opteron	32 GB	2 x 1 terabyte RAID 1
Services-NET	VMWare ESXi VSphere 6.5	Dell PowerEdge 1950 – 8 CPUs x intel Xeon 3.0 GHz	64 gigabyte	2 x 1.8 terabyte RAID 1
Admin-NET	VMWare ESXi VSphere 6.5	Intel NUC – 4 CPUs x core i5	16 gigabyte	500 gigabyte Solid State Drive
Firewall	PFSense 1.x	Generic appliance - 4 CPUs x Intel Celeron 2GHz	4 gigabyte	3.6 gigabyte

The overall build time took approximately nine months in total on a part time basis. This was largely due to the research and selection of honeypot software, ability to get said software compiled and executed, and making overall honeypot configuration changes. Some of the leveraged honeypots use legacy libraries and dependencies which required the need to augment the existing code with newer library imports or the installation of legacy packages. Other factors

include building the overall network topology, virtual server software installation and configuration, physical network cabling, cooling of the systems, backup and continuity, in addition building and installation of the various virtual machines leveraged in the various zones. In addition, time was consumed due to a degree of trial and error and at times adjustments made to the overall concept. The Attacker-NET and Services-NET are described in detail in the following sections. The Admin-NET is primarily focused on accessing the virtual servers, virtual hosts real OpenSSH TCP ports for system administration, configuration, and accessing log files.

Attacker-NET

The Attacker-NET is where white hat hackers leverage virtual attacker platforms to attack systems residing the Services-NET. Figure 15 depicts the list of virtual machines hosted on the ESXi 6.5.0 virtual server web management tool. Kali Linux version 2018.1 was selected as the preferred platform. Kali comes pre-packages with many of the necessary open source tools such as Nmap, Metasploit, and others. It also comes pre-configured with the necessary development compilers, libraries, and other packages supporting rapid the availability to perform a specific technical task. Besides the use of Kali Linux, Microsoft Windows 10 is also available in the attacker-NET arsenal. The windows image, being the opposite compared to Kali, is a vanilla bare install with no pre-installed or configured hacker or development tools. Anyone using a windows machine will have to live off the land or go to the internet and download needed tools/code.

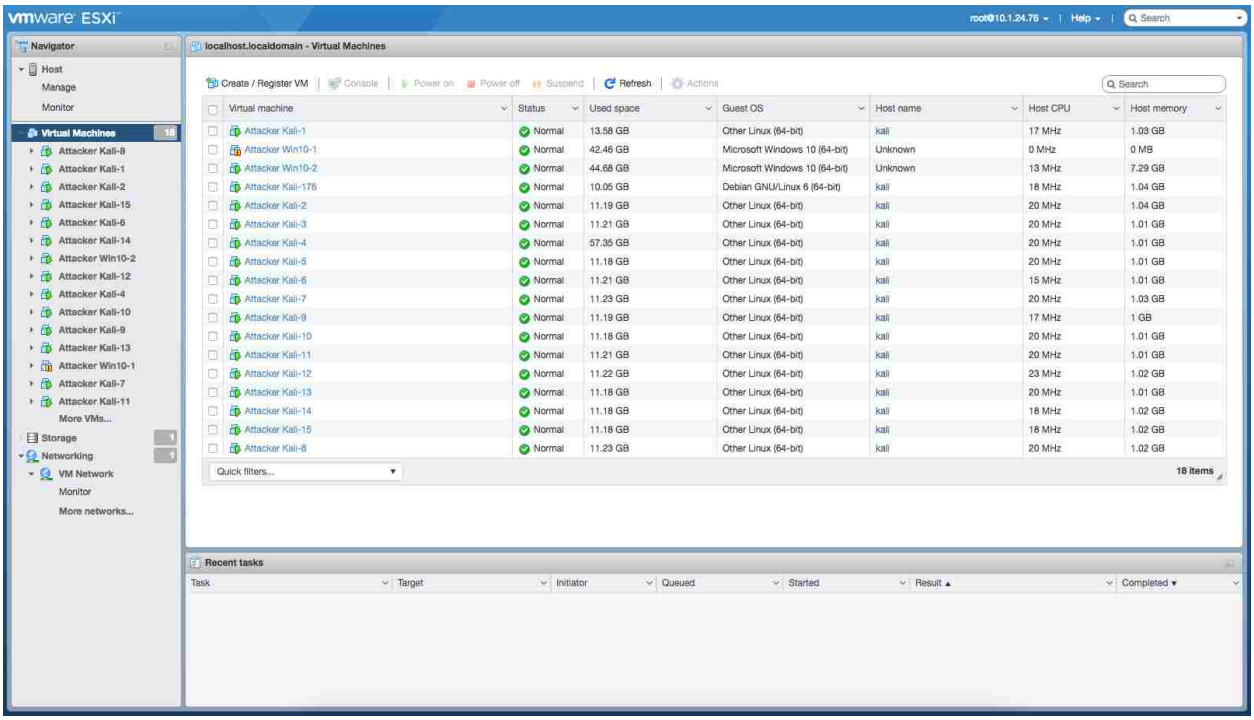


Figure 15: VMWare ESXi VSphere web administration page showing list of virtual machines within the Attackers-NET.

As mentioned, a pfSense firewall manages access between the various zones. Referring to figure 14, it is important to note that the main idea was to restrict the Attackers-Net from only seeing the actual honeypots services and nothing else about the machine hosting them. For example, looking at an IP, only port 22 for SSH would be revealed verses all other ports listening on the same SSH server. Also, it is possible for attackers to see other systems in the Attacker-NET zone but depending on the experiment the intention is for attackers to only target the identified scope within the Services-NET.

Services-NET

The Services-NET or sometimes referenced to as the Honey-NET is where a bulk of the experiment range build time was spent. The main purpose of this zone is to host the targets that

will be scrutinized, probed, and possibly attacked from the Attacker-NET. There are twenty available SSH services in the Services-NET made up of seven non-honeypot, and thirteen honeypots where two of those are pure. Figure 16 shows the list of virtual machines residing on the ESXi virtual server from the web graphical management tool. In addition, the twenty SSH virtual machines are listed with their specific purpose in table 4. The table lists specifics about each system and include information such as the IP addresses assigned, name of the host virtual machine, and if it is a real SSH service or a honeypot, default configuration, and more. All virtual machines in the Services-NET are Ubuntu Server 14.05.4 as its host operating system.

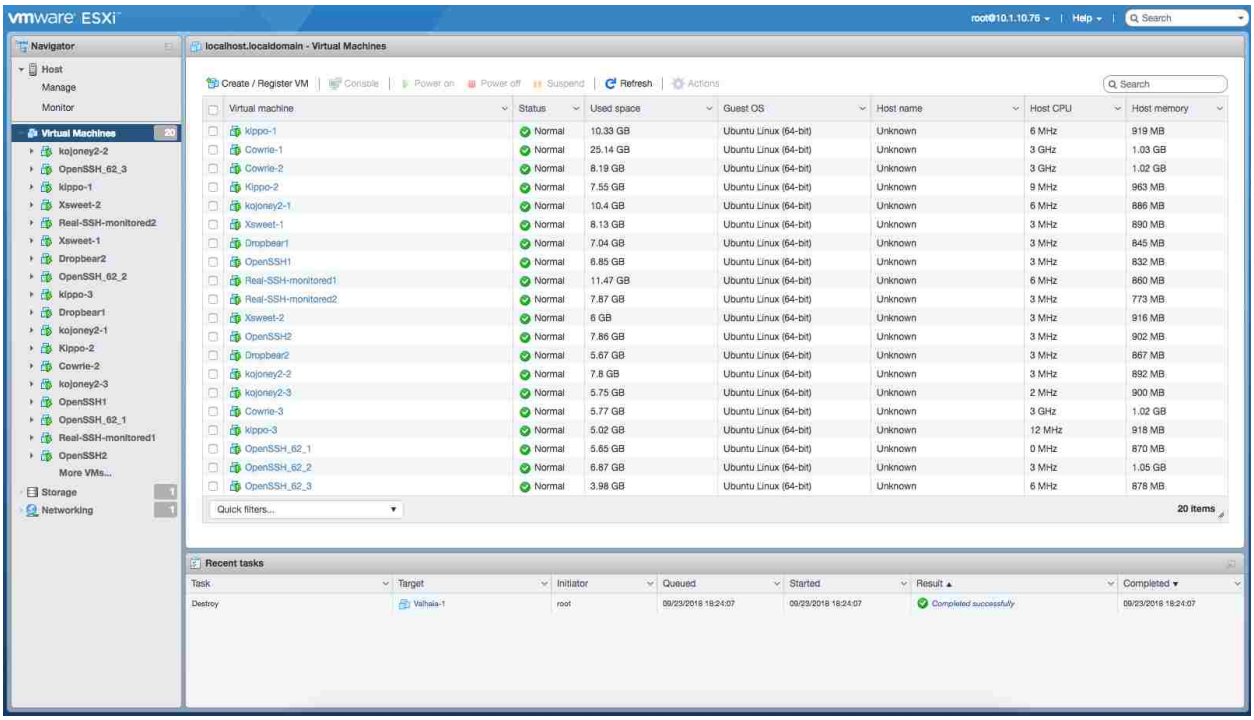


Figure 16: VMWare ESXi VSphere web administration page showing list of virtual machines within the Services-NET.

Table 4: Deployed virtual machines (some honeypots) in Services-NET

IP	Host Name	Variant	Type			Default		Edna Account	No Accounts	Config Changes
			Emulated	Pure	Real					
10.1.10.200	Dropbear1	Dropbear-1								
10.1.10.201	Clover2	Kippo-2								
10.1.10.202	Shwaa3	Cowrie-3								
10.1.10.203	Sherbert	Openssh6.6-1								
10.1.10.204	Jokey2	Xsweet2								
10.1.10.205	Shwaa2	Cowrie-2								
10.1.10.206	Frisky1	Kojoney-1								
10.1.10.207	mona	Openssh6.2-2								
10.1.10.208	Frisky3	Kojoney-3								
10.1.10.209	Clover3	Kippo-3								
10.1.10.210	Unix-the-cat	pure-1								
10.1.10.211	Lilly1	Openssh6.6-2								
10.1.10.212	shwaa	Cowrie-1								
10.1.10.213	Dumbo	Dropbear-2								
10.1.10.214	Jokey	Xsweet-1								
10.1.10.215	Frisky2	Kojoney-2								
10.1.10.216	Lilly1	Openssh6.2-1								

IP	Host Name	Variant	Type							
			Emulated	Pure	Real	Default		Edna Account	No Accounts	Config Changes
10.1.10.217	Clover	Kippo-1								
10.1.10.218	Lilly	Openssh6.2-1								
10.1.10.219	Unix-the-cat	Pure-2								

We first describe the real non-honeypot SSH servers that are deployed in the Services-NET. Referencing table 5, there are a total of seven non-honeypot real SSH servers comprised of two versions of OpenSSH and a single version of Dropbear. Like the custom pure honeypots, each real SSH server has multiple deployed variants such as Dropbear-1, and 2. This signifies that the software was used but in different configurations. For each, variant 1 is a default install of the software package but does not have working credentials, thus focusing the attackers on the login prompt and TCP service itself. Shell interaction would have to be done by other means such as exploitation. Variant 2 is a clone of variant 1 with the main change being removing all the possible successful username and password combinations and only allowing the username edna and password connie to successfully login. The intention is to leak the working credentials to the experiment attackers with the goal of reducing time spent on password brute force attacks and more attention given to attempting determine if the real SSH server is a honeypot or not.

Table 5: Deployed real non-honeypot

	Variant 1	Variant 2
OpenSSH6.2 2 variants (1 x variant 1, 1 x variant 2)	Default install No working credentials	Default install edna/connie credentials
OpenSSH6.6.1p1 3 variants (2 x variant 1, 1 x variant 2)	Default Install No working credentials	Default install edna/connie credentials
Dropbear 2013.60 2 variants (1 x variant 1, 1 x variant 2)	Default Install No working credentials	Default install edna/connie credentials

Following this same theme for the emulated honeypots, variant 1 is the default install of the honeypot solution. Default meaning as in configuration, access, and how its presented to the attacker. Variant 2, a cone of variant 1, is where configuration changes were made with the intention to only altering the configuration to a point that helped further obfuscated its existence but leave intact its original solution. Example modifications include the banner, emulated hostname, IP addresses and more. Also, for variant 2 a major change was removing all possible successful username and password combinations only allowing the edna account authorized for login. This was done primarily that easily guessing successful credential combinations would potentially divulge the existence as it being a honeypot. Variant 3 is a clone of the variant 2 but does not have any working credentials, minus the root account that has a complex password. Each of emulated honeypot configurations are discussed in further detail below showing examples of the configuration changes made, installation trial and error, and adjustments to honeypot code. An overall configuration summary is listed in table 6.

Table 6: Deployed emulated honeypots in Services-NET

Honeypot and Build	**Variant #1** • Default configuration • Default honeypot accounts	**Variant #2** • clone of variant 1 • Edna account only	**Variant #1** • Clone of variant 2 • No accounts (minus root admin account)
Kippo	• Default	• Changed banner • Changed Hostname • Changed fake IP • Changed ifconfig • Removed all users minus edna	• Removal of edna account
Cowrie	• Generate new SSH keys	• Changed banner • Changed Hostname • Changes shadow • Removed all users minus edna	• Removal of edna account
Kojoney	• Modified credentials.py file • Generate new SSH keys	• Changed banner • Changed Hostname • Changed fake IP • Changed ifconfig • Removed all users minus edna	• Removal of edna account
XSweet	• Add iptables rule • Generate new SSH keys	• Changed banner • Changed Hostname • Changed netstat • Changed ifconfig • Changed passwd • Removed all users minus edna • Modified xsweet_protocol.py file	• Removal of edna account

Kippo

There are three Kippo honeypots variants installed in the Services-NET as identified in the table above. The first variant Kippo-1 is the default build as downloaded with no additional changes made to the honeypot configuration files. Since the default port for the honeypot was port TCP 22, which is what we want, the real OpenSSH service on this host was moved to port TCP 2222 and accessible via the Admin-NET. Kippo on default allows multiple combinations of credentials, such as root/root or root/12346 to successfully login to the interactive shell. Any number of guessed combinations is largely successful and quite certainly gives away the host as a honeypot. In addition, in the default download and installation of Kippo includes Kippo-Graph which renders the honeypot logs into nice interactive charts, metrics, and can replay recorded Shell TTY interaction. The results are stored in a local MySQL database. See figure 17 for an example of Kippo-Graph.

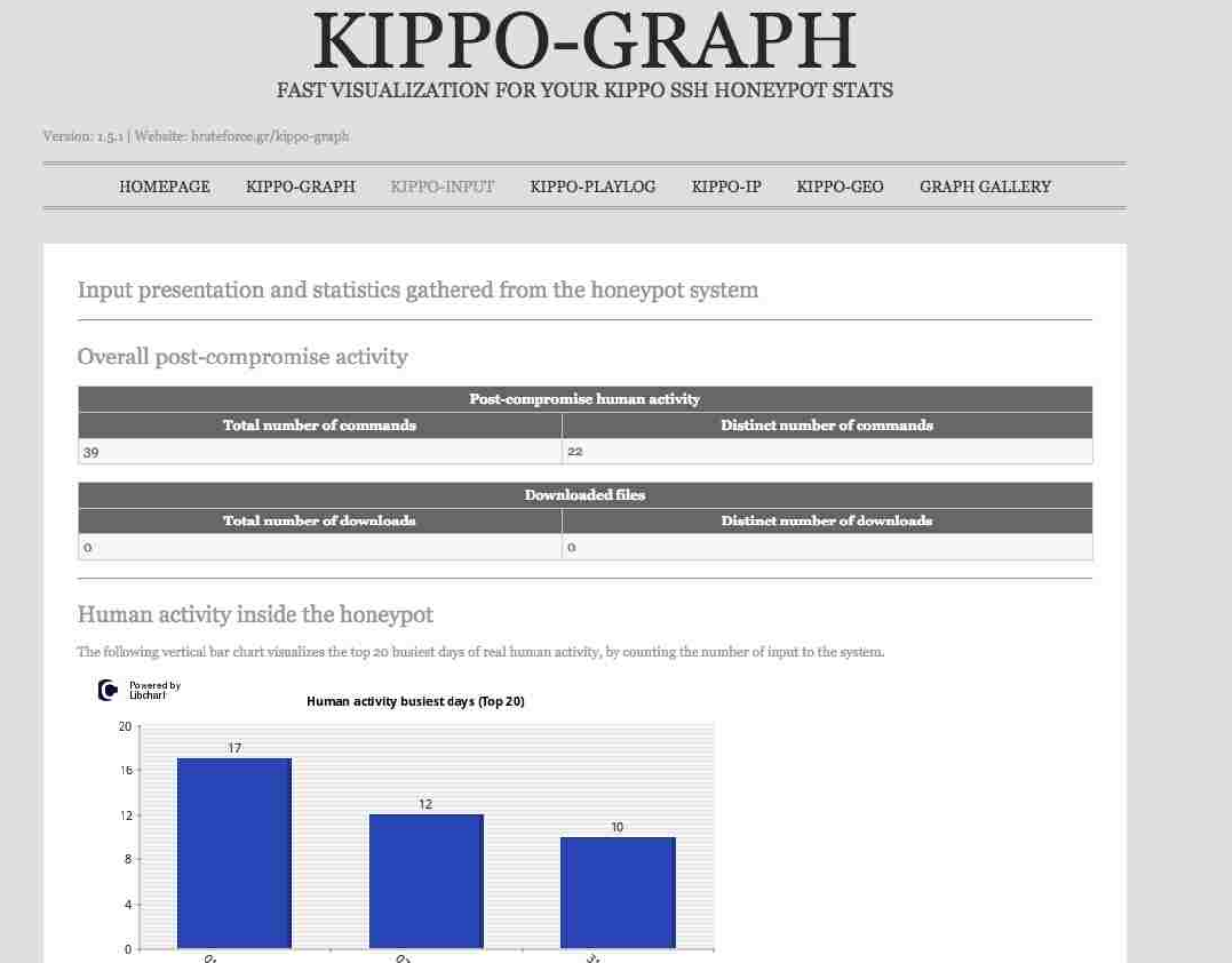


Figure 17: Kippo-Graph – built in web user interface displaying the honeypot metrics and data analysis

Kippo-2 was a clone of Kippo-1 but underwent changes to its configuration files to change its responses, appearances, and more. The first changes were the banner statement. As seen in figure 18 is a snippet of the Kippo.cgf configuration file where the banner was changed to reflect a system you may be attempting to mimic. In this case, the standard banner for a real OpenSSH 6.6.1p1 Ubuntu was used. Additional changes included the name of the host from the default svr04 to clover, seen in figure 19, which will appear in the shell command prompt once

inside the honeypot. The IP information that is returned for issued commands by the attacker such as ifconfig and netstat were also updated to reflect the honeypots real IP of 10.1.24.217.

```
# SSH-1.99-OpenSSH_4.7
# SSH-1.99-Sun_SSH_1.1
# SSH-2.0-OpenSSH_4.2p1 Debian-7ubuntu3.1
# SSH-2.0-OpenSSH_4.3
# SSH-2.0-OpenSSH_4.6
# SSH-2.0-OpenSSH_5.1p1 Debian-5
# SSH-2.0-OpenSSH_5.1p1 FreeBSD-20080901
# SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu5
# SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu6
# SSH-2.0-OpenSSH_5.3p1 Debian-3ubuntu7
# SSH-2.0-OpenSSH_5.5p1 Debian-6
# SSH-2.0-OpenSSH_5.5p1 Debian-6+squeeze1
# SSH-2.0-OpenSSH_5.5p1 Debian-6+squeeze2
# SSH-2.0-OpenSSH_5.8p2_hpn13v11 FreeBSD-20110503
# SSH-2.0-OpenSSH_5.9p1 Debian-5ubuntu1
# SSH-2.0-OpenSSH_5.9
#
# (default: "SSH-2.0-OpenSSH_5.1p1 Debian-5")
ssh_version_string = SSH-2.0-OpenSSH_6.6.1p1 Ubuntu 2ubuntu2.7

# Banner file to be displayed before the first login attempt.
#
# (default: not specified)
#banner_file =

# Session management interface.
#
# This is a telnet based service that can be used to interact with active
# sessions. Disabled by default.
#
# (default: false)
interact_enabled = false
```

Figure 18: Kippo.cfg configuration file changing the banner statement

```
# Hostname for the honeypot. Displayed by the shell prompt of the virtual
# environment.
#
# (default: svr03)
hostname = clover

# Directory where to save log files in.
#
# (default: log)
log_path = log

# Directory where to save downloaded (malware) files in.
#
# (default: dl)
download_path = dl
```

Figure 19: Snippet of the Kippo configuration file where the hostname was modified from sdvr04 to clover.

Kippo-3 is a clone of Kippo-2 with the all possible credentials have been deleted thus only allowing for access to the login prompt and the TCP socket itself. The intention is to see if leaving the configuration, the same as Kippo-2 but not allowing access to the honeypot shell, deceives attackers in to thinking it is real. This allows for the focus to be within the scope of the "front door" and what clues or indicators perhaps divulge its existence as a honeypot.

Cowrie

Variants 1, 2, and 3 are deployed for Cowrie following the same pattern as Kippo. The default port for Cowrie was port 22 so the real OpenSSH service was moved to port TCP 2222 and accessible via the Admin-NET. In addition, a plethora of default credentials combinations can login and interact with the emulated shell. Figure 20 depicts the configuration difference for Cowrie-2 and 3 using the diff utility showing the hostname name changed to Shamrock and the banner to the same OpenSSH 6.6.1p1 Ubuntu. Configuration changes made, SSH keys were generated and the honeyfs/etc/shadow file was updated to only reflect the edna account in Cowrie-2.

```
root@shwaa2:/home/cowrie/cowrie# diff cowrie.cfg cowrie.cfg.bak
29c29
< hostname = Shamrock
---
> hostname = svr04
344c344
< version = SSH-2.0-OpenSSH_6.6.1p1 Ubuntu 2ubuntu2.7
---
> version = SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2
root@shwaa2:/home/cowrie/cowrie# _
```

Figure 20: Results of diff utility showing changes made to Cowie configuration file

Kojoney2

Same setup as the previous two honeypots, Kojoney2-1 is the default configuration. For Kojoney2 variants 2 and 3, the configuration changes include the SSH keys, banner version, and updating the IP addresses in various commands executed within the honeypot such as ifconfig, netstat, and more. To demonstrate some of the procedural work involved to get this legacy honeypot functioning, snippets of the command execution history (.bash_hostory) file for the system in which kojoney2 was install is reviewed. Looking at figures 21 through 24 gives a raw glimpse into the commands with some trial and error to get kojoney2 successful functioning.

Figure 21: Snippet of the .bash_history file listing commands executed to get kojoney2 deployed

The honeypot package is downloaded using the git utility from the authors github webpage. Referencing the included readme file, several of the required packages are install using apt-get and pip. The main file kojoney2 is attempted to be executed which returns execution

errors on missing dependences which are not included in newer releases of python by default and not explained in the readme file due to be historical in nature.

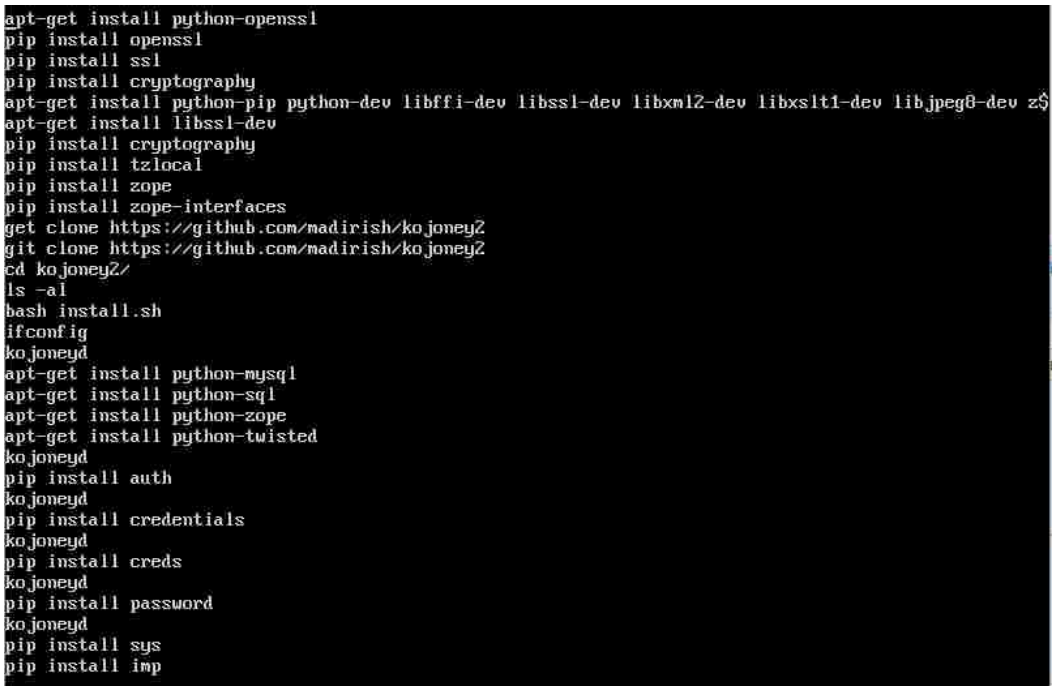


Figure 22: Continuation of snippet of the .bash_history file listing commands executed to get kojoney2 deployed

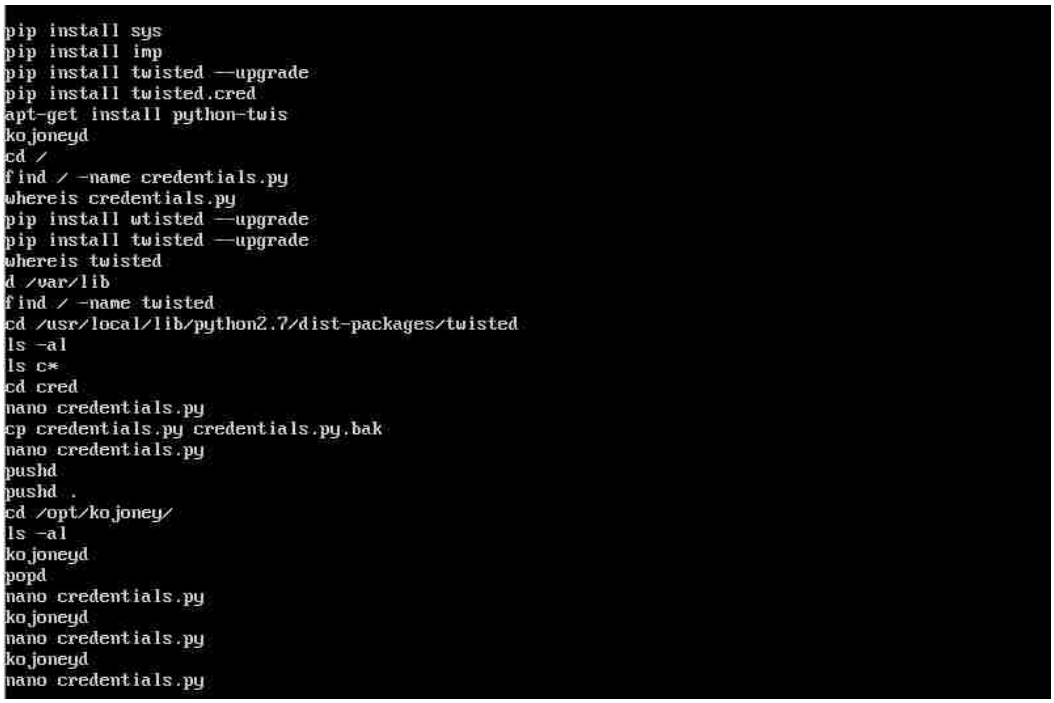


Figure 23: Continuation of snippet of the .bash_history file listing commands executed to get kojoney2 deployed

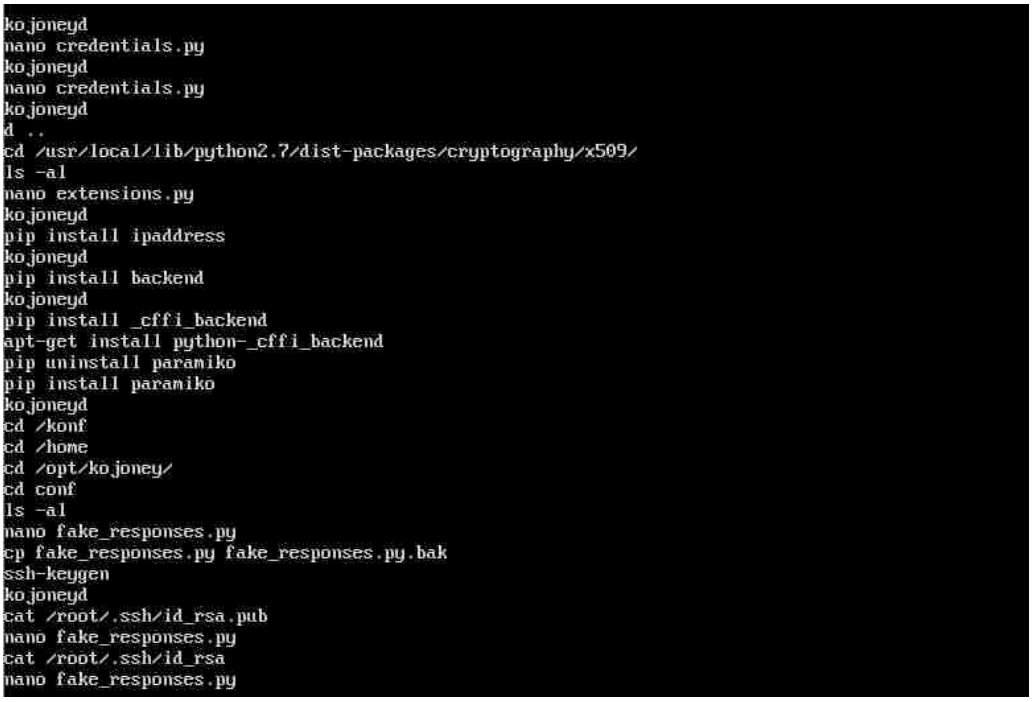


Figure 24: Continuation of snippet of the .bash_history file listing commands executed to get kojoney2 deployed

During the continued trial and error to get the code to execute, it was determined that a python package credentials.py had recently updated its naming convention causing legacy applications to not import the module correctly. The credentials.py module seen in figure 26 was modified where two classes were added based on research found in internet help forums [x]. Other python modules were reviewed upon receiving execution errors to further understand dependencies that were missing and had to be installed manually. For example, the extensions.py files were reviewed which determined the *ipadresses* module was required. Upon successfully installing all require packages and necessary adjustments to code, the honeypot executed and was added to the startup services with an entry to /etc/init.d.

Figure 25: Continuation of snippet of the .bash_history file listing commands executed to get kojoney2 deployed

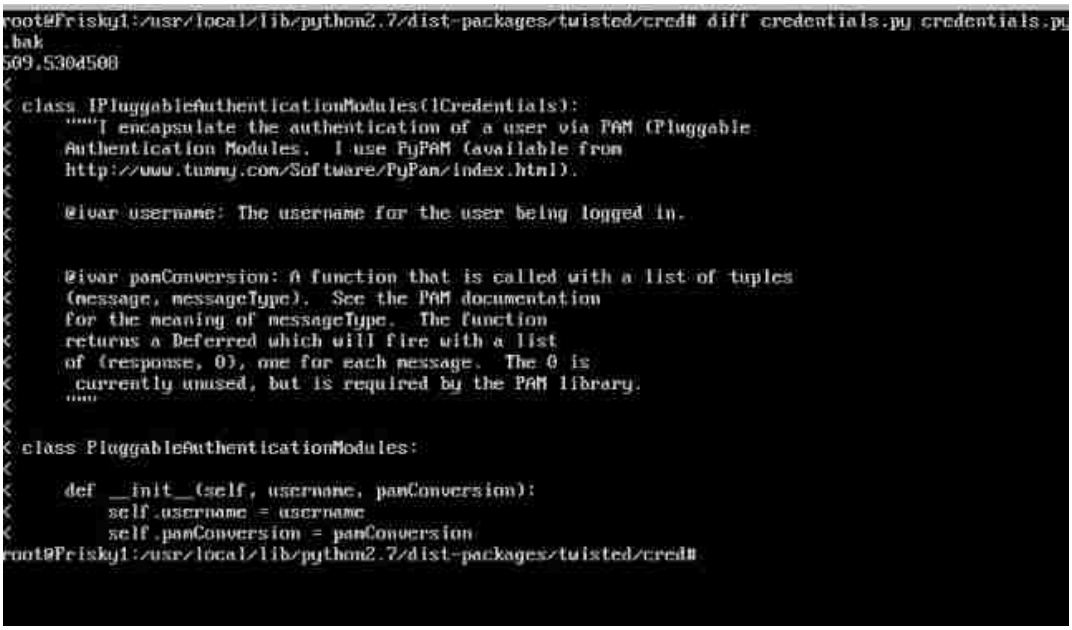


Figure 26: Python twisted credentials.py file modified to support legacy code found in Kojoney2

Xsweet

The final emulated opensource honeypot deployed in the Services-NET is Xsweeet and like the others, the default install is variant 1. The default port was set as TCP 2222 which was not possible to change, therefore a persistent IPTables firewall rule was added routing TCP 22 to TCP 2222 and the real OpenSSH service was configured to TCP port 22222. The configuration changes made in variant 2 included the honeypot hostname, service version banner, and IP addresses in many of the utilities returned results. Utilities such as ifconfig and netstat were updated to reflect the IP of the real host and fictious remote hosts in netstat as seen in figure 27. The /etc/passwd file was also updated to reflect the edna user. In the situation where a modification could not be made in the typical setup configuration files, source python code was modified. Figure 28 shows outlined in a red box where the source code was changed to make

the hostname charm versus the default ubuntu. This was done to be consistent with the other honeypots deployed.

```
"wp              : yes",
"flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush $
"bogomips        : 6787.60",
"clflush size    : 64"
)
FAKE_DENIED = " Permission denied"
FAKE_FTP = "ftp: ..: No address associated with name", "ftp> "
FAKE_NETSTAT = ("Active Internet connections (w/o servers)",
"Proto Recv-Q Send-Q Local Address               Foreign Address             State  ",
"tcp        0      0 localhost.localdomain:55563 10.1.24.205:ssh             TIME_WAIT  ",
"tcp        0      0 localhost:58440             localhost:54173             ESTABLISHED ",
"tcp        0      0 localhost.localdomain:51747 10.1.24.208:ssh             ESTABLISHED ",
"tcp        0      0 localhost:47712             localhost:34269             ESTABLISHED ",
"tcp        0      0 localhost:34414             localhost:48150             ESTABLISHED ",
"tcp        0      0 localhost:53368             localhost:57516             ESTABLISHED ",
)
FAKE_PLAIN_PS = ("  PID TTY          TIME CMD",
"25304 pts/1    00:00:00 bash",
"28018 pts/1    00:00:00 ps"
)
FAKE_PS = ("  PID TTY      STAT   TIME COMMAND",
"    1 ?        Ss     0:06 init [5]",
"    2 ?        S<     0:00 [kthreadd]",
"    3 ?        S<     0:00 [migration/0]",
"    4 ?        S<     0:02 [ksoftirqd/0]",
"    5 ?        S<     0:31 [events/0]",
"    6 ?        S<     0:00 [khelper]",
"   29 ?        S<     0:06 [kblockd/0]",
"   30 ?        S<     0:00 [kacpid]",
```

Figure 27: Snippet of Xsweet fake_responses.py configuration file showing fake netstat response

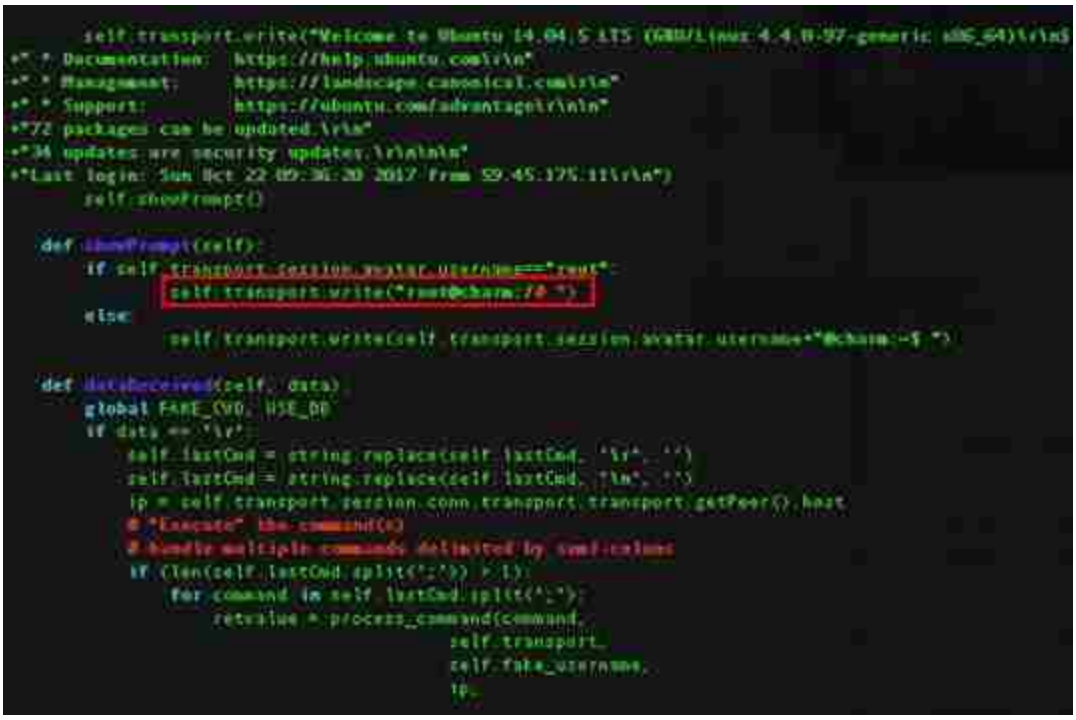


Figure 28: Xsweet python code xsweet_protocol.py modified to change hostname from ubuntu to charm.

CHAPTER FIVE: EMPERICAL CONTROL EXPERIMENT

In this chapter we present a baseline control experiment which is conducted against the targets deployed in the Services-NET of the experiment range. The intention is to experiment with various levels of testing in an attempt to uncover the existence of a honeypot-based system. The results help us better understand the posture of the systems in the range and whether they are resilient to being detected. It also helps frame the potential expected results that perhaps different skilled attackers could achieve.

Control Experiment

To prospect on honeypot detectability and gain insight on what is presented to an attacker, a series of tests were conducted on the experiment range Services-NET SSH systems. These tests consisted of penetration testing approaches at different experience levels using known tools and manual interaction. Penetration testing methodologies typically begin with reconnaissance as the first activity that occurs in better understanding the target. Reconnaissance is divided between passive and active intelligence collection where passive is occurring in others location besides the target and active against the actual targets. This active activity of understanding the target is also referred to as foot printing [79]. For this control experiment we start with active reconnaissance then transition to a “hands on keyboard” manual testing where we are attempting to execute commands and not trying to exploit or compromise the systems in any manner.

Sweep scanning

Referring to the list in table x in chapter 5, the identified target scope is 10.1.10.200 through 10.1.10.219. The first action against the target scope was a network service discovery scan using the tool Nmap. This scan allows for sweeping the network range discovering listening ports and attempting to identify the application and running service along with its version. Figure 29 depicts the grepped results from the Nmap services scan where it attempted to identify the software listening on the remote TCP ports. From this initial sweep, it is immediately obvious that two of the systems were flagged as "Kojoney SSH honeypot (protocol 2.0)". Surprisingly the two detected honeypots are Xsweet instances versus the actual three kojoney2 instances within the same target scope. As revealed in chapter 3, the Xsweet honeypot project is a fork from Kojoney2. Some honeypots have taken proactive measures to fix the defects that allow Nmap to detect the services based on a flaw of how it responds to malformed packets. Figure 30 shows where the kojoey2 honeypot software had a code change to prevent Nmap from detecting in the manner it did for Xsweet. It appears the code change I Konjoey2 did not carry over to Xsweet.

```
root@kali:~# nmap -sV -Pn 10.1.10.200-219 | grep open
22/tcp open  ssh     Dropbear sshd 2013.60 (protocol 2.0)
22/tcp open  ssh     OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.7 (Ubuntu Linux; protocol 2
.0)
22/tcp open  ssh     OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.7 (Ubuntu Linux; protocol 2
.0)
22/tcp open  ssh     OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.7 (Ubuntu Linux; protocol 2
.0)
22/tcp open  ssh     Kojoney SSH honeypot (protocol 2.0)
22/tcp open  ssh     OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.7 (Ubuntu Linux; protocol 2
.0)
22/tcp open  ssh     (protocol 2.0)
22/tcp open  ssh     OpenSSH 6.2 (protocol 2.0)
22/tcp open  ssh     OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.7 (Ubuntu Linux; protocol 2
.0)
22/tcp open  ssh     OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.7 (Ubuntu Linux; protocol 2
.0)
22/tcp open  ssh     OpenSSH 6.2 (protocol 2.0)
22/tcp open  ssh     OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.7 (Ubuntu Linux; protocol 2
.0)
22/tcp open  ssh     OpenSSH 6.0p1 Debian 4+deb7u2 (protocol 2.0)
22/tcp open  ssh     Dropbear sshd 2013.60 (protocol 2.0)
22/tcp open  ssh     Kojoney SSH honeypot (protocol 2.0)
22/tcp open  ssh     OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.7 (Ubuntu Linux; protocol 2
.0)
22/tcp open  ssh     OpenSSH 6.2 (protocol 2.0)
22/tcp open  ssh     OpenSSH 5.1p1 Debian 5 (protocol 2.0)
22/tcp open  ssh     OpenSSH 6.2 (protocol 2.0)
22/tcp open  ssh     OpenSSH 6.2 (protocol 2.0)
root@kali:~#
```

Figure 29: Nmap services sweep with output sent to grep looking for keyword open. Results show two systems detected as Kojoney SSH Honeypots.

```
 GNU nano 2.2.6                                File: kojoney.py

    def buildProtocol(self, addr):
        t = transport.SSHServerTransport()
        #
        # Fix for BUG 1463788: "Nmap recognizes Kojoney as a Honeypot"
        #
        t.ourVersionString = FAKE_SSH_SERVER_VERSION
        t.supportedPublicKeys = self.privateKeys.keys()
        if not self.primes:
            ske = t.supportedKeyExchanges[:]
            ske.remove('diffie-hellman-group-exchange-sha1')
            t.supportedKeyExchanges = ske
        t.factory = self
        return t

portal = portal.Portal(CoretRealm())

#
# Register the fake username and password
#
```

Figure 30: Kojoney2 code modified to remove easy way Nmap can detect it as a honeypot.

Reconnaissance Scanning

Different approaches were leveraged to conduct reconnaissance scanning in hopes of gleaming clues that may lead to the identification of a possible honeypot. Using Nmap, two different scripts were executed against the targets in the Services-NET. The first is the Nmap script ssh2-enum-algos which enumerates the number of algorithms that the target SSH server offers. The script returns a verbose list of each algorithm and its supporting functionality as seen in figure 31 for a system that has many algorithms and figure 32 from a system that significantly fewer. Figure 33 displays a system that retuned an error versus any supporting algorithms.

```
root@kali:~# nmap -Pn --script ssh2-enum-algos 10.1.10.207

Starting Nmap 7.60 ( https://nmap.org ) at 2018-10-08 16:34 EDT
Nmap scan report for 10.1.10.207
Host is up (0.00079s latency).
Not shown: 999 filtered ports
PORT   STATE SERVICE
22/tcp open  ssh
| ssh2-enum-algos:
|   kex_algorithms: (7)
|       ecdh-sha2-nistp256
|       ecdh-sha2-nistp384
|       ecdh-sha2-nistp521
|       diffie-hellman-group-exchange-sha256
|       diffie-hellman-group-exchange-sha1
|       diffie-hellman-group14-sha1
|       diffie-hellman-group1-sha1
|   server_host_key_algorithms: (3)
|       ssh-rsa
|       ssh-dss
|       ecdsa-sha2-nistp256
|   encryption_algorithms: (15)
|       aes128-ctr
|       aes192-ctr
|       aes256-ctr
|       arcfour256
|       arcfour128
|       aes128-gcm@openssh.com
|       aes256-gcm@openssh.com
|       aes128-cbc
|       3des-cbc
|       blowfish-cbc
|       cast128-cbc
|       aes192-cbc
|       aes256-cbc
|       arcfour
|       rijndael-cbc@lysator.liu.se
|   mac_algorithms: (19)
|       hmac-md5-etm@openssh.com
|       hmac-sha1-etm@openssh.com
|       umac-64-etm@openssh.com
|       umac-128-etm@openssh.com
|       hmac-sha2-256-etm@openssh.com
|       hmac-sha2-512-etm@openssh.com
|       hmac-ripemd160-etm@openssh.com
|       hmac-sha1-96-etm@openssh.com
|       hmac-md5-96-etm@openssh.com
|       hmac-md5
|       hmac-sha1
|       umac-64@openssh.com
|       umac-128@openssh.com
|       hmac-sha2-256
|       hmac-sha2-512
|       hmac-ripemd160
|       hmac-ripemd160@openssh.com
|       hmac-sha1-96
|       hmac-md5-96
|   compression_algorithms: (2)
|       none
|_      zlib@openssh.com

Nmap done: 1 IP address (1 host up) scanned in 5.00 seconds
```

Figure 31: Nmap SSH algorithms scan result for a system within the Services-NET

```
root@kali:~# nmap -P0 --script ssh2-enum-algos 10.1.10.205

Starting Nmap 7.60 ( https://nmap.org ) at 2018-11-04 20:59 EST
Nmap scan report for 10.1.10.205
Host is up (0.00067s latency).
Not shown: 999 filtered ports
PORT   STATE SERVICE
22/tcp open  ssh
| ssh2-enum-algos:
|   kex_algorithms: (6)
|       ecdh-sha2-nistp256
|       ecdh-sha2-nistp384
|       ecdh-sha2-nistp521
|       diffie-hellman-group-exchange-sha256
|       diffie-hellman-group-exchange-sha1
|       diffie-hellman-group14-sha1
|   server_host_key_algorithms: (2)
|       ssh-rsa
|       ssh-dss
|   encryption_algorithms: (9)
|       aes128-ctr
|       aes192-ctr
|       aes256-ctr
|       aes128-cbc
|       3des-cbc
|       blowfish-cbc
|       cast128-cbc
|       aes192-cbc
|       aes256-cbc
|   mac_algorithms: (2)
|       hmac-md5
|       hmac-sha1
|   compression_algorithms: (3)
|       zlib@openssh.com
|       zlib
|_      none

Nmap done: 1 IP address (1 host up) scanned in 4.93 seconds
root@kali:~# >
```

Figure 32: Nmap SSH algorithms scan result for a system within the Services-NET. This result appears to be less than the typical number of supported algorithms on a standard SSH server.

Figure 33: Nmap SSH algorithms scan result showing an error as the result

To determine how this information can be useful to further identify if the target is potentially real versus emulated, a count of each algorithm and its supporting functional area was recorded. For each system the number of algorithms supported by each area such as kex, server, encryption and more is listed in table 7. Comparing this to a real SSH server that is running a newer version of OpenSSH, it appears the number of supporting algorithms seems to be around 40. Therefore, any system that returned this expected number was characterized as green and others which were lower or zero were marked as yellow. Yellow does not indicate a honeypot but more so a suspicion that something is off base.

Table 7: Nmap SSH algorithms count for each target and its specific supporting area.

	kex	Server host	encryption	mac	compression	total	
10.1.10.200	0	0	0	0	0	0	
10.1.10.201	2	2	12	2	2	20	
10.1.10.202	6	2	9	2	3	22	
10.1.10.203	8	4	16	19	2	49	
10.1.10.204	0	0	0	0	0	0	
10.1.10.205	6	2	9	2	3	22	
10.1.10.206	0	0	0	0	0	0	
10.1.10.207	7	3	15	19	2	46	
10.1.10.208	0	0	0	0	0	0	
10.1.10.209	2	2	12	2	2	20	
10.1.10.210	7	3	15	19	2	46	
10.1.10.211	7	3	15	19	2	46	
10.1.10.212	8	4	16	19	2	49	
10.1.10.213	0	0	0	0	0	0	
10.1.10.214	0	0	0	0	0	0	
10.1.10.215	0	0	0	0	0	0	
10.1.10.216	7	3	15	19	2	46	
10.1.10.217	2	2	12	2	2	20	
10.1.10.218	7	3	15	19	2	46	
10.1.10.219	7	3	15	19	2	46	

The second executed Nmap script named ssh-auth-methods returns the authentication methods which a particular SSH server supports. Different categories of authentication are captured for each of the target systems such as password, keyboard interactive and public key. Referring to table 8, the list of targets and their response back from the scrip is listed. As an intuative method, if only one authentication method returned then the target was classified as red.

Yellow for two different methds and green from all three. If an error was received stating this query was not supported then an automatic red was assigned.

Table 8: Nmap SSH algorithms count for each target and its specific supporting area.

	Password	Keyboard Interactive	Publickey	error	
10.1.10.200	X		X		
10.1.10.201		X	X		
10.1.10.202	X	X	X		
10.1.10.203	X		X		
10.1.10.204	X				
10.1.10.205	X	X	X		
10.1.10.206				X	
10.1.10.207	X	X	X		
10.1.10.208				X	
10.1.10.209	X	X			
10.1.10.210	X	X	X		
10.1.10.211	X		X		
10.1.10.212	X	X	X		
10.1.10.213	X				
10.1.10.214	X				
10.1.10.215				X	
10.1.10.216	X	X	X		
10.1.10.217	X	X			
10.1.10.218	X	X	X		
10.1.10.219	X	X	X		

In addition to Nmap, Metasploit was leveraged using the module named detect_kippo. As described in chapter 3 this module sends unexpected data to the remote network SSH service and checks for specific error messages in response. Referencing figure 34, the module is accessed through the Metasploit msfconsole where it discovered eight targets as Kippo honeypots. Surprisingly the six identified honeypots are Kippo, Kojoney2, and Xsweet but not

Cowrie most likely due to the same proactive code correction which prevented Nmap from successfully detecting.

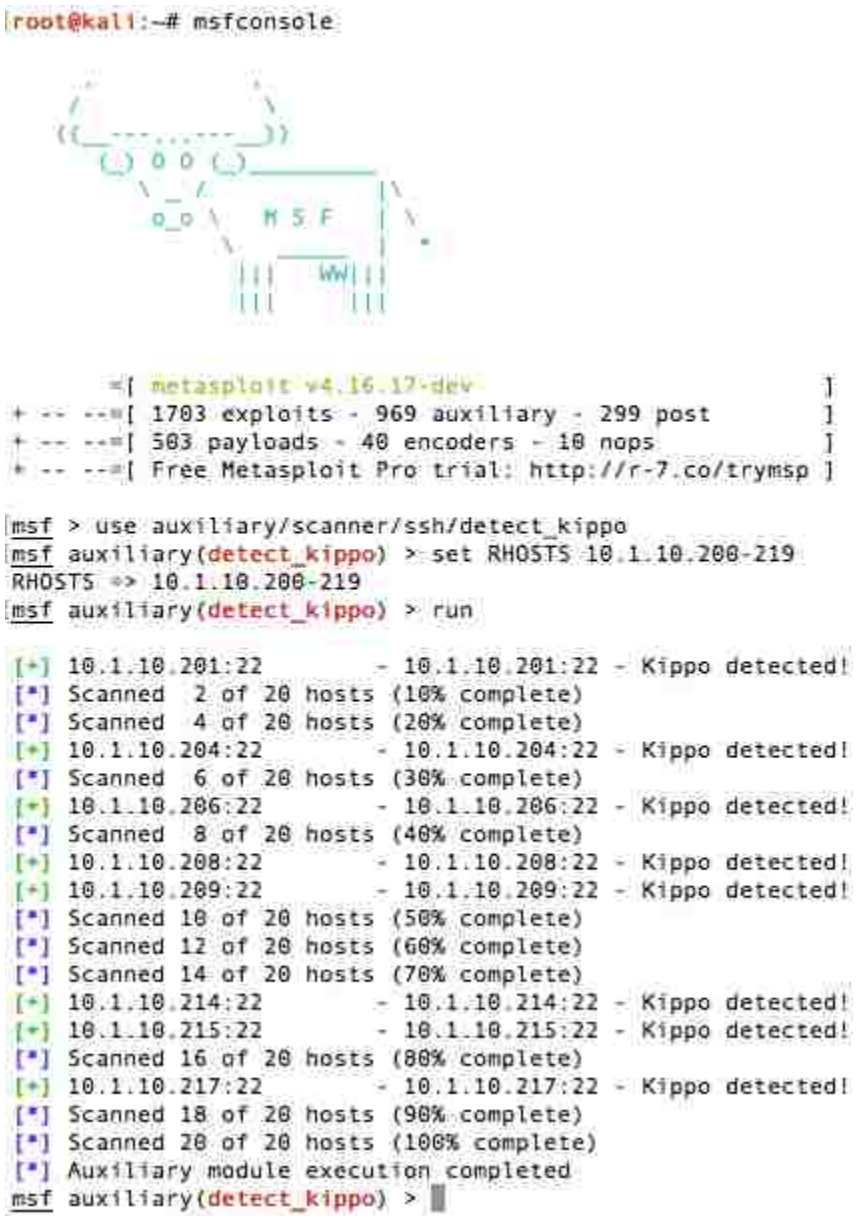


Figure 34: Metasploit detect_kippo module executed the against Services-NET target scope which returned eight systems detected as Kippo honeypots.

Manual testing

After performing a sweep and conducing reconnaissance scans, manual interaction testing was performed against the various systems. The first approach attempted to create a persistent file within the filesystem, for those that allow login. Persistent meaning creating a file, that is visible in a directory listing, that can be read or written to, and exists after a logout and re-login. Table 9 lists the results of this test where green designates normal results, red the persistence failed, and "No login" signifying it was not possible to get past the login prompt. Figure 35 displays an example of how a file being created is not persistent after login.

Table 9: Results of manual testing showing persistent file creation and execution commands

	Persistent file
10.1.10.200	No login
10.1.10.201	
10.1.10.202	No login
10.1.10.203	No login
10.1.10.204	
10.1.10.205	
10.1.10.206	
10.1.10.207	
10.1.10.208	No login
10.1.10.209	No login
10.1.10.210	
10.1.10.211	
10.1.10.212	
10.1.10.213	
10.1.10.214	
10.1.10.215	
10.1.10.216	No login
10.1.10.217	
10.1.10.218	No login
10.1.10.219	No login

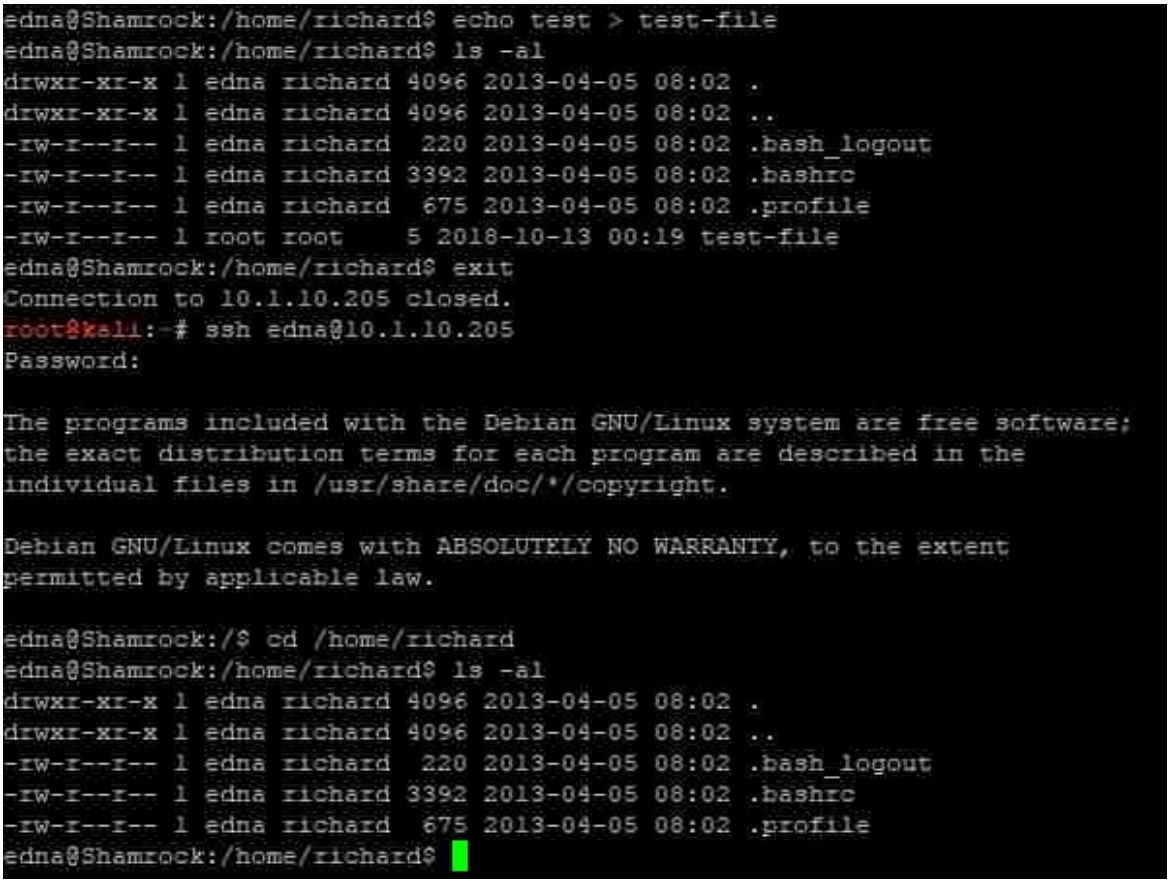


Figure 35: Example of where persistence failed after a file was created and no longer visible after re-logging back into the system

Going beyond the persistence file test, specific commands are attempted on each accessible system to see how or if they execute as expected. These living off the land commands are typically found natively on a Linux operating systems and are commonly referred to for use in the Red Team Field Manual [80]. Many of the emulated honeypots have built in commands such as ifconfig that simulate the functionality. These commands and their response can be modified within the honeypot configuration as seen in chapter 4. The commands which were executed included in many cases a specific argument that added a bit more complexity than just executing as is. For example, attempting to execute the command “ifconfig eth0” versus just

“ifconfig” should return the network configuration for just the eth0 network interface and not a generic response. Figure 36 displays where this failed to show just the eth0 adapter information versus the entire generic output. Additionally figures 36 through 39 gives examples of commands being executed on different systems where some fail and other fully succeed.

```
[Frisky$ ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:590 errors:0 dropped:0 overruns:0 frame:0
          TX packets:590 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:176413 (172.2 KiB)  TX bytes:176413 (172.2 KiB)
eth0  Link encap:Ethernet  HWaddr 0A:00:27:00:00:00
          inet addr:10.1.10.215  Bcast:10.1.10.255  Mask:255.255.255.0
          inet6 addr: fe80::800:27ff:fe00:0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:215 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:19754 (19.2 KiB)
[Frisky$ ifconfig eth0
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:590 errors:0 dropped:0 overruns:0 frame:0
          TX packets:590 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:176413 (172.2 KiB)  TX bytes:176413 (172.2 KiB)
eth0  Link encap:Ethernet  HWaddr 0A:00:27:00:00:00
          inet addr:10.1.10.215  Bcast:10.1.10.255  Mask:255.255.255.0
          inet6 addr: fe80::800:27ff:fe00:0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:215 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:19754 (19.2 KiB)
Frisky$
```

Figure 36: Example of how the ifconfig command did not function properly where it showed of only showed the eth0 adapter information in the second command

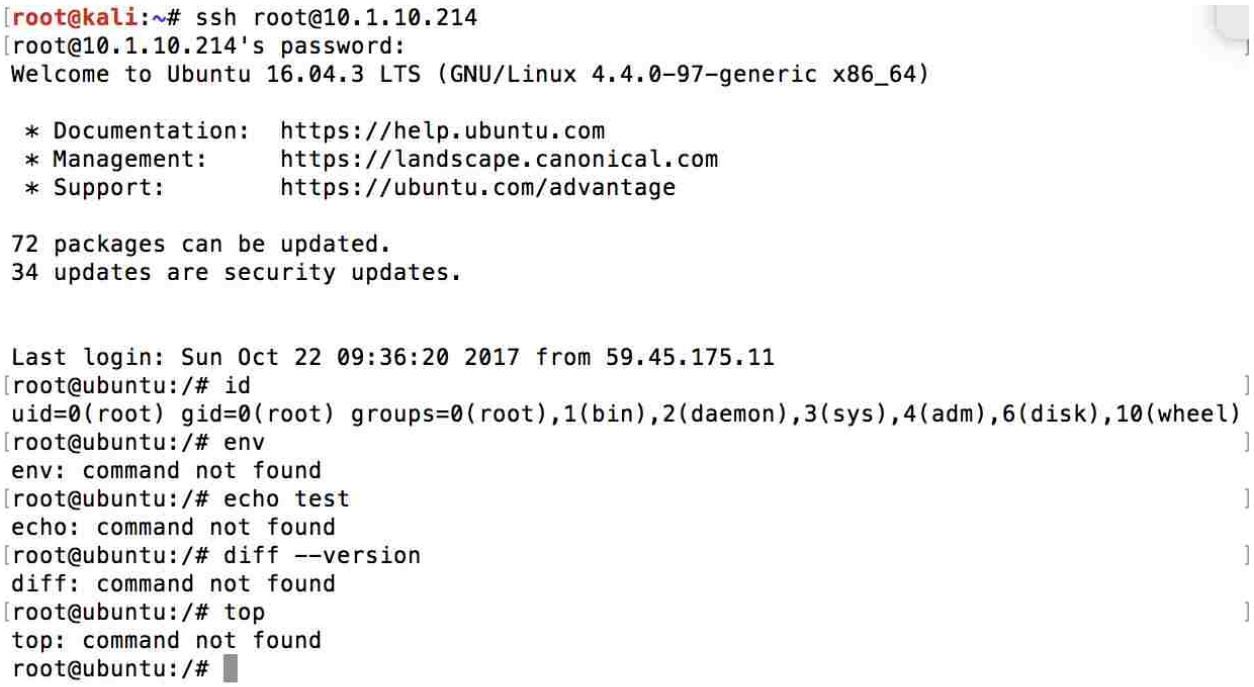


Figure 37: Example of an emulated honeypot where basic Linux commands fail to properly execute or are not found

```
root@kali:~# ssh root@10.1.10.212
Password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

root@svr04:~# id
uid=0(root) gid=0(root) groups=0(root)
root@svr04:~# top
E82: Cannot allocate any buffer, exiting...
root@svr04:~# echo test
test
root@svr04:~# pushd .
bash: pushd: command not found
root@svr04:~# ls -al
drwx------ 1 root root 4096 2013-04-05 08:25 .
drwx------ 1 root root 4096 2013-04-05 08:25 ..
drwx------ 1 root root 4096 2013-04-05 07:58 .aptitude
-rw-r--r-- 1 root root  570 2013-04-05 07:52 .bashrc
-rw-r--r-- 1 root root  140 2013-04-05 07:52 .profile
drwx------ 1 root root 4096 2013-04-05 08:05 .ssh
root@svr04:~# ls -g
root@svr04:~# ls
root@svr04:~#
```

Figure 38: Example of an emulated honeypot where basic Linux commands fail to properly execute or are not found

```
edna@Shamrock:/$ uname
Linux
edna@Shamrock:/$ uname -m
x86_64
edna@Shamrock:/$ uptime
10:40:43  up 2 days, 11:46,  1 user,  load average: 0.00, 0.00, 0.00
edna@Shamrock:/$ uptime -s
10:40:46  up 2 days, 11:46,  1 user,  load average: 0.00, 0.00, 0.00
edna@Shamrock:/$ echo test
test
edna@Shamrock:/$ top
E82: Cannot allocate any buffer, exiting...
edna@Shamrock:/$ diff --version
bash: diff: command not found
edna@Shamrock:/$ env
LANG=en_US.UTF-8
TERM=xterm-256color
SHELL=/bin/bash
TMOUT=1800
LOGNAME=edna
USER=edna
PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
HOME=/home/edna
edna@Shamrock:/$
```

Figure 39: Example of an emulated honeypot where basic Linux commands fail to properly execute or are not found

Table 10 lists the successes and failures of executing these specific commands. The attempted commands are listed across the top of the table. These commands should be default on a standard Linux build. For each command and any respective argument, if execution fails outright or is not found then it is an immediate red, scored as a 1. If the command executes, for the most part, but acts strangely then it is yellow, as a 2. If the issued command executes fully with no abnormalities, then its marked as green with a score of 3. The use of a "-" signifies that the system could not be logged into and thus was not able to be tested in this manner.

As a tally or heatmap, a final color is given for each system where the average is calculated. The heatmap scoring follows the schedule of 1 - 1.99 is red, 2 – 2.99 is yellow, and 3 is green. As an example, using 10.1.10.204 there are 10 yellow points, 5 red, and 6 green which

sums to a total of 21 points. The 21 points are divided by the total number of executed commands which is 11 and produces the average of 1.45.

Table 10: Results of manual testing showing persistent file creation and execution commands

	Ifconfig eth0	Diff --version	Echo test	Netstat --tcp	env	top	id	Pushd/popd	Uname -m	Uptime -s	Ls -g	Heatmap
10.1.10.200	-	-	-	-	-	-	-	-	-	-		
10.1.10.201	2	1	3	1	1	1	3	1	2	2	2	1.72
10.1.10.202	-	-	-	-	-	-	-	-	-	-		
10.1.10.203	-	-	-	-	-	-	-	-	-	-		
10.1.10.204	2	1	1	2	1	1	1	1	2	2	2	1.45
10.1.10.205	1	1	3	2	3	1	3	1	3	2	2	2.00
10.1.10.206	2	1	1	2	1	1	3	1	2	2	2	1.63
10.1.10.207	3	3	3	3	3	3	3	3	3	3	3	3.00
10.1.10.208	-	-	-	-	-	-	-	-	-	-		
10.1.10.209	-	-	-	-	-	-	-	-	-	-		
10.1.10.210	3	3	3	3	3	3	3	3	3	3	3	3.00
10.1.10.211	3	3	3	3	3	3	3	3	3	3	3	3.00
10.1.10.212	2	1	3	2	3	1	3	1	3	2	2	2.09
10.1.10.213	3	3	3	3	3	3	3	3	3	3	3	3.00
10.1.10.214	2	1	1	2	1	1	3	1	2	2	2	1.63
10.1.10.215	2	1	1	2	1	1	3	1	2	2	2	1.63
10.1.10.216	-	-	-	-	-	-	-	-	-	-		
10.1.10.217	2	1	3	1	1	1	3	1	2	2	2	1.72
10.1.10.218	-	-	-	-	-	-	-	-	-	-		
10.1.10.219	-	-	-	-	-	-	-	-	-	-		

The ending high level results are seen in Table 11 for each of the three testing categories. Using the same heatmap approach seen in the above command execution testing table, a final tally is presented across all testing areas. For those tests that were not possible due to not being able to login in where marked with a "-"and not included in the overall calculation. These

holistic results are then used to determine if it is believed to be a honeypot, there is suspicious it is a honeypot, or not a honeypot. Also, if a honeypot is confirmed detected by tools then they ae marked with a "*" and automatically disbelieved (as red). The more detailed overview of the results will be reviewed in chapter seven.

Table 11: Heatmap of overall results for all three testing categories

Testing Type	Sweep	Scanning			Manual		
Approach	Nmap service scan	Low Ciphers #	Metasploit	SSH Supported Algos	persistence	Command execution	Resulting score
10.1.10.200	3	2	3	2	-	-	2.50
10.1.10.201	3	2	*1	2	1	1	1.66
10.1.10.202	3	2	3	3	-	-	2.75
10.1.10.203	3	3	3	2	-	-	2.75
10.1.10.204	*1	2	*1	1	1	1	*1.16
10.1.10.205	3	2	3	3	1	2	2.33
10.1.10.206	2	2	*1	1	1	1	*1.33
10.1.10.207	3	3	3	3	3	3	3.00
10.1.10.208	3	2	*1	1	-	-	*1.75
10.1.10.209	3	2	*1	2	-	-	*2.00
10.1.10.210	3	3	3	3	3	3	3.00
10.1.10.211	3	3	3	2	3	3	2.83
10.1.10.212	3	3	3	3	1	2	2.00
10.1.10.213	3	2	3	1	3	3	2.50
10.1.10.214	*1	2	*1	1	1	1	*1.16
10.1.10.215	3	2	*1	1	1	1	*1.50
10.1.10.216	3	3	3	3	-	-	3.00
10.1.10.217	3	2	*1	2	1	1	*1.66
10.1.10.218	3	3	3	3	-	-	3.00
10.1.10.219	3	3	3	3	-	-	3.00

The scores in table 11 are raw and represent a total as a part of a scoring mechanism. As the control experiment was conduct and he various tests passed or failed, scores for each were summed and averaged to deduce if closer to a score of 3 the more real the system is. To get the percent of success in detection, the percent calculation is the score minus one with the sum divided by two. For example, in 10.1.10.207 the score is 3.0. The calculation is (3.0-1)/2 which equates to 1 or 100%. Another example is system 10.1.10.214 where the score was 1.16. The calculation is 1-((1.16-1/)2) which equals .92. In the case of the pure honeypot 10.1.10.210 and 10.1.10.219 came back as 3 based on the experiment results suspecting and scoring it was a real system. In this case knowledge of the actual state of each machine is required to accurately score their success percent.

CHAPTER SIX: PARTICIPANT-BASED EXPERIMENT

In this chapter we introduce the use of participants into the experiment range. Like the control experiment, each participant will attempt to identity which services in the Services-NET are indeed honeypots, are suspicious of being a honeypot or not a honeypot. The main idea is to introduce a diverse set of individuals and skills that will be corelates to a defined attacker profile discussed in chapter 2. Upon successful recruitment, each participant will be asked to take a survey of their skills, complete a score card of their testing results, and conduct and option exit survey on the results.

Participant Recruitment

The UCF Institute Review Board (IRB) approved the participant-based experiment using human testing on August 17th, 2018. Being a cyber professional myself, the availability and ability to engage cyber and IT professionals of all skill levels was trivial. Recruitment began immediately after IRB approval by sending the flyer in Appendix A to 15 individuals. Some of those individuals referred additional potential participants which equated to a total of 20.

After identifying the potential participates, each was sent the IRB Adult Consent form to review and respond by electronically via email that they acknowledge and agree to participate. The Adult Consent form reviewed the overall experiment, expectations for involvements, restrictions for minors or those that do not have the right skills or computer access, privacy and more. The total number of participates that agreed to the consent was 16. The outcome of participation and metrics will be discussed at the end of this chapter.

Survey, Connecting and Instructions

Upon agreement to the Adult Consent form, each participant was sent material that included a skills survey, connectivity information, and a set of instructions. The survey is seen in appendix B where the red outlined portion is shared with the participants and the rest is used to calculate the overall skills scoring. The survey consists of questions that evaluate the participants overall technical skill acumen in multiple disciplines to include cyber and general information technology. Each category was asked to be rated 0 through 10 where 0 is no skills in that area and 10 is a subject matter expert. Some questions were yes or no such as "Have you ever discovered a zero day?". In this case yes equals 10 and no is 0. Each question score can be weighed differently if necessary to help calibrate based on an intuitive view of its much threat impact in this range and participant experiment. For example, having knowledge of database technologies may be .25 where discovering a zero day is 5.0. The weights for each question are summed and averaged to a total of 1.0. Using this weight, the participant entered score for a question is multiplied by the weight to produce the score result. So, if a participant put 4 out of 10 on the question about database knowledge then its score result is 1. All question score results are summed and averaged to produce a final participant skill score from 0-10.

The connection information consists of the participants VPN certificates and configuration file, usernames, passwords, and assigned endpoint attacker Kali platform IP which is hosted in the Attacker-NET portion of the range. The VPN configuration file automatically provides all the connection information to the participants VPN client to connect to a dynamic Domain Name Service (DNS) of deception.ddns.net. Their certificate enables two factor authentication with the use of their VPN password. Once a VPN connection is successfully

established then they can access all devices on the Attacker-NET or Services-NET from their remote system. It is encouraged to use their assigned Kali platform to have a record of their activity, but the option exists to use their own.

The set of instructions given to each participant is seen in appendix C. The instructions start by first establishing the objective with the focus being on attempting to discover which services are either an SSH honeypot service or not. They are told to use any TTP or tool available, but they must document how you came to their decision for each IP/port. The documentation is requested to include each IP/Port tested, the outcome (yes, a honeypot, no not a honeypot, suspicious of being a honeypot), why they decided one way or the other, supporting tool information, screen shots or anything else that would be helpful. Furthermore, the objective states that if they can not find any indication or supporting evidence of the service being a honeypot, then assume it is not and select "No, not a honeypot".

The instructions depict the experiment range to give a visual topology of how the systems are interconnected. Details are provided on the Attacker-NET and how they are provided a Kali Linux system and can request a Windows 10 system if desired. The target scope IPs in the Services-NET are provided with the focus solely being on the SSH service on TCP port 22. They are asked not to deviate from the defined scope or laterally move from inside an accessed target system. For example, once they access a system in the Services-NET, do not from that point of presence scan, probe, or SSH to other systems. This was to prevent them from possibly seeing information about other ports listening on the honeypot systems that were not intended to be in scope for this experiment.

The instructions also cover some ancillary items such as privacy, OSINT file, prizes, and scoring. First is the overview of privacy of each participant and how their, identify and participation will be protected from disclosure and will abide by UCF policies and procedures in data retention, security, and destruction. Next is describing the OSINT file provided on each Kali system as a potential accurate guide to what systems have some working passwords and more. The intention of the OSINT file was to prevent wasted time attempting to brute force passwords that would never work. As described in Chapter 4, some systems in the Services-NET had no username and password combinations that would work as the intention was to assess the system from the login prompt only. The Scoring is described where each correct answer is worth 1 point. Each wrong answer is -1 point. If suspicious is selected, then no points are given or reduced. Maximum number of points is 20. Finally, prizes are discussed on the amount and as primarily a token of appreciation for participating in the experiment and providing valuable results.

The instructions end with describing the expected returned results, namely the skills survey, scorecard and exit survey. An overview of how to complete the skills survey is described with heavy emphasis on the importance of being as honest as possible. A scorecard template is provided to give an example of what the expected returned results should look like. Using this exact template is not required but more so to reflect what information is to be collected for each IP/port. The exit survey is the last item to be discussed in the participant instructions as the final task that can be optional to complete. The exit survey is given to each participant upon receiving their competed skills survey and scorecard asking a series of questions about their experience, what they found hard or easy, and more. The option was also given to

provide feedback on anything that was not mentioned or asked for specifically. The exist survey questions are seen below, and a summary of the results is in the next chapter:

Exit Survey Questions:

1. What tool did you seem to leverage the most?
2. What was your overall strategy to attempt and identify which services were honeypots and which were not?
3. Did having the OSINT intelligence help you in any way?
4. Can you identify what sort of honeypot software projects were used (i.e.: software name)?
5. Have you heard of any other honeypots for secure shell that were not used in this experiment?
6. What was the easiest part in identifying potential honeypots?
7. What was the hardest part of identifying potential honeypots?
8. If you were to design an SSH honeypot, what sort of features would you focus on most to further obfuscate it from being identified?
9. About how many hours did you spend on the experiment overall?
10. What IP was the fastest to identify and why?
11. What IP made you seem the most unsure?
12. Do you have any other information or feedback you wish to share?

Participant Outcome

The experiment range was open for participant testing on September 3rd, 2018 November 20th. During this time participants have taken part in completing the survey, conducting testing, providing a scorecard, and in most cases answering the exit survey questions. Table 12 will list the contribution for each participant while figure 40 will give the totals. Out of the total individuals sent the flyer, 80%, or 19 individuals, agreed to the Adult Consent form. During the IRB submission, the outline protocol document for this experiment listed the desired completion to be from 8-10 participants.

Table 12: Participation outcome thus far results for experiment

Recruit and Participant	Sent Flyer	Agreed to Adult Consent	Actually, Logged on	Completed Survey	Completed Scorecard	Completed Exit Survey
1	x	x	x			
2	x	x	x	x	x	x
3	x	x	x	x	x	x
4	x	x	x	x	x	x
5	x	x	x			
6	x	x	x	x		
7	x	x	x	x	x	x
8	x	x	x	x	x	x
9	x	x				
10	x	x	x			
11	x	x				
12	x	x	x	x	x	x
13	x	x				
14	x	x				
15	x	x	x	x	x	
16	x	x	x	x	x	
17	x	x				
18	x	x	x			
19	x	x	x	x	x	
20	x	x				
21	x					
22	x					

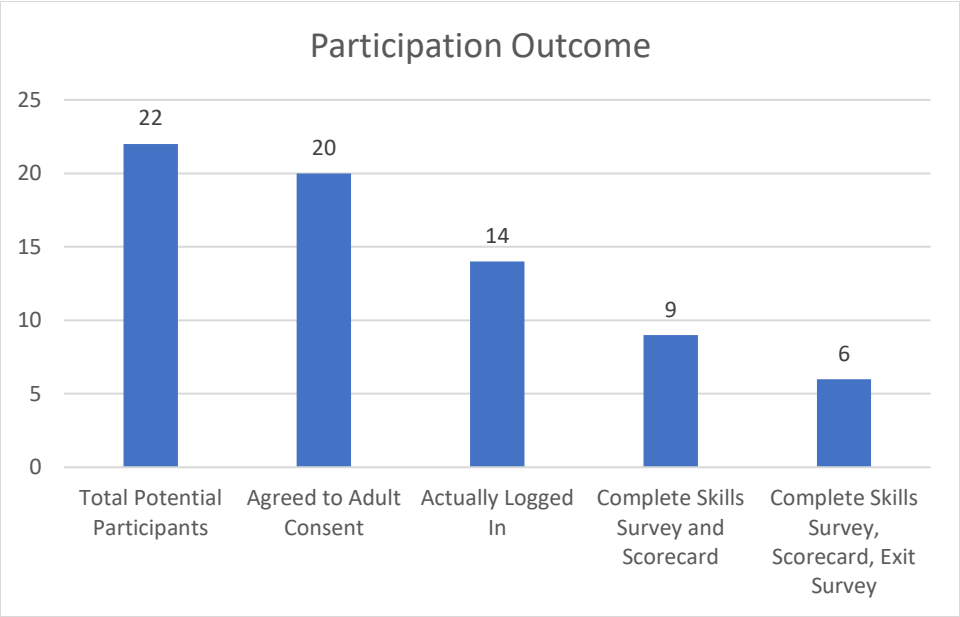


Figure 40: Metrics on participation outcome for participant-based experiment

The impetus is having skilled individuals participate in the experiment to give insight into the ease or difficulty of detecting honeypots systems of different characteristic and configurations. Having different participants also opens the possibility for identifying systemic or unique approaches. The results for of the participant-based experiment will be reviewed in the next chapter. The results of each participant will be compared to one another and to the outcome of the control experiment.

CHAPTER SEVEN: EXPERIMENT RESULTS

This chapter covers the results for both the empirical control and participant experiments. We want to answer specific questions from the experiment results that will be used as input to conduct modeling and develop a framework. For this we consider the detectability and in choosing the right honeypot application. We also explore how different levels of attacker skills corelate to the success of detectability. The questions to be answered are as follows:

- How do real systems, pure, and emulated honeypots compare from an interactive and non-interactive state in their ability to not being detected?
- How effective are the different types of emulated honeypots both from an interactive and non-interactive state in not being detected?
- Do custom changes to the honeypots change its detection susceptibility?
- How does varying participant skill scores influence the overall detectability success across all the different tested systems?

Using the data and in answering the above questions, we corelate and compare the results of each tested system by grouping them into their respective category by type and interaction. Additionally, we create random generated values for both the scorecards and participant skill surveys leveraging standardized parameters from the actual results, such as the mean and standard deviation. We use this to deduce a set of predictable results based solely on the empirical and participant data which is leveraged in further modeling and a framework. Lastly using the participant sills survey results, we corelate and observe the aptitude for diverse attacker

levels to detect the presence of the deceptive software. Figure 41 is the high-level approach to the experiment results and the major areas covered in this chapter.

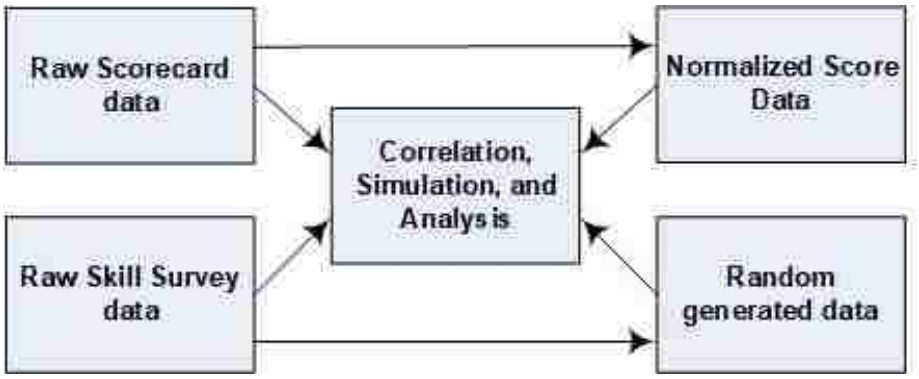


Figure 41: High-level areas of focus for experiment results data

Experiment Scorecard Data

Beginning with the participant-based experiment, the raw scorecard data is listed in table 13. The raw date follows the scoring model where if a participant got the answer correct and identified the system they received 1 point. If they marked the system as incorrect they received -1. If they marked as suspicious they got 0 points. To simply the results and make it easier for data comparison, table 14 was constructed to normalize the data in which 1 point was assigned if the participant got the right answer and 0 points otherwise. If any participants marked suspicious, then their supplemental notes and reasoning were used to make a case by case decision on if their suspicion was correct or not.

Table 13: Raw data results for participant-based experiment

Honeypot type	IP/Honeypot/Can Interact	Partipant 4	Partipant 2	Partipant 15	Partipant 3	Partipant 12	Partipant 7	Partipant 16	Partipant 8	Partipant 19
	200-No-No	1	-1	-1	0	1	-1	1	1	1
Kippo	201-Yes-Yes	1	1	1	1	1	1	1	1	1
Cowrie	202-Yes-No	1	-1	0	0	1	-1	-1	1	-1
	203-No-No	1	1	0	1	0	1	0	1	1
XSweet	204-Yes-Yes	1	1	1	1	1	1	1	1	1
Cowrie	205-Yes-Yes	1	1	1	-1	1	-1	0	1	0
Kojoney2	206-Yes-Yes	1	1	1	1	1	1	1	1	1
	207-No-Yes	1	1	-1	0	0	1	-1	1	-1
Kojoney2	208-Yes-No	1	1	1	1	0	1	1	1	1
Kippo	209-Yes-No	1	1	1	1	1	1	1	1	1
pure	210-Yes-Yes	-1	1	1	-1	0	-1	-1	-1	-1
	211-No-Yes	1	1	0	1	0	1	0	1	1
Cowrie	212-Yes-Yes	1	-1	1	-1	1	1	0	1	-1
	213-No-Yes	1	-1	-1	1	0	-1	1	1	-1
XSweet	214-Yes-Yes	1	1	1	1	0	1	1	1	1
Kojoney2	215-Yes-Yes	1	1	1	1	1	1	1	1	1
	216-No-No	1	1	0	1	0	1	0	1	1
Kippo	217-Yes-Yes	1	1	1	1	1	1	1	1	1
	218-No-No	1	1	0	1	0	1	0	1	1
pure	219-Yes-No	-1	-1	-1	-1	0	-1	-1	-1	-1

Table 14: Normalized data results for participant-based experiment

Honeypot type	IP/Honeypot/Can Interact	Partipant 4	Partipant 2	Partipant 15	Partipant 3	Partipant 12	Partipant 7	Partipant 16	Partipant 8	Partipant 19
	200-No-No	1	0	0	0	1	0	1	1	1
Kippo	201-Yes-Yes	1	1	1	1	1	1	1	1	1
Cowrie	202-Yes-No	1	0	0	0	1	0	0	1	0
	203-No-No	1	1	0	1	0	1	0	1	1
XSweet	204-Yes-Yes	1	1	1	1	1	1	1	1	1
Cowrie	205-Yes-Yes	1	1	1	0	1	0	0	1	0
Kojoney2	206-Yes-Yes	1	1	1	1	1	1	1	1	1
	207-No-Yes	1	1	0	0	0	1	0	1	0
Kojoney2	208-Yes-No	1	1	1	1	0	1	1	1	1
Kippo	209-Yes-No	1	1	1	1	1	1	1	1	1
pure	210-Yes-Yes	0	1	1	0	0	0	0	0	0
	211-No-Yes	1	1	0	1	0	1	0	1	1
Cowrie	212-Yes-Yes	1	0	1	0	1	1	0	1	0
	213-No-Yes	1	0	0	1	0	0	1	1	0
XSweet	214-Yes-Yes	1	1	1	1	0	1	1	1	1
Kojoney2	215-Yes-Yes	1	1	1	1	1	1	1	1	1
	216-No-No	1	1	0	1	0	1	0	1	1
Kippo	217-Yes-Yes	1	1	1	1	1	1	1	1	1
	218-No-No	1	1	0	1	0	1	0	1	1
pure	219-Yes-No	0	0	0	0	0	0	0	0	0

After normalizing the participant scorecard data, analysis was conducted against each system type, comparing emulated honeypots from an interaction and configuration perspective. In figure 42, the system types for all configurations are compared where pure honeypots scored the least at only 22% detection success. Emulated honeypots reached a total of 84% successful detection with real systems at 59%. Looking closer at the interaction level, figure 43 shows a similar trend with pure being the lowest and emulated the highest. It is with no surprise that the interactive emulated honeypots had the highest detection success based on having the ability to

interact with the system. Figure 44 focuses specifically on the emulated honeypot type where non-interactive Cowrie was the most successful at not being detected. Non-interactive XSweet was N/A for non-interactive as it was not a part of the experiment range target scope. Figure 45 considers the build type and whether the emulated honeypot was a default deployment or custom configured. There was no real difference between the custom and default builds from a detection success. Configuration as being what was changed within the emulated honeypot software config files.

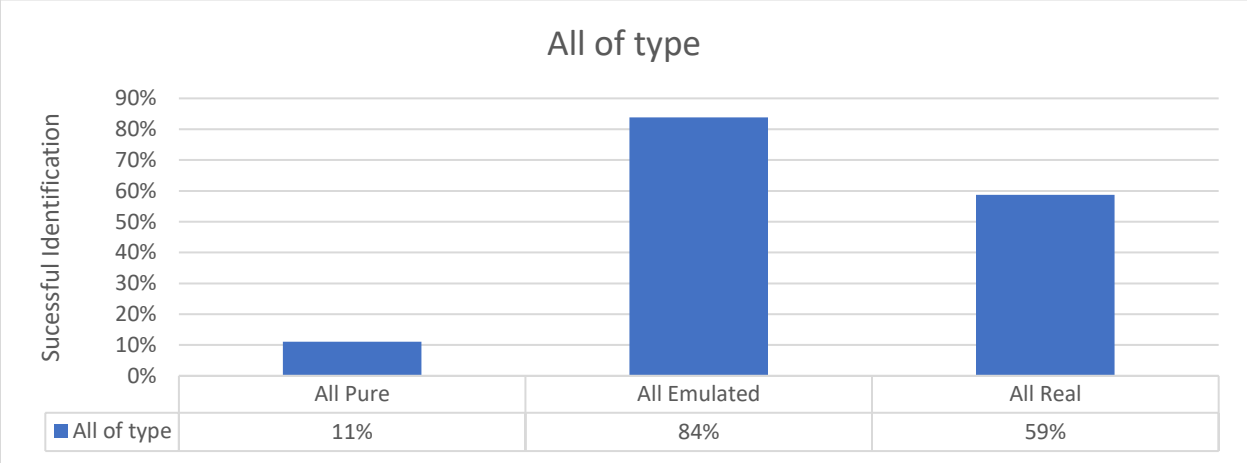


Figure 42: Comparing system types from a successful detection perspective.

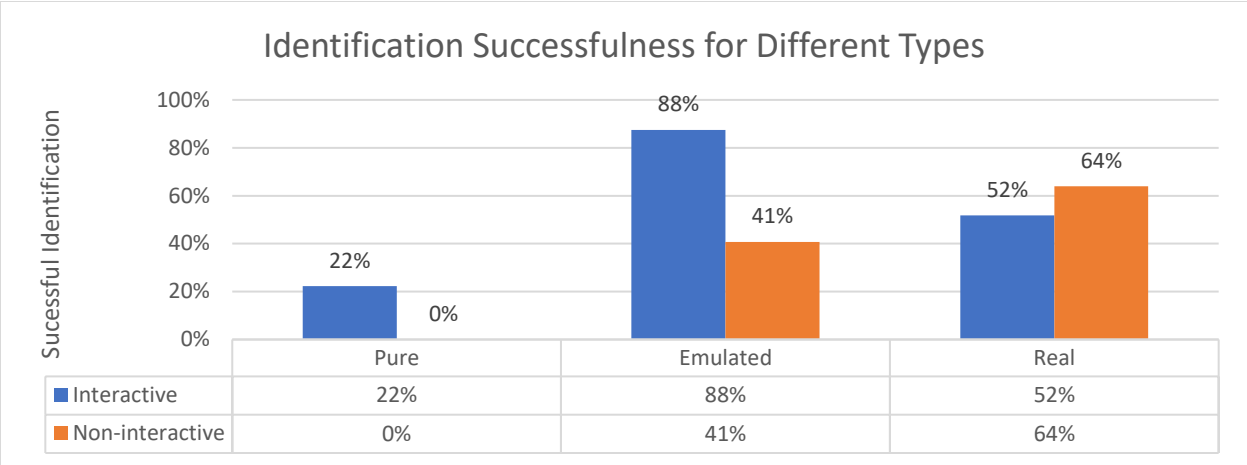


Figure 43: Comparing different system types and interaction level from a successful detection perspective.

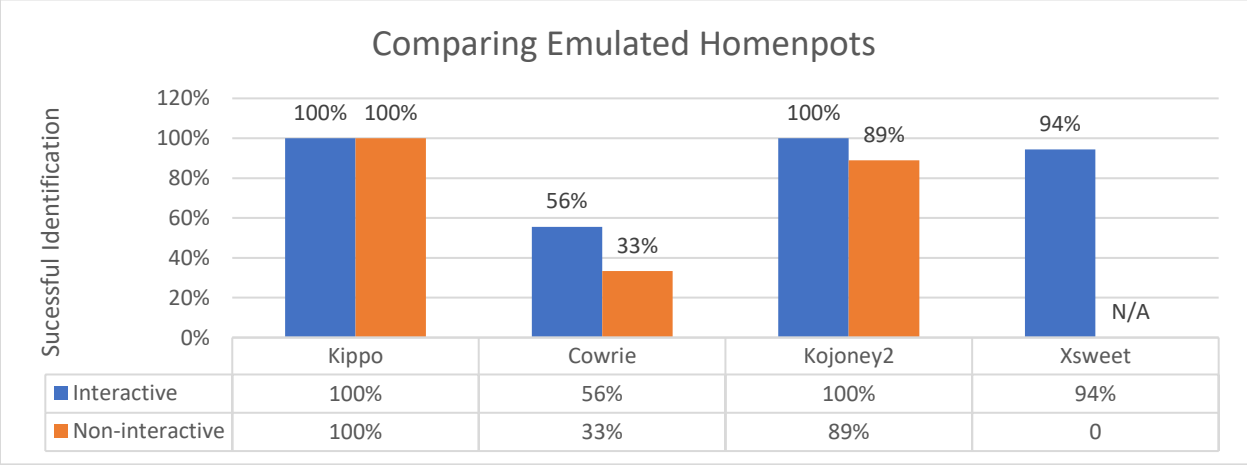


Figure 44: Comparing emulated honeypots and its interaction from a successful detection perspective.

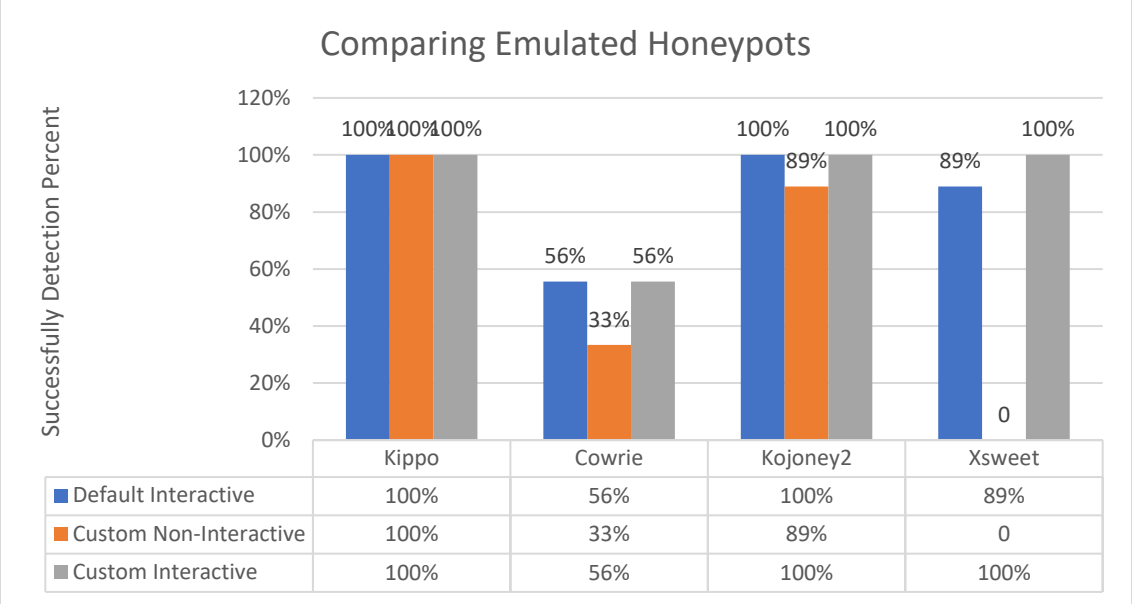


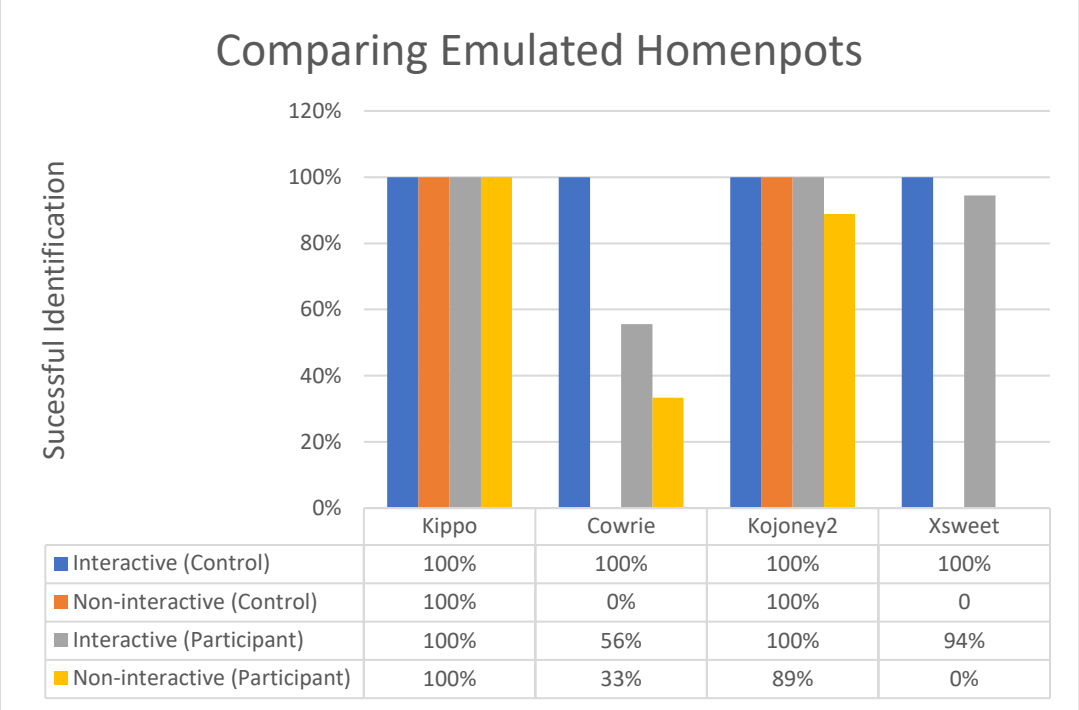
Figure 45: Comparing successful detection for emulated honeypots from an interaction and configuration perspective.

Referencing the control experiment results depicted in chapter 5 table 11, each system is given a score of 0-3. Recalling that 0-1.99 defines 100% detected as a honeypot and 3 as a 100% real system while the range in between 2-2.99 is considered suspicious and not fully confirmed. The closer to 3 the more likely the odds of the system being real and equally for honeypots in the other direction. To take the suspicious scores into consideration, the raw score data will be normalized like the participant score data where it will be a 1 for correct and 0 for false. For this we will use a threshold to set the limits that can take on any value between 2-2.99. North of the threshold is a real system and south a honeypot. For this research we will use the average score for all twenty systems which is 2.64 or 82% where the mean is considered real. Below in table x are the empirical control experiment results. The percent are based on the calculation described at the end of the control experiment chapter.

Table 15: Normalizing control experiment data

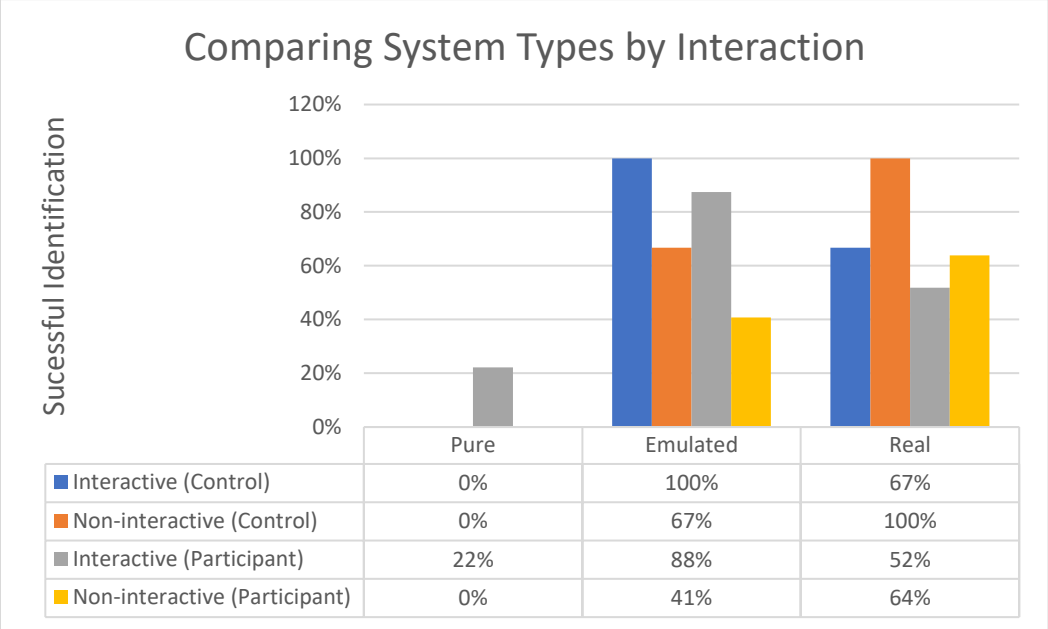
System/ Honeypot/ Interaction	Control Experiment Score	Percent	Resulting Normal score
200-No-No	2.50	75%	1
201-Yes-Yes	1.66	67%	1
202-Yes-No	2.75	13%	0
203-No-No	2.75	88%	1
204-Yes-Yes	1.16	92%	1
205-Yes-Yes	2.33	34%	1
206-Yes-Yes	1.33	84%	1
207-No-Yes	3.00	100%	1
208-Yes-No	1.75	63%	1
209-Yes-No	2.00	50%	1
210-Yes-Yes	3.00	0%	0
211-No-Yes	2.83	92%	1
212-Yes-Yes	2.00	50%	1
213-No-Yes	2.50	75%	0
214-Yes-Yes	1.16	92%	1
215-Yes-Yes	1.50	75%	1
216-No-No	3.00	100%	1
217-Yes-Yes	1.66	67%	1
218-No-No	3.00	100%	1
219-Yes-No	3.00	0%	0

Figures 46 and 47 depict comparisons between the control and participant experiment scores. Both experiments followed a similar trend particularly for the Cowrie honeypot. The control experiment scored higher for Cowrie on interactive while the participant-based experiment scored higher in non. Both experiments resulted in almost no scores for the pure honeypots with only participant scoring 22% in the interactive variant. The control experiment scored slightly higher in both the emulated and real systems.



	Kippo	Cowrie	Kojoney2	Xsweet
Interactive (Control)	100%	100%	100%	100%
Non-interactive (Control)	100%	0%	100%	0
Interactive (Participant)	100%	56%	100%	94%
Non-interactive (Participant)	100%	33%	89%	0%

Figure 46: Comparing emulated honeypots from participant and control experiments both for non-interactive and Comparing emulated honeypots and its interaction from a successful detection perspective.



	Pure	Emulated	Real
Interactive (Control)	0%	100%	67%
Non-interactive (Control)	0%	67%	100%
Interactive (Participant)	22%	88%	52%
Non-interactive (Participant)	0%	41%	64%

Figure 47: Comparing different system types from participant and control experiments both for non-interactive and Comparing emulated honeypots and its interaction from a successful detection perspective.

Experiment Skill Survey Data

The raw participant skill survey data is listed in table 16. Each subject area is scored by each participant where 0 is no skills and 10 is an expert. An Intuition weight is used to help calibrate the questions based on relevance. Some questions are more pertinent to the skills needed to conduct cyber testing activities within the participant-based experiment focus areas. The main idea was to determine their overall technical and logical skill set to get a good idea of their overall technical capabilities and potential attacker threat. For all the results discussed in this chapter and used in subsequent chapters, the intuition was set to 1. The totals for each skill focus area and for each participant are also listed. The totals highlight the areas which seem to be the strongest and likewise the weakest. Figure 48 shows this visually where "Have you ever compromised (exploited) a computing system" is the highest and "Unique coding languages (Lisp, Prolog, etc)" the weakest. Since the last four questions are not calibrated the same as the other questions, being its either 0 or 10 for yes or no versus a number in between, the highest skill is more appropriately "Using Security tools (nmap, Metasploit, scanners, enum, queries, etc.)".

Table 16: Raw skill survey data results from participant-based experiment

	intuition weight	Partipant 4	Partipant 2	Partipant 15	Partipant 3	Partipant 12	Partipant 7	Partipant 16	Partipant 8	Partipant 19	Totals
Ordered Participant #		9	4	8	2	6	7	3	5	1	
Reverse Engineering	1	5	4	7	1	5	5	1	2	2	30
Exploit writing/creating	1	4	3	9	2	4	6	1	3	2	32
Assembly Language	1	4	2	7	1	6	5	0	2	4	27
Windows OS Internals	1	9	10	6	8	5	6	7	9	3	60
Linux OS internals	1	9	2	5	7	6	6	5	6	3	46
Networking essentials knowledge of (TCP, UDP, IP, ICMP, sniffing, pcap, config, etc.)	1	10	6	8	8	7	6	7	8	5	60
Using Security tools (nmap, Metasploit, scanners, enum, queries, etc.)	1	10	6	9	9	8	7	6	9	6	64
Database knowledge (oracle, sql, MySQL, etc.)	1	8	8	6	6	7	8	3	9	4	55
Malware analysis (viruses, packing, botnets, obfuscation, etc)	1	7	2	7	1	2	4	2	2	4	27
Reverse Engineering and Assembly tools (IDA, OllyDbg, GDB, Hex editors, etc)	1	7	3	7	2	5	6	1	2	3	33
Web application Penetration Testing	1	7	6	6	6	9	8	5	10	3	57
Evade Endpoint protections (HIPPS, AV, Sysmon, etc.)	1	8	3	8	5	6	6	3	5	0	44
Electronic Warfare (SDR, SATCOM, RF)	1	5	0	4	1	0	7	1	1	0	19
Wireless hacking (802.11, ZigBee, Bluetooth)	1	7	1	5	2	6	7	2	5	1	35
web application development (HTML, PHP, Python, etc.)	1	5	7	5	5	8	9	4	9	1	52

	intuition weight	Partipant 4	Partipant 2	Partipant 15	Partipant 3	Partipant 12	Partipant 7	Partipant 16	Partipant 8	Partipant 19	Totals
Offensive Development (exploit, overflows, etc.)	1	4	3	9	1	7	8	3	5	1	40
Scripting (Bash, PowerShell, Python, Perl, Bat)	1	5	3	8	6	8	8	3	8	4	49
Low level coding languages (Assembly, Fortran, C, etc)	1	3	3	7	1	7	7	1	3	5	32
Higher level coding languages (C++, Java, VB, etc)	1	3	8	7	3	8	8	4	9	5	50
Unique coding languages (Lisp, Prolog, etc)	1	1	0	5	0	2	5	0	5	0	18
System forensics and cybercrime skills	1	4	1	7	0	1	5	7	5	5	30
General Penetration Testing Skills/methodology	1	10	6	8	7	8	8	4	9	5	60
Incident Response (uncovering and detecting attacks)	1	4	2	6	1	1	5	8	4	5	31
Discovered a zero day before?	1	0	10	0	0	10	0	0	0	0	20
Have you ever compromised (exploited) a computing system	1	10	10	10	10	10	10	10	10	10	80
Do you work on puzzles often (Sudoku, Crossword, etc.)?	1	10	10	0	0	10	0	10	0	10	40
Have you ever coded an exploit?	1	10	10	10	0	10	10	0	10	0	60
Totals divided by 10		169	129	176	93	166	170	98	150	91	
Average		6.3	4.8	6.5	3.4	6.1	6.3	3.6	5.6	3.4	

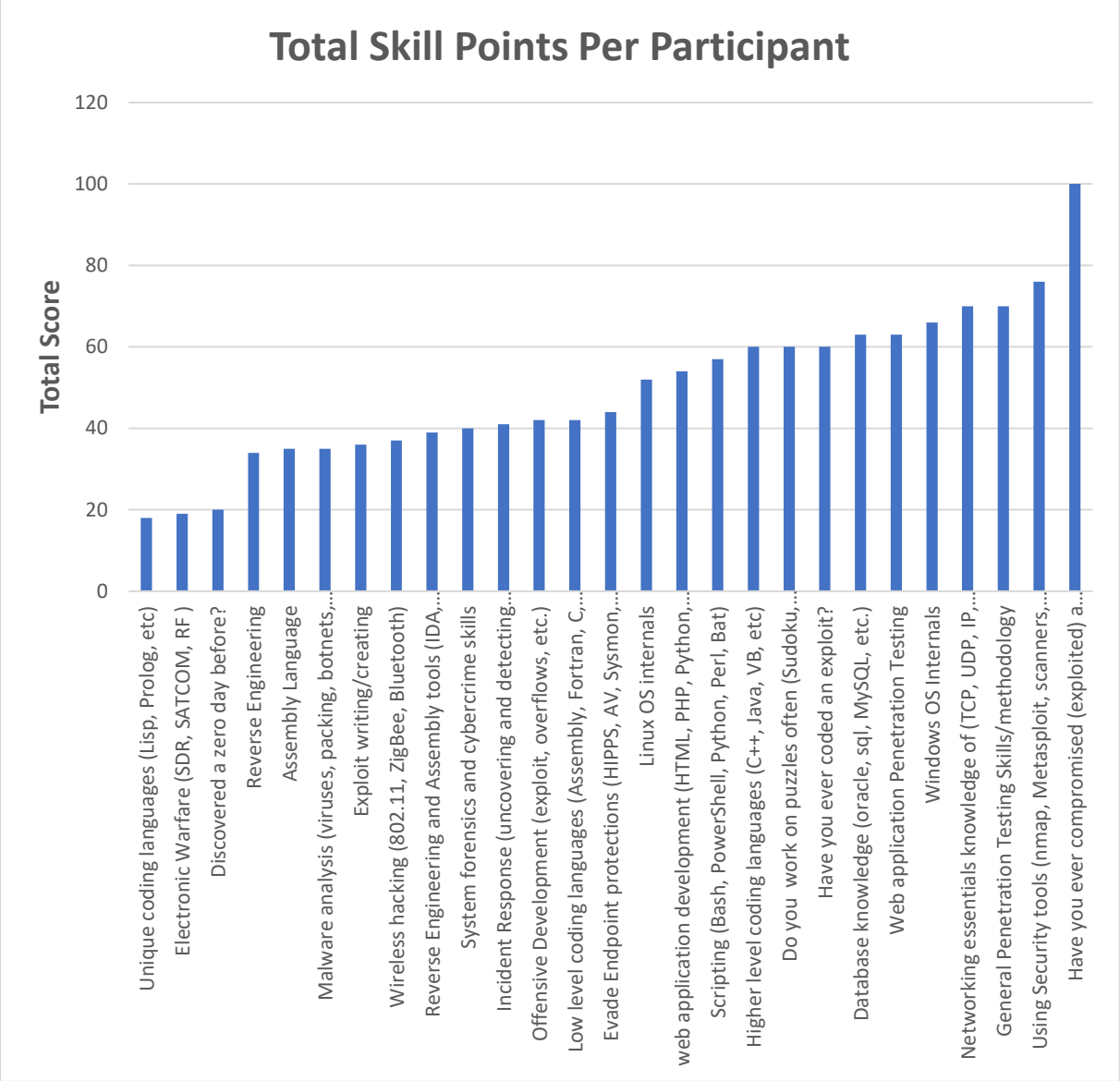


Figure 48: Total participant scores for each survey question

The skill and scorecard results are correlated together to depict participant performance based on their skills. Figure 49 plots the detection success (scores) for all systems compared to the average skill level for each participant. The 16th, 7th, and 4th participants variance between skills and scores almost align with very little offset. Many of the other participants do closely align except for participant 12 and 15 whose scores came in much lower than their documented

skill results. This can be caused by several factors such as over inflating their skill level versus their actual ability. It can also mean that the higher skilled participants ran through the experiment quicker due to over confident in their ability to detect the systems.

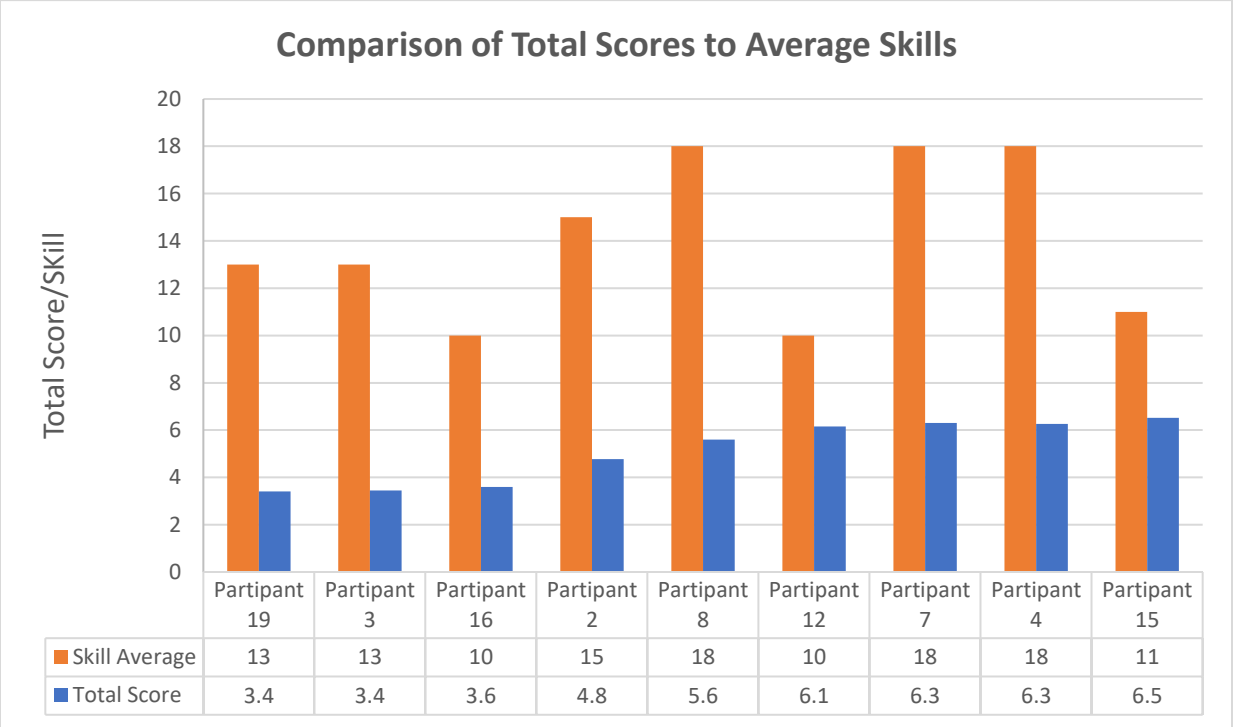
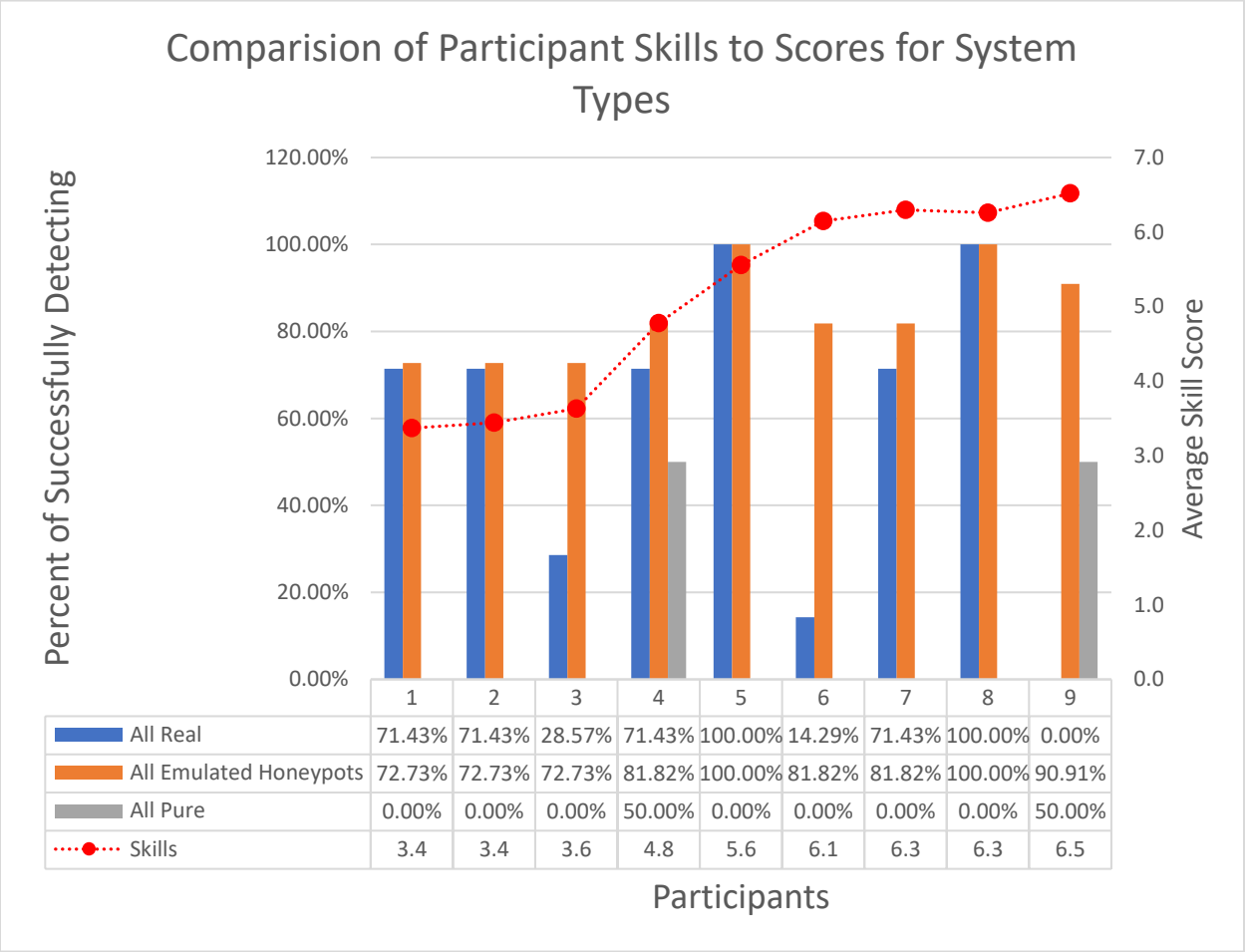


Figure 49: Comparing total experiment scores to average skill survey scores for each participant

Observing specific system types that include emulated, real, and pure compared against the participant skill set revels a slightly different viewpoint as seen in figure 50. The closest alignment is participant 4 with the emulated system detection and participant 5 with both emulated and real. Participants 1-5 appear mostly aligned where participants 6-9 demonstrate higher self-surveyed skills but lower scores. Figure 51 also give a bullseye viewpoint by plotting the average participant score compared to the 20 target systems. The closer to the center, the lower the scores are in successfully detecting the system. It is easy to see that the lowest score

are the two pure honeypots where the non-interactive is in the center at 0 and the interactive at .22. The emulated honeypots are the highest and furthest away at 1.0.



	1	2	3	4	5	6	7	8	9
All Real	71.43%	71.43%	28.57%	71.43%	100.00%	14.29%	71.43%	100.00%	0.00%
All Emulated Honeypots	72.73%	72.73%	72.73%	81.82%	100.00%	81.82%	81.82%	100.00%	90.91%
All Pure	0.00%	0.00%	0.00%	50.00%	0.00%	0.00%	0.00%	0.00%	50.00%
Skills	3.4	3.4	3.6	4.8	5.6	6.1	6.3	6.3	6.5

Figure 50: Comparing different system types from a successful detection perspective to average skill scores for each participant

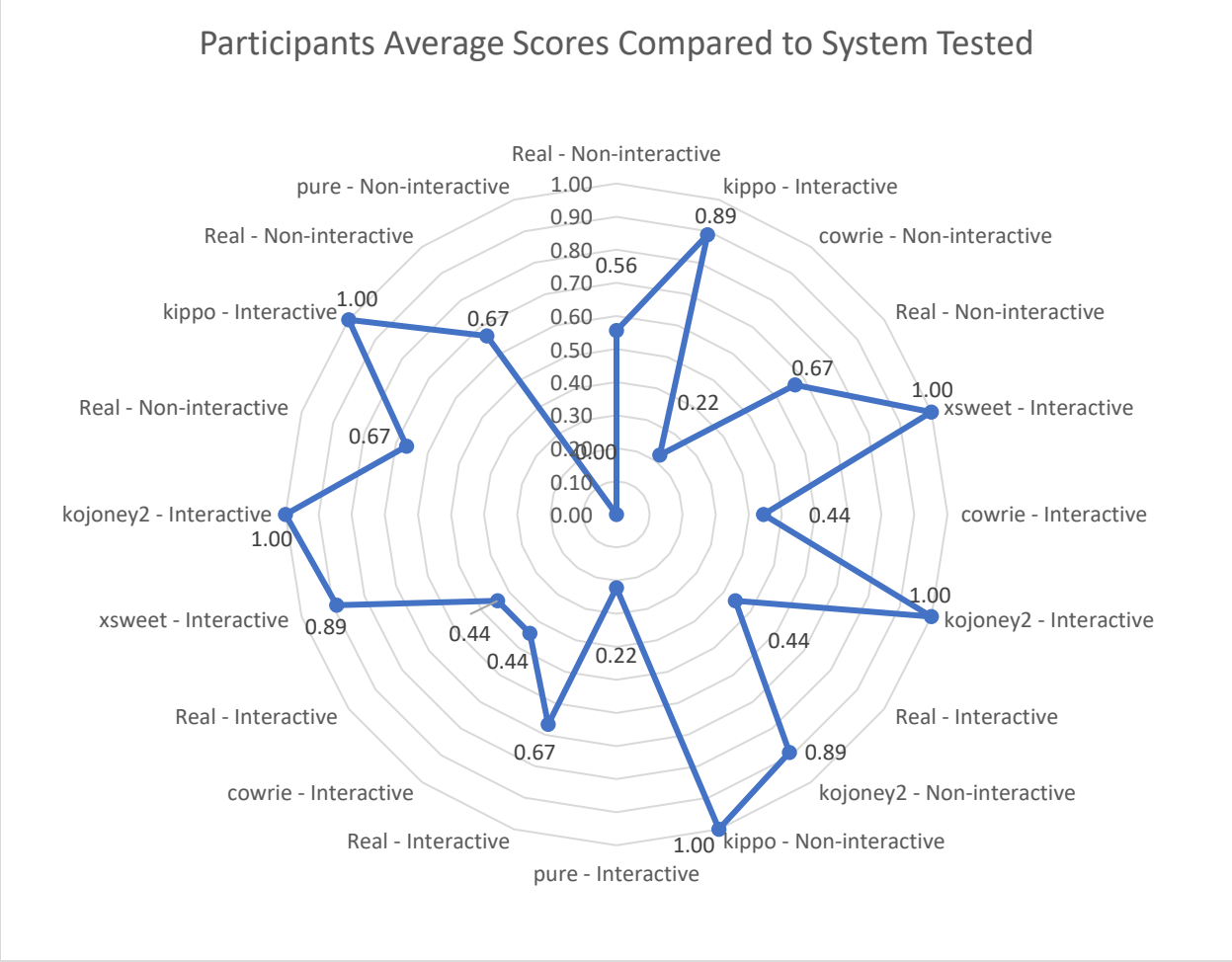


Figure 51: Bullseye for each tested system and the average participant score. The further away from the middle the higher detection success rate.

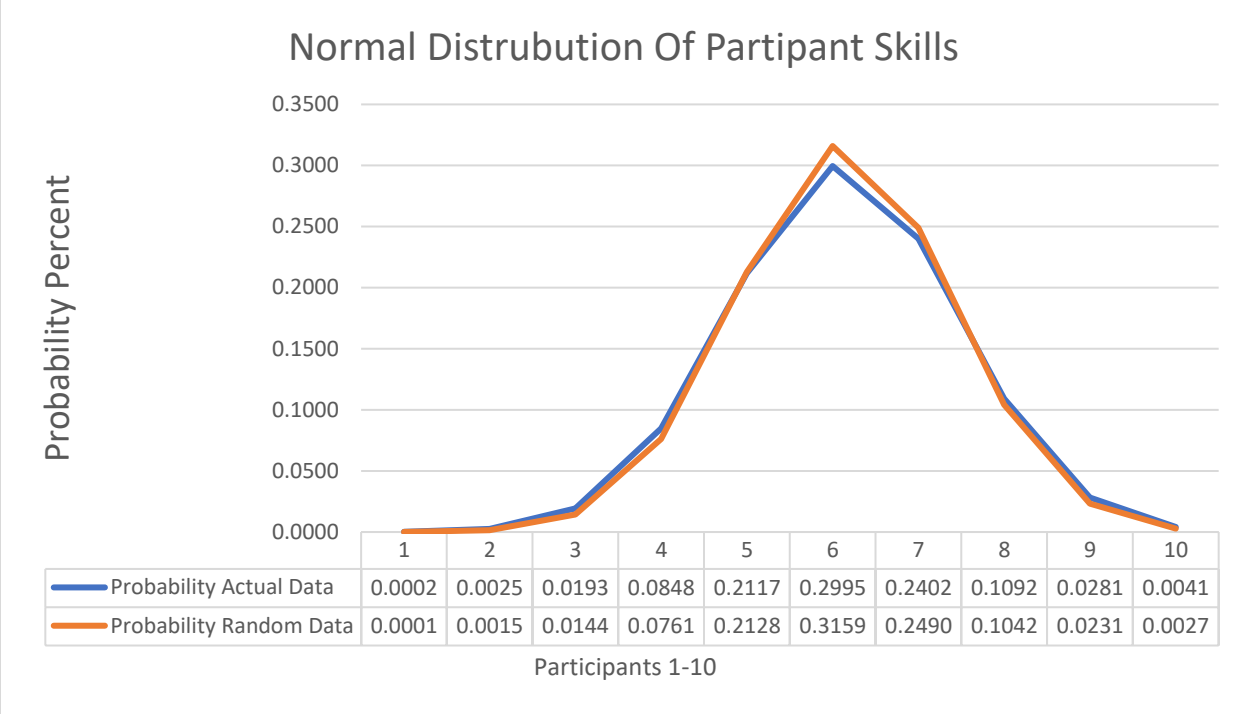
Data Analytics and Simulation

To create a larger sample size larger than the one provided in the participant-based experiment, random data is generated. The first task is to create a set of 2000 random skilled participants which range the same 1-10 seen in the actual data. To give the data realistic constraints and boundaries, parameters from the actual data such as the standard deviation 1.3271 and mean of 5.1 were used. A histogram of the actual data compared to the random data

is listed in table 17. The normal distribution producing the probabilities of achieving each skill level is seen for both the actual data and randomly generated date in figure 52. The distribution makes it easy to see what the probability is for a given score. It also shows how using the parameters from the actual data produced similar result for the randomly generated data.

Table 17: Histogram results from actual participant data and generate random data using actual data parameters.

Histogram Actual Data		Histogram Random Data	
0	*Frequency*	*0*	*Frequency*
0.5	0	0.5	0
1	0	1	0
1.5	0	1.5	2
2	0	2	10
2.5	0	2.5	21
3	0	3	70
3.5	1	3.5	103
4	1	4	170
4.5	1	4.5	241
5	0	5	287
5.5	0	5.5	329
6	2	6	274
6.5	1	6.5	212
7	1	7	152
7.5	1	7.5	69
8	0	8	38
8.5	0	8.5	12
9	0	9	7
9.5	0	9.5	1
10	0	10	1
More	0	More	0



	1	2	3	4	5	6	7	8	9	10
Probability Actual Data	0.0002	0.0025	0.0193	0.0848	0.2117	0.2995	0.2402	0.1092	0.0281	0.0041
Probability Random Data	0.0001	0.0015	0.0144	0.0761	0.2128	0.3159	0.2490	0.1042	0.0231	0.0027

Figure 52: Normal distribution of actual experiment skill scores compared to random generated data skill scores.

To create the scorecard data for each of the newly created random participants, a combination of logic and assumption was used. Figure 53 illustrates the process where we will start with the actual scorecard data. The data is first normalized where the participant results were either marked right 1 or wrong 0 versus including negative values. After the data is normalized each of the random participants skill values are aligned to an attacker profile level 1-6 as seen in table 18. Once each random participant is aligned based on their skill rating, the score *lookup table* is created.

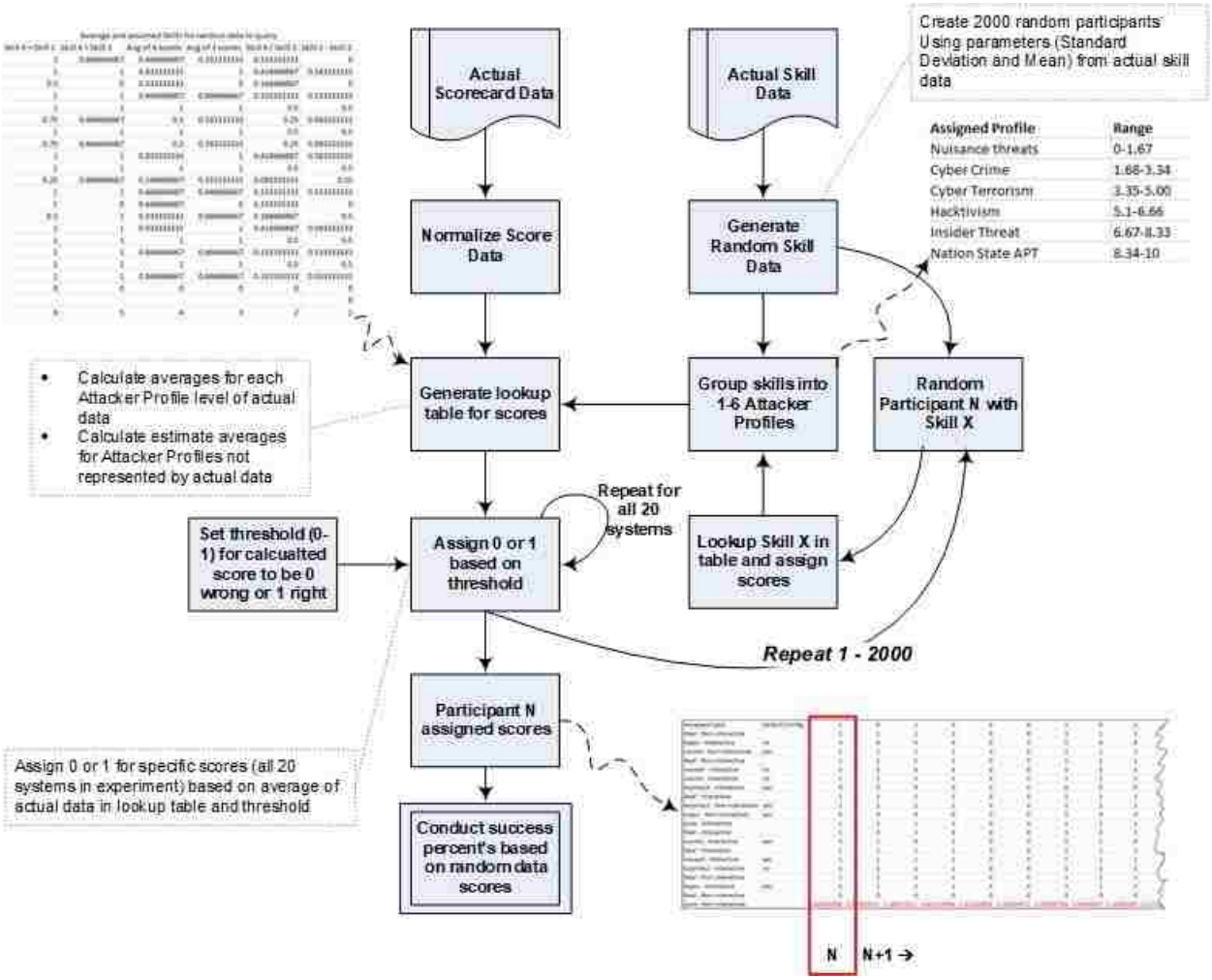


Figure 53: Dataflow of how random data is generated from actual data parameters.

Table 18: Attacker Bias perception profiles divided in skill ranges.

Skill Level	Assigned Profile	Range
1	Nuisance threats	0-1.67
2	Cyber Crime	1.68-3.34
3	Cyber Terrorism	3.35-5.00
4	Hacktivism	5.1-6.66
5	Insider Threat	6.67-8.33
6	Nation State APT	8.34-10

The *lookup table* is comprised of the average scores for each of the attacker profile levels. These scores are derived from the actual data. The attacker profiles levels are assigned to the

actual participant skill score data and matched for levels 3 and 4. To get values for the other levels 1-2 and 5-6, some logical math and assumptions were made. Referring to table 19, the *lookup table* shows that levels 3 and 4 were the average of the 3 and 4 scores from the actual data. Level 3 and 4 scores were leveraged to create all other remaining levels as a baseline. For example, the first item on the list is "Real – Non-interactive" where level 3 is 0.3333. There are many combinations that exist in creating the math behind creating the scores for the other levels that are not derived from actual data. Many approaches produce the same results while others differ slightly. In our approach set level 1 as level 3 scores minus level 2. For example, 0.33333 minus 0.33333 resulting in 0. Level 2 as level 4 divided in 2. Level 5 as level 4 plus level 1. Level 6 as level 4 plus level 2. If the score was greater than 1 then 1 was assigned. If the value was less than 0 then 0 was used.

Table 19: Random data scoring lookup table that was constructed from actual data skill levels 3 and 4. After random data looks up skills, it obtains percent for a particular system and then compares to threshold if it is a 0 or 1.

Average and assumed Skills for random data to query						
Skill 4 + Skill 2	Skill 4 + Skill 1	Avg of 4 scores	Avg of 3 scores	Skill 4 / Skill 2	Skill 3 - Skill 2	Honeypot type
1	0.666666667	0.666666667	0.333333333	0.333333333	0	Real - Non-interactive
1	1	0.833333333	1	0.416666667	0.583333333	kippo - Interactive
0.5	0	0.333333333	0	0.166666667	0	cowrie - Non-interactive
1	1	0.666666667	0.666666667	0.333333333	0.333333333	Real - Non-interactive
1	1	1	1	0.5	0.5	xsweet - Interactive
0.75	0.666666667	0.5	0.333333333	0.25	0.083333333	cowrie - Interactive
1	1	1	1	0.5	0.5	kojoney2 - Interactive
0.75	0.666666667	0.5	0.333333333	0.25	0.083333333	Real - Interactive
1	1	0.833333333	1	0.416666667	0.583333333	kojoney2 - Non-interactive
1	1	1	1	0.5	0.5	kippo - Non-interactive
0.25	0.666666667	0.166666667	0.333333333	0.083333333	0.25	pure - Interactive
1	1	0.666666667	0.666666667	0.333333333	0.333333333	Real - Interactive
1	0	0.666666667	0	0.333333333	0	cowrie - Interactive
0.5	1	0.333333333	0.666666667	0.166666667	0.5	Real - Interactive
1	1	0.833333333	1	0.416666667	0.583333333	xsweet - Interactive
1	1	1	1	0.5	0.5	kojoney2 - Interactive
1	1	0.666666667	0.666666667	0.333333333	0.333333333	Real - Non-interactive
1	1	1	1	0.5	0.5	kippo - Interactive
1	1	0.666666667	0.666666667	0.333333333	0.333333333	Real - Non-interactive
0	0	0	0	0	0	pure - Non-interactive
6	5	4	3	2	1	

Referring to the process flow in figure 53, now that the *lookup table* is complete, and the random skills are generated, the assignment of scores for each random participant N can occur. Each random participant N takes its skill and converts it to the attacker profile level then queries the scores for each system in the lookup table. A threshold from 0 to 1 is used which checks the retrieved lookup table score and determines if it is to assigns a 1 for right and 0 for wrong. In the case of these results, the mean for all 2000 random participants which was 5.1 was used as .51. This Is the same mean as in the participant-based experiment. The process of assigned scores for each system repeated for all 2000 random participants. Table 20 shows a snipped of the first 6 random participants with scores.

Table 20: Snippet of randomly generated data with assigned scores based on actual data through the lookup table.

Honeypot type	Default Config						
Real - Non-interactive		1	0	1	0	0	0
kippo - Interactive	no	1	1	1	1	0	0
cowrie - Non-interacti	yes	0	0	0	0	0	0
Real - Non-interactive		1	1	1	1	0	0
xsweet - Interactive	no	1	1	1	1	0	0
cowrie - Interactive	no	0	0	0	0	0	0
kojoney2 - Interactive	yes	1	1	1	1	0	0
Real - Interactive		0	0	0	0	0	0
kojoney2 - Non-interac	yes	1	1	1	1	0	0
kippo - Non-interactive	yes	1	1	1	1	0	0
pure - Interactive		0	0	0	0	0	0
Real - Interactive		1	1	1	1	0	0
cowrie - Interactive	yes	1	0	1	0	0	0
Real - Interactive		0	1	0	1	0	0
xsweet - Interactive	yes	1	1	1	1	0	0
kojoney2 - Interactive	no	1	1	1	1	0	0
Real - Non-interactive		1	1	1	1	0	0
kippo - Interactive	yes	1	1	1	1	0	0
Real - Non-interactive		1	1	1	1	0	0
pure - Non-interactive		0	0	0	0	0	0
		14	13	14	13	0	0
	skill	5.864835548	4.920928522	5.480536321	4.601019648	2.423249091	2.150364472
	attacker profile	4	3	4	3	2	2
	attacker profile	4	3	4	3	2	2

Using the generated data, comparisons were conducted to understand how the random participants performed. Figure 54 compares the detection success for all system types for both actual and random data. In figure 55, the interaction type is included, and the actual participants scored higher with a 9% delta in success is for interactive pure honeypot systems. For emulated honeypot systems, the variance for interactive was 7% while non-interactive was 34%. Real interactive delta was 4% while non-interactive was 22%. Figure 56 focuses solely on the different emulated honeypots where actual participants follows the same trend of scoring slightly higher than the random data. The build type is also considered with no real difference between custom and default for interactive and non-interactive was only deployed as custom. The

random participants scored worse in Cowrie compared to the actual participants, especially in the non-interactive variant.

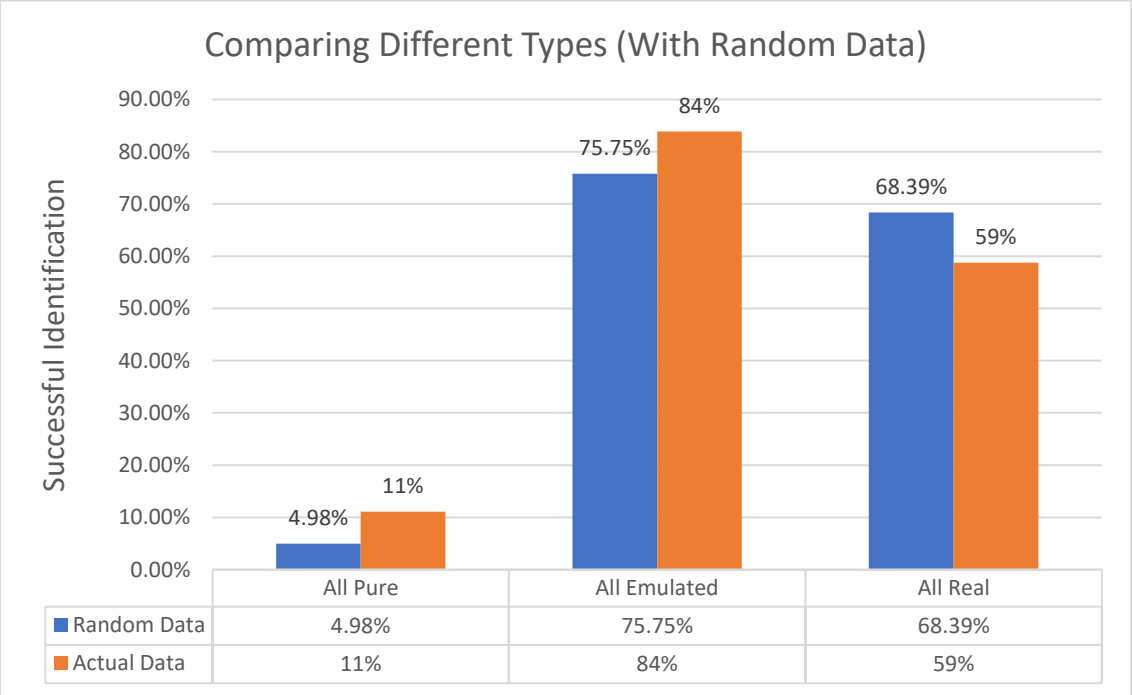
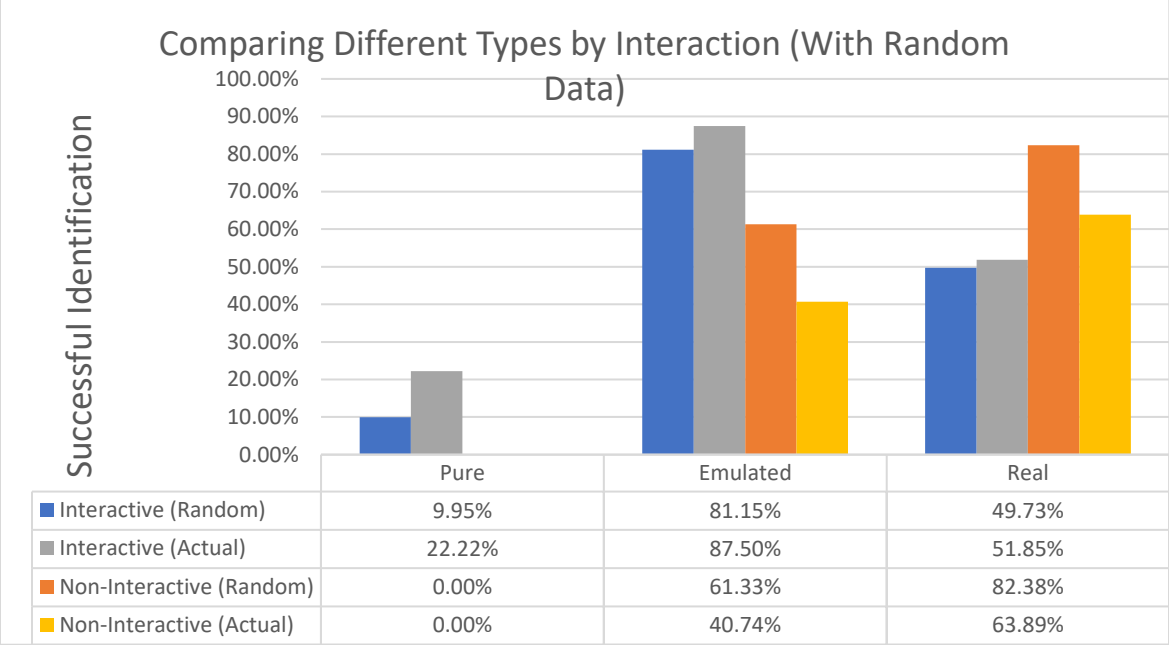
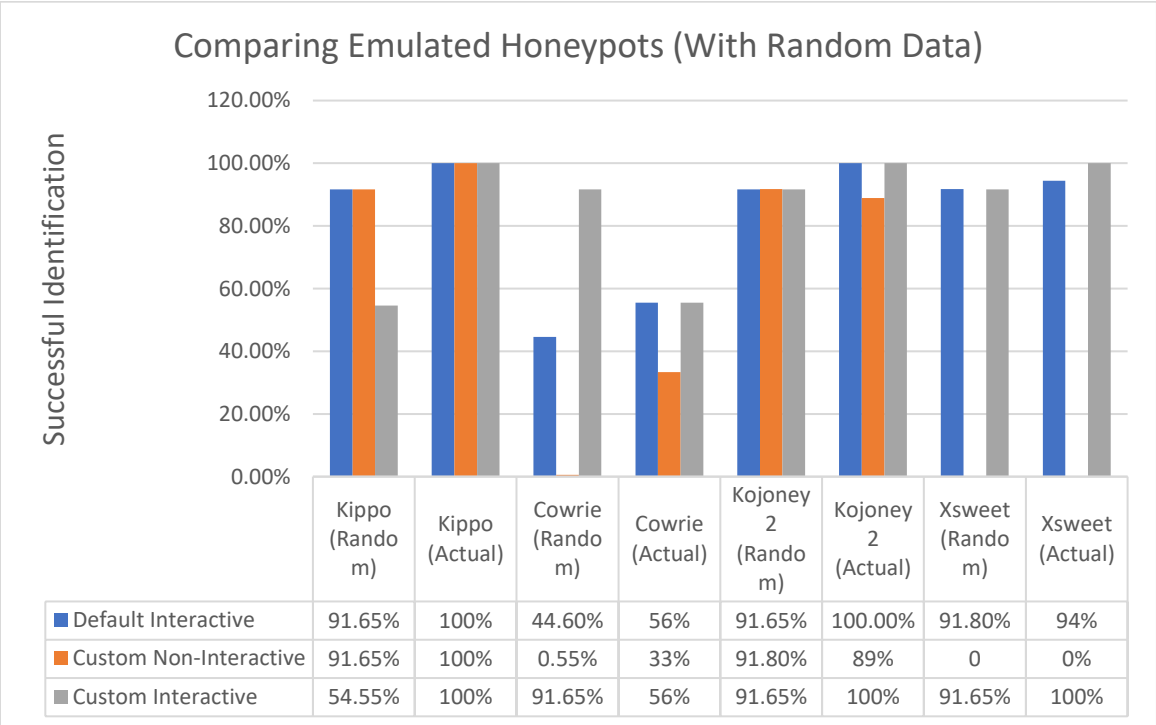


Figure 54: Comparing different system types from a successful detection perspective for both random generate and actual data.



	Pure	Emulated	Real
Interactive (Random)	9.95%	81.15%	49.73%
Interactive (Actual)	22.22%	87.50%	51.85%
Non-Interactive (Random)	0.00%	61.33%	82.38%
Non-Interactive (Actual)	0.00%	40.74%	63.89%

Figure 55: Comparing different system types and their interaction level from a successful detection perspective for both random generate and actual data.



	Kippo (Random)	Kippo (Actual)	Cowrie (Random)	Cowrie (Actual)	Kojoney 2 (Random)	Kojoney 2 (Actual)	Xsweet (Random)	Xsweet (Actual)
Default Interactive	91.65%	100%	44.60%	56%	91.65%	100.00%	91.80%	94%
Custom Non-Interactive	91.65%	100%	0.55%	33%	91.80%	89%	0	0%
Custom Interactive	54.55%	100%	91.65%	56%	91.65%	100%	91.65%	100%

Figure 56: Comparing different emulated honeypots and their interaction level from a successful detection perspective for both random generate and actual data.

Experiment Participant Data examples

Each of the experiment participants provided a scorecard which in most cases contained notes and screenshots. The below figures are examples of participant retuned artifacts with the first being an actual scorecard. Table 21 lists each system in the target scope and the decision of their testing. It also includes useful notes on the approach or TTP they used to come to their decision. This participant and several others also included a series of screenshots showing their actual command execution.

Table 21: Example returned scorecard

IP	Port	No	Suspicious	Yes	Why/explanation/screenshots
10.1.10.200	22			X	Metasploit Honeypot Fingerprint
10.1.10.201	22			X	Files do not persist with multiple sessions Metasploit Honeypot Fingerprint
10.1.10.202	22	X			
10.1.10.203	22	X			
10.1.10.204	22			X	Nmap Honeypot Fingerprint Metasploit Honeypot Fingerprint
10.1.10.205	22			X	Files do not persist with multiple sessions
10.1.10.206	22			X	Metasploit Honeypot Fingerprint
10.1.10.207	22	X			
10.1.10.208	22			X	Metasploit Honeypot Fingerprint
10.1.10.209	22			X	Metasploit Honeypot Fingerprint
10.1.10.210	22			X	"Tee: command executes upon login, cannot kill it. Google doesn't resolve. Found log files containing all shell interactions.
10.1.10.211	22	X			
10.1.10.212	22	X			
10.1.10.213	22			X	Metasploit Honeypot Fingerprint
10.1.10.214	22			X	Nmap Honeypot Fingerprint Metasploit Honeypot Fingerprint
10.1.10.215	22			X	Metasploit Honeypot Fingerprint
10.1.10.216	22	X			
10.1.10.217	22			X	Metasploit Honeypot Fingerprint
10.1.10.218	22	X			
10.1.10.219	22	X			

The first screen shot in figure 57 was a returned example of Metasploit msfconsole shell executing the detect_kippo module and detecting several honeypots as Kippo. The next screen shot in figure 58 is an interesting and unique TTP that one participant leveraged. A new vulnerability was published approximately at the experiment execution start which allowed for the querying of users on a remote SSH OpenSSH systems. Executing the exploit against all systems in scope acts as an indicator to determine if it successfully executes thus a real system. If the exploit does not work and the confidence on the retuning banner version fidelity is weak then it is most likely not OpenSSH or is an emulated operating system. Likewise, this exploit worked on the pure honeypots and gave a false indication to the tester which dismissed it as a real OpenSSH server thus not a honeypot.

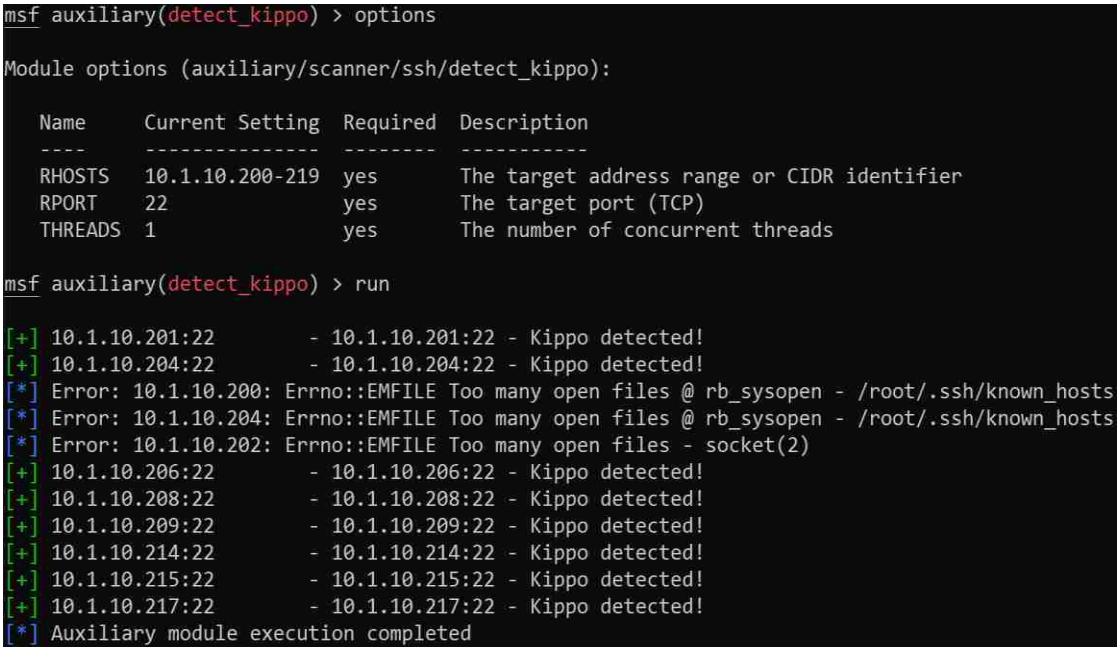


Figure 57: Participant screenshot depicting Metasploit console and detect_kippo module being executed

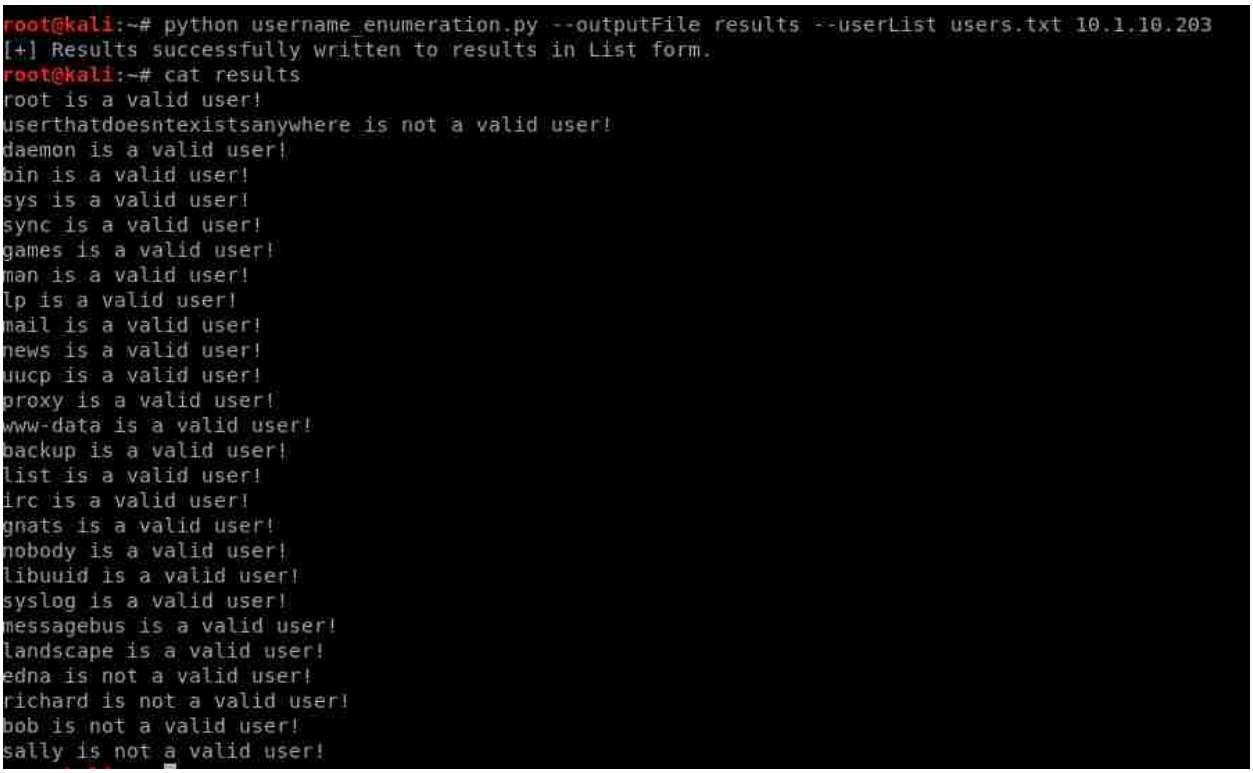


Figure 58: Participant screenshot depicting the execution of a newly released exploit that enumerates the users on a remote vulnerable OpenSSH server.

A small set of individuals were indeed able to identify the interactive pure honeypot. The main indicator was the running *tee* process which captures the screen output to file. Figure 59 shows a process list output snippet with the process running and the path to the output text file. Figure 60 shows the directory listing of the identified text file path and the output file with the date and time. The intention is to review this file later to see what sort of command and activity was conducted while an attacker was trolling the system.

Figure 59: Participant screenshot depicting the process list showing the discovery of the tee process on a pure honeypot which was used to capture shell interaction.

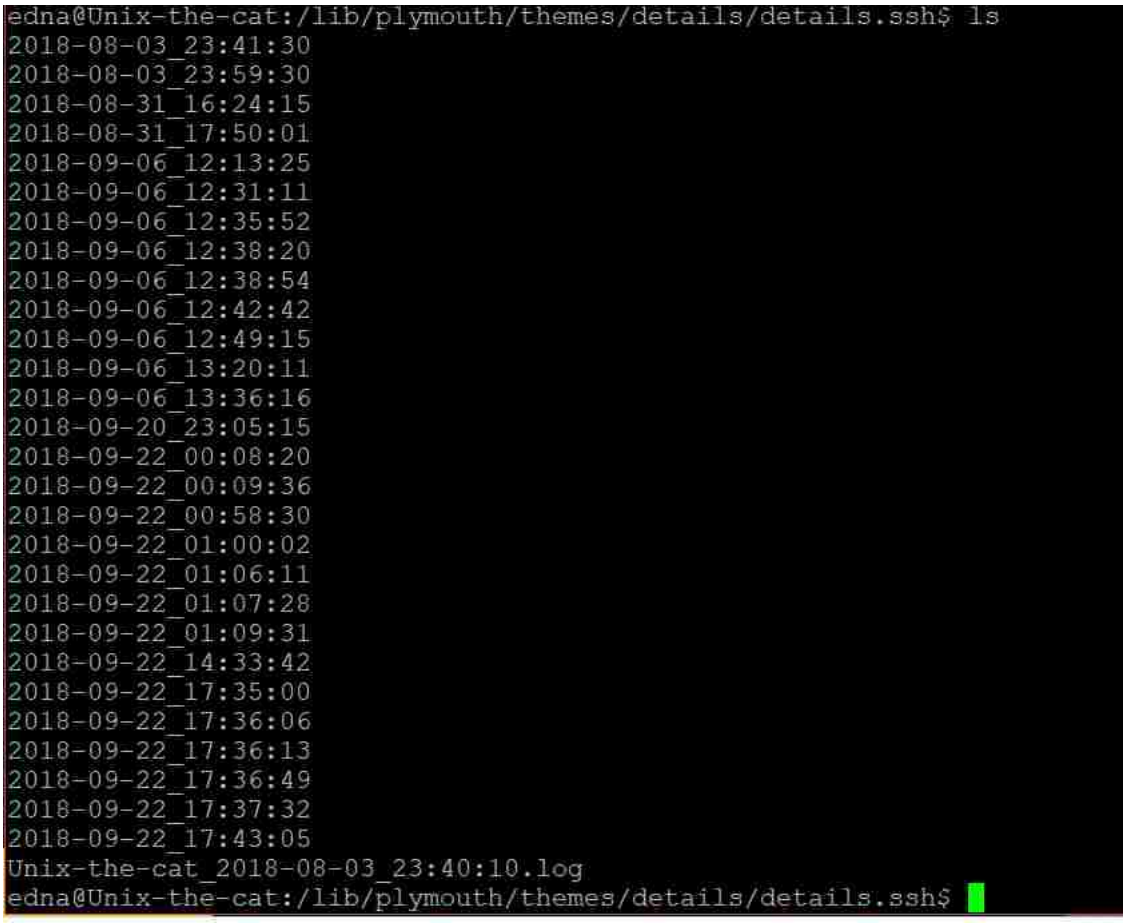


Figure 60: Participant screenshot depicting the directory in which the tee process dumps the shell input/output to file for later analysis on the pure honeypots.

Exit Survey results

The exit survey was completed by most participants and included the questions listed at the end of chapter 6. The table below summarizes the questions that are related directly to the results which can influence modeling and a framework. The first question on most common tool used was almost uniform across most participants. Nmap, Metasploit were the top mentioned. The second question on using the OSINT file and if it was helpful was also mostly a common

answer of yes but one individual said no with no other remarks. The third question on how many hours were spent varied therefore the average was taken at 4.81 hours. The forth question no which system seems the easiest had two top responses and the rest varied. The final question on what system was the hardest also had varied answered but one system was repeated by three participants. The interactive pure honeypot was also marked by one participant as the hardest system. The non-interactive pure honeypot was saw by most as a real OpenSSH server.

Table 22: Most related exit survey responses by experiment participants.

Question	Common Answer/Average
What tool did you seem to leverage the most?	Metasploit, Nmap, Wireshark, Netcat
Did having the OSINT intelligence help you in any way?	Most say yes but some responded with no or only a little
About how many hours did you spend on the experiment overall?	Avg is 4.81 hours with 2 being the lowest and 8 being the max. Most common is 4 hours
What IP was the fastest to identify and why?	10.1.10.204 (Honeypot XSweet variant 1) and 10.1.10.214 (Honeypot XSweet variant 1) were top two answers
What IP made you seem the most unsure?	10.1.10.216 (Real System Openssh6.2-1)

Results Summary

The ending results primarily focus on the honeypots which include both emulated and pure. For the first set of summarized results, table 23 lists the participants and their identifying number, average skill and total score. Table 24 lists actual participant data where each honeypot scores were averaged for the different interaction levels and build types. In addition, the skill ranges for those participants that successfully detected the honeypot were also included. For example, for Cowrie, the interactive default average was 56% with a skill range between 5.6 and

6.5. In table 25, the random and control results are considered where the percent of success is based on the normalized scores. To get a get sense of the overall likelihood of successfully detecting, the averages were taken for all three data sets. A similar trend exists where pure honeypots are the lowest, Cowrie the least detected emulated honeypot, and the remaining high.

Table 23: Overall scores and skill ranges for each type of honeypot.

Participant #	19	3	16	2	8	12	7	4	15	AVG
Ordered Participant #	1	2	3	4	5	6	7	8	9	
Skill Average	3.4	3.4	3.6	4.8	5.6	6.1	6.3	6.3	6.5	5.1
Normalized Total Score (Out of 20)	13	13	10	15	18	10	14	18	11	13.5
Normalized Emulated Honeypot Only Score (Out of 11)	8	8	8	9	11	9	9	11	10	9.2
Normalized Pure Honeypot Only Score (Out of 2)	0	0	0	1	0	0	0	0	1	.22

Table 24: Overall scores and skill ranges for each type of honeypot.

Honeypot	Interactive - Default			Interactive - Custom			Non-Interactive Default			Averages		
	Percent Success	Min Participant Skill	Max Participant Skill	Percent Success	Min Participant Skill	Max Participant Skill	Percent Success	Min Participant Skill	Max Participant Skill	Average Percent Success	Min Participant Skill	Max Participant Skill
Kippo	100	3.4	6.5	100	3.4	6.5	100	3.4	6.5	100	3.4	6.5
Cowrie	56	5.6	6.5	56	4.8	6.5	33	5.6	6.3	48	5.3	6.4
Kojoney2	100	3.4	6.5	100	3.4	6.5	89	3.4	6.5	96	3.4	6.5
Xsweet	89	3.4	6.5	100	3.4	6.5	N/A	N/A	N/A	95	3.4	6.5
Pure	N/A	N/A	N/A	22	4.8	6.5	0			11	4.8	6.5

Table 25: Honeypot scores for each experiment and data type (actuals and random) based on theri interaction and build type.

Honeypot	Actual Detection			Random Detection			Control			Average		
	Non-Interactive	Interactive Default Config	Interactive Custom Config	Non-Interactive	Interactive Default Config	Interactive Custom Config	Non-Interactive	Interactive Default Config	Interactive Custom Config	Non-Interactive	Interactive Default Config	Interactive Custom Config
Kippo	100	100	55	92	92	92	33	47	47	75	80	80
Cowrie	33	56	92	55	45	55	8	33	22	32	45	56
Kojoney2	89	100	100	92	92	92	42	56	50	74	83	81
Xsweet	N/A	94	100	N/A	92	92	N/A	62	62	N/A	83	85
Pure	0	N/A	22	0	N/A	10	0	N/A	0	0	N/A	11

To summarize the results of our experimentation, we can say that pure honeypots scored the lowest in being detected, especially for non-interactive with zero detection and interactive having only 2 out of 9 participants discovering. We observed that real systems came in second for detection success, additionally with varying results on interaction levels. Emulated honeypots placed last with the highest detection rate. Most of the emulated honeypots were successfully detected easily with security tools except for Cowrie that continues to attempt to remove detection with code fixes. Cowrie had the lowest detection scores, especially within the non-interactive where attackers could not interface with the actual filesystem versus just the login prompt and TCP service. The customization to the emulated honeypots versus the default install as downloaded from their source location did not prove to be as effective in further

preventing the detection success. Finally, the higher skilled participants (6 and above) found most of the problems and had the highest scores to include discovering the interactive pure honeypot. There were exceptions where two highly skilled participants scored lower on detection which is suspected to be an inflation of self-skills within the survey.

CHAPTER EIGHT: ATTACKER MODELING AND DECEPTION FRAMEWORK

In this research we focus on sampling deceptive technologies with the secure shell service to determine their effectiveness in being detected. We observed our experimentation outcome for a small number of participants. Results indicate honeypots mechanisms such as the type, interaction, and configuration dictate the potential success in its ability to be detected as a deception while considering attacker threat. For this we present a deception framework to describe these results and conclusions. This new framework will observe how a planned and deployed deception of varying mechanisms interacts with the system, attackers, and leverages cyber intelligence for alerting. As deception mechanisms differ, the likelihood of subverting detection rests partially in the attacker skills level, in the TTPs they execute, and the cyber intelligence to deconflict.

To understand the attacker, we first present the Attacker Bias Profile Perception model with the profiles discussed in chapter 2. In this model, we illustrate the various attacker levels with their likelihood of success and threat impact. We then use this to present the skill levels of each participant in the experiment and the success of detection for each of the honeypots. Next, we present a set of models for abstracting the deception process. The three referenced deception models discussed in the chapter 2 are reviewed and presented in a simplified manner representing their commonalities and individual strengths. This abstraction is then used as a foundation for the deception framework.

Attacker Bias Profile Perception Model

We present the attacker bias profile perception model based on the "bias" concept found in a dissertation by M. H. Almeshekah [17]. The concept proposed that biases are the cornerstone component to the success of any deception-based approach. The attacker needs to be presented with a plausible deceit to fit how they are to be perceived. Our original model is used to perceive and derive intuitive likelihood and impact ratings that can be further characterized and used for comparison of overall attacker skill and deception effectiveness.

The model represents each attacker bias profile, outlined in chapter 2, across the average and empirical threat impact and likelihood of successfully detecting [6]. We define three threat impact levels where level 1 is low, 2 moderate, and 3 severe. Each is assigned a likelihood percent for the three impact levels individually. The likelihood is the ability to successfully complete an attack within a specific threat impact level. The model is not meant to represent an individual attacker with specific skills but more so the overall threat they can deliver. To have data points useful for comparison against actual experiment data, assumptions where made based on perceiving the attacker profiles past incidents, professional experience, and educated intuition. These values are listed in table 26 and as an example, Annoyance is 50% likely to succeed at threat impact level 1, 5% at level 2, and 0% at level 3.

Table 26: Threat likelihood and Impact profile assumptions

Attacker Profile	Threat Likelihood Level 1	Threat Likelihood Level 2	Threat Likelihood Level 3	Threat Likelihood Level n	Threat Impact Index (0-3)	Overall Likelihood of success	Attacker Skill Level
Annoyance	50%	5%	0%	…	0.55	18.33%	1
Cyber Crime	60%	30%	35%	…	1.25	41.67%	
Terrorism	70%	45%	45%	…	1.60	53.33%	2
Hacktivism	80%	60%	50%	…	1.90	63.33%	
Insider	90%	75%	55%	…	2.20	73.33%	3
Nation Sponsored	100%	100%	90%	…	2.90	96.67%	
… …	…	…	…	…	…	…	

To obtain the overall threat "impact index" (0-3) we sum all three of the percent values at each impact level to compute how far it penetrates that specific level. Continuing with the Annoyance example, the threat impact index, seen in equation 1, is calculated as .50+.05+0 equaling .55 out of 3. To obtain the overall likelihood percent we use equation 2 and divide the threat impact index by the total number of levels. Annoyance likelihood percent is (.50+.05+0)/3 = .183 meaning at the threat level .55, there is an 18% likelihood of success. In figure 45 we further illustrate the model in an easily decipherable form.

$$\sum_{k=1}^{\#\ of\ levels}(\text{likelhood level k}) \quad (1)$$

$$\left(\sum_{k=1}^{\#\ of\ levels}(\text{likelhood level k})\right)/\#\ number\ of\ levels \quad (2)$$

$$\text{score/total possible score} \quad (3)$$

$$\text{(skill average/10) * 3} \quad (4)$$

(((Min Skill + Max Skill)/2)/10) *3 (5)

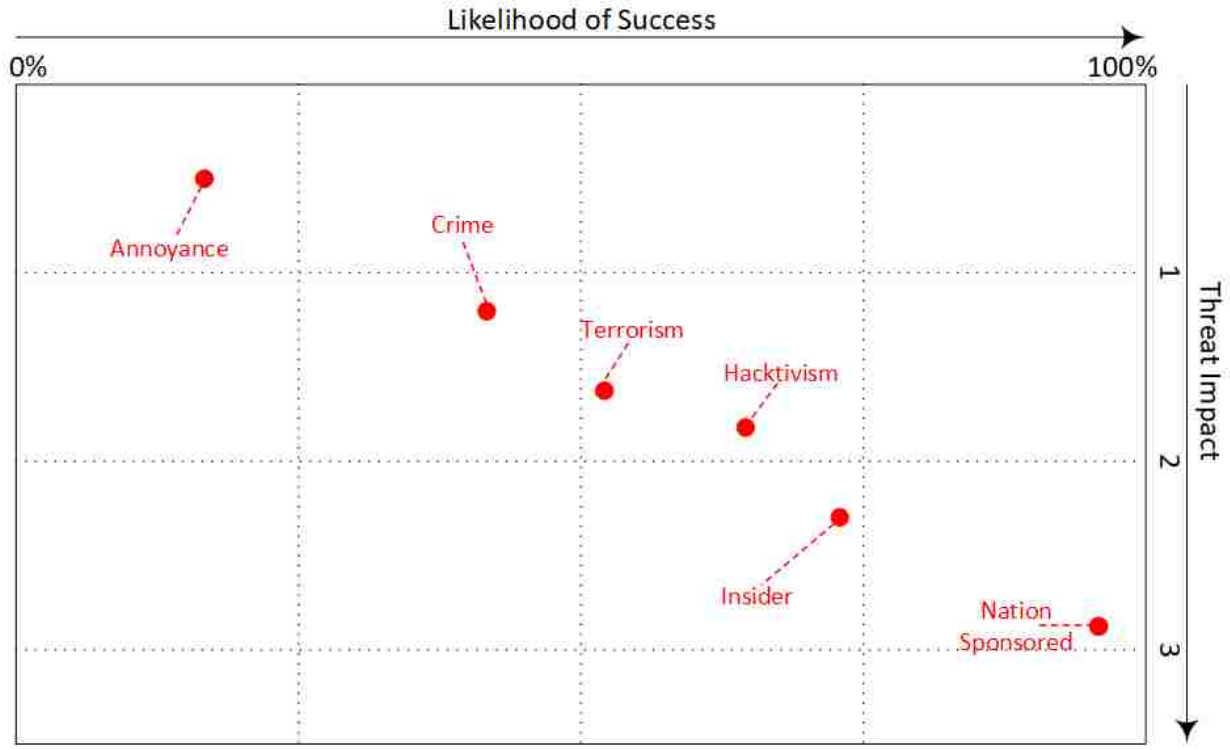


Figure 61: Abstract characterization of attacker bias profiles measured against overall likelihood of success and overall threat impact.

The model numbers can be adjusted to represent newfound research or intelligence on any attacker profile capabilities and resulting threat. In addition, the model is dynamic enough to grow to a changing landscape. Signified by the "…", both threat likelihood levels and attacker profiles can be added [9].

Using this model approach, we can gain insight into how each participant performed in their scores following (3) and their overall skill impact using (4). This is overlaid and compared to the overall attacker bias profile threat levels. As seen I figure 62, referring to participant number 2 as an example, their average likelihood score following (3) is 75%. Their impact is approximately between levels 2 and 3 while their likelihood of success concentrates mostly on

levels 5-6. Participants 4 and 8 had the largest likelihood score of 90% and impact skills score of 6.3 or 1.89 using (4) and 5.6 or 1.68 respectively. 1-3. In both figures, the skills remain the same thus the impact is unchanged. Participant 4 had the highest skill impact and highest likelihood of success.

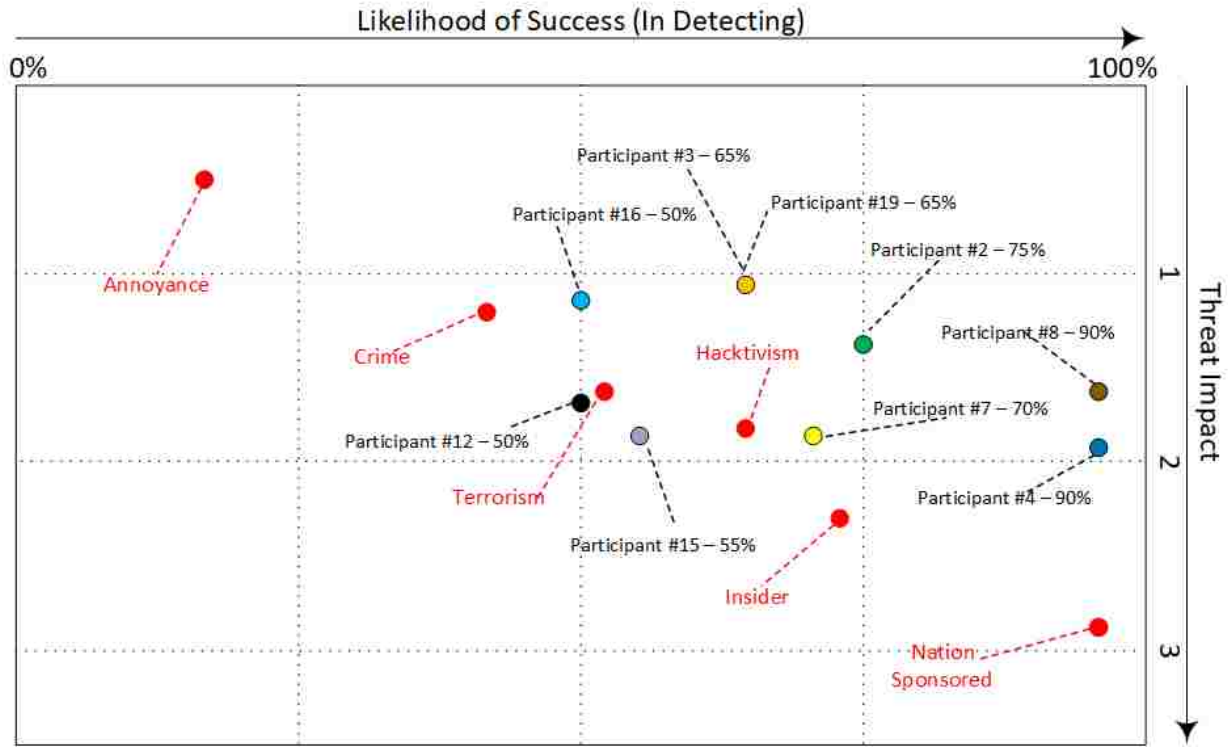


Figure 62: Abstract characterization of experiment participants and their average skill impact level and system scores likelihood compared to the attacker bias profiles for ability to detect.

Applying the attacker bias profile model to the honeypots used within the experimentation is seen in figure 63. Here the likelihood is the success of not being detected or more so subverting the attacker. For this model, the threat impact is represented in (5) as the skill range from minimum, the average, and the maximum in successfully detecting the honeypot. For example, the pure honeypot had a minimum impact score of 1.44, and average of 1.70 and a maximum skill of 1.95. In the situation where, multiple variants of a honeypot were

not all detected, in this case the non-interactive pure honeypot was never detected by any participants, assumptions need to be made on the possibility of it being detected in the future. This is represented by the red dashed line from the maximum skill to the max impact of 3. For XSweet, the non-interactive honeypot was not a part of the experiment target scope so there is an unknown possibility. Equally the red dashed line represents the possibility where in this case it ranges from 0-3 since no one had an opportunity before.

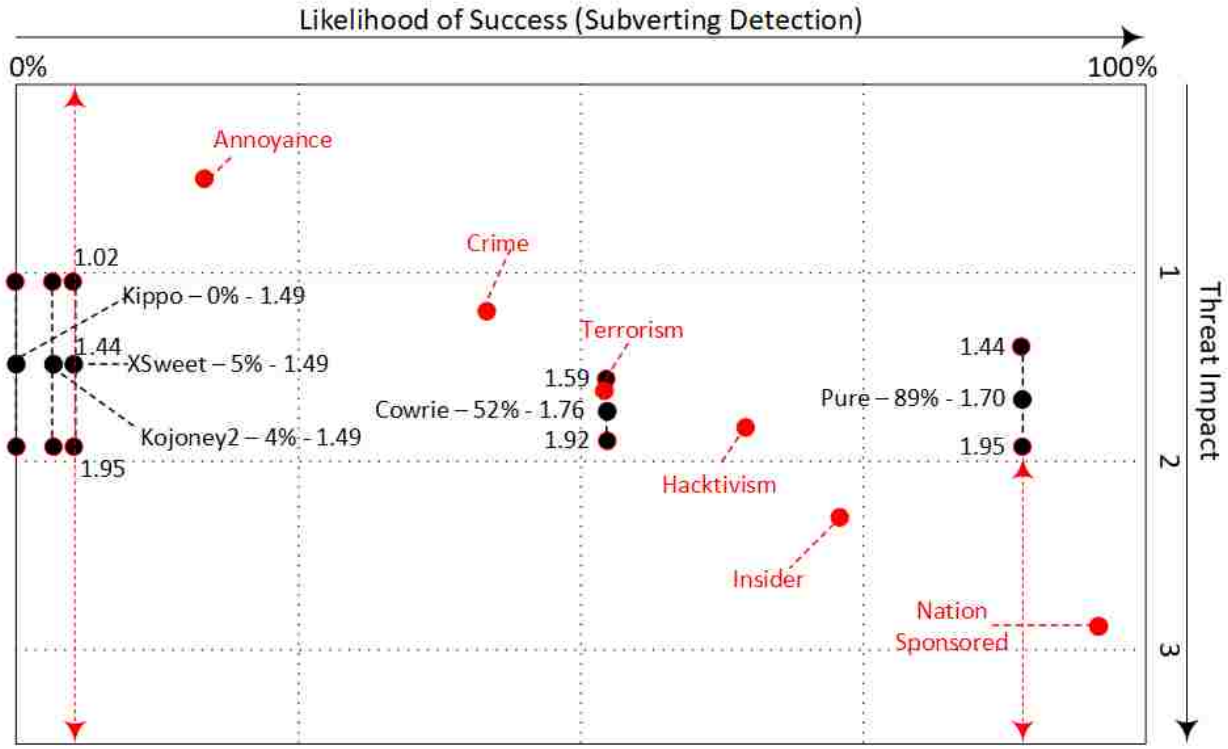


Figure 63: Abstract characterization of attacker bias profiles measured against honeypot overall likelihood of successfully subverting detection and ranging threat impact levels seen.

Abstracting of Deception Models

The three deception models discussed in the background material in chapter 2 is viewed in abstract terms by simplifying their existence to a basic elementary view. Each of the

deception models share common elements which make up the general premise and main idea of what the deception is supposed to do. Granted each of the models have their own unique elements where some are more advanced and specific than others. These similar elements can be grouped into three sets that include Plan & Deploy, Monitoring, and Analysis. Using this grouping, the Basic model is shown in Figure 64. The many details of planning and implementing the deception to be "Ready" for deployment is characterized as Plan & Deploy. This next major abstraction of the three deception models is Monitoring which involves the deception being in "Production" and ready for interaction with adversarial actors. Anyone may interact with a deception if visible to the outside world. During this time the deception is observed for feedback and analyzed for potential intelligence links. The last set or the end in some cases for the deception is Analysis where the "Determination" is made on what occurred and if the deception was successful. Analysis may result in further Monitoring and Plan & Deploy shown in the abstract model [8].

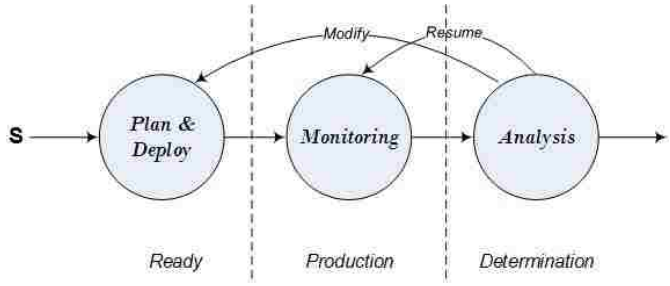


Figure 64: Abstract representation of the Basic Model

Other deception models in abstract form reveals the same fundamental generalization with different flow of events. In the Incorporated model, the deception as seen in figure 65 ends after the determination is made based on the feedback channels being analyzed. For the D&D model

being spiral in nature, the deception can return to the Plan & Deploy state and re-start the process once again as seen in figure 66 [8].

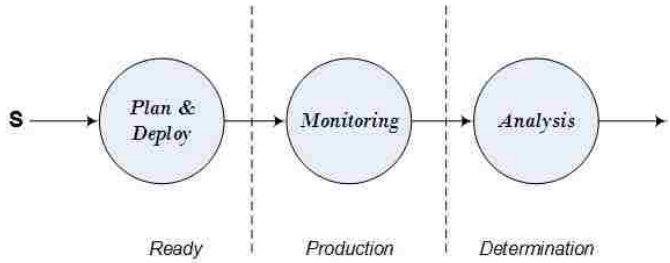


Figure 65: Abstract representation of the Incorporated Model

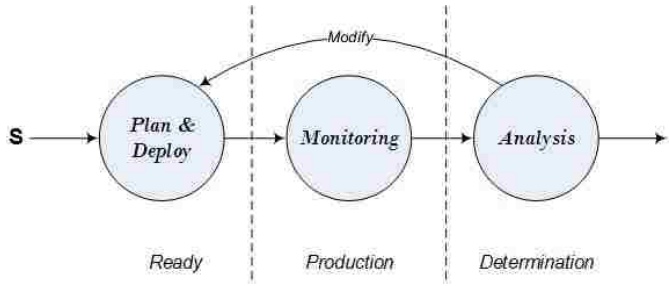


Figure 66: Abstract representation of the D&D Model

Framework for Deception

It is proposed in this section to illustrate a deception framework system state model focusing specifically on the honeypot characteristics, attackers, and intelligence. The progression of the state model will be used to describe and illustrate and outline the procedural components of a deception with interference from attackers. This section expands on prior work in modeling the state of system with deception.

As a deception is deployed it inherently changes some characteristics about the system. As computing systems are extremely complex, especially at the operating system level, many variations during a deception can theoretically occur. For example, System Software and

Services, System Activity, System weaknesses, System configuration and several others may be considered. To simplify this scope, three elements are considered for scrutiny within a system which are alpha (α), beta (β), and mu (μ). These three scrutiny elements are used for developing our abstract model of the state of the system. Any number of elements could also be considered for inclusion but for simplicity in our example, only three suffice. For example, alpha could be system files, beta system processes, and mu network traffic. Initially, the system will be in a steady state which we call "Default State". The action plan for deception changes the default state to a new state. This action plan is identified as "Plan/Build Transition" which transitions the deception to the "ready state". The deception is then deployed. Actions by attackers may trigger an alerts and changes to the system state. Each of the concepts are formalized below and is illustrated in figure 67.

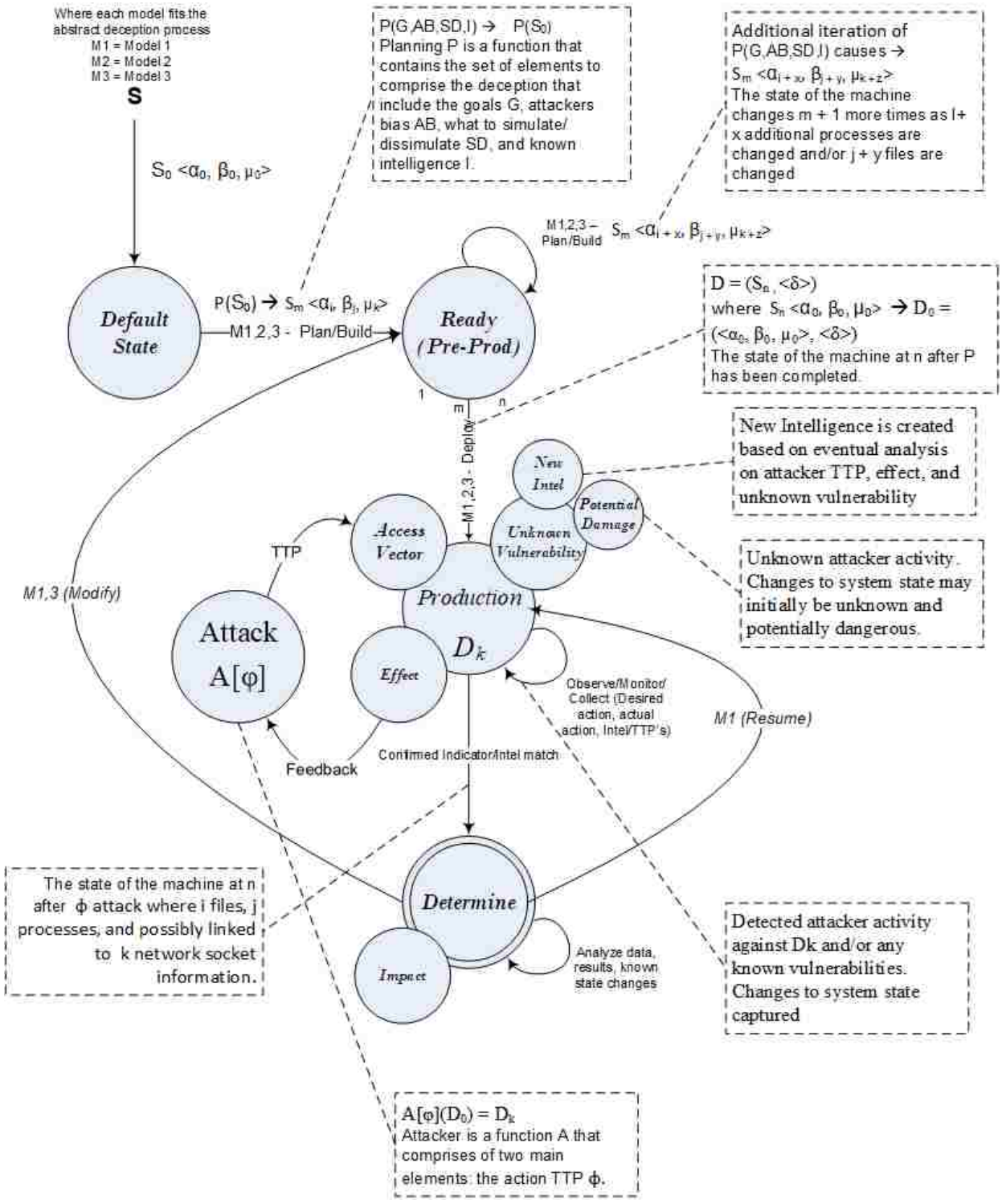


Figure 67: Deception Framework System State Model

Default State

The system prior to implementing a deception is in a *default state* meaning the posture of the system in its current configuration regardless of what else already exists. As the deception is initially implemented, various components changed thus the component is updated, the default state deviates. Other factors such as operating system variations, user changes or a previous undetected compromise can cause subtle changes skewing what is understood as the default state. For simplicity, we will limit changes to the system state based only on what we know namely the deception being implemented and the attacker actions. In (5) we look at the default state as S_0 where S represents the System at state 0 with the three components set to 0 as a baseline.

$$S_0 = <\alpha_0, \beta_0, \mu_0> \quad (5)$$

Plan/Build Transition

The state machine takes in to account the minimum common planning and deploying elements of each of the three models and represents them by the function P where

$$P(S_0) = S_m \quad \text{and} \quad (6)$$
$$S_m = <\alpha_i, \beta_j, \mu_k> \quad (7)$$

The function P is based on some of the following information:

- G - Deception Goals and Objectives
- AB – Attackers bias or knowledge of the deception target
- SD – What the deception is supposed to simulate or dissimilate. The storyline of the deception.

- I – Incoming known intelligence, indicators, signatures that alert to the presence of attack and their TTPs.
- T – Type of honeypot such as emulated or pure
- L – Interaction level ranging from high to low.

The planning and eventual deployment function P transitions the default state to the ready state which causes the system state to change from 0 to m after possible i, j, and k changes of three scrutiny elements. These changes are shown in (7).

Ready State

During the *ready state* the deception planning commences with incorporating any additional changes shown in (8). Variants of the planned deception may occur as configuration commences.

$$S_m = <\alpha_{i+x}, \beta_{j+y}, \mu_{k+z}> \quad (8)$$

Deploy Transition/Production State

Once the planning is complete the deception variant can be deployed indicated by the component $< \delta>$ shown in (9). The state of the machine is now at n represented by (10) where it is no longer being altered by the deception planning but by potential outsiders interacting with the deception and system. These changes from the default state 0 to the deployable state n now include the implemented deception. The state D_0 shown in (11) includes delta which signifies the deception is in play on the new system state n which is shown as initial state of deception.

$$D = (S_n, <\delta>) \text{ where} \quad (9)$$
$$S_n = <\alpha_n, \beta_n, \mu_n> \quad (10)$$
$$D_0 = (<\alpha_0, \beta_0, \mu_0>, <\delta>) \quad (11)$$

The production state also has the attacker access, attack effect, and unknown vulnerability sub components [10]. The attacker accesses the system in multiple way depending on what is available to them from an interaction and technological perspective (TCP ports, etc.). As the attacker accesses the system it can produce effects that ultimately give clues about the actions taking place. The attacker receives some sort of feedback on their access attempt or success. For example, the access could be a port scan and the effect a TCP feedback response of the port being open, and a possible log enter of the attempt. There exists a chance the attacker accesses the system in a manner that is unknown or zero day. This indeed will hopefully produce useful intelligence on how the attack occurred unless the attacker has gained access to the honeypot and circumvented the ruse. The attacker could delete logs, evidence, destroy the honeypot or possibly use it to attack other systems.

The Attacker and the Problem

Depending on the type or purpose of the deception, it may be welcoming to see an attacker or quite the opposite if the deception was intended to stray the attacker away. For example, an attacker tries to gain access to a system and the deception attempts to deceive and confuse them. Another instance is the deception is used for research to understand attacker TTPs and generating new intelligence or ultimately indicators. With the use of indicators introduced in the planning part of the deception, when attackers perform various nefarious activities they may match an indicator or signature and trigger an alert to their presence and will bring the system to determine

state. In this case, an analysis may provide information about the attacker. The function A in (12) is used to denote attack(s) where phi (φ) is the attacker TTP. The The attack(s) are against the system state in (11) where the state could possibly be changed by the attack resulting D_k.

$$A[\varphi](D_0) = D_k \tag{12}$$

Indicator Transition/Determination State

As known adversary TTPs match intelligence provided during planning, it is then conceivable to determine the impact and if the correlation causes a "success" or "Failure" for the deployed deception. With this ideal situation it is possible to know if the deception is effective or credible like intended or planned. The activity generated by the attacker or anyone else for that matter is de-conflicted. As unknown amounts of activity traverse the deception, it is hard or in most cases impossible to tell if this activity caused the deception to be successful or what the attacker is really executing. The impact can range based on the attack. In most cases the overall impact is minimal and just noise and possible new intelligence being collected. For the less common cases, the impact can be severe as the attack can counter, kill, or use the deception to their advantages, hide evidence, and more.

Case Studies

We use the deception framework to present some of the participant-based experiment results. Each deception honeypot starts at an initial default state on a clean bare Ubuntu 14.04.5 system represented as (5). As the honeypot is planned, the function P in (6) is defined as the

goal, attacker bias, type, what to simulate/dissimulate, intelligence, type and interaction. Many of the deceptions deployed exhibit similar values for P.

As the honeypot is downloaded, installed and configured on the server, S_m in (7) represents the incremented changes to the system as it transitions for the default to ready state. While the honeypot is pre-production and resides in the ready state, additional changes such as the configuration files, software packages, and more are represented by further increments to S_m in (8). Once the honeypot is ready for deployment, (9) represented as D contains <δ> to indicate a deception variant transitions to the production state and subject to attack. The state of the system is understood and set as S_n in (10) and increments as changes occur with attacker access attempts. The final honeypot deception is represented as D_0 in (11) or as an abstract instance D_k in (12). As the deception was subject to attack in the experimentation, the access, effects, and impact is evaluated with the deception transitioning from the production to determination state, and in most cases right back to production for the next attacker possible interaction.

Case One – Kippo Emulated honeypot

The Kippo honeypot was easily detected and thus had a low success rate for evading detection across all experimentation. This was evident for participants of all skill. All variants of Kippo exhibited the same behavior but for this example we picked the low interaction custom build using the range system 10.1.10.209. The function P is defined as having:

- G – To measure the effectiveness of the SSH honeypot from detection evasion
- AB – Attacker profile levels 2-4
- SD – Simulate SSH and underlying operating system shell.

- I – All intelligence for attacks related to known CVE, shared repositories, opensource, intrusion detection packages, and more
- T – Emulated
- L – Low Interaction

Access

The attacker used Metasploit and executed the detect_kippo module. The module looks for a "Protocol mismatch" and "bad packet length" in the TCP response. Phi in (12) is the Metasploit module that looks for the protocol mismatch. The below screenshot in figure 68 depicts the captured network traffic from the Kippo honeypot, identifying the attackers attempt to detect Kippo.

```
00:00:13.488815 IP 10.1.10.209.22 > 10.1.24.176.46525: Flags [P.], seq 3589258173:358
9258192, ack 3313325588, win 227, options [nop,nop,TS val 43279129 ecr 1381838909], l
ength 19
        0x0000:  4500 0047 d886 4000 4006 2aa8 0a01 0ad1  E..G..@.@.*.....
        0x0010:  0a01 18b0 0016 b5bd d5ef bbbd c57d 5614  ............}V.
        0x0020:  8018 00e3 37bc 0000 0101 080a 0294 6319  ....7.........c.
        0x0030:  525d 303d 5072 6f74 6f63 6f6c 206d 6973  R]0=Protocol.mis
        0x0040:  6d61 7463 682e 0a                        match..
00:00:13.488963 IP 10.1.10.209.22 > 10.1.24.176.46525: Flags [F.], seq 3589258192, ac
k 3313325588, win 227, options [nop,nop,TS val 43279129 ecr 1381838909], length 0
        0x0000:  4500 0034 d887 4000 4006 2aba 0a01 0ad1  E..4..@.@.*.....
        0x0010:  0a01 18b0 0016 b5bd d5ef bbd0 c57d 5614  ............}V.
        0x0020:  8011 00e3 37a9 0000 0101 080a 0294 6319  ....7.........c.
        0x0030:  525d 303d                                R]0=
```

Figure 68: TCP network traffic capture showing detect_kippo module execution

Effects

There is little to no effect to the Kippo honeypot. The effect or feedback to the attacker was the success of the detect_kippo module execution alerting to the presence of a deception. The changes to the system such as a log entry being created, and the network traffic recorded is captured in D_0 using (11) as alpha, beta and mu increment.

Impact

The impact was the ability for the attacker to recognize the system was a deception and cease all other activities. No impact to the actual system and its ability to function was detected. The attacker traffic in figure 68 was detected as known TTPs and the determination was to resume the honeypot deployment as is.

Case Two – Cowrie Emulated Honeypot

Cowrie exhibited a better detection evasion across the empirical and participant-based experimentation than any other emulated honeypot. On average the detection was 52% for Cowrie for all three variants. This was largely due to the code fix that removed the ability for known security tools to detect any known flaws in the honeypot TCP traffic responses. Thus, the Metasploit detect_kippo module that worked on Kippo and once worked on Cowrie no longer functioned. The average skill to detect was also higher than for Kippo. For this case we look at the high interaction custom variant of Cowrie which also included leaked credentials (edna/connie) to login. The range IP for the honeypot is 10.1.10.205. In this case the function P is as follows:

- G – To measure the effectiveness of the SSH honeypot from detection evasion
- AB – Attacker profile levels 1-3
- SD – Simulate SSH and underlying operating system shell.
- I – All intelligence for attacks related to known CVE, shared repositories, opensource, intrusion detection packages, and more
- T – Emulated
- L – High Interaction

For this case we will look at two attacks. The first attacker (A1) is a lower skill scoring participant that failed in detecting the honeypot. The second attacker (A2) is higher skilled and used an unknown TTP which enabled a successful detection. The effect and impact will follow the same number for each attacker.

Access

A1 accessed the system by conventional means and logged in to the shell interface with the given leaked credentials. Phi in (12) is normal SSH login and interaction traffic. The access to the system by executing commands is seen in figure 69.

```
 channel open
ns@openssh.com request
 pty request: 'xterm' (24, 80, 0, 0)
 Terminal Size: 24 80
 request_env: LANG=en_US.UTF-8
 getting shell
 Opening TTY Log: log/tty/20181013-081832-fbbc0f7e4d8d-81.log
 CMD: clear
 Command found: clear
 Reading txtcmd from "share/cowrie/txtcmds/usr/bin/clear"
 CMD: ls -al
 Command found: ls -al
 CMD: pwd
 Command found: pwd
 CMD: cd home
 Command found: cd /home
 CMD: cd edna
 Command found: cd edna
 CMD: ls -al
 Command found: ls -al
 CMD: cd /tmp
 Command found: cd /tmp
 CMD: cd /home/richard
 Command found: cd /home/richard
 CMD: ls -al
 Command found: ls -al
 CMD: clear
 Command found: clear
 Reading txtcmd from "share/cowrie/txtcmds/usr/bin/clear"
 CMD: ls -al
 Command found: ls -al
 CMD: exit
```

Figure 69: Snippet of grepped Cowrie honeypot log showing simple commands being executed by a lower skilled participant. This participant did not detect the system as a honeypot.

A2 accessed the system using a newly released (as the experiment range came online) exploit code, Common Vulnerability Exposure (CVE)-2018-1547, which allowed for the enumeration of user accounts on remote SSH server [93].

Effects

There were no effects to the actual honeypot. The changes to the system by A1 and A2 such as log files being created, and the network traffic recorded is captured as alpha, beta and mu which causes D_0 to increment.

A1 received feedback by the effects on the system that gave no indication of a deception in play.

A2 received feedback by the lack of a response in the exploit producing enumeration results. The same exploit executed by A2 on a different real system is seen in Figure 58.

Impact

A1 – The impact was minimal as the attacker was not able to detect the system as a deception. The effects by the commands executed in figure 70 did not trigger an alert or match within the intelligence analysis. The deception resumed.

A2 – The impact was the attacker detecting the system as abnormal in its inability to execute the exploit code. Since the code executed just fine on real systems, the ones where it failed were flagged as deceptive. No impact to the actual system and its ability to function was detected. The attacker traffic was added to the known TTPs and the determination was to resume the honeypot deployment as is. In this case phi in (12) was unknown to the intelligence within the deception system and was caught after analysis was conducted. The deception resumed as further research was conducted on how to change the response to the attacker for the honeypot software.

Case Three – Pure Honeypot

The pure honeypots displayed the best results regarding evading detection. The average detection was 22% and the highest skilled participant contributed to detecting the honeypot. There were two variants with one being low interaction and the other high. For this case the high interaction version was studied, 10.1.10.210, and P is defined as following:

- G – To measure the effectiveness of the SSH honeypot from detection evasion
- AB – Attacker profile levels 3-6
- SD – Simulate SSH and underlying operating system shell.
- I – All intelligence for attacks related to known CVE, shared repositories, opensource, intrusion detection packages, and more
- T – Pure
- L – High Interaction

For this case we will look at three attacks. The first attacker (A1) is a lower skill scoring participant that failed in detecting the honeypot. The second attacker (A2) is low to medium skilled that attempted to launch exploit code and failed in detecting the system as a honeypot. The third attacker (A3) is the highest skilled participant that executed the right commands and was able to detect the system as a honeypot.

Access

A1, A2 and A3 all accessed the system using normal convention means by logging into the shell terminal by leaked credentials. Phi in (12) is normal SSH login and interaction traffic. Figure 70 depicts the commands executed by A1, figure 71 for A2 and figure 72 for A3.

```
edna@Unix-the-cat: ~^Gedna@Unix-the-cat:~$ uptime
edna@Unix-the-cat: ~^Gedna@Unix-the-cat:~$ hostname
edna@Unix-the-cat: ~^Gedna@Unix-the-cat:~$ uname -r
edna@Unix-the-cat: ~^Gedna@Unix-the-cat:~$ uname -a
edna@Unix-the-cat: ~^Gedna@Unix-the-cat:~$ date
edna@Unix-the-cat: ~^Gedna@Unix-the-cat:~$ whoami
edna@Unix-the-cat: ~^Gedna@Unix-the-cat:~$ w
edna@Unix-the-cat: ~^Gedna@Unix-the-cat:~$ exit
```

Figure 70: Snippet of Pure honeypot log showing a lower skilled participant shell interaction. Participant did not detect as a honeypot.

```
anonymous@Unix-the-cat:/$ wget https://www.exploit-db.com/raw/40871/
anonymous@Unix-the-cat:/$ nano 40871
anonymous@Unix-the-cat:/$ ls
anonymous@Unix-the-cat:/$ cd ~
anonymous@Unix-the-cat:/$ vim
anonymous@Unix-the-cat:/$
anonymous@Unix-the-cat:/$
anonymous@Unix-the-cat:/$ ls
anonymous@Unix-the-cat:/$ wget google.com
anonymous@Unix-the-cat:/$ wget http://www.google.com/
anonymous@Unix-the-cat:/$ curl google.com
anonymous@Unix-the-cat:/$ exit
anonymous@Unix-the-cat:/$ pwd
anonymous@Unix-the-cat:/$ cd /tmp
anonymous@Unix-the-cat:/tmp$ ls
anonymous@Unix-the-cat:/tmp$ wget http://10.1.24.203/40871
anonymous@Unix-the-cat:/tmp$ wget http://10.1.24.203:8000/40871
anonymous@Unix-the-cat:/tmp$ chmod +x 40871
anonymous@Unix-the-cat:/tmp$ ./40871
anonymous@Unix-the-cat:/tmp$ ls
anonymous@Unix-the-cat:/tmp$ rm 40871
anonymous@Unix-the-cat:/tmp$ exit
```

Figure 71: Snippet of Pure honeypot log file listing commands by a lower to mid level attacker. They attempted to execute a local exploit to develop a race condition to gain root/system level privileges. They failed and did not successfully detect system as a honeypot.

Figure 72: Snippet of Pure honeypot log file showing shell interaction by a higher skilled participate that discovered the tee process with ps -ef and location of the log files under /lib/Plymouth/themes/detail/details.ssh.

Effects

There were no effects to the actual pure honeypot. The changes to the system by A1, A2, and A3 such as log files being created, and the network traffic recorded is captured as alpha, beta and mu which causes D_0 to increment.

A1 received feedback that gave no indication that the system was a deception.

A2 executed commands and attempted to download and launch an exploit to elevate their privileges on the system as another access method. The exploit failed, and no commands execute gave indication the system was a deception.

A3 executed commands in figure 72 which revealed the tee process. The “ps -ef” command followed by changing directory using “cd” to the location where user shell interaction is recorded

were feedback to the attacker that this was a deception. In this case the attacker explored a bit further on the system and log contents.

Impact

A1 had no impact as they did not discover the presence of the honeypot monitoring. No actions or commands conducted matched any known intelligence and the deception resumed. The determination was to resume the deception.

A2 had no impact as they did not discover the presence of the honeypot monitoring. A3. There were some intelligence matches for executing the privilege escalation exploit code. No other commands or auction triggered any other alerts. The determination was to resume the deception.

A3 had the impact of identifying the deception monitoring tee process and the location of the logged interaction. The "ps -ef" command followed by changing directory using "cd" to the location where user shell interaction is located triggered an alert to the defender. In this case the attacker browsed the logs then the system a bit more before departing. The determination was to stop the deception and do further analysis on what was conducted.

In this chapter we presented a series of models and a deception framework. We first presented the Attackers Bias Perception Profile model that enabled us to see how each attacker profile ranked from a likelihood and impact perspective. We used this model to see how each participant ranked and how each honeypot system faired in its deceptive ability. We then created an abstraction of each deception process to develop a common foundation which is used to create the deception framework. Three case studies were presented each with a unique outcome based

on the results from the small number of participants in our experimentation. The case studies showed how each participant attacker stepped through the framework starting with their initial access attack vector, the effects and feedback, and overall impact. We also showed how the deception is initially planned, deployed, decisions are made on continuation, and the use of cyber intelligence.

CHAPTER NINE: CONCLUSION AND FUTURE WORK

We have contributed to further researching and experimenting with deception software. We identified based on the empirical control testing and the small number of experimentation participants that pure versus emulated honeypots performed better at evading detection. We also have also demonstrated that indeed there is a correlation between the skill level of the attacker and the success in detecting systems as a deception. There is of course an advantage to participant versus real attackers as they are aware are potential honeypots to discover. The assumption is real attackers may be further deceived, not as conscious in considering deceptive techniques are in use.

We have identified and configured the SSH honeypot systems, constructed an experiment range, recruited participants, commenced experiment testing, and produced results. The build out of the range took approximately nine months to construct and consists of three zones that are accessible from anywhere via virtual private networking. Using the range, we conducted control and participant-based experiments against SSH honeypots and real SSH systems from an attacker system. The empirical control experiment baselined and empirically explored ways the honeypots are detectable amongst all the systems through adversarial analysis and testing. We conducted a series of tests which covered a network service sweep, then moved to scanning and probing of the discovered services, and finally manual hands on keyboard interaction and command execution. The participant-based experiment introduced attackers of varying skills and potentially unique tactics, techniques and procedures in attempting to detect the honeypots.

The participants were recruited from a sample of individuals with professional cyber and information technology experience.

We bridged the experimental with the theoretical and used the results to create a set of models and present a deception framework. The Attacker Bias Profile Perception model was formed to measure the likelihood and impact of attackers at different levels. We then used our model to see how each participant ranked skill wise compared to the six outlined attacker profiles. In addition, the honeypots were overlaid on the model to see how they fared from a subverting detection perspective.

We abstracted the researched deception models to create a foundation for how deception is perceived. Using this abstraction, we presented a deception framework which steps though and illustrates the interaction of a deception to system elements changing its overall state. The framework then depicts how the attacker interacts with a deception, their TTPs, along with the access, effects, and impact. The deception framework is made to help cyber defender understands the deception process, how it interacts with attackers, and how known intelligence is used to better prepare for a more successful deployment.

Future Work

There are several areas to further understand the successful use of deception in Cyber. The ability to develop a more robust honeypot solution will always be evolutionary. The first area being considered is analysis on the log files produced by each honeypots and systems. Analysis would include data mining, data identification, correlation and more. The intention is

to research the possibility of discovering identifiers in attacker behavior when detecting honeypots which could be used to alert to this specific behavior.

Another area involves the development of a robust simulator. The simulation would follow a similar construction as the experiment range with the ability to change the deception software, its settings, responses, and more. Random attackers would be generated at different skill levels following one of the attacker profiles. Simulating could enable diverse sample and sizes where adjustments could be made on anticipated threat, specific deception deployments, and more.

Another area for extended work which can possibly reach a wider range of actual participants is an online web-based survey. The website can consist of several screenshots for remote SSH systems, tool output, command execution of actual honeypots, and more. The participant could be given the opportunity to type in commands or select tools they would want to use to see the specific information or output. Based on the screenshots, each participant selects if they believe the output is based on a honeypot or not.

A bug bounty, or vulnerability reward program, could be stood up with the goal of hackers attempting to research and devise new detection techniques, uncover vulnerabilities, and discover unknown vectors of attack. A prize schedule can be established to help entice the discovery of new attacks where the more complex the larger the sum. As new items are discovered, they can be posted on a forum for others to see, view the prizes earned, and perhaps build on further. Findings can be applied to improve the honeypot code.

Putting deception on a real production system to jolt the attacker is another area being considered. A normal production server could have an obvious honeypot software installed to deceive attackers. The honeypot would possibly cause an instinctive dismissal, further masking the real purpose of the system. The reverse idea of desiring the deception to be as detectable as possible while having the lowest interaction level would be ideal. The low interaction level would help minimize successful attacks and more.

APPENDIX A: PARTICIPANT FLYER

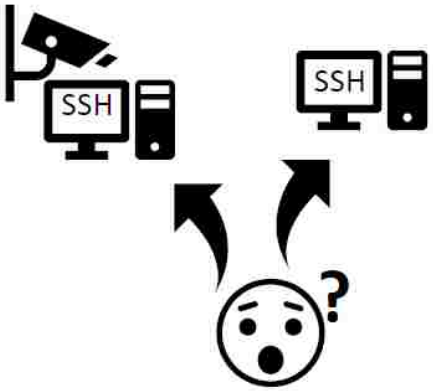
Participants Needed

for Studying the Identification of Cyber Defensive Deception Honeypots

What will you be doing?

Assisting in research for a dissertation - each participant will complete a skills survey, conduct hands on experimentation in attempting to identify honeypot services, complete a scorecard, and finally conduct a exit survey on the research and results. All individuals participating remains confidential.

*More information is available upon request!



Who is the ideal participant?

- ❑ **Must be 18 years or older**
- ❑ **Must speak the English language**
- ❑ **Posses skills in one or more of the following disciplines: IT System administration, networking, coding, Cyber Security, Cyber assessments, Cyber testing, exploiting and scanning.**

*** In Summary, you must be able to interact, attack, analyze, or scan listening TCP ports/services.**

When: Tentatively ready around September and will be made available for 6-8 weeks.

Anticipated time of involvement is 2-3 hours but you can spend more or less time as desired or needed

Thank You!

A $50 Amazon gift card will be given to everyone who participates

Contact me for more Information and if you would like to participate!

sharifhassan@knights.ucf.edu
407-808-5401

Sharif Hassan - 2018 - Research overseen by Dr. Ratan Guha, the college of Computer Science and by the University of Central Florida Institute Review Board on Human Research participation

APPENDIX B: PARTICIPANT SURVEY

Participant only given red portion

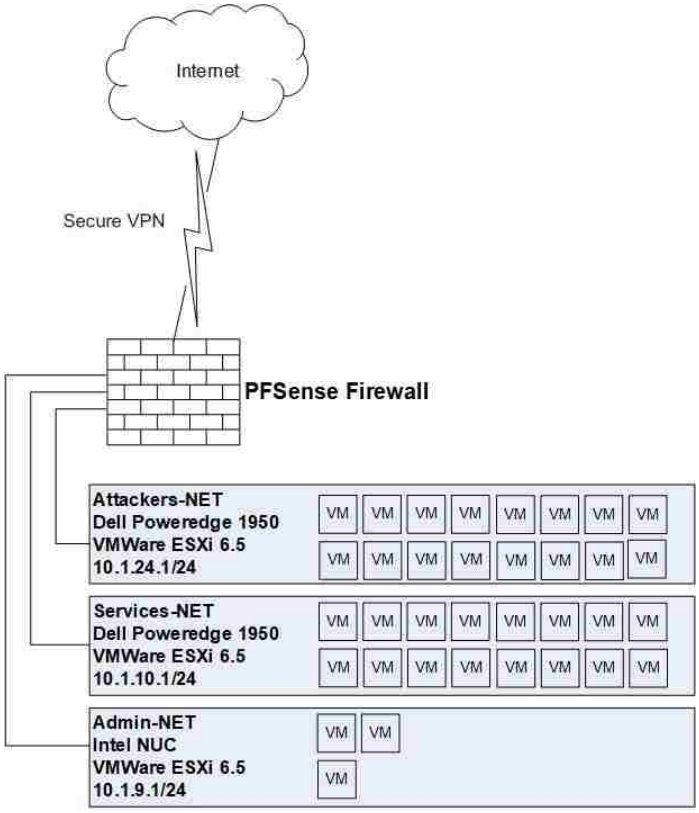
Skill	Score (0-10)	Notes	Internal Notes	intuition weight	value
Reverse Engineering	2			1.2	24
Exploit writing/creating	3			1.1	33
Assembly Language	2			1	20
Windows OS Internals	8			0.9	72
Linux OS internals	8			0.9	72
Networking essentials knowledge of (TCP, UDP, IP, ICMP, sniffing, pcap, config, e	7			0.9	63
Security tools (nmap, Metasploit, scanners, enum, queries, etc.)	9			0.8	72
Database knowledge (oracle, sql, MySQL, etc.)	7			0.9	63
Network traffic analysis (sniffer, pcap, tcpdump, traffic patterns, etc.)	7			1	70
Web application Penetration Testing	7			1.1	77
Evade Endpoint protections (HIPPS, AV, Sysmon, etc.)	7			0.9	63
Electronic Warfare (SDR, SATCOM, RF)	2			1.1	22
Wireless hacking (802.11, ZigBee, Bluetooth)	3			1	30
web application development (HTML, PHP, Python, etc.)	2			1	20
Offensive Development (exploit, overflows, etc.)	5			1.1	55
Scripting (Bash, PowerShell, Python, Perl, Bat)	4			1	40
C, Java, Coding	4			1	40
System forensics	7			0.9	63
General Penetration Testing Skills/methodology	10			1	100
Incident Response (uncovering and detecting attacks)	9			0.9	81
Discovered a zero day before?	0	Either yes or no	yes=10, no = 0	1.5	0
Have you ever compromised (exploited) a computing system	10	Either yes or no	yes=10, no = 0	1.3	130
Do you work on puzzles often (Sudoku, Crossword, etc.)?	10	Either yes or no	yes=10, no = 0	0.5	50

Total Weight	Participant Score
1	54

APPENDIX C: PARTICIPANT EXPERIMENT INSTRUCTIONS

Objective: To attempt to discover which services are either a SSH honeypot service or not. You can use any tool, tactic technique or procedure (TTPs), etc. but you must document how you came to your decision for each IP/port. Document each IP/Port tested, the outcome, why you decided one way or the other, supporting tool information, screen shots and anything else that is helpful. If you can not find any indication or supporting evidence of the service being a honeypot, then assume it is not and select "No, not a honeypot". In this experiment we will focus solely on secure shell (SSH).

Honeypots can be built in numerous ways and can range in their interaction abilities ranging from low to high. Some honeypots are a real system that has been modified into a honeypot to deceive the attacker. Others are fully developed mock ups or simulations of a system or port. Do not be paranoid and think every service is a honeypot. Assume you do not expect any honeypots but are always on alert. Then if something triggers that makes you believe it is a honeypot or deception document and score as such. Some honeypots will be "savvy" and may require deeper poking to determine if its deceptive or not. But to formalize the definition we shall use in this experiment; a honeypot is any system that is deceiving an attacker while monitoring their login attempts and possible shell activity to either study their actions or lure them away.



Attackers-NET foothold: Every participant will connect into the Attackers-NET and be given access to their assigned kali Linux and/or Windows 10 attacker platforms. Account and connectivity information will be emailed separately to each participant.

Target Scope: Services-NET 10.1.10.200 through 10.1.10.220 all focused on port 22, SSH. No other IPs or devices are in scope for testing or attacking. Also no lateral movement from inside any SSH server in the Services-NET. For example, do not try to use any SSH server you gained SHELL access to as a pivot point to scan, attack or look at other network devices within or outside of the scope.

OSINT: Each participant will be given a piece of simulated Open Source Intelligence (OSINT) data that may have a username/password combination to one or more of the services hosted on an IP/port. The intention is to see how further interaction with the one honeypots differs in the outcome. This OSINT will be saved in the /root folder of the kali Linux attacker platform.

Scoring: Each correct answer is worth 1 point. Each wrong answer is -1 point. If suspicious is selected, then no points are given or reduced. Maximum number of points is 20. Referring to table one below as an example, let us assume .50 and .51 are honeypots while .52 and .53 are not. The total current score would be -1. Minus one for .50, no change for .51, minus one for .52 and plus one for .53.

IP	Port	No, not a honeypot	Suspicious	Yes, a honeypot.
10.1.10.50	22	x		
10.1.10.51	22		x	
10.1.10.52	22			x
10.1.10.53	22	X		

Prizes: I appreciate your support in helping me with my dissertation research. As such everyone will get a prize ($50 Amazon gift card). Please do not simply guess. If you must guess, please indicate such in your notes for each IP/port. Scoring for guessing will still follow the same

schedule but it indicates in the research that you were not sure either way. Please email me your address where you would like your prize sent.

Participation Privacy: Each participant data, whether directly given in response to or generated because of this exercise will be protected from disclosure and will abide by UCF policies and procedures in data retention, security, and destruction. No data will be shared in a manner that is attributed or correlated back to you as an individual. Survey results and results from the scorecard will be generalized and referred to as "tester number x" where x is a sequential number or something to that effect. All data during the exercise and while research is being conduct for the completion of a dissertation will be secured.

Expected returned results

Honest Survey on skill level: Attached to this email is a survey of various skills/traits that exist in a wide range of penetration testers and Cyber attackers alike. The intention is to be 100% honest and rate yourself from 0 through 10, where 0 is none and 10 is expert (some questions are marked for yes/no only). The survey helps determine the skill range of participants and to map that against possible cyber threat. No details of this survey will be attributed back to anyone name/person specifically. Please be as accurate and honest as you can and rate yourself on each skill/trait. If you do not have Microsoft Word please email me and I will provide a PDF version. Please complete the survey and return, preferably before you complete the entire exercise.

Score Card template:

The below table is a template of what the expected returned results format follows**.** You do not need to follow this table format exactly as its meant to reflect what information needs to be collected for each IP/port.

IP	Port	No	Suspicious	Yes	Why/explanation/screenshots
10.1.10.200	22				
10.1.10.201	22				
10.1.10.202	22				
10.1.10.203	22				
10.1.10.204	22				
10.1.10.205	22				
10.1.10.206	22				
10.1.10.207	22				
10.1.10.208	22				
10.1.10.209	22				
10.1.10.210	22				
10.1.10.211	22				
10.1.10.212	22				
10.1.10.213	22				
10.1.10.214	22				
10.1.10.215	22				

10.1.10.216	22				
10.1.10.217	22				
10.1.10.218	22				
10.1.10.219	22				
10.1.10.220	22				

Exit Survey Questions:

Upon completing the skills survey and scorecard, a final exit survey will be sent with some questions about the research, your experience, and more. The survey will be emailed to you for you to complete and email back.

APPENDIX D: IRB APPROVAL

University of Central Florida Institutional Review Board
Office of Research & Commercialization
12201 Research Parkway, Suite 501
Orlando, Florida 32826-3246
Telephone: 407-823-2901 or 407-882-2276
www.research.ucf.edu/compliance/irb.html

Approval of Human Research

From: **UCF Institutional Review Board #1**
FWA00000351, IRB00001138

To: **Sharif Hassan**

Date: **August 17, 2018**

Dear Researcher:

On 08/17/2018 the IRB approved the following human participant research until 08/16/2019 inclusive:

Type of Review:	UCF Initial Review Submission Form Expedited Review
Project Title:	Participant Based Challenge in Identifying Defensive Deception Honeypots
Investigator:	Sharif Hassan
IRB Number:	SBE-18-13872
Funding Agency:	
Grant Title:	
Research ID:	N/A

The scientific merit of the research was considered during the IRB review. The Continuing Review Application must be submitted 30days prior to the expiration date for studies that were previously expedited, and 60 days prior to the expiration date for research that was previously reviewed at a convened meeting. Do not make changes to the study (i.e., protocol, methodology, consent form, personnel, site, etc.) before obtaining IRB approval. A Modification Form **cannot** be used to extend the approval period of a study. All forms may be completed and submitted online at https://iris.research.ucf.edu .

If continuing review approval is not granted before the expiration date of 08/16/2019, approval of this research expires on that date. When you have completed your research, please submit a Study Closure request in iRIS so that IRB records will be accurate.

Use of the approved, stamped consent document(s) is required. The new form supersedes all previous versions, which are now invalid for further use. Only approved investigators (or other approved key study personnel) may solicit consent for research participation. Participants or their representatives must receive a copy of the consent form(s).

All data, including signed consent forms if applicable, must be retained and secured per protocol for a minimum of five years (six if HIPAA applies) past the completion of this research. Any links to the identification of participants should be maintained and secured per protocol. Additional requirements may be imposed by your funding agency, your department, or other entities. Access to data is limited to authorized individuals listed as key study personnel.

In the conduct of this research, you are responsible to follow the requirements of the Investigator Manual.

This letter is signed by:

Signature applied by Gillian Morien on 08/17/2018 10:52:01 AM EDT

Designated Reviewer

LIST OF REFERENCES

[1] Tzu, Sun, "The Art of War Special Edition" El Paso Norte Press; Special edition (March 21, 2005), ISBN 978-0976072690.Advanced Persistent Threats: How to Manage the Risk to your Business. ISA ©2013 ISBN:1604203471 9781604203479

[2] Rowe, Neil C. (2006) “Measuring the Effectiveness of Honeypot Counter-Counterdeception” HICSS ’06 Proceedings of the 39th Annual Hawaii International Conference o System Science, Vol 06, pp 129

[3] Rachna Dhamija, J. D. Tygar, and Marti Hearst. 2006. Why phishing works. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06), Rebecca Grinter, Thomas Rodden, Paul Aoki, Ed Cutrell, Robin Jeffries, and Gary Olson (Eds.). ACM, New York, NY, USA, 581-590. DOI=http://dx.doi.org/10.1145/1124772.1124861

[4] Ari Juels. 2014. A bodyguard of lies: the use of honey objects in information security. In Proceedings of the 19th ACM symposium on Access control models and technologies (SACMAT '14). ACM, New York, NY, USA, 1-4. DOI: https://doi.org/10.1145/2613087.2613088

[5] Advanced Persistent Threats: How to Manage the Risk to your Business. ISA ©2013 ISBN:1604203471 9781604203479

[6] S. Hassan & R. Guha, "Honeypots and the Attackers Bias," 2018 13th International Conference on Cyber Warfare and Security (ICCWS), Washington, DC 2018, pg. 533, ISBN: 978-1-5108-5963-0

[7] S. Hassan and R. Guha, "Security and Integrity Analysis Using Indicators," 2012 International Conference on Cyber Security, Washington, DC, 2012, pp. 127-135. doi: 10.1109/CyberSecurity.2012.23

[8] S. Hassan & R. Guha, "Modelling of the State of Systems with Defensive Deception," 2016 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, pp 1031-1036. doi: 10.1109/CSCI.2016.0197.

[9] S. Hassan and R. "Probabilistic Deception and Attacker Experimentation," 2017 International Conference on Security and Management (SAM'17), Las Vegas, NV, Copyright © 2017 CSREA Press, pp. 252-258. ISBN: 1-60132-467-7

[10] S. Hassan & R. Guha, "A Probabilistic Study on the Relationship of Deceptions and Attacker Skills," 2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing (DASC), Orlando, FL, PP 693-698. ISBN 978-1-5386-1956-8/17 © 2017 IEEE doi: 10.1109/DASC-PICom-DataCom-CyberSciTec.2017.121

[11] U.S. DoD, "The Department of Defense Cyber Table Top Guidebook", 2018. Deputy Assistant Security of Defense Development Test and Evaluation. https://www.acq.osd.mil/dte-trmc/docs/The DoD Cyber Table Top Guidebook v1.pdf

[12] Miles A McQueen and Wayne F Boyer "Deception used for cyber defense of control systems" HSI'09 Proceedings of the 2nd conference on Human System Interactions Pages 621-628 IEEE Press Piscataway, NJ, USA. 2009 table of contents ISBN: 978-1-4244-3959-1

[13] Andrea M. Matwyshyn, Material Vulnerabilities: Data Privacy, Corporate Information Security, and Securities Regulation, 3 Berkeley. Bus. L.J. 129 (2005). Available at: http://scholarship.law.berkeley.edu/bblj/vol3/iss1/4

[14] Jim Yuill, Fred Feer and Dorothy Denning “Designing Deception Operations for Computer Network Defense” Fall 2004 DoD Cyber Crime Conference

[15] James J Yuill “Defensive computer-security deception operations: principles, and techniques” Doctoral Dissertation North Carolina State University 2006 ISBN: 978-0-549-37681-1

[16] M. H. Almeshekah and E. H. Spafford, “ Planning and Integrating Deception into Computer Security Defenses”

[17] M. H. Almeshekah “Using Deception To Enhance Security: A Taxonomy, Model, and Novel Uses” Doctoral Dissertation Purdue University August 2015.

[18] McAfee Threats Report Second Quarter (2015).

[19] Kristin E. Heckman, Frank J. Stech, Ben S. Schmoker, Roshan K. Thomas, "Denial and Deception in Cyber Defense", Computer, vol.48, no. 4, pp. 36-44, Apr. 2015, doi:10.1109/MC.2015.104

[20] Woo, Hyung-Jin, "The Hacker Mentality: Exploring the Relationship Between Psychological Variables and HAcking Activities", 2003, Doctoral Dissertation, University of George, Athens, GA

[21] Honeynet Project, "Know your enemy, Learning About Security Threats, 2nd edition", 2004, , Addison-Wesley, ISBN: 9780321166463

[22] O.A, A., & Sardana, A “Design and Implementation of a Medium Interaction Honeypot.” . International Journal of Computer Applications (0975 – 8887). Volume 70– No.22, May 2013

[23] Seifert, C., Welch, I., Komisarczuk, P. “Taxonomy of Honeypots”. 2006, Victoria University of Wellington, Technical report CS-TR-06/12.

[24] Dagon D. et al. "HoneyStat: Local Worm Detection Using Honeypots". In: Jonsson E., Valdes A., Almgren M. (eds) Recent Advances in Intrusion Detection. RAID 2004. Lecture Notes in Computer Science, vol 3224. Springer, Berlin, Heidelberg

[25] Iyatiti Mokube and Michele Adams. 2007. Honeypots: concepts, approaches, and challenges. In Proceedings of the 45th annual southeast regional conference (ACM-SE 45). ACM, New York, NY, USA, 321-326. DOI=http://dx.doi.org/10.1145/1233341.1233399

[26] Spitzner, L. (2004). Honeypots: Catching the insider threat. 170- 179. 10.1109/CSAC.2003.1254322.

[27] Github.com

[28] Docker store store.docker.com, Docker Incorporated

[29] Desaster (2010). *Kippo Honeypot*. Project and source code hosted at GitHub, https://github.com/desaster/kippo

[30] Keane, Justin K. Kojoney honeypot. Project and source code hosted at GitHub https://github.com/madirish/kojoney2

[31] Oosterhof, Michael. (2014) *Cowrie Honeypot*. http://www.micheloosterhof.com/cowrie/

[32] David Watson HoneyNet Prject – GDH1

[33] David Watson HoneyNet Prject – GDH2 HooNeeBox

[34] Network Working Group of the IETF, January 2006, RFC 4251, The Secure Shell (SSH) Protocol Architecture

[35] Khan, Mazhar I. (2017) *TELPOT Capturing Cyber Attacks with generic Telnet Based Honeypot,* Project and source code hosted at GitLab, https://github.com/MazharIITK/TELPOT

[36] Tatu Ylönen. 1996. “SSH: secure login connections over the internet”. In Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6 (SSYM'96), Vol. 6. USENIX Association, Berkeley, CA, USA, 4-4

[37] Morris, Andrew – Metasploit module - Detect_kippo - https://www.rapid7.com/db/modules/auxiliary/scanner/ssh/detect_kippo

[38] Grimes, Roger A. “Your guide to the seven types of malicious hackers”, 2011, ComputerWorld, https://www.computerworld.co.nz/article/498979/your_guide_seven_types_malicious_hackers/

[39] Ashoor, Asmaa, S. "What is the difference between Hackers and Intruders". 2011, International Journal of Scientific & Engineering Research Volume 2, Issue 7, July-2011 1 ISSN 2229-5518

[40] Vidalis, Stilianos & Jones, Andy. "Analyzing Threat Agents and Their Attributes." 2005, . 4th European Conference on Information Warfare and Security, 369-380.

[41] Howlett, William IV, "The Rise of China's Hacking Culture: Defining Chinese Hackers" (2016). Electronic Theses, Projects, and Dissertations. Paper 383.

[42] Giboney, Justin S.; Proudfoot, Jeffrey G.; Goel, Sanjay; and Valacich, Joseph S., "Measuring Hacking Ability Using a Conceptual Expertise Task" (2015). Annual ADFSL Conference on Digital Forensics, Security and Law. 8. http://commons.erau.edu/adfsl/2015/wednesday/8

[43] MCAffee Securities. Net Losses: Estimating the Global Cost of Cybercrime. Report. Santa Clara: Center for Strategic and International Studies, 2014.

[44] US DHS. (2016). What is Wannacry/WanaCrypt0r, US DHS. (2016). What is Wannacry/WanaCrypt0r

[45] Chen, Thomas M (2014). *Cyberterrorism after Stuxnet*, US Army War College, Strategic Studies Institute, US Army College Press, http://ssi.armywarcollege.edu/pdffiles/pub1211.pdf

[46] US Naval Academy. (2016). Operation Blockbuster – Unraveling the Long Threat of the Sony Attack, NOVETTA Threat Research Group, https://www.usna.edu/CyberCenter/_files/documents/Operation-Blockbuster-Report.pdf

[47] Mikhaylova, Galina. (2014). *The "anonymous" Movement: Hacktivism as an Emerging Form of Political Participation*, Graduate Thesis Texas state university. December. 2014, https://digital.library.txstate.edu/bitstream/handle/10877/5378/MIKHAYLOVA-THESIS-2014.%20pdf?sequence=1

[48] Casserly, Martyn. "Who Is Anonymous? A Short History of Hacktivism." PC Advisor. November 18, 2015. Accessed March 30, 2016. http://www.pcadvisor.co.uk/feature/internet/what-is-hacktivism-shorthistory-anonymous-lulzsec-arab-spring-3414409/

[49] Searle, Nicola. (2012). *The Criminalization of the Theft of Trade Secrets: An Analysis of the Economic Espionage Act* IP Theory: Vol 2, lss 2, Article 2, http://www.repository.law.indiana.edu/ipt/vol2/iss2/2

[50] Mandiant. (2013). APT1 report - https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf - FireEye Corporation.

[51] Lau, Hon. "The Truth Behind the Shady RAT." Symantec Security Response. August 4, 2011. Accessed March 30, 2016. http://www.symantec.com/connect/blogs/truth-behind-shady-rat

[52] US-CERT. 2017. "Hidden Cobra - North Korea's DDos Botnet Infrastructure. https://www.us-cert.gov/ncas/alerts/TA17-164A

[53] Kushner, David. “The Real Life Story of Stuxnet - How Kaspersky Lab tracked down the malware that stymied Iran’s nuclear-fuel enrichment program”. 26 February 2013. IEEE Spectrum. https://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet/

[54] Islam, A., Oppenheim, N., Thomas, W. "SMB Exploited: WannaCry Use of "EternalBlue" 26 May 2017. FireEye "https://www.fireeye.com/blog/threat-research/2017/05/smb-exploited-wannacry-use-of-eternalblue.html

[55] Haggard, S. and Lindsay, J. "North Lorea and the Sony Hack: Exporting Instability Through Cyberspace" May 2015. East-West Center No. 117 https://scholarspace.manoa.hawaii.edu/bitstream/10125/36444/1/api117.pdf

[56] U.S. Attorney's Office. "Bala Cynwyd Man Sentenced to Prison for Hacking Computers of Public Utilities". 15 June 2017. Department of Justice Eastern District of Pennsylvania. https://www.justice.gov/usao-edpa/pr/bala-cynwyd-man-sentenced-prison-hacking-computers-public-utilities

[57] U.S. Attorney's Office. "Bala Cynwyd Man Sentenced to Prison for Hacking Computers of Public Utilities". 13 April 2017. Department of Justice Southern District of New York. https://www.justice.gov/usao-sdny/pr/software-engineer-arrested-attempted-theft-proprietary-trading-code-his-employer

[58] Mikhaylova, G. "THE “ANONYMOUS” MOVEMENT: HACKTIVISM AS AN EMERGING FORM OF POLITICAL PARTICIPATION". December 2014 M.A. Thesis. Texas State University.

[59] Jordan, Tim. 2002. “Hacktivism: All Together in the Virtual.” Pp. 119-137 in Activism!: Direct Action, Hacktivism and the Future of Society, edited by Barrie Bullen and Peter Hamilton. London, UK: Reaktion Books Ltd

[60] US DHS ICS-CERT. "Cyber-Attack Against Ukrainian Critical Infrastructure". 25 February 2016. Alert (IR-ALERT-H-16-056-01) https://ics-cert.us-cert.gov/alerts/IR-ALERT-H-16-056-01

[61] Microsoft. "Customer Guidance for WannaCrypt attacks". 12 May 2017. https://blogs.technet.microsoft.com/msrc/2017/05/12/customer-guidance-for-wannacrypt-attacks/

[62] Moore, H.D. Metasploit Project. Rapid7. https://www.metasploit.com/.

[63] Deutsche Telekom’s Honeypot Team. "T-Pot 17.10 - Multi-Honeypot Platform rEvolution". 2018. T-Pot 17.10 - http://dtag-dev-sec.github.io/

[64] Merriam Webster. https://www.merriam-webster.com/dictionary/deception

[65] 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, 1:80, 2011

[66] Ghoshal, Devarshi & Plale, Beth. (2013). “Provenance from log files: A BigData problem”. ACM International Conference Proceeding Series. 290-297. 10.1145/2457317.2457366.

[67] Liao, Xiaojing & Yuan, Kan & Wang, Xiaofeng & Li, Zhou & Xing, Luyi & Beyah, Raheem. (2016). “Acing the IOC Game: Toward Automatic Discovery and Analysis of Open-Source Cyber Threat Intelligence”. 755-766. 10.1145/2976749.2978315.

[68] OpenIOC. 2011. http://www.openioc.org/ Mandiant.

[69] US DoD Department of Defense Cyber Crime Center. http://www.dc3.mil/

[70] Lyon, Gordon. Nmap – Network Scanning Utility - https://nmap.org/

[71] J. Hong, “The State of Phishing Attacks” Communications of the ACM, Vol. 55, No. 1. 2012.

[72] M. Cloppert, “Security Intelligence: Attacking the Kill Chain” 2009. http://computer-forensics.sans.org/blog/2009/10/14/security-intelligence-attacking-the-kill-chain/

[73] Sardiwal, S. et al. “New Targeted Attack in the Middle East by APT24, a suspected Iranian Threat Group using CVE-2017-11882 exploit”. (December 07 2017). FireEye Threat Research. https://www.fireeye.com/blog/threat-research/2017/12/targeted-attack-in-middle-east-by-apt34.html

[74] Srijon, Schuvradeb B. “How to capture terminal session (remote/local) in Linux”. (June 28 2012). The Voice of Srijon. Wordpress.com. https://iamsrijon.wordpress.com/2012/06/28/how-to-capture-terminal-sessions-remotelocal-in-linux/

[75] Garcia, Jeronimo. “How I’ve captured all the passwords trying to ssh into my server!” (Dec 3, 2017) Hackernoon.com Short Tech Stories. https://hackernoon.com/how-ive-captured-all-passwords-trying-to-ssh-into-my-server-d26a2a6263ec

[76] Bharti, Santosh. (2016). Honeypot-based Intrusion Detection System: A Performance Analysis. 10.13140/RG.2.1.4599.9768.

[77] M. Aharoni, *et al*. “Metasploit Unleased, Mastering the Framework” http://www.offensive-security.com/metasploit-unleashed/Main_Page. 2011.

[78] Alexander Vetterl and Richard Clayton, "Bitter Harvest: Systematically Fingerprinting Low- and Medium-interaction Honeypots at Internet Scale", 12th USENIX Workshop on Offensive Technologies (WOOT 18),2018, Baltimore, MD, https://www.usenix.org/conference/woot18/presentation/vetterl.

[79] N. Veerasamy, "High-Level Methodology for Carrying out Combined Red and Blue Teams," *2009 Second International Conference on Computer and Electrical Engineering*, Dubai, 2009, pp. 416-420. doi: 10.1109/ICCEE.2009.177

[80] Clark, Ben. (2013). *RTFM: Red Team Filed Manual.* Amazon, ISBN 9781494295509

[81] Yahyaoui, Aymen. (2014) “Testing Deceptive Honeypots”. Naval Postgraduate School, Computer Science Thesis

[82] Cohen, F., Marin, I., Sappington, J., Stewart, C., and Thomas, E. (2001) “Red Teaming Experiments with Deception Technologies” all.net Fred Cohen research

[83] BackOfficer Friendly

[84] L. Spitzner, "The Honeynet Project: Trapping the Hackers," in *IEEE Security & Privacy*, vol. 1, no. , pp. 15-23, 2003. doi:10.1109/MSECP.2003.1193207

[85] Stoll, Clifford. “The CooKuu’s Egg”

[86] Nawrocki, M et al. “A survey on Honeypots Software and Data Analysis”,

[87] OpenSSH Project - https://www.openssh.com/history.html

[88] VirtualEnv – Python virtual environments - https://virtualenv.pypa.io/en/stable/

[89] Dropbear SSH - https://matt.ucc.asn.au/dropbear/CHANGES

[90] OpenWrt - https://wiki.openwrt.org/doc/uci/dropbear

[91] Kojoney - kojoney.sourceforge.net/

[92] Kojoney2 - https://github.com/madirish/kojoney2

[93] Xsweet - https://github.com/techouss/xsweet

[94] Mitre, CVE-2018-15473, Common Vulnerability and Exposure, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-15473

[95] Internet Security Threat Report. 2017. Symantec Corporation. - http://ecee.colorado.edu/~ekeller/classes/fall2017_advsec/papers/symantec_2017.pdf