USING DECOY DOCUMENTS TO DETECT MASQUERADE ATTACKS

A Thesis by

Maciej Berdychowski

Bachelor of Science, Wichita State University, 2018

Submitted to the Department of Electrical Engineering and Computer Science
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

May 2021

USING DECOY DOCUMENTS TO DETECT MASQUERADE ATTACKS

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Sergio A. Salinas Monroy, Committee Chair

Yumi Suzuki, Committee Member

Zhiyong Shan, Committee Member

Monowar Hasan, Committee Member

DEDICATION

I dedicate this thesis to my Mom.

ACKNOWLEDGEMENTS

Special thanks to Dr. Sergio A. Salinas Monroy for his input in much of this work.

ABSTRACT

Privacy, security and integrity of data are becoming ever more important in a connected world. While every effort is being made to keep cyber criminals away from sensitive information, sometimes they succeed in bypassing perimeter defenses and masquerade as legitimate users. If that happens, deception remains the last line of defense. One of deception's forms, decoy documents, or honeyfiles, offers a chance to detect the presence of an attacker without large capital investments. In laboratory tests, honeyfiles demonstrated big potential. This thesis describes the results of an experiment in which a server with decoy documents was exposed to real hackers through the Internet. The data collected shows that honeyfiles can be an effective complement to traditional Intrusion Detection Systems. Suggestions for further research and improvements to the method are also discussed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Ever since computers gained the ability to store data, preserving the privacy and integrity of that data has been one of the most important issues faced by the IT industry. Logins and passwords were an obvious choice as means to eliminate unauthorized access, but their weaknesses became clear rather soon – if it is easy for users to remember, it is not very hard to guess for attackers.

Multiple data breaches that exposed personal information of millions of people from all over the world have clearly demonstrated the importance of privacy protection. While many efforts have been made to prevent any kind of access to sensitive information, a good security strategy should plan for those rare instances when an adversary manages to bypass protections and his presence inside a system needs to be detected as soon as possible.

If an intrusion takes place, it may seem intuitive to block the attacker from any further access to the system. However, it may often be more beneficial to allow him to penetrate the system and collect information about his behavior in order to improve defenses against future attacks, while keeping him away from valuable assets.

These are the two functions of deception. Deception has been used for many years and can take on various forms. This thesis discusses one of them. Decoy documents, also called honeyfiles, are bait files placed in the system among other, ordinary data files for the sole purpose of luring a hacker into interacting with them [1]. Honeyfiles are placed at locations that the hacker is likely to visit and named in such a way that they appear to contain high-value information for an attacker, enticing him to open them. The usefulness of decoy documents depends on the ability of legitimate users to recognize them as traps. When a honeyfile is opened, or "touched" in any way, an alarm is set off, indicating possible presence of an attacker.

Initially, it was believed that a single interaction with a decoy document would

provide evidence of a masquerade attack [2]. However, practical experiments showed that legitimate users may also accidentally access their own honeyfiles [3, 4, 5]. Due to their unfamiliarity with the file system, attackers were expected to interact with decoys at a higher rate. Therefore, a threshold was proposed as a way to filter out the adversaries [4]. The higher the number of decoy file touches, the higher the probability that there has been an intrusion. A series of lab experiments [3, 4, 5] determined that, when given a specific scenario, volunteers acting as hackers access honeyfiles at a rate much higher than legitimate users. Doubts remain, however, about the ability of that scenario to imitate behavior of real hackers.

To the best of my knowledge, this work presents the first experiment that exposes a server with systematically deployed honeyfiles to attacks from remote adversaries over the Internet. Moreover, the experiment compares the frequency of decoy access by real hackers to that of legitimate users, which allows me to determine if using honeyfiles as an intrusion detection system is practical.

Five different honeyfile deployment scenarios were tested, including two introduced in [4]. The recorded decoy access rates were high enough to detect 77% of attacks. The decoy document concept showed that it can be used in practice.

Compared to previous works, the experiment in this thesis confirmed that the visibility of decoys is more important than the attractiveness of their names [3]. The honeyfile access rate by attackers was higher than that of legitimate users, although not as high as in the lab experiments. The honeyfile placement scenarios did not show a significant influence on the system's true positive rate.

The experiment highlighted some directions which future research of decoy documents could follow. One of the most obvious is the naming of decoys. In more than 90% of undetected cases, the adversaries did not interact with a single honeyfile. Due to multiple interactions with honeyfiles, or an extended interaction with one decoy, being required for attack detection, decoy believability, analyzed in early research and later ignored, should be revisited.

The collected data showed that the concept of decoy documents could be expanded to include system files and calculating the duration of period during which a honeyfile is open. Suggestions for improving the system's efficiency, as well as broader ramifications of some of the observed behavior, are also discussed.

CHAPTER II

LITERATURE REVIEW

In this section, I provide an overview of the existing security mechanisms designed to prevent and mitigate unauthorized access to computer systems. I first give a brief summary of passwords and their shortcomings, and then survey the literature of honeypots and honeytokens, which can be used to mitigate unauthorized access to a system. I focus on honeyfiles, a type of honeytoken that I use in this work's experiments.

2.1 Passwords

Passwords are the most common authentication mechanism. In a system protected by a password, in addition to a username unique within the system, users prove their identity to a system by providing a secret set of characters that is only known to the user and the system. Unfortunately, users across all demographics and different, even seemingly distant, language groups, routinely choose weak, easy to guess passwords [6]. Adversaries can launch various attacks that allow them to guess other users' passwords within a reasonable amount of time [7, 8]. One such attack is statistical-guessing, where the adversary attempts to access the system by trying the words from a word-list ordered by popularity [9]. Coupled with people's tendency to use the same username on different online platforms [10], weak passwords create a serious risk to privacy and data security.

To aid users in choosing a strong password, researchers have proposed techniques to generate secure password suggestions. Houshmand and Aggarwal [11] developed a software system that slightly modifies users' passwords based on their likelihood of being cracked by an attacker. The password changes demonstrated in the paper include adding or modifying one character, and rearranging the order of the characters. The resulting passwords are stronger that the original ones primarily due to the increased number of possible character combinations in longer passwords.

Forget et al. [12] propose to improve password security by inserting randomly-chosen characters at random positions into user-created passwords. Users may shuffle between combinations until they find one that they can remember. Although an 83-participant study found that this method improves password security, the gain in security was limited due to users deliberately choosing weaker pre-improvement passwords that are easier to remember after modifications. Therefore, suggesting stronger passwords to users has only a modest effect on the security of passwords that they choose.

Besides suggesting stronger passwords, researchers have also attempted to enforce policies aiming to increase password security. One such policy is password expiration, where users are required to change their password after a certain time interval. Password expiration reduces the amount of time available to the attacker to access the system before he has to guess the victim's password again. However, Zhang et al. [13] found that users often derive new passwords from old ones by making only very minor changes to them. Since new passwords are very easy to guess, the concept of expiring passwords is ineffective. Schechter et al. [9] propose to prohibit users from choosing passwords that are already common. Unfortunately, this approach still allows users to choose insecure passwords that are unpopular.

To improve the security of password-protected systems in the presence of easily guessable passwords, researchers have proposed multi-factor authentication (MFA). In MFA, the system asks for both the password and additional information that is only known to the user and the system. The additional information can be answers to security questions that the user provides when creating the account or an algorithmically generated token that changes over time. However, MFA is vulnerable to open source intelligence attacks where adversaries can find the answers to the security questions on social media, business websites, etc. [14, 15], and obtain the tokens through social engineering like email or text/SMS phishing [16, 17].

2.2 Intrusion Detection Systems

Due to the challenges involved in enforcing the use of secure passwords and MFA, cyber security professionals are forced to assume that attackers will gain unauthorized access to user systems from time to time. The extent of threat depends on the type of account that fell victim to a masquerade attack, since the attacker has access to all resources that the legitimate user has access to.

If the adversary compromises a standard account, he can modify information or transfer it outside the attacked organization. The latter is usually considered the most serious threat associated with any kind of cyber attacks. If transfer of files between remote hosts and the server is permitted, the hacker can upload malware and use the server as a tool in attacks against third parties.

When a masquerade attack succeeds against a server administrator's account, the situation gets even more serious. The hacker can change passwords of all users of the server, including the administrator. That way he can gain full control of the server and access to all information that all legitimate users had access to, while preventing them from accessing the server. Regaining the control of the server by its legitimate users may be very troublesome, costly and time-consuming.

Therefore, it is important to have additional security mechanisms to detect and mitigate the adversary's presence. Significant amount of research has been dedicated over the years to finding efficient ways of uncovering masquerade attacks [18, 19]. The most popular tool are Intrusion Detection Systems, network or host-based. However, they sometimes produce high numbers of false alarms and are usually ineffective against unknown attacks [1].

Another important aspect of defenses against masquerade attacks is deciding how to react to an adversary's presence. It may seem intuitive and natural to block him, but oftentimes it may be more beneficial to observe his behavior without allowing him to access any valuable resources. That is when deception comes in handy [20, 21, 22].

2.3 Cyber deception

According to Climek et al., cyber deception is a deliberate attempt to conceal the true state of defended systems, create uncertainty and confuse the adversary in order to disrupt his efforts to establish situational awareness, and to influence and misdirect the adversary's perceptions and decision processes [23].

Deception usually serves two purposes: deceiving the attacker about the real state of the system to deflect his malicious actions away from sensitive assets, and gathering information about the attacker's behavior to improve the system's defenses.

2.4 Honeypots and Honeytokens

One of the most common tools of cyber deception are honeypots [24]. Honeypots are intentionally vulnerable servers designed to encourage attackers to break into them instead of real servers. To an attacker that has breached a system, honeypots seem to perform important functions and store high-value data. However, honeypots only contain information that has no value to the organizations that use them.

Honeypots have proven to be very informative and useful for both researchers and administrators [24]. Barron and Nikiforakis [25] showed that the contents, location as well as difficulty of breaking into honeypots influence behavior of hackers, who tend to execute more commands on honeypots with realistic file and folder structures.

However, dedicating a server to be a honeypot is expensive, since that computer cannot be used for any other purpose. It also takes a lot of time to create a server with a realistic structure of folders that does not contain any useful data.

To overcome the challenges of operating a dedicated honeypot server, Spitzner [26] introduced the concept of honeytokens. Similarly to honeypot servers, a honeytoken is a resource whose value lies in its unauthorized or illicit use by an attacker [26]. Unlike a honeypot server, a honeytoken is not a computer. It can be any other type of digital entity.

There are many types of honeytokens, including honeypatches, honeywords, honeyconnections, honeynetworks, and honeyfiles. I briefly summarize some honeytokens

here and postpone the discussion of honeyfiles, which are the main topic of this work, to the next subsection.

Honeypatches [27] redirect an attacker who is targeting a server, looking for a specific vulnerability, to a purposely vulnerable virtual machine. The virtual machine is built without the patch that fixes that vulnerability, which had already been applied to the targeted server. By allowing the attacker to compromise the virtual machine instead of the real server, the server defender wastes the attacker's time and resources and can collect information about the attacker's pattern of behavior and interests.

Honeywords [28] aim to improve the security of hashed passwords. Each account has a few false passwords, or honeywords, associated with it. Even if the attacker steals the file with hashed passwords and succeeds in inverting the hash function, they cannot tell which one is the real password. The use of a honeyword sets off an alarm.

Honeyconnections [29] create artificial traffic around the most important hosts of an organization by having several computers pretend that they perform the same function. The purpose of this activity is to disguise each host's true function and confuse a potential hacker who compromised modern networking hardware like routers or Software Defined Network controllers and is possibly conducting passive reconnaissance. The deception is intended to discourage him from launching an attack, and locate him if he does.

Honeynetworks create virtual networks to mislead an adversary. He can be redirected to a completely separate network [30], where his behavior can be observed safely. Another concept [31] calls for creating a software defined network, where real, vulnerable hosts that need to be protected are placed among multiple virtual hosts. The topology of such network, visible to the attacker, differs significantly from the topology of the real network.

More hands-on approaches combine active and passive deception [32]. Active deception works proactively, trying to anticipate actions of an adversary and adapting to them. This may include secretly modifying key network elements during the attack, while passive deception focuses on misdirecting and recording the actions of an attacker.

The game-theoretic model of one-sided partially observable stochastic games has found a new application in active deception [33]. In it, it is the defender, not the attacker, who has complete information. The latter, however, is assumed to be aware of the use of deception mechanisms against him. By placing numerical values on the losses resulting from each part of the network being compromised by the attacker, it is possible to find optimum balance between engaging the attacker to collect more information about him and blocking him, while limiting the amount of data he accesses. Such scenario minimizes the total loss, but, contrary to conventional wisdom and practices, does not always mean blocking the adversary immediately after detection.

The main problem with the above concepts is that they mostly propose using deception as a reaction to detecting the presence of an adversary. Given the limited effectiveness of Intrusion Detection Systems, there may be instances when an intrusion is never detected, and as a result deception may never begin. Moreover, most deception concepts implicitly assume that detection mechanisms will immediately detect the presence of a hacker with complete certainty, which does not hold true in practice. A delay in starting the deception may give the adversary sufficient time to gather enough information about the real state of the attacked system so as to render deception ineffective. Honeyfiles can address these shortcomings.

2.5 Honeyfiles as an Intrusion Detection System

Honeyfiles are decoy files strategically placed in the system to lure attackers. They use naming conventions that entice attackers to open them, but make it obvious to legitimate users that they are dummy files. Since legitimate users seldom interact with decoy documents, it is possible to design intrusion detection mechanisms that use interactions with honeyfiles as a trigger. That way, honeyfiles complement traditional Intrusion Detection Systems and add a third functionality to deception: detecting the presence of an adversary. This section summarizes the state of the art of honeyfiles used as intrusion detection mechanisms.

The concept of honeyfiles was pioneered in the 1980s by Cliff Stoll [34]. An astronomer by trade, he was transferred from the Keck Observatory at the Lawrence Berkeley National Laboratory to the computer center in the same building after his grant money ran out. There, he noticed that a computer system was compromised by an attacker. After realizing that the hacker was looking for files that contained words such as "nuclear" or "SDI" (Strategic Defense Initiative) and the intrusion was coming from West Germany via satellite, Stoll created a honeypot account named "SDInet". He filled it with large files containing language that gave the impression they were part of secret documentation of interest to the hacker, in order to entice the hacker to reveal more about himself. Ultimately, the hacker was located at his home in Hanover, Germany. It was later determined that he had for years been selling information gathered through his hacking to the Soviet Union's secret service, KGB.

The idea of using honeyfiles as an automated intrusion detection system for end users was formally introduced in 2004 by Yuill et al. [1]. Decoy documents are placed in locations that a hacker may visit and given names that entice an attacker to open them. At the same time, honeyfiles should remain easy to identify as traps by legitimate users. When a user interacts with a decoy, an alarm is set off, indicating the presence of an attacker. Yuill et al. [1] provide a theoretical analysis of the benefits and limitations of honeyfiles. In particular, they note their low cost and ability to detect unknown attacks, which are important improvements over traditional Intrusion Detection Systems. They also evaluate the performance of decoy documents with an experiment. Specifically, they implement a honeyfile system on a server containing error reports and manuals. They ask three students skilled in computer security to hack into the server. Using honeyfiles, they detect three hacking incidents, each involving a different attacker. Unfortunately, few details about the experiment are reported, which makes it hard to evaluate or replicate.

Bowen et al. [2] propose a honeyfile system, called the Decoy Document Distributor, or D^3. D^3 generates decoy files with embedded beacons that send an alert when they are opened. The authors also describe a set of properties that they think honeyfiles should

posses in order to maximize the likelihood of attackers accessing them, while minimizing the risk of legitimate users setting off false alarms. These include: believability, enticingness, conspicuousness, detectability, variability, non-interference and differentiability. To measure the performance of D^3, Bowen et al. deployed their decoy files over a university-run honeynet consisting of several virtual Linux machines. After a brief test where the honeynet was accessible to everyone through the Internet, the authors conclude that decoy files were almost always among the first few files opened by users.

Voris et al. [4] expand the D^3 concept introduced in [2] by adding automated decoy document deployment functionality, which allows for more flexibility and scalability than manual placement of honeyfiles. The D^3 enhancement, called Decoy Distributor Tool, or DDT, allows for two different types of honeyfile placement arrangements: "integrated", where bait files are located among other non-decoy files in existing folders, and "separated", in which decoy files are placed in their own subfolders.

In [5], Voris et al. propose enhancing the concept of password protection with continuous authentication, combining honeyfiles and behavior profiling. The idea is based on the premise that an attacker is unfamiliar with the file structure of a system he penetrates and his pattern of behavior will vary significantly from that of the legitimate user, who knows his or her system very well. The presence of decoy documents is intended to complement comparing the observed behavior with the model behavior. If either of the protection methods sets off an alarm, the system provides an authentication challenge to the user. This additional authentication step uses information different from the login and password, which are already known to the hacker.

Lee et al. [35] extend the decoy concept from data files to system files. They assume that the attacker gains access to the system through a method different than acquiring the credentials of a legitimate user. Their approach uses a hidden interface for legitimate users and system services, and a real interface for attackers. The hidden interface shows only real files. The real interface shows decoy files that can alert the system administrator about the presence of an attacker. However, this approach cannot detect masquerade attacks where

the attacker logs in with the compromised credentials of a legitimate user and gains access to the hidden interface.

2.6 Experiments to Measure the Effectiveness of Honeyfiles as IDS

There have been three major studies conducted with human subjects to measure the practicality and effectiveness of honeyfiles as intrusion detection systems, [3, 4] and [5]. The authors also attempt to estimate the extent of disruption to normal workflow of legitimate users caused by the presence of decoy documents.

Specifically, Ben Salem and Stolfo [3] conduct experiments to determine the best locations and optimum quantity of decoy files placed in a system. Their goal is to minimize the number of instances where legitimate users touch their own decoys, while maximizing the frequency of interactions with honeyfiles by attackers. In the first experiment, four groups of thirteen students each were tasked with naming and placing 10, 20, 30 and 40 honeyfiles, respectively, on their systems. Their accidental access of the decoys was monitored for around seven days on average. It turned out that after the number of honeyfiles placed exceeded 20, the amount of false alarms increased super-linearly.

In the second experiment in [3], forty students were asked to act as hackers, and try to collect personal, potentially profitable information about their coworker who left for a 15 minute lunch break, leaving his or her computer unprotected. The student-hackers were given a scenario in which they were going through financial difficulties while the coworker was unfairly promoted over them. It was intended to make the information theft morally acceptable and justified. The experiment was conducted on four configurations of the target system, which contained 10, 20, 30 and 40 decoy documents, respectively. The median number of honeyfiles accessed was comparable for the last three options, and noticeably lower when 10 decoys were present. Combining the results of both experiments, the authors conclude that 20 is the optimum number of decoys, since it offers the best tradeoff between detection capability and non-interference with the regular workflow [3].

The authors of [3] also observe that decoys placed in easy to reach places, such as

subfolders of Desktop and the `Documents` folder, were accessed multiple times. They point out that files located deeper in the folder structure were touched less frequently, including nine honeyfiles that were completely ignored by all attackers. Therefore, the authors conclude that the conspicuousness, or visibility, of decoys is more important than their enticingness, or attractiveness of their name.

The third experiment in [3] measured the correlation between user motivation and the number of decoy accesses. Three groups of ten participant each were given one of three possible motivations: malicious - described above, benign - they were allowed to use a coworker's computer for 15 minutes to complete a task because their own computer suffered a hard disk failure, and neutral - they could use a coworker's computer for 15 minutes, but were given no reason to do so. The number of decoys accessed was highest for users with malicious intention and lowest for users with neutral intention. The authors conduct a T-test of decoy files' effectiveness and find that, with a 98% probability, they can detect a masquerader within 10 minutes of their activity on the victim's system with at least 90% probability.

Voris et al. [4] measure interference of decoy files with legitimate users' workflow by monitoring a group of 27 volunteers over a period of two months. This study generated more than 318 hours of data. Each volunteer had their honeyfiles placed according to one of two decoy placement arrangements offered by the Decoy Distributor Tool: "integrated" or "separated". The volunteers were asked to use their computers the same way they would do otherwise. The authors observe that the majority of the volunteers accessed honeyfiles at an average rate significantly lower than one decoy per fifteen minutes. This indicates that a simple threshold scheme could be used to differentiate between most masquerade attackers and legitimate users.

Voris et al. also attempt to experimentally evaluate the attack detection performance of the Decoy Distributor Tool proposed in [4]. They ask a group of college students to follow the same scenario used by Ben Salem and Stolfo in [3]. Folder naming convention used while deploying decoys was tailored to that scenario [4].

In addition to placing decoys using the Decoy Distributor Tool in both the "separated" and "integrated" configurations, Voris et al. also deployed honeyfiles manually as in [3]. Then, the authors measured how many times the students accessed the decoy files under the different arrangements.

The authors observe that there is no statistically significant difference between the number of files touched in the separated and manual configurations. Students accessed decoy files under the integrated configuration at a much lower rate compared to the manual and separated approaches. The authors also notice that student-hackers who knew about the presence of the decoy files accessed them at the same rate as students who did not know about them.

In [5], Voris et al. investigate the ability of honeyfiles to detect masquerade attacks using sophisticated statistics and machine learning. The data for the analysis was collected using an experimental setup similar to [3] and [4]. However, instead of asking student volunteers to act as hackers, they recruited 22 individuals, who were compensated $10 each for their participation in the experiment.

2.7 Discussion of Existing Experiments on Honeyfiles

I make observations about the experiments conducted in [3, 4] and [5], and use these observations to design the experiment in this work.

Some of the results presented in [3] and [4] can be generalized to scenarios beyond the specific group of students or volunteers that were used in the studies. These results are listed below.

1. Legitimate users access decoy files infrequently. In both [3] and [4], it was observed that decoy files could be added to legitimate users' systems without significantly interfering with their normal workflow. There is no reason to believe that another group of legitimate users would demonstrate behavior fundamentally different from the one observed in those studies.

2. The location of the honeyfiles is more important than the enticigness of their names

[3]. Even though the folders containing honeyfiles seem to have their names tailored to the attack scenario, files located in easy to reach places, especially in the subfolders of Desktop and `Documents` folder in the Windows OS, were accessed at a higher rate than files in harder to reach places.

3. Similarly, the optimal number of decoys, determined to be 20 [3], should apply to any system.

I also observe that the experiments conducted by [3, 4] and [5] suffer from the following limitations:

1. The scenario used to emulate hackers' behavior, while interesting, wasn't very realistic. In organizations where offices are arranged as open spaces with glass walls and doors, an attacker is unlikely to be able to sit at a coworker's computer for fifteen minutes without being noticed.

2. Employees are unlikely to store their personal data like medical records, mortgage information, bank statements etc. on their work computers. Employers may have policies that prohibit such practices. Besides, devices like laptops and tablets are widely available.

3. We learn from the Verizon's data breach investigation report [36] that during the analyzed period, 92% of the analyzed data breaches were perpetrated by external attackers. There is no reason to expect this number to change significantly over time. External attackers are not likely to have any kind of personal agenda against anyone in the targeted organization. They are more likely to be interested in information about Research & Development, patents or any kind of intellectual property. Each hacker may have a different plan of attack, or no plan at all. The scenario described above does not test the ability of decoy documents to detect the presence of such attackers.

4. Ben Salem and Stolfo [3] assume that all honeyfiles have a "shelf-life" and their effectiveness decreases as a function of time. However, the justification for this

assumption is unclear.

Therefore, the above scenario may fail to capture all possible ways in which hackers interact with bait files. In this work, I aim to investigate how real hackers interact with honeyfiles to better assess the usefulness of decoy documents in practice.

CHAPTER III

EXPERIMENTAL EVALUATION

The experiment was conducted on a virtual server in the Amazon Web Services cloud. To closely replicate a real-world scenario, I chose to use the Windows Server 2019 Operating System. All versions of Windows had a 72.1% share of the server operating systems market in 2019 [37]. Popularity among users translates into a higher number of attacks against an operating system. By choosing a popular OS, I was able to collect a large amount of data. The virtual server ran between the middle of November 2020 and late March 2021 over 133 consecutive days. The collected data was analyzed using Microsoft Excel.

The server was configured to be accessed using the Remote Desktop Protocol through a public IP address. One user account was configured with a weak password. The account's name is "admin", to give an impression that it is the system administrator's account, and the password is set to the word "password". According to [38], this password is the most popular in the US and the second most popular one in the world. Although the name intentionally suggests otherwise, the "admin" account is a standard one, i.e., it has no administrator's privileges, to prevent the hijacking of the server.

To prevent visitors from uploading or downloading any malware and using the server for attacking other targets, I disabled clipboard sharing through RDP and blocked all outbound connections using the Windows Defender Firewall. To make the server look realistic, nine fictitious users were created. The usernames were formed by concatenating first and family names randomly selected from the list of the most popular names [39] and surnames [40] in the United States, respectively. These accounts, as well as the "Administrator" account, were protected by strong passwords.

3.1 Non-Decoy File Placement

Before describing the placement of the decoy files, I describe the placement of non-decoy files, which were intended to make the virtual server appear more realistic to the attackers. Specifically, to make the "admin" account appear active, two hundred ninety three non-decoy documents were placed in its folders. The files were obtained from [41], the website of Microchip Technology Incorporated, a popular microcontroller manufacturer. Files containing data for nineteen families of 8-bit microchips were placed in nineteen subfolders of the `C:/Users/admin/Documents` folder. Each subfolder was named after one processor family and contains the files relevant to that family.

Moreover, copies of two non-decoy files, with names slightly modified to make them sound less technical, were placed on the Desktop folder, along with shortcuts to Adobe Reader and the `C:/Users/admin/Documents` folder. The shortcut to `C:/Users/admin/Documents` was intended to entice visitors to look there for more data files similar to the non-decoy files on the Desktop.

3.2 Decoy File Configuration Scenarios

To measure if attackers interact with honeyfiles at a higher rate than legitimate users, decoys were placed on the server in multiple configurations. In each honeyfile placement scenario, the server contains twenty decoy documents, which is the optimal number [3]. Also note that all the decoy file placement scenarios contain the non-decoy files described in Section 3.1. I describe in details the decoy file placement scenarios in the following sections.

3.2.1 Integrated Scenario I

The objective of this scenario is to measure the file access rate of attackers when honeyfiles are located directly in all existing folders among real files. This scenario follows the "integrated" pattern description in Voris et al. [4]. Specifically, a `todo.txt` honeyfile was placed on the Desktop folder. In each of the nineteen subfolders of the `C:/Users/admin/Documents` folder, a copy of the file containing the data sheet for the

entire family of processors was turned into a honey file by appending the word "Final", "Latest", "Update", "Updated", "Complete", "Detailed" or "Summary" to the original file's name, to satisfy the requirement of variability proposed by Bowen et al. [2]. This way of naming decoy documents is also similar to one of the methods used by Decoy Deployment Tool described by Voris et al. [4], where "-final" or "-updated" is appended to the name of an existing file. An underscore was added at the beginning of each resulting filename to maximize the file's visibility by ensuring it is listed as the first one in alphabetical order in the folder.

3.2.2 Separated Scenario

The objective of this scenario is to measure the file access rate of attackers when honeyfiles are placed in separate subfolders. This scenario followed the "separated" pattern described in Voris et al. [4]. The `todo.txt` file was moved from the Desktop to its newly created subfolder named `TODO`. In each subfolder of the `C:/Users/admin/Documents` folder, a new subfolder named `Summary` was created and the honeyfile was moved into it.

3.2.3 Mixed Scenario

The objective of this scenario is to measure the file access rate of attackers when ten decoy documents are placed according to the "integrated" and ten according to the "separated" pattern. This scenario was a combination of the first two scenarios. The `todo.txt` file was placed back directly on the Desktop and the empty `TODO` folder was deleted. In nine out of nineteen folders in `C:/Users/admin/Documents`, honeyfiles were moved back from the `Summary` subfolders, which were then deleted.

3.2.4 Integrated Scenario II

The scenario was guided by the results of the first three scenarios, where the `todo.txt` decoy document was by far the most popular file among the attackers. To determine if the popularity of the file is caused by its name, this scenario aims to measure the file access rate of attackers when the honeyfile located on the Desktop has a different name.

Specifically, a decoy document named `important.txt` was placed on the Desktop. The name was intended to imply that it contained more important information than the `todo.txt` file. Also, Ben Salem and Stolfo [3] reported that the most popular honeyfile in their experiment was located in the `C:/Documents and Settings/username/Desktop/Important/` folder. The rest of the files were arranged following the Integrated Scenario I described in Section 3.2.1. The only difference is that every interaction with `todo.txt` was counted as an access to a non-decoy file.

3.2.5 Decoy-less Scenario

The purpose of this scenario is to measure the impact that decoy files have on the average number of real documents touched by attackers. Even though the creators of the honeyfiles concept clearly intended them to be a detection tool, the deception aspect deserves some attention. If an attacker is accessing a decoy document, he is not accessing another file at the same time. He either has to stay connected longer to access the same number of real files, or touch fewer real files when decoys are present and he accesses the latter.

3.3 Data Collection

To record the attackers' behavior, I monitored their interactions with all of the placed data files. To this end, I used the auditing functionality of the Windows Server OS.

The Windows Server OS was configured using the Group Policy Editor to record successful and failed logon attempts, logoffs and access attempts to all data files used in this experiment. The system creates a set of entries for each such event in its Security log. For the purpose of this research, a "case" is a session in which an attacker successfully logs in and interacts in any way, including deletion attempt, with at least one data file, decoy or real. Since the concept of honeyfiles was only intended to detect attempts to exfiltrate data, I ignore sessions where the attackers tried to hijack the server and did not access any data documents.

For each case, the source IP address, logon and logoff times, number of honeyfile

interactions and number of real data files touched were extracted from the Windows Security log with the use of Event Viewer. Whenever the OS created more than one entry for a single logon event, the time stamp of the last of those entries was considered the start of the session. Duration of each case and the rates of file interactions per fifteen minutes were calculated. This allows comparisons with legitimate and masquerade users' decoy access frequency reported in most previous studies. Notes were also taken of which files were accessed and in what way.

If a decoy document was touched n times during the same case, it was recorded as n decoy interactions. The reason is that legitimate users can be expected to instantly realize they have accessed a honeyfile and to subsequently avoid this file. Therefore, multiple interactions with the same honeyfile in one logon session likely indicate an intrusion.

In contrast, n interactions with the same regular data file during one case were recorded as one regular file being accessed. The reason is that an adversary gains no additional information by opening the same regular data file multiple times.

3.4 Collected Data

The data consists of 536 cases for all scenarios collected with the help of the auditing functionality of the Windows Server. There are 110 cases for the Integrated Scenario I, 105 for the Separated Scenario, 107 for the Mixed Scenario, 104 for the Integrated Scenario II, and 110 for the Decoy-less Scenario.

Some general data was also collected to provide a broader context of the experiment. Out of more than 12.3 million failed logon attempts to the server recorded, around 64% were attempts to log into the "Administrator" account, which suggests that the attackers were primarily interested in hijacking the server. About 1% of all logon attempts targeted the "admin" account. Around 6.9% of those attempts were successful.

3.5 Experiment Results

The main objective of using decoy files is to detect intrusions into a system. To decide whether a user that interacts with decoy files is legitimate or malicious, we need to set a detection rule. In my experiment results, I use a threshold detection rule that works as follows. After a user interacts with data files on the virtual server, I calculate the number of times he or she interacts with decoy documents per fifteen minutes. If the number of interactions is above the threshold, the the user is classified as an attacker. Otherwise, the user is considered legitimate. However, in this experiment, all users are malicious, and thus when a user is classified as legitimate by the detection rule, it is always a false negative. For the same reason, there are no false positives and no true negatives.

To set the threshold of the detection rule, I use the rate at which legitimate users access decoy files observed in previous works. Specifically, [4] report that in the separated placement, all legitimate users accessed decoy files at a rate below 0.6 decoy documents interactions per fifteen minutes. In the integrated arrangement, at least 75% of legitimate users stayed below that level. Therefore, in this experiment every case in which the attacker accessed decoy documents at a rate of 0.6 per fifteen minutes or greater was considered detected by the honeyfile system. The remaining cases were counted as undetected.

3.5.1 Integrated Scenario I

Figure 1 shows the Integrated Scenario I cases in descending order of the honeyfile access rate of the corresponding attacker. Cases to the left of the red vertical line have an access rate above the threshold of 0.6 interactions per fifteen minutes. We observe a true positive rate of 86.36%, with 95 out of 110 cases being successfully detected. Thus, the honeyfiles in Integrated Scenario I along with the threshold detection rule can serve as an effective intrusion detection system.

Moreover, the file most commonly opened by the attackers was the `todo.txt` decoy on the Desktop. In 93 out of 95 detected cases, the attackers interacted with the `todo.txt` file, and in 89 of those cases it was the only honefile they interacted with. In all undetected

cases, the attackers only interacted with real files. This suggests that the name `todo.txt` has an effect on attackers' honeyfile access rates. I further investigate that effect in the Integrated Scenario II.

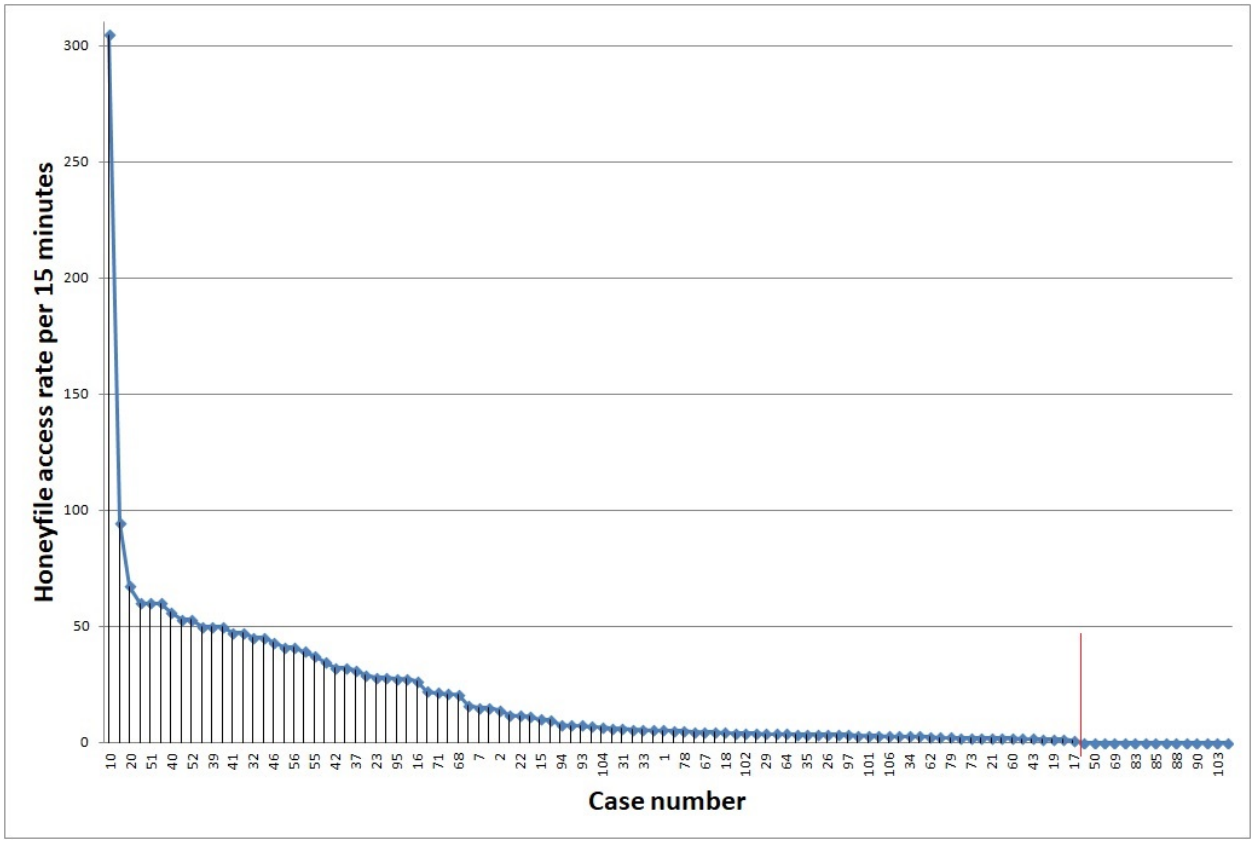

Figure 1: Distribution of decoy access rate for Integrated Scenario I

3.5.2 Separated Scenario

Figure 2 presents cases for Separated Scenario in descending order of the hacker's rate of interaction with decoys. Cases left of the red vertical line were detected due to their honeyfile access rates exceeding the threshold. 76 of 105 cases were detected, giving a 72.38% true positive rate. This scenario, combined with the threshold, can be used in practice as an intrusion detection system.

In 68 of the detected cases the attacker interacted with the `todo.txt` decoy document located in the `TODO` subfolder of the Desktop. In 63 of these cases `todo.txt` was the only honeyfile that the adversaries touched. Of the 29 undetected cases, only one

included an interaction with a decoy document, the `todo.txt` file. That case, however, lasted more than 30 minutes, which caused the honeyfile access rate to fall below the 0.6 per 15 minutes threshold. The `todo.txt` file once again showed that it has an important influence on attacker behavior that warrants further examination in the Integrated Scenario II.

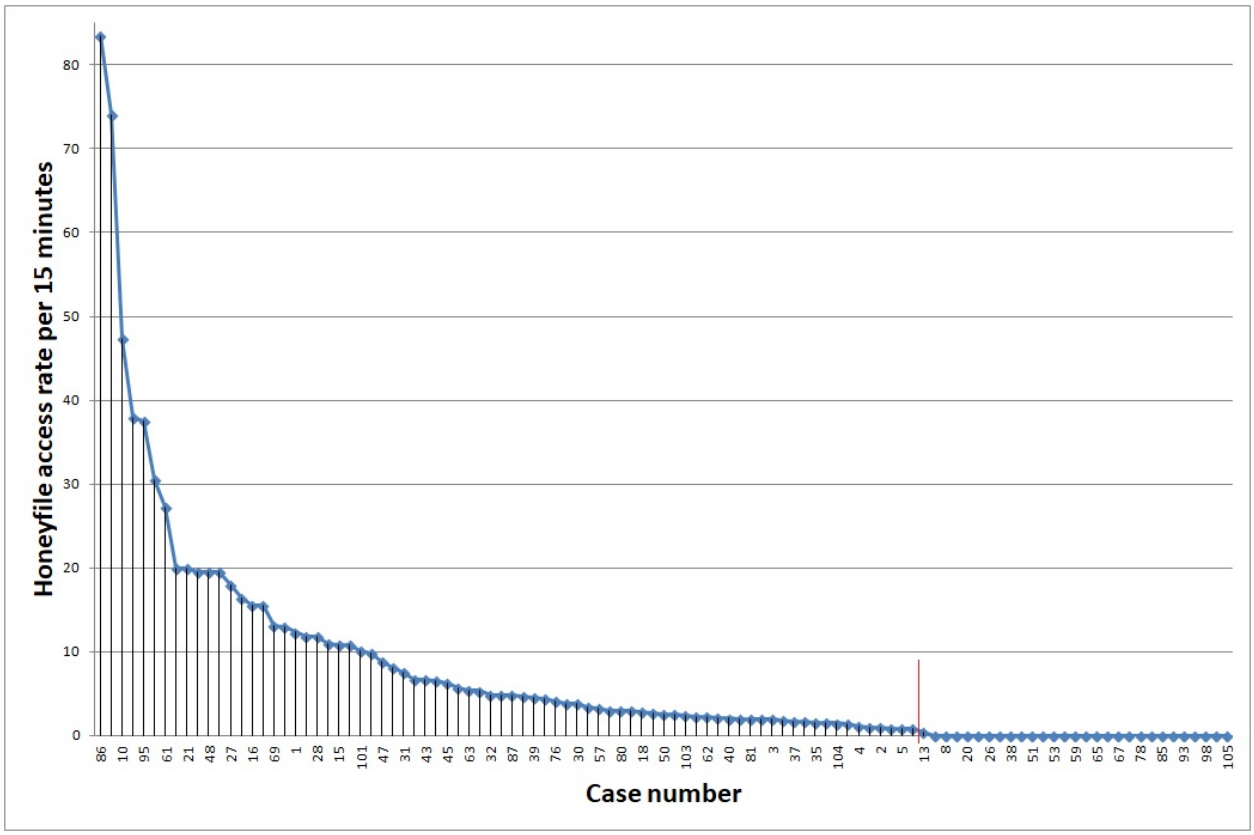

Figure 2: Distribution of decoy access rate for Separated Scenario

3.5.3 Mixed Scenario

The results of this scenario are shown in Figure 3 in descending order of decoy document access rate by the attacker. Due to their rate being above the threshold, cases to the left of the red vertical line were detected. Of 107 cases recorded, 84 were detected, resulting in a 78.50% true positive rate. These results show that the Mixed Scenario can be effective in real life as an intrusion detection system.

In all except 3 of the detected cases the attacker interacted with the `todo.txt`

decoy document located on the Desktop. 19 of 23 undetected cases involved no honeyfile interactions. In the remaining 4 attackers touched one decoy (`todo.txt` in each case), but spent so much time trying to hijack the server that the sessions lasted between 27 and more than 30 minutes, resulting in honeyfile access rate below the threshold. The effect of the `todo.txt` file was visible yet again.

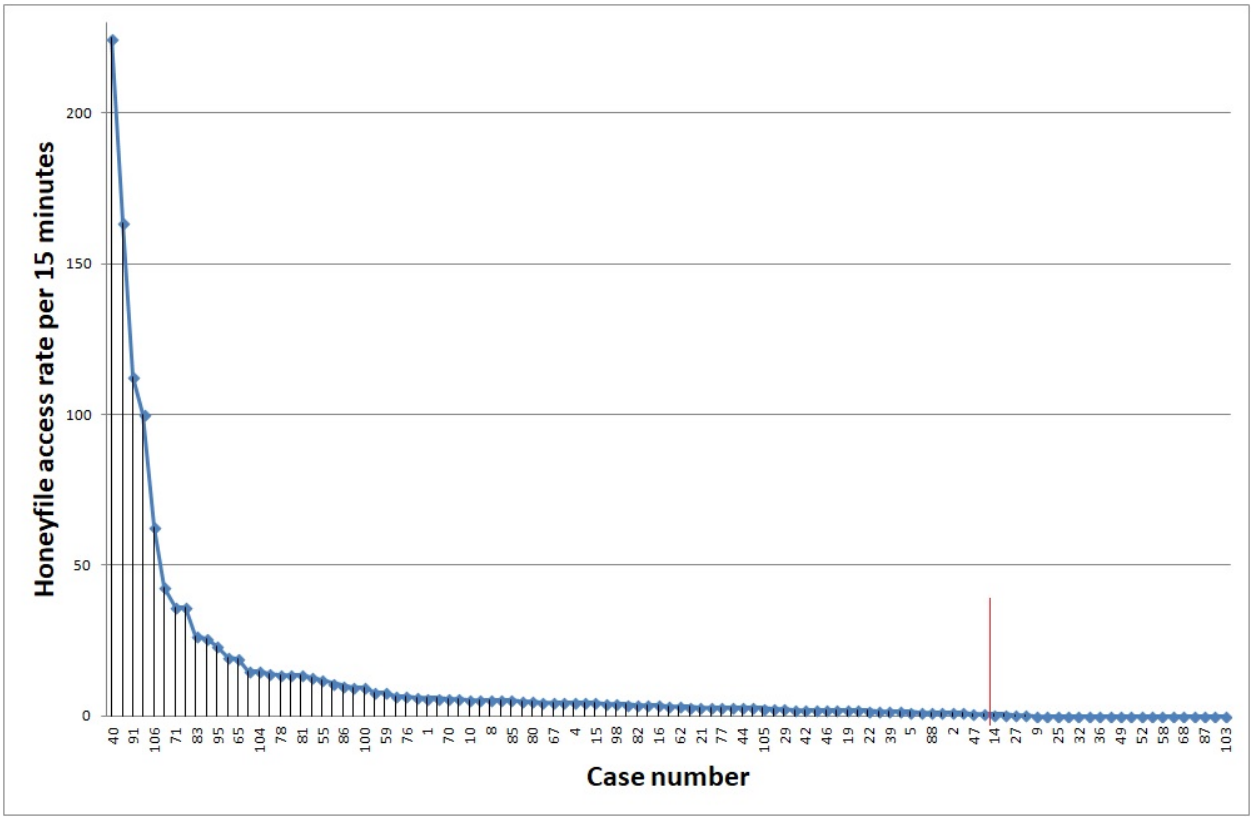

Figure 3: Distribution of decoy access rate for Mixed Scenario

3.5.4 Integrated Scenario II

Data for this scenario is presented in Figure 4 in descending order of honeyfile access rate by the adversary. Cases to the left of the red vertical line have an access rate above the threshold of 0.6 interactions per fifteen minutes and were therefore detected.

Out of 104 cases registered for this scenario, 73 were detected. That amounts to a 70.19% true positive rate. This would allow the scenario to be used as an intrusion detection system.

Of the 31 undetected cases, 29 involved no honeyfile interactions. In the other 2, attackers touched the "important.txt" decoy file on the Desktop, but the cases lasted more than 30 and more than 32 minutes, respectively. The honeyfile access rate was therefore below 0.6 per 15 minutes.

The results of this scenario show that the `todo.txt` file has a significant impact on attackers' behavior, but it is not the only one that attracts hackers' attention. There were 19 cases in which visitors interacted with the `important.txt` file, ignoring `todo.txt`. There were also, however, 20 cases when visitors touched `todo.txt` and ignored `important.txt`. Therefore, it is unlikely that another file, or least not one named `important.txt`, can completely eliminate interactions with a file named `todo`.

It is also worth noting that if `todo.txt` had been a second honeyfile in this scenario, the 2 cases with decoy access rate below the threshold, as well as 20 cases without decoy access, would have been detected. The resulting true positive rate would have equalled 95 out of 104, or 91.35%. This also shows that, while the `todo.txt` honeyfile was successful by itself, the system's performance would benefit from the presence of additional decoys on the Desktop.

3.5.5 Decoy-less Scenario

As its name indicates, this scenario contained no honeyfiles. It was intended to measure if the presence of decoy documents affects the number of interactions with regular data files. Table 1 shows the statistics of regular data documents touches per 15 minutes for all scenarios. We can see that the presence of honeyfiles reduced the attackers' interactions with regular files by at least half.

Note that although the median regular file access rate for two scenarios equals zero, it does not mean that honeyfiles completely eliminated interactions with regular files in those scenarios. The average values are above zero for all decoy scenarios, but lower than half of the average number of honeyfiles touched in the decoy-less scenario.

Figure 5 shows the split of all cases into 2 sets: those when attackers interacted only

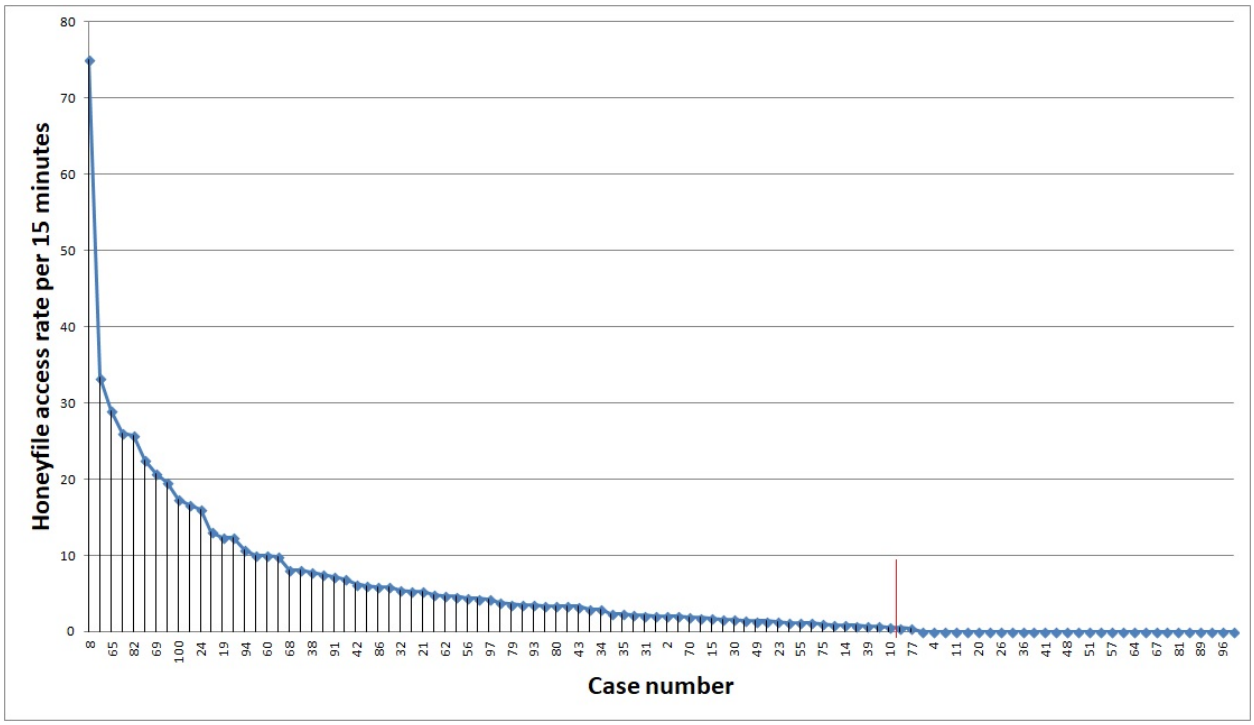

Figure 4: Distribution of decoy access rate for Integrated Scenario II

Scenario	Integrated 1	Separated	Mixed	Integrated 2	Decoy-less
Median	0	1.93	0	2.64	7
Average	56.46	29.08	12.54	7.91	117.02

Table 1: Access rate to regular files per 15 minutes.

with files located on the Desktop and those when they touched at least one file in the subfolders of the `Documents` folder. We can see that honeyfiles placed on the Desktop reduce the number of interactions with files not located on the Desktop. In particular, the Decoy-less scenario has the highest percentage of cases with file interactions inside the `Documents` folder, and is the only scenario where that percentage is above 50.

Moreover, from Figure 5 we can observe that among the decoy scenarios, the Separated Scenario has the highest rate of cases involving file touches inside the `Documents` folder. Since the Separated Scenario is the only honeyfile scenario with no decoys located directly on the Desktop (the `todo.txt` file was deployed inside the `TODO` subfolder of the Desktop), placing decoy files on the Desktop is more effective than in subfolders of the Desktop.

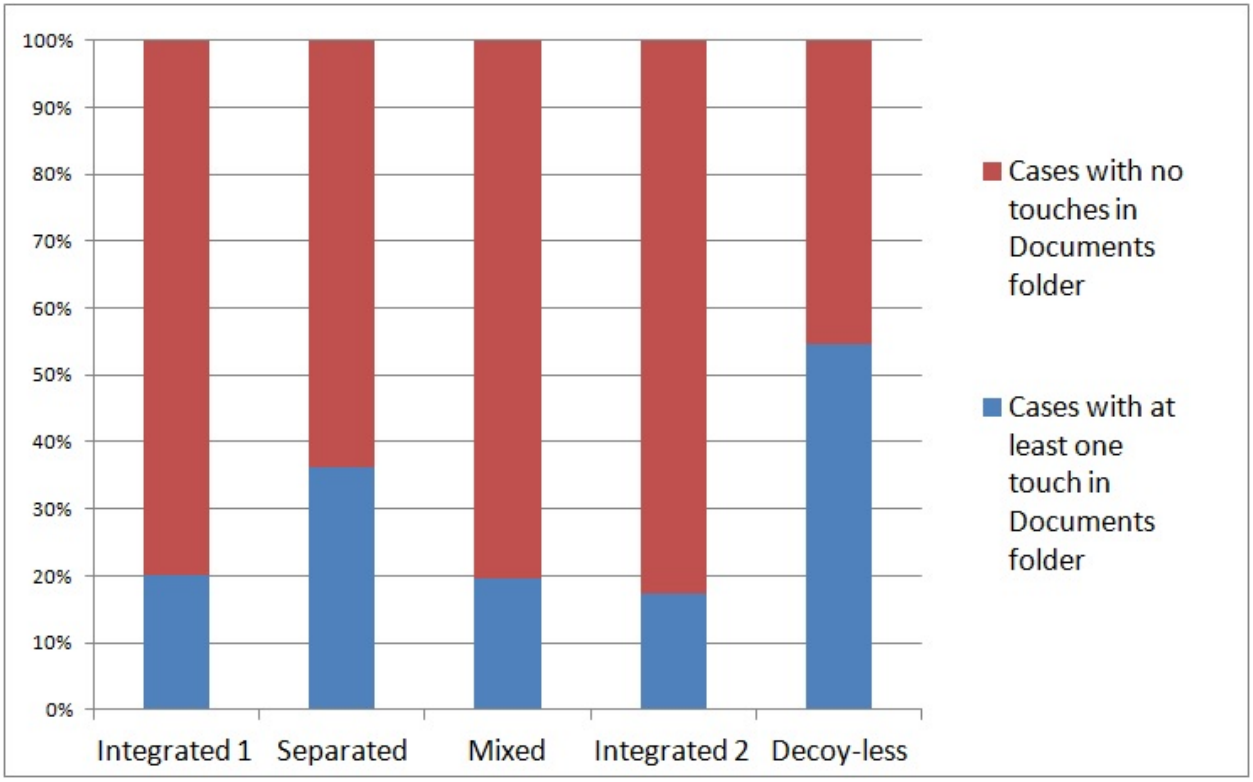

Figure 5: Cases with and without file access in Documents folder for each scenario

Scenario	Integrated 1	Separated	Mixed	Integrated 2
Maximum	304.6153846	83.41463415	225	75
Upper quartile	27.91193182	8.866995074	7.221867008	5.916575791
Median	5.0993749	2.601156069	3.383458647	2.054837364
Average	18.44259631	7.48539415	11.71851128	5.445331379
Lower quartile	2.175291307	0	1.109152042	0
Minimum	0	0	0	0

Table 2: Statistical data for decoy scenarios.

3.6 Comparison between decoy placement scenarios

To compare the performance of the different decoy placement scenarios, I plot the attacker decoy files access rate per 15 minutes for all four of those scenarios. I present the results in Figure 6, Figure 7, and Table 2.

Figure 6 shows the box and whisker plot of decoy access rate per 15 minutes for all four honeyfile scenarios. To improve readability, Figure 7 presents enlarged lower portion of Figure 6. I also provide Table 2 that shows the most important statistical data for all decoy scenarios.

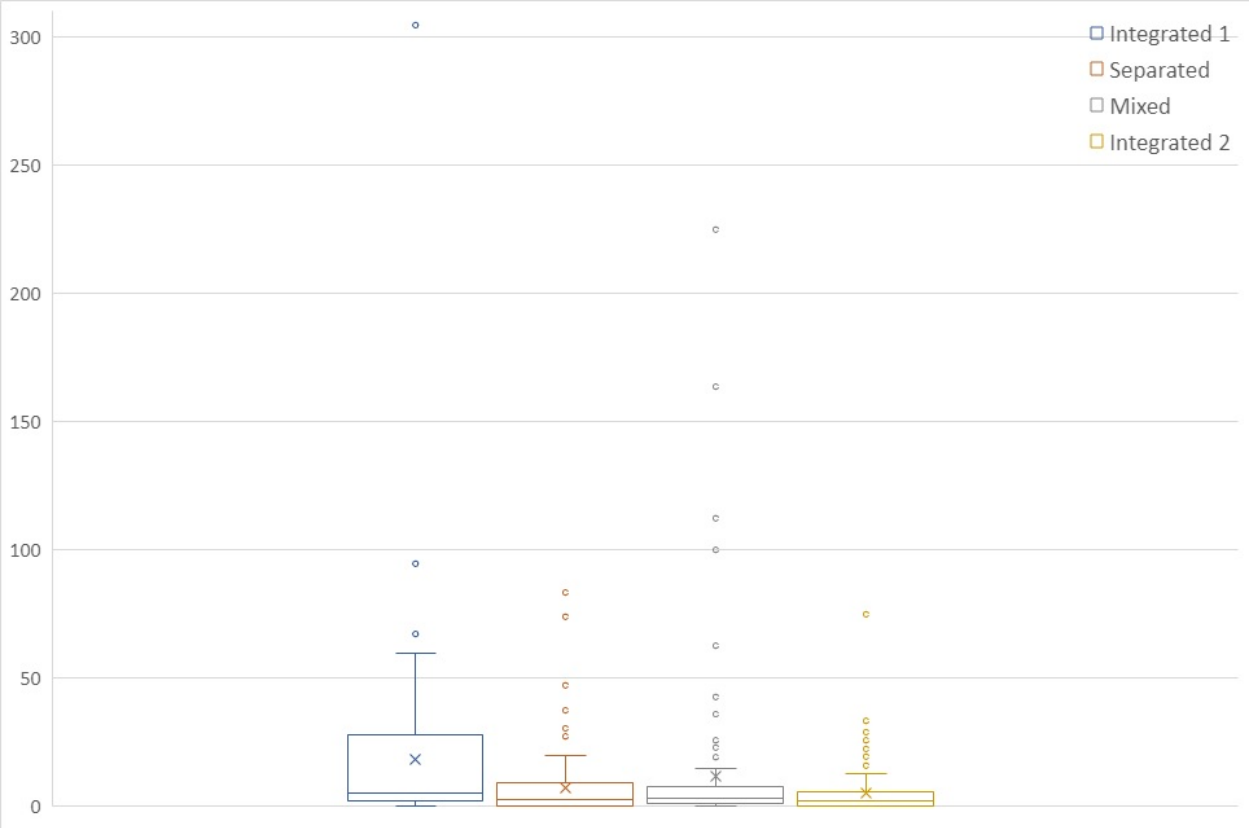

Figure 6: Comparison of decoy access rates per 15 minutes between decoy scenarios

Based on Figure 7 and Table 2, we can observe that the Integrated Scenario I had the highest median, average and maximum value of honeyfile interactions per fifteen minutes among all scenarios, followed by the Mixed, Separated and Integrated II Scenarios. Moreover, Figure 8 shows the detection rates for all decoy scenarios. We can see that the ranking of scenarios by the true positive rate looks exactly the same as their ranking by the median honeyfile access rate.

In contrast to lab experiments described in previous works [4, 5], where the "Integrated" file arrangement fared the worst, its equivalent here, Integrated Scenario I, had the highest median rate of decoy access per 15 minutes, and achieved the best detection rate.

Overall, the four decoy scenarios combined detected 328 of 426 cases recorded. This results in a 77% success rate. While not low, this rate leaves plenty of room for adversaries to cause significant harm.

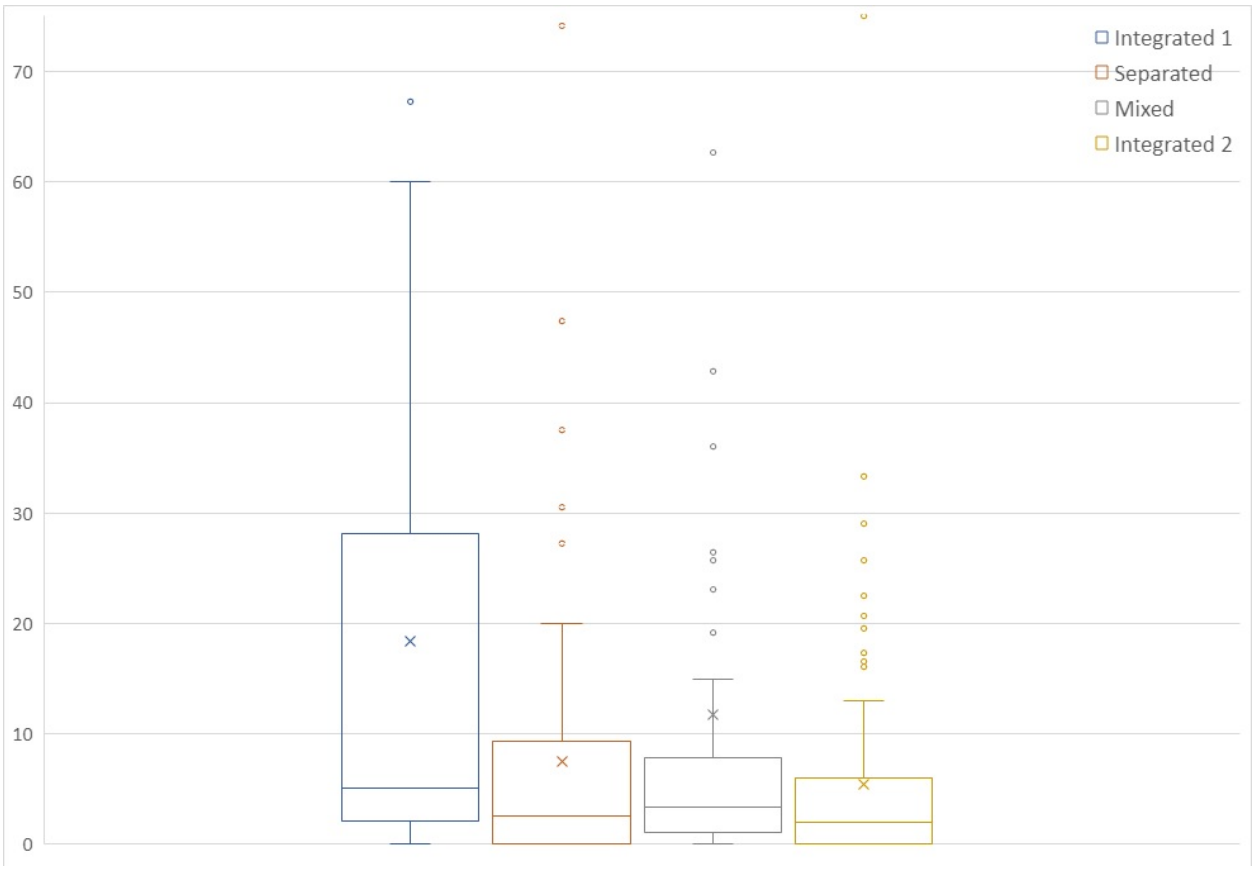

Figure 7: Enlarged lower portion of Figure 6

Figure 9 shows decoy access rates in all cases for all four decoy scenarios in chronological order. No declining trend in the decoy access rate can be observed here. Hence, contrary to expectations outlined in [3], there seems to be no evidence of a limited "shelf-life" of honeyfiles, or that life must be much longer than the duration of this experiment.

The observation that conspicuousness, or visibility of decoy documents, is more important than enticingness, or attractiveness of their names, made in [3], was confirmed in the real life experiment. Out of 328 detected cases, 298, or 90.85%, were detected only due to the adversary's interaction with a honeyfile placed directly on the Desktop, or in its subfolder (Mixed Scenario).

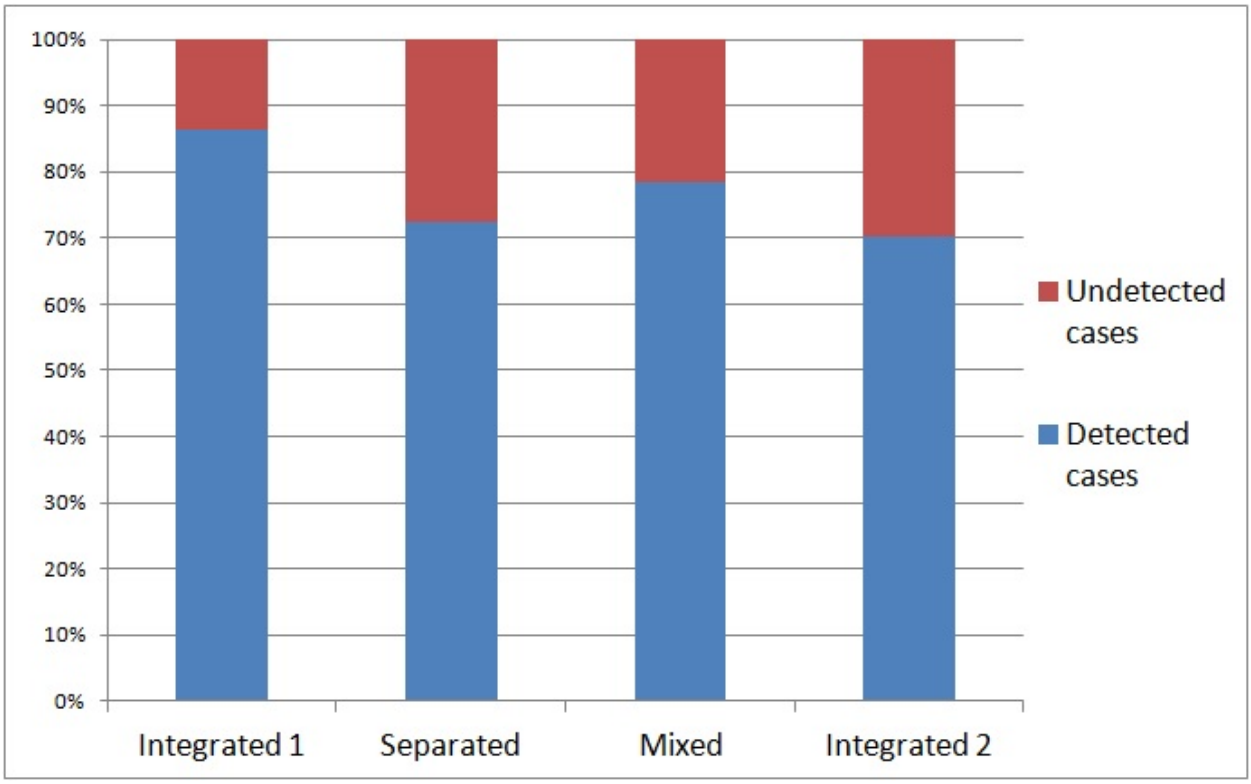

Figure 8: Detection rates for all decoy scenarios

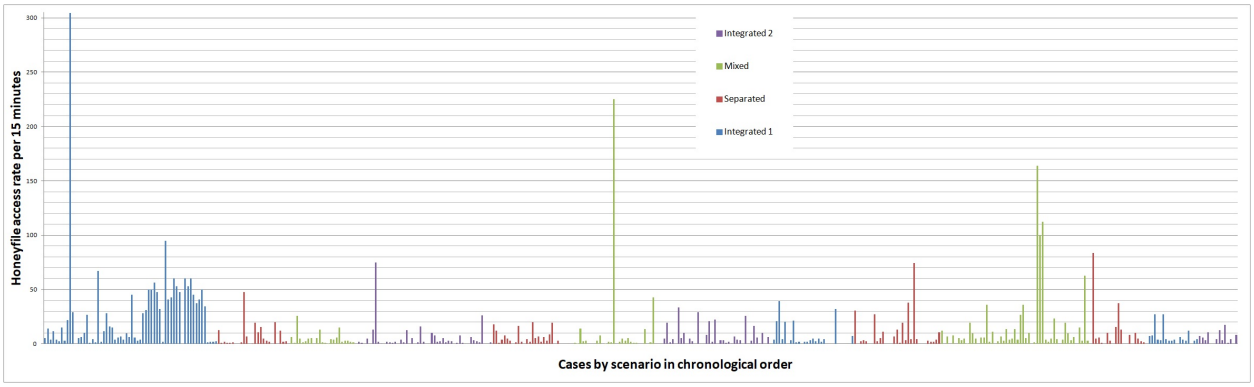

Figure 9: All cases for all decoy scenarios in chronological order

CHAPTER IV

DISCUSSION

It is important to mention that the score achieved by the Integrated Scenario I is very likely inflated by the timing. This scenario was in place primarily at the beginning of the experiment. After the server was indexed in early December 2020 by Shodan.io, the search engine for devices with open ports connected to the Internet, the interest in the server received a significant boost. The first 61 cases for the Integrated Scenario I were collected by December 21, 2020, and all but 2 of them were detected. The remaining 13 undetected cases were among the last 49 cases, recorded over 2 separate periods in March 2021. Those 49 cases had a 73.47% detection rate.

It seems safe to assume that any scenario in place when the server was indexed by Shodan.io would have received a similar boost. Without that boost, the Integrated Scenario I would have most likely performed like it did in cases collected later in the experiment.

That would have elevated the Mixed Scenario, a combination of Integrated and Separated arrangements, to the first place. This in turn would seem rather intuitive. Therefore, there is little basis to believe that decoy placement scenario is the most important factor in determining the ability of the honeyfile system to detect intrusions. Rather, visibility of decoys, or their proximity to the attacker's point of entry to the system, seems to be the most important, followed by enticing honeyfile names.

It is also worth noting that if the Integrated Scenario II involved 2 honeyfiles, with `todo.txt` being the 2nd decoy as discussed in Section 3.5.4, the scenario's efficiency would rise to 91.35%. Not only would it move from last to first place in the ranking, but it would be the only scenario with a true positive rate above 90%, as seen in Figure 8. It suggests that even with few icons on the Desktop, additional decoys placed there could improve the system's performance noticeably.

CHAPTER V

LIMITATIONS

The experiment in this work provides an observation of how attackers interact with honeyfiles in the real world. As in any experiment, there are some aspects that were not covered and certain questions that require further investigation. I summarize these limitations in the following.

1. Due to the location of the virtual server in the North Virginia region of Amazon Web Services, attackers may have been able to infer that the server was a research project. It is well known that all virtual machines created as part of AWS Educate service are placed in the North Virginia region, which may have affected the behavior of attackers interacting with the server.

2. Due to time constraints and the success of decoys placed on the Desktop in limiting interactions with files located in subfolders of the `Documents` folder, there is not enough data to determine if the decoy naming convention, patterned after the Decoy Deployment Tool [4], was successful.

3. Due to time constraints, the non-decoy documents on the server consisted of only one type of files, Portable Document Format, originating from a single source. Thus, the effect of different types of data files on the attackers' behavior was not observed.

4. Due to time constraints I could not research how attackers behave and how many decoys are needed on the Desktop when there are multiple icons there. On a desktop containing dozens of icons, one or two honeyfiles may become barely noticable and the resulting true positive rate may be lower than in this experiment.

CHAPTER VI

FURTHER RESEARCH

During this experiment I identified the following topics that need to be further investigated.

Given that out of 98 undetected cases, 90, or 91.84%, involved no honefyile interactions, further research on naming and placement of decoys is obviously warranted and necessary.

The honeyfile concept could also be extended to some system files. These are essential files that are needed by the operating system to perform its functions like allocating memory and processor use, running applications or controlling which users and programs can access certain resources. Due to the fact that Object Access auditing was apparently turned on by default on certain system files during the installation of the Windows Server 2019 OS in the virtual server, I inadvertently recorded at least some activities of visitors who tried to hijack the server. A legitimate user of a standard account has no reason to access administrative tools like Local User and Group Management, Group Policy Editor, Component Services, try to modify Windows Defender Firewall settings or share the system partition `C:` with anyone over the Internet, to name just a few examples.

Even a single attempt to use some of the functionalities mentioned above should most likely raise an alarm. Furthermore, all interactions with important system files or tools could be treated on par with decoy document touches, added to the count of honeyfile interactions and used in calculating the decoy access rate. This would not only improve the system's efficiency, but could also help detect intrusions in which the attacker focuses on hijacking the server and does not interact with any data files.

Another way of increasing the effectiveness of decoy documents could include measuring the duration of period when a honeyfile is open. A legitimate user, who accesses their own trap by accident, will quickly realize their mistake and close the file. One or two seconds should be enough for it, but some practical experiment could determine the exact

number. If a decoy is open longer than that, it most likely indicates the presence of an attacker, even if the honeyfile access rate per unit of time remains within the range typical for legitimate users.

This leads us to another important topic that deserves further research - decoy believability, or the ability of honeyfiles to pretend that they contain valuable information. This concept was briefly introduced by Yuill et al. [1] with the focus on ensuring that decoys pretending to be in-use files have their modification and access times updated regularly. Bowen et al. [2] conduct a more thorough theoretical analysis of decoy believability, but note that this property may be less important than others, since the attacker may have to open a decoy to determine if it's a real document or not, and the act of opening may be enough to detect the presence of an adversary. Ben Salem and Stolfo [3] go as far as saying that mere access to a decoy document constitutes evidence of a masquerade attack.

We already know that a single interaction with a honefile may be caused by the legitimate user, and may not suffice to detect an adversary. It may take multiple interactions with one or more decoys, or keeping such a document open for an extended period of time. In both cases, honeyfiles' contents should make them look like real documents.

To this end, Voris et al. [42] investigated the use of automated translation in creating foreign language decoy documents. As such, these would be easier to identify as traps for legitimate users, while forcing attackers to exfiltrate the files' contents to translate them. That action should provide an obvious signal of malicious intent. The method would also allow the use of publicly available articles as decoys. Their origin could be obfuscated by multiple successive translations between a few languages.

CHAPTER VII

CONCLUSION

This thesis describes an experiment in which a server containing decoy documents, deployed in a systematic way, was for the first time exposed to attacks by real hackers over the Internet. It was also the first known case where real adversaries' rate of interaction with honeyfiles was measured. Two decoy placement arrangements from previous research, as well as three new arrangements were tested. The collected results were analyzed thoroughly and showed that the concept of decoy documents has a great potential in practice. Even though the honeyfile access rate recorded during the real life experiment was lower than in lab conditions, it was mostly higher than the rate for legitimate users. That allows the use of a threshold to detect intrusions.

The concept of decoy documents has passed a practical test. A combination of honeyfiles and a threshold was able to detect 77% of observed attacks. The details of the system need some refining and fine tuning, but it can definitely be used in real-life implementations to improve the effectiveness of existing Intrusion Detection Systems.

It is also worth pointing out that quite a few visitors tried to use the Remote Desktop Protocol clipboard to upload something, most likely malware, to the server. It is not unthinkable for organizations to enable clipboard sharing through RDP to allow people to upload their work done at home. If such an account is compromised, there is little to prevent an adversary from using the server in attacks. This reminds us once again that strong passwords are absolutely essential, and tools like Intrusion Detection Systems or honeyfiles should be used as the last line of defense, as a supplement and not a replacement for perimeter defenses.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] J. Yuill, M. Zappe, D. Denning, and F. Feer, "Honeyfiles: deceptive files for intrusion detection," in *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.*, pp. 116–122, IEEE, 2004.

[2] B. M. Bowen, S. Hershkop, A. D. Keromytis, and S. J. Stolfo, "Baiting inside attackers using decoy documents," in *International Conference on Security and Privacy in Communication Systems*, pp. 51–70, Springer, 2009.

[3] M. B. Salem and S. J. Stolfo, "Decoy document deployment for effective masquerade attack detection," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 35–54, Springer, 2011.

[4] J. Voris, J. Jermyn, N. Boggs, and S. Stolfo, "Fox in the trap: Thwarting masqueraders via automated decoy document deployment," in *Proceedings of the Eighth European Workshop on System Security*, pp. 1–7, 2015.

[5] J. Voris, Y. Song, M. B. Salem, S. Hershkop, and S. Stolfo, "Active authentication using file system decoys and user behavior modeling: results of a large scale study," *Computers & Security*, vol. 87, p. 101412, 2019.

[6] J. Bonneau, "The science of guessing: analyzing an anonymized corpus of 70 million passwords," in *2012 IEEE Symposium on Security and Privacy*, pp. 538–552, IEEE, 2012.

[7] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, and J. Lopez, "Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms," in *2012 IEEE symposium on security and privacy*, pp. 523–537, IEEE, 2012.

[8] M. Weir, S. Aggarwal, B. De Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *2009 30th IEEE Symposium on Security and Privacy*, pp. 391–405, IEEE, 2009.

[9] S. Schechter, C. Herley, and M. Mitzenmacher, "Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks," in *Proceedings of the 5th USENIX conference on Hot topics in security*, pp. 1–8, 2010.

[10] D. Perito, C. Castelluccia, M. A. Kaafar, and P. Manils, "How unique and traceable are usernames?," in *International Symposium on Privacy Enhancing Technologies Symposium*, pp. 1–17, Springer, 2011.

[11] S. Houshmand and S. Aggarwal, "Building better passwords using probabilistic techniques," in *Proceedings of the 28th Annual Computer Security Applications Conference*, pp. 109–118, 2012.

[12] A. Forget, S. Chiasson, P. C. Van Oorschot, and R. Biddle, "Improving text passwords through persuasion," in *Proceedings of the 4th symposium on Usable privacy and security*, pp. 1–12, 2008.

[13] Y. Zhang, F. Monrose, and M. K. Reiter, "The security of modern password expiration: An algorithmic framework and empirical analysis," in *Proceedings of the 17th ACM conference on Computer and communications security*, pp. 176–186, 2010.

[14] S. Schechter, A. B. Brush, and S. Egelman, "It's no secret. measuring the security and reliability of authentication via "secret" questions," in *2009 30th IEEE Symposium on Security and Privacy*, pp. 375–390, IEEE, 2009.

[15] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda, "All your contacts are belong to us: automated identity theft attacks on social networks," in *Proceedings of the 18th international conference on World wide web*, pp. 551–560, 2009.

[16] A. Kak, "Mounting targeted attacks with trojans and social engineering." engineering.purdue.edu/kak/compsec/NewLectures/ Lecture30.pdf, 2020.

[17] A. K. Ghazi-Tehrani and H. N. Pontell, "Phishing evolves: Analyzing the enduring cybercrime," *Victims & Offenders*, vol. 16, no. 3, pp. 316–342, 2021.

[18] M. B. Salem, S. Hershkop, and S. J. Stolfo, "A survey of insider attack detection research," in *Insider Attack and Cyber Security*, pp. 69–90, Springer, 2008.

[19] M. Schonlau, W. DuMouchel, W.-H. Ju, A. F. Karr, M. Theus, and Y. Vardi, "Computer intrusion: Detecting masquerades," *Statistical science*, pp. 58–74, 2001.

[20] J. Yuill, D. Denning, and F. Feer, "Using deception to hide things from hackers: Processes, principles, and techniques," *Journal of Information Warfare*, vol. 5, no. 3, pp. 26–40, 2006.

[21] F. Cohen, "The use of deception techniques: Honeypots and decoys," *Handbook of Information Security*, vol. 3, no. 1, pp. 646–655, 2006.

[22] X. Han, N. Kheir, and D. Balzarotti, "Deception techniques in computer security: A research perspective," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–36, 2018.

[23] D. Climek, A. Macera, and W. Tirenin, "Cyber deception," *Journal of Cyber Security and Information Systems*, vol. 4, no. 1, pp. 14–17, 2016.

[24] L. Spitzner, "Honeypots: Catching the insider threat," in *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, pp. 170–179, IEEE, 2003.

[25] T. Barron and N. Nikiforakis, "Picky attackers: Quantifying the role of system properties on intruder behavior," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, pp. 387–398, 2017.

[26] L. Spitzner, "Honeytokens: The other honeypot." community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=74450cf5-2f11-48c5-8d92-4687f5978988&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68&tab=librarydocuments, 2003.

[27] F. Araujo, K. W. Hamlen, S. Biedermann, and S. Katzenbeisser, "From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pp. 942–953, 2014.

[28] A. Juels and R. L. Rivest, "Honeywords: Making password-cracking detectable," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 145–160, 2013.

[29] I. Anjum, M. Zhu, I. Polinsky, W. Enck, M. K. Reiter, and M. Singh, "Role-based deception in enterprise networks," *arXiv preprint arXiv:2008.02979*, 2020.

[30] D. B. Rawat, N. Sapavath, and M. Song, "Performance evaluation of deception system for deceiving cyber adversaries in adaptive virtualized wireless networks," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, pp. 401–406, 2019.

[31] S. Achleitner, T. La Porta, P. McDaniel, S. Sugrim, S. V. Krishnamurthy, and R. Chadha, "Cyber deception: Virtual networks to defend insider reconnaissance," in *Proceedings of the 8th ACM CCS international workshop on managing insider security threats*, pp. 57–68, 2016.

[32] A. Underbrink, "Effective cyber deception," in *Cyber Deception*, pp. 115–147, Springer, 2016.

[33] K. Horák, Q. Zhu, and B. Bošanskỳ, "Manipulating adversary's belief: A dynamic game approach to deception by design for proactive network security," in *International Conference on Decision and Game Theory for Security*, pp. 273–294, Springer, 2017.

[34] C. Stoll, *The cuckoo's egg: tracking a spy through the maze of computer espionage*. Simon and Schuster, 2005.

[35] J. Lee, J. Choi, G. Lee, S.-W. Shim, and T. Kim, "PhantomFS: File-based deception technology for thwarting malicious users," *IEEE Access*, vol. 8, pp. 32203–32214, 2020.

[36] W. Baker, A. Hutton, C. D. Hylender, J. Pamula, C. Porter, and M. Spitler, "Data breach investigations report." cybersecurity.idaho.gov/wp-content/uploads/sites /87/2019/04/data-breach-investigations-report-2013.pdf, 2013.

[37] T. Alsop, "Global server share by operating system 2018-2019." www.statista.com/statistics/915085/global-servershare-by-os/, 2020.

[38] M. Marino, "The 20 most hacked passwords in the world: Is yours here?." www.safetydetectives.com/blog/the-most-hacked-passwords-in-the-world/, 2020.

[39] "Top names over the last 100 years." www.ssa.gov/oact/babynames/decades/century.html, 2020.

[40] K. Powell, "Top 100 most common last names in the United States." www.thoughtco.com/most-common-us-surnames-1422656, 2020.

[41] Microchip Technology Incorporated. www.microchip.com, 2020.

[42] J. Voris, N. Boggs, and S. J. Stolfo, "Lost in translation: Improving decoy documents via automated translation," in *2012 IEEE Symposium on Security and Privacy Workshops*, pp. 129–133, IEEE, 2012.