USING DECEPTION TO ENHANCE SECURITY: A TAXONOMY, MODEL, AND NOVEL USES

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Mohammed H. Almeshekah

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2015

Purdue University

West Lafayette, Indiana

In dedication to my mother, Aljoharh, who has given up so much to make me into who I am; to my wife, Asma, who taught me the meaning of sacrifice; and to my sons, Abdullah and Feras, who provided me the extra motivation to finish my PhD.

ACKNOWLEDGMENTS

I would like to express my heartfelt thanks to my advisors, Prof. Eugene H. Spafford and Prof. Mikhal J. Atallah, for their time, guidance, and invaluable comments throughout the whole span of this dissertation work. I am honored to have the opportunity to learn from such renowned scientists and highly respected mentors. They taught me the skills that would guide me in my career for years to come. I am forever indebted to them and aspire to follow in their footsteps. I would also like to express my deepest appreciation to my committee members, Prof. Samuel Wagstaff and Prof. Matt Bishop, for their invaluable advice and help. Without their insights and hard questions, this dissertation would not have been possible.

I am also extremely grateful for my beloved wife, Asma, for her continuous encouragement and unwavering support. She provided light when the rigors of intellectual pursuit were casting a shadow. I would not have finished this dissertation without her by my side. She has sacrificed beyond what I wished for and has done so with love. Also, I am grateful for my parents for their unwavering support and unforgettable endorsement, especially to my dearest mom. She is the pillar I stand by during hard times and I owe her my life for her constant love and encouragement.

Special thanks are also due to Northrop Grumman and Saudi Arabian Cultural Mission for supporting me throughout my PhD. Finally, I take this opportunity to record my sincere thanks to all the faculty, staff, and friends at the Computer Science Department and CERIAS; they provided me with one of the best academic environment during my PhD.

TABLE OF CONTENTS

Page

LIST OF TABLES viii

LIST OF FIGURES ix

ABBREVIATIONS xi

ABSTRACT xii

1 Introduction 1
- 1.1 Motivation and Overview 1
  - 1.1.1 Thesis Statement 2
  - 1.1.2 Dissertation Overview 3
- 1.2 Terminology 4
- 1.3 Dissertation Organization and Contribution 5

2 A Taxonomy of Computer Systems' Defenses 8
- 2.1 The Four Categories of Protection Mechanisms 10
  - 2.1.1 Denial and Isolation 12
  - 2.1.2 Degradation and Obfuscation 13
  - 2.1.3 Negative Information and Deception 13
  - 2.1.4 Attribution and Counter-Operations 16
- 2.2 Fitting the Pieces Together – Cyber Kill-Chain Model 17
  - 2.2.1 The Role of Deception 17
- 2.3 Chapter Summary 19

3 Deception 21
- 3.1 General Definition of Deception 21
- 3.2 Deception and the Truth – A Taxonomy 22
  - 3.2.1 Simulation and Dissimulation – Bell and Whaley 22
  - 3.2.2 Linguistic Case Theory 24
- 3.3 Deception Maxims 25
  - 3.3.1 Truth/Reality 25
  - 3.3.2 Deceit 26
  - 3.3.3 Denial, Misdirection and Confusion 26
- 3.4 Deception and Biases 26
  - 3.4.1 Personal Biases 28
  - 3.4.2 Cultural Biases 28
  - 3.4.3 Organizational Biases 30

Page
3.4.4 Cognitive Biases 30
3.5 The Use of Deception in War, Military and Conflicts 31
3.6 General Use of Deception in Computing 32
3.6.1 In Human-to-Human Digital Interaction 32
3.6.2 In Human Computer Interaction (HCI) 32
3.6.3 In Robotics and Human Robot Interaction (HRI) 34
3.6.4 In Computer-to-Computer Interaction 35
3.7 The Use of Deception to Enhance Security 36
3.7.1 Honeypots 37
3.7.2 Honey–* Tools 39
3.7.3 Incorporating Deception into Other Security Defenses 40
3.8 Deception Operations and Tactics 43
3.9 Chapter Summary 44

4 A Framework for Using Deception to Enhance Security 45
4.1 Definition 46
4.2 Limitations of Isolated Use of Deception 46
4.3 The Role of Deception 47
4.3.1 Advantages of Using Deception in Computer Defenses 48
4.4 Related Work of Modeling the Use of Deception in Security 50
4.5 A Framework for Integrating Deception-Based Defenses 51
4.5.1 Planning Deception 52
4.5.2 Implementing and Integrating Deception 62
4.5.3 Monitoring and Evaluating the Use of Deception 62
4.6 Deception and Related Concepts 63
4.6.1 Kerckhoff's Principle and Deception 63
4.6.2 Deception and Hacking Back 65
4.6.3 Deception and Consistency 65
4.6.4 Deception and Abstraction 66
4.7 Applying the Framework 66
4.7.1 To Previous Uses of Deception 66
4.7.2 To the Work in This Dissertation – A Case Study 70
4.8 Chapter Summary 73

5 Deceptive Covert Channel 74
5.1 Background 75
5.1.1 Authentication Schemes 75
5.1.2 Use of Smartphones 77
5.1.3 Use of Deception and Covert Channels 77
5.2 Creating a Deceptive Covert Channel 78
5.2.1 Threat Model 79
5.2.2 Scheme's Setup 80
5.2.3 Logging In 81

Page
5.2.4 Creating Deceit and Covert Communication 85
5.3 Enhancements 86
5.4 Security Analysis 87
5.5 Comparison with Other Schemes 89
5.6 Chapter Summary 92
6 Deceptive Passwords — ErsatzPasswords 93
6.1 Background 94
6.1.1 Passwords 94
6.1.2 Password-Related Threats 94
6.1.3 Injecting Deceit 96
6.2 Technical Specification 97
6.2.1 Background 97
6.2.2 One-time Initialization 99
6.2.3 Login 101
6.2.4 Password Administration 102
6.3 ErsatzPasswords – The Use of Deception 104
6.3.1 ErsatzPasswords Generation 105
6.3.2 ErsatzPasswords Properties 108
6.4 Implementation and Analysis 111
6.4.1 Implementation Details 111
6.4.2 Analysis 112
6.5 Chapter Summary 118
7 Deceptiver — A Centralized Deceptive Server 120
7.1 Background 121
7.2 Overview 122
7.3 Deceptiver Design 123
7.3.1 Deceptive Responses 124
7.3.2 Centralized Deception 130
7.4 Implementation and Deployment 130
7.4.1 Apache Server Hook 131
7.5 Security Discussion 132
7.6 Performance Analysis 134
7.7 Chapter Summary 136
8 Conclusions 137
8.1 Summary 137
8.2 Future Work 138
LIST OF REFERENCES 141
A Deceptiver Implementation 153
A.1 Deceptiver Command Line Inputs 153

VITA 155

LIST OF TABLES

Table Page

2.1 Mapping Security Mechanisms to the Kill-Chain Model 18

4.1 Using the Deception Framework to Secure Web Applications 72

5.1 Schemes Comparison 90

6.1 Number of Instructions for Creating a New User Under the new `pam_unix` Module 115

7.1 Comparing the Performance of Apache With ModSecurity and Deceptiver 134

LIST OF FIGURES

Figure Page

1.1 Computer Systems Components Where Deception is Applied in the Dissertation . . . 3

1.2 Using Deception to Enhance Security – Dissertation Overview . . . 4

2.1 Taxonomy of Information Protection Mechanisms . . . 9

2.2 Plotting the Taxonomy Over Multiple Scales . . . 11

3.1 Deception and Perception . . . 22

3.2 Deception Target Biases . . . 27

4.1 Framework to Incorporate Deception in Computer Security Defenses . . . 52

4.2 Computer Systems Components Where Deception Can Be Integrated With 55

4.3 Creating Deceit . . . 60

4.4 System Components Used to Create Deceit in Rowe's Work . . . 67

4.5 System Components Used to Create Deceit in Bowen et al. Work . . . 69

4.6 Web Application Case Study . . . 70

4.7 Web Application Generic Threats . . . 71

4.8 Web Application Deception-Based Defenses . . . 73

5.1 Man-in-the-Middle (MitM) vs. Man-in-the-Browser (MitB) . . . 75

5.2 Protocol Run . . . 82

5.3 Sending a Covert Message . . . 84

6.1 Distribution of Password Update Latency in the Ersatzpassword Scheme 113

6.2 Distribution of Password Update Latency in the Original `pam_unix` . . . 113

6.3 Comparison of Password Update Latency Between Ersatzpassword and Original `pam_unix` . . . 114

6.4 Comparison of Real and Decoy Password Authentication in the Ersatzpassword Scheme . . . 114

6.5 Distribution of User Authentication Latency in the Ersatzpassword scheme 116

Figure Page

6.6 Distribution of Detecting the Use of an Ersazpassword to Authentication 117
6.7 Distribution of User Authentication Latency in the Original `pam_unix` . 117
6.8 Comparison of Password Authentication Latency Between Ersatzpassword and the Original `pam_unix` 118
7.1 Deceptiver's Overall Design 124
7.2 An Example of a Deceptiver Trap in `.htaccess` 125
7.3 An Example of a Deceptiver Trap – Response Meta Data 127
7.4 Users Categories in Deceptiver 128
7.5 Admin Section of Deceptiver 131
7.6 Intercepting All Requests Inside Apache 132
7.7 Hooking Deceptiver with Apache 132

ABBREVIATIONS

A	Accumulation Function
APT	Advanced Persistent Threats
DBIR	Verizon's Data Breach Investigation Report
DEP	Data Execution Prevention
DNS	Domain Name System
DTK	Cohen's Deception Toolkit
H	Hash Function
HCI	Human Computer Interaction
HIDS	Host Intrusion Detection System
HIPS	Host Intrusion Prevention System
HMAC	Keyed-Hash Message Authentication Code
HRI	Human-Robot Interaction
HSM	Hardware Security Module
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
NAT	Network Address Translation
OWASP	The Open Web Application Security Project
RBAC	Rule-Based Access Control
SQL	Structured Query Language
VPN	Virtual Private Network
WAF	Web Application Firewall
XSRF	Cross-Site Request Forgery
XSS	Cross-Site Scripting

ABSTRACT

Almeshekah, Mohammed H. PhD, Purdue University, August 2015. Using Deception to Enhance Security: A Taxonomy, Model, and Novel Uses. Major Professors: Eugene H. Spafford and Mikhail J. Atallah.

As the convergence between our physical and digital worlds continue at a rapid pace, securing our digital information is vital to our prosperity. Most current typical computer systems are unwittingly helpful to attackers through their predictable responses. In everyday security, deception plays a prominent role in our lives and digital security is no different. The use of deception has been a cornerstone technique in many successful computer breaches. Phishing, social engineering, and drive-by-downloads are some prime examples. The work in this dissertation is structured to enhance the security of computer systems by using means of deception and deceit.

Deception-based security mechanisms focus on altering adversaries' perception of computer systems in a way that can confuse them and waste their time and resources. These techniques exploit adversaries' biases and present them with a plausible alternative to the truth bringing a number of unique advantages to computer security. In addition, deception has been widely used in many areas of computing for decades and security is no different. However, deception has only been used haphazardly in computer security.

In this dissertation we present a framework where deception can be planned and integrated into computer defenses. We posit how the well-known Kerckhoffs's principle has been misinterpreted to drive the security community away from deception-based mechanisms. We present two schemes that employ deception to protect users' passwords during transmission and at rest when they are stored on a computer server. Moreover, we designed and built a centralized deceptive server that can be hooked to internet-facing servers giving them the ability to return deceptive responses. These three schemes are designed, implemented, and analyzed for their security and performance.

The use of deception in security, and in computing in general, shows some fruitful results. This dissertation discusses some of the unique advantages of such mechanisms and presents a framework to show how they can be integrated into computer defenses. Also, it provides three practical schemes that employ deception in their design to address some existing security challenges. We postulate that the use of deception can effectively enhance the effectiveness of current security defenses and present novel ways to address many security challenges.

1 INTRODUCTION

1.1 Motivation and Overview

Most data is digitized and stored in organizations' servers, making them a valuable target. Advanced persistent threats (APT), corporate espionage, and other forms of attacks are continuously increasing. Companies reported 142 million unsuccessful attacks in the first half of 2013, as reported by Fortinet [158]. In addition, a recent Verizon Data Breach Investigation Report (DBIR) points out that currently deployed protection mechanisms are not adequate to address current threats [158]. The report states that 66% of the breaches took months or years to discover, rising from 56% in 2012. Furthermore, 84% of these attacks only took hours or less to infiltrate computer systems [158]. Moreover, the report states that only 5% of these breaches were detected using traditional intrusion detection systems (IDSs) while 69% were detected by external parties [158].

These numbers are only discussing attacks that were discovered. Because only 5% of the attacks are discovered using traditional security tools, it is likely that the reality is significantly worse as there are unreported and undiscovered attacks. These findings show that the status quo of organizations' security posture is not enough to address current threats.

Within computer systems, software and protocols have been written for decades with an intent of providing useful feedback to every interaction. The original design of these systems is structured to ease the process of error detection and correction by informing the user about the exact reason why an interaction failed. This behavior enhances the efforts of malfeasors by giving them information that helps them to understand why their attack was not successful, refine their attacks and tools, and then re-attack. As a result, these systems are helpful to attackers and guide them throughout their attack.

1.1.1 Thesis Statement

The use of deception to enhance the security of computer systems has occurred since at least the 1980s. However, many computer defenses that use deception were ad-hoc attempts to incorporate deceptive elements in their design. We hypothesize that

> *"It is possible to develop a framework where the act of deceit is incorporated in the design of software to give system defenders an edge in the conflict, increase the information obtained from a compromise attempt, and increase the entropy of leaked information of targeted systems during such an event. By using this framework, it is possible to augment a computer system with a set of deception techniques to enhance its security by achieving the goals of the aforementioned framework."*

To validate this hypothesis, we present a framework that can be used to plan and integrate deception in computer security defenses. In addition, we discuss three novel security defenses based on deception to enhance the security of computer systems. We show how to use deceit to enhance the security of computer systems. The main focus is to investigate methods of using deception to increase the entropy of information leaked to the adversary about our systems and the information gained by the adversary from its compromise attempts. Counter-deception, counter-attacking, legal, and ethical issues of using deception are considered out-of-scope of this work, despite their importance.

1.1.2 Dissertation Overview

The work in this dissertation discusses the unique advantages deception-based security mechanisms can bring. In section 1.3, we give a detailed discussion of the contribution of each chapter in the dissertation. During our work, we developed a framework to plan and integrate deception into computer security defenses. We discuss the different system components where deception can be applied – these components are discussed in detail in section 4.5.1. The work in this dissertation applies deception to three components as depicted in figure 1.1.

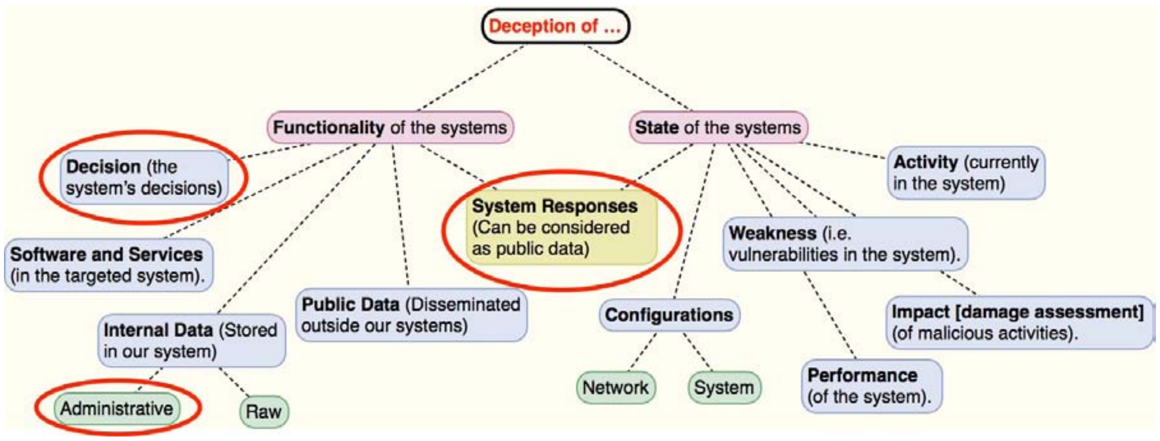

Deception of ...
Functionality of the systems
State of the systems
Decision (the system's decisions)
Software and Services (in the targeted system).
System Responses (Can be considered as public data)
Activity (currently in the system)
Weakness (i.e. vulnerabilities in the system).
Public Data (Disseminated outside our systems)
Internal Data (Stored in our system)
Configurations
Impact [damage assessment] (of malicious activities).
Administrative
Raw
Network
System
Performance (of the system).

Figure 1.1.: Computer Systems Components Where Deception is Applied in the Dissertation

To enhance the security of users' credentials and passwords, we describe applying deception to a system's *administrative internal data* and a system's *decisions* in chapter 5 and 6. As illustrated in figure 1.2, we introduce a deceptive covert communication channel in authentication protocols to enhance passwords' security and reduce their exposure. This communication channel eliminates the need to send users' password in the clear or to type them in the client. In addition, we raise stored passwords' security at the server side by implementing the *Ersatzpasswords* scheme. This scheme eliminates the possibility of cracking users' passwords, without physical access to the server's hardware, while returning fake – i.e. ersatz – passwords to adversaries when they attempt to crack stored password files. Moreover, the ersatzpassword scheme gives servers the ability to apply deception to a system's *decisions* when adversaries login using the cracked fake passwords. We discuss the details of these deceptive techniques in chapters 5 and 6.

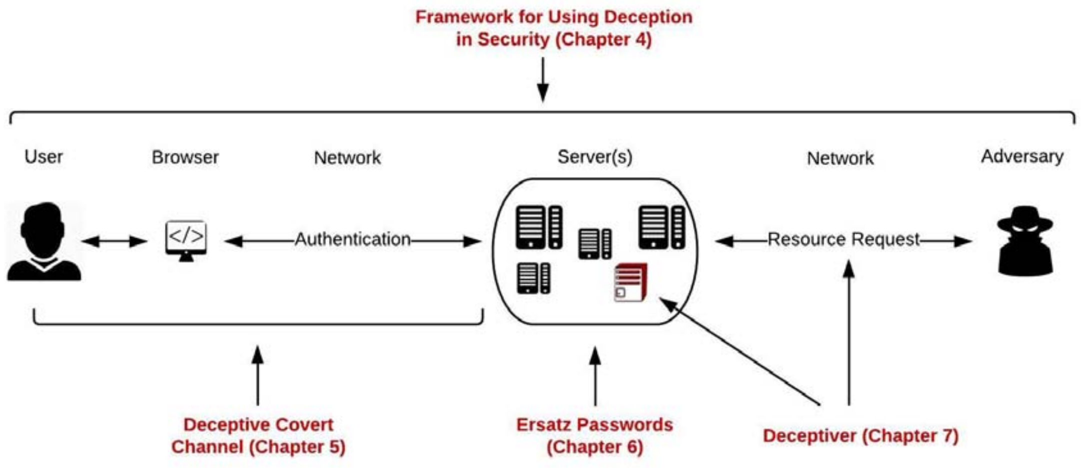

Framework for Using Deception in Security (Chapter 4)
User
Browser
Network
Server(s)
Network
Adversary
Authentication
Resource Request
Deceptive Covert Channel (Chapter 5)
Ersatz Passwords (Chapter 6)
Deceptiver (Chapter 7)

Figure 1.2.: Using Deception to Enhance Security – Dissertation Overview

Often, attackers use system responses to calibrate their attack during the reconnaissance stage. We apply deception to system *responses* to enhance the overall security of such systems. We build a deceptive server, which we refer to as *Deceptiver*, that tightly integrates with real production public-facing servers and alters their replies to deceive adversaries, as depicted in figure 1.2. Deceptiver works as a centralized server that can be hooked to public-facing servers to alter their responses by injecting deceit. The design and implementation of this deceptive server is discussed in detail in chapter 7.

1.2 Terminology

Adversary Attribution — learn some information about computer adversaries that can ultimately lead to their identification.

Covert Channel — the *covert channel* term was introduced by Lampson in 1973 and defined as *"channels not intended for information transfer at all"* [89]. The covert channel we are introducing in chapter 5 is designed to "not carry information" as perceived by the adversary.

Cyber Kill-Chain — is an intelligence-driven security model introduced by Lockheed Martin [80].

Deceptiver — a deceptive internal server that is hooked to public-facing servers to give them the ability to send deceptive responses.

Ersatzpasswords — fake passwords that are returned when an adversary tries to crack the hashed passwords file using general tools such as John the Ripper[1].

Honeyaccount — fake account in a computer system.

OODA — the OODA loop (for Observe, Orient, Decide, and Act) is a cyclic process model, proposed by John Boyd, by which an entity reacts to an event [20]. The victory in any tactical conflict requires executing this loop faster than the opponent.

1.3 Dissertation Organization and Contribution

This dissertation contains most of the ideas published in a number of papers and technical reports. Below is a description of the original work in this dissertation that was developed as part of my research.

In chapter 2, we discuss a novel taxonomy of information protection mechanisms. The original work was published in a paper that appeared at the 9th International Conference on Cyber Warfare & Security conference (ICCWS'14) [6]. We present four major categories of security controls, the objectives of each category, and investigate the inter-relationships among different categories. In addition, we examine how our taxonomy maps to different scales within organizations. Finally, we investigate how the proposed categories interplay with each other to enhance the security of computer systems. We also map these categories to the cyber kill-chain framework.

[1]`http://www.openwall.com/john/`

Chapter 3 discusses the concept of deception. We present a discussion of deception definition and maxims. In addition, we highlight the role of biases in the success of any deceptive component. We discuss the different categories of biases and give a number of examples on how they can be exploited to present a plausible alternative of the truth. We also present an overview of the use of deception in military conflicts, digital life, computing in general, and in security. We discuss how deception has been used in HCI, HRI, robotics, and other areas of computing. Additionally, we present an analysis of the previous work in using deception to raise the security of computer systems. We conclude the chapter by discussing the principles of deception operations and tactics. Part of the discussion in this chapter appeared in a paper at the New Security Paradigm Workshop NSPW'14 [5].

In chapter 4 we discuss the role of deception in security – protecting or compromising computer systems. We examine some of the unique advantages deception-based security defenses have over traditional tools. We posit how Kerchoff's principle has been misinterpreted to drive the community away from deception-based security. In addition, we present a discussion on how deception can be modeled and argue that deception is not the same as hacking back. In addition, we investigate how deceptive techniques can be planned and integrated into computer security defenses. We develop a framework that can be used to achieve such a goal along with assessing the additional risks and monitoring these controls. We discuss the different system's components where deception can be applied and analyze the methods that can be used to create a plausible deception-based security defense. Additionally, we map two previous uses of deception against our framework and show how it does capture all the elements in their design. We conclude this chapter by presenting a case study where our framework is used to enhance the security of a web application. The contribution in this chapter has been published at New Security Paradigms Workshop (NSPW'14) [5] and in the International Journal of Cyber Warfare and Terrorism (IJCWT) [7].

Chapters 5, 6, and 7 present three deception-based security mechanisms we developed. In each chapter we present a brief discussion of the security problem we are trying to solve, discuss our solution in detail, and finally present an investigation of the security and performance of each scheme. Chapter 5 shows how the use of a deceptive covert channel can enhance the security of authentication protocols. Ersatzpassword scheme in chapter 6 presents a solution to the problem of passwords storage and cracking. Finally, in chapter 7 we discuss our deceptive server (Deceptiver) that can be hooked to public-facing servers giving them the ability to respond with deceptive responses. The contribution in chapter 5 appeared at the International Conference on ICT Systems Security and Privacy Protection (IFIP SEC'15) [4].

In chapter 8 we present a summary of the contribution of this dissertation and possible directions for future research.

2 A TAXONOMY OF COMPUTER SYSTEMS' DEFENSES

Achieving security cannot be done with a single, silver-bullet solution; instead, effective security involves a collection of mechanisms that work together to balance the cost of securing our systems with the possible damage caused by security compromises, and drive the success rate of attackers to the lowest possible level. In Figure 2.1, we present a taxonomy of protection mechanisms commonly used in computer systems. The diagram shows four major categories of protection mechanisms and illustrates how they intersect achieving multiple goals.

The main motivation of this approach is to show the range of security controls that an organization can deploy, the objectives of each category, and how these mechanisms interact with each other to achieve better overall security. In this chapter, we discuss some of the interesting relationships among these categories and examine how this can be exploited to link isolated security controls. To fit all the pieces of our taxonomy together, for a holistic and practical approach to security, we map our taxonomy to the cyber kill-chain model introduced by Lockheed Martin in [80]. We develop and expand some of the stages of the cyber kill-chain model and show that we can have more effective security controls at each stage.

The rationale behind having these intersecting categories is that a single layer of security is not adequate to protect organizations, so multi-level security controls are needed [141]. This model follows a natural chronological progression of security defender goals when interacting with an attacker. First, we would like to deny unauthorized access and isolate our information systems from untrusted agents. However, if adversaries succeed in penetrating these security controls, we should have degradation and obfuscation mechanisms in place to slow their lateral movement in penetrating our internal systems. At the same time, these tools makes the extraction of information from penetrated systems more challenging.

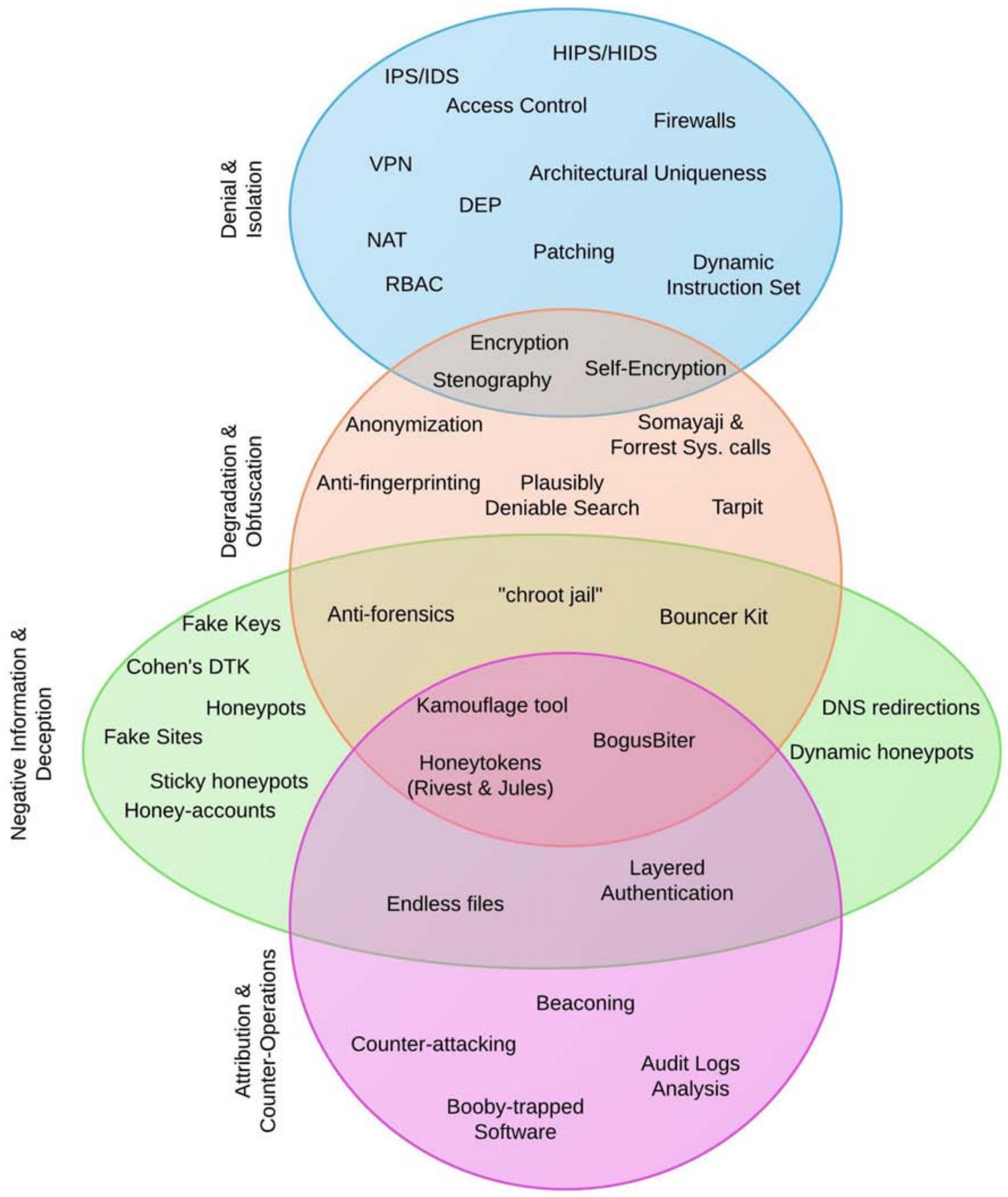


Figure 2.1.: Taxonomy of Information Protection Mechanisms

Often, even if we slow attackers down and obfuscate stored information, advanced adversaries may explore our systems undetected. This motivates the need for a third category of security controls that involves using means of deceit and negative information. These techniques are designed to lead attackers astray and augment our systems with decoys to detect stealthy adversaries. Furthermore, this deceitful information wastes attackers' time and adds risk during their infiltration. The final group of mechanisms in our taxonomy is used to gain information about the attackers and give us the ability to have counter-operations. Booby-trapped software is one example of counter-operations that can be employed.

To show how our taxonomy can be applied at different levels and granularity within computer systems, we plot these four categories across five levels of scale. We use figure 2.2 to depict this. These five levels are

- Data items; this includes files and objects.
- Databases; which are collections of data items creating larger, coherent objects.
- Systems; this refers to individual systems within our organization. For example, end-points and servers fall into this category.
- Networks; which are a network of systems connected together with communication equipment such as switches and routers.
- Enterprises; which refers to the highest level of abstraction in the digital realm. This abstraction also includes parts that deal with users and human actors.

2.1 The Four Categories of Protection Mechanisms

Securing a system is an economic activity and organizations have to strike the right balance between cost and benefits. Our taxonomy provides a holistic overview of security controls, with an understanding of the goals of each group and how they interact with each other. This empowers decision makers on what and which security controls they should deploy. In the four sections that follow we discuss each one of the four categories, illustrating their goals and providing some practical examples.

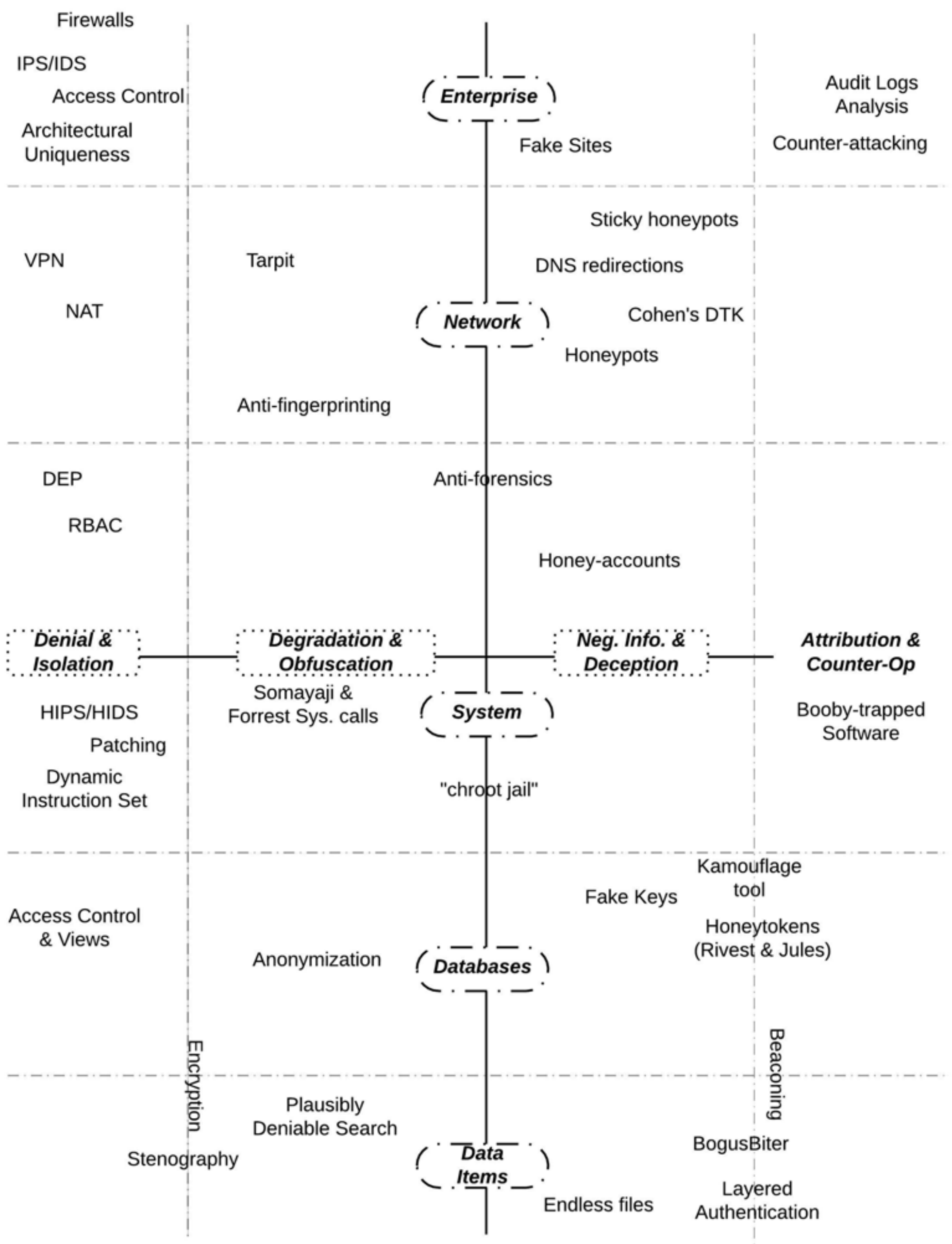


Figure 2.2.: Plotting the Taxonomy Over Multiple Scales

2.1.1 Denial and Isolation

The first, and most common, mechanism used to protect information systems is to deny all access, execution, and manipulation of our systems and data unless explicitly allowed. This gives us the ability to create a boundary around our systems isolating them from the outside. This group covers a wide variety of security controls that can be sub-grouped into three major categories: (i) controls installed around the perimeter, such as firewalls; (ii) within our internal systems, such as access control; and (iii) at the end-points, such as anti-virus and intrusion prevention.

We give several examples of mechanisms in the upper-most oval of figure 2.1. Security controls in this category are designed to achieve two main goals:

- Prevent unauthorized access to information stored in our systems.
- Hide the existence and/or the nature of our systems and/or the data stored in them.

Such mechanisms can be applied at all scales within our information systems as presented in figure 2.2. At the enterprise level, we employ security controls such as firewalls and access control systems. More advanced mechanisms such as having unique system architecture and advanced intrusion prevention systems can be used. At the network level technologies such as network address translation (NAT) and virtual private networks (VPNs) are used to isolate and hide parts of our systems denying unauthorized access to them. Denial mechanisms can also be applied at the systems level. Tools such as data execution prevention (DEP) [123] and patching security vulnerabilities are commonly used. More sophisticated mechanisms such as dynamic instruction sets can be used to obfuscate the instruction set a computer can execute and, therefore, prevent any unauthorized programs from running [172]. At the database and data item granularity level, mechanisms such as encryption can be used.

2.1.2 Degradation and Obfuscation

When adversaries overcome the first line of defense, we have three general classes of objectives: detect them, slow them down, and disguise and/or hide our data. Many security mechanisms are used to address these issues. Security controls in this category are designed to achieve the following goals:

- Slow down the attackers.
- Prevent and significantly reduce the probability that an adversary can recover sensitive data.
- Obfuscate the value/nature of our systems and/or the data stored in them.
- Create noise around valuable information to reduce its utility.

At the data item level, mechanisms such as k-anonymity [148] and plausibly deniable search [108] have been used to degrade the information obtained – directly and indirectly – from users' data. At the systems' level, slowing down the response of system calls when detecting anomalies has been proposed to degrade adversaries' infiltration speed [77]. At the network level, tarpits are used to throttle the spread of malware and spam within organizations [53]. We note that there is a shortage of these techniques to employ at the top level in our hierarchy – the enterprise level.

2.1.3 Negative Information and Deception

Despite all the controls organizations have in place, attackers might infiltrate information systems and operate without being detected or slowed. In addition, persistent adversaries might infiltrate the system and passively observe for a while to avoid being detected and/or slowed when moving on to their targets. As a result, the next layer of defense is needed to augment our systems with negative and deceptive information to lead attackers astray. We may also significantly enhance organizational intrusion detection capabilities by deploying deception-based detection methods.

Negative information alters the way computer systems are perceived, which includes the use of deception [166]. However, deception alters such perception in a way that is advantageous to system defenders. Deceptive techniques are an integral part of human behavior. As an example, deception is widely used in sports; teams attempt to deceive the other team into believing they are following a particular plan so as to influence their course of action. Use of cosmetics may also be viewed as a form of mild deception. We use lies in conversation to hide mild lapses in etiquette. In cyber security, deception and decoy-based mechanisms have been used in security for more than two decades in technologies such as honeypots and honeytokens. We present a survey of the use of deception in computing and in security in sections 3.5, 3.6, and 3.7.

Deception-based techniques are increasingly gaining interest within the information security community [5, 7, 66, 74]. Security controls in this category are designed to achieve four main goals.

- Lead the attackers astray and waste their time and resources, giving defenders an edge in the OODA loop [20].
- Add decoys to our system to detect data leakage and intrusions, enhancing the understanding of the attackers' goals and tools.
- Add doubt to the data obtained by the adversary.
- Increase the risk of attacking our computer systems.

We discuss the advantages of using deception-based security defenses further in section 4.3.1.

When attackers infiltrate the system and successfully overcome traditional detection and degradation mechanisms we would like to have the ability to not only obfuscate our data, but also lead the attackers astray by deceiving them and drawing their attention to other pieces of data that are false or intentionally misleading. Furthermore, exhausting attackers and causing frustration is also a successful defensive outcome. Planting fake keys and using schemes such as endless files [142] can achieve this. These files look small on the organization servers but when downloaded to be exfiltrated will exhaust the adversaries' bandwidth and raise some alarms. We provide a survey of previously used deception-based defenses in section 3.7. With carefully designed deceiving information we can even cause damage at the adversaries' servers. A traditional, successful, deception technique can be learned from the well-known story of the Farewell Dossier during the cold war where the CIA provided modified hardware and software designs to a Soviet spy ring [169]. When the Soviets used these products thinking they were legitimate, it resulted in a major disaster affecting a trans-Siberian pipeline.

A relationship can be observed between the first category and deception, especially in the concept of hiding. By definition, both denial and deception can involve hiding things from adversaries. However, in this dissertation we consider the purpose of hiding as an important distinguisher. If we hide things to alter the way adversaries perceive targeted systems, this is considered deception, otherwise, we refer to it as denial.

Another relationship can be observed between the last group of protection techniques, namely attribution, and deception techniques. Deception-based mechanisms are an effective way to lure attackers to expose themselves and their objectives when we detect them accessing things and conducting unusual activities. Other tools, such as anomaly-based IDS have similar goals, but the advantage deception-based tools have is that there is a clear line between normal user activities and abnormal ones. This is because legitimate users are not supposed to access implanted fake information. This difference significantly enhances the effectiveness of deception-based security controls and reduces the number of false-positives, as well as the size of the system's log file.

2.1.4 Attribution and Counter-Operations

Sun Tzu, the Chinese military strategist, once wrote;

> *"if you know your enemies and know yourself, you will not be imperiled in a hundred battles; if you do not know your enemies but do know yourself, you will win one and lose one" [156].*

This brilliantly summarizes the current security state of many organizations around the world. We need to know the attackers, attribute them and understand their objectives. Security controls in this last category are designed to achieve three main goals:

- Attribute the adversaries.
- Cause damage to attackers.
- Increase overall risk in attacking our systems.

One of the traditional ways of learning about adversaries is analyzing the logs generated by our systems. However, one of the main challenges that has been hindering the adaptation of such mechanisms – intended for attributing adversaries – is mixing those mechanisms with counter-attacking and "hacking back". This misconception is discussed further in section 4.6.2. We argue that attribution can be achieved using a wide variety of mechanisms without having to address the ethical and political issues surrounding counter-attacking.

We argue that intelligently planting deceptive information within our information systems can help us both in attributing some adversaries and detecting leakage. Steganographic watermarking data can also serve as a means of detecting leakage and possibly providing attribution of sources. In chapter 6, we present a scheme of integrating deceptive information in password files to enhance their security and detect their leakage.

2.2 Fitting the Pieces Together – Cyber Kill-Chain Model

Employing techniques from all four categories provides a more effective approach than only using one or two. Additionally, security is an economic activity and distributing budget at multiple layers may provide a better return on investment than more focused spending.

The cyber kill-chain introduced by Lockheed Martin researchers advocates for an intelligence-driven security model [80]. The main premise behind this model is that for attackers to be successful they need to go through all these steps in the chain in sequence. Breaking the chain at any step will break the attack and the earlier that we break it the better we prevent the attackers from attacking our systems.

The deployment of the cyber kill-chain was seen as fruitful for Lockheed when they were able to detect an intruder who successfully logged into their system using the SecurID vulnerability [76]. To show how all the protection categories discussed above can fit together in protecting organizations we map them against the cyber kill-chain model summarized in table 2.1.

2.2.1 The Role of Deception

The consensus is that we would like to be at least one step ahead of adversaries when they attack our systems. We argue that by intelligently incorporating deceit in our security models we can start achieving that. This is because the further we enhance our abilities to detect adversaries the further ahead of them we position ourselves. We discuss the advantages of deception-based security mechanisms in further detail in section 4.3.1.

If we take an example of external network probing, if we simply detect an attack and identify a set of IP addresses and domain names as “bad,” we do not achieve much; these can be easily changed and adversaries will become more careful not to raise an alarm the next time they probe our systems. However, if we go one more step and attribute them by characteristics that are more difficult to change we can

Table 2.1: Mapping Security Mechanisms to the Kill-Chain Model

	Denial & Isolation	***Degradation & Obfuscation***	***Deception & Negative Information***	***Attribution & Counter Operations***
Reconnaissance	Firewalls, Architectural Uniqueness, NAT	Anti-fingerprinting	Artificial ports, Fake Sites	Audit Logs Analysis
Weaponization & Delivery	In-line Filters, IPS, Tarpit, IDS		Artificial call back, Sticky Honeypots	
Exploitation & Installation	Dynamic Instruction Set, Adaptive sys. calls, HIPS, Patching, DEP, “chroot jail”		Create artificial exploitation response	
Command & Control (operation)			Honeypot	
Lateral Movement & Persistence	VPN, Access Control, RBAC	Encryption, Self-Encryption	HoneyAccounts, HoneyFiles	
Staging & Exfiltration		Stenography	Honeytokens, Endless files, Fake Keys	Beaconing, Counter-Attacking, Booby Trapped Software

cause them greater difficulty for their future attacks. For example, if we are able to deceive attackers in manners that allow us to gather more information about them – distinguishing them based on their fixed artifacts (such as distinctive protocol headers, known tools, and/or behavior and traits) – we have a better position for defense. The design of the deceptive server "Deceptiver" in chapter 7 is structured to achieve such goals.

The cyber kill-chain model is a good framework to demonstrate the effectiveness of incorporating deception at multiple levels in the chain. With the same underlying principle of the kill-chain – early detection of adversaries – we argue that the earlier we detect adversaries, the better we are at deceiving them and learning more about their methods and techniques. We postulate that full intelligence cannot be gathered without using some means of deception techniques.

As Sun Tzu noted, the better we know our enemies the better we can defend against them. By using means of deception we can continuously learn about attackers at different phases of the kill-chain and enhance our capabilities of detecting them and reducing their abilities to attack us. This negative correlation is an interesting relationship between our ability to detect attackers and their ability to probe our resources.

2.3 Chapter Summary

In this chapter we discussed how different categories of information protection relate and interact. This taxonomy gives us a holistic view of how to protect computer systems. We discussed the four categories in our taxonomy, their goals and gave a number of examples from currently deployed tools. We analyzed how the intelligence-driven security model – cyber kill-chain – can be used to plan and deploy security tools. We concluded the chapter by highlighting the role of deception and negative information in enhancing the security of computer systems. This discussion is a preface to the next two chapters that investigate the concept of deception and how it can be used to enhance the security of computer systems.

3 DECEPTION

Deception has been in use for many millennia, perhaps for nearly as long as life has existed on planet Earth. Plants, animals, and insects have been using deceptive techniques as a means for defense and survival. Humans are no exception to the use of deception. Illusionists use it to entertain us, con artists to cheat us, and military strategists to attack and defend us. Digital realms are no different from the "real world" as deception has found its way into computerized systems. In this chapter, we give an overview of the concept of deception and some of the major areas where deception has been used. We primarily focus on areas where deception has been used in conflicts between different parties.

As human beings, we are not good at detecting deception. In 39 different studies by Vrij, he found that the mean accuracy rate for college students to detect deception was 57%, which is almost as poor as random choice (i.e., 50%) [162]. This rate is slightly worse with law enforcement officers, who scored a mean accuracy rate of 54% [162]. Whaley clearly states in his seminal book *"Stratagem: Deception and Surprise in War,"* which is the largest open source empirical analysis of the use of deception in conflicts, that "indeed, this is a general finding of my study – that is, the deceiver is almost always successful regardless of the sophistication of his victim in the same art." [171].

3.1 General Definition of Deception

A misperception that is "*intentionally* induced by actions of *other* entities" is a deception [170]. It is important to note that deception is targeted at altering perceptions to gain an advantage as illustrated in figure 3.1 – adapted from [170].

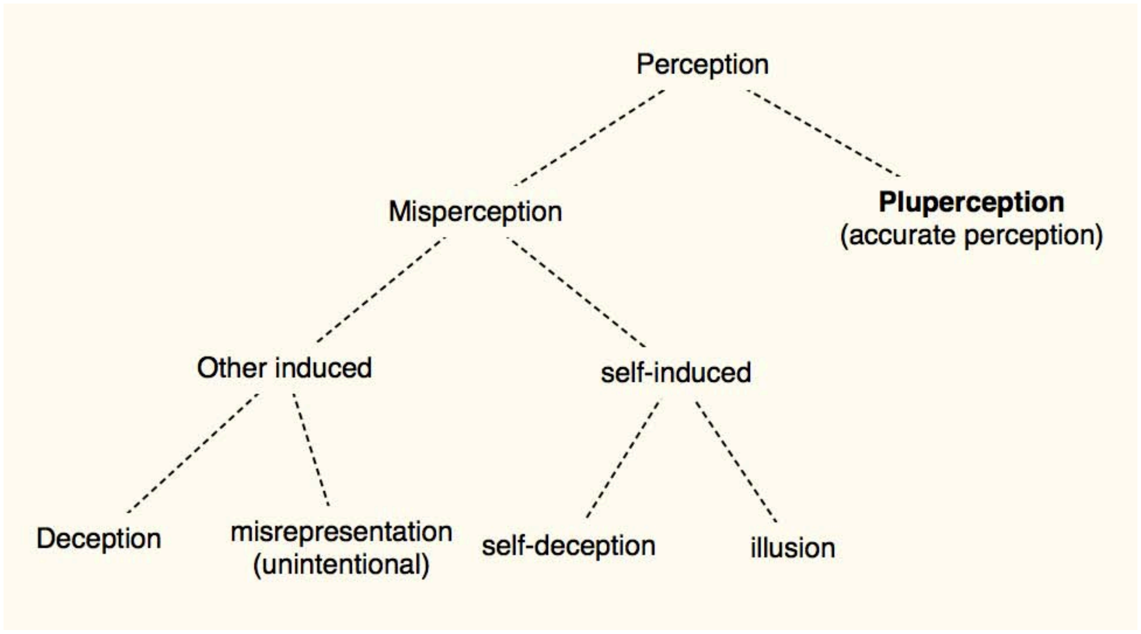


Figure 3.1.: Deception and Perception

3.2 Deception and the Truth – A Taxonomy

3.2.1 Simulation and Dissimulation – Bell and Whaley

Bell and Whaley argue that deception always involves two steps: *dissimulation*, hiding the real, and *simulation*, showing the false [11]. Deception must involve these two together, even if only implicitly. The act of hiding and showing can applied to the (i) nature, (ii) existence and/or (iii) the value of targeted information. The authors also offered a taxonomy of deceptive techniques where they distinguished among three ways of dissimulating — masking, repackaging, and dazzling — and three ways of simulating — mimicking, inventing, and decoying. A brief discussion of each one of those is given below. Additionally, later in section 4.5.1 we discuss how to create deceptive security techniques using this taxonomy.

Dunnigan and Nofi propose another taxonomy in [51]. Their taxonomy has the following groups: concealment, camouflage, false and planted information, lies, displays, ruses, demonstrations, feints, and insights. We found that each one of these categories either has a direct mapping to one of Bell and Whaley's categories or is an example of one of them.

Masking

The obvious act of any deceptive technique is to hide the real by masking it such that it can remain undetected. A traditional example can be seen in behavior of the *chameleon* where it changes its color to blend with the background deceiving any predator and masking itself as if it does not exist. As we discussed earlier in section 2.1.3, hiding is only considered deception if it is an act of deceit, otherwise it is considered denial.

Repackaging

Fully masking something as if it does not exist can be challenging. In some cases, it might be much easier to "repackage" that thing as something else. The repackaging can go both ways by making something dangerous appear as harmless or vice versa. Moreover, repackaging can make something important look totally irrelevant, thus driving attention away from it. A traditional example of this can be seen in the behavior of the "mantis" insects where they repackage themselves as sticks to avoid bird predators.

Dazzling

This is considered to be the weakest form of dissimulation, where we simply confuse the targeted objects with others. When an object cannot be masked or repackaged, we use dazzling to dissimulate it. One prime example of using this in technology is injecting traffic to reduce the exposure of traffic analysis attacks [60].

Mimicking

When we hide the real we necessarily show the false, even if only implicitly by showing “nothing.” The first method of simulation techniques is to show the false, by mimicking something true, to gain an advantage. As an example, when the mantis insects hide, by mimicking a stick, they are also luring prey close enough to be seized.

Inventing

Mimicking requires the item to look like something else, however, when this is not so easy to achieve, invention can be used instead. When inventing we create a new reality instead of mimicking the existence of another one. Rubber tanks are one prime example of inventing a reality [170].

Decoying

Decoying is one of the most commonly used simulation techniques to deceive. In decoying, the deceiver simply tells a common truth but then resort to something different, and often less predictable. This is common in many sports where a team might formulate their position to give the impression that they are defending, but then they play an offensive play.

3.2.2 Linguistic Case Theory

Rowe used linguistic case theory to provide a taxonomy for the use of deception in cyber space [129]. He argues that every deceptive action can be categorized by an “associated semantic and case(s).” Adopting the list of semantic cases by Copeck et al. [38] with additional relationships from AI, Rowe suggested a taxonomy of deception consisting of 32 semantic cases grouped in seven categories: spatial cases, time cases, participant cases, causality cases, quality cases, essence cases, and speech-act cases.

This taxonomy can be useful when brainstorming possible deceptive techniques to be used in defending computer systems.

3.3 Deception Maxims

Bennett and Waltz discussed four deception maxims that are core to any investigation of the user of deception; namely *truth*, *denial*, *deceit*, and *misdirection* [12]. In this section we discuss the relationships among these principles adding a fifth one that is equally as important; namely *confusion*.

3.3.1 Truth/Reality

Truth is the accurate perception of everything about the observed. Deception is an active act directed at manipulating such perception. For deception to succeed, there must be an *accurate* perception that we are trying to manipulate [12]. Truth should constitute most of the information that is perceived by an adversary. Mitchell and Thompson highlight this principle by stating that *"all deception works within the context of honesty"* [98]. Handel provides four rules of what truth should be presented to the target [71]:

1. The deceiver should supply the target with correct low-grade information; i.e. "chicken-feed."

2. Correct information that is already known by the opponent should always be presented to the target.

3. The deceiver should often pass correct information to the target when he can control its arrival time to be after it is of any use.

4. The deceiver might need to sacrifice some important information such that he can lure the target into believing some deceit that would have not been believed otherwise.

Handel summarizes his discussion with this quote *"The more one has a reputation of honesty – the easier it is to lie convincingly. Even more concisely, honest people/states can deceive the best"* [71].

3.3.2 Deceit

"All deception requires deceit" as said by Bennett and Waltz [12]. In other words, all deception requires the deceiver to intentionally lie about something to the target. Everyone lies in their daily lives. Ford cites some studies showing that 90% of Americans admitted that they lie about their feelings, income, sex lives, accomplishments, life, and age [57].

There is a fundamental difference between simple lies and deception. The former focuses on only one side of the communicated message; namely the liar [43]. The latter adds to that the other side of the message, namely the receiver, and how this lie affects his perception and/or actions [43].

3.3.3 Denial, Misdirection and Confusion

There are three general way to manipulate a target's perception of truth and deceit with respect to deception. We can *deny* the target access to the truth and show him the deceit instead. When we cannot stop the truth from being observed we can *misdirect* the target's focus to the deceit. When we cannot influence the target's focus, we can *confuse* the target by presenting him with the truth and one or more plausible deceits.

3.4 Deception and Biases

In cognitive psychology a bias refers to

> "An inclination to judge others or interpret situations based on a personal and oftentimes unreasonable point of view" [12]

Biases are a cornerstone component to the success of any deception-based mechanisms. The target of the deception needs to be presented with a plausible “deceit” to successfully deceive and/or confuse him. If the target perceives this deceit to be non-plausible she is more inclined to reject it instead of believing it, or at least raise her suspicions about the possibility of currently being deceived. A successful deception should *exploit* a bias in the attackers’ perception and provide them with one or more plausible alternative information other than the truth.

Thompson et al. discuss four major groups of biases any analysts need to be aware of: personal biases, cultural biases, organizational biases, and cognitive biases [151]. It can be seen in figure 3.2 that the more specific the bias being exploited in a deceptive security tool is, the less such a tool can be generalized, For example, exploiting a number of personal biases, specific to an attacker, might not be easily generalized to other adversaries who attack your system. However, the more specific the choice of bias enhances the effectiveness of the deceptive component. This is true partly because cognitive biases are well-known and adversaries might intentionally guard themselves with an additional layer of explicit reasoning to minimize their effects in manipulating their perceptions. In the following paragraphs we discuss each one of these classes of biases.

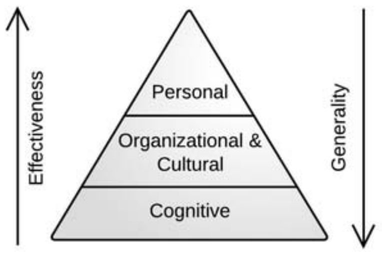


Figure 3.2.: Deception Target Biases

3.4.1 Personal Biases

Personal biases are those biases that originate from either first-hand experiences or personal traits, as discussed by Jervis in [82]. These biases can be helpful in designing deceptive components/operation; however, they are (i) harder to obtain as they require specific knowledge of potential attackers and (ii) they make deceptive components less applicable to a wider range of attackers while becoming more powerful against specific attackers. Personal biases have been exploited in traditional deception operations in war, such as exploiting the arrogance of Hitler's administration in World War II as part of Operation Fortitude [12].

3.4.2 Cultural Biases

Hofstede refers to cultural biases as the *"software of the mind"* [78]. They represent the mental and cognitive ways of thinking, perception, and action by humans belonging to these cultures. In a study conducted by Guss and Dorner, they found that cultures influenced the subjects' perception, strategy development and decision choices, even though all those subjects were presented with the same data [68]. Hofstede discusses six main dimensions of cultures and assigns quantitative values to those dimensions for each culture in his website (geerte-hofstede.com). Also, he associates different behavior that correlates with his measurements. Theses dimensions are:

1. **Power Distance Index (PDI)** — PDI is a measure of the expectation and acceptance that "power is distributed unequally." Hofstede found that cultures with high PDI tend to have a sense of loyalty, show of strength, and preference to in-group-person. This feature can be exploited by a deception planner focusing on the attacker's sense of pride to reveal himself, knowing that the attack is originating from a high PDI culture with a show-of-strength property.

2. **Individualism versus Collectivism (IVC)** — A collectivist society values the "betterment of a group" at the expense of the individual. Hofstede found that most cultures are collectivist, i.e. with low IVC index.

3. **Masculine versus Feminine (MVF)** — A masculine culture is a culture where "emotional gender roles are clearly distinct." For example, an attacker coming from a masculine culture is more likely to discredit information and warnings written by or addressed to a female. In this case, this bias can be exploited to influence attackers' behaviors.

4. **Uncertainty Avoidance Cultures (UAI)** — This measures the cultural response to the unknown or the unexpected. High UAI means that this culture has a fairly structured response to uncertainty making the attackers' anticipation of deception and confusion a much easier task.

5. **Long-Term Orientation versus Short-Term Orientation (LTO vs. STO)** — STO cultures usually seek immediate gratification. For example, the defender may sacrifice information of lesser importance to deceive an attacker into thinking that such information is of importance, in support of an over-arching goal of protecting the most important information.

6. **Indulgence versus Restraint (IVR)** — This dimension characterizes cultures on their norms of how they choose activities for leisure time and happiness.

Wirtz and Godson summarize the importance of accounting for cultures while designing deception in the following quote; *"To be successful the deceiver must recognize the target's perceptual context to know what (false) pictures of the world will appear plausible"* [65].

3.4.3 Organizational Biases

Organizational biases are of importance when designing deception for an target within a heavily structured environment [12]. In such organizations there are many keepers who have the job of analyzing information and deciding what is to be passed to higher levels of analysts. This is one example of how organizational biases can be used. These biases can be exploited causing important information to be marked as less important while causing deceit to be passed to higher levels. One example of organizational biases is uneven distribution of information led to uneven perception and failure to anticipate the Pearl Harbor attack in 1941 by the United States [12].

3.4.4 Cognitive Biases

Cognitive biases are common among all humans across all cultures, personalities, and organizations. They represent the "innate ways human beings perceive, recall, and process information" [12]. These biases have long been studied by many researchers around the world in many disciplines (particularly in cognitive psychology); they are of importance to deception design as well as computing.

Tversky and Kahneman proposed three general heuristics our minds seem to use to reduce a complex task to a simpler judgment decision – especially under conditions of uncertainty – thus leading to some predictable biases [153]. These are: representativesness, availability, and anchoring and adjustment. They defined the representativeness heuristic as a *"heuristic to evaluate the probability of an event by the degree to which it is (i) similar in essential properties to its parent population; and (ii) reflects the salient features of the process by which it is generated"* [153]. The availability heuristic is another bias that assess the likelihood of an uncertain event by the ***ease*** with which someone can bring it to mind. Finally, the anchoring/adjustment heuristic is a bias that causes us to make estimations closer to the initial values we have been provided with than is otherwise warranted.

Solman presented a discussion of two reasoning systems postulated to be common in humans: associative (system 1) and rule-based (system 2) [140]. System 1 is usually automatic and heuristic-based, and is usually governed by habits. System 2 is usually more logical with rules and principles. Both systems are theorized to work simultaneously in the human brain; deception targets System 1 to achieve more desirable reactions.

In 1994, Tversky and Koehler argued that people do not subjectively attach probability judgments to events; instead they attach probabilities to the description of these events [154]. That is, two different descriptions of the same event often lead people to assign different probabilities to their likelihood. Moreover, the authors postulate that the more *explicit* and detailed the description of the event is, the higher the probability people assign to it. In addition, they found that unpacking the description of the event into several disjoint components increases the probability people attach to it. Their work provides an explanation for the errors often found in probability assessments associated with the *"conjunction fallacy"* [155]. Tversky and Kahneman found that people usually would give a higher probability to the conjunction of two events, e.g. P(X and Y), than a single event, e.g. P(X) or P(Y). They showed that humans are usually more inclined to *believe* a detailed story with explicit details over a short compact one.

3.5 The Use of Deception in War, Military and Conflicts

Deception has long been used as a prime tool within the intelligence community and historically in war. The Greek's Trojan horse illustrates the age of such techniques. The Chinese military strategist Sun Tzu states that "All warfare is based on deception" [156]. The Joint Publication (JP) 3-13.4 defines military deception as [25];

> *"Actions executed to deliberately mislead adversary military decision makers as to friendly military capabilities, intentions, and operations, thereby*

> *causing the adversary to take specific actions (or inactions) that will contribute to the accomplishment of the friendly mission."*

Earlier in this chapter, we discussed many example of deceptive techniques used by living creatures and existing in nature. Gerwehr and Glenn give a detailed discussion of the use of deception in military applications [64]. Latimer, in his book *"Deception in War"* [90], provides an extensive discussion of the use of deception in war and military conflicts. Additionally, Brown discusses the deception operation in the D-Day invasion in his book *"Bodyguard of Lies"* [22].

3.6 General Use of Deception in Computing

3.6.1 In Human-to-Human Digital Interaction

Jeff Hancock studies the act of deception by normal users in the digital age [70]. His studies focus on users' behavior investigating why and how they lie in the digital world. He developed a number of algorithms to distinguish between fake and true user's generated content. In addition, he examined people's online behaviors and the fake information they post about themselves and others.

Galanxhi and Nah studied the behavior of deceivers and truth-tellers in cyberspace [61]. They investigated whether the use of avatars influences one's perception of the truthfulness of the other communicating partner. Their research mainly focuses on the communication aspects of such behavior. They primarily investigate how deception happens in such environments and explore the features that enable such behavior.

3.6.2 In Human Computer Interaction (HCI)

Most of the research on the use of deception in HCI focuses on the user of malevolent deception, often referred to as *dark patterns*[1] [1,37]. This goes inline with many design guidelines that asserts that a good design *should not lie* to users [134]. In

[1] `http://darkpatterns.org/`

addition, Conti and Sobiesk characterized user interfaces that *trick* – mislead or deceive the user – as malicious [37]. They discuss a number of examples of how such techniques can be used to lie to users and spoof their content. They examined the affect of using such techniques on increasing user frustration.

Nevertheless, Adar et al. make a distinction between malevolent and benevolent deception [1]. They argue that the latter often is a helpful technique in improving users' experiences in HCI. The authors also discussed how regular users employ deception to avoid unwanted interruptions. For example, using tools that auto-respond to your contacts and make you appear online at random times are deceptive techniques. Moreover, Adar and his group contend that the use of deception in HCI often helps users rather than harm them [1]. They examined the use of deception in HCI arguing that such techniques are often used to: (i) create users' delight (e.g. by providing the user control over the system or hinting to the existence of some features), or (ii) mask computer failures. Often such techniques help system designers to direct users into acting in predictable ways. They argue that the common gap between the user's desire from a computer system and the reality of such system motivates and enables the use of deception to cover it. They divide this gap into the following four categories.

- A gap between the user's metal model and the underlying system's model.

  This is one of the most common gaps in HCI where deceptive techniques are used to bridge the gap between users' and systems' models. Such a gap can occur because of performance and failure issues, to hide uncertainties, to guarantee a certain level of pleasure and entertainment, or to increase the level of comfort and credibility [1]. An example can be be seen in the early phone call routing systems, 1ESS. In such systems, when a failure occurred in connecting two users the system connected the caller to a random number instead of dropping the call. This technique was used to deceive the user into believing that they misdialed the number instead of experiencing a system failure [1]. A more recent example is deployed by Netflix in their recommendation system. Their system will use the general “popular movies” recommender engine when their personalized recommender fails [31]. The user is not aware of this switch and would continue to interact with the system as if the viewed recommendations are based on her personal preferences

- Where the needs of an individual must be balanced with the needs of a group.

  An example of such interface design can been seen in password failed authentication responses. Clearly, it is more useful for the user to tell her exactly if the typo was in the username or the password, instead of asking her to type them both again. However, the security of the whole system is raised by not explicitly specifying which part of the credentials is wrong. In this case, it is a recommended security practice to “lie” to users and tell them to type both credentials again raising the cost of brute-forcing other accounts’ passwords.

- When a person must be protected from oneself.

  When a user deletes a file or drags it to the trash, the file is not immediately deleted. In the physical world, the town of Dusseldorf has a fake bus stop set next to a senior care center to catch Alzheimer patients who sneak out of the center [120]. They wait at the bus stop instead of wandering around and getting lost.

- When trying to meet conflicting design goals.

It is important to note that there is a fundamental difference between deception and abstraction. Often, the line between the two is fuzzy. We discuss this further, in section 4.6.4, where we make a clear distinction between deception and abstraction.

3.6.3 In Robotics and Human Robot Interaction (HRI)

The application of deception in robotics has been used to improve the user’s experiences or add additional features [102, 139, 157, 163]. Such behavior adds value to those machines, for example, calming patients or helping them to overcome their self-imposed limits. In addition, deploying deceptive behavior in robots bring a set of advantages to their use in the military domain.

Camouflage and motion camouflage are widely used deceptive techniques that have found their way into robotics. Researchers at Harvard university developed a “soft” robot that is capable of changing the color of its body to match the surrounding environment [102]. Motion camouflage, which is used by dragonflies, is a deceptive behavior where the creature follows an indirect trajectory to appear stationary while approaching its target. Rano discusses the use of such techniques in robots for stealth approaching [121].

Wagner and Arkin used interdependence theory [150] – which is a psychological theory stating that interacting parties adjust their behavior in response to their perception of social situations of reward and costs – to develop algorithms to be used by robots to decide when and how to deceive [163]. Shim and Arkin adopted the deceptive behavior used by squirrels in robots for resource allocation [139].

Within HRI deception has been used to instrument robots’ behavior to enhance users’ experiences. In a study by Vazquez et al., they showed an increase in engagement and enjoyment in a multi-player robotic game in the presence of a deceptive robot referee [157]. Brewer et al. used deception in physical therapy robotic systems [21]. They presented rehabilitating patients with deceptive visual feedback on the amount of force they are currently exerting. By making patients perceive a force level lower than what they are really exerting, they will add additional force exceeding their self-imposed mental limits.

3.6.4 In Computer-to-Computer Interaction

DeRosis et al. provide an extensive examination of such techniques in [45]. The authors challenge the *“sincerity principle”* and discuss a number of scenarios where computers should deliberately “lie.” In situations such as bargaining and personal assistance, software agents may "lie" in the short-term for optimal longer-term goals. Christian and Young discussed in their work how agents can strategically "lie" to achieve optimal goals [30].

3.7 The Use of Deception to Enhance Security

Throughout history, deception has evolved to find its natural place in our societies and eventually our technical systems. Deception and decoy-based mechanisms have been used in security for more than two decades in mechanisms such as honeypots and honeytokens. An early example of how deception was used to attribute and study attackers can be seen in the work of Cheswick in his well-known paper "An Evening with Berferd" [28]. He discusses how he interacted with an attacker in real time providing him with fabricated responses. Two of the earliest documented uses of deceptive techniques for computer security are in the work of Cliff Stoll in his book *"The Cuckoo's Egg"* [146] and the work of Spafford in his own lab [142]. The Deception Toolkit (DTK)[2], developed by Fred Cohen 1997 was one of the first publicly available tools to use deception for the purpose of computer defenses.

In late 1990s, "honeypots" – "a component that provides its value by being attacked by an adversary" i.e. deceiving the attacker to interact with them – have been used in computer security. In 2003, Spitzner published his book on *"Honeypots"* discussing how they can be used to enhance computer defenses [143]. Following on the idea of honeypots, a proliferation of "honey-*" prefixed tools have been proposed. We discuss the *honey* technologies in detail later in this section. With the release of Tripwire, Kim and Spafford suggested the use of planted files that should not be accessed by normal users, with interesting names or locations and serving as bait that will trigger an alarm if they are accessed by intruders [85].

Offensively, many current, common attacks use deceptive techniques as a cornerstone of their success. For example, phishing attacks often use two-level deceptive

[2] http://www.all.net/dtk/

techniques; they deceive users into clicking on links that appear to be coming from legitimate sources, which take them to the second level of deception where they will be presented with legitimate-looking websites luring them to give their credentials. The "Nigerian 419" scams are another example of how users are deceived into providing sensitive information with the hope of receiving a fortune later.

In many of these cases, attackers focus on deceiving users as they are usually the most vulnerable component. Kevin Mitnick showed a number of examples in his book, *"The Art of Deception"* [99], of how he used social engineering, i.e., deceptive skills to gain access to many computer systems. Trojan horses, which are more than 30 years old, are a prime example of how deception has been used to infiltrate systems.

Phishing, Cross-site Scripting (XSS) [161], and Cross-site Request Forgery (XSRF) [10] are some examples of using deception. Despite more than a decade of research by both the academic and private sectors, these problems are causing more damage every year. XSS and XSRF have remained on the OWASP's top 10 list since the first time they were added in 2007 [112]. The effectiveness of offensive deception techniques should motivate security researchers to think of positive applications for deception in security defenses.

3.7.1 Honeypots

Honeypots have been used in multiple security applications such as detecting and stopping spam[3] and analyzing malware [42]. In addition, honeypots have been used to secure databases [56]. They are starting to find their way into mobile environments [106] where some interesting results have been reported [164].

Honeypots in the literature come in two different types: server honeypot and client honeypot. The server honeypot is a computer system that contains no valuable information and is designed to appear vulnerable for the goal of enticing attackers to access them. Client honeypots are more active. These are vulnerable user agents

[3] http://www.projecthoneypot.org

that troll many servers actively trying to get compromised [137]. When such incidents happen, the client honeypots report the servers that are infecting users' clients. Honeypots have been used in computing in four main areas as we discuss in the following paragraphs.

Detection. Honeypots provide an additional advantage over traditional detection mechanisms such as *Intrusion Detection Systems (IDS)* and anomaly detection. First, they generate less logging data as they are not intended to be used as part of normal operations and thus any interaction with them is illicit. Second, the rate of false positive is low as no one should interact with them for normal operations. Angnostakis et al. propose an advanced honeypot-based detection architecture in the use of *shadow honeypots* [9]. In their scheme they position *Anomaly Detection Sensors (ADSs)* in front of the real system where a decision is made as whether to send the request to a *shadow* machine or to the normal machine. The scheme attempts to integrate honeypots with real systems by seamlessly diverting suspicious traffic to the shadow system for further investigation. Finally, honeypots are also helpful in detecting industry-wide attacks and outbreaks, e.g. the case of the Slammer worm as discussed in [100].

Prevention. Honeypots are used in prevention where they assist in slowing down the attackers and/or deterring them. *Sticky honeypots* are one example of machines that utilize unused IP address space and interact with attackers probing the network to slow them down [96]. In addition, Cohen argues that using his Deception ToolKit (DTK) we can deter attackers by confusing them and introducing risk on their side [33]. However, we are not aware of any studies that investigated those claims.

Beyond the notion of enticement and traps used in honeypots, deception has been studied from other perspectives. For example, Rowe et al. present a novel way of using honeypots for deterrence [131]. They protect systems by making them look like a honeypot and therefore deter attackers from accessing them. Their observation stemmed from the developments of anti-honeypots techniques that employ advanced methods to detect if the current system is a honeypot [79].

Response. One of the advantages of using honeypots is that they are totally independent systems that can be disconnected and analyzed after a successful attack on them without hindering the functionality of the production systems. This simplifies the task of forensic analysts as they can preserve the *attacked* state of the system and extensively analyze what went wrong.

Research. Honeypots are heavily used in analyzing and researching new families of malware. The honeynet project[4] is an *"international non-profit security research organization, dedicated to investigating the latest attacks and developing open source security tools to improve Internet security."* For example, the HoneyComb system uses honeypots to create unique attack signatures [88]. Other more specific tools such as *dionaea*[5] are designed to capture a copy of computer malware for further study. Furthermore, honeypots help in inferring and understanding some widespread attacks such as Distributed Denial of Service (DDoS) [101].

3.7.2 Honey–* Tools

The prefix "honey-*" has been used to refer to a wide range of techniques that incorporate the act of deceit in them. The basic idea behind the use of the prefix word "honey" in these techniques is that they need to entice attackers to interact with them, i.e. fall for the bait — the "honey." When such an interaction occurs the value of these methods is realized.

The term honeytokens has been proposed by Spitzner [144] to refer to honeypots but at a smaller granularity. Stoll used a number of files with enticing names and distributed them in the targeted computer systems, acting as a beaconing mechanism

[4]www.honeynet.org
[5]http://dionaea.carnivore.it/

when they are accessed, to track down Markus Hess [146]. Yuill et al. coined the term *honeyfiles* to refer to these files [176]. HoneyGen was also used to refer to tools that are used to generate honeytokens [14].

3.7.3 Incorporating Deception into Other Security Defenses

There have been a number of interesting proposals to use deceit for enhancing the security of computer systems beyond the traditional notion of honeypots. In this section we give an overview of some of these schemes.

Passwords and Credentials Protection

Li and Schmitz proposed a framework to address phishing by using deception and honeypot-like techniques [94]. The authors propose a framework that introduces the concept of fake credential, referred to as a phoneytoken, and a number of client honeypots, referred to as phoneybots. The main idea in their framework is that when phishing is detected a number of phoneytokens will be sent to the phishing site. If phishing is detected by a spamtrap, a real user will have to submit a phoneytoken, however, if the detected phishing is using pharming or malware attacks, a phoneybot will submit the phoneytoken. Banks can monitor these phoneytokens and then follow the money trail when phishers are detected stealing money.

BogusBiter is a similar scheme proposed by Yue and Wang in [174]. The authors develop a client add-on to the user's browser that intercepts username/password submissions when users override a phishing warning. Instead of stopping the submission they submit additional $(N - 1)$ username/password pairs generated based on the user's credentials. The scheme also works with savvy users who obey the warnings where the add-on submits a large number of randomly generated credentials to the phishing website. The scheme also requires the installation of a server side component that analyzes a username/passwords submission and triggers a silent alarm when a "Bogus" credential has been submitted.

Most recently, a scheme named *Honeywords* was proposed by Jules and Rivest to confuse attackers when they crack a stolen hashed password file [84] by hiding the real password among a list of "fake" ones. Their scheme augmenting password databases with an additional $(N-1)$ fake credentials [84]. If the DB is stolen and cracked, attackers are faced with N different passwords to choose from where only one of them is the correct one. However, if they use any of the fake ones the system triggers an alarm alerting system administrators that the DB has been cracked.

Kontaxis et al. proposed a similar scheme in [87]. Their proposal relies on the fact that users need to supply a voucher obtained from a vouching server along with their username/password, which is an extra step they introduce in their scheme. The vouching request must originate from the target server. They also add $(N-1)$ decoy passwords to the credentials DB. Unlike Honeyword, these password actually log the user in and their main goal is to address the issue of a user using the same passwords with the target and vouching servers.

Moreover, Zhao and Mannan used deceptive techniques to limit the effectiveness of automated online password guessing [177]. They provide "fake" sessions to an adversary who is launching automated attacks while real users will detect the authentication outcome implicitly from the presented user data.

Defaming Botnets

Ormerod et al. proposed a scheme that inject deceptive fake information to current botnet zombies for two main goals: dilute the real stolen information and trace end-users of botnet's stolen information when they use this fake information [111]. Similar to the honeywords proposal above, this fake information signals an alarm that a "stolen" credentials/credit card/identity is currently being used.

Obfuscation and Anti-Reconnaissance

Murphy et al. investigated the efficacy of using a host-based operating system (OS) obfuscation as an integral part of Air Force computer defenses [107]. The observation that motivated their study is that identifying the target's OS is a key component in any computer attack. Successfully masking this information can give computer defenders an advantage. They used the *OSfuscate* tool [40], by Crenshaw, and concluded that it is effective in continuously obfuscating the host OS. They recommend deploying this technique as part of Air Force computer defenses. However, a challenge to deploying these methods can arise from the need to use administrative tools that rely on accurately fingerprinting the OSs of managed computer systems to undergo regular maintenance and patching operations.

Active Defense

Crane et al. discuss the use of "Booby Trapping Software" — an active security defense mechanism for code-reuse attacks where deceptive techniques are used [39]. In additiona, Cohen and Koike presented a set of experiments where they successfully induced skilled red-team attackers to attack the targeted system in a particular sequence [35]. The main goal was to mimic physical attack tactics where such techniques can be used to drive prey into kill-zones by influencing their decisions, by means of deception techniques, taking a specific path desired by the defenders. This was part of a larger set of experiments where they used different deceptive mechanisms against red-team attackers [36].

Trassare takes a different approach of using deception and presents a technique to deceive attackers, who attack DoD networks by giving them a fake internal network topology of the defender's choice [152]. He presents a prototype implementation showing positive results.

Rowe et al. used a "testbed" for automated defensive deception planning for cyber-attacks [132]. Their approach was to make a complete system available for attackers to understand and plan deceptive operations for other systems. They reported interesting findings of attackers' behavior that can be used to design effective deceptive computer defenses.

Supply-Chain Protection

Spiegel published a report showing how the NSA intercepts equipment shipped by Cisco and installs eavesdropping implants in it. To address this problem, Cisco announced that they will help their customers by using some deceptive techniques to mislead NSA. They offered to ship customer equipment to a fake address making it harder for the NSA to target and contaminate their supply chain [114].

3.8 Deception Operations and Tactics

Deception has long been used as an effective operational tactic in warfare and military conflicts. Fowler and Nesbitt highlight six rules for a successful deception operation [58].

1. **Expectedness.** A successful deception should cause the enemy to believe what he expects. The deceptive act should be designed to look no different than the normal expected act, while the real act should be the surprising one.

2. **Timely Feedback.** A successful deception operation should involve a continuous and timely feedback of the adversary's reaction to the deceptive information. This is crucially important as the targeted system could be vulnerable if attackers successfully avoid the deceptive operation or conduct a counter-deception operation.

3. **Integration.** The deception operation must be tightly integrated with the real operation. In other words, real and deceptive plans must work together supplementing each other's activities.

4. **Suppression.** A deception plan must not only provide believable activities for the deception operation, but also hide any activity of the real operation.

5. **Realism.** The realism of any deception operation is a function of two important factors; the adversary's *capabilities* of observing responses and the *time* available to analyze these responses. As an example, deceiving a drone attack with fake tanks requires a different level of realism than deceiving an attacking army with tanks on the borders of another country.

6. **Creativity.** A successful deception operation should be imaginative and creative.

Rowe and Rothstein used these rules and applied them to the case of cyberwar in [133]. It can be seen that many of these rules highlight a number of limitations of current deception-based defenses. As an example, honeypots violate the third rule of integration as they are, in the default case, a standalone system(s) that are only useful if the attackers decide to interact with them.

3.9 Chapter Summary

In this chapter, we discussed the concept of deception and how it has been used. We illustrated some of the well-known taxonomies of deceptive techniques; we adapt these techniques in next chapter to show how they can be used to enhance computers' security. We gave an overview of deception maxims and concepts. We presented an investigation of the role of biases in ensuring the success of any deceptive technique. After that, we then gave an overview of the use of deception in military conflicts and computing. We discussed how deception has been widely used to enhance the utility of technology and improve users' experiences. Additionally, we gave an overview of the previous uses of deception in computer security. We discussed the well-known example of using honeypots to aid computer security. We concluded the chapter by discussing some of the most important principles in deception operations and tactics.

4 A FRAMEWORK FOR USING DECEPTION TO ENHANCE SECURITY

In everyday security, deception plays a prominent role in our lives. We leave lights on to deter thieves and deceive them by pretending that someone is inside. To automate such deceptive behavior, we might even have a timer that switches the light on and off. Through history, deception has evolved to find its natural place in our societies and eventually our technical systems. Deception and decoy-based mechanisms have been used in security for more than two decades in techniques such as honeypots and honeytokens, as discussed in the previous chapter. Nevertheless, little work has been done in incorporating deception beyond such traditional concepts.

Deception-based techniques provide significant advantages over traditional security controls. Currently, most security tools are responsive measures to attackers' probes to previously known vulnerabilities. Whenever an attack surfaces, it is hit hard with all preventative mechanisms at the defender's disposal. Eventually, persistent attackers find a vulnerability that leads to a successful infiltration by evading the way tools detect probes or by finding new unknown vulnerabilities. This security posture is partially driven by the assumption that *"hacking-back"* is unethical, while there is a difference between the act of "attacking back" and the act of deceiving attackers, which is further discussed in section 4.6.2. With such behavior, attackers progressively learn about systems' defensive capabilities with their continuous probing. As a result, average computer systems are guiding their adversaries in how to successfully infiltrate their own defenses. Meanwhile, targeted systems learn nothing about these attempts, other than a panic in the security team. In fact, in many cases multiple attempts that originate from the same entity are not successfully correlated.

There is a fundamental difference in how deception-based mechanisms work in contrast to traditional security controls. The latter usually focuses on attackers' actions — detecting or preventing them — while the former focuses on attackers' perceptions — manipulating them and therefore inducing adversaries to take actions/inactions in ways that are advantageous to targeted systems; traditional security controls position themselves in response to attackers' actions while deception-based tools are positioned in prospect of such actions. Later, in section 4.3.1, we discuss some of the unique advantages deception-based security defenses bring.

4.1 Definition

One of the most widely accepted definitions of computer-security deception is the one by Yuill [175]; Computer Deception is *"Planned actions taken to mislead attackers and to thereby cause them to take (or not take) specific actions that aid computer-security defenses."* We adapt this definition and add "confusion" as one of goals of using deceit (the expression of things that are not true) in computer system protection, as we will discuss later in section 4.6.3. Therefore, the definition of defensive computer deception we will use throughout this dissertation is

> *"Planned actions taken to mislead and/or confuse attackers and to thereby cause them to take (or not take) specific actions that aid computer-security defenses"*

4.2 Limitations of Isolated Use of Deception

Honeypot-based tools are a valuable technique used for the detection, prevention, and response to cyber attacks as we discussed in section 3.7.1. Nevertheless, those techniques suffer from the following major limitations:

- As the prefix *honey-** indicates, for such techniques to become useful, the adversary needs to interact with them. Attackers and malware are increasingly becoming sophisticated and their ability to avoid honeypots is increasing [27].
- Assuming we manage to lure the attacker into our honeypot, we need to be able to *continuously* deceive them that they are in the real system. Chen et

al. study such a challenge and show that some malware, such as polymorphic malware, not only detects honeypots, but also changes its behavior to deceive the honeypot itself [27]. In this situation, attackers are in a position where they have the ability to conduct counter-deception activities by behaving in a manner that is different than how would they do in a real environment.

- To learn about attackers' objectives and attribute them, we need them to interact with the honeypot systems. However, with a high-interaction honeypot there is a risk that attackers might exploit the honeypot itself and use it as a pivot point to compromise other, more sensitive, parts of the organization's internal systems. Of course, with correct separation and DMZs we can alleviate the damage, but many organizations consider the risk intolerable and simply avoid using such tools.

- As honeypots are totally "fake systems" many tools currently exist to identify whether the current system is a honeypot or not [27, 79]. This fundamental limitation is intrinsic in their design.

4.3 The Role of Deception

Most of the previous deception techniques work in isolation and independently of other parts of information systems. This design decision has been partly driven by the security risks associated with honeypots. We argue that intelligently augmenting our systems with interacting deception-based techniques can significantly enhance our security and gives us the ability to achieve deception in depth.

If we examine table 2.1, we can see that we can apply deception at every stage of the cyber kill-chain, allowing us to break the chain and possibly attribute attackers. At the reconnaissance stage we can lure adversaries by creating a site and having honey-activities that mimic a real-world organization. As an example, an organization can subscribe with a number of cloud service providers and have honey activities in place while monitoring any activities that signal external interest. Another example to address the problem of spear-phishing, we can create a number of fake personas and disseminate their information online while monitoring their contact details to detect any probing activities; some commercial security firms currently do this.

4.3.1 Advantages of Using Deception in Computer Defenses

Reginald Jones, the British scientific military intelligence scholar, concisely articulated the relationship between security and deception. He referred to security as a *"negative activity, in that you are trying to stop the flow of clues to an opponent"* and it needs its other counterpart, namely deception, to have a competitive advantage in a conflict [83]. He refers to deception as the "positive counterpart to security" that provides false clues to be fed to the opponents.

By intelligently using deceptive techniques systems defenders can mislead and/or confuse attackers enhancing their defensive capabilities over time. By exploiting attackers' unquestioned *trust* of computer system responses, system defenders can gain an edge and position themselves a step ahead of compromise attempts. In general, deception-based security defenses bring the following unique advantages to computer systems:

1. *Increases the entropy of leaked information about targeted systems during compromise attempts.*

   When a computer system is targeted, the focus is usually only on protecting and defending it. With deception, extra defensive measures can be taken by feeding attackers false information that will, in addition to defending the targeted system, cause intruders to make wrong actions/inactions and draw incorrect conclusions. With the increased spread of APT attacks and government/corporate espionage threats such techniques can be effective.

   When we inject false information we cause some confusion for the adversaries even if they have already obtained some sensitive information; the injection of negative information can degrade and devalue the correct information obtained by adversaries. Heckman and her team, from MITRE, conducted an experiment between a red and a blue team using deception techniques, where they reported interesting results [74]. They developed a tool, referred to as "Blackjack," that dynamically copies an internal state of a production server – after removing sensitive information and injecting deceit – and then directs adversaries to that instance [74]. Even after the red team successfully attacked and infiltrated the blue systems and obtained sensitive information, the blue team injected some false information in their system that led the red team to devalue the information they had obtained, believing that the new values were correct.

2. *Increases the information obtained from compromise attempts.*

   Many security controls are designed to create a boundary around computer systems automatically stopping any illicit access attempts. This is becoming increasingly challenging as such boundaries are increasingly blurring partly as a result of recent trends such as "consumerization"[1] [73]. Moreover, because of the low cost on the adversaries' side, and the existence of many automated exploitation tools, attackers can continuously probe computer systems until they find a vulnerability to infiltrate undetected. During this process, systems defenders learn nothing about the intruders' targets. Ironically, this makes the task of defending a computer system harder after every unsuccessful attack. We conjecture that incorporating deception-based techniques can enhance our understanding of compromise attempts using the illicit probing activity as opportunity to enhance our understanding of the threats and, therefore, better protect our systems over time.

3. *Give defenders an edge in the OODA loop.*

   The OODA loop (for Observe, Orient, Decide, and Act) is a cyclic process model, proposed by John Boyd, by which an entity reacts to an event [20]. The

[1]This term is widely used to refer to enterprises' employees bringing their own digital devises and using them to access the companies' resources.

victory in any tactical conflict requires executing this loop in a manner that is faster than the opponent. The act of defending a computer system against persistent attacks can be viewed as an OODA loop race between the attacker and the defender. The winner of this conflict is the entity that executes this loop faster. One critical advantage of deception-based defenses is that they give defenders an edge in such a race as they actively feed adversaries deceptive information that affects their OODA loop, more specifically the "observe" and "orient" stages of the loop. Furthermore, slowing the adversary's process gives defenders more time to decide and act. This is especially crucial in the situation of surprise, which is a common theme in digital attacks.

4. *Increases the risk of attacking computer systems from the adversaries' side.*

   Many current security controls focus on preventing the actions associated with illicit attempts to access computer systems. As a result, intruders are using this accurate negative feedback as an indication that their attempts have been detected. Subsequently, they withdraw and use other, more stealthy, methods of infiltration. Incorporating deceit in the design of computer systems introduces a new possibility that adversaries need to account for; namely that they have been detected and currently deceived. This new possibility can deter attackers who are not willing to take the risk of being deceived, and further analyzed. In addition, such technique gives systems' defenders the ability to use intruders' infiltration attempts to their advantage by actively feeding them false information.

4.4 Related Work of Modeling the Use of Deception in Security

Cohen et al. was one of the first to develop a model for using deception in computer defenses [34]. They provided a general overview of human and computer deception. Their work was motivated by the deception toolkit (DTK), discussed in section 3.7, and discusses how a system can be designed to deceive attackers.

Game theory has been used to study some deception-based techniques. Carroll and Grosu presented an analysis of using deception for network security modeling the problem as a *signaling game* [24]. They modeled the interaction between the defender and the attacker where defenders can deploy a honeypot or a normal system; or they can camouflage any of the two. The authors developed an equilibrium of defenders action. In addition, Garg and Grosu analyzed the deception of honeynets using game theory to provide defenders the best strategy in deploying deception [62].

Rowe modeled attackers interaction with a computer system and discussed how can we plan a deceptive "resource denial" response effectively [130]. Such responses are designed to waste adversaries' time and resources while alerting systems' defenders of potential attacks.

4.5 A Framework for Integrating Deception-Based Defenses

In this section, we present a framework that can be used to plan and integrate deception in computer security defenses. Many computer defenses that use deception were ad-hoc attempts to incorporate deceptive elements in their design. We show how our framework can be used to incorporate deception in many parts of a computer system and discuss how we can use such techniques effectively. A successful deception should present plausible alternative(s) to the truth and these should be designed to exploit specific adversaries' biases, as discussed in section 3.4.

The framework discussed in this section is based on the general deception model discussed by Bell and Whaley in [11]. There are three general phases of any deceptive component; namely planning, implementing and integrating, and finally monitoring and evaluating. In the following sections we discuss each one of those phases in more detail. The framework is depicted in figure 4.1.

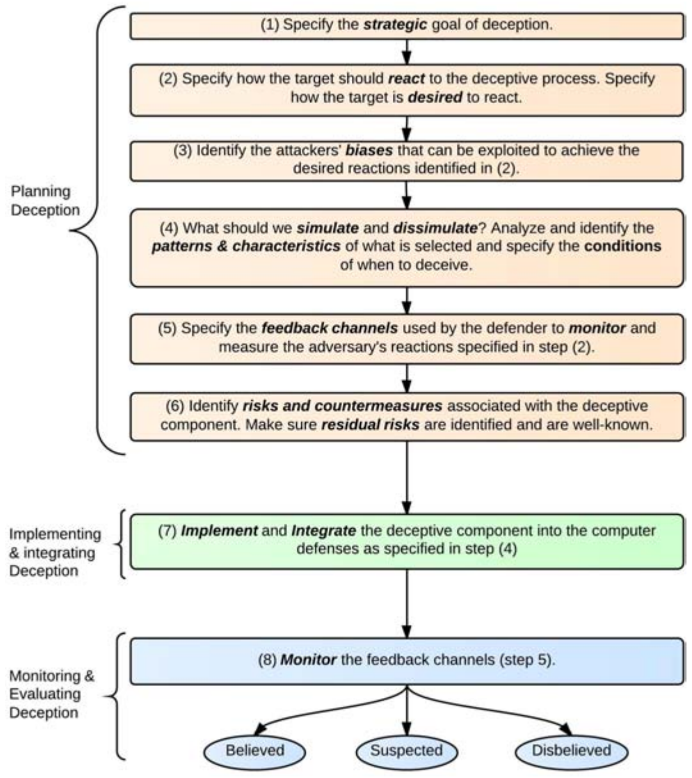

(1) Specify the strategic goal of deception.
(2) Specify how the target should react to the deceptive process. Specify how the target is desired to react.
(3) Identify the attackers' biases that can be exploited to achieve the desired reactions identified in (2).
(4) What should we simulate and dissimulate? Analyze and identify the patterns & characteristics of what is selected and specify the conditions of when to deceive.
(5) Specify the feedback channels used by the defender to monitor and measure the adversary's reactions specified in step (2).
(6) Identify risks and countermeasures associated with the deceptive component. Make sure residual risks are identified and are well-known.
Planning Deception
(7) Implement and Integrate the deceptive component into the computer defenses as specified in step (4)
Implementing & integrating Deception
(8) Monitor the feedback channels (step 5).
Monitoring & Evaluating Deception
Believed
Suspected
Disbelieved

Figure 4.1.: Framework to Incorporate Deception in Computer Security Defenses

4.5.1 Planning Deception

There are six essential steps to planning a successful deception-based defensive component. The first, and often neglected, step is specifying exactly the *strategic goals* the defender wants to achieve. Simply augmenting a computer system with honey-like components, such as honeypots and honeyfiles, gives us a false sense that we are using deception to lie to adversaries. It is essential to detail exactly what are the goals of using any deception-based mechanisms. As an example, it is significantly different to set up a honeypot for the purpose of simply capturing malware than having a honeypot to closely monitor APT-like attacks.

After specifying the strategic goals of the deception process, we need to specify – in the second step of the framework – how the target (attacker) should react to the deception. This determination is critical to the long-term success of any deceptive process. For example the work of Zhao and Mannan, discussed in section 3.7.3, deceive attackers launching online guessing attacks into believing that they have found a correct username and password. The strategic goal of this deception process is to direct an attacker to a “fake” account thus wasting their resources and monitoring their activities to learn about their objectives. It is crucial to analyze how the target should react after the successful “fake” login. The obvious reaction is that the attacker would continue to laterally move in the target system, attempting further compromise. However, an alternative response is that the attacker ceases the guessing attack and reports to its command and control that a successful username/password pair has been found. In consideration of the second alternative we might need to maintain the username/password pair of the fake account and keep that account information consistent for future targeting.

Moreover, part of this second step is to specify how we desire an attacker to react such that we may try to influence his perception and thus lead him to the desired reaction. Continuing with the example in the previous paragraph, if we want the attacker to login again so we have more time to monitor and setup a fake account, we might cause an artificial network disconnection that will cause the target to login again.

Adversaries’ Biases

Deception-based defenses are useful tools that have been shown to be effective in many human conflicts. Their effectiveness relies on the fact that they are designed to exploit specific biases in how people think, making them appear to be plausible but false alternatives to the hidden truth, as discussed in section 3.4. These mechanisms give defenders the ability to learn more about their attackers, reduce indirect information leakages in their systems, and provide an advantage with regard to their defenses.

Step 3 of planning deception is to understand the attackers' biases. As discussed in section 3.4, biases are a cornerstone component to the success of any deception-based mechanisms. The deceiver needs to present a plausible deceit to successfully deceive and/or confuse an adversary. If attackers decide that such information is not plausible they are more inclined to reject it, or at least raise their suspicions about the possibility of currently being deceived. When the defender determines the strategic goal of the deception and the desired reactions by the target, he needs to investigate the attacker's biases to decide how best to influence the attacker's perception to achieve the desired reactions.

One example of using biases in developing some deceptive computer defenses is using the *"confirmation bias"* to lead adversaries astray and waste their time and resources. Confirmation bias is defined as *"the seeking or interpreting of evidence in ways that are partial to existing beliefs, expectations, or a hypothesis in hand"* [110]. A computer defender can use this bias in responding to a known adversarial probing of the system's perimeter. Traditional security defenses are intended to detect and prevent such activity, by simply dropping such requests or actively responding with an explicit denial. Taking this a step further by exploiting some pre-existing expectation, i.e. the confirmation bias, we might provide a response that the system is being taken down for some regular maintenance or as a result of some unexpected failure. With such a response, the defender manages to prevent illicit activity, provide a pause to consider next steps for the defender, and perhaps waste the adversary's time as they wait or investigate other alternatives to continue their attacks.

Cultural biases play an important role in designing deceptive responses, as discussed in section 3.4.2. For example, some studies found relationships between the type of computer attacks and the culture/country from which the attack originated [135].

In computing, the conjunction fallacy bias, discussed in section 3.4.4, can be exploited by presenting the deception story as a conjunction of multiple detailed components. For example, if deceivers want to misinform an attacker probing their system by creating an artificial network failure, instead of simply blocking these attempts, it is better to give a longer story. A message that says "Sorry the network is down for some scheduled network maintenance. Please come back in three hours" is more plausible than simply saying "The network is down" and thus more likely to be believed.

Creating the Deception Story

After analyzing attackers' biases the deceiver needs to decide exactly what components to simulate/dissimulate; namely step 4 of the framework in figure 4.1.

In figure 4.2 we provide an overview of the different system components where deception can be applied, exploiting the attacker's biases to achieve the desired reaction. Overall, deceit can be injected into the functionality and/or state of our systems. We give a discussion of each one of these categories below and present some examples.

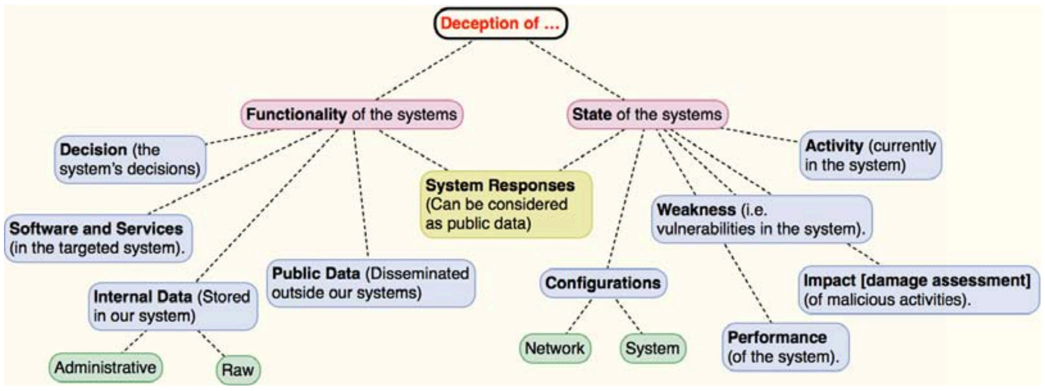

Deception of ...
Functionality of the systems
State of the systems
Decision (the system's decisions)
Software and Services (in the targeted system).
Internal Data (Stored in our system)
Administrative
Raw
Public Data (Disseminated outside our systems)
System Responses (Can be considered as public data)
Configurations
Network
System
Weakness (i.e. vulnerabilities in the system).
Performance (of the system).
Activity (currently in the system)
Impact [damage assessment] (of malicious activities).

Figure 4.2.: Computer Systems Components Where Deception Can Be Integrated With

System's Decisions. We can apply deception to the different decisions any computer system makes. As an example, Zhao and Mannan work, discussed in section 3.7.3, apply deception at the system's authentication decision where they deceive adversaries by giving them access to "fake" accounts in the cases of online guessing attacks. Another system's decision we can use concerns firewalls. Traditionally, we add firewall rules that prevent specific IP addresses from interacting with our systems after detecting that they are sources of some attacks. We consider this another form of data leakage in accordance with the discussion of Zhao and Mannan in [177]. Therefore, we can augment firewalls by applying deception to their decisions by presenting adversaries with plausible responses other than simply denying access. We discuss this further in the design of the deceptive server *"Deceptiver"* in chapter 7.

System's Software and Services. Reconnaissance is the first stage of any attack on any computing system, as identified in the kill-chain model [80]. Providing fake systems and services has been the main focus of honeypot-based mechanisms. Honeypots, discussed in section 3.7.1, are intended to provide attackers with a number of fake systems running fake services. Moreover, we can use deception to mask the identities of our current existing software/services. The work of Murphy et al., discussed in section 3.7.3, recommended the use of operating system obfuscation tools for Air Force computer defenses [107].

System's Internal and Public Data. A honeyfile, discussed in section 3.7.2, is an example of injecting deceit into the system's internal data. It can be applied to the raw data in computer systems, e.g., files and directories, or to the administrative data that are used to make decisions and/or monitor the system's activities. An example applying deception to the administrative data can be seen in the *honeywords* proposal, discussed in section 3.7.3. Deceit can also be injected into the public data about our systems. Wang et al. made the case of disseminating public data about some "fake" personnel for the purpose of catching attacks such as spear phishing [165]. Cliff Stoll did this during the story of his book [146]. In addition, we note that this category also includes offline stored data such as back-ups that can be used as a focus of deception. In chapter 6, we present a scheme that applies deception to system's administrative data to enhance the security of users' credentials.

System's Activity. Different activities within a system are considered as one source of information leakage. For example, traffic flow analysis has long been studied as a means for attackers to deduce information [60]. Additionally, a system's activity has been used as a means of distinguishing between a "fake" and a real system [27]. We can intelligently inject some data about activities into our system to influence attackers' perception and, therefore, their reactions.

System's Weaknesses. Adversaries probe computer systems trying to discover and then exploit any weakness (vulnerability). Often, these adversaries come prepared with a list of possible vulnerabilities and then try to use them until they discover something that works. Traditional security mechanisms aid adversaries by quickly and promptly responding back to any attempt to exploit fixed, i.e. patched, vulnerabilities with a denial response. This response leaks information that these vulnerabilities are known and fixed. When we inject deceit into this aspect of our systems we can misinform adversaries by confusing them – by not giving them a definitive answer whether the exploit has succeeded – or by deceiving them by making it appear as if the vulnerability has been exploited.

System's Damage Assessment. This relates to the previous component; however, the focus here is to make the attacker perceive that the damage caused is more or less than the real damage. We may want the adversary to believe that he has caused more damage than what has happened so as to either stop the attack or cause the attacker to become less aggressive. This is especially important in the context of the OODA loop discussed earlier in section 4.3.1. We might want the adversary to believe that he has caused less damage if we want to learn more about the attacker by prompting a more aggressive attack.

System's Performance. Influencing the attacker's perception of system's performance may put the deceiver at an advantageous position. This has been seen in the use of *sticky honeypots* and tarpits, discussed in section 3.7.1, that are intended to slow the adversary's probing activity. Also, tarpits have been used to throttle the spread of network malware. In a related fashion, Somayaji et al. proposed a method to deal with intrusions by slowing the operating system response to a series of anomalous system calls [77].

System's Configurations. Knowledge of the configuration of the defender's systems and networks is often of great importance to the success of the adversary's attack. In the lateral movement phase of the kill-chain adversarial model, attackers need to know how and where to move to act on their targets. In the red-teaming experiment by Cohen and Koike, discussed in section 3.7.3, they deceived adversaries to attack the targeted system in a particular sequence from a networking perspective.

After deciding which components to simulate/dissimulate, we can apply one of Bell and Whaley's techniques discussed earlier. We give an example of how each one of these techniques can be used in the following paragraphs.

- *Using Masking* – This has been used offensively where attackers hide potentially damaging scripts in the background of the page by matching the text color with the background color. When we apply hiding to software and services, we can hide the fact that we are running some specific services when we detect a probing activity. For example, when we receive an SSH connection request from a known bad IP address we can mask our SSHd demon and respond as if the service is not working or as if it is encountering an error.

- *Using Repackaging* – In several cases it might be easier to "repackage" data as something else. In computing, repackaging has long been used to attack computer users. The infamous cross-site scripting (XSS) attack uses this technique where an attacker masks a dangerous post as harmless to steal the user's cookies when they view such post. Another example can be seen in the cross-site request forgery (XSRF) attacks where an adversary deceives a user into visiting some innocuous looking web pages that silently instruct the user's browser to engage in some unwanted activities. In addition, repackaging techniques are used by botnet Trojans that repackage themselves as anti-virus software to deceive users into installing them so an attacker can take control of their machines. From the defensive standpoint, a repackaging act can be seen in HoneyFiles, discussed in section 3.7.2, that repackage themselves as normal files while acting internally as silent alarms to system administrators when accessed.

- *Using Dazzling* – This is considered to be the weakest form of dissimulation, where we confuse the targeted objects with others. An example of using dazzling can be seen in the *"honeywords"* proposal, discussed in section 3.7.3. The scheme confuses each user's hashed password with an extra $(N - 1)$ hashes of other, similar, passwords dazzling an attacker who obtains the credentials database.

- *Using Mimicking* – In computing, phishing attacks are a traditional example of an unwanted deceiving login page mimicking a real website login. An attacker takes advantage of users by deceiving them into giving up their credentials by appearing as the real site. From a defensive perspective, we can apply mimicking to software and services by making our system mimic the responses of a different system, e.g., respond as if we are running a version of Windows XP while we are running Windows 7. This will waste attackers' resources in trying to exploit our Windows 7 machine thinking it is Windows XP, as well as increase the opportunity for discovery. This is seen in the work of Murphy et al. in operating system obfuscation discussed in section 3.7.3.

- *Using Inventing* – Mimicking requires the results to look like something else; when this is not easy to achieve invention can be used instead. This technique has seen the most research in the application of deception to computer security defenses. Honeypots, discussed in section 3.7.1, are one prominent example of inventing a number of nodes in an organizations with the goal of deceiving an attacker that they are real systems.

- *Using Decoying* – This technique is used to attract adversaries' attention away from the most valuable parts of a computer system. Honeypots are used, in some cases, to deceive attackers by showing that these systems are more vulnerable than other parts of the organization and therefore capture attackers' attention. This can be seen in the work of Carroll and Grosu [24].

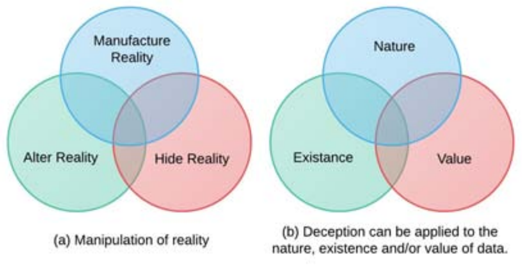

Manufacture Reality
Alter Reality
Hide Reality
(a) Manipulation of reality
Nature
Existance
Value
(b) Deception can be applied to the nature, existence and/or value of data.

Figure 4.3.: Creating Deceit

After deciding which deceptive technique to use we need to analyze the patterns attackers perceive and then apply one or more of those techniques to achieve the desired reactions.

Deceit is an active manipulation of reality. We argue that reality can be manipulated in one of three general ways, as depicted in figure 4.3-a. We can *manufacture* reality, *alter* reality, and/or *hide* reality. This can be applied to any one of the components we discussed in the previous section.

In addition, reality manipulation is not only to be applied to the existence of the data in our systems — it can be applied to two other features of the data. As represented in figure 4.3-b, we can manipulate the reality with respect to the *existence* of data, *nature* of the data, and/or *value* of the data. The existence of the data can be manipulated not only for the present but also when the data has been created. This can be achieved for example with the manipulation of time stamps. With regard to the nature of the data, we can manipulate the size of the data, such as in the example of endless files, discussed in section 2.1.3, when and why the data has been created. The value of the data can also be manipulated. For example, log files are usually considered important data that adversaries try to delete to cover their tracks. Making a file appear as a log file will increase its value from the adversary's perspective.

At this step, it is crucial to specify exactly when the deception process should be activated. It is usually important that legitimate users' activity should not be hindered by the deceptive components. Optimally, the deception should only be activated in the case of malicious interactions. However, we recognize that this may not always be possible as the lines between legitimate and malicious activities might be blurry. We argue that there are many defensive measures that can apply some deceptive techniques in place of the traditional denial-based defenses that can make these tradeoffs.

Feedback Channels and Risks

Deception-based defenses are not a single one-time defensive measure, as is the case with many advanced computer defenses. It is essential to monitor these defenses, and more importantly measure the impact they have on attackers' perceptions and actions. This is step 5 in the deception framework. We recognize that if an attacker detects that he is being deceived, he can use this to his advantage to make a counter-deception reaction. To successfully monitor such activities we need to clearly identity the deception channels that can and should be used to monitor and measure any adversary's perceptions and actions.

In the sixth and final step before implementation and integration, we need to consider that deception may introduce some new risks for which organizations need to account. For example, the fact that adversaries can launch a counter-deception operation is a new risk that needs to be analyzed. In addition, an analysis needs to done on the effects of deception on normal users' activities. The defender needs to accurately identify potential risks associated with the use of such deceptive components and ensure that residual risks are accepted and well identified.

4.5.2 Implementing and Integrating Deception

Many deception-based mechanisms are implemented as a separate disjoint component from real production systems, as in the honeypot example. With the advancement of many detection techniques used by adversaries and malware, attackers can detect whether they are in real system or a “fake” system [27], and then change behavior accordingly, as we discussed in section 4.2. A successful deception operation needs to be integrated with the real operation. The honeywords proposal, discussed in section 3.7.3, is an example of this tight integration as there is no obvious way to distinguish between a real and a “fake” password.

4.5.3 Monitoring and Evaluating the Use of Deception

Identifying and monitoring the feedback channels is critical to the success of any deception operation/component. Hesketh discussed three general categories of signals that can be used to know whether a deception was successful or not [75]:

1. The target acts in the wrong time and/or place.
2. The target acts in a way that is wasteful of his resources.
3. The target delays acting or stop acting at all.

Defenders need to monitor all the feedback channels identified in step 5 of the framework. We note that there are usually three general outputs from the use of any deceptive components. The adversary might (i) *believe* it, where the defender usually sees one of the three signs of a successful deception highlighted above, (ii) *suspect* it or (iii) *disbelieve* it. When an attacker suspects that a deceptive component is being used, we should make the decision whether to increase the level of deception or stop the deceptive component to avoid exposure. Often deception can be enhanced by presenting more (and perhaps, true) information that makes the deception story more plausible. This can be included as a feedback loop in the framework. This observation should be analyzed by the defender to review his analysis of the attacker's biases, (i.e., step 3), and the methodology used to create the deceit (i.e., step 4). Furthermore, the deceiver might employ multiple levels of deception based on the interaction with the attacker during the attack.

When an attacker disbelieves the presented deceit we need to have an active monitoring and a detailed plan of action. This should be part the sixth step of planning in our framework where risks are assessed. In addition, during our discussions with security practitioners many have indicated that some attackers often act aggressively when they realize that they have been deceived. This can be one of the signals that is used during the monitoring stage to measure attackers' reaction of the deceptive component. In addition, this behavior can be used as one of the biases to be exploited by other deceptive mechanisms that may focus on deceiving the attacker about the *system's damage assessment*, as discussed in section 4.5.1.

4.6 Deception and Related Concepts

4.6.1 Kerckhoff's Principle and Deception

Deception always involves two basic steps, hiding the real and showing the false, as we discussed earlier. This, at first glance, contradicts the widely believed misinterpretation of Kerckhoff's principle; *"no security through obscurity."* A more correct English translation of Kerckhoff's principle is the one provided by Petitcolas in [117];

> *"The system must not require secrecy and can be stolen by the enemy without causing trouble."*

The misinterpretation leads some security practitioners to believe that any "obscurity" is ineffective, while this is not the case. Hiding a system from an attacker or having a secret password does increase the work factor for the attacker — until the deception is detected and defeated. So long as the security does not materially depend on the obscurity, the addition of misdirection and deceit provides an advantage. It is therefore valuable for a designer to include such mechanisms in a comprehensive defense, with the knowledge that such mechanisms should not be viewed as primary defenses.

In any system design there are three levels of viewing a system's behavior and responses to service requests:

- *Truthful.* In such systems, the processes will always respond to any input with full "honesty." In other words, the system's responses are always "trusted" and accurately represent the internal state of the machine. For example, when the user asks for a particular network port, a truthful system responds with either a real port number or denies the request giving the specific reason of such denial.

- *Naively Deceptive.* In such systems, the processes attempt to deceive the interacting user by crafting an artificial response. However, if the user knows the deceptive behavior, e.g. by analyzing the previous deceptive response used by the system, the deception act becomes useless and will only alert the user that the system is trying to deceive her. For example, the system can designate a specific port that is used for deceptive purposes. When the attacker asks for a port, without carrying the appropriate permissions, this deceptive port is sent back.

- *Intelligently Deceptive.* In this case, the systems "deceptive behavior" is indistinguishable from the normal behavior even if the user has previously interacted with the system. For example, an intelligently-deceptive system responds to unauthorized port listening requests the same as a normal allowed request. However, extra actions are taken to monitor the port, alert the system administrators, and/or sandbox the listening process to limit the damage if the process downloads malicious content.

The mechanisms discussed in chapters 5, 6, and 7 are designed to be intelligently deceptive. In other words, an adversary who know the design details of these security controls, i.e. read this dissertation, will not be able to easily distinguish between real and fake information without expending extra time and computation.

4.6.2 Deception and Hacking Back

These two terms are orthogonal to each other, however, they are often mixed together driving the security community away from the use of deceptive techniques, as discussed earlier. Hacking back is an activity that involves the use of many techniques, and deception can be one of those techniques. The confusion between the two terms is partially driven by the abundant use of deception in war and military conflicts to launch offensive attacks. Moreover, the extensive use of deception by adversaries, and the negative connotations associated with it, contributed to the creation of the mental model that using deception is equal to hacking back.

4.6.3 Deception and Consistency

Most of the work in using deception is designed to provide plausible and *consistent* alternatives to the truth to adversaries. Neagoe and Bishop argue that deception can still be achieved without maintaining consistency [109]. Moreover, they postulate that inconsistency is favorable in some scenarios. When inconsistent deception is used, the goal of deception focuses of "discombobulate and disorient" – i.e. confuse – adversaries [109]. This wastes attackers time where they try to reason about the system's behavior and decide which perception reflects the reality of the computer system. This is why we added "confusion" in the definition of deception in computer security in section 4.1.

We argue that inconsistency can be an integral part of any deceptive-based techniques. Maintaining a fully consistent "fake" image of a sophisticated computer system may not be an easy task especially when considering that there are many ways to access the sought after data. If the sole goal of security administrators is to *confuse* the attacker, then the extra cost associated with implementing consistent deception is not needed. In addition, inconsistent deception can be used as a tactic when realizing that the consistent "fake" image can no longer be maintained.

4.6.4 Deception and Abstraction

It is crucial to distinguish between the act of deceit and abstraction. Although the line is fuzzy, Adar et al. suggests a simple test to distinguish between the two [1]. In abstraction, unlike deception, the user's behavior will remain largely unchanged if the user knows the real truth. In addition, we point out that deception always requires the act of simulation and dissimulation as discussed in section 3.1. However, abstraction only involves simulation, where we try to show a simpler version of the reality, but we do not actively dissimulate the truth.

Finally, another fundamental difference between deception and abstraction is the difference in their ultimate goals. Abstraction aids humans interacting with computers to make them better reason about these systems and their behavior. This is usually achieved by hiding complexities and suggesting useful analogies. In contrast, deception's goal is to corrupt such reasoning and influence humans perception to reach false conclusions about the systems they are interacting with.

4.7 Applying the Framework

4.7.1 To Previous Uses of Deception

In this section, we apply our framework to some of previous uses deception to enhance security. We discuss how the framework captures the design and implementation of those tools. Moreover, we highlight some of the missing components in the design that are identified by the framework.

Rowe developed a deceptive security tool that sends deceptive responses when it detects a real attack. We will take his work discussed in [128] and [127] and apply it to the framework discussed in this chapter. Rowe points out that the *goal* of his proposed deceptive tool is to “waste the attacker’s resources while permitting time to organize a better defense,” which is the first step in our framework. In addition, his goal is to consistently deceive attackers while interacting with the security tool. To achieve this goal, he implicitly discusses exploiting the *expectedness bias* – i.e. attackers expect computers to tell the truth. Rowe did not explicitly discuss how he desires the adversary to *react* to his system other than simply believing, omitting the second step of the framework. We argue that not explicitly discussing adversaries’ desired reaction leads the design of such tools to not realize their goals. When we do not explicitly specify what we consider success when an adversary interacts with the deceptive security tools, it becomes harder to quantify their value or how they can be integrated effectively with other security mechanisms.

Matching the fourth step of the framework, Rowe presents how this tool is going to *create* the deceit. Figure 4.4 shows the different system components where deception has been applied. In all of those, the tool *manufactures* a reality and presents to the adversary. Rowe explains in detail how deceit is created in each one of these examples in [128].

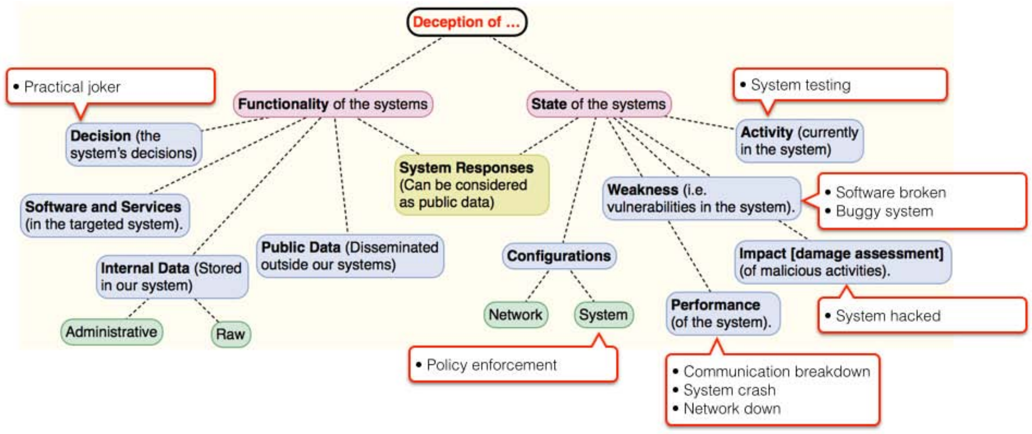

Deception of ...
Functionality of the systems
State of the systems
• Practical joker
Decision (the system's decisions)
Software and Services (in the targeted system).
Internal Data (Stored in our system)
Administrative
Raw
Public Data (Disseminated outside our systems)
System Responses (Can be considered as public data)
Configurations
Network
System
• Policy enforcement
• System testing
Activity (currently in the system)
Weakness (i.e. vulnerabilities in the system).
• Software broken
• Buggy system
Impact [damage assessment] (of malicious activities).
• System hacked
Performance (of the system).
• Communication breakdown
• System crash
• Network down

Figure 4.4.: System Components Used to Create Deceit in Rowe’s Work

After the creation of deceit, we found that Rowe did not explicitly discuss the *feedback channels* that should be monitored to observe the attacker's reaction. He implicitly touches on this concept when investigating the plausibility of the created deceit and how to maintain it. Moreover, his work does not discuss the additional risks that could be introduced by the use of such tools, if any, and possible counter measures.

Rowe examines how to *integrate* this security tool into a computer system. He uses a Bayesian belief update model to estimate the attacker's belief and alter the system's behavior. He generates a Markov graph by running a predicative-calculus planning specification hundreds of times using some probabilistic estimations obtained from a number of questionnaires. The system moves to different states based on the input received at every stage.

Another example we will discuss in this section to apply our framework is the work of Bowen et al. to mitigate the insider threat [19]. They integrated a decoy documents distributor and a mechanism to monitor whether the insider accessed the decoys with behavioral based host-based sensors. The authors clearly stated that the *goal* is to "confuse and confound attackers." After that, Bowen's group points out that their design would lead an adversary to *react* by expending more effort into distinguishing between the real and fake information. Even though they discussed how they desire the adversary to react, there was no discussion of other possible ways an attacker might react. This missing part of the second part of our framework causes the designer of deceptive defenses to focus on what they desire and possibly eliminate other undesired reactions by the adversary which could lead to additional risks. Additionally, the researchers present no discussion of any biases their tool is exploiting to make the deceit believable.

Figure 4.5 illustrates the two system components Bowen et al. applied deception to in their work [19]. They embedded honeytokens, discussed in 3.7.2, in the targeted system and these are internal administrative information. In addition, they distributed a number of beacons that alert a remote server when accessed and markers in every file to distinguish between real and fake file by host-base sensors. These two deceptive techniques apply deception to the raw internal data. To create the deceit, Bowen and his group manufacture reality, in the case of honeytokens, and alter reality, in the case of the beacons and documents markers.

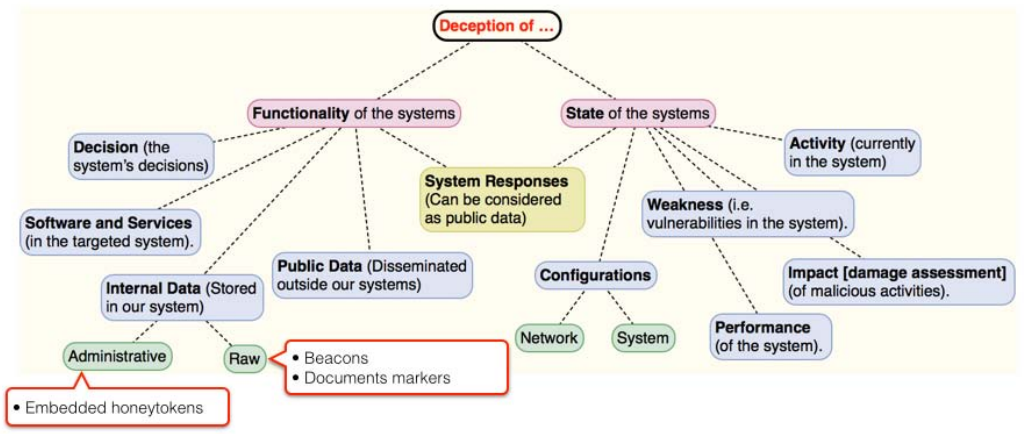

Deception of ...
Functionality of the systems
State of the systems
Decision (the system's decisions)
Software and Services (in the targeted system).
Internal Data (Stored in our system)
Public Data (Disseminated outside our systems)
System Responses (Can be considered as public data)
Activity (currently in the system)
Weakness (i.e. vulnerabilities in the system).
Configurations
Impact [damage assessment] (of malicious activities).
Performance (of the system).
Network
System
Administrative
Raw
• Beacons
• Documents markers
• Embedded honeytokens

Figure 4.5.: System Components Used to Create Deceit in Bowen et al. Work

Bowen et al. system *monitors* the deceptive components they have in place in two ways: a beacon calling back to their SONAR server, or using the host-based sensor. The only side-effect they discuss is the issue of false positives and how to go about reducing them. In addition, the researchers discuss generically how they would integrate their controls with an existing computer system. However, they defer the details to future research. Finally, Bowen and his group do not investigate the case where the adversary suspects the use of deception. It could be argued that because their goal is to require an adversary to expend more time in discerning the deceit from the truth, there is no need to consider a suspecting attacker as a separate case.

In this section, we discussed how our framework captures all the details on two previous work of deception; namely the work of Rowe [128] and Bowen et al. [19]. We showed that every part of the design of these two tools can be captured in using our framework. More importantly, we point out to some of the omitted steps in the design of these two security control that were highlighted in our framework. We show that the functionality of these tools could be improved if all steps in the framework were addressed explicitly.

4.7.2 To the Work in This Dissertation – A Case Study

In this section we present a case study of a web application to show how we can use the framework presented in this chapter and the tools in the next three chapters to enhance its security. In our discussion, we are assuming that this web application is developed to provide customers with a peer-to-peer payment service. Each customer has an account and she needs to login using her username and password whenever she needs to use the service.

As in common web applications, traditional security controls are used to ensure their security. In our case study, we use a *firewall* that only allows SSL/TLS connections to port 443. This firewall is configured to block all known bad requests using common blacklists. In addition, customers need to login to their accounts before making any requests. Each customer has a unique username and a password. Locally, at the application server, all user's passwords are salted and hashed. The overall structure of the web application and its security is depicted in figure 4.6.

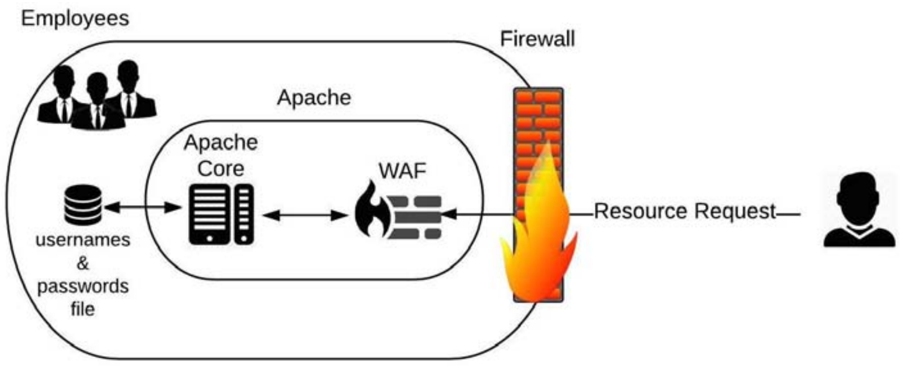


Figure 4.6.: Web Application Case Study

Despite all these protection mechanisms, customers occasionally receive phishing emails asking them to urgently login to the service or they risk loosing their account balances. Additionally, support emails posted publicly on the public webpages are often used by adversaries. Support teams often receive malicious attachments, delivered through a spear phishing email, as a mean to compromise internal servers. In addition, operators are suspect that they are targeted by some advanced attacks to steal users' credentials, similar to the ones that compromised other companies [63]. All these threats are illustrated in figure 4.7.

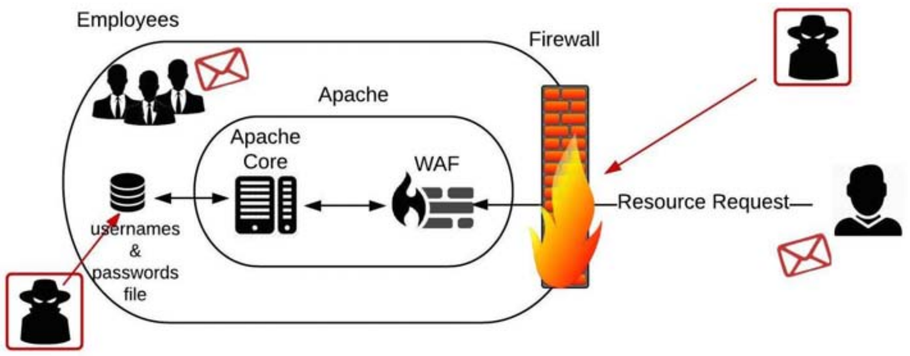


Figure 4.7.: Web Application Generic Threats

Using the framework discussed in this section, we will plan and integrate a number of deceptive security mechanisms to enhance the security of this web application. Table 4.1 summaries the result of using the framework discussed earlier in this chapter.

The overall design of augmenting the deception-based defenses with the web application is illustrated in figure 4.8. The covert deceptive communication channel, discussed in chapter 5, is used to both limit the exposure of users' passwords and communicate the user's context during authentication, e.g. whether the user is logging in as a response to an email solicitation. Ersatzpasswords, presented in chapter 6, are deployed to detect password files compromise. In addition, the scheme eliminates the possibility of password cracking without physical access to the application's server. Finally, Deceptiver (discussed in chapter 7) is used to disseminate deceptive email addresses to catch any malware received as part of targeted attacks. Moreover, deceptive responses will be sent when an adversary tries to probe the web application instead of simply blocking those attempts.

Table 4.1: Using the Deception Framework to Secure Web Applications

Strategic goal	*Reducing passwords exposure*	*Detecting password files compromise*	*Limiting information leakage*
How should an adversary react	Cannot obtain the password information during authentication	Acting on the cracked passwords, an adversary would reveal himself	Act on the fake information that was obtained from interacting with the application
Exploited Biases	Expectedness	Confirmation bias	Personal Bias – servers do not lie
Simulation and Dissimulation	System Decisions	Internal Administrative Data	Public Data and System Responses
Risks and countermeasure	Deducability of real password from the users' submission	Deducability of real passwords from password files	False positives or conflicting information
Integration	Use one-time codes as authentication token carrying context	Change the way we store passwords and eliminate the possibility of passwords cracking	Augment internet-facing servers with deceptive responses
	Deceptive Channel – *Chapter 5*	**Ersatzpasswords** – *Chapter 6*	**Deceptiver** – *Chapter 7*

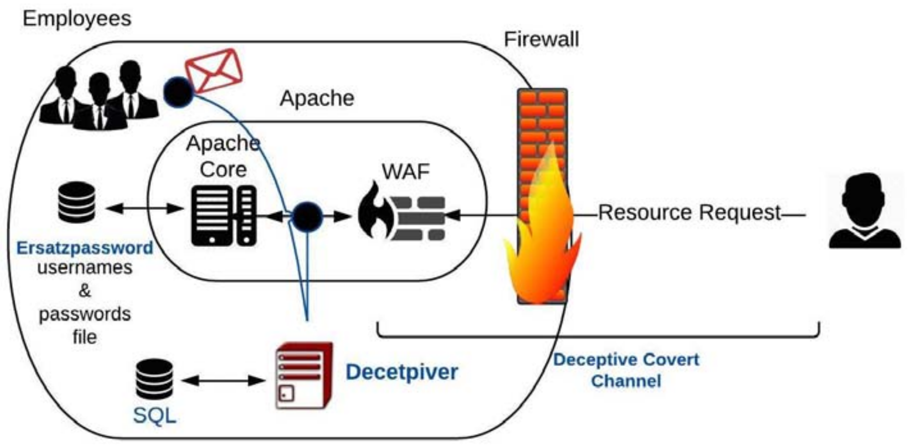


Figure 4.8.: Web Application Deception-Based Defenses

Throughout the next three chapters, we discuss the design of each one of those three deceptive defenses. We discuss the threat, or threats, the proposed defensive mechanism is designed to address. In addition, we present a security analysis of the presented solution.

4.8 Chapter Summary

In everyday security, we often use deception, and computer security is no different. In this chapter, we presented a discussion of the major limitations of previous work. After that, we examined the major advantages deception-based security tools have in comparison to traditional security mechanisms. Moreover, we presented a novel framework for planning and integrating deception into computing defenses. We discussed how a defender should monitor and evaluate the success of any such mechanisms. In addition, we provided some details of how defenders should integrate deception into their computer security defenses, and how they can create plausible alternatives to reality, thus misinforming the attackers and wasting their resources. Finally, we discussed a case study of how the framework can be used and give a brief overview of the relationships with the next three chapters.

5 DECEPTIVE COVERT CHANNEL

A recent American Banking Association (ABA) reported 62% of customers named online banking as their preferred banking method, a substantial rise from 36% in 2010 [8]. At the same time, phishing has been an increasing threat — rising at an alarming rate despite all the security mechanisms banks have in place [160]. Criminals have been stealing money by means of exploiting the ubiquity of online banking. It is estimated that the Zeus trojan alone resulted in $70 million dollars stolen from bank accounts [124]. Many of the currently deployed two factor authentication schemes by banks remain vulnerable to a number of attacks [97]. Zeus managed to bypass two factor authentication schemes employed by banks [124]. Adham et al. presented a prototype of a browser add-on that, even with two factor authentication, can successfully manipulate banking transactions on-the-fly [2]. There is clearly a need to improve the currently deployed schemes and address their shortcomings.

In this chapter we show how deception can be used to enhance the security of passwords and authentication protocols. We introduce a deceptive covert channel that conveys security information to the server, limits the exposure of users' passwords, and reduces the probability of them falling for phishing attacks. We start the chapter by presenting an overview of the problem we are trying to solve and discussing some of the relevant work in this area. After that, we present the details of our scheme with an examination of its security. We then compare our scheme to previous proposals that attempt to address the problem. We conclude the chapter by discussing some possible enhancements of our scheme.

5.1 Background

5.1.1 Authentication Schemes

In this chapter we are concerned with two general classifications of attacks against client-server communication: man-in-the-middle (MitM) attacks and man-in-the-browser (MitB) attacks, as depicted in figure 5.1. In the former attack, the adversary places herself in the communication channel between the user's computer and the server. End-to-end encryption schemes, such as SSL/TLS and IPSec, are intended to address this so that the adversary cannot observe or alter the data in the communication channel. Attackers overcome this protection by forcing the user to have an end-to-end encrypted channel with them instead of the real server, which is the case in phishing attacks. In the latter attack, MitB, the attacker places herself between the user and his computer by altering the interface (browser) and manipulates the information displayed to the user in real-time. In this case even if the user employs an end-to-end encryption scheme, such as SSL/TLS, the attacker accesses the information when it is decrypted and can actively modify it before it is shown to the user.

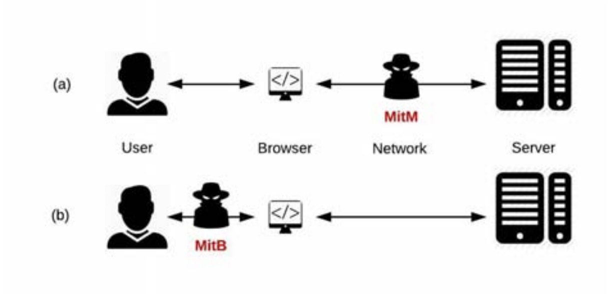


Figure 5.1.: Man-in-the-Middle (MitM) vs. Man-in-the-Browser (MitB)

Adham et al. identified three main authentication schemes built on the traditional username and password in the area of online banking [2]. These schemes are one-time password (OTP), partial transaction authentication, and full transaction authentication. They have shown that OTP schemes such as HMAC-Based One-time

Password (HOTP) [103] or Time-based One-time Password (TOTP) [104] are not secure against active man-in-the-middle attacks (MitM) or man-in-the-browser (MitB) attacks [2]. The former can be orchestrated using an active phishing attack, in which the adversary immediately uses the stolen credentials to impersonate the user to the bank, while the latter can be seen, as an example, in the Zeus trojan [15].

To address the problem of active MitB attacks, banks started to use transaction authentication [2,49]. The Chip Authentication Program (CAP) introduced by many banks requires a piece of dedicated hardware, and its protocol has a number of vulnerabilities [49]. A number of these hardware devices degrade the full transaction authentication to only part of the transaction, as a consequence of usability challenges [2]. CrontoSign [50] is a full-transaction authentication scheme that utilizes a smartphone to verify the information. The scheme requires a new phone registration process that stores information on the phone, which makes the user vulnerable if her phone is compromised or stolen. In addition, it ties the user to a specific phone, hindering the usability of the scheme if the user does not have this particular phone at transaction time. Moreover, this scheme only deals with transaction authentication, and does not focus on providing enhanced user authentication.

Full transaction authentication gives a bank the ability to ask the user to confirm her banking transaction to detect if MitB attacks are taking place and modifying the transaction *on-the-fly*. It is an essential step to enhance the security of online banking, as pointed out by Adham et al. [2]. The scheme we present in this chapter achieves such goals without the need for additional hardware, as in CAP [49] or hPIN/hTAN [93], or for a long term secret stored in the user's smartphone. It also has the other features mentioned earlier, of covertly conveying information to the bank and supporting deceiving the adversary (honeyaccounts).

5.1.2 Use of Smartphones

Clarke et al. were the first to suggest the use of a camera-based device when connecting from untrusted computers [32]. While they did not explicitly discuss the use of QR codes, their paper is considered seminal in this approach of enhancing authentication. A number of follow-on proposals presented other camera-based schemes, using smartphones and other devices to improve authentication (see, e.g., [72,91,92,95,105,145]).

Each one of these schemes suffers from one or more of the following shortcomings: (i) requiring an extra piece of hardware; (ii) storage of long-term secret on the smartphone; (iii) requiring a new registration process for associating the user's bank account with a particular smartphone; (iv) requiring the smartphone to have (network or cellular) connectivity to carry out the authentication process. The scheme we present in this chapter does not suffer from any of these shortcomings.

5.1.3 Use of Deception and Covert Channels

As we discussed in chapter 4 the use of deception has shown a number of promising results in aiding computer defenses. We incorporate deceptive elements in our scheme in two ways: (i) an active MitM will be deceived such that it is forwarding the covert messages back-and-forth that send an alarm to the service provider, (ii) we introduce honeyaccounts in our scheme to dismantle an attack before it takes place, and to gather information about the attacker's goals, objectives, and resources.

The *covert channel* term was introduced by Lampson in 1973 and defined as *"channels not intended for information transfer at all"* [89]. Such a channel has been extensively studied as a security vulnerability that undermines the security of a system and leaks out private information. The covert channel we are introducing in this scheme is observed to "not carry information" by the adversary and is created by design to *enhance* the overall security of the system. In this work we are overloading the term, although we see the functionality as similar.

Our method introduces the use of covert deceptive messages between the user and/or her client and the service provider. One of the choices of covert message is that the user is logging in as a response to an email; we discuss how this can be achieved in the next section. If the bank has no record of a recent communication, that response may trigger an enhanced defense, such as enabling read-only access. This would directly address many forms of phishing. Other messages can be automatically embedded by the user's client, such as the use of a public network.

Honeyaccounts are fake bank accounts that banks can use to lure attackers and deceive them into believing that they have successfully broken into the user's account at the bank. They provide an effective mechanism to monitor attackers' activities – to learn who is targeting a certain bank, and learn the other accounts being used to launder users' stolen funds. This information is usually gathered by banks during the forensic investigations following a money-theft episode (when it is too late to follow the money trail). A user who covertly conveys to the bank her belief in the present transaction offers some hope of dismantling the financial infrastructure of a large-scale phishing campaign before it does real damage. We all experience situations where we *know* that an email is a phishing attempt, yet many of us limit our reaction to not falling prey to it — it would be nice to have an easy-to-use mechanism for conveying our belief and thereby triggering the deception mechanisms of the bank. The covert communication we propose can achieve this.

5.2 Creating a Deceptive Covert Channel

This section discusses the technical specifications of our scheme. We show how to perform the initial setup at the server and seamlessly enroll users. We also discuss how the covert channel can be deployed within the authentication scheme. At the end of this section, we discuss some the potential enhancements that our scheme brings that can be incorporated in future work.

5.2.1 Threat Model

There are many attacks against password-based authentication systems including the following common attacks.

- *Stolen Passwords.* The security of password-based authentication systems fundamentally relies on the fact that each user's password is only known to the user alone. When an adversary obtains the user's password he has the ability to *continuously* impersonate the user to the server, without any of the two parties noticing. Many attacks, such as phishing, keylogging, and shoulder-surfing are centered on the goal of obtaining users' passwords to gain unbounded access to their accounts.

- *Stolen Password Hashes File.* An adversary who obtains the passwords hashes file of many users can apply an offline cracking process (such as dictionary attacks) to retrieve the users' passwords from their hashes.

- *Poor/Easily Guessable Passwords.* When the user chooses an easily guessable password, an adversary can easily guess it and impersonate the user to the server.

- *Repeated Password Use.* A person may use the same passwords across multiple systems where a compromise against one system undermines the security of all other systems.

Our focus in this chapter is to address the first attack scenario. In addition, it provides an improvement to address the problem of cracking passwords. However, the Ersatzpassword scheme presented in chapter 6 provides a strong protection of password cracking as we will discuss later.

5.2.2 Scheme's Setup

As depicted in figure 5.2, there is no new registration required for bank customers, and the bank can deploy the scheme either all at once, or progressively by selecting a specific subset of their customers (in which case a user who prefers the old system can easily be accommodated). In addition to a cryptographic one-way hash function **H** and a cryptographic message authentication code such as $\mathbb{HMAC}$, we use a one-way accumulator function A whose output is to have the same number of bits as **H** (so that the format of the bank server's password file does not need to be modified – only the nature of the bits stored changes).

As discussed by Fazio and Nicolosi, an accumulator function can be constructed such that it behaves as a one-way function [55]. In addition to the usual one-way property required of cryptographic hashes, a one-way accumulation of n items has the properties that (i) the order of the accumulation does not matter (i.e., any two permutations of the same n items give rise to the same result) [i.e. $A(x_1, x_2) = A(x_2, x_1)$]; and (ii) given a new item/s and the accumulation of a previous item $A(x_1)$, a new accumulation that includes the new item/s (as well as the old one) can be efficiently obtained without needing to know the previous item (x_1) which equals $A(x_1, new\_items)$. To illustrate the second property using an example, if we have the modular exponentiation of x_1 (g^{x_1}) and we want to compute the new accumulation including a new item x_2, we compute this as $g^{{x_1}^{x_2}} = g^{x_1 * x_2}$. A real world realization of such a function can be done by using a modular exponentiation where the accumulation of x_1 can be implemented as $A(x_1) = g^{x_1}$.

As the most common ways of implementing such an accumulator A function involve modular exponentiation, it is typically the case that $A(x, y) = A(x * y)$ (where arithmetic is modular). In that case the security of A hinges on the Computational Diffie-Hellman assumption; that given $A(x_1)$ and $A(x_2)$ it is computationally intractable to compute $A(x_1, x_2)$ without knowing either x_1 or x_2. We give our presentation assuming the existence of such an A, without going into any details of how it is actually implemented; our scheme depends only on A's one-way property, its above-mentioned order-independence, and its above-mentioned incremental accumulation.

Recall that a user's entry in a traditional password file contains $h = \mathbf{H}(passwd \,||\, salt)$ and *salt*, where the purpose of the salt bits is to make a wholesale dictionary attack against all users harder (but it does not make it harder to attack an individual user, because the salt is stored in-the-clear). To switch to the new system, the bank simply replaces h with $A(h)$. This can handle users who select to remain in the traditional username/password authentication (in the obvious way). But replacing h by $A(h)$ is essential for users who select to switch to our proposed smartphone-based scheme, which we describe next.

5.2.3 Logging In

As usual, the login starts with the user entering her username on the computer. We assume that the smartphone has the needed app (which knows nothing about the user or the bank).

- The bank verifies that the username exists and, if so, generates a nonce R. Then it computes and sends the following information to the user's browser, encoded in a QR-code (recall that a QR code is an optically machine-readable two-dimensional barcode).
  - $A(R)$.
  - $\mathbb{HMAC}_{key}(A(R))$ where $key = A(A(h), R) = A(h, R)$.
  - The user's salt.
- The user scans the QR code using the smartphone app and inputs his password to the smartphone. The app computes $h' = \mathbf{H}(password \,||\, salt)$ and then generates the $\mathbb{HMAC}$ key by computing $A(A(R), h') = A(R, h')$ — the user's phone does not need a copy of R to make this computation. The $\mathbb{HMAC}$ is recomputed locally and then the app verifies that the received $\mathbb{HMAC}$ matches

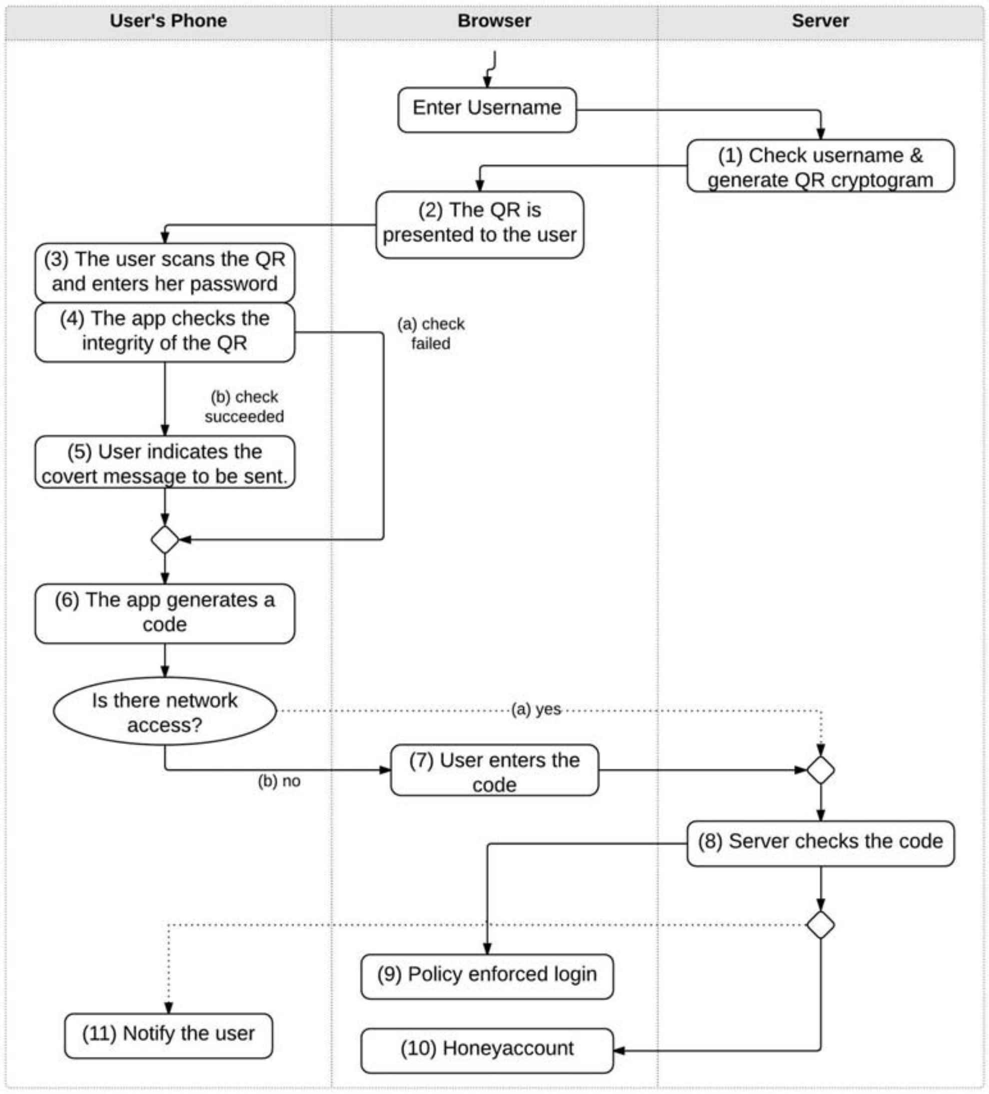

User's Phone
Browser
Server
Enter Username
(1) Check username & generate QR cryptogram
(2) The QR is presented to the user
(3) The user scans the QR and enters her password
(4) The app checks the integrity of the QR
(a) check failed
(b) check succeeded
(5) User indicates the covert message to be sent.
(6) The app generates a code
Is there network access?
(a) yes
(b) no
(7) User enters the code
(8) Server checks the code
(9) Policy enforced login
(10) Honeyaccount
(11) Notify the user

Figure 5.2.: Protocol Run

the $\mathbb{HMAC}$ it computed. If the local check succeeds (meaning the user entered the correct password and $h == h'$) the user moves into the next step of the protocol – phase 5. If the check fails there are two scenarios for what comes next; a *safe case* (branch a), and a *decoy case* (branch b). With the *safe case* the scheme continues to phase 5; in the *fail case* the scheme jumps to phase 6. Before sending the MitM/MitB to a honeyaccount, the app might ask the user to type their password three time to make sure that the failuer is not a result of a mistyped password. In the latter case, the app can simply skip the covert messaging part if it detects a MitM/MitB impersonating the bank, and either terminate or continue with a honeyaccount. In this case, the failure of the $\mathbb{HMAC}$ verification can be treated as a special kind of covert message.

- In phase (5), the user is provided with the optional facility to covertly signal a simple message to the server. This covert messaging mechanism enables different behaviors from the current practice of "all-or-nothing" authentication and access. We give users the ability to choose from a fixed set of possible messages they could convey to the server; an example can be seen in figure 5.3. Giving users the ability to convey their level of trust in the computing or network facilities being used, e.g., using a public or a friend's computer, wireless network at an airport, etc. Later in this section, we show how these messages can be easily embedded in the code generated, in phase (6) of the scheme. Users can use this same facility to covertly request a limited-access login (e.g., read-only), in cases where they are following an email-solicited invitation to login to view an "important message." This covert message can alternatively be realized by other means than the above, such as those proposed by Almeshekah et al. [3].

- In phase (6), a one-time code is generated by the smartphone by computing the following accumulation;

$$y = A(A(R), h, msg_1, .., msg_i) = A(R, h, msg_1, .., msg_i)$$

  The covert messages are conveyed by setting the bit of any *covert message* (of the i possible messages) to one.

- In phase (7), the user types the generated code into the computer (copied from the smartphone screen). To make the code readable we can use base64 encoding and selecting the first n characters (the size of n is discussed later). Branch (a)

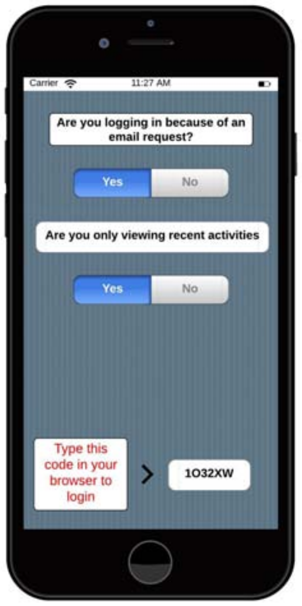

Carrier
11:27 AM
Are you logging in because of an email request?
Yes
No
Are you only viewing recent activities
Yes
No
Type this code in your browser to login
1O32XW

Figure 5.3.: Sending a Covert Message

of the previous phase, i.e. the existence of networking facility in the phone, will be discussed shortly.

- When the bank receives the code, in phase (8), it will check the validity of the code and whether a covert message has been signaled or not. It first accumulates into $A(h)$ the item R, if it matches the y sent by the user sent then the login succeeds (and the user did not convey a message), if it does not match y then the bank further accumulates (in turn) every possible covert message until the result matches y (or, if none matches, the login fails). In the *safe case*, if the bank receives a valid code with no message, phase (9) of the protocol is reached. However, if a message is sent, there are two possible options depending on the message:

  1. Take policy-specified action as per to the message conveyed before reaching phase (9). This can incorporate a variety of policies including the requirement of carrying out additional authentication measures or offer limited

access. This gives service providers the ability to implement risk-based authentication and access control, and enforce a rich set of policies.

2. Redirect the authentication session to a honeyaccount and, optionally, notifying the user of this access decision.

Length of code (y). As we will discuss below, the accumulator function is a one-way function and its output can be viewed as a random sequence of bits. As a result, the adversary succeeds if he can guess all the characters in this code. If we have 64 possible characters (including alphanumeric characters and symbols), the probability of guessing a single character is 2^{-6}. If we set the length of y to 5, the probability of guessing the code y is roughly equal to 2^{-30}.

In addition, as presented above, the calculation of y includes a random number R. As a result, the adversary gains no advantage by learning any previous runs of the protocol and the value of y as it is a one-way function of a number of variables including a random variable.

5.2.4 Creating Deceit and Covert Communication

The introduction of covert channels in our scheme gives the user and app the ability to convey a number of pre-determined messages without the knowledge of any party positioning itself at any place in the communication channels. This can be done by appending a number of bits to the input of the accumulation function in step (6). To give an example, assume the protocol is designed to signal two different messages to the server: (i) msg_1 the user is accessing from a new wireless network, (ii) msg_2 the user selected read only access. When the app computes y in step (6) it can append two bits to the hash output as the following; $y = A(A(R), h(passwd||salt)||msg_1||msg_2)$ where msg_1 and msg_2 are single bits that are set to 1 if the user want to signal this message and 0 if the message is not being signaled.

The multitude of applications that can utilize such a mechanism is large and it incorporates status communication as part of the authentication protocol. For example, the bank can take extra precautions if the user is authenticating from a new networking environment. As another example, the user can signal duress if he has been threatened and forced to transfer money to other accounts. Duress can be signaled covertly, for example, by measuring rapid changes in the phone's built-in accelerometer where the user can subtly shake his phone during login. Another example to signal duress is when the user presses the physical volume buttons during the authentication process.

5.3 Enhancements

Full-Transaction Authentication After the user logs in, the same steps can be repeated for every sensitive transaction with two main differences: (i) instead of sending the username, it is the transaction information that is sent, so that the QR code will contain additional information about the transaction details along with the $\mathbb{HMAC}$ and the user can verify those details on the app itself and make sure it is what they really want; and (ii) the covert message part can be eliminated, only keeping the part related to the failure of MAC checks. This part can be used, as we discussed before, to lure attackers who are launching MitB attacks manipulating transactions "on-the-fly."

Phone Connectivity If the smartphone happens to have (optional) network connectivity (step (a) in figure 5.2), it can spare the user the trouble of manually entering the code displayed on its screen, and send it itself to the bank's server (user sessions can be uniquely identified by the server using the nonce R).

Storage of Insensitive Information The security of our scheme does not require the long term storage of any information in the phone itself. Nevertheless, we can benefit from storing information that can increase the utility of the covert communication. As an example, the app can store the name(s) of user's home network(s) and automatically send a covert message when the user is using a non-trusted network to login. Such knowledge gives service providers the ability to deploy risk-based authentication. For example, when the user is using an untrusted network to login, limited control can be provided and an extra level of authentication can be enforced when significant transactions are required.

5.4 Security Analysis

Within our scheme when the bank sends $A(R)$, the only party that can successfully respond with y is one who knows the password and gets the smartphone to compute $h = \mathbf{H}(password||salt)$ and thus the code y that is conveyed back to the server. This is true because an adversary who gets $A(h)$ and $A(R)$ is unable to compute $y = A(h, R)$ without knowing either R or h, neither of which is available to the attacker. Also note that, if the credentials database at the bank is leaked, no one can impersonate the user without cracking the passwords, as in traditional password schemes. One minor advantage this scheme provides is that cracking is slower for the adversary because of the introduction of the accumulation function A – it is significantly slower to accumulate every password in the cracking dictionary than to simply hash it.

Central to the security of our scheme is the fact that the only information of use to an adversary (the password) is entered on the cell phone and not on the client computer being used to remotely access the bank. The cell phone has no permanent record of any sensitive information. In addition, the bank's server never contains (even ephemerally) the user's password in the clear, providing a measure of defense against a snooping insider at the bank.

Finally, we point out that there are a number of additional security advantages of entering the user's password in a smartphone application instead of the browser:

- *The use of Software Guards.* Traditional password based-schemes ask the user to enter her password in the browser running on the client operating system. Current browsers are not self-protected, as identified in [26], and they are a complex pieces of software that are exposed to many vulnerabilities. For that, our scheme uses a specific phone application that can have intrinsic software protection against tampering as illustrated in [26, 54].

- *Automated Trust Decision.* Adversaries using social-engineering attacks to lure users to give up their credentials, such as in the case of phishing, exploit the users' decision-making process by presenting them with legitimate-looking web pages. Our scheme aids users in making trust decision about the authenticity of a web page mandating that the website provides a cryptographic proof of their knowledge of a shared secret; namely the password. This process is done in total transparency to the user and the user is only asked to capture the picture of a QR code.

  This cryptographic proof can be computed by the web server without the need of explicitly storing the password value and, more importantly, without storing any information on the user's phone. This significantly increases the difficulty of social engineering attacks, such as phishing, as it reverses the game – demanding that the web site provides proof of authenticity before the user logs in.

- *Smaller Chance for Shoulder-Surfing.* Traditionally, users enter their passwords using a large keyboard where shoulder surfing is an easy task for adversaries. Asking the users to input their passwords on their phone increases the difficulty of such activity.

It worth noting that if the user logs-in to the service provider using a phone browser, our scheme cannot be directly used to scan the QR code as we discussed above. However, the basic protocol and feature can still be applicable with only a change in how the QR is input. This can be achieved by developing a browser extension that can automatically detect a QR code in the webpage and button on the corner of such codes to be clicked by users to launch the authentication app where the QR is automatically read. Nevertheless, the advantages of separating the service login, previously done on the computer, and the authentication process on the phone are slightly degraded. If the phone browser is infected with a MitB trojan, it would be easier to circumvent the security on the scheme as it can communicate directly with the authentication app. However, we note that most security sensitive transactions on a phone are done using dedicated apps that are hardened for a specific application. In addition, the underlying principle of using a covert channel presented in this chapter can be incorporated in these dedicated apps.

5.5 Comparison with Other Schemes

In table 5.1 we evaluate the different schemes using the following criteria.

Requirement of phone enrollment. Schemes such as CrontoSign and QRP [118] require the user to register her phone with the bank, i.e. phone enrollment. Such schemes store phone information, such as the IMEI number, and use it as part of their protocol to achieve assurances about the user's identity. One of the major issues of tying the user's identity to his phone is that the user may lose his phone, forget it or run out of battery power. In these circumstances, the user wants to be able to use an alternative phone to login to his account. If the user loses his phone he is vulnerable to the threat of impersonation until he reports the incident to every bank he banks with. In the case where he does not have his phone the usability of such a scheme becomes an issue as the user cannot login to his account anymore. This could result in lost business if the user moves to other banks that are supporting more usable schemes.

Our approach addresses these concerns in two ways. First, we allow customers to use many phones without degrading the security of the scheme or asking the user to register all his phones. Second, we challenge the all-or-nothing assumption allowing users to fall back to other authentication mechanisms dynamically, possibly setting the privileges to only allow non-sensitive transactions.

Requirement of long-term secrets. Many of the previously proposed schemes require the storage of long-term secret(s) either on the users' phones or on another piece of specialized hardware [93, 118, 145]. Our scheme is the first scheme that provides full transaction authentication and user authentication that resist MitB without the need to store long-term secrets or require additional hardware.

Resisting MitB. A recent paradigm in banking Trojans is to bypass two factor authentication by launching MitB attacks that change transaction information on-the-fly. We compare the schemes in table 5.1 based on their resistance to MitB. When our scheme is used to authenticate transactions, as discussed in section 5.3, a MitB attack can be defeated. This is because the MitB needs to send the modified transaction information to the bank, where an $\mathbb{HMAC}$ is created. However, when the user verifies this information on his phone after scanning the QR-code he can see that the transaction details have been changed. He can click on a button to say that the details have been changed and a *deceptive* code can be generated. The MitB attacker would end up in phase (10) where they will be deceived.

Table 5.1: Schemes Comparison

	no phone enrollment	no long-term secret	resists MitB	no special hardware	no phone connectivity	compatible with existing
Our Scheme	✓	✓	✓	✓	✓	✓
CrontoSign [50]	–	–	✓	✓	✓	–
QR-Tan [145]	–	–	✓	✓	✓	–
hPin/hTan [93]	N/A	–	✓	–	N/A	–
QRP [118]	–	–	✓	✓	✓	–

Use of special hardware. Many proposals introduce a new piece of hardware to the authentication scheme to achieve a higher level of assurance and to verify banking transactions, such as the CAP scheme [49]. There are two major disadvantages with those approaches: cost and usability. As an illustrative example, Barclay's Bank in the UK equipped users with special full-transaction authentication hardware, but ended up having to reduce the functionality to only partial transaction authentication because of many customer complaints. This degradation led to a number of security vulnerabilities [2].

Requiring phone connectivity. A number of schemes are intended to maximize their usability by making the smartphone or the special hardware act on the users' behalf. In all the mechanisms we examined this comes with the cost of either requiring the phone to have network connectivity, which is not always possible, or requiring a direct communication between the users' computers and their smartphones, which hinders usability. In our scheme we share the same goals and enhance the usability of our scheme by giving users the ability to login even though they do not have any connectivity in their phone and without having to connect their phones to their computers.

Compatible with existing infrastructure. Banks perceive security as an economic and risk reduction activity. Protocols that require radical changes to current infrastructure usually do not get adopted because of the associated high cost. In addition, the ability to dynamically fall back to traditional authentication methods is a preferred property giving banks the ability to dynamically deploy their new scheme and progressively enroll their users. This is why we use this as a comparison factor with other schemes.

5.6 Chapter Summary

In this chapter, we have shown how deception can be used to enhance the security of passwords and authentication protocols. We presented an authentication mechanism that has many attractive features, including compatibility with deployed authentication infrastructure; flexible use of smartphones without requiring phone registration or storage of permanent information in the phone; without any requirement of phone connectivity (i.e., using the phone as a computational device rather than as a storage or communication device); resistance to many common forms of attack; and a facility for user-friendly (pull-down menu on the cell phone app) covert communication from the user to the bank. The covert communication in turn makes possible different levels of access (instead of the traditional all-or-nothing), and the use of deception (honeyaccounts) that makes it possible to dismantle a large-scale attack infrastructure before it succeeds (rather than after the painful and slow forensics that follow a successful phishing attack).

6 DECEPTIVE PASSWORDS — ERSATZPASSWORDS

Passwords are the most dominant form of online authentication and likely to remain so for a while despite their weaknesses. It thus behooves us to protect them as much as possible. Within authentication servers, passwords are usually stored in a salted hashed format to prevent easy pre-image recovery. Nevertheless, an adversary who steals the list of hashed passwords can use brute-force techniques to find a password p with a hash value $H(p)$ that equals the value stored for a given user. Later, the adversary can use p to impersonate the user at the authentication server.

There are a number of threats that come with the use of passwords. These threats fall into three main categories: technical, procedural, and human related – these will be discussed in more detail in the following section. There have been a number of high-profile thefts of user passwords files in recent years. For example, Evernote reported the leakage of the hashed passwords for more than 50 million users [67]. Other attacks against Yahoo, RockYou, LinkedIn, and eHarmony has been reported [63] [167]. Furthermore, password cracking is often a precursor to more significant attacks as illustrated in [116].

In this section we show how deception can be used to protect stored passwords. We present a scheme that eliminates the possibility of any offline password cracking without physical access to the target's machine. We designed the scheme such that passwords' hashes file will appear no different than a traditional file. However, we incorporate "fake" passwords such that when an attacker uses traditional cracking tools to recover users' passwords he will "discover" these fake passwords. When such passwords are used to login to the targeted systems, they will trigger an alarm. We refer to these fake passwords as"ersatzpasswords."

6.1 Background

6.1.1 Passwords

There have been many high profile incidents involving the leaking of hashed passwords files [47]. Users are still using poor passwords, even with the existence of password policies that try to guide users towards choosing more secure passwords. This can be seen in the analysis of more than 70 million users' passwords [17]. Bonneau et al. presented an extensive comparative analysis of many authentication schemes replacing passwords [18]. However, passwords will remain in use because of their convenience, ease of use, and ease of deployment.

6.1.2 Password-Related Threats

The convenient and versatile use of passwords comes with its own challenges. We define password-related threats as the attacks adversaries can launch to retrieve one or more valid passwords of current legitimate users of the systems. These host-based[1] threats can be grouped into three main categories.

Technical Threats

There are two sub-categories of technical threats associated with the use of passwords: server-side and client-side. Any piece of malware or key logger that can be installed at the user's machine to exfiltrate the user's password is a threat to any password-based authentication system. At the server side, adversaries can obtain the file of stored password information and then impersonate the system user using the stolen passwords. Strong host security is needed to protect the client and server systems, but there are multiple opportunities for an attacker to capture a copy of the stored password information.

[1]We are ignoring network snooping and other such remote mechanisms as our attention is directed only at securing host-based password databases.

A computer system needs to save an “authenticator” for every user during user enrollment that is used to verify the identity claim during the login phase. Current computer systems store a salted cryptographic hash (H) of the password along with the username. In a system with n users, we have the following pairs:

$$(u_1, H(p_1)),\ (u_2, H(p_2)),\ \dots,(u_n, H(p_n))$$

where u_i is the username of user i and p_i is the password of user i.[2] An attacker who steals this list can launch an offline attack to recover the hashed passwords using some dictionary and replicating the hashing algorithm used. Many tools already exist to automate an attack, such as John the Ripper[3]. There have been many attempts to address this challenge, usually falling into one of three major approaches: (i) significantly increasing the resources needed to match a password, (ii) strengthening user passwords to make their recovery process unlikely as they will be unlikely to be found in a dictionary, and (iii) instrumenting passwords files with fake decoy passwords triggering an alarm when used indicating that the password file has been attacked.

The development of password hashing algorithms from crypt to bcrypt, scrypt, and others is mainly driven by the goal of increasing the resources needed to crack the users' passwords [115]. The introduction of *private* salts [86] was also intended to increase the work required for cracking the password files. In addition, increasing the number of rounds these algorithms apply to a password is a parallel approach to increasing the work factor.

Cappos and Torres proposed “PolyPasswordHasher” [23] as a scheme to protect passwords from offline dictionary attacks. Their scheme additionally protects passwords with a secret share obtained using the Shamir Secret Sharing scheme [138]. The secret is saved in memory and used to verify passwords. One of the limitations of their scheme is that it requires additional fields in the password file specifying which

[2]Salts, as an additional item in many systems, are described later.
[3]http://www.openwall.com/john/

share to use. Also, if an attacker obtains access to the system memory he can steal the secret.

Deception has been used to address the threats associated with cracking password files. One approach is to inject fake accounts with passwords into the password file. Another approach is to place decoy password files in the system luring the attackers to access them believing they are the real files. Schemes such as *Honeywords* [84] are intended to confuse the attacker by presenting him with many passwords associated with a single username, where all of them are fake except one.

Procedural Threats

Password-recovery procedures associated with password-based authentication systems are sometime exploited to override current user passwords [136].

User-Centric Threats

Threats such as phishing, shoulder-surfing, password re-use, and others can be used to undermine the security of password-based authentication systems. Our approach does not address these issues.

6.1.3 Injecting Deceit

In chapter 4 we discussed some of the unique advantages deception-based mechanisms. We use deception in the previous chapter to enhance passwords' at the clients' side and in transit. In this chapter we discuss how deception is used to enhance the security of passwords at the server side.

Rivest and Jules proposed augmenting the password database in Unix with negative information such that cracked password files can be detected [84]. Their proposal is similar to Rao's proposal of using "Failwords" [122]. Bojinov et al. proposed Kamouflage, a scheme that is intended to protect the list of passwords used by a user and saved locally by a password manager [16]. Their scheme hides the real list with a set of “fake” lists. Kontaxis et al. proposed an authentication scheme (SAuth) that requires each user’s login attempt to be vouched for by another service provider, so an attacker cannot impersonate a user by simply obtaining the password for one web site [87]. They use deception in their scheme as a way to address the common behavior of password reuse across multiple service providers.

Unlike previous proposals, our mechanism has the following advantages: (i) eliminating the requirement of any additional server/components, (ii) never presenting the real user credentials to the attackers, and (iii) making password cracking impossible without physical access to the targeted machine. The scheme runs internally in the server without requiring any changes to the user interfaces, clients, and/or experiences. A more detailed discussion of related literature is presented in the next section.

One additional contribution our scheme provides is that it imposes risks to any adversary who obtains a file of leaked usernames and passwords, causing mistrust within the market for such files, and rendering their use risky for many parties. This is because the unique property of our scheme of having the username and password file look identical to the file generated by the traditional authentication scheme. This property benefits not only the early adopters of the scheme, but the overall security of other (non-adopting) systems. This is one of the distinguishing features of using ersatzpassword in comparison to Honeywords [84], PolyPasswordHasher [23], SAuth [87], and others.

6.2 Technical Specification

6.2.1 Background

A number of cryptographic functions have been used in computer systems to protect passwords, including crypt, bcrypt, and scrypt. As discussed earlier, part of the motivation to develop additional algorithms is to make the cracking process of stolen password hashes files a resource-intensive process. Our scheme works with any of these underlying functions; we will denote the function used as **H**. In later discussion we will use bcrypt to give a concrete example, but without any loss of generality.

Throughout this section we will assume the following format of the stored password file. For each user (i) in the system we have the following triplet, at a minimum, (u_i, s_i, α_i) saved in the password file:

- Username (u_i).
- Multibyte (multi character) public salt (s_i).
- The hash of the user's password p_i as $\alpha_i = H(p_i \| s_i)$.

In addition, we will use a hardware-specific function denoted as $\mathbb{HDF}$. This can be implemented as a physically unclonable function (PUF) [147], a hardware security module (HSM) [59] with a unique key, or any other mechanism of equivalent general functionality.

Our goal is to enhance the security of the storage of passwords in three ways: (i) require the process of computing the hash of the password to require access to a hardware dependent function, thereby thwarting offline cracking of stolen password files, (ii) when an adversary attempts to crack the password file he will be presented with a fake password that can trigger an alarm at the server when used, and (iii) maintain the same appearance and format of the password file while implementing the new scheme. The final property is essential to the success of the deceptive process of injecting "fake" passwords. Unlike the Honeyword scheme, which mixes real passwords with fake ones, our scheme eliminates the ability of an adversary to obtain the real password (without physical access to the targeted machine during the cracking process) and seamlessly presents a fake password during an offline cracking process.

6.2.2 One-time Initialization

The initialization steps in our scheme are performed in two stages: *system-side* initialization and *user-specific* initialization. The former makes all the users' saved, hashed, passwords machine-dependent – applying the hardware-dependent function as follows. The hardware-dependent function $\mathbb{HDF}$ is applied to each stored password hash α_i and is then fed to the same hashing function, $\mathbf{H}$, with the original salt, s_i. After that, the output is stored in the password file replacing the old stored value. This system-wide initialization will have each user password stored in the file as the following

$$\beta_i = \mathbf{H}(\mathbb{HDF}(\alpha_i) \| s_i)$$

If an adversary obtains this file and tries to crack any user passwords, the probability that he will get any apparent match is negligible, even if a user password is from a standard dictionary. The cracking software will be searching its dictionary for a password equal to $p_i{}' = \mathbb{HDF}(\alpha_i)$ and when hashed will give β_i. An adversary with knowledge of the scheme cannot distinguish between a password file that was computed using our scheme or using the traditional scheme. Even under a stronger assumption, where the adversary knows that the file has been computed using the new scheme, the attacker gains no advantage as he cannot crack the user passwords without access to hardware used to compute the function $\mathbb{HDF}$. In the case where the attacker is an insider, any extensive use of the $\mathbb{HDF}$ can be easily noticed with a clear spike in API usage.

To incorporate the additional deceptive alarm component into our scheme — returning an "ersatzpassword" when the adversary attempts cracking the password file — we need to involve each user in a seamless fashion during any normal user authentication. This process requires the user to enter her password, which is a natural step during any authentication (because the password is not actually stored or recoverable), and can be done during the first login process after the system wide initialization.

When the user attempts the first login after the initialization of our system, the password is checked using the original hash function to see if it matches. If so, the scheme will recompute the stored password value β_i as follows. The hardware-dependent function will be applied to the actual password p_i and then an ersatz-password (p^*) will be chosen – we will discuss the use, choice, and characteristics of ersatzpassword later in this chapter. A new user-specific salt is then selected, to be used when computing the function **H**, to satisfy the following property; [${s_i}' = \mathbb{HDF}(p_i) \;\oplus\; p^*$]. The scheme will take the first 128-bits of the result, assuming we are using a function **H** such as *bcrypt* that uses 128-bits salts, as the new salt overwriting the existing salt s_i.

We note that the ersatzpassword password length can be, at maximum, as long as the salt. In the current implementation of the *bcrypt* function, widely adopted to implement the hash function **H**, the salt is 128-bits long. This gives us an ersatz-password of up to 16 characters. This does not impact the plausibility feature of the ersatzpassword, which will be discussed below. In the largest user passwords study analyzing more than 70 million real user passwords, Bonneau reports that users tend to use passwords with 6-8 characters [17]. If the ersatzpassword is shorter than the salt, the above computation will result in having the salt include some of the output of the $\mathbb{HDF}$ function. This does not affect the security of the system as such output does not leak any useful information about the real password even to someone who has knowledge of the scheme and the length of the ersatzpassword $p{*}$.

To compute the stored value β our scheme calculates the following;

$$\beta_i = \mathbf{H}[\ (\mathbb{HDF}(p_i) \oplus {s_i}') \parallel {s_i}'\ ]$$

If the output of the $\mathbb{HDF}$ is longer than the salt, we address this as follows. We divide this output into chucks of length equal to the salt length. After that, we XOR these chunks together and then XOR the result with the salt ${s_i}'$. Finally, this becomes the input to the hash function **H** along with the concatenated salt.

The stored value in the password file will be in the same format used in traditional schemes. When an adversary tries to crack the password file, he will try to find a password p_i' that when hashed using $\mathbf{H}$ will give β_i. In our scheme, we compute beta in a format equivalent to the traditional password storage where the password is p^*, i.e. $\beta = \mathbf{H}(p^* \parallel s_i')$. As a result, an attacker who is launching a dictionary attack against a stolen password file will likely find a result identifying p^* as the user password, which is the *ersatzpassword* injected in the system. When the adversary uses this password to login, an internal alarm will be triggered alerting the administrator that someone exfiltrated and attempted to crack the user password file.

6.2.3 Login

There are three main cases of login in our scheme: successful login, when the user enters the correct user/password pair; malicious login, when the adversary uses an ersatzpassword; and error login, when the username/password pair does not match anything. In this section we discuss how to evaluate the login request, in the presented order, and determine a login decision.

When the user i wants to login she presents the username and password $\bar{p}$ to the authentication server. The system identifies the username record and obtains the stored value β_i and the salt s_i associated with it. The scheme computes

$$\beta_i' = \mathbf{H}[\ (\mathbb{HDF}(\bar{p}) \oplus s_i) \parallel s_i\ ]$$

and checks whether β_i' equals β_i, and if so the user is successfully authenticated.

If the authentication fails, the scheme checks whether the password presented is the *ersatzpassword*. This is done by computing

$$\beta_i'' = \mathbf{H}[\ \bar{p} \parallel s_i\ ]$$

and checking whether this equals β_i. If they are equal, this indicates that someone is trying to impersonate the user after cracking the password file and an internal alarm is triggered.

If the two values are not equal, this can be treated as an erroneous login. The system's policy for erroneous login can then be applied.

6.2.4 Password Administration

Password Change

The user's password change requests can be treated exactly as a new password. The only difference from traditional password schemes is that our approach mandates the generation of a new salt that satisfies the property discussed above, the XOR operation between the salt and the output of applying $\mathbb{HDF}$ on the password gives an ersatzpassword.

Backup

One of the major factors that hinders the use of hardware-dependent functions is the fact that the system catastrophically fails in the rare case where the hardware associated with the $\mathbb{HDF}$ fails or is no longer available. Thus, we outline a secure backup feature that can be used to recover the system in such a failure scenario. This process utilizes public-key encryption and is initialized by generating a suitably strong public/private key pair. The private key is never used in normal operation and can be stored in a secure vault offline. It is only needed in the recovery process. The public key is used during the system wide initialization process and during the process of password change.

When the system is initialized to adopt the new authentication scheme, all the current username, password hash, and salt triplets (u_i, α_i, s_i) are encrypted using the public key and stored as a backup. In addition, whenever a user changes her password, the new value $\alpha_i{}'$ (the new hash value resulting from the new password using the traditional hash) is computed and the new triple overwrites or is appended to the backup log, along with the u_i and s_i values. As a result, the backup file with have the following list (u_i, s_i, α_i), for every user i in the system, encrypted under the public key.

If a recovery is needed after failure, the private key is fetched and used to recover the log file, which is then used to restore a traditional version of the password file. That file can be instantiated on new hardware, with a new $\mathbb{HDF}$, and users can be forced to reset their passwords — leading to transition to our new scheme as they do so.

It worth noting that decrypting the backup file using a brute-force attack should not be practical. Even if the adversary, hypothetically, manages to recover the information in the backup file the resultant password security is at least as strong as the currently deployed schemes. The cost in storage and computation to build the recovery log is minimal.

Previous Passwords Storage

It is common for many authentication server to store previously used users' passwords to prevent users from recycling them [46]. This can put users at risk when such files are compromised. Although users are not using these passwords to login, they can be used to impersonate users at other websites. If systems need to store these passwords nevertheless, our scheme provides an additional advantage over traditional methods of securely storing these passwords.

As our scheme saves the user passwords in a machine-dependent format, using the function $\mathbb{HDF}$, we can have some assurance that this password cannot be cracked offline without physical access to the target machine. Later, when attempting to store the previous passwords used in the system, we can save the passwords using the $\mathbb{HDF}$ function.

Fail-Safe Procedure

We finally point out that in addition to the backup mechanism discussed above to recover the system in the rare case of $\mathbb{HDF}$ function failure, our scheme comes with an intrinsic fail-safe procedure. In this case, we can use the traditional authentication method to check the passwords, comparing $\mathbf{H}(p_i \mid s_i)$ with the stored value β_i, where the effective user password becomes the ersatzpasswords.

6.3 ErsatzPasswords – The Use of Deception

The scheme presented in this chapter provides the guarantee that stored users passwords cannot be cracked without physical access to the hardware-dependent function ($\mathbb{HDF}$). With the increased complexity of computer systems and targeted attacks computer systems are still vulnerable to security compromise and the list of stored passwords can be stolen. In addition, the latest Verizon Data Breach Investigation Report (DBIR) shows that about 50% of attacks thwarting authentication mechanisms take months or longer to be discovered. Even worse, 88% of these attacks are discovered by external parties. Integrating deceptive passwords in the design of our scheme addresses these two issues.

When attackers obtain the stolen credentials and apply the cracking process, we can design our scheme to negatively respond to this activity as in [41]. This allows an attacker, who obtains this file, to notice such behavior and simply look for other vulnerabilities to exploit. Instead, the scheme is designed to present an attacker with plausible deceptive passwords leading him to believe that he successfully cracked the password file. When a login is attempted using the deceptive passwords, system defenders will be immediately alerted to two facts: (i) that the login credentials database was leaked; and (ii) that an attacker is currently trying to impersonate the system's users to gain access. This design enables system defenders to use internal controls for detecting credentials' database leakages, and for alerting them of an ongoing attack before it succeeds.

6.3.1 ErsatzPasswords Generation

The process of generating an ersatzpassword for each user account can be formalized as follows. Let $Gen(p_i)$ be the function that takes the user's password and outputs the selected ersatzpassword. This function should provide two essential properties: plausibility and non-deducibility. The former ensures that an ersatzpassword generated by $Gen()$ is plausible to an adversary as a real user password. The latter provides the guarantee that even when an adversary knows the scheme, he cannot deduce any information from the ersatzpassword about the real user password. We define these two properties more formally later in the chapter. We want this function to be randomized and to give us an ersatzpassword every time we use it. Of course, the generated ersatzpassword should have the properties discussed later in this chapter. We present below several constructions of how to realize this function and discuss the advantages and disadvantages of each construction.

Total Password Replacement

When $Gen()$ receives the user's password it can generate the ersatzpassword using the following procedure. For every character in the user password, replace it with a randomly chosen character from the same category (alphabetical with alphabetical, a digit with a digit, and a special character with a special character). After this replacement process, a cyclic shift is applied to the password by a random number of positions to generate the ersatzpassword.

We note that this process reveals two properties of the real password to an adversary when he views the ersatzpassword: the password's length and its character composition. In this case adversaries can use probabilistic context-free replacement to significantly narrow down the space of possible user passwords using knowledge of the ersatzpassword [168]. One of the potential ways to overcome this is to randomly truncate or append some random characters to generate the ersatzpassword.

List-Based

One of the most straightforward ways of generating the ersatzpassword using $Gen()$ is to randomly choose a word from an internal dictionary of candidates. This realization of $Gen()$ has two major limitations: the generation of ersatzpassword is not influenced by user-specific information and the existence of such a list in the system can affect the stealthiness of the deceptive component (the existence of the list is a sign that such a scheme is currently being used by the system). The former limitation is not as significant because the attacker never sees the "real" users' passwords. The advantage of using such method is the ability to have a high degree of plausibility of the ersatzpasswords. We can compile a list of the some of the previously leaked passwords used by real users and use them as our ersatzpasswords.

Grammar-Based Methods

Bojinov et al. propose a new method of generating plausible user passwords in [16] extending the work of Ross et al. in [126] and Weir in [168]. Their method is similar to our *total password replacement* method, however they tokenize the password representing distinct syntactic elements. For example, the password *"wtyy234ou*"* has the following token sequence $W_1 = \{wtyy\} \,|\, D_2 = \{234\} \,|\, W_3 = \{ou\} \,|\, S_4 = \{*\} \,|\,$. When generating the ersatzpassword, each token will be replaced with another token, of the same length, from a dictionary.

The main drawback of this method is that it leaks the type, number, and length of tokens of the original password. We address this concern by enhancing their implementation of $Gen()$ as follows. After tokenizing the password, we perform the following:

- We can randomly append or delete k tokens. For example, let say we add token S_5 to the above password.

- After that, we can randomly shuffle the order of these tokens. In the above example, the shuffle can give us the following order $W_3 \mid S_5 \mid D_2 \mid S_4 \mid W_1$.

- Finally, we randomly choose a word from a dictionary that matches each token. The chosen token can have length that is different from the original token. In our example, let's say that $W_3 =$ "abc", $S_5 =$ "!", $D_2 =$ "10", $S_4 =$ "$+$" and $W_1 =$ "$test$".

Using the grammar-based method with our modification can generate the following ersatzpassword *"abc!10+test."*

Using User Input

Our discussion so far assumes that the scheme can work without any interaction with the users. However, we note that ersatzpassword can be constructed with implicit or explicit user input. Many authentication servers save previously used user passwords in the system preventing users from recycling their old password when their current password expires. This implicit user input, previously chosen user passwords, can be used as the ersatzpassword for this user account. With explicit user input, the system can prompt the user to enter another password during registration and use this as the ersatzpassword password.

The main advantage of using implicit user input is ensuring a high degree of plausibly, discussed later, of the ersatzpassword as this has been previously used as a real password. However, this method suffers from two major disadvantages. First, if an adversary cracks the password file and recovers the ersatzpassword, this might put the user in danger as users are known to reuse passwords across multiple sites [44]. Second, this has the potential of signaling a false alarm in the case where the user forgets and uses his previous password to login.

Explicit user input requires some changes to the user interfaces. More importantly, users are likely to pay less attention, choosing very guessable passwords and/or confusing the ersatzpasswords with their real ones leading to the problem of false alarms.

In addition, users may provide an additional, ersatzpassword that is closely related to their real password, e.g. by appending a number or a character to their real password to create the ersatzpassword.

A combination of several of these methods may be the best approach.

6.3.2 ErsatzPasswords Properties

Incorporating deception in this scheme actively feeds an adversary cracking a stolen password file with some ersatzpasswords chosen to trigger internal alarms when used. These passwords should have the following properties to ensure their effectiveness.

Plausibility

When an adversary is cracking a password file, these words will present themselves as a successful outcome, i.e. when hashed along with the salt they will match the stored hashed user password in the traditional way. For the effectiveness of the scheme, these need to be plausible user passwords. Plausibility in critical in this case because these passwords must appear as they have been chosen by users as their login credentials. Thus, some dictionary and generation algorithms should be present to produce plausible ersatzpasswords (so their generation is random subject to plausibility rather than in absolute terms).

We can define a plausible generator function $Gen()$ more formally by using the following game:

- The adversary views many runs of the function $p^* = Gen(p)$ along with their associated usernames, where p^* is the ersatzpassword. The adversary can choose the values of u and/or p.
- The adversary sends a username to $Gen()$.

- $Gen()$ selects a password p, either from the real user, existing user or public password files, and computes the ersatzpassword $p^* = Gen(p)$. Then $Gen()$ sends both p and p^* back without distinguishing them.
- The adversary outputs (1) if she thinks $p*$ is the ersatzpassword and (0) otherwise. The adversary wins if she distinguishes the ersatzpassword from the real password with probability Pr.

We say that $Gen()$ is a plausible function if the probability for adversarial success is one-half — an adversary cannot do better than random guessing which of the two passwords is the ersatzpassword. That is, $Pr = 1/2 + \epsilon$ where ϵ is increasingly negligible as the number of trials increases.

Typo-Resilience

When the user is typing her real password, she may make a mistake by mistyping some characters. The ersatzpassword associated with the account should have enough edit distance from the actual password to ensure that an alarm is not triggered by mistake. As the real user password is present when selecting which ersatzpassword to use, the server can easily compute an edit distance to ensure that the user does not mistype the ersatzpassword during the login process.

Non-Deducibility

It is essential for the ersatzpassword to not reveal any useful information about the real user password. Even though we do not actively give adversaries the ersatzpasswords, we store them with the same level of protection used to store current real users' passwords. We define the function $Gen()$ to provide non-deducibility using the following game:

- The adversary views many runs of the function $p^* = Gen(p)$ where she can choose the values of u and/or p (p^* is the ersatzpassword).

- The adversary chooses two passwords p_1 and p_2, and sends then to function $Gen()$.
- $Gen()$ flips a coin and computes $p^* = Gen(p_1)$ if it gets heads or $p^* = Gen(p_2)$ otherwise. p^* is then presented to the adversary.
- The adversary outputs (1) if she thinks $p^* = Gen(p_1)$ and (0) otherwise with probability Pr.

We say that $Gen()$ is a non-deducible function if the probability for adversary success is half. — the adversary cannot do better than randomly guessing which of the two passwords was used to generate p^*. That is, $Pr = 1/2 + \epsilon$ where ϵ is increasingly negligible as the number of trials increases.

Policy Adherence

It is essential that ersatzpasswords adhere to any system-wide policy of how users' password should appear. For example, some restrictions can be imposed on the length, format, and composition of user passwords. An adversary who sees any password violating the system's policy can detect that this cannot be a real password as the system would not have accepted it. In addition, some websites mandate that user password cannot be dictionary words. In these cases, using a password list as the method to generate ersatzpassword can be challenging as it is not trivial to come up with a long list satisfying each server's policy. In addition, any change to the policy would require recomputing the list. However, the use of grammar-based approaches, similar to the one illustrated above, can be much simpler as grammar can become part of the input of the generator function $Gen()$.

Crackable

Part of the plausibility aspect of our scheme is deciding whether all ersatzpasswords should be crackable or not. Generally, this should not be the case. Many current systems add more stringent requirements of password choice to high privileged users. When they become easily crackable, this might increase adversary suspicion. In addition, it would also look suspicious if all user passwords were crackable. It might be wise to use some randomly-generated ersatzpasswords within a system to enhance the scheme's plausibility.

6.4 Implementation and Analysis

In this section, we describe the implementation details of the our system. A preliminary evaluation is also presented followed by a discussion driven by the observed results.

6.4.1 Implementation Details

We implemented the our scheme by modifying the authentication mechanism in an FreeBSD operating system. The *pam_unix* Pluggable Authentication Module (PAM), which handles the user authentication process, is modified to incorporate our system. The design decision is driven by the simplicity of PAM modules as well as the preservation of expected behavior during user authentication. The effectiveness of the deception relies on the fact that the user authentication system appears no different than standard FreeBSD user authentication.

The system relies on two key components: the hardware dependent function $\mathbb{HDF}$ and the ersatzpassword generation function $Gen()$. We used the basic Yubico YubiHSM [173], a USB hardware security module, as our $\mathbb{HDF}$. Specifically, $\mathbb{HDF}$ is a HMAC-SHA1 with a fixed secret key (k) internally stored inside the HSM;

$$\mathbb{HDF}(p) := \text{HMAC-SHA1}_k(p)$$

For the ersatzpassword generation, $Gen()$, we implemented the List-Based approach described in section 6.3.1. This choice was mainly driven by the fact that we can preselect ersatzpasswords and have more accurate measurements. The code can be easily modified to choose any ersatzpassword generation algorithm. As a proof of concept, we used a list of six-character dictionary words as our ersatzpasswords from [29]. A password is selected from the dictionary of 15,788 and used as the ersatzpassword during user account initialization.

6.4.2 Analysis

We analyze two authentication processes when comparing our implementation of the new authentication scheme and the standard FreeBSD authentication. First, we compare the latency for adding a new user into the system and the latency for authenticating a valid user. Second, the storage of cryptographic hashes of the user's password must appear and behave as in a typical FreeBSD operating system. In addition to maintaining the fidelity for accurate user authentication user password hashes must also work with conventional password cracking tools such as John the Ripper [4] to ensure the plausibility of the ersatzpasswords. We conducted our analysis on a FreeBSD virtual machine with a single core clocked at 2.7 Ghz.

Password Update and Authentication Latency

To evaluate the performance of our authentication module, we compare the latency with the standard `pam_unix` module found in FreeBSD. Two measurements are considered: the latency to update an existing password and the latency to authenticate a user. The password is fixed to "password" for all experiments. Additionally, the authentication evaluation also considers the latency of using "ersatz" for the ersatzpassword. The evaluation consists of running the `pam_chauthtok` and `pam_authenticate` as found in `passwd` and `login`. Password update and authentication latencies are sampled 1000 times independently on an idle FreeBSD virtual machine.

[4]http://www.openwall.com/john/

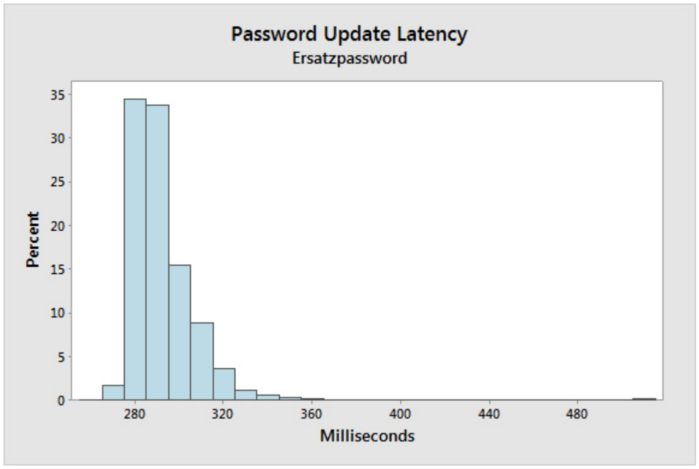


Figure 6.1.: Distribution of Password Update Latency in the Ersatzpassword Scheme

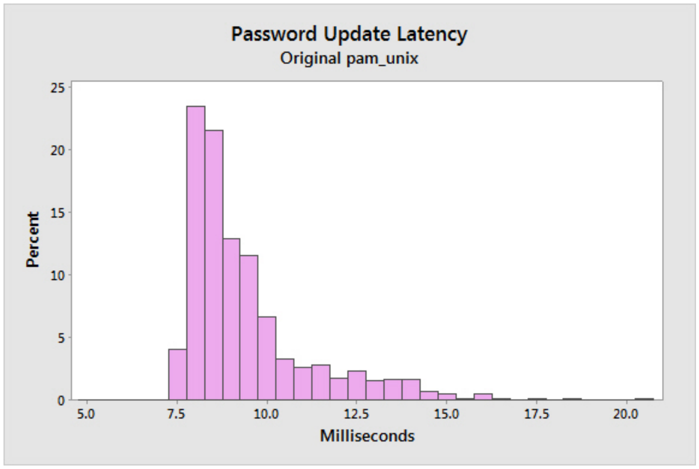


Figure 6.2.: Distribution of Password Update Latency in the Original `pam_unix`

As shown in figures 6.1 and 6.2, the median latency time to update a user's password for our ersatz system is 287.3 ms while the latency on a standard FreeBSD system is 8.8 ms. These results indicate that further optimization is needed to reduce the latency for our module to match the expected behavior of the standard FreeBSD `pam_unix` module. However, this difference is unlikely to be noticed by a user and it is a one-time cost.

A similar pattern is observed when comparing authentication latency. Figures 6.5, 6.6, and 6.7 illustrate the latencies in system response observed when providing a

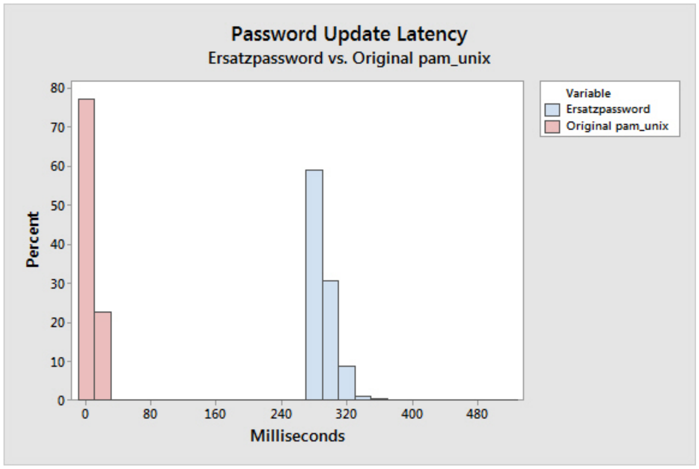


Figure 6.3.: Comparison of Password Update Latency Between Ersatzpassword and Original `pam_unix`

valid password and an ersatzpassword in our system in comparison with the latency in system response when providing a valid password in a conventional FreeBSD system. Note that the latency difference compared between our system and the conventional system are similar to the password update latency. The median system latency for authentication in our system is 277.76 ms when providing the correct password and 281.95 ms when providing the ersatzpassword, as depicted in figure 6.4. The system's latency for authenticating a valid user on a standard FreeBSD system is 5.14 ms.

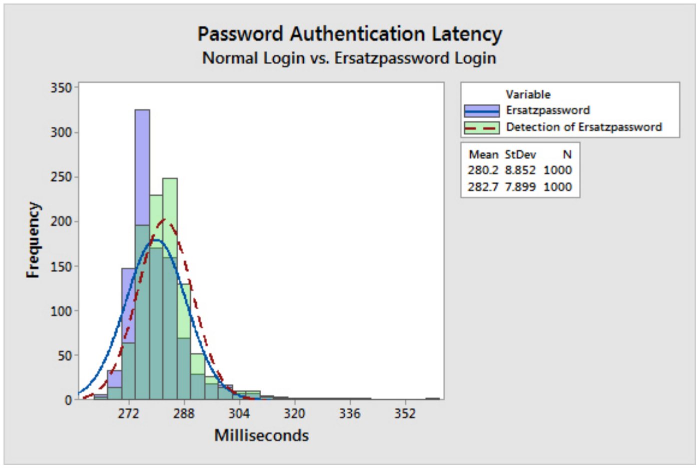


Figure 6.4.: Comparison of Real and Decoy Password Authentication in the Ersatzpassword Scheme

We note that there are a number of reasons for the observed performance difference. The YubiHSM APIs are written in python and the implementation of our scheme is written in C as a modified `pam_unix` module. A call from C to Python has an impact on the system performance. To validate our concern, we used `pmcstat`[5] to profile our modified `pam_unix` module. The results from `pmcstat` showed that the largest bottleneck is found in the `libpython2.7.so` library. Specifically, the bottleneck is `PyEval_FrameEx` which interprets and executes bytecodes from a given frame. Another bottleneck is `PyObject_Malloc` which is indirectly called when converting a C string to a Python string. Such conversion is needed in our modified `pam_unix` module when initializing the YubiHSM and generating a salt or the hash.

Table 6.1: Number of Instructions for Creating a New User Under the new `pam_unix` Module

Step	**# of instructions**	**Percentage**
Initialize	77,737,691	68.47%
Generate Ersatz	6,816	0.006%
Create Salt	273,322	0.24%
Hash Password	28,551,342	25.5%
Close	5,325,039	4.75%

To investigate other potential bottlenecks, we used `valgrind`[6] with the `callgrind` toolset to compare the number of instructions executed in the ersatz `pam_unix` module and the standard freeBSD `pam_unix` module. For each module, we looked at the amount of time it takes to enroll a new user in the system. For the ersatz `pam_unix` module, this includes initializing and closing the YubiHSM module in addition to generating an ersatzpassword, creating a salt value, and hashing. Table 6.1 contains the number of instructions for each function needed to create a new user. Note that initiating and closing the session with YubiHSM accounts for more than 70% of the instructions. In comparison to the standard freeBSD `pam_unix` module, creating a salt takes 24,171 instructions and generating a password has takes 28,202,224 instructions.

[5]`https://wiki.freebsd.org/PmcTools`
[6]`http://valgrind.org/`

Generating a salt in the ersatz *pam_unix* module takes roughly 10 times longer than in the standard freeBSD `pam_unix` module. The total number of instructions to enroll a new user in the ersatz `pam_unix` module takes about 4.96 times more than the standard freeBSD `pam_unix` module.

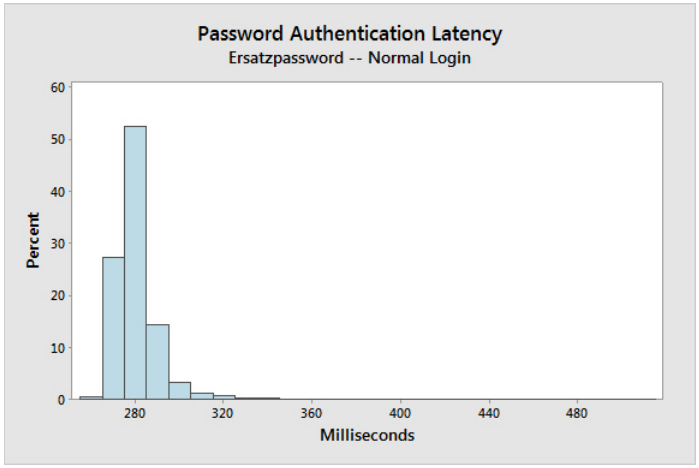


Figure 6.5.: Distribution of User Authentication Latency in the Ersatzpassword scheme

We also investigated the I/O overhead for both modules with `ktrace`, which enables kernel trace logging in freeBSD. In the ersatz `pam_unix` module, about 280ms are spent waiting for I/O, while in the `pam_unix` module only 0.008ms is spent waiting for I/O. These numbers indicate that the overhead to communicate with the YubiHSM accounts for the largest bottleneck in our ersatz `pam_unix` module.

The main reason for the performance deficiencies above is the fact that we are using a basic $\mathbb{HDF}$ function, namely the YubiHSM, which is not optimized for performance. A built-in device rather than a USB device should provide a speed improvement and reduce the I/O overhead. We believe that a combination of optimizations might bring the times close enough that it would not be obvious to an observer what might be in use on the system. If that is not a consideration, the additional latency of the current, unoptimized implementation would be clearly insignificant in normal operation.

It worth noting that the performance impact of using ersatzpasswords does not have an impact on normal users' experience. Despite the performance impact depicted

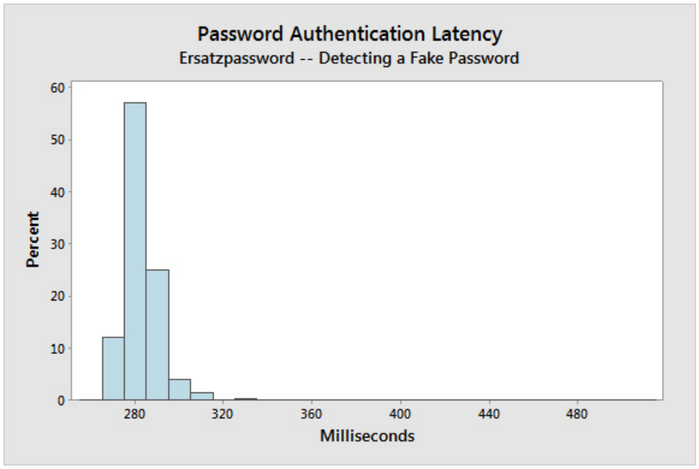


Figure 6.6.: Distribution of Detecting the Use of an Ersazpassword to Authentication

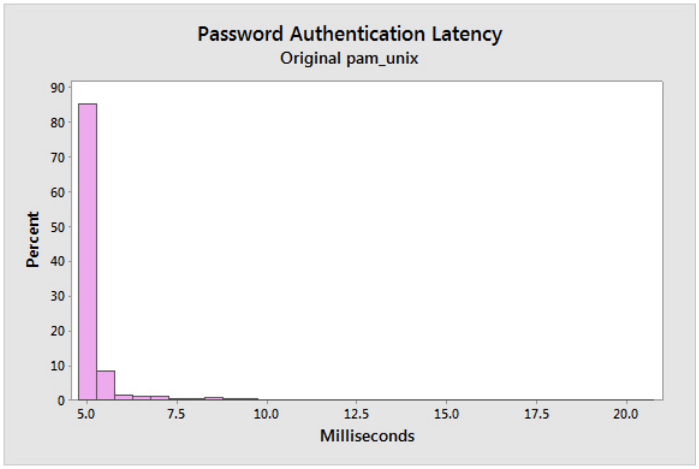


Figure 6.7.: Distribution of User Authentication Latency in the Original `pam_unix`

in figures 6.3 and 6.8, the user gets a response whether her credentials are accepted or not within a fraction of a second – this is hardly noticeable by a human user. In fact, often operating systems impose an artificial delay when the first login attempt fails using the `pam_faildelay` module.

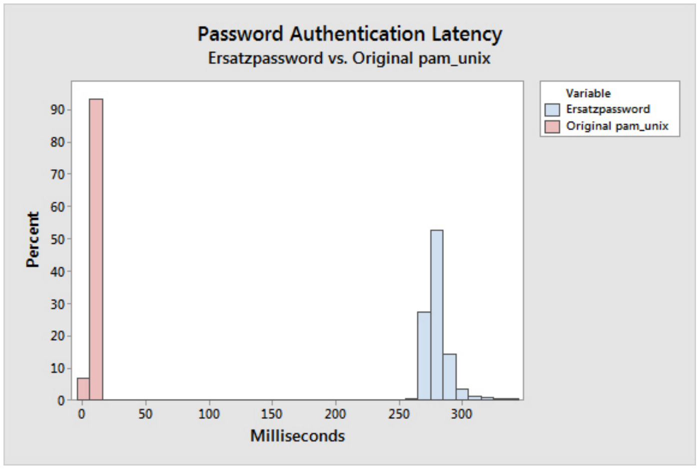


Figure 6.8.: Comparison of Password Authentication Latency Between Ersatzpassword and the Original `pam_unix`

Crackable Ersatz Hashes

To demonstrate that the our scheme produces crackable hashes, we generated 1000 hashes with the real passwords of `password1, password2, ··· , password1000` and ersatzpasswords randomly selected from our list of six-character dictionary words. We ran *John the Ripper* on all 1000 hashes created by our scheme to crack the generated hashes. John the Ripper successfully cracked all 1000 hashes and retrieved all the ersatzpasswords.

One interesting observation is that if two users select the same exact password and if the ersatzpassword selected is less than the length of the salt, then some of the bits in the salt are the same between both users. Such an anomaly would be unlikely in a conventional `master.passwd` file and may raise suspicion of the deception. This can be mitigated by properly generating ersatzpasswords to avoid such situations.

6.5 Chapter Summary

Passwords have been widely regarded as one of the weak points of securing any digital system. They come with their inherent weaknesses in how they are chosen, stored, memorized, and managed. In this chapter, we presented a scheme that address the wide-spread threat of stealing hashed password files and cracking them offline to impersonate user accounts to further infiltrate computer systems. Our scheme makes it impossible for an adversary to recover user passwords from their hashed format without physical access to the targeted machine. We show how we can instantaneously protect any system with the involvement of its user. Furthermore, we discussed how we can deceive an attacker who steals the hashed users' password file by presenting him with ersatzpasswords that work as "decoy" passwords that trigger an alarm when used to access the system. We discussed how to generate these passwords and their properties. Finally, we implemented our scheme discussing the design decisions and the performance analysis. Our goal is with the deployment of our scheme, we can end the possibility of cracking user passwords and, at the same time, detect any exflitration and cracking attempt on users' hashed password file.

7 DECEPTIVER — A CENTRALIZED DECEPTIVE SERVER

Reconnaissance is a cornerstone step for any successful cyber attack [80]. Current systems are designed to respond truthfully to all request coming to public facing network services. If the requester is identified to be malicious, our systems are also designed to respond *truthfully* by responding with an error message or dropping the request. Thus, these systems are designed to aid computer attacker through their reconnaissance step where they gather information about our systems and determine all the information our systems know about them.

The latest Verizon Data Breach Investigation Report (DBIR) identified web application attacks as the most common incident in 2013 accounting for 35% of all incidents [159]. Moreover, more than 90% of those attacks are being discovered by external parties, making the matter worse for breached organizations. The report also shows that more than 60% of web application attacks take minutes or less to compromise the target, while more than 40% of these attacks take months to be discovered. In addition, Gartner states that more than 70% of threats are at the web application layer [48].

In this chapter we present a deceptive system, referred to as *Deceptiver*, that gives public facing servers the ability to respond with deceptive responses. Deceptiver maintains a consistent deceptive story across all public-facing servers connected to it. It works as a centralized deceptive server behind all these processes. Each public-facing server, e.g. WWW servers and FTP servers, connect to the Deceptiver server through a server-specific hook that augments their responses with deceit.

7.1 Background

Most organizations use a number of internet-facing services to connect with their customers and provide their services. By design, these services respond to all requests unless the request's identifier(s), e.g. IP address, are specifically blacklisted, in which case usually the connection is silently dropped. However, internet-facing servers are continuously being targeted as one of the entry points of compromise. Six of the OWASP's top 10 security risks are cause by the behavior of computer systems of always responding faithfully to all requests [112].

Moreover, spiders scan websites collecting data that can be used as part of the reconnaissance stage. Spammers harvest users' emails and contact information to target their inboxes with spam and/or fraud [119]. More advanced adversaries use this information to craft a targeted attack such as spear phishing or socially engineering their targets. Often this information is readily available from organizations' public web pages. Targeted attacks, such as spear phishing, are four times more likely to be successful – according to [113] – and their success is a result of the public information we make readily available in public pages.

Basic deception techniques have been used to address some of the challenges imposed by these risks. Honeybots have been used to monitor common attack vectors and adding the origin of those attacks to a common blacklist database [119]. In addition, Anagnostakis et al. used "shadow honeypots" where they dynamically direct suspected server's traffic [9]. The shadow honeypot mimics the internal state of the real production system. This loose coupling between the shadow honeypot and production systems poses a number of challenges though. In the red-teaming experiment by Heckman and her team, discussed in section 4.3, the red-team was able to detect that they were interacting with the fake instance even with the use of the advanced "Blackjack" tool. Another deceptive tool called Web Labyrinth was developed to create a web structure to entrap and exhaust a web scanner [81].

There are a number of limitations to the previous proposals. They are mainly passive systems and only activated when the adversary touches some resources they are not supposed to. This is mainly the case with the Web Labyrinth tool. In addition, the loose coupling between fake and real systems and the challenge of presenting a plausible version of the truth hinders the effectiveness of using deception – as discussed in the MITRE experiment discussed in section 4.3.

7.2 Overview

The main idea behind the Deceptiver is to apply deception at a higher logical view of the protected systems. Traditional similar uses of deception, such as honeypots, use deception by creating a fake image of production systems and trying to lure adversaries to them. Such systems come at a high cost of maintenance in three ways. First, to ensure the plausibility of these deceptive techniques, the fake image needs to continuously reflect all the changes made in the real system. Second, fake systems are yet another set of systems that need to be administered and updated. It is vital that these systems are not used as an entry point to real systems because one might pay less attention to their maintenance. Third, when injecting deceit the fake system resources need to be individually changed.

Unlike the previous uses, Deceptiver applies deception to the resource either (i) by responding with a new deceptive response which is created on-the-fly, or (ii) by modifying systems resources on-the-fly before sending them to the adversary. In other words, Deceptiver provides a deceptive "view" of the system resources in a consistent manner. There is no need to create a duplicate of production systems to respond to attackers with deceptive responses.

There are two categories of the deceptive responses created by Deceptiver. The first category aids computer defenders whether the current requester is malicious or not. In this category, deceit is injected in a way that does not change the resource. Instead, deceit is injected to create "traps" that only users with malicious intent will fall for. We discuss these further in section 7.3.1. The second category of responses assume that the request has been determined as malicious and deceit is injected to lead attackers astray, confuse them, attribute them, and/or waste their time and resources. The request is determined to be malicious in one of two ways: (i) the computer system determines that the request is malicious, e.g. blacklists, firewall etc., or (ii) an adversary has fallen for the Deceptiver traps.

7.3 Deceptiver Design

Deceptiver works as a server in a computer system providing centralized decisions on how to augment internet-facing servers with deception. There are two main components in the design: *Deceptiver core* and *Deceptiver hooks*. The core makes the decision on how to respond to a particular request. It also provides a full history of all the requests that are received by an Internet-facing server and whether deception has been applied or not. The hooks translate the Deceptiver decision into application-specific actions that can be realized by the servers. The overall design is depicted in figure 7.1. The logical flow of requests goes through the following steps:

1. Requests arrive at a client server where the request meta-data, such as headers, is parsed.
2. Deceptiver's hooks intercept all the variables and passes them to the core where a decision is made on how to respond. There are four general outcomes:
   (a) Request is **blocked** and standard responses are returned.
   (b) Request is **blocked** and deceptive responses are returned.
   (c) Request is **allowed** to go through as usual.
   (d) Request is **allowed** to go through. However, before the response is returned, deceit is injected in the response.

As we discussed earlier, there are two general types of deceit that we can inject to responses: creating traps to identify adversaries, or responding with fake information

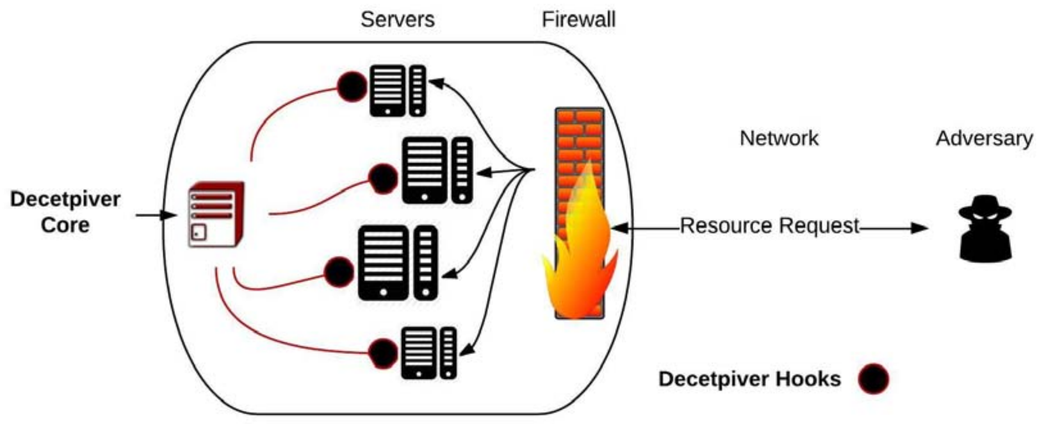


Figure 7.1.: Deceptiver's Overall Design

to give a plausible fake view of the system. We discuss these in more details in the next section.

7.3.1 Deceptive Responses

Every time a request arrives at a server the Deceptiver core needs to make a decision whether to inject deceit into the response. As discussed earlier, deceit can be used as a mechanism to identify adversaries or as a mechanism to confuse them or lead them astray. The former case is usually non-intrusive and does not affect normal user interactions. Subtle deceitful data points are added to the response as traps to detect adversaries. The latter category of responses is used to present a full or partial fake view of the resource requested.

Most of the discussion will focus on the Deceptiver core and an example of one of its hooks, namely the WWW server hook that we developed for Apache servers. We might discuss some application specific example, but the general concept behind them would still apply to any server.

It worth noting that the fake resources discussed in this chapter do not need to physically exist in the production server. The Decetpiver Hooks can create those on-the-fly and add them or append them to the real system responses. In section, 7.4 we discuss some of the possible implementations and designs of how this can be achieved.

Deceptiver Traps

At the reconnaissance phase, adversaries usually interact with targeted servers to find vulnerabilities and gauge all the relevant information. This is usually achieved with an iterative process where an adversary alters the request to the server and observes the response. Typical systems guide adversaries throughout this response by responding truthfully to all requests. Deceptiver traps embed enticing, yet plausible, fake variables that have no real function at the server. When such variables are altered, manually by an adversary or automatically by automated tools, security administrators are pro-actively alerted of the existence of an attack at its early stages. Instead of simply blocking the malicious users, the second type of response, namely *Active Responses*, are injected to confuse attackers and/or lead them astray.

Traps can be categorized into the following categories:

- **Administrative Resources** — Servers use a number of administrative files that are used by administrators to configure the behavior of server and/or the set of rules applied to responses. Example of those files in WWW servers are the `.htaccess` and the `robot.txt` files. Deceit can be injecting in these files by adding some rules that only adversaries would fall for. As an example, in the `.htaccess` file, a fake password protected web page can be added as illustrated in figure 7.2.

```
AuthType Basic
<Files "admin-access.html">
   Require valid-user
</Files>
```

Figure 7.2.: An Example of a Deceptiver Trap in `.htaccess`

When the page `admin-access.html` is not linked to from any other page normal users should not normally reach this page. Only malicious users who are looking for holes or vulnerabilities in the server will attempt to access this page. When access is attempted, Deceptiver can flag the user and *active deceptive responses* can be used to feed the user fake information or obtain more information from them.

As another example, Deceptiver can intercept all requests to access files in FTP servers. When a malicious user asks for the `etc/passwd` file, the server can respond back with a fake list of users. When someone tries to connect to accounts associated with these users or browse their files, we can flag this user as malicious.

- **Isolated Resources** — Servers can inject a number of isolated resources that should not be accessed by normal users during their normal operations. As an example, in WWW servers these can be a number of web pages that are *not linked* with any other pages. Normal users would not be able to reach those web pages by following links on the website.

- **Response Meta or Hidden Data** — Server responses send meta data. Deceptiver can inject some enticing, yet useless meta-data to the server response and monitor how the user interacts with them. Normal user interactions are not affected by such information. However, when such data is altered or used, Deceptiver can flag such users.

  For example, Deceptiver can append a number of URL variables that are enticing to attackers, see figure 7.3. When the user attempts to change the variable to `debug=true`, they can be flagged and an active deceptive response can be returned. Hidden data can also be used as traps. Deceptiver can inject some hidden form elements with some interesting names. When the user assigns some data to those or alters the default data, they can also be flagged.

```
https://www.example.com/payment.php?debug=false
```

Figure 7.3.: An Example of a Deceptiver Trap – Response Meta Data

Such data can persist over multiple sessions where they get stored at the client machine. For example, cookies are stored at the users' machine and should always be submitted "as is." We can include some enticing, yet unusable, variables in those cookies, such as `admin=false`, or create new cookies. When these are altered or partially submitted the user can be flagged.

- **Known Vulnerabilities or Violation of Policies** — Often an attack uses a preexisting vulnerability to gain some access or additional useful information to launch subsequent attacks. In addition, malicious users usually try to cross security boundaries to gain access to sensitive information. Deceptiver can be used to design traps using known vulnerabilities, applying deception, when those boundaries are crossed.

  For example, Deceptiver can take advantage of a vulnerability such as Heartbleed [52] to its advantage. A group of researchers at University of Texas at Dallas created decoy software that fixes the Heartbleed vulnerability but at the same time sent some fake responses as though the Heartbleed vulnerability is still unfixed [69]. This functionality can be incorporated in the design of Deceptiver.

  As another example, in FTP servers we can set up a fake view of files when an adversary attempts to access a file to which they do not have permission. This can be configured to problematically present the fake view to ensure its plausibility.

Active Deceptive Responses

Most servers rely on a blacklist of known malicious users to deny attempts to access their resources. However, it is becoming increasingly easier for adversaries to change their attacks to evade such simple detection mechanisms. Deceptiver is designed to utilize these blacklists, in addition to the use of its traps, discussed above, and enhance the security of these servers by sending deceptive responses to those adversaries to confuse them and/or lead them astray. Moreover, Deceptiver gives security administrators the ability to use deception in the case where they are not certain that the user is malicious or not.

Traditional security measures often treat most requests as benign unless they are part of a firewall blacklist. Deceptiver distinguishes between three main categories of users and treats them differently, namely *benign*, *suspected*, and *malicious*. In general, Deceptiver traps help security administrators to distinguish benign users from the others. If users interact with a trap, they transition into the *suspected* category, see figure 7.4. Then, Deceptiver sends them some active deceptive responses, as we will discuss in this section, and when adversaries respond to that they transition into the malicious category of users.

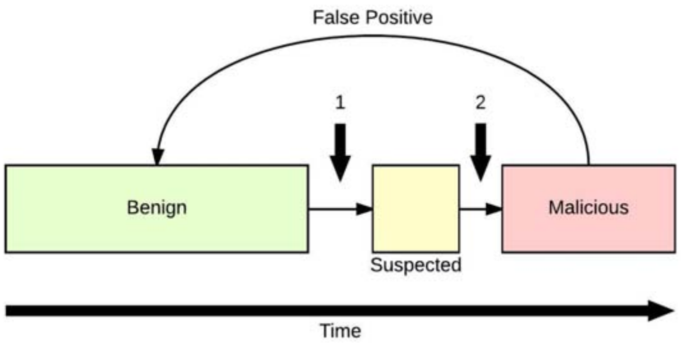


Figure 7.4.: Users Categories in Deceptiver

In section 4.5.1, we discussed how deception can be applied to different parts of a computer systems. The categories we illustrated in figure 4.2 also applies to how can we apply deception to servers' responses. Below we discuss how deception can be applied to a number of these categories.

System Performance When Deceptiver detects a malicious interaction with connected servers, it can apply deception to provide a fake view of servers' performance. One way to respond is to present adversaries a fake error message that the server is down and currently under maintenance. This is an improvement over what current firewalls do, where they either block the attack or drop the requests silently, in two ways. An opportunistic adversary will pass over this server and go to the next server in their list. The other advantage is that such a response will give security administrators more time to enhance their defenses and be proactive. Tailored honeypots can be set up and live monitoring of the threat can be achieved. Traditional security mechanisms would alert adversaries that their attack has been detected by our tools and they need to be more stealthy and change their known blacklisted artifact next time.

System Public Data One example of a malicious server request is data harvesting bots that gather organizations' public information as a precursor to their attack campaign. Later, this data is used in a wider range of attacks including phishing, spear phishing, scamming, and others. Often organizations blacklist these bots from accessing all their public information, which simply requires them to change their artifacts.

Deceptiver can apply deception to this public information by replacing organization contact details on-the-fly by plausible fake ones. Furthermore, organizations can monitor these emails to spot any advanced threats or targeted phishing attacks. In addition, monitoring those accounts can help organizations recover from false positives if this is a priority.

System Software and Services As part of the reconnaissance stage, adversaries gather as much information they can to tailor their attack to the targeted machines. Often, servers respond with detailed information of the type and version of software being used. Such behavior can be exploited be Deceptiver where we can mask the identities of the software and services we are running. Crenshaw found this to be useful to confuse attackers when obfuscating the identity of operating systems organizations' servers are running [40]. Deceptiver can also pretend to be running a number of fake services and monitor any requests that attempt to interact with them.

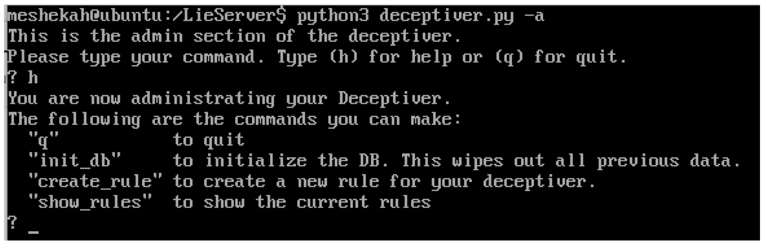
7.3.2 Centralized Deception

The main goal of the Deceptiver is to provide a centralized decision making for the use of deception. It gives computer defenders a holistic centralized view of how and when deception has been applied. It also gives an easy way to manage the use of deception within computer systems. Unlike traditional uses of deception, such as the use of honeypots, the design of Deceptiver gives security defenders a better control over the deception being used with a server and across different servers.

It is worth noting that although the Deceptiver is designed to provide *consistent* deception over many servers, it can be configured to confuse adversaries by showing inconsistent deception. As discussed in section 4.6.3, inconsistency can be helpful when our goal is to frustrate adversaries and confuse them. Also, inconsistent deception can be used as a tactic when an attacker suspects that deception is being used to feed them false information.

7.4 Implementation and Deployment

Deceptiver is implemented in Python as a command line tool. It has two main operation modes: administration and decision making. Security administrators can use the administration part to set up and configure the deception server. When the system admin launches Deceptiver with the admin option they can choose an administrative action list as depicted in figure 7.5.



```
meshekah@ubuntu:/LieServer$ python3 deceptiver.py -a
This is the admin section of the deceptiver.
Please type your command. Type (h) for help or (q) for quit.
? h
You are now administrating your Deceptiver.
The following are the commands you can make:
  "q"           to quit
  "init_db"     to initialize the DB. This wipes out all previous data.
  "create_rule" to create a new rule for your deceptiver.
  "show_rules"  to show the current rules
? _
```

Figure 7.5.: Admin Section of Deceptiver

Deceptiver uses an internal SQL database to store all the deceptive rules and all the connection information that was parsed to determine the action. The database interaction in the code has been implemented using an Object Relational Mapper (ORM) library called `peewee`[1].

Inputs and outputs to Deceptiver are standardized and are application independent. Individual server hooks prepare the request to Deceptiver and translate the response into application specific instruction. Deceptiver receives all its input as command line arguments. An explanation of each one of these inputs is presented in appendix A.1.

7.4.1 Apache Server Hook

We implemented a hook for Deceptiver with the Apache web server. We used the `mod_security` Apache module, which is used to enforce Apache's web application firewall (WAF), to hook a communication script with Deceptiver [125]. We created a Lua script that runs inside Apache and intercepts all requests coming to the server after the headers are parsed as in figure 7.6. The script prepares the request and sends it to Deceptiver. It also parses Deceptiver's response and enforces it inside the WAF. The overall design of the Apache integration is depicted in figure 7.7.

[1]http://www.peewee-orm.com/

```
SecRuleScript 'deceptiver.lua' 'id:1001,t:none,phase:2,pass,nolog,ctl:ruleRemoveById=9000'
SecAction 'id:9000,pass,nolog,phase:2,skipAfter,END_OF_RULES'
```

Figure 7.6.: Intercepting All Requests Inside Apache

Currently, Apache's hook is implemented as an internal configuration, i.e., `.conf`, file for Apache that is loaded when Apache starts. It is part of the production Apache server configuration and runs seamlessly in the background. In addition, because of the flexibility of Apache's configuration, Deceptiver can be integrated with only part of the server under a specific directory or only when a certain type or specific resource is requested.

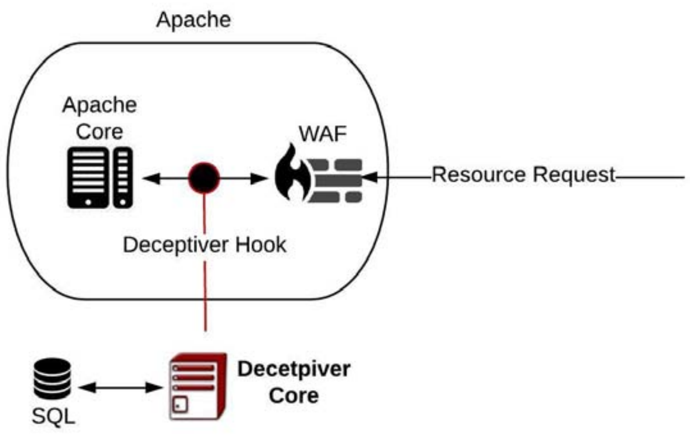


Figure 7.7.: Hooking Deceptiver with Apache

7.5 Security Discussion

Deceptiver is designed to centralize and manage the integration of deception with organization's servers. It works by intercepting server requests and responses and injecting deceit to confuse adversaries and/or lead them astray. The main advantage of using Deceptiver is that adversaries do not interact with it directly, as in the case with honeypots. It does not run a copy of resources and lure adversaries to interact with the fake system instead of the real one. This has been one of the reasons that hindered the adoption of honeypot technologies because they come with the risk that adversaries might use them as an entry point to the real organization's resources. Additionally, Deceptiver does not add another set of fake system for system administrators to maintain and keep updated and patched — unlike traditional uses of honeypots.

By design, Deceptiver does not maintain fake copies of targeted resource; instead it injects deceit to the server's response on-the-fly and, in some cases, omit real information. This design increases the plausibility of Deceptiver responses in comparison to traditional uses of deceptive response. This is true because real-time changes to system resources are immediately reflected in the adversary's view of the systems. This comes without the extra cost of maintaining fake copies of those resources or the need to update a separate fake image of the system. This design ensures that there is no time lapse and real-time changes are reflected in Deceptiver responses. This time lag needed to update the fake copies in traditional uses of deception is often used as an indicator that deception is currently being used as a defense mechanism.

In addition, Deceptiver uses *deceptive traps* to reduce the number of false positives and better distinguish between different levels of trust with respect to a system's users. These traps do not affect a normal user's behavior and should be enticing enough for some adversaries to fall for them hoping to find vulnerabilities in their targets.

It is worth pointing out that the current design of Deceptiver make it an attractive target of an attack. An adversary who compromises Deceptiver would be able to view the interactions of all connected Internet-facing servers. Even though Deceptiver is not public-facing, it receives input from many servers that is potentially injected by an adversary. It is crucial to sanitize all inputs forwarded to Deceptiver. In addition, Deceptiver should be placed in a separate process space internally with strict API that only accepts requests from known hooks. To address the allure of Deceptiver, we can change the design of the server to move the decision making to the hooks instead of the core. The role of the core would thus be simply to log interactions and connect between different servers when needed.

7.6 Performance Analysis

To analyze the performance of Deceptiver and its integration with Apache, we designed a number of experiments. We measured the performance difference between running Apache with and without Deceptiver. To do so, we used the `httperf` tool[2] to benchmark Apache's performance in these two cases. More specifically, we used `autobench`[3] that works as a wrapper over `httperf` to automate the benchmarking process. The performance results of our experiment are summarized in table 7.1. A simultaneous number of requests were sent and the average response time was recorded. The time was measured in milliseconds.

Table 7.1: Comparing the Performance of Apache With ModSecurity and Deceptiver

# of Requests	**No ModSecurity**	**Only ModSecurity**	**ModSecurity with CRS**	**ModSecurity with Deceptiver**
10	0.2	0.8	0.4	258.1
20	0.2	0.7	0.4	491
30	0.2	0.6	0.4	538.7
40	0.2	0.7	0.4	628.2
50	0.2	0.7	0.4	638.2
60	0.3	0.7	0.4	665.9
70	0.2	0.7	0.3	672
80	0.3	0.6	0.4	678.7
90	0.3	0.6	0.4	696.5
100	0.2	0.6	0.4	744.8

We noted a clear reduction in performance when using Deceptiver. To understand the cause of such performance degradation, we measured the performance of four different configurations of Apache – with respect to content filtering using Apache's

[2]`http://www.hpl.hp.com/research/linux/httperf/`

[3]`http://www.xenoclast.org/autobench/`

`mod_security` modules. The first column in table 7.1 presents the performance of running Apache without enabling the `mod_security` module. The second column shows the performance after enabling the module, with its default rules and configurations; there was a negligible reduction in performance in this case. The third column illustrates Apache's average response time while enabling `mod_security` and enforcing OWASP's ModSecurity Core Rule Set version 2.2.9 [149]. The fourth column illustrates Apache's performance when Deceptiver is in use and `mod_security` is enabled to enforce Deceptiver rules. After viewing the results, we concluded that simply enabling `mod_security` is not the cause of the performance degrade and that we need to analyze Deceptiver further to measure its performance.

We ran Unix's utility function `time`[4] with `deceptiver.py` as its input – this the main file that executes Deceptiver's logic – and we got 126ms of total execution time. To investigate this further we used the `line_profiler` python package[5] that gives us the run time for each line in the program. We found nine lines of code in `deceptiver.py` that accounts for 99.2% of the total execution time of `deceptiver.py`. All these lines are querying the Deceptiver database for rules and actions to take. In addition, Deceptiver saves the request information in the database.

Optimizing database querying is crucial to enhance the performance of Deceptiver. One way of improving the performance is to use a persistent database connection instead of having to create and tear down the database connection for every query. This can be achieved by the use of `QuerySet` API in python and optimizing the number of database queries sent. In addition, standard database optimization techniques, such as indexing, can be used to enhance the performance. Finally, Deceptiver can asynchronously save the request information in the database after responding back to Apache instead of blocking the response until the update is done.

In addition, it should be possible to eliminate the need of an SQL server to significantly reduce the performance impact when using Deceptiver. Running all Deceptiver's logic gives us an average response time of less than a millisecond – which

[4]`http://man7.org/linux/man-pages/man1/time.1.html`
[5]`https://github.com/rkern/line_profiler`

is similar to other running modes in table 7.1. Deceptiver uses an SQL server for two main reasons; (i) find whether a rule exists to apply to the current request; and (ii) save the current request information and the rules applied to it. As discussed earlier, the former can be done in a non-blocking operation. In addition, we can save such information by logging it in a file. The former goal can be achieved by having an in-memory hash data structure that maps request information into the rule that needs to be applied. This data structure can be initialized once during the start of the server and kept in memory. The operation checking whether a rule exists for the following request would be executed in a constant time and it will be a simple hashing of the request variables and checking for a match. This should be an interesting task to further extend the functionality of Deceptiver and enhance its performance.

7.7 Chapter Summary

In this chapter, we introduced a deceptive fake server referred to as Deceptiver. The server hooks into a company's servers and injects deceit creating a fake view of an organization's resources to confuse them and/or lead them astray. We discussed the design of our deceptive server and how it can be attached to many servers within an organization to provide a centralized deception.

The server provides two different categories of deceptive responses: deceptive traps and active deceptive responses. The former focuses on detecting adversaries' activities in their reconnaissance stage where they monitor servers' responses to their input. Deceptiver creates some enticing, yet non-functional, traps for attackers. The latter case of response focuses on presenting a fake, yet plausible view of systems resources.

We built a proof-of-concept implementation of Deceptiver and a hook to the Apache server. We discussed the deployment and implementation details. In addition, we present a discussion of Deceptiver performance and how it can improved. At the end of the chapter we gave a discussion of the security of our deceptive server.

8 CONCLUSIONS

8.1 Summary

In this dissertation we discussed the concept of deception and how it has been an integral part of human activity throughout history. Deception is extensively documented in animal behavior (mainly as a defensive mechanism), human physical security behavior, and in conflict and war. In computing, the discussion of using deception as a defenses started in the 1980s. We gave a broad overview of the concept of deception, highlighting its role in many areas of computing.

In addition, we presented a framework of how deception can be planned and integrated into computer security defenses. Our framework provides a holistic overview of the role of deception and how it can be applied. Within our model we present an analysis of the role of adversary biases in the success of any deceptive technique. We discuss how such biases can be exploited to enhance the security of computer systems.

Deception has been used haphazardly within computer defenses. We presented a working definition of the use of deception in security and highlighted some of the unique advantages deception-based mechanisms bring to computer defense. An overview of the proliferation of honey-prefixed tools in the early 2000s, where many deceptive techniques have been applied, is provided. Moreover, we analyzed two previous proposals to use deception as a defensive technique and mapped them against our framework.

In the second part of the dissertation, we presented three computer security defenses that use deceptive techniques in the core of their design. We show how we can enhance the security of passwords at the client side and in transit using a deceptive covert channel. After that, we discuss how the *ersatzpasswords* scheme can be used to significantly address the challenge of stolen stored password files. We discuss how we make passwords cracking insuperable without physical access to the victim's servers. Also, we show we can deceive adversaries by presenting them with fake passwords when they attempt to crack stolen password files.

We concluded the dissertation by discussing the design of a centralized deceptive server, referred to as *Deceptiver*, that employs deceptive techniques to Internet-facing servers. We show how this design reduces the information leaked about protected computer systems. We integrated our deceptive core with Apache's web server and discussed the design, implementation, and performance of our prototype.

8.2 Future Work

The use of deception has shown a number of interesting and promising results in enhancing the security of computer systems. Previous attempts to use deceit were mostly ad-hoc attempts to integrate deception into computer defense. Many successful computer attacks rely on deceiving humans to infiltrate their targets and exploit their biases. Our framework provides the first steps for a holistic view of using deception as a defensive mechanism with consideration of how deception works, where to apply it, and how it is evaluated and monitored.

One of the interesting areas for future work is a deeper understanding of how to successfully use deception. Humans, and by extension the software they write, have biases that should be understood and effectively exploited to have an effective security mechanism. Researching and understanding these biases, including cognitive, cultural, and organizational biases, is essential to the effective use of deceptive techniques.

Another interesting area of future research is exploring when and how deception can be applied. We discussed the cyber kill-chain model earlier in this dissertation and pointed out that deception can be applied at any phase in the kill chain. The original kill chain paper suggested that the earlier one stops an attacker the better. We argue that the earlier one spots an attack and then uses deceit and misinformation is better for defense. Security administrators would not only stop an attack, but also lead attackers astray, wasting their time and resources, and learn about their motives and targets. Investigating different deception-based defensive techniques that can be applied at different stages of the kill-chain is one area of future research.

Moreover, it would be interesting to investigate how different deceptive mechanisms can be used with different bad actors. It would be interesting to investigate how different actors would react to different types of deception. Another area of future research is examining the relationships between consistency and deception. Investigating the properties of using inconsistent deception is an interesting area of further research.

The area of counter-deception is another fruitful area of further research. Many of the long-standing cyber attacks such as phishing, scams, and others employ deception as a cornerstone in their design. Further understanding human biases in interacting with such attack venues would give computer security experts better ways to alter user behavior to enhance security. When security tools can identify the set of biases being exploited by adversaries, they can give users more meaningful warnings and better guide their behavior.

In this dissertation, we developed a high-level framework describing how deception can be planned and integrated. Modeling deception is an area that needs further research. Game theoretical models, such as *hypergames*, can help computer security defenders get a better understanding of the role of perception in computer attacks. Hypergames model how multi-level misperceptions determine the final outcome of a conflict [13]. Furthermore, more detailed frameworks would guide security administrators on how can they integrate deception in their defenses.

Within the three practical novel uses of deception there are a number of areas in which our work can be further developed. It would be interesting to integrate our deceptive covert channel with some two-factor authentication clients such as Google Authenticator and analyze how that increases users' security. In addition, a more rigorous user study can be done to evaluate the usability of our scheme.

Ersatzpasswords deployment can be further extended in a number of ways. In our implementation, we used a simple ersatzpasswords generation algorithm. It would be interesting to see how other generation algorithms can be used and how they compare. One especially important aspect is how to make them appear plausible when cracked by an adversary. Further analysis of publicly leaked passwords can give us insights on how the collective ersatzpasswords appear plausible. In addition, one of the issues we discussed in our implementation analysis is performance. We used a primitive HSM that plugs to a computer's USB port. It would be interesting to see how performance can be improved with dedicated HSMs or some PUFs.

We discussed aspects in the design and implementation of the Deceptiver server. There are a number of ways in which Deceptiver can be enhanced. First, Deceptiver can be integrated with more servers such as FTP servers. In addition, more deceptive responses and traps can be added to Deceptiver. Second, it would also be interesting to deploy Deceptiver and analyze real world results. This might reveal some patterns that can further help refine the design and implementation of Deceptiver. Third, we discussed how we can add inconsistent deception as a defensive technique. Investigating how inconsistent deception can be used and applied is an interesting area of further research. Finally, we identified a number of ways of improving Deceptiver's performance. Further analyzing and implementing ways to improve the performance of Deceptiver is part of the future work.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Eytan Adar, Desney S. Tan, and Jaime Teevan. Benevolent Deception in Human Computer Interaction. In *CHI 2013*, pages 1863–1872, 2013.

[2] Manal Adham, Amir Azodi, Yvo Desmedt, and Ioannis Karaolis. How to Attack Two-Factor Authentication Internet Banking. In *Financial Cryptography*, 2013.

[3] Mohammed H. Almeshekah, Mikhail J. Atallah, and Eugene H. Spafford. Back Channels Can Be Useful! - Layering Authentication Channels to Provide Covert Communication. In *Security Protocols XXI*, Lecture Notes in Computer Science, pages 189–195. Springer, 2013.

[4] Mohammed H. Almeshekah, Mikhail J. Atallah, and Eugene H. Spafford. Enhancing Passwords Security Using Deceptive Covert Communication. In Hannes Federrath and Dieter Gollmann, editors, *International Conference on ICT Systems Security and Privacy Protection (IFIP SEC'15)*, pages 159–173. Springer International Publishing, Hamburg, Germany, 2015.

[5] Mohammed H. Almeshekah and Eugene H. Spafford. Planning and Integrating Deception into Computer Security Defenses. In *New Security Paradigms Workshop (NSPW'14)*, Victoria, BC, Canada, 2014.

[6] Mohammed H. Almeshekah and Eugene H. Spafford. The Case of Using Negative (Deceiving) Information in Data Protection. In *9th International Conference on Cyber Warfare and Security ICCWS'14*. Academic Conferences and Publishing International, 2014.

[7] Mohammed H. Almeshekah and Eugene H. Spafford. Using Deceptive Information in Computer Security Defenses. *International Journal of Cyber Warfare and Terrorism (IJCWT)*, 4(3):46–58, 2014.

[8] American Bankers Association (ABA). Popularity of Online Banking Explodes. `http://www.aba.com/Press/Pages/090811ConsumerPreferencesSurvey.aspx`, September 2011.

[9] Kostas G. Anagnostakis, Stelios Sidiroglou, Periklis Akritidis, Konstantinos Xinidis, Evangelos Markatos, and Angelos D. Keromytis. Detecting Targeted Attacks Using Shadow Honeypots. In *Proceedings of the 14th USENIX Security Symposium*, 2005.

[10] Adam Barth, Collin Jackson, and John C. Mitchell. Robust Defenses for Cross-Site Request Forgery. *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*, 2008.

[11] J. Bowyer Bell and Barton Whaley. *Cheating and Deception.* Transaction Publishers New Brunswick, 1991.

[12] Michael Bennett and Edward Waltz. *Counterdeception Principles and Applications for National Security.* Artech House, 2007.

[13] Peter G. Bennett. Hypergames: Developing a Model of Conflict. *Futures*, 12(6):489–507, 1980.

[14] Maya Bercovitch, Meir Renford, Lior Hasson, Asaf Shabtai, Lior Rokach, and Yuval Elovici. HoneyGen: An Automated Honeytokens Generator. In *IEEE International Conference on Intelligence and Security Informatics (ISI'11)*, pages 131–136. IEEE, 2011.

[15] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang. On the Analysis of the Zeus Botnet Crimeware Toolkit. In *8th International Conference on Privacy, Security and Trust*, pages 31–38, 2010.

[16] Hristo Bojinov, Elie Bursztein, Xavier Boyen, and Dan Boneh. Kamouflage: Loss-Resistant Password Management. In *Proceedings of the 15th European Conference on Research in Computer Security*, pages 286–302. Springer-Verlag, 2010.

[17] Joseph Bonneau. The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords. In *IEEE Symposium on Security and Privacy*, pages 538–552, 2012.

[18] Joseph Bonneau, Cormac Herley, Paul C. Van Oorschot, and Frank Stajano. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *IEEE Symposium on Security and Privacy*, pages 553–567, 2012.

[19] Brian M. Bowen, Malek Ben Salem, Shlomo Hershkop, Angelos D. Keromytis, and Salvatore J. Stolfo. Designing Host and Network Sensors to Mitigate the Insider Threat. *IEEE Symposium on Security and Privacy*, 7(6):22–29, 2009.

[20] John Boyd. The Essence of Winning and Losing. `http://www.danford.net/boyd/essence.htm`, 1995.

[21] Bambi R. Brewer, Roberta L. Klatzky, and Yoky Matsuoka. Visual-Feedback Distortion in a Robotic Rehabilitation Environment. *Proceedings of the IEEE*, 94(9):1739–1750, 2006.

[22] Anthony C. Brown. *Bodyguard of Lies: The Extraordinary True Story Behind D-Day.* Lyons Press, 2007.

[23] Justin Cappos and Santiago Torres. PolyPasswordHasher: Protecting Passwords In The Event Of A Password File Disclosure. `http://polypasswordhasher.github.io/PolyPasswordHasher/`, 2014.

[24] Thomas E. Carroll and Daniel Grosu. A Game Theoretic Investigation of Deception in Network Security. *Security and Communication Networks*, 4(10):1162–1172, 2011.

[25] Chairman of the USA Joint Chiefs of Staff. Joint Publication 3-13.4: Military Deception, July 2006.

[26] Hoi Chang and Mikhail J. Atallah. Protecting Software Code by Guards. In *Security and privacy in Digital Rights Management*, pages 160–175. Springer, 2002.

[27] Xu Chen, Jonathon Andersen, Zhuoqing Morley Mao, Michael Bailey, and Jose Nazario. Towards an Understanding of Anti-Virtualization and Anti-Debugging Behavior in Modern Malware. In *IEEE International Conference on Dependable Systems and Networks*, pages 177–186. IEEE, 2008.

[28] Bill Cheswick. An Evening with Berferd in Which a Cracker is Lured, Endured, and Studied. In *Proceedings of Winter USENIX Conference*, San Francisco, 1992.

[29] John Chew. Common Six-Letter Words. `http://www.poslarchive.com/math/scrabble/lists/common-6.html`.

[30] David Christian and R. Michael Young. Strategic Deception in Agents. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 218–226. IEEE Computer Society, 2004.

[31] John Ciancutti. Five Lessons We Have Learned Using AWS. `http://techblog.netflix.com/2010/12/5-lessons-weve-learned-using-aws.html`, 2010.

[32] Dwaine Clarke, Blaise Gassend, Thomas Kotwal, Matt Burnside, Marten Van Dijk, Srinivas Devadas, and Ronald Rivest. The Untrusted Computer Problem and Camera-Based Authentication. In *Pervasive Computing*, pages 114–124. Springer, 2002.

[33] Fred Cohen. The Deception Toolkit. `http://www.all.net/dtk/`, 1998.

[34] Fred Cohen. A Framework for Deception. *National Security Issues in Science, Law, and Technology*, page 123, 2007.

[35] Fred Cohen and Deanna Koike. Misleading Attackers with Deception. In *Proceedings from the 5th annual IEEE SMC Information Assurance Workshop*, pages 30–37. IEEE, 2004.

[36] Fred Cohen, Irwin Marin, Jeanne Sappington, Corbin Stewart, and Eric Thomas. Red Teaming Experiments with Deception Technologies. `http://www.all.net/journal/deception/experiments/experiments.html`, 2001.

[37] Gregory Conti and E. Sobiesk. Malicious Interface Design: Exploiting the User. In *Proceedings of the 19th International Conference on World Wide Web*, pages 271–280, 2010.

[38] Terry Copeck, Sylvain Delisle, and Stan Szpakowicz. Parsing and Case Interpretation in TANKA. In *Proceedings of the 14th Conference on Computational Linguistics*, pages 1008–1012. Association for Computational Linguistics, 1992.

[39] Stephen Crane, Per Larsen, Stefan Brunthaler, and Michael Franz. Booby Trapping Software. *New Security Paradigms Workshop (NSPW'13)*, pages 95–106, 2013.

[40] A. Crenshaw. OSfuscate: Change Your Windows OS TCP/IP Fingerprint to Confuse P0f, NetworkMiner, Ettercap, Nmap and Other OS Detection Tools. `http://goo.gl/LRXMC8`.

[41] Dan Cvrcek. Hardware Scrambling – No More Password Leaks. `https://goo.gl/AR4R4f`.

[42] David Dagon, Xinzhou Qin, Guofei Gu, Wenke Lee, Julian Grizzard, John Levine, and Henry Owen. Honeystat: Local Worm Detection using Honeypots. In *Recent Advances in Intrusion Detection*, pages 39–58. Springer, 2004.

[43] Donald C. Daniel and Katherine L. Herbig. Propositions on Military Deception. *The Journal of Strategic Studies*, 1(5):155–177, 1982.

[44] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The Tangled Web of Password Reuse. In *The 2014 Network and Distributed System Security Symposium (NDSS'14)*, San Diego, CA, 2014.

[45] Fiorella De Rosis, Valeria Carofiglio, Giuseppe Grassano, and Cristiano Castelfranchi. Can Computers Deliberately Deceive? A Simulation Tool and Its Application to Turing's Imitation Game. *Computational Intelligence*, 19(3):235–263, 2003.

[46] Defense Information Systems Agency (DISA). Application Security and Development: Security Technical Implementation Guide (STIG). Technical report, Department of Defense (DOD), 2013.

[47] Matthew DeLuca and Julianne Pepitone. eBay Warns Customers to Change Passwords After Database Hacked. `http://goo.gl/npv0Cy`.

[48] Jeremy D'Hoinne, Adam Hils, Greg Young, and Joseph Feiman. Magic Quadrant for Web Application Firewalls. Technical report, Gartner, Stamford, CT, 2014.

[49] Saar Drimer, Steven J Murdoch, and Ross Anderson. Optimised to Fail: Card Readers for Online Banking. In *Financial Cryptography and Data Security*, pages 184–200. Springer, 2009.

[50] Igor Drokov, Elena Punskaya, and Emmanuel Tahar. System and method for dynamic multifactor authentication, January 27 2015. US Patent 8,943,548.

[51] James F. Dunnigan and Albert A. Nofi. *Victory and Deceit: Deception and Trickery at War*. Writers Club Press, 2001.

[52] Zakir Durumeric, James Kasten, David Adrian, J. Alex Halderman, Michael Bailey, Frank Li, Nicolas Weaver, Johanna Amann, Jethro Beekman, and Mathias Payer. The Matter of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 475–488. ACM, 2014.

[53] Tobias Eggendorfer. Combining the SMTP Tar Pit Simulator with White Listing. In *Security and Management*, pages 333–339, 2008.

[54] Paolo Falcarin, Christian Collberg, Mikhail Atallah, and Mariusz Jakubowski. Software Protection. *IEEE Software*, 28(2):24–27, 2011.

[55] Nelly Fazio and Antonio Nicolosi. Cryptographic Accumulators: Definitions, Constructions and Applications. Technical report, Computer Science Department, New York University, 2002.

[56] C. Fiedler. Secure Your Database by Building HoneyPot Architecture Using a SQL Database Firewall. `http://goo.gl/yr55Cp`.

[57] Charles Ford. *Lies! Lies!! Lies!!!: The Psychology of Deceit.* American Psychiatric Publishing, Inc., 1st edition, 1999.

[58] Charles A. Fowler and Robert F. Nesbit. Tactical Deception in Air-Land Warfare. *Journal of Electronic Defense*, 18(6):37–45, 1995.

[59] Dirk Fox. Hardware Security Module (HSM). *Datenschutz und Datensicherheit-DuD*, 33(9):564–564, 2009.

[60] Xinwen Fu. *On Traffic Analysis Attacks and Countermeasures.* PhD Dissertation, Texas A & M University, 2005.

[61] Holtjona Galanxhi and Fiona Fui-Hoon Nah. Deception in Cyberspace: A Comparison of Text-Only vs. Avatar-Supported Medium. *International Journal of Human-Computer Studies*, 65(9):770–783, 2007.

[62] Nandan Garg and Daniel Grosu. Deception in Honeynets: A Game-Theoretic Analysis. In *Information Assurance and Security Workshop*, pages 107–113. IEEE, 2007.

[63] Chris Gaylord. LinkedIn, Last.fm, Now Yahoo? Don't Ignore News of A Password Breach. `http://goo.gl/obSJla`.

[64] Scott Gerwehr and Russell W. Glenn. *The Art of Darkness: Deception and Urban Operations.* Rand Corporation, 2000.

[65] Roy Godson and James Wirtz. *Strategic Denial and Deception.* Transaction Publishers, 2002.

[66] Andy Greenberg. A Different Approach To Foiling Hackers? Let Them In, Then Lie To Them. `http://goo.gl/zuKtWx`, April 2013.

[67] Doug Gross. 50 Million Compromised in Evernote Hack. `http://goo.gl/HDCuux`, March 2013.

[68] Dominik Gus and Dietrich Dorner. Cultural Difference in Dynamic Decision-Making Strategies in a Non-lines, Time-delayed Task. *Cognitive Systems Research*, 12(3-4):365–376, 2011.

[69] Kevin Hamlen. Cybersecurity Researchers Roll Out A New Heartbleed Solution. `https://www.utdallas.edu/news/2014/4/14-29531_Cybersecurity-Researchers-Roll-Out-A-New-Heartblee_story-wide.html?WT.mc_id=NewsTwitter`, 2014.

[70] Jeffrey T. Hancock. Digital Deception. *Oxford Handbook of Internet Psychology*, pages 289–301, 2007.

[71] Michael Handel. *War, Strategy and Intelligence.* Routledge, London, UK, 1989.

[72] N. Harini and T. R. Padmanabhan. 2CAuth: A New Two Factor Authentication Scheme Using QR-Code. *International Journal of Engineering and Technology*, 5(2):1087–1094, 2013.

[73] Malcolm Harkins. A New Security Architecture to Improve Business Agility. In *Managing Risk and Information Security*, pages 87–102. Springer, 2013.

[74] Kristin Heckman. Active Cyber Network Defense with Denial and Deception. `http://goo.gl/Typwi4`, March 2013.

[75] Roger Hesketh. *Fortitude: The D-Day Deception Campaign.* Overlook Hardcover, Woodstock, NY, 2000.

[76] Kelly Jackson Higgins. How Lockheed Martin's 'Kill Chain' Stopped SecurID Attack. `http://goo.gl/r9ctmG`, 2013.

[77] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion Detection Using Sequences of System Calls. *Journal of Computer Security*, 6(3):151–180, 1998.

[78] Geert Hofstede, Gert Hofstede, and Michael Minkov. *Cultures and Organizations.* McGraw-Hill, 3rd editio edition, 2010.

[79] Thorsten Holz and Frederic Raynal. Detecting Honeypots and Other Suspicious Environments. In *Information Assurance Workshop*, pages 29–36. IEEE, 2005.

[80] Eric M. Hutchins, Michael J. Cloppert, and Rohan 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.

[81] Ben Jackson. Web Labyrinth. `https://github.com/mayhemiclabs/weblabyrinth`, 2012.

[82] Robert Jervis. *Deception and Misperception in International Politics.* Princeton University Press, 1976.

[83] Reginald Victor Jones. *Reflections on Intelligence.* William Heinemann Ltd, London, 1989.

[84] Ari Juels and Ronald L. Rivest. Honeywords: Making Password-Cracking Detectable. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, pages 145–160. ACM, 2013.

[85] Gene H. Kim and Eugene H. Spafford. Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection. Technical report, Department of Computer, Purdue University, West Lafayette, IN, 1994.

[86] Daniel V. Klein. Foiling the Cracker; A Survey of, and Improvements to Unix Password Security. In *14th DoE Computer Security Group*, May 1991.

[87] Georgios Kontaxis, Elias Athanasopoulos, Georgios Portokalidis, and Angelos D. Keromytis. SAuth: Protecting User Accounts From Password Database Leaks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, pages 187–198. ACM, 2013.

[88] Christian Kreibich and Jon Crowcroft. Honeycomb: Creating Intrusion Detection Signatures using Honeypots. *ACM SIGCOMM Computer Communication Review*, 34(1):51–56, 2004.

[89] Butler W. Lampson. A Note on the Confinement Problem. *Communications of the ACM*, 16(10):613–615, 1973.

[90] Jon Latimer. *Deception in War: Art Bluff Value Deceit Most Thrilling Episodes Cunning mil hist from The Trojan.* Penguin, 2003.

[91] Young Sil Lee, Nack Hyun Kim, Hyotaek Lim, HeungKuk Jo, and Hoon Jae Lee. Online Banking Authentication System Using Mobile-OTP With QR-code. In *International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*, pages 644–648. IEEE, 2010.

[92] Youngsook Lee, Jeeyeon Kim, Woongryul Jeon, and Dongho Won. Design of a Simple User Authentication Scheme Using QR-Code for Mobile Device. In *Information Technology Convergence, Secure and Trust Computing, and Data Management*, pages 241–247. Springer, 2012.

[93] Shujun Li, Ahmad Reza Sadeghi, Soren Heisrath, Roland Schmitz, and Junaid Jameel Ahmad. hPIN/hTAN: A Lightweight and Low-Cost e-Banking Solution Against Untrusted Computers. In *Financial Cryptography and Data Security*, pages 235–249. Springer, 2012.

[94] Shujun Li and Roland Schmitz. A Novel Anti-Phishing Framework Based on Honeypots. In *eCrime Researchers Summit (eCRIME '09)*, pages 1–13, Sept 2009.

[95] Kuan-Chieh Liao and Wei-Hsun Lee. A Novel User Authentication Scheme Based on QR-code. *Journal of Networks*, 5(8):937–941, 2010.

[96] Tom Liston. LaBrea: "Sticky" Honeypot and IDS. `http://labrea.sourceforge.net/labrea-info.html`, 2009.

[97] Michael Mimoso. Two-Factor Authentication No Cure-All for Twitter Security Woes. `http://goo.gl/mmEphG`.

[98] Robert Mitchell and Nicholas Thompson. *Deception: Perspectives on Human and Nonhuman Deceit.* State University of New York Press, 1985.

[99] Kevin D. Mitnick and William L. Simon. *The Art of Deception: Controlling the Human Element of Security.* Wiley, 2003.

[100] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Inside the Slammer Worm. *IEEE Security & Privacy*, 1(4):33–39, 2003.

[101] David Moore, Colleen Shannon, Douglas J. Brown, Geoffrey M. Voelker, and Stefan Savage. Inferring Internet Denial-of-Service Activity. *ACM Transactions on Computer Systems (TOCS)*, 24(2):115–139, 2006.

[102] S. A. Morin, R. F. Shepherd, S. W. Kwok, A. A. Stokes, A. Nemiroski, and G. M. Whitesides. Camouflage and Display for Soft Machines. *Science*, 337(6096):828–832, 2012.

[103] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen. HOTP: An HMAC-Based One-Time Password Algorithm (RFC 4226). Technical report, IETF, 2005.

[104] D. M'Raihi, S. Machani, M. Pei, and J. Rydell. TOTP: Time-Based One-Time Password Algorithm (RFC 6238). Technical report, IETF, May 2011.

[105] Syamantak Mukhopadhyay and David Argles. An Anti-Phishing Mechanism for Single Sign-On Based on QR-code. In *International Conference on Information Society (i-Society)*, pages 505–508. IEEE, 2011.

[106] Collin Mulliner, Steffen Liebergeld, and Matthias Lange. Poster: Honeydroid-Creating a Smartphone Honeypot. In *IEEE Symposium on Security and Privacy*, 2011.

[107] Sherry B. Murphy, J. Todd McDonald, and Robert F. Mills. An Application of Deception in Cyberspace: Operating System Obfuscation. In *Proceedings of the 5th International Conference on Information Warfare and Security (ICIW 2010)*, pages 241–249, 2010.

[108] Mummoorthy Murugesan and Chris Clifton. Providing Privacy through Plausibly Deniable Search. In *International Conference on Data Mining*, 2009.

[109] Vicentiu Neagoe and Matt Bishop. Inconsistency in Deception for Defense. In *New Security Paradigms Workshop (NSPW'06)*, pages 31–38, 2006.

[110] Raymond S. Nickerson. Confirmation Bias: A Ubiquitous Phenomenon in Many Guises. *Review of General Psychology*, 2(2):175–220, June 1998.

[111] Thomas Ormerod, Lingyu Wang, Mourad Debbabi, Amr Youssef, Hamad Binsalleeh, Amine Boukhtouta, and Prosenjit Sinha. Defaming Botnet Toolkits: A Bottom-Up Approach to Mitigating the Threat. In *4th International Conference on Emerging Security Information Systems and Technologies (SECURWARE)*, pages 195–200. IEEE, 2010.

[112] Open Web Application Security Project (OWASP). OWASP Top 10. `http://owasptop10.googlecode.com/files/OWASPTop10-2013.pdf`, 2013.

[113] Bimal Parmar. Protecting Against Spear-Phishing. *Computer Fraud and Security*, 2012(1):8–11, 2012.

[114] Darren Pauli. Cisco Posts Kit to Empty Houses to Dodge NSA Chop Shops. `http://www.theregister.co.uk/2015/03/18/want_to_dodge_nsa_supply_chain_taps_ask_cisco_for_a_dead_drop/`, 2015.

[115] C. Percival. Stronger Key Derivation Via Sequential Memory-Hard Functions. `http://www.bsdcan.org/2009/schedule/attachments/87_scrypt.pdf$\delimiter"026E30F$nhttp://www.unixhowto.de/docs/87_scrypt.pdf`, 2009.

[116] Nicole Perlroth. Hackers in China Attacked The Times for Last 4 Months. `http://goo.gl/L03TMK`.

[117] Fabien Petitcolas. La Cryptographie Militaire. `http://goo.gl/e5IOj1`.

[118] David Pintor Maestre. QRP: An Improved Secure Authentication Method Using QR Codes. Technical report, Universitat Oberta de Catalunya, 2012.

[119] Matthew B. Prince, Lee Holloway, Eric Langheinrich, Benjamin M. Dahl, and Arthur M. Keller. Understanding How Spammers Steal Your E-Mail Address: An Analysis of the First Six Months of Data from Project Honey Pot. In *Conference on Email and Anti-Spam (CEAS)*, 2005.

[120] Harry Quetteville. Fake Bus Stop Keeps Alzheimer's Patients from Wandering Off. `http://goo.gl/MXs1Ys`, 2008.

[121] Inaki Ranó. An Optimal Control Strategy for Two-Dimensional Motion Camouflage with Non-Holonimic Constraints. *Biological Cybernetics*, 106(4-5):261–270, 2012.

[122] Shrisha Rao. Data and System Security with Failwords, January 20 2005. US Patent App. 11/039,577.

[123] Ryan Riley, Xuxian Jiang, and Dongyan Xu. An Architectural Approach to Preventing Code Injection Attacks. *IEEE Transactions on Dependable and Secure Computing*, 7(4):351–365, 2010.

[124] Risk Analytics. 70 Million Dollars Stolen From US Banks With Zeus Trojan. `http://goo.gl/VaFl3T`.

[125] Ivan Ristic. *ModSecurity Handbook: The Complete Guide to the Popular Open Source Web Application Firewall*. Feisty Duck Limited, 2nd edition edition, 2012.

[126] Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C. Mitchell. Stronger Password Authentication Using Browser Extensions. In *Proceedings of the 14th conference on USENIX Security Symposium*, 2005.

[127] Neil Rowe. Counterplanning Deceptions to Foil Cyber-Attack Plans. In *Information Assurance Workshop*, pages 203–210, 2003.

[128] Neil Rowe. Designing Good Deceptions in Defense of Information Systems. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, pages 418–427, 2004.

[129] Neil Rowe. A Taxonomy of Deception in Cyberspace. In *International Conference in Information Warfare and Security*. Monterey, California. Naval Postgraduate School, 2006.

[130] Neil Rowe. Planning Cost-Effective Deceptive Resource Denial in Defense to Cyber-Attacks. In *Proceedings of the 2nd International Conference on Information Warfare & Security*, page 177. Academic Conferences Limited, 2007.

[131] Neil Rowe, E. John Custy, and Binh T. Duong. Defending Cyberspace with Fake Honeypots. *Journal of Computers*, 2(2):25–36, 2007.

[132] Neil Rowe, Han Goh, Sze Lim, and Binh Duong. Experiments With a Testbed for Automated Defensive Deception Planning for Cyber-Attacks. In *Proceedings of the 2nd International Conference on Information Warfare & Security*, page 185. Academic Conferences Limited, 2007.

[133] Neil Rowe and Hy S. Rothstein. Two Taxonomies of Deception for Attacks on Information Systems. *Journal of Information Warfare*, 2004.

[134] Richard Rubinstein and Harry Hersh. *The Human Factor: Designing Computer Systems for People.* Morgan Kaufmann Publishers Inc., 1987.

[135] Charmaine Sample. Applicability of Cultural Markers in Computer Network Attacks. In *12th European Conference on Information Warfare and Security*, pages 361–369, University of Jyvaskyla, Finland, 2013.

[136] Stuart Schechter, A. J. Bernheim Brush, and Serge Egelman. It's No Secret Measuring the Security and Reliability of Authentication via 'Secret' Questions. In *IEEE Symposium on Security and Privacy*, pages 375–390, 2009.

[137] Christian Seifert, Ian Welch, and Peter Komisarczuk. Honeyc: The Low Interaction Client Honeypot. *Proceedings of the 2007 NZCSRCS*, 2007.

[138] Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[139] Jaeeun Shim and Ronald C. Arkin. Biologically-Inspired Deceptive Behavior for a Robot. In *Lecture Notes in Computer Science*, pages 401–411. Springer, 2012.

[140] S. A. Sloman. The Empirical Case for Two Systems of Reasoning. *Psychological Bulletin*, 119(1):3–22, 1996.

[141] Meharouech Sourour, Bouhoula Adel, and Abbes Tarek. Ensuring Security-In-Depth Based on Heterogeneous Network Security Technologies. *International Journal of Information Security*, 8(4):233–246, 2009.

[142] Eugene H. Spafford. More than Passive Defense. `http://goo.gl/5lwZup`, 2011.

[143] Lance Spitzner. *Honeypots: Tracking Hackers.* Addison-Wesley Reading, 2003.

[144] Lance Spitzner. Honeytokens: The Other Honeypot. `http://www.symantec.com/connect/articles/honeytokens-other-honeypot`, 2003.

[145] Guenther Starnberger, Lorenz Froihofer, and Karl M. Goeschka. QR-TAN: Secure Mobile Transaction Authentication. In *International Conference on Availability, Reliability and Security (ARES'09)*, pages 578–583. IEEE, 2009.

[146] Clifford P. Stoll. *The Cuckoo's Egg: Tracing a Spy Through the Maze of Computer Espionage.* Doubleday, 1989.

[147] G. Edward Suh and Srinivas Devadas. Physical Unclonable Functions for Device Authentication and Secret Key Generation. In *Proceedings of the 44th annual Design Automation Conference*, pages 9–14, 2007.

[148] Latanya Sweeney. K-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[149] The Open Web Application Security Project (OWASP). ModSecurity Core Rule Set (CRS) 2.2.9. `https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project`, 2012.

[150] John Thibaut and Harold Kelley. *Interpersonal Relations: A Theory of Interdependence.* John Wiley & Sons Inc, 1978.

[151] J. R. Thompson, R. Hopf-Wichel, and R. E. Geiselman. The Cognitive Bases of Intelligence Analysis. Technical report, US Army Research Institute for the Behavioral and Social Sciences, 1984.

[152] Samuel T. Trassare. A Technique for Presenting a Deceptive Dynamic Network Topology. Technical report, Naval Postgraduate School, 2013.

[153] A. Tversky and Daniel Kahneman. Judgment Under Uncertainty: Heuristics and Biases. *Science*, 185(4157):1124–31, September 1974.

[154] A. Tversky and D. Koehler. Support Theory: A Nonextensional Representation of Subjective Probability. *Psychological Review*, 101(4):547, 1994.

[155] Amos Tversky and Daniel Kahneman. Extensional Versus Intuitive Reasoning: The Conjunction Fallacy in Probability Judgment. *Psychological review*, 90(4):293–315, 1983.

[156] Sun Tzu. *The Art of War*. Orange Publishing, 2013.

[157] Marynel Vazquez, Alexander May, Aaron Steinfeld, and Wei Hsuan Chen. A Deceptive Robot Referee in AaMultiplayer Gaming Environment. In *Proceedings of the 2011 International Conference on Collaboration Technologies and Systems (CTS'11)*, pages 204–211, 2011.

[158] Verizon. Threats on the Horizon – The Rise of the Advanced Persistent Threat. `http://goo.gl/ZnuJ9g`.

[159] Verizon. Data Breach Investigations Report (DBIR'14). `http://www.verizonenterprise.com/DBIR/2014/`, 2014.

[160] Verizon. Data Breach Investigations Report (DBIR'15). `http://www.verizonenterprise.com/DBIR/2015/`, 2015.

[161] Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. In *The 2007 Network and Distributed System Security Symposium (NDSS'07)*, 2007.

[162] Aldert Vrij. *Detecting Lies and Deceit: Pitfalls and Opportunities*. John Wiley & Sons Inc, 2008.

[163] Alan R. Wagner and Ronald C. Arkin. Acting Deceptively: Providing Robots With the Capacity for Deception. *International Journal of Social Robotics*, 3(1):5–26, 2011.

[164] Matthias Wählisch, André Vorbach, Christian Keil, Jochen Schönfelder, Thomas C. Schmidt, and Jochen H. Schiller. Design, Implementation, and Operation of a Mobile Honeypot. Technical report, Cornell University Library, 2013.

[165] Wei Wang, Jeffrey Bickford, Ilona Murynets, Ramesh Subbaraman, Andrea G. Forte, and Gokul Singaraju. Detecting Targeted Attacks by Multilayer Deception. *Journal of Cyber Security and Mobility*, 2(2):175–199, 2013.

[166] Marc G. Weinberger, Chris T. Allen, and William R. Dillon. Negative Information: Perspectives and Research Directions. *Advances in Consumer Research*, 8(1):398–404, 1981.

[167] Matt Weir, Sudhir Aggarwal, Michael Collins, and Henry Stern. Testing Metrics for Password Creation Policies by Attacking Large Sets of Revealed Passwords. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 162–175. ACM, 2010.

[168] Matt Weir, Sudhir Aggarwal, Breno De Medeiros, and Bill Glodek. Password Cracking Using Probabilistic Context-Free Grammars. In *IEEE Symposium on Security and Privacy*, pages 391–405, 2009.

[169] Gus W. Weiss. The Farewell Dossier. `https://goo.gl/llh2Xa`, 2007.

[170] Barton Whaley. Toward a General Theory of Deception. *The Journal of Strategic Studies*, 5(1):178–192, 1982.

[171] Barton Whaley. *Stratagem: Deception and Surprise in War.* Artech House Information Warfare Library, 2007.

[172] Michael J. Wirthlin and Brad L. Hutchings. A Dynamic Instruction Set Computer. In *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 99–107. IEEE, 1995.

[173] YubiCo. YubiHSM Manual. Technical report, YubiCo, Palo Alto, CA, 2015.

[174] Chuan Yue and Haining Wang. BogusBiter: A Transparent Protection Against Phishing Attacks. *ACM Transactions on Internet Technology (TOIT)*, 10(2):6, 2010.

[175] James Joseph Yuill. *Defensive Computer-Security Deception Operations: Processes, Principles and Techniques.* PhD Dissertation, North Carolina State University, 2006.

[176] James Joseph Yuill, Mike Zappe, Dorothy Denning, and Fred Feer. Honeyfiles: Deceptive Files for Intrusion Detection. In *Information Assurance Workshop*, pages 116–122. IEEE, 2004.

[177] Lianying Zhao and Mohammad Mannan. Explicit Authentication Response Considered Harmful. In *New Security Paradigms Workshop (NSPW '13)*, pages 77–86, New York, New York, USA, 2013. ACM Press.

APPENDIX

A DECEPTIVER IMPLEMENTATION

A.1 Deceptiver Command Line Inputs

The sever is invoked by executing `python3 deceptiver.py (arguments)`. The following list are the inputs to the server as command line arguments.

- **'-a' or '–admin'**

  Takes you to the administrative section of Deceptiver.

- **'-s' or '–server'**

  To provide the type of server initiating the Deceptiver. E.g., http, ftp, etc.

- **'–sd' or '–server-details'**

  To provide the server details such as version, name, etc.

- **'-p' or '–port'**

  The port the server is listening on.

- **'–uri'**

  To provide the server the requested resource URI.

- **'–ip'**

  To provide the IPv4 address of the client requesting access. The IP address must be the in the format of x.x.x.x.

- **'–secure'**

  When this argument is included, it informs the Deceptiver that the connection was made over a secure channel.

- **'-m' or '–malicious'**

  When this argument is included, it informs the Deceptiver that the server knows that this client is malicious.

- When the server is an HTTP server, the following arguments can be used:
  - **'–uagent' or '–user-agent'**

    HTTP's user's agent.
  - **'–referrer'**

    HTTP's referrer.
  - **'–method'**

    HTTP's method used.

  The following arguments are always required '-s', '–sd', '-p', '–uri' and '–ip' and the following arguments are required from HTTP servers '–method'.

VITA

VITA

Mohammed H. Almeshekah has been working in the area of Information Security for more than seven years. He holds a PhD in Computer Science from Purdue University, a Masters in Information Security from Royal Holloway, University of London and a Bachelor in Computer Science from King Saud University. Broadly, his area of interest is Information Security. Within security, he is interested in three main areas: the use of deception in security, usable security, and authentication. His work has appeared in a number international conferences and workshops; and has been profiled by CIO, ACM TechNews, Kaspersky Labs and other international outlets in more than four languages.

Previously, Mohammed worked on improving the security of the Chrome and Firefox browsers working at both Google and Mozilla as a Security Engineer. In addition, he won a number of awards including the ”Diamond Award” awarded by the Center for Education and Research in Information Assurance and Security (CERIAS) at Purdue; and a “Teaching Fellowship” awarded by Purdue’s Computer Science Department.