CredProxy: A Password Manager for Online Authentication Environments

By

Mohammad Saleh Golrang

A thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.Sc. degree
In Computer Science

Ottawa-Carleton Institute for Computer Science
School of Electrical Engineering and Computer Science
University of Ottawa

Abstract

Internet users are increasingly required to sign up for online services and establish accounts before receiving service from websites. On the one hand, generation of strong usernames and passwords is a difficult task for the user. On the other hand, memorization of strong passwords is by far more problematic for the average user. Thus, the average user has a tendency to use weak passwords, and also reuse his passwords for more than one website, which makes several attacks feasible. Under the aforementioned circumstances, the use of password managers is beneficial, since they unburden the user from the task of memorizing user credentials. However, password managers have a number of weaknesses. This thesis is mainly aimed at alleviating some of the intrinsic weaknesses of password managers. We propose three cryptographic protocols which can improve the security of password managers while enhancing user convenience. We also present the design of a phishing and Man-in-the-Browser resistant password manger which best fits into our scheme. Furthermore, we present our novel virtual on-screen keyboard and keypad which are designed to provide strong protection mechanisms against threats such as keylogging and shoulder surfing.

Acknowledgements

I would like to thank my supervisor, Dr. Carlisle M. Adams for his invaluable guidance, insightful feedback and constructive suggestions without which this thesis would not have been possible in the first place. I gratefully appreciate his patience, constant encouragement and the trust he put in me.

I would like to thank my fellow lab colleagues and friends for the wonderful and useful discussions we always had. Their insightful suggestions and discussions are much appreciated.

Lastly, I would like to thank my family and friends for their constant support and encouragement. Special thanks to my father for his precious help and support without which I would not have made it through the tough times.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

The Internet has provided us with services and opportunities which were not available through traditional channels of communication. In recent years, this invention has realized many of mankind's dreams. Now, many services ranging from education to entertainment to financial services are delivered through the Internet at a fraction of the cost of older channels. As a direct result of its ever-growing popularity and convenience, the number of Internet users has grown five-fold during the last decade from 361 million users in 2000 to approximately 2 billion global users in 2010 [1]. Moreover, as more people have moved online, the operating costs are reduced, and consequently, online services are offered at lower prices. This, as a result, has provided many a business the opportunity to emerge and grow and the ability to deliver their services worldwide at competitive prices.

As the technology advances, newer computing platforms and technologies have come into existence. Thus, the effort has always been to offer online services through these newer platforms and channels. In recent years, we have witnessed such a trend with the ever-increasing popularity of smart phones: more and more mobile applications and online services are created and made available to the public on mobile devices at little or no cost.

However, this has been a two-edged sword; unfortunately all the benefits of the Internet and technological advancements are also available to criminals, or in this context "cyber criminals." They have shown themselves to be quick to exploit zero-day vulnerabilities [2] and software and hardware technologies' weaknesses. Nonetheless, computer errors and technological weak points have not been their only instruments of crime: they also use sophisticated tactics to persuade victims to give up information or to do some specific actions which might result in the user divulging confidential information such as account and online banking credentials, credit card information or even information resulting in access to the entire machine of the victim. To make things even worse, cyber criminals always seem to be a few steps ahead of the “good guys”: security specialists and cyber defenders have to relentlessly create and change technologies, tools, procedures, etc., just to be able to ward off some portion of cybercrime.

Financial institutions and services such as banks and credit card companies have become an area of growing interest for cyber criminals. They indeed have turned into a primary target and victim of cybercrime all around the world [3]. The losses caused by cybercrime in this industry were around $54 billion in 2009 [3].

While the number of cyber-attacks has increased by 81% in 2011, the total number of vulnerabilities discovered has decreased by 20% in 2011 [4]. This is indicative of an attention-grabbing fact: cyber-attacks have become more sophisticated. In fact, attackers might be pursuing different purposes in these attacks. One possible objective, as an example, could be to steal identities. The attackers could, then, use the stolen identities to apply for new credit cards, open new bank accounts and write checks in the victims’ names. In 2011 alone, hacking attacks exposed more than 232.4 million identities worldwide and almost 1.1 million identities were exposed in each data breach with an average incident cost of USD$5.5 million in total [4].Also, according to a report released by the Canadian Bankers Association (CBA), there were 2442 fraudulent credit card applications with false identities in Canada in 2009 which resulted in a loss of CAD$ 4,707,088 [5].

However, online banking credentials and credit card information are of more interest to cyber criminals since they are much easier to use and sell. In underground forums and marketplaces, stolen credit cards are sold from $0.07 to $100 and bulk purchasing would certainly reduce the prices [6].All in all, this comes as no surprise, as one of the primary purposes of cyber criminals is to generate revenue: they are after information, and information is of value.

Online authentication on websites still heavily relies on text username and passwords regardless of their intrinsic problems [7]. First, the average user might find it difficult both to generate and memorize strong passwords. Weak and reused passwords make guessing and dictionary attacks feasible. Moreover, as the average user creates more online accounts, password reuse increases [8]. Second, in order for the user to perform online authentication, he has to type in his username and password in the browser, which makes user credentials subject to threats such as keylogging and Man-in-the-Browser attacks. Keyloggers are malicious applications that keep a log of all the keystrokes typed on a physical keyboard by a user in order to be further analyzed by the attacker. Symantec has listed keyloggers as one of the most common computer security threats with both prevalence and danger level as high [9]. In fact, what makes keyloggers devastatingly dangerous is that they are largely undetectable by most virus killers [10]. On the other hand, virtual on-screen keyboards, which gained much attention due to their ability to bypass keyloggers, are susceptible to shoulder surfing and screen grabbers (or screen scrapers) [10]. Screen grabbers are malicious applications which take screenshots of the victim's display for every mouse click the victim does. These screenshots are later sent to the attacker for analysis in the hope of extracting and stealing information (e.g. user credentials) input through virtual on-screen keyboards.

Another fast-growing method of stealing information from online users is Man-in-the-Browser (often abbreviated MITB) attacks which is a combination of eavesdropping and code injection in the browser. This method has come under the spotlight by attackers since browsers have become the gateway to our banks, email accounts, and many more services that we receive through the Internet. Launching this attack, cyber criminals are able to steal credentials and online banking information, bypass transaction authentication numbers (TAN), log keystrokes in the browser, take screen captures and so on. All this information will then be transferred to the attacker at the intervals already specified by the attacker for further analysis.

One of the most sophisticated and destructive Trojan horses ever developed which is capable of launching MITB attacks is called Zbot or Zeus [11] which infected machines in more than 196 countries [12]. It is estimated that more than 3.6 million computers were infected in the United States of America [12]. A survey carried out in 2010 revealed that more than half of financial services professionals at 70 banks believed that banking Trojan horses such as Zeus are considered the greatest threat to online banking [13].

Now, some questions arise: What measures can be taken? Are there any solutions to these problems or should we only wait for threats such as the ones mentioned above to emerge and then devise defence mechanisms? Could we be proactive in this matter? Could we devise a mechanism that would ward off such threats and keep us safe in the wild? This thesis discusses the questions mentioned above and proposes a combination of techniques to defend against such attacks in the context of password managers.

1.2 Motivation

Creation and memorization of strong passwords seems like a Herculean task to the average Internet user. Passwords that are weak make brute-force attacks disquietingly feasible; as a result, common password attacks [14] could also bear fruit in the follow-up procedures by adversaries. Furthermore, when password managers are used, the password database stored on the client machine is not well protected, since the master password to the password database is still chosen by the user, which could probably be the Achilles' heel of password managers. On the other hand, strong passwords do not do much to protect online users from password stealing attacks such as phishing and keylogging [15].

Furthermore, Man-in-the-Browser attacks where malware resides in the browser differ from other types of attack, since they are specifically designed to avoid signature-based detection mechanisms [16]. The behavior of this type of malware is in such a way that is not flagged "dangerous" under normal circumstances, and requires extra effort to detect it. The MITB malware resides between the browser and a website and collect data (including authentication information) before even data input in the browser being encrypted using SSL; as a consequence, SSL-based solutions cannot provide protection against such attacks. MITB attacks include monitoring, intercepting and modifying user activities in the browser in real time. Therefore, when deployed successfully, they give an adversary a huge advantage. Moreover, the detection of keyloggers is nothing short of a challenge: keyloggers have sophisticatedly simple designs which enable them to perform their tasks without attracting the attention of users and antivirus programs so that they could go undetected for a relatively long time [10]. In late 2011, Duqu [17] was detected by Symantec [4]. Duqu is a descendent of Stuxnet [18] that records keystroke and other system information by installing spyware through a zero-day vulnerability [4]. Symantec reports that machines in more than 8 countries in the world were infected by this malware [17], which resulted in a huge loss of data from users and companies.

Given the aforementioned scenario, we believe we should be searching for new techniques as counter-measures protecting user credentials. In this thesis, we propose a number of different techniques, the collection of which can provide the average online user with more security in practice.

1.3 Organization of the Thesis

We begin in **Chapter 2** with defining the problem of securing sensitive data input. Then, we discuss a well-known piece of malware that can perform MITB attacks. We continue with a survey of common protection mechanisms in the literature, and identify their shortcomings with respect to the problem definition. In **Chapter 3**, we present the design of CredProxy. In **Chapter 4**, we discuss Function CPGen for automatic username and password generation along with the credential vault. **Chapter 5** discusses Function CPGuard for encryption/decryption of the credential vault. In **Chapter 6**, we discuss our proposed virtual on-screen keyboard and keypad, and we also provide the implementation and informal usability data. Finally, **Chapter 7** discusses the future work and concludes.

1.4 Contribution

We analyze a typical Man-in-the-Browser attack, and a well-known MITB attack kit called Zeus. We present a survey of the related work, especially mechanisms to protect user credentials and password generation and management approaches. We present the design of a password manager which improves phishing and keylogging protection. We propose three different two-factor authentication protocols through which the credential vault on the client machine is protected in the face of password database theft (e.g. loss of the computer or theft of data from the machine). In these protocols, we make use of a hand-held device such as a smartphone as the second-factor authentication. In our model, we encrypt user credentials with keys of adequate entropy stored in a distributed model and store the credential vault on the client machine. We propose a novel way to generate anonymous usernames and strong passwords in order to unburden the user from the difficult task of username and password creation. Moreover, we propose a new virtual on-screen keyboard and keypad to secure input of sensitive data. It should be noted, however, that we have borrowed some ideas from a commercial product. In fact, we undertook a project of reverse engineering and security and vulnerability analysis of a commercial product prior to this thesis. This project was undertaken at an Ottawa-based company's request, and was funded by NSERC[1] (National Sciences and Engineering Research Council of Canada). In the aforementioned project, we had access to only the binary code and not the source code of the software, and our findings are the result of taking the software apart, decompilation of the binaries of the code, and observing the functional behavior of the software in specialized environments. The proposed methods in this thesis are, in part, aimed at enhancing the security mechanisms of that software. Therefore, whenever borrowing from the software has happened, we have explicitly pointed it out in this thesis in order to give credit to the original developers. It should also be noted that we have refrained from naming the company particularly in order to preserve their privacy.

[1] http://www.nserc-crsng.gc.ca/

Chapter 2

Background and Related Work

This chapter defines the problem of securing the user's sensitive data input. We also analyze the anatomy of a Man-in-the-Browser attack and a very well-known piece of malware called Zeus or Zbot [11]. This chapter, then, continues with a survey of existing literature on how to protect credentials from being stolen and also on password management strategies. We highlight some of the advantages of each approach, and also point out some aspects of them which can be perceived as shortcomings.

2.1 The Problem of Securing Sensitive Data Input by Users

Untrusted Browsers. Browsers are the gateway to the Internet. We almost use them every day for different purposes from connecting to friends and family, to checking our emails, to making online purchases and many more. However, in order to access some services we must identify ourselves to the intended online party. The dominating and perhaps most popular form of authentication used in the browser to access online services is the combination of text usernames and passwords. We also use our sensitive banking information to buy products, pay bills, purchase services, etc., online; consequently, we tend to trust browsers. However, recent attacks and detected malware [11] have proved that this trust may be misplaced. Malware is capable of stealing information from the browser by using different techniques such as code injection or vulnerabilities in browsers. Code injection is the process of introducing code into a program in order to change the course of execution [19]. Zeus [12] has the capability of injecting code in the browser to steal information and even perform unauthorized banking transactions. On the other hand, browser vulnerabilities have made it possible for attackers to perform all sorts of illegal actions. As an example, JailbreakMe [20] exploits vulnerabilities in

Safari [21] to inject code through the browser to perform Apple's iOS mobile operating system [22] jailbreaking [23]. Surprisingly, unlike other iOS jailbreaks that require connecting the phone to a computer and running the jailbreak application from the computer, JailbreakMe only requires the user to visit their website, and through the website the necessary code will be injected onto Safari and the phone will be simply jailbreaked [20].

Keyloggers and Screen Grabbers. In addition to the vulnerabilities of browsers, keyloggers and screen grabbers have added to the problems. A keylogger is a program that tries to learn sensitive information through recording all the victim's keystrokes during a session [24]. Afterwards, the recorded information is analysed by an adversary in order to extract sensitive user input such as usernames/passwords or credit card information. In fact, a keylogger is considered one of the most insidious threats to a user's personal information [25]. There are also many ways and strategies to implement keylogging applications; for instance, Microsoft Windows user32.dll make available some event handlers that can be invoked to trap all keyboard and mouse events [25]. In response, on-screen keyboards came under the spotlight in order to ward off keylogging attacks. On-screen keyboards allow key strokes to be passed directly from the screen to the application in order to protect the data input onto a computer. They are also developed in JavaScript in order to make the online user input more secure. However, this defence mechanism gave more incentive to attackers to use screen grabbers. A screen grabber is a program which can take screenshots of the system display based on some defined events in order to record visible information displayed on the screen. An on-screen keyboard might protect against keyloggers; nonetheless, with every mouse click, a screen grabber is able to take screen shots which may divulge sensitive information. After screenshots are taken, they will usually be sent to be analysed by an attacker.

Phishing. Phishing is the cyber-crime of stealing users' login credentials and online banking information (e.g., credit card details) through masquerading as real websites and luring victims to fake websites that look very much like the intended websites. An average user typically lacks the ability to distinguish real websites from fake ones, which eventually leads to victims' loss of money, identity and data. Nowadays, almost all websites that require authentication use SSL-encrypted channels at least for the authentication part. However, phishers may use fake SSL [26] certificates and rely mostly on victims' naivety to ignore SSL warnings in the browser, although there are several more approaches (e.g. using fake SSL padlock icons [27]). Sunshine et al. [28] in 2009 conducted a survey of more than 400 Internet users to investigate their reactions to current SSL warnings in 3 popular browsers. Their result suggests that too many of these participants exhibited dangerous behavior in all SSL warning situations, and that a better approach to maintain security is to minimize SSL warnings by preventing users from making potentially unsafe connections to sensitive websites such as banks, etc., when the certificates they present do not pass all the checks.

Untrusted Display. Besides the ease of recording keystrokes and screenshots, the authenticity and integrity of the screen content are yet another issue. The computer display is the primary means of data output, and users usually find it trustworthy. However, an adversary could overwrite any area on the screen and cause bogus information to be displayed [24]. Unfortunately, the assumption that what is displayed to us is what is really happening is not valid any more. We must find ways to have more reassurance for sensitive actions performed on the computer other than what is displayed on the computer display.

Untrusted Mobile Devices. The ability to communicate and perform computation while on the move has given rise to a sharp proliferation of mobile devices and smart phones in particular [29]. According to [30], in 2011, the number of mobile phones in the world was over 5.6 billion, and the number is still growing. In parallel with this growth, however, the number of security threats and abuses has gone up as well. A report [31] by the United States Computer Emergency Team [32] reveals that the number of vulnerabilities in mobile operating system increased by 42 percent from 2009 to 2010. As a more specific example, the number of malware on the Android OS surged more than three-fold between the first quarter of 2011 and the third quarter of 2011 [33]. All in all, as can be seen from the statistics, the world of mobile devices has attracted a lot of attention from attackers and cyber criminals, and its prevalence has given them ever-growing motivation for cyber-attacks. Thus, the assumption that mobile devices are trusted [24] [34] [35] is no longer a valid assumption.

2.2 Anatomy of a Typical MITB Attack

Man-in-the-Browser attacks are a result of stronger security mechanisms and increased consumer awareness [36]. They are designed to intercept data while data is exchanged between a browser and a web server, and they are usually executed through Trojan horses which are, unlike worms, non-self-replicating malware. They can inject code in browsers, steal credentials, modify banking transaction details on the fly while displaying users' intended transaction details and, of course, without users knowing it. This type of attack is very different from phishing attacks in which browsers are involved as well. Phishing attacks are also intended for theft of information, particularly credentials. In this type of attack, users are usually lured to fake websites that look the same as users' intended websites. Once users are there, they input their credentials to authenticate their identities and gain access to whatever service was intended. Although users might gain the intended access and results, cyber criminals have already stolen users' credentials.

A very effective defense mechanism against phishing attacks is to use an out-of-band channel to confirm authentications and transactions. Using this method, users receive a text message on a separate mobile device or receive an automated phone call whenever users intend to authenticate or to perform a banking transaction. This out-of-band message can deliver one-time pass codes (or they may be called "transaction authorization numbers" or TANs or sometimes "transaction authorization codes" or TACs) which can, in turn, be input to the website. This type of code acts as a piece of confirming evidence that authentications are initiated by intended users. For banking purposes, along with one-time pass codes, other transaction details such as recipients, amounts of transaction, time and dates, etc., can be sent to users. Consequently, after reviewing transaction summaries, if everything looks as it should, users can, then, input pass codes and finalize transactions. This can, indeed, add an extra layer of security to the existing systems, which is considered to be an effective way to mitigate both phishing and Man-in-the-Browser attacks [37]. As an example, Google Authenticator [38] is one such application which can be installed on some mobile devices (as of the time of writing, it is supported on Android version 2.1 or later, iPhone iOS 3.1.3 or later, BlackBerry OS 4.5 - 6.0), and implements two-step verification for Gmail and Google Apps users. Using this application, pass codes are sent to Gmail and Google Apps users whenever they want to access their accounts, and users must input the already-sent pass codes in order to be authenticated [38].

Although it might sound foolproof, this layer of security can be broken, and is not the ultimate answer we have been seeking to the problem. In the next sections, we discuss the shortcomings of this mechanism and how it got compromised in recent attacks.

As mentioned earlier, a Man-in-the-Browser attack utilizes what is known as a Trojan which is not a self-replicating type of malware. Thus, it needs to be pre-configured and installed on the victim's device. However, with current levels of user awareness, antimalware software programs, hardened web browsers and many more security mechanisms which are embedded in today's operating systems, it is not so easy and simple to deceive users into downloading and installing malware applications as it used to be. Thus, cyber criminals are forced into revising and refining their methods of deception on a regular basis. This type of attack, indeed, is the result of such obligation and refinement. Every MITB attack is fulfilled in two phases:

1. Phase 1, in which attackers strive to infect victims' machines.
2. Phase 2, in which attackers steal information or take over banking transactions or both.

These phases are described below.

2.2.1 Phase 1: Infection and Installation

In this phase of the attack, cyber criminals try to infect the targets' computers. These targets are categorized into two major groups:

1. Random users who happen to get infected by the malware.
2. People who are precisely targeted by attackers such as employees of an organization or members of a group.

These two groups are dissimilar, since methods used for luring the second group into getting infected are specifically customized for them.

For both groups, social engineering plays a key role. Social engineering is an instance of psychological manipulation in which a victim is deceived into performing an action that might result in giving unauthorized computer access to an attacker or divulging confidential information without the use of any technical methods of hacking [39] [37]. In this context, the attacker tries to use deceptive ways to scare, attract, stimulate or entice the victim to do an action for the purpose of infecting the machine or gathering information about the victim. Scareware [40], for instance, uses this technique. Many of us have experienced pop-ups in the browser, where we are alerted that everybody can see our IP address or that our computer is infected with a virus or it has some Windows Registry issues. It may not be so hard to convince a naïve user to download a free antivirus or Windows Registry fixing tool which might turn out to be a malicious application.

Phishing and phishing emails have also proven to be effective to infect victims. Using this technique, the attacker sends victims emails suggesting that users visit some web sites in order to download a piece of free software or to see some celebrity's leaked photos, etc. Then, users are asked to download an application which is the malware in disguise. Of course, in this phase, the objective is not to get users' credentials. It is only to deploy the malware on victims' machines. Thus, techniques used here are very well crafted to not have a suspicious look. In the second half of 2011, there were at least 83,083 unique phishing attacks throughout the world in the 200 top-level domains alone [41].

Furthermore, there are websites that users might visit without being forwarded to them in phishing emails. For instance, users might look for a cracked version of some software or a free online movie streaming or a pirated e-book. Users usually look for these digital contents on popular search engines and as soon as clicking on links they are taken to some malicious websites where they can apparently download the intended content or watch a movie online. There, they download the malware infected software or they are prompted that they are missing a video codec, etc. and that they should download it to able to watch the video. When users open the downloaded content, the malware installs and infects the computer silently [37].

Drive-by downloads and unpatched browser vulnerabilities also play an important role in malware distribution. Drive-by download attacks are usually launched from websites which are owned by cyber criminals or are compromised by attackers. They can also be sites that allow third parties to put content such as ads, etc., on them. The first step, in this attack, is to insert a Malware Bootstrap Function (MBF) into the address space of the browser and subsequently through some vulnerability (ies) in some browser, this small piece of code is executed [42]. This, in turn, results in downloading more malware and finally accomplishing the infection step [42].

However, when a specific person or a group of people are targeted more information is needed to launch an attack. This information is used to convince users that spear-phishing emails, etc., originate from a legitimate source that knows them. This information can mainly be obtained from online sources and social networking sites that can directly or indirectly be exploited to dupe victims. For instance, many people list their real names, birthdates, education and employment details on social networking sites. By exploiting profile information, attackers can disguise themselves as legitimate sources and mount their targeted attacks. In second half of 2011, APWG (Anti-Phishing Working Group) reported that 487 target institutions including users of banks, social networking sites, e-commerce sites, government organizations, etc., were counted [41]. There are also other methods of infection such as OS vulnerabilities, DNS cache poisoning, P2P networks, etc., which can also be used for malware infection; however, their discussion is beyond the scope of this thesis.

After downloading the malware, now it is time for infection. These Trojan horses which carry out MITB attacks are usually in the form of ActiveX controls, user scripts or browser helper objects which mange to stay undetected from traditional virus scanning [37]. They are hard to detect since malware developers usually customize and morph the malware for different targets [3].

MITB attacks are also difficult to detect from the server side, since all the user’s actions seem to be legitimate and seem to originate from a legitimate user’s browser. What a server needs to authenticate a user and authorize a task can totally come from a compromised machine with correct certificates, IP addresses and other hardware properties which a user may be identified by. Thus, it is a big challenge for companies, especially financial institutes, to distinguish between legitimate and malicious users [36].

2.2.2 Phase 2: Exploiting Victims

After the infection phase, comes the time for the attacker to harvest what he has sown. So far, the victim has been infected and the malware has been deployed. This malware can be in the form of a rootkit (a collection of tools which enables the attacker to hide traces of a computer compromise from the operating system and the user [6]) that infects the operating system as well in order to go undetected for a longer period of time, although this is just an option which highly depends on the capabilities of the malware and the intentions of the attacker.

Consequently, the next time the victim starts his browser, the malware activates. It stays silent until the victim visits a website. Then, the URL of the website is searched against a list of URL addresses which is predefined by the attacker. If the URL is one of those on the list, the malware awakens; otherwise, it keeps waiting in the background.

Suppose the victim wants to log into a website which is on the list. The malware can subsequently inject code on the browser to grab whatever information is input by the victim on the form, and it usually makes no difference if the victim logs in through http [43] or https [44]. As an example, the malware can register a button event handler that takes out all the form's data by clicking the submit button. Thus, in this way, the credentials and log-in information can be stolen from the victim.

Of course, not all types of MITB malware use this method. Another possible avenue could be that the malware waits until all the phases of authentication are completed by the victim. Then it modifies the content of a transaction. Receiving the transaction details by an authenticated user, the server does not usually treat it as suspicious and proceeds with the transaction, and sends back the transaction receipt to the victim. However, as soon as the receipt is received, the malware will detect it and will change the transaction details to those intended by the victim [45]. Furthermore, the use of one-time pass codes (OTP) through OTP tokens and out-of-band OTP (delivered on a separate device) by financial institutes and banks have been rendered ineffective, because the malware changes the transaction details but displays the details of what the victim has intended to do in the browser. The victim, who sees no discrepancies between his request and the transaction details sent over to his browser, enters the OTP in the browser.

The aforementioned step, without a doubt, requires a piece of sophisticated and lengthy malware. The attacker might not even bother going that far. The malware might just wait for the victim to finish his business with the website and log off. When user requests log-off, the request is overridden by the malware and a fake message is shown to the victim. The control then is passed to the attacker, or new transactions are made in the background [36]. During all the stages of MITB attacks, the victim is not suspicious that something is wrong, since everything he sees in the browser looks as it should, and nothing fishy appears to be going on.

A more effective way of warding off the MITB attack could be to give users more information about transactions through an out-of-band channel [37]. In this method, along with OTP, transaction details such as recipient's name, amount of transaction, time and date are sent to an out-of-band device [37]. Then, users can make sure that details are the same as the details in the browser and then authorize the transaction. Thus, even if the malware changes everything in the background, the user will realize that details displayed on the machine are different than the details sent to him and will not authorize the transaction. The user might consequently become suspicious that something is wrong on his machine and might find out that his machine is malware-infected. Although considered an effective way, in the next section we point out how this layer of security got compromised in recent attacks.

2.3 The Zeus Attack Kit

Zeus or Zbot [11] is one the most sophisticated and dangerous attack kits ever developed by cyber criminals. It is believed to have been first developed in 2007, and has been enhanced over time by its developers [11]. Zeus is likely to have been developed in a Russian-speaking country since the initial help files were all in Russian [11]. According to a report by the FBI, Zeus has infected up to 4 million computers in the USA, and the financial losses due to this malware are estimated to be up to 60 million US dollars in the USA alone [46].

What gives common criminals the ability to launch attacks using this sophisticated malware package is the fact that it has been sold by its developers and other criminals in the possession of it. Zeus can be both purchased for as low as 700 US dollars in underground forums and among hackers, and can be found for free on the Internet [11]. Consequently, it has still remained a common tool of compromising unprotected machines.

The Zeus package contains a builder that can generate the necessary executable client and server side files. The client side file is a bot which should be downloaded and installed through one of the aforementioned fashions (drive-by downloads, spam, etc.)The server side files include PHP images and SQL templates which are used as the command and control server. The primary purpose of Zeus is to generate revenue both for utilisers and sellers of it.

Once the bot is downloaded and executed, Zeus checks to see under which account it is run. There are two possible scenarios:

1. An account with Administrative privileges: the main bot copies itself and the necessary files into Windows/System32 directory, and make some modifications to the Windows Registry on the victim’s machine. Then, it increases its privileges through changing winlogon.exe in order to be able to inject code into this process, and also creates a thread to execute the code. This injected code, in turn, injects more code into svchost.exe, which is responsible for network communication and other code injection operations in order to steal credentials and information from banking sites.
2. An account without Administrative privileges: in this case, instead of winlogon.exe, the code is injected into explorer.exe. Moreover, the bot copies itself to UserProfile/AppData instead of Windows/System32, and makes necessary Windows Registry modifications.

However, there are always two main files on the victim's machine regardless of the installation mode:

- Local.ds: in this file, the latest dynamic configuration is located. Initially, the configuration is what has been shipped with the bot. However, later through the Internet, the bot can update its configuration file by downloading up-to-date configuration files from its server.
- User.ds: all the credentials and stolen information are stored in this file which is transmitted to its server as frequently as what has been defined in the local.ds.

After installation, it grants the attacker full unauthorized access to the victim's machine. Thus, the attacker can carry out tasks through scripts sent to the victim's machine, or define some tasks to be carried out when a certain condition is met. Credential and data stealing, which is the objective, can occur in various scenarios regardless of the security mechanisms used. Some of these credential capturing methods are as follows:

Unencrypted passwords: Zeus can steal passwords sent in clear text format such as FTP and POP3 passwords and can also steal passwords stored in the PSTORE (Protected Storage).

WebFilters: Through web filters, a list of URLs is specified and monitored for activity. Whenever one of these URLs is browsed, the data sent to it is captured and later sent to the command and control server. It should be noted that this data is captured before it is encrypted using SSL on the client machine. Furthermore, it can be specified that with each left-button click a screenshot be taken. These screenshots can help circumvent the use of virtual keyboards.

WebFakes: They redirect a specific URL to another URL specified by the attacker.

TANGrabber: The Transaction Authentication Number grabber routine is a routine that allows the attacker to configure match patterns to search for one-time passwords and TANS in data sent to banks. The captured TANs can be used to finalize unauthorized transactions.

As can be seen above, most common security mechanisms are circumvented in the case of being infected by this malware. SSL [47], which has been regarded as a successful security defence, has proven ineffective to ward off this kind of attack. Moreover, using TANs and one-time passwords cannot stop this kind of malware as attackers can wait for the victim to enter TANs and then use them to carry out unauthorized transactions.

Although some companies such as Entrust[2] consider the use of out-of-band transaction details along with one-time passwords an effective way to block Man-in-the-Middle attacks [37], this too has been compromised in recent attacks [48]. In order to bypass this mechanism by Zeus, the malware can generate a fake pop-up message box in the browser asking the victim to input his type of cell phone and the phone number which he uses to receive the messages from his bank. Then, through social engineering an additional piece of malware is installed on the victim's phone. One example of this complementary phone malware is called Zeus Mitmo [49] [50] which works in tandem with Zeus. This malware acts as a backdoor and receives commands from its designated command and control server through SMS messages. Next, the phone malware monitors all the incoming SMS messages, waiting for the right SMS messages from the bank. When a message from the bank is received, the message will not be displayed to the victim and will be sent to attackers. Using the information in the message, attackers can carry out unauthorized transactions.

In spite of all the efforts by governments and security companies such as Symantec [51], we can still witness that not only have bots like Zeus not disappeared, but they also have been evolving and using more sophisticated ways to stay ahead of detection tools. For instance, Symantec has recently revealed that newer versions of Zeus use a peer-to-peer model instead of bot-to-command-and-control-system architecture in order to keep the botnet alive and let attackers go undetected [52]. As a result, even if a command and control server were taken down, the bot could still steal data which could be relayed through the botnet peer-to-peer network. To make it even worse, a tight rivalry between bots developers to steal data and credentials has given cyber criminals an added incentive to enhance their methods and architectures. In early 2010, a new Trojan horse called Spyeye [53] was discovered and believed to be the killer of Zeus. This malware, which is very similar to Zeus in functionality, first copies all the data

[2] http://www.entrust.com

gathered by Zeus, and then tries to kill this malware and take its place in the world of cyber-crime [54] [55].

All in all, it is all about financial gain for this kind of cybercriminal. They have proved less to be after cyber sabotage and more to be pursuing money. They, indeed, have shown growing interest to gain access to our credentials and banking information, and much to our disappointment, defence mechanisms such as SSL [47]and TAN have been rendered ineffective to protect us against them.

2.4 Survey of Related Work

In this section we firstly investigate the existing literature and methods to protect credentials and sensitive data input on the computer. We focus on methods to protect credentials against keyloggers, spyware and Trojan horses which monitor user inputs and take screen shots. In particular, our focus is on methods designed to protect users on untrusted computers. Later on, we also examine some approaches and techniques regarding password generators and managers, since our scheme is indeed a password manager.

In 2004, **Wu et al.** proposed an authentication protocol based on a trusted proxy and mobile devices. As can be seen in figure 2.1, the user who wants to use a remote service on an untrusted computer must connect to a proxy first to complete the authentication process. On the proxy the user’s ID with his phone number is stored. Every time the user wants to authenticate to the remote service through the proxy, an SMS message is sent to the user’s mobile device. Upon the user’s confirmation, the proxy authenticates to the intended server on behalf of the user, and the connection is established between the untrusted computer and the server through the proxy. In fact, the proxy relays all the data except for long-term authenticators (e.g., cookies) between the server and the untrusted computer.

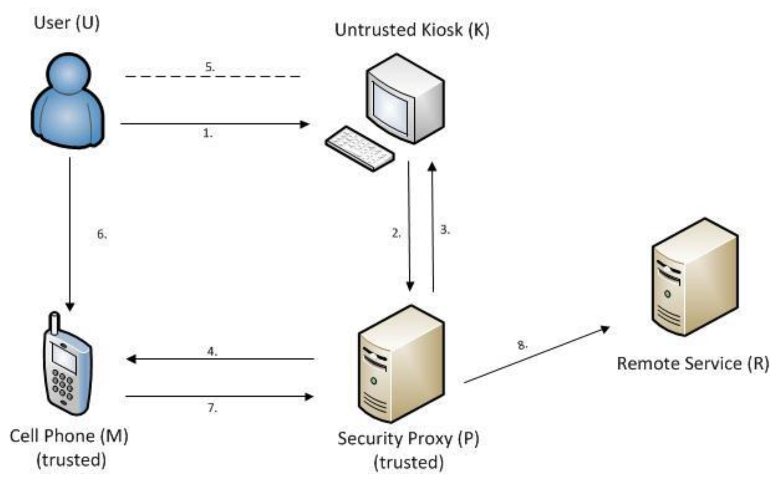


Figure 2.1: Wu et al. Authentication Protocol

The authentication protocol is as follows:

1. The user launches a browser on the untrusted computer, and directs the browser to the proxy.
2. The user types in his user name which is subsequently forwarded to the proxy.
3. The proxy chooses a random word and sends it to the browser to be shown as the session name.
4. The proxy also looks up the user's cell phone number based on his user name, and sends the user the same word and a link to a dynamically-generated page on the proxy via SMS. This page is viewed on the cell phone and gives the user the ability to allow or deny the session
5. The user checks the session name displayed on the untrusted computer's browser.
6. The user also checks the session name displayed on the mobile device to find out if they match.
7. Upon verification, the user can allow or disallow the session. When the user is authenticated, he can use the remote service, and the proxy is transparent to the user.

The authors believe that the security of their system relies on the security of SMS messages which uses A5/1 [56]. However, in 2006 Barkan and Biham [57] broke the A5/1 algorithm, which leaves the system vulnerable. Moreover, the architecture requires a trusted proxy where all user credentials are stored. If the proxy is compromised, all users will be exposed to great danger.

Jonathan M. McCune et al. presented **BitE** [24] in 2006. BitE is an approach to protect user sensitive data input against user-space malware, and to give users additional confidence that user input will be given to user-specified applications. Basically, user input is sent via a trusted tunnel from a trusted mobile device to some specified application on a computer. The authors believe that their approach is resilient to keyloggers, screen grabbers and malware such as Trojan horses and spyware; however, the assumption is that the operating system kernel of the computer is not infected with malware. There are some hardware and software prerequisites necessary for BitE to work, which are as follows:

- The computer or computing platform must be equipped with a Trusted Platform Module (TPM) [58].
- The Bios and operating system must be TPM-enabled, and an integrity measurement architecture (IMA) [59] must be incorporated in the operating system. Thus, the operating system can perform integrity measurements of code loaded for execution.
- A mobile device and an external keyboard are necessary: a mobile device's display and an external keyboard serve as the output and input channels. The connection between the external keyboard and the mobile device is encrypted.
- Two pieces of code: the BitE Kernel Module and the BitE Mobile Client

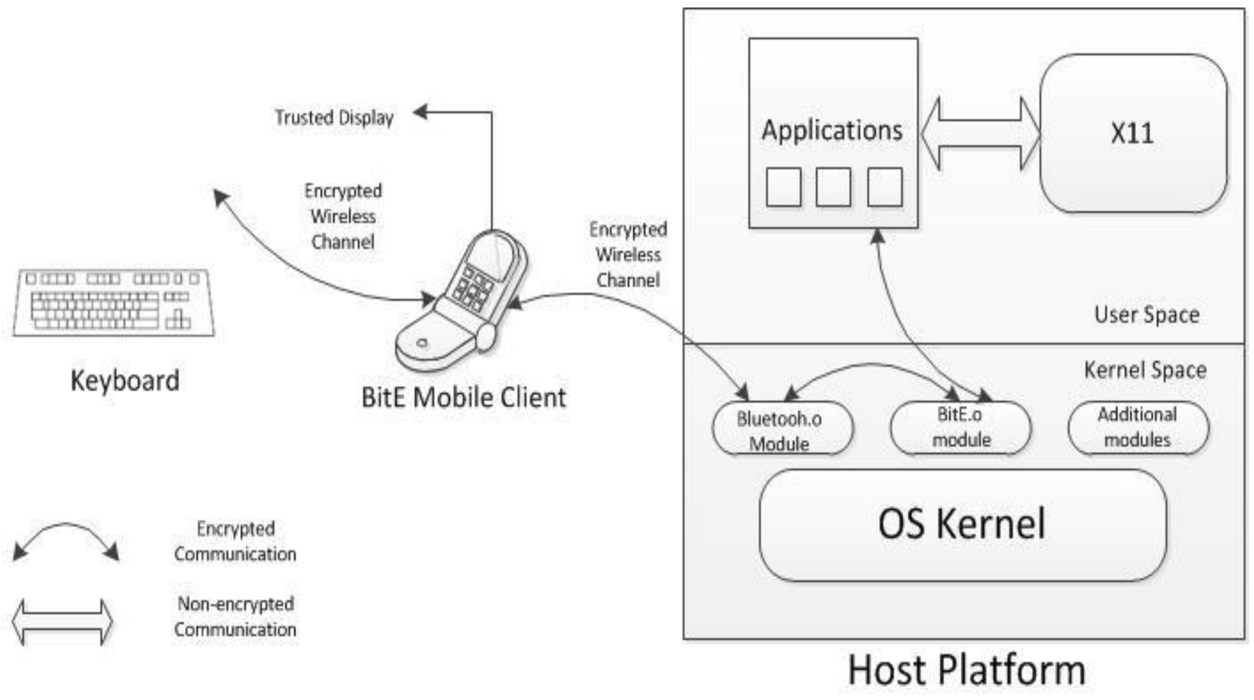


Figure 2.2: BitE Architecture

BitE utilizes two features of a TPM-equipped computer: sealed storage and attestation. A TPM can generate a SRK (Storage Root Key) which never leaves the chip, and any data which is needed to leave the TPM chip in order to be stored on a different storage, can be encrypted with this SRK. Thus, many keys can be stored encrypted on different media when they are not in use. On the other hand, attestations, which are produced by a TPM and an IMA in the operating system, are basically integrity measurements (cryptographic hashes) of any code loaded for execution. An attestation has two parts: a list of all measurements of code to be executed and a list of PCR registers values signed with an RSA key called the AIK (Attestation Identity Key) which is maintained by the TPM. With a public component of the AIK, a verifier can re-compute the values of PCR registers which have been used at the time of signing and check to see if signatures match.

BitE uses two pieces of software: the BitE Kernel Module and the BitE Mobile Client, and an association needs to be created between these two components. This association serves two purposes: first, the BitE Mobile Client can verify attestations from the computer and second, the keys, used to encrypt communication tunnels between the BitE Kernel Module and the BitE Mobile Client, are negotiated.

The BitE Kernel module is installed on the computer, and all the applications to be used with BitE must be registered with it. The registration process for an application is, simply put, to compute measurements of the application and its required libraries and dependencies, and then to store them on the IMA measurement list. This list will be signed with the private signing key of the AIK. For the verification purpose, the BitE Mobile Client, which is equipped with the public AIK key, can verify the list. In fact, each application's integrity measurements (cryptographic hashes) are used as access control for application-specific cryptographic keys with which the encrypted tunnel between the application and the user input on the mobile device is established.

After establishing the association and completing the registration process, the end-to-end trusted tunnel between the application and the user data which is input on the mobile device can be established. As can be seen in figure 2.2, when the user intends to run an application, a message is sent by the BitE-aware application (if an application is not BitE aware, a wrapper here called the BitE Wrapper can be used) to the Bite Kernel Module on the user's computer to register an input-event callback function. Then, the attestation which is produced in the registration process will be checked by the BitE mobile client to verify that the trusted tunnel is established with the same application which was initially registered with the BitE Kernel Module. Upon the verification, user must also verify on the mobile device that the application is, indeed, the intended application in order to make sure that the user will interact with the right application. Then the tunnel will be established.

It should be noted that there is a separate tunnel per each application; cryptographic keys are only known to individual applications and the BitE Mobile Client. In other words, when a user presses a key on the external keyboard which is connected to the mobile device, the value of the key (or the event) is encrypted and sent to the application where it is decrypted. Consequently, eavesdropping on keystrokes would not expose the user to threat as the communication is not understood by an external party. Furthermore, BitE makes sure the application which the user wants to give input to, is the right application by the use of attestations and also the user's explicit choice of the application on the mobile device. In other words, in addition to choosing to run the application on the computer, the user also has to explicitly choose the same application on the mobile device (BitE Mobile Client). As a result, if some malicious code even changed the content of the computer screen in order to deceive the user into giving data to a malicious application, the user could figure it out.

Certain aspects and assumptions of BitE may be perceived as shortcomings which include:

1. One of the assumptions is that the mobile device is not compromised. According to [4], the total number of mobile vulnerabilities discovered in 2011 almost doubled in comparison to 2010, which leaves us doubtful about the honesty of mobile devices and the validity of this assumption in real world computing.
2. The OS kernel on the computing platform must be trusted. One form of malware that has the capability to modify the operating system is called rootkits [60] which are a growing problem [4]. With the use of a secure bootstrap architecture such as AEGIS, this issue can be resolved [61].
3. The use of external keyboard with the mobile device might have negative effects on usability.
4. Not many computers are delivered with TPM chips, nor is the use of TPM chips allowed in all countries [62].

Mannan et al. proposed Mobile Password Authentication (**MP-Auth**) [34] in 2007 in an effort to protect sensitive data input and as a defence mechanism against phishing. MP-Auth uses a hand-held personal device such as a cell phone or a PDA (Personal Digital Assistant [63]) to encrypt a tunnel from the cell phone to a server. In their protocol, they have assumed that the personal computer (the browser and perhaps the operating system) is not trustworthy, and that the hand held device is trusted. In this fashion, credentials and long-term passwords are not input on the computer; they are conveyed to the server via a secure tunnel which is established between a hand-held device and the server. The hand-held device in MP-Auth does not use an out-of-band channel; it does all the communication through the computer.

However, the communication between the hand-held device and the server is encrypted separately, and a malicious or curious computer might not stand much chance of extracting useful information. One of the assumptions in MP-Auth is that the initial password setup phase has been accomplished via an out-of band channel. In other words, in this protocol no facilities have been incorporated to secure users' initial registration and password setup.

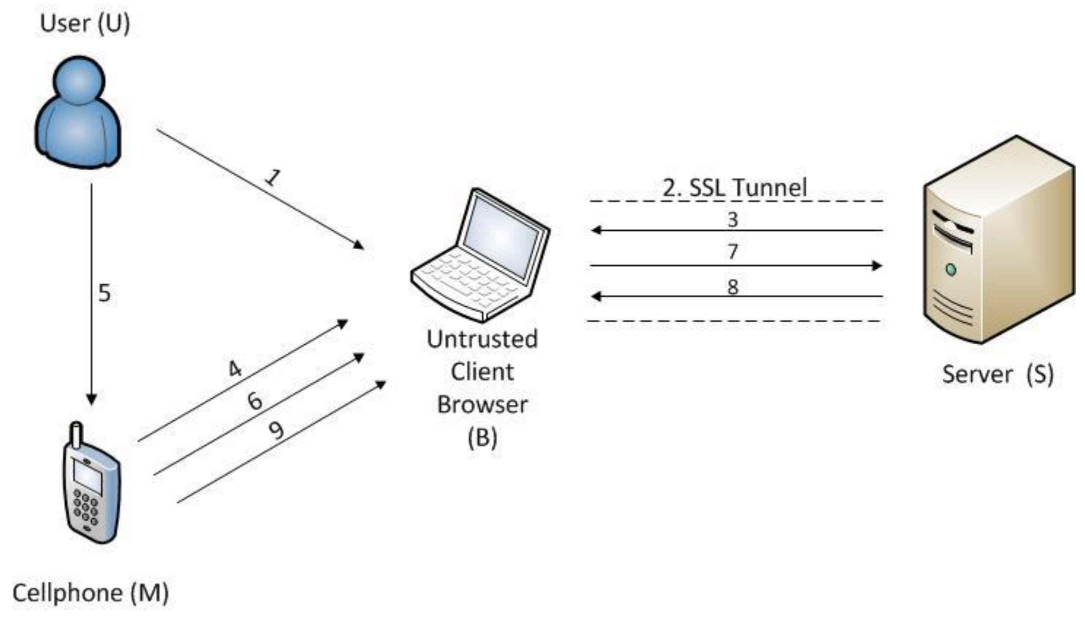


Figure 2.3: MP-Auth

The following notation has been used in the original protocol:

- M, S, U, B: a hand-held device, a server, a user, a browser on the user's computer
- ID_U and ID_S: user ID and server ID. In the server domain, ID_U is unique.
- R_S: a random number generated by S.
- P: a password shared between U and S.
- $\{data\}_K$: data is encrypted symmetrically using key K.
- $\{data\}_{Es}$: data is encrypted asymmetrically with server S's public key Es.
- X, Y: X concatenated to Y
- K_{BS}: symmetric encryption key between S and B(could be an SSL key)
- v(.): a function which maps any binary sting of bits into easy-to-read words
- f(.): a secure hash function

1. U opens browser B on the computer to visit website S.
2. The connection is established between B and S. This connection is SSL-encrypted, and the symmetric secret key is called K_{BS}.
3. S sends its ID_S along with a generated nonce R_S to B in the following format:

$$B \leftarrow S: \{IDs, R_s\}_{KBS}$$

4. Upon receipt of the message, B decrypts it, and forwards the following to M:

$$M \leftarrow B: ID_S, R_S$$

5. To avoid and detect change to R_s by a malicious B, this step is necessary; however, it is optional in the protocol. M and B calculate a session ID in this manner: sid = v (R_s). Then, the results are displayed to U both on the computer and the hand-held device. If they match, the protocol proceeds.
6. On the hand-held device, the user is prompted to input his ID and password for the website S. This user ID can be anything: a web mail account, an online bank number, etc. For users' convenience, IDs can be stored on the hand-held device; however, it is not recommended to store passwords.
7. User inputs his ID and password. Subsequently, M generates a random nonce R_M and the session key K_{MS} and sends the following message to B:

$$K_{MS} = f\ (R_S, R_M)$$

$$M \rightarrow B: \{R_M\}_{ES}, \{f\ (R_S), ID_U, P\}_{KMS}$$

8. Upon receipt of the message, B encrypts it with K_{BS}, and forwards it to S.
9. S decrypts the message with K_{BS} to extract the message sent by M. Then, using its private key, it decrypts R_M, and calculates K_{MS}. Using K_{MS}, the rest of the message is decrypted and P, ID_U and f (R_S) are taken out, and verified. If the verification succeeds, access will be granted to the user, and S sends the following message to B to be relayed to M:

$$B \leftarrow S: \{\{f\ (R_M)\}_{KMS}\}_{KBS}$$

10. B decrypts the message and forwards the result to M. M, in turn, decrypts the message with K_{MS} and take out f (R_M). Then, it checks to see that its local R_M matches with the R_M sent by the server. Upon verification, M displays success or failure to the user on its screen.

As mentioned above, in MP-Auth, sensitive data is transferred via a separate encrypted tunnel between the trusted mobile device and the server. In the sensitive data exchange, the computer only plays the role of a medium between the mobile device and the server. Thus, even if the computer were infected with malware, the communication between the mobile device and server would look like gibberish to the malware. Nonetheless, as mentioned in the previous section, today's sophisticated malware might not need users' credentials to steal from users, and perform unauthorized transactions. Man-in-the-Browser Trojans can wait until the authentication phase is over, and then proceed with unauthorized transactions. To ward off this threat, MP-Auth has an additional feature called Transaction Integrity Confirmation, which augments the original protocol. Using this feature, MP-Auth requires the user to explicitly confirm each transaction on the hand-held device. Let T be the transaction summary details sent by the server, and R_{S1} be another generated random nonce by the server S in order to prevent replay attacks. To fulfill this phase, the following are exchanged between M, B and S:

$$\begin{tikzcd}
M & B \arrow[l, "\{T, R_{S1}\} K_{MS}"'] & S \arrow[l, "\{\{T, R_{S1}\}_{KMS}\}_{KBS}"'] \\
M \arrow[r, "\{f\ (T, R_{S1})\} K_{MS}"] & B \arrow[r, "\{\{f\ (T, R_{S1})\}_{KMS}\}_{KBS}"] & S
\end{tikzcd}$$

S sends M through B the transaction details T along with a new random number R_{S1}, and asks M to confirm it. M, in turn, displays T to the user and asks him to verify the transaction. Upon confirmation by the user, M hashes T and R_{S1}, and sends it back to S via B. Subsequently, S verifies T and R_{S1} sent by M; consequently, if the verification is successful, the transaction will be authorized, otherwise it will be denied.

For password renewal, after user U's successful login, user U has to complete the following step:

$$\begin{tikzcd}M \arrow[r, "{X, \text{ where } X = \{ID_U, P_{old}, P_{new}\}_{KMS}}"] & B \arrow[r, "\{X\}_{KBS}"] & S\end{tikzcd}$$

If P_{old} (the old password) matches U's old password, P_{new} (the new password) will be set as the user's new password.

The authors of MP-Auth believe that use of this protocol for banking and other sensitive data exchange can bring about protection against keylogging, phishing and pharming [64] attacks. However, the initial assumption is that the hand-held device is malware-free, which is not a safe assumption. Moreover, the prerequisite of initial out-of-band account setup makes it difficult for online websites and services which, unlike banks, do not have brick-and-mortar infrastructure, and can hardly provide services in person, or do registration process offline.

Wang et al. presented **PwdCaVe** [65]in 2008 which is a password caching and verifying approach based on the use of TPM. Their approach's novelty relies not on the use of TPM, but rather on keeping passwords away from memory. In fact, some password managers only use TPM to encrypt/decrypt passwords and to store passwords encrypted in the protected storage (the medium used to store inactive keys and other data in an encrypted format). Nonetheless, every time the user would like to perform authentication to a web server, the corresponding password must be decrypted and delivered to the browser or other applications, so that it could be used for verification purposes. Although using such password managers could improve system security against keylogging attacks, they would not achieve much when the attacker has access to the computer's memory or when the computer is compromised. In fact, through taking memory dumps [66] and using applications that allow attackers to view the content of memory such as HeapMemView [67], attackers can successfully recover user passwords and sensitive data from memory.

PwdCaVe uses a novel way not to deliver passwords in plain-text format to applications; thus, passwords are not exposed to conventional threats associated with unauthorized memory dumping and malware. This is achieved through a series of challenges and responses between the TPM and a web server directly. As a result, all the password verification operations are handled within TPM, which eliminates the need to deliver passwords to untrusted applications on the computer.

To cache passwords, PwdCaVe makes use of the authorization data (authdata) field of objects in the TPM. When passwords are not needed, they are stored encrypted in protected storage. In order to not release passwords to memory, the authentication phase must be done by TPM; however, given the limited capabilities of TPM, this cannot be achieved using regular methods. PwdCaVe uses OIAP (Object Independent Authorization Protocol) [58] to verify a password which is stored as the content of the authdata field of an object. OIAP is originally used to check whether an external party is authorized to issue a command to access an object in TPM by making sure that the external party has the knowledge of the authdata field of the corresponding object. This is indeed achieved by hashing the command message using HMAC [68] with the authdata field content as the secret key. Then, the HMAC code is appended to the command message. Upon receipt, TPM does the HMAC calculation with the object's authdata field as the secret key. If these two hash values match, the external party will be authorized to access the object; otherwise, the access will be denied. Thus, using OIAP, the server can checks whether or not the user is really the genuine user or an impostor.

In PwdCaVe, three parties are involved in the authentication process: the TPM on the user's machine, the client software used to access the TPM, and the server. The communication between the TPM and the server are relayed by the client software. The password verification process is as follows:

1. In order to verify the password, the client software issues a command to the TPM to load the intended object whose authdata field holds the password for the intended server.
2. The client software also sends an "OIAP" command the TPM to create an OIAP authorization session.
3. The TPM starts a new session, and it generates a random number n_{even}, and sends it along with the handle of the session called *authhandle*. It should be noted that the random number is used as protection against replay attacks [69].
4. The client software sends *authhandle* and n_{even} to the server. Upon the successful receipt of the two values, the OIAP session between the server and the TPM will be established.
5. The server generates a random number n_{odd} ,and then creates the command message *comm*, and computes its HMAC code $auth_c$ with the password as the secret key.
6. The sever sends n_{odd}, *comm*, $auth_c$ to the client software.
7. Upon receipt, the client software passes these values to the TPM.
8. Subsequently, the TPM checks *comm*'s HMAC code $auth_c$ with the intended object's authdata field. If they match, the command will be executed. The result of the command *ret* and its HMAC code $auth_{r'}$ (using the password as the secret key) along with a new randomly-generated number $n_{even'}$ will be prepared.
9. The TPM returns $n_{even'}$, *ret*, $auth_{r'}$ to the client software.
10. The client software, in turn, relays these values to the server.
11. The server checks *ret* and verifies its HMAC code $auth_{r'}$ based on the corresponding password. If $auth_{r'}$ verifies, access will be granted to the user.

As can be seen, during the whole password verification process, the password does not leave the computer or the TPM in plain-text format. Furthermore, there is no need to fetch the password from the TPM and load it onto the memory, since PwdCaVe bypasses the necessity of divulging the password to untrusted applications on the computer. However, a major drawback of this protocol might be that the password should be stored in clear-text format on the server; because, every time the user intends to log onto the server, the server should use the user's password for HMAC code verifications. Moreover, due to the unencrypted communication between the TPM and the server, PwdCaVe might still be susceptible to offline dictionary attacks [70]. It is well-known that users tend to choose weak and predictable passwords [70] and salting techniques [71] could not apply to this scenario. As a result, an attacker could eavesdrop on the communication, and captures data exchanged between these two parties. Then, having the plain-text message and its hash value, he could perform an offline dictionary attack and might recover the user's password.

Bryan Parno et al. presented **Phoolproof** [35] in 2006. Phoolproof is a system designed to thwart keylogging, phishing and Man-in-the-Middle attacks through mutual authentication between the user and the server. A trusted device such as a PDA or a cell phone is needed to act as an additional authenticator which stores passwords, and plays an active role both in the setup and authentication phases. Their approach, in fact, reduces reliance on the correct decision of the user to do the right thing where security is concerned. In other words, the design of this protocol is such that after the setup phase, the user is automatically protected against the aforementioned threats, and even unwitting users cannot be exposed to danger unless they let go of their trusted device and its access code. It should be noted that the assumption in this protocol is that the additional device, which could be a cell phone, is malware-free; therefore, it is trusted. This protocol has two phases: the setup phase and the connection establishment phase. There is only one setup phase per server; however, the second phase is always used whenever the user would like to establish a secure connection to the server.

The setup phase requires the user to get a shared secret (randomly chosen and of enough length and entropy) from the server through some out-of-band channel such as a mail or even in person. Of course, this could be done through different methods, the discussion of which is beyond the scope of this thesis. For the sake of this writing, we assume that the user has established a secret key with the website from which he wants to receive service. Then, the following takes place:

1. The user launches a browser and visits the website in order to start the setup phase. The connection between the browser and the server is SSL [47]encrypted.
2. The user inputs his user information onto the website.
3. The server sends back a message containing a crafted HTML tag (e.g., <!—SECURE-SETUP -->) signaling to the browser that the account setup has been started, and a MAC [72] of the server's SSL/TLS certificate using the shared key η as the secret key to authenticate itself to the user.
4. The browser contacts the cell phone and relays what has been received in the previous message along with the site name and the domain name.
5. Upon receipt, the cell phone notifies the user of the account creation, and prompts the user to enter the shared secret η established through the out-of-band channel. Then, the cell phone calculates the MAC of the server's certificate using the shared secret η as the secret key, and compares it with the MAC sent by the server. If they match, the process will continue; otherwise, the process will be aborted.
6. Upon successful verification, a private/ public key pair {K_1, K_1^{-1}} is created along with a record associating the key pair with the certificate of the server. In addition, the application on the cell phone will create and store a secure bookmark entry for the site. This bookmark entry is used to thwart phishing attacks.
7. The cell phone sends the server (through the browser), the public key of the key pair just created along with the MAC of the public key concatenated to the server's certificate using the shared key η as the secret key. This MAC is used as proof of authenticity to the server. The server, in turn, calculates the MAC, and checks whether both MACs match. If they do, the account for the user will be created and the public key will be associated with the user.

After account creation and activation, the user can only authenticate himself to the server using both the combination of username/password and the private key, the public key of which is stored on the server. Therefore, even if an attacker would gain access to the user's username/password, he could not access the user's account, since he is not in possession of the key pair established between the cell phone and the server. In order to access the account, the user should start the connection on the cell phone. As mentioned earlier, in the registration process, a secure bookmark entry is created for the website, and the user must use the bookmark to initiate the connection. The secure connection establishment steps are as follows:

1. The user selects a secure bookmark previously stored on the cell phone to initiate a secure connection, and subsequently the cell phone directs the browser to the web site URL.
2. The server sends the browser its certificate along with Diffie-Hellman key agreement [73] parameters and the signature of the Diffie-Hellman key agreement parameters.
3. Then, the browser sends the certificate and the domain name to the cell phone. Next, the cell phone checks to see whether or not the certificate for this domain matches the one it has stored at the time of registration. This step is done to ward off phishing attacks performed on the client browser. In other words, if the certificate does not match the one which has been used at the time of account registration, it can be indicative of a spoofed URL and the cell phone closes the browser, and notifies the user.
4. If the certificates match, the cell phone signs the certificate with the corresponding key K_1 for the website, and sends it to the browser.
5. The browser, in turn, sends the cell phone message *h* which is as follows: h = MD5 (k || pad2 ||MD5 (HM || K || pad1)) ||SHA-1 (K ||pad2||MD5 (SHA-1 ||k ||pad1))

   k is the master key of SSL/TLS tunnel derived from the Diffie-Hellman negotiation, and HM is all of the handshake messages exchanged between the browser and the server along with the browser’s choice of Diffie-Hellman key material. MD5 [74] and SHA-1 [75] are both hash algorithms. || denotes concatenation.
6. Upon the receipt of *h,* the cell phone calculates the signature of *h* using the corresponding K_1^{-1}, and sends it to the browser.
7. The browser sends the server the client’s certificate, its Diffie-Hellman key material and the signature of *h*.
8. The server and browser finalize the SSL/TLS connection negotiation phase.

Once all the above steps are completed, the browser can be used to perform transactions and other activities. It should be noted that SSL/TLS protocol remains intact in Phoolproof, and the cell phone is only used to authenticate the user and to sign the master key. In addition, in case the cell phone is lost, broke or needed to be replaced, the recommended method for the user to revoke the old keys is through traditional phone calls. In other words, if the user happens to lose his keys, he should call the company and ask them to revoke his old keys. Then, he needs to sign up again, and get new keys.

In 2012**, Sun et al.** presented **oPass** [76]. oPass is an authentication protocol designed to thwart attacks which are associated with inputting passwords onto untrusted computers in order to access websites. The authors believe that the burden to remember passwords for users has two negative effects in terms of security. First, users tend to choose weak passwords, which gives the attacker a better chance to crack the user's password through dictionary attacks [77]. Second, users have tendency to reuse passwords (the same passwords, or some variations of one single password) for different websites. In [78], Florencio and Herley pointed out that a user, on average, reuses a password across 3.9 different websites, which substantially increases the likelihood of success of password reuse attacks. Consequently, oPass tries to place this burden of strengthening on the software rather than the user in order to avoid such negative implications. oPass utilizes the user's cell phone along with SMS (short message service) as the out-of-band communication channel. In addition, it requires the website to have a unique phone number, and also requires a telecommunication service provider to take part in the registration and recovery processes. Through this protocol, users only have to memorize one long-term password to access all websites, and the rest of the authentication process is taken care of by the software on the cell phone.

oPass uses the one-time password method presented in [79] in 1981. In order to prepare one-time passwords, a hash chain of the given input c constructs the passwords. The construction is as follows:

$$\delta_0 = H^N (c)$$

where N is the number of one-time passwords needed, and H is a secure hash function. As an example, if we want to prepare 3 different one-time passwords, we must calculate three hashes of the input in a chain-like manner, which is as follows:

$$1.\ \delta_0 = H(c),\ 2.\ \delta_1 = H(\delta_0),\ 3.\ \delta_2 = H(\delta_1)$$

Of course, one-time passwords are used in reverse order, since calculating δ_{N-1} from δ_N is not feasible; however, calculating δ_N from δ_{N-1} is easy in theory. The input c is the hash value of the concatenation of the identity of the server (ID_S), a server-generated random seed (ϕ) and the user's long-term password (P_u), which is as follows:

$$c = H(P_u \,||\, ID_s \,||\, \phi)$$

The *H* above represents a secure hash function. The hash function used by oPass is SHA-256 [80].

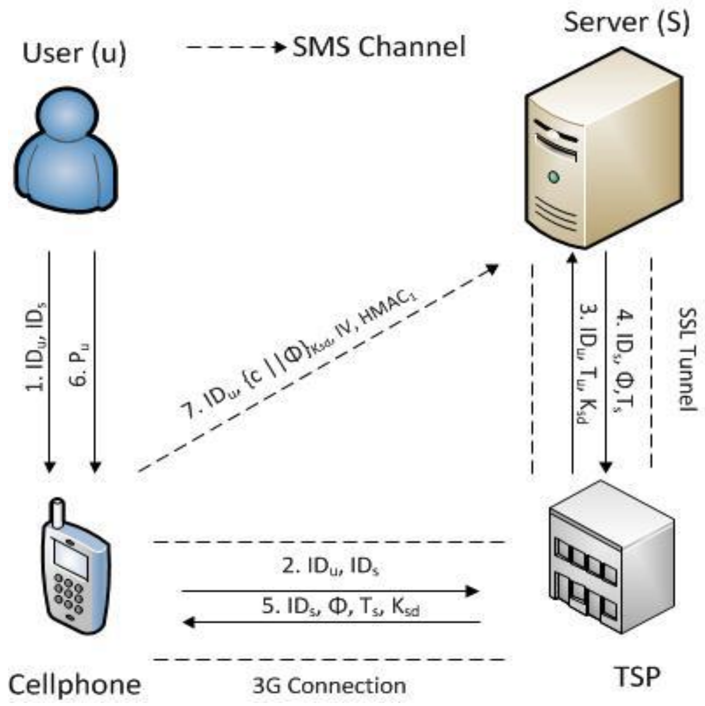


Figure 2.4: oPass Registration Phase

The registration phase is the process through which a user can sign up to use a website, and to establish a shared secret between the user and the server for further logins. Four parties participate in the process: the user (u), the server(S), the cell phone and the TSP (Telephone Service Provider). The registration phase is as follows:

1. The user launches the oPass application on his cell phone, and enters ID_u (the identity of the user), ID_s (the identity of the server, which could be its URL) into the program.
2. The mobile application sends these parameters to the TSP through SMS to make a request for registration. The TSP also makes a record of the user's phone number (T_u) based on his SIM card.
3. The TSP then chooses a shared key K_{sd}, and sends it along with the user's ID_u and T_u to the server through an SSL-encrypted connection. The shared secret K_{sd} is used to encrypt the registration SMS message.
4. The server, in turn, sends the TSP a random numberϕ, ID_s and T_s (the server's phone number).
5. The TSP forwards to the cell phone ID_s,ϕ, T_s and K_{sd}.
6. Upon the receipt of the message from the TSP, the user creates a long-term password P_u on the cell phone and the cell phone computes the secret credential c through the following function:
$$c = H(P_u \,||\, ID_s \,||\, \phi)$$
7. Subsequently, the cell phone sends the server ID_u, $\{c \,||\, \phi\}_{Ksd}$, the initialization vector IV and HMAC-SHA1 digest of $ID_u \,||\, IV \,||\{c \,||\, \phi\}_{Ksd}$ with the K_{sd} as the secret key through SMS. It should be noted that the server's phone number was obtained through the previous step. Moreover, to maintain communication security, the SMS message is encrypted using $K_{sd.}$

Upon the receipt, the server decrypts the message, and verifies the message digest. If the digest calculated matches the digest received, it verifies the source of the SMS message with regard to the T_u received from the TSP in step 3 in order to prevent spoofing attacks. Then, the server stores the values: $\{ID_u, T_u, c, \phi, i\}$ for the user. The values stored on the user's cell phone are {ID_s, T_s, ϕ, i}. i is the current index of the one-time password (initially set to 0). Also for security reasons, the long-term password P_u is not stored on the cell phone.

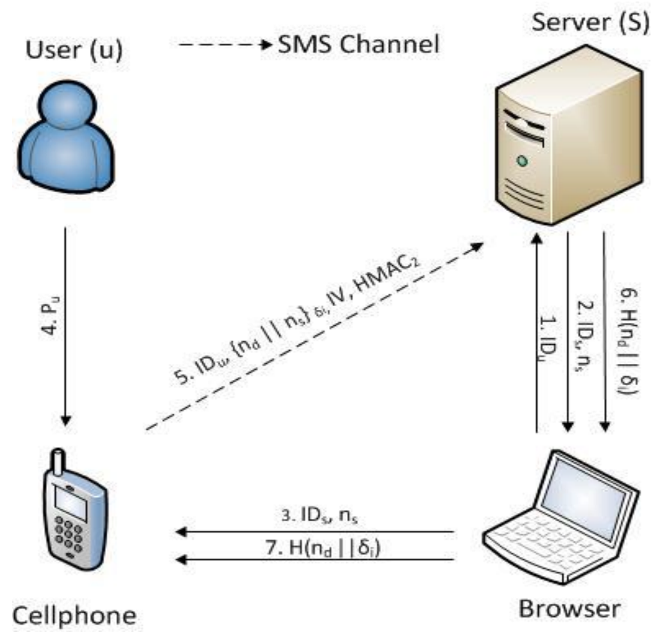


Figure 2.5: oPass Login Phase

In order for the user to use the website, he should have his cell phone at hand, and initiates a browser on a computer. The steps to carry out the login phase are as follows:

1. The user initiates a browser, visits the intended websites and inputs his ID_u.
2. The server generates the nonce n_s and sends it along with its ID_s to the browser.
3. Upon the receipt of the message, it is forwarded to the cell phone through wireless or Bluetooth interfaces.
4. The cell phone, then, looks for the data associated with the server's ID_s on its local storage. Upon successful lookup, it takes out the server's phone number T_s, ϕ and i. Then, it prompts the user to input the long-term password.
5. Subsequently, the following computations are done by the cell phone:

$$c = H(P_u || ID_s || \phi)$$

$$\delta_i = H^{N-i}(c)$$

As mentioned earlier, δ_i is only used for i^{th} login. The cell phone also generates a new nonce n_d. The cell phone encrypts $\{n_d || n_s\}$ with δ_i. Next, it sends the server ID_u, $\{n_d || n_s\}_{\delta i}$, IV(the initialization vector) and their message digest $HMAC_2$.

6. Upon the receipt of the login SMS message, the server generates δ_i, and decrypts the SMS message. If the n_s received matches the n_s which the server has initially sent to the browser, the user will be granted access. Otherwise, the server will terminate the connection. As soon as the server verifies the authenticity of the communication and the user, it sends the browser the hash value of $(n_d || \delta_i)$.
7. The browser forwards the message to the cell phone, and the cell phone verifies the message. This step is needed to thwart phishing and Man-in-the-Middle attacks. If the verification fails, the user will be notified and the value of i will not increment. However, if the user logs in successfully, the i will be incremented both on the server and on the cell phone. After N-1 successful logins, the server resets the random numberϕ, and the user should go through the recovery phase.

If the user loses his cell phone, or the value i reaches the pre-determined number of logins, the user must go through the recovery phase. This process allows the user to recover his oPass lost data, and regain access to the website. The recovery phase is as follows:

1. The user installs the oPass application on his cell phone (if he does not have the application already), and inputs his ID_u and the server's ID_s and places a recovery request on the oPass application.
2. The cell phone sends the IDs along with the request to the TPS.
3. The TPS traces the user's phone number and sends ID_u and T_s to the server through a SSL-encrypted connection.
4. Upon the receipt of the message, the server looks up the ID_u to make sure that this user exists. If the result is positive, the server will extract the data which has been used to make the secret credential c for the user. The server also computes a new nonce n_s and sends back the TPS the following: ID_s, ϕ, T_s , i, and n_s.
5. The TSP forwards the message received from the server.

6. Upon the receipt of the message, the cell phone prompts the user to input his long-term password. Using i received from the server, the cell phone computes δ_{i+1} and encrypts {c || n_s} with δ_{i+1}.
7. Finally, it sends ID_u, $\{c \,||\, n_s\}_{\delta i+1}$, IV (the initialization vector) and the hash value of them all $HMAC_3$ to the server. The server, in turn, decrypts the message with δ_{i+1} and verifies the content. If the content verifies, the server will know that the recovery phase has been successful, and that the next time the user must use δ_{i+2} as the password. If an attacker impersonated the user, he would not get any confirming information from the server regarding success or failure of the recovery phase. This could help thwart password guessing attacks.

It should be noted that the reason a TSP is used both in registration and recovery processes is to make sure that the phone number T_s, which the cell phone receives for the server S, is not from a phishing website. Using oPass, users are able to have one long-term password for all websites; moreover, due to the use of random numbers in secret credentials computation, oPass reduces the negative impacts of weak and reused passwords of the user.

PwdHash [81] was presented by **Ross et al.** in 2005. PwdHash is a browser extension which is installed on the client machine, and produces strong passwords based on the user-chosen password and the website domain name. For each website, these two values are fed into a hashing function to generate the appropriate password for the intended website. The password generation is performed as follows:

$$\text{Hash (pwd, dom)} = \text{PRF}_{pwd}(\text{dom})$$

where PRF is a pseudo random function [82], *pwd* is the user's password, and *dom* is the website's domain name. In fact, the pseudo random function is keyed by the password, and is input with *dom*. As a result, the user has to remember only one master password for all the websites that he uses. No changes are also necessary on the server side, since the computation is carried out on the client machine, and the hashed passwords are further tailored to meet the password encoding rules of websites. Furthermore, a website [83] is provided by the authors to help users generate passwords using their master password and the website's domain name when users do not have access to their browser extension. Passwords, generated by the website, are subsequently copied and pasted into the password fields of intended websites.

The authors mention a number of JavaScript attacks such as JavaScript keyloggers which are warded off by the browser extension; nonetheless, they do acknowledge that their extension is vulnerable to malware, keyloggers and other competing browser extensions which log keyboard events the same way that PwdHash does. The authors also believe that their method can deter password phishing, since the password received at a phishing website cannot be used further at any other website.

Password Multiplier [84] was presented by Helderman et al. in 2005. Password Multiplier is a scheme which is based on one single master password along with a site name and the application of iterated hashing to it. The construction of passwords for sites is carried out as follows:

Let f(x) be a secure hash function,

$$V = f^{k_1}(\text{username: master-Password})$$

$$\text{Site-password} = f^{K_2}(\text{site-name: master-password: V})$$

As can be seen from above, both V and Site-password are iterated K_1 and K_2 times in order to ward off brute-force attacks. The username can be any string that the user wishes to choose. The iterated hashing used to generate and harden site-specific passwords could incur drastically more effort and time to crack the master password. However, all hash-based systems such as the above approach and PwDHash [14] are susceptible to offline dictionary attacks [84]. An adversary would just have to entice a victim to register on a website belonging to the attacker or which has been compromised by the attacker. Afterwards, the attacker would perform offline guessing attacks on the hashed password and might be able to crack it. Nonetheless, by adding V to the input of the Site-password, the above scheme has taken further measures to improve the security of the user password. That is, even if the attacker could manage to extract the master password, he would not be able to use it on other websites without the knowledge of the victim's username. As a result, the adversary has to do more work to crack that as well.

The aforementioned adversary, of course, is assumed to not have access to the client machine. On the client machine, the values K_1, K_2 and sometime V (since it is independent of any specific website) should be stored in order to make it possible for the system to generate the site-specific passwords again. Therefore, having access to the client computer, the probability of breaking into the victim's account would increase. The worst-case scenario would be the case where the adversary had both a site-specific password and had compromised the client machine. This would indeed decrease sharply the time and the password space needed to be searched to take out the master password.

Bojinov et al. presented **Kamouflage**: Loss-Resistant Password Management [85] in 2010 which is an architecture for building password managers. Kamouflage is aimed at strengthening the password database stored on the client computer against offline dictionary attacks. In order to achieve the aforementioned goal, Kamouflage stores *N* set of user passwords which are comprised of one set of real user passwords along with *N-1* decoy sets. These decoy sets stored encrypted on the client machine are meant to impose more time, effort and processing power upon the adversary to find and crack the password set. However, it should be noted that it is of utmost significance that the real password set be indistinguishable from the attacker's view point. Kamouflage utilizes two key derivation functions:

$$K_i \leftarrow KDF_1 (MP_i + IV_i);$$

$$L_i \leftarrow KDF_2 (MP_i)$$

MP_i is the master password for set S_i and IV_i is stored in clear text in set i. KDF_1 generates the key to each set of passwords, and KDF_2 generates value L_i which determines the position of the password set i in the password database.

What makes Kamouflage different from other password managers is that more than one set of passwords are stored on the computer. Not only does the attacker have to perform several offline dictionary attacks separately on each password set of the whole password database, but he must also determine which set is the real set of passwords. Thus, the adversary must do a substantial amount of online work to determine whether or not he has found the good set.

McCarney et al. presented **Tapas** [86] in 2012. Tapas is a password manager which employs dual-possession authentication, and is implemented as a Firefox plug-in (called the Manager) and a smartphone (called the Wallet) on which user passwords are stored in encrypted form. In this scheme, two asymmetric key pairs and one symmetric key are generated and used by the devices. Each device (the client machine and the smartphone) has their own key pair and the other party's public key in order to identify itself to the other party and also to be able to communicate with each other securely. Moreover, the symmetric key stored only on the client machine is used to encrypt/decrypt user passwords stored on the Wallet. Generation and storage of keys are carried out during the pairing process. User passwords are encrypted and stored as follows:

$$C_i = Enc_k (p_i \,||\, s_i)$$

P_i is the user-chosen password for a specific website, s_i is the site information. In addition, for each website, the user should create a tag t_i for referencing the site. After the generation of t_i and c_i , the Manager will wipe them off the client machine, and send them along with the symmetrically encrypted username to the Wallet via an encrypted channel through a Rendezvous Server. The Rendezvous Server is employed to provide a means of direct communication between the two devices when they are on different networks.

The password retrieval is initiated when the user intends to log onto a website. Tapas requires the user to explicitly choose the account on the smartphone and on the browser (the Manager) through entering the tag t_i for the site. Then, the account information is transferred from the Wallet to the Manager via an encrypted channel through The Rendezvous Server. Subsequently, the Manager decrypts the data, verifies the site information, and populates the username and password fields upon successful verification.

As mentioned earlier, the user credentials are encrypted with a symmetric key and stored on the smartphone while the symmetric key along with the computer's key pair is stored on the client machine. While Tapas is resistant to the theft of the client computer, it is still susceptible to offline guessing attacks in case of smartphone loss or theft of the password database from the smartphone. Moreover, in case of crossover viruses, Tapas is extremely vulnerable, since both the key and the credential database are stored on the devices. In other words, if some malicious code could have access to the both devices, it would not be a big challenge to steal both the credential database and the key to it. To us, this does not sound infeasible, since many people connect their smartphones to their computers in order to transfer files, pictures, music, etc. from the smartphone to their computer and vice versa, which could provide a perfect means of crossover virus infection.

The **Janus** Personalized Web Anonymizer [87] was presented by Gabber et al. in 1997. Janus is a proxy server, which filters the HTTP information flow in order to preserve privacy, and also a password generator which relieves the user from the burden of creating and memorizing usernames and passwords for websites through realization of the Janus Function. The Janus Function produces usernames and passwords derived securely from a master username and password along with the domain names of websites. In order for the user to use the Janus proxy, the browser must be configured to forward all HTTP traffic to the Janus Proxy, and after authentication to the Janus Proxy, the user could just use scape strings such as "\U" for username and "\P" for password. These strings are recognized and replaced with the username and password for the intended website by the Janus Proxy. It should also be noted that usernames and passwords are not stored on the client machine, yet they are generated on the fly by the Janus Proxy. Nonetheless, the Janus proxy can only work with HTTP, which makes it almost impractical in today's online world where a non-HTTPS website requiring authentication can hardly be found.

Furthermore, several commercial and non-commercial password managers such as built-in password managers of web browsers, etc. usually are available which store the password database encrypted on the client machine. The password database is usually locked with a master password which has been already chosen by the user, and the user can access the credential database by logging into the password manager. However, user-chosen passwords usually suffer from lack of complexity. “The problem is that the average user won’t even try to remember complex enough passwords” Schneier says [8]. Of course, some schemes use salts to slow down dictionary attacks on the master password [85]. However, this would add little value to the scheme, since the salt must also be stored on the client machine in order to make it possible to decrypt the database again. Therefore, it would not be hard for a savvy attacker, who had access to the client machine, to steal the salt along with the encrypted password database. Afterwards, he would consequently perform offline guessing attacks on the database to extract the user passwords.

2.5 Chapter Summary

This chapter has introduced the problem of securing user sensitive input. It has also described how a typical Man-in-the-Browser attack could infect and mount the attack on a victim, and eventually how devastating a properly engineered piece of malware such as Zeus could be. This chapter has also provided a review of the literature with focus on protecting the user against sensitive data theft (e.g. user credentials). Some schemes require modification done to the server, and some require additional hardware, which could make them hard to be adopted by the general public. Moreover, we reviewed some previous work regarding password managers and some of their intrinsic characteristics which could be perceived as shortcomings. We have pointed out that password generators such as PwdHash are more subject to offline guessing attacks, since deceiving an unwitting victim into registering on an adversary-controlled website would not be very difficult. Furthermore password managers which store the password database encrypted on the client computer usually suffer from weak master passwords. Thus, we conclude that existing solutions all have limitations and shortcomings. We address this problem in Chapter 5 where we use a distributed scheme to generate and store the master key which is of adequate entropy. The next chapter introduces CredProxy, the design of a password manager to mitigate phishing and Man-in-the-Browser attacks.

Chapter 3

CredProxy

In this chapter, we present the design of our password manager. This password manager is based on a personal proxy situated on the client machine, which is meant to mitigate phishing and Man-in-The-Browser attacks. It also provides a novel way to secure user credentials on the client computer (Function CPGuard presented in Chapter 5). We aim at alleviating key-logging, phishing and MITB attacks, and also making authentication both more secure and less cumbersome for the user. Our goal is to give the user peace of mind along with enough security with current available technologies and without involving additional parties [76] and additional hardware which are not available for everyday users. Our objective, in fact, is to make online authentication more secure and to keep user credentials away from adversaries.

It should be noted that "CredProxy" is both the name of the proxy-based password manager and the name of the whole package including other components such as the protocols etc. Moreover, while we have not implemented the proxy-based password manager, other components discussed in later chapters have been implemented.

3.1 Introduction

In these uncertain times, an internet user is always doubtful about the security of his computer and his accounts. A typical internet user is in constant fear of losing sensitive information while using Internet services such as online banking, email services, etc. According to a study published by Symantec in 2011 [88], cybercrime costs 114 billion U.S dollars per year globally. Of course, this does not count the time lost by companies to recover from the damages; taking that into consideration, we have to yet add another 247 billion U.S dollars to the initial amount. Symantec believe that every second, 14 adults are victimized by cybercrimes, and that there are more than one million cyber-attacks every day. Amongst these cyber-attacks, computer malware (54%) and phishing (10%) are the most common types of attacks. The common types of malware include keyloggers, screen grabbers and spyware, the main purpose of which is to generate revenue for attackers. These attacks are performed with one intention in attackers' mind: to steal information from the victim's computer. This information, indeed, can be anything from user credentials and online banking information to email addresses and web surfing habits. Stolen credentials and banking information can be directly sold in underground markets. According to Symantec [6], stolen credit cards are acquired for 0.07 to 100 U.S dollars in the underground economy. On the other hand, stolen email addresses and illegally obtained web surfing habits are used for spamming and marketing purposes. In 2011 alone, there were 42 billion spam messages on average a day worldwide, Symantec reports [4]. Phishing also has proved effective to steal credentials from a victim. APWG (Anti Phishing Working Group) [89] revealed in its recent report [41] that in the second half of 2011 there were more than 83000 phishing attacks globally in the 200 top-level domains alone. More and more reports indicate that despite vast security measures and increasing user awareness, cyber criminals have yet proved to be ahead of security experts, and they have proved yet able to circumvent complex security measures deployed by banks and other online service providers. Many solutions to protect user credentials have been proposed by researchers, yet not many have been deployed due some limiting factors such as

- Requiring additional hardware on the client side
- Inefficacy or high cost of solutions
- Assumptions which are far from reality

Despite the above mentioned, still many Internet services use the combination of text usernames and passwords, which makes it easy for cyber criminals to commit cyber theft. One good example could be banks. Still, many banks all around the world use text passwords as the primary means of authentication for users. In fact, the only thing a typical bank user needs nowadays is a bank account number and a password to access his account online. This information, as mentioned earlier, is very easy to steal, given the current security measures. An unwitting user could unknowingly enter his login information on a phishing website that looks like the original website, and become a victim of cybercrime. For instance, according to [90], a phishing attack was detected in Brazil which targeted the customers of Santander Bank [91]. In this attack, the cyber criminals managed to perform a DNS cache poisoning attack on several DNS servers forcing them to resolve Santander.com.br to some IP addresses under their control. In such cases, the average user would not have any clue that he were under attack.

Another common case is computer malware. Without an up-to-date antivirus and a fully patched operating system along with IDP in place, a computer could get easily infected with malware, and would not survive long in the wild. The report by Sans [92] reveals that the average time between attacks for an average target IP address is less than 10 minutes. In this case, a client computer could get infected with malware such as keyloggers and screen grabbers, which could result in data theft.

JavaScript keyloggers [14] are yet another issue. JavaScript keyloggers are not like traditional keyloggers. They do not register call back functions to receive key strokes from lower levels. They, in fact, add JavaScript functions to web pages in order to record keyboard events sent to password or any other fields on a web page, and after collection they are sent to attackers. JavaScript keyloggers can be injected into both compromised websites (the discussion of which is beyond the scope of this writing) or into client computers compromised by malware. Zeus [12] is one example of such loggers; it can add fields to a web page on a victim's computer to steal credentials through HTML injection techniques even before passwords are hashed [12]. For the sake of this thesis, we tested the JavaScript keylogger developed by Google [93] on a fully patched Windows 7 computer with an up-to-date Kaspersky Internet Security[3] 2012 installed, and realized, to our astonishment, that, unlike traditional keyloggers, the behaviour of the code is not flagged as malware. Thus, our guess is that JavaScript keylogging is easier than traditional keylogging.

In an attempt to protect user credentials and to ward off the aforementioned attacks, we present the design of our CredProxy. Our idea to protect credentials from the aforementioned attacks is based on a simple idea: to not give the browser real information and to protect real information on the computer. CredProxy is a personal proxy that acts as an intermediary

[3] http://www.kaspersky.ca/

between a client computer and a website. Our model does not allow credentials and online banking information be revealed to an untrusted browser, yet it shoulders the burden of securing credentials on the computer. CredProxy makes sure that credentials are safely and securely transferred to authorized parties. In order to log onto websites, real credentials are not used. In fact, we use an alias user name and password for authentication on all websites registered on CredProxy.

Not only does CredProxy protect credentials from malware and an untrusted browser, but it also stores them securely on the computer; thus, it relieves the user from the cumbersome task of memorizing several online credentials. We propose a novel way to store credentials encrypted on a client computer that increases security and simplifies the retrieval of forgotten passwords. CredProxy realizes Function CPGen and Function CPGuard. Function CPGen automates the generation of strong usernames and passwords. Function CPGuard protects the credential vault on the client computer through a suite of protocols. In these protocols, we propose a method that can utilize a hand-held device such as a smart phone as a token. The study of the current literature made us realize that despite all the newly discovered vulnerabilities and malware on mobile devices, researchers have a tendency to trust mobile devices more than computers. Indeed, this could be a roughly acceptable idea, since with the proliferation of mobile devices, there has not yet been a corresponding rise in the number of mobile malware [4]. However, this is changing now, and the change is dramatic. Juniper reports that the number of Android malware has soared up 3000% in 2012 [94], which leaves us doubtful about the assumption of malware-free hand-held devices.

3.2 Password managers

Password managers are software components which organize usernames and passwords for the user, and store credential databases either locally on the client machine, on another device such as a smartphone or even in the cloud. Many of these password managers implemented as browser plugins also work as form fillers; they fill user credentials into HTML forms. Thus, they relieve the user from the task of memorizing user credentials, and inputting them on websites.

Passwords managers are convenient and usually make online authentication process more secure. However, there are a number of security threats to password managers. First, inputting passwords into the password managers could divulge user credentials. Malware might log keystrokes or grab screenshots while authentication information was input to the password manager. Keylogging and shoulder-surfing threats are addressed in Chapter 6. Second, credential databases which are stored on the client machine are always in danger of theft and offline dictionary attacks. Offline dictionary approaches often bear fruit, since master passwords (to password databases) chosen by users are not usually complex enough and of sufficient entropy. Thus, there is a high likelihood that an adversary with access to the encrypted credential database would be able to crack it. Enhancing the security of password databases is discussed in Chapter 5. Moreover, some password managers do not store password databases locally but in the cloud. In this case, however, the issue is that the user has to trust the third party service provider which stores his password database. A curios service provider might also try offline dictionary attacks on the password database of a person of interest. Furthermore, cloud-based password managers might also give rise to more issues which could be harder to deal with. For instance, the server(s) where password databases are stored could become a single point of failure in the system, or could be compromised resulting in loss of some or a substantial portion of password databases of users. Therefore, the average user might have a tendency to not store his password database on a third-party server.

Lastly, filling out HTML forms by password managers implemented as plugins in browsers could give away user authentication information in cases where attacks such as Man-in-the-Browser and phishing were successfully mounted. In order to alleviate such problems, we have used a proxy-based password manager. Our password manager is resistant to Man-in-the-Browser threats, since no real credential is input in the browser; as a result, an infected browser could not divulge user sensitive data input in the browser. This could, indeed, be perceived as an intrinsic property of personal proxy servers, as the need to interact with the browser for user sensitive data input can be drastically decreased through them. Moreover, a comprehensive checking of SSL certificates presented by websites is performed by the proxy-based password manager in order to ward off phishing attacks. In the next sections, we discuss the design of the proxy-based password manager.

3.3 Proxy Servers

Proxy servers play a key role in our everyday computer networks. They are extensively used for caching, auditing, policy enforcement, link-translation and VPN functions. They act as an intermediary between the client computer and the server, and they are usually transparent to network users. Client requests are usually processed by these entities before being forwarded to intended servers, and this process can affect how client requests are forwarded, or even whether or not they are forwarded at all. Proxy servers are usually categorized into three distinct categories: forward proxy, reverse proxy and intermediate proxy servers.

A forward proxy server is usually situated between the client computer and the origin server. Whenever the client wants to retrieve content residing on the server, the client has to send a request to the forward proxy server instead of sending the request directly to the origin server. Upon the receipt of the request, the proxy processes the request, and makes a decision whether or not to forward it, based on the policy which has been defined on it by administrators. These policies can vary depending on the network requirements. Forward proxy servers usually intercept traffic at the application layer or lower [95] and make forwarding decisions based on the content of packets. The main purpose of these proxy servers are caching, web traffic shaping, content filtering and maintaining anonymity for client computers. As an example, Microsoft ISA Server [96] can act, among other roles, as such forward proxy server.

In contrast, a reverse proxy server sits usually closer to the origin server, and all the requests for the origin server go through it first. In fact, a reverse proxy server presents a single interface to clients [95], and in the client's point of view, there is only one server serving their requests- it is a case where many servers are accessible by a single valid IP address through the network. As a matter of fact, a reverse proxy server receives all requests for the origin server, and then distributes them among a set of backend servers. The backend servers, in turn, are configured to send all the replies to clients through the reverse proxy server. This model can bring about several benefits in terms of security, load sharing and load reduction. For security, a reverse proxy server can hide all the characteristic information about the backend servers, and also prevents clients from having direct and unmonitored access to backend servers [95], thus making servers more secure. It can also perform load balancing among backend servers, assuring that each server receives roughly equal number of web requests. Furthermore, in case of secure websites (websites using SSL/TLS), the burden of SSL/TLS connections can be carried by a reverse proxy server, which leaves less load on the backend servers.

Intermediate proxy servers are yet another type of proxy server. These proxy servers are usually provided by third-parties in the Internet [95], and they sit between the client and the origin server. However, unlike forward and reverse proxy servers, they do not sit in the client network, or on the origin server network. The purpose of this type of proxy servers is mainly to maintain user anonymity, or to circumvent network and browsing policies enforced in a network by its administrators. For instance, a user in a hypothetical network might not be able to browse some specific websites (as an example, “facebook.com” is blocked in China [97]), or might not be able to run some applications that send and receive content from the Internet (applications that use the BitTorrent [98] protocol are an example of such applications). In such cases, an intermediate proxy server can help evade the enforced policy by the network administrators. The connection through an intermediate proxy server is usually achieved by changing the proxy settings in a browser or any application that accommodates proxy configuration settings such as: μTorrent [99].

However, the architecture of the proxy server we use in this thesis is different from the aforementioned models. Our CredProxy is a personal proxy server that resides on the client computer, and acts as an intermediary between the browser and the server; consequently, it provides supervision on the content sent and received by the browser from the Internet. In other words, with personal proxy servers, users can control how much information they want to share, and with whom they want to share it [100]. Personal proxies have been widely used for debugging, content-filtering and privacy purposes [95]. Since all the conversation between websites and a browser has to go through the proxy server, they can examine the traffic and allow or block certain pieces of information such as web bugs, browser cookies, web referrers and so on; as a result, personal proxy servers provide users with more security and privacy.

Privoxy [101]is a well-known example of such a proxy server that runs on various operating systems and supports popular browsers. Privoxy is a non-caching personal proxy server which provides advanced filtering capabilities. It sits between the browser and the Internet, and inspects all the traffic that passes through it, which makes it capable of modifying, adding or removing the content. Privoxy can block advertising content in a webpage, enhance privacy, perform access control and debugging functions, etc. [101].

3.4 CredProxy

Nowadays in the world of the Internet, more and more websites offer personalized services [87], and more and more websites require users to register and perform authentication before they can receive service from them. This requirement might burden our user in two ways. First, the user, who is concerned about his security and privacy in our scenario, might have to create different usernames and passwords which ideally should be both unrelated to his real identity. In fact, the user has to create a username that does not exist in the current user database of an intended website, which might make the username hard both to create and to remember. In addition, it is a well-known fact that users tend to choose passwords which are easy to predict and can thus be cracked through attacks such as dictionary attacks and probabilistic methods [70]. Of course, having a password strength policy in place could prevent users from creating weak passwords; however, many users cannot handle complexity of passwords, which might lead to workarounds as a consequence [102]. Second, even if the user chooses good usernames and high-entropy passwords, the burden of memorizing them for the user might be too much to handle. The user might, then, resort to behaviors analogous to writing passwords on stickers and placing them somewhere near his computer, which, in fact, contradicts the whole idea of password security.

As mentioned earlier, CredProxy is situated between the browser and the server; hence, it captures all the traffic throughout the whole communication with the server. The traffic goes through CredProxy for modification purposes, and then modified data will be sent to the server. However, this whole process is invisible to the server or any other party residing outside the client computer. CredProxy is a credential intermediary. CredProxy automates the generation of usernames and secure passwords (Function CPGen), and enhances the security of the credential database in the face of theft (Function CPGuard). When CredProxy is installed on the computer, the user should create one alias username and a password to use for login purposes on all intended websites. We later explain how this alias username and password are translated to real credentials for an intended website. CredProxy can parse HTML [103]. It can modify the content of form fields such as username and password fields in order to replace an alias username and password with real credentials.

In addition, in order for the user to access the CredProxy on his computer, the user must authenticate himself to the proxy. This authentication step occurs whenever the user runs the CredProxy on his computer. After the execution of the CredProxy application, the user will be prompted to input a 6 digit pin (the number of digits is not fixed, and can be modified based on usability requirements). We use a specific technique to ward off keylogging and shoulder surfing attacks, which could take place in this phase of authentication, through our special virtual keyboard. This keyboard will be explained in Chapter 6. This extra authentication step makes sure that the user is the genuine user, and it blocks out imposters. In other words, if an attacker gained access to the user’s alias username and password through phishing, keylogging or any other means, he would not still be able to use the user’s real credentials since he could not log into the user’s CredProxy application, and decrypt the credential vault.

Whenever CredProxy is run, it changes the proxy settings of the user’s browsers automatically in order to allow all the communications (HTTP and HTTPS) go through the CredProxy. Thus, no proxy setting alteration is required by the user, which can bring a novice user some relief from having to get involved with the setting change. Figures 3.1 and 3.3 depict the situations where HTTP and HTTPS protocols are used with CredProxy.

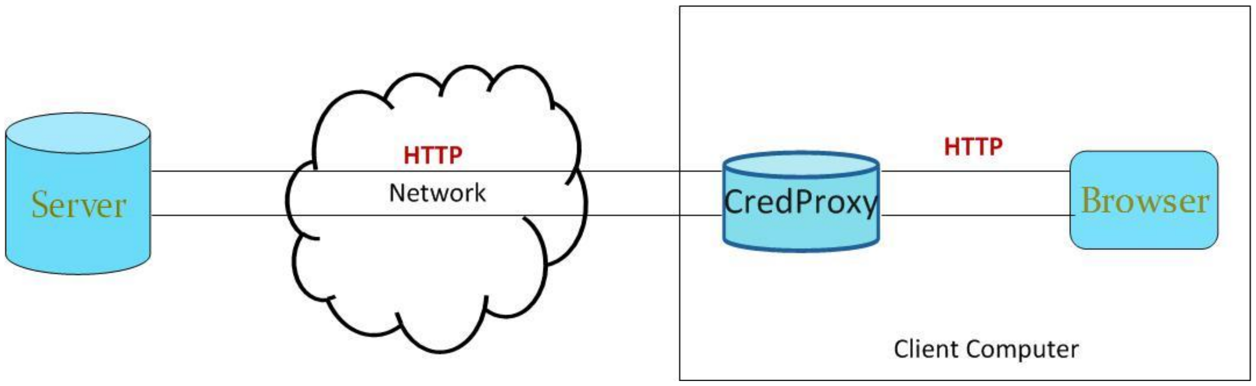
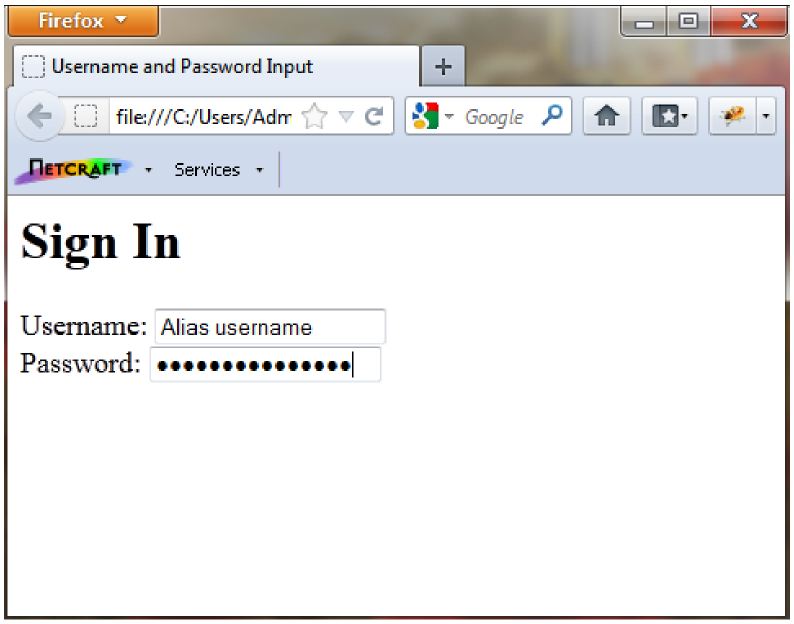


Figure 3.1: HTTP Communication through CredProxy

As can be seen in figure 3.1, CredProxy can be used with the HTTP protocol. HTTP [43]is an application-layer protocol which is used throughout the Internet extensively. This protocol is based on a request-response function; in other words, a client, which needs to access some resource in the network, sends a HTTP request message to a server and the server sends back a response to the client. Therefore, through this request-response process, resources such as HTML files, etc. are retrieved by the client. The user can use CredProxy with HTTP protocol. When HTTP is used, the process is very simple. Whenever the user wants to log into a website, he inputs his alias username and password on the login page of the website. The proxy, which is in the middle of communication, receives the alias username and password. CredProxy can parse HTML. HyperText Markup Language (HTML for short) is the main markup language which is used to display web pages and other information in a web browser environment [104], and it is maintained by W3C [105]. A web browser reads HTML files and interprets them into web pages.

CredProxy can parse HTML documents and identify HTML forms and the fields defined in them. HTML forms are used to collect information from the user and to pass it to an intended server. In fact, HTML forms can contain input elements such as text fields, password fields, submit buttons, etc. Web pages usually use text fields for username inputs. In reality, text fields resemble a box which has a single line of text characters, and they are usually preceded by a line of description. As the user types in his username, the characters show up in the box in clear text. On the other hand, password fields are used to capture the user's password, and they are usually preceded by a line of description as well. However, when the user types in his password, the typed characters are shown as asterisks or bullets for security reasons. A simple login page can be seen in figure 3.2.



Sign In

Username: Alias username

Password: •••••••••••••••

Figure 3.2: Simple Login Page

CredProxy can identify username and password fields in forms in HTML documents. When the user needs to log into a website, he types in his alias username and password into an HTML web page such as the one depicted in figure 3.2. Subsequently, CredProxy, which can hear all the communication, receives the alias username and password and verifies them. Upon successful verification, CredProxy replaces the alias username and password with real credentials for the intended website, and forwards them to the website. The website, in turn, verifies the credentials and based on the result it grants or denies access. Finally, the authentication process is complete. It should be noted that this whole process is transparent to the server, and the user is not involved more than typing in his alias username and password.

However, HTTP does not provide encryption mechanisms to protect user sensitive data, which means that the communication is not encrypted, and it is in plain-text format. As a result, an adversary can simply tap into the communication, and steal sensitive information. In other words, since no encryption is used, an adversary can simply captures the outgoing data and extract the user's real credentials, which might defy the whole idea of our proxy which is to secure credentials. Although we have embedded this capability in our model, we do not recommend using it due to the well-known security shortcomings of this protocol.

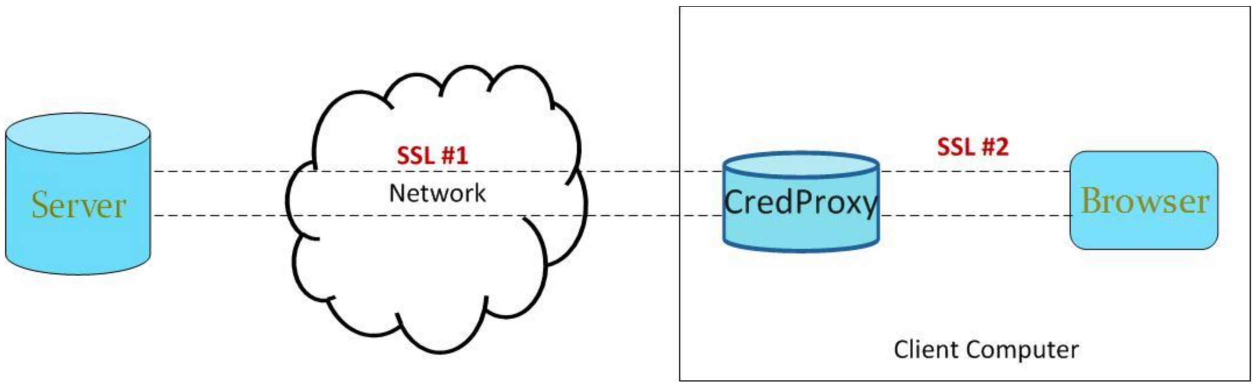


Figure 3.3: HTTPS Communication through CredProxy

As can be seen in figure 3.3, CredProxy also works with HTTPS. Hyper Text Transfer Protocol Secure (HTTPS for short) [106] has a similar syntax to the standard HTTP protocol with one main difference: whenever it is used, HTTPS signals the browser to add an extra encryption layer of SSL [26]/TLS [107] to protect the communication [108]. TLS (successor of SSL) is a cryptographic protocol which is used to provide privacy and data integrity between two communicating parties [26], and it contains a set of cryptographic functions. These cryptographic functions are utilized to provide data encryption, data integrity and the website's identity verification (this verification can be mutual; in other words, the client computer's identity as well as the website's identity can authenticated by the other party. However, authenticity verification of the client-side identity is optional). In HTTPS, the entire HTTP communication is encrypted, which brings about a private end-to-end communication channel between the client browser and the server. Thus, eavesdroppers cannot learn much from HTTPS communications, and most likely, HTTPS communication modification cannot go undetected from participating parties.

Unlike HTTP, HTTPS makes our task much more difficult. We need to tap into the communication between the client browser and the server, in order to be able to replace the alias username and the password with real credentials. However, TLS in the browser uses X509 [109]certificates to first check the identity of the website, and then to establish an encrypted private network between them. The communication is fully encrypted between the client and the server, which leaves us with little chance of decrypting the communication and replacing the alias username and password with real ones for an intended website through interception of HTTPS communication.

To resolve the aforementioned problem, we use the SSL Man-in-the-Middle technique [110] [111] for the sake of our Proxy. In our model, two different encrypted private networks are established for each communication. The first one is from the server to the CredProxy, and the second one is from CredProxy to the browser. We use different certificates and negotiations for both encrypted tunnels which are the server-CredProxy tunnel and the CredProxy-browser tunnel. For each tunnel, HTTPS negotiation is carried out separately. Thus, session keys of tunnels are separately computed, and are different.

The first SSL-encrypted connection is from CredProxy to the server. We suppose that the user has already registered for the website hosted over HTTPS, and that his real credentials are stored securely in the credential vault. Later, we fully explain about the registration process. The following are the steps that take place when such a web resource is requested by the user:

1. The user requests to visit an HTTPS website in his browser.
2. The browser sends the request to CredProxy.
3. On behalf of the user, CredProxy initiates the HTTPS connection with the server, and requests the server to provide identification.
4. The server sends back to CredProxy a copy of its certificate.
5. CredProxy verifies the certificate. The certificate verification is carried out thoroughly; in other words, the issuer, subject, validity, signature and certification path are all checked for consistency. If there is anything wrong with the certificate (such as the DNS name of the website not matching the certificate subject), the connection is denied by CredProxy, and a clear message is sent to the browser to notify the user of the possible threat.
6. If the certificate is valid, SSL negotiation will be carried out, and a symmetric session key will be generated. The symmetric session key is the key with which the tunnel between

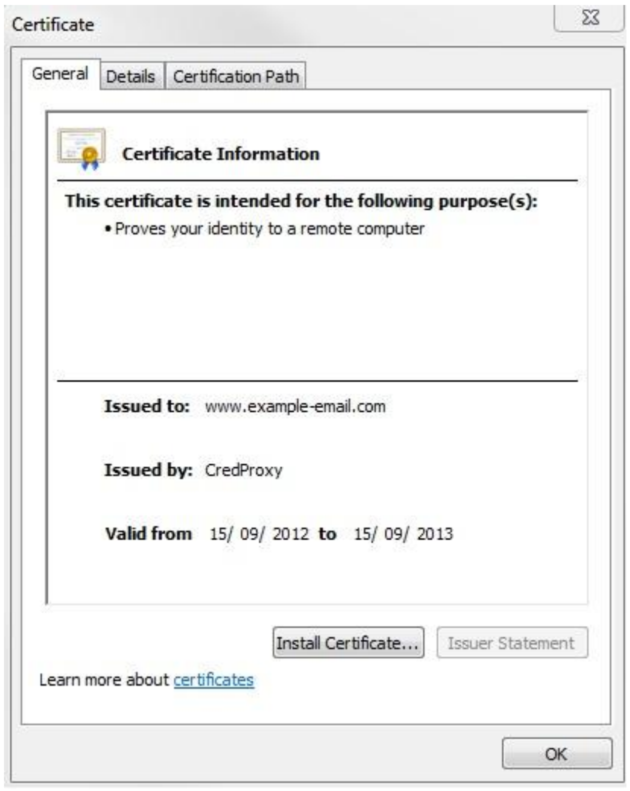
the server and CredProxy is encrypted and decrypted using symmetric encryption algorithms. Now, the encrypted communication is established between the server and CredProxy.

The second SSL-encrypted tunnel is between CredProxy and the browser. This part is only performed when the certificate verification results in no error. As mentioned earlier, if there is any error, CredProxy will deny the connection, the user will be notified of it. The steps to establish the second SSL tunnel are as follows:

1. The user requests to visit a website over HTTPS in his browser.
2. The first SSL tunnel is established between the server and CredProxy.
3. CredProxy sends the appropriate certificate (it will be explained later) to the browser.
4. Just like CredProxy, the browser performs the certificate verification process. Since this certificate is issued by CredProxy, the browser will encounter no error in the process of verification.
5. The browser realizes that the certificate is valid, and proceeds with the rest of the SSL negotiation. A symmetric session key will be generated by the browser and will be acknowledged by CredProxy. From then on, all the communication between the browser and CredProxy is symmetrically encrypted with the agreed-upon symmetric session key.

CredProxy can generate X509 certificates based on the websites' DNS names. It utilizes public key cryptography to generate and sign certificates for use with intended websites. Every certificate has different fields such as subject, issuer, validity period, types of cryptographic algorithms used, public key, certificate path, etc. For each HTTPS website the user would like to receive service from, CredProxy generates an appropriate certificate. By appropriate we mean that all the fields in an X509 certificate are populated according to the information of the website which the user intends to use.

Suppose that the user intends to access the "www.example-email.com" website to check his emails. CredProxy proceeds with creating a certificate such as the one depicted in figure 3.4.


Certificate

Certificate Information

This certificate is intended for the following purpose(s):

- Proves your identity to a remote computer

Issued to: www.example-email.com

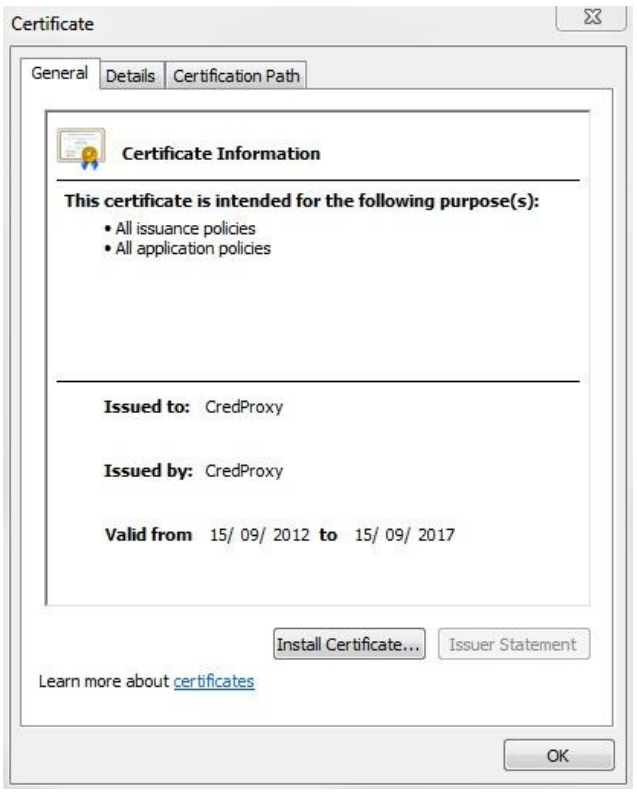
Issued by: CredProxy

Valid from 15/ 09/ 2012 to 15/ 09/ 2013

Figure 3.4: Certificate Generated by CredProxy (Mock-up)

As can be seen in figure 3.4, the issued-to field contains the DNS name of "www.example-email.com", and it is issued by CredProxy.

When a typical browser receives a certificate from a server, it performs several steps to verify the authenticity of the certificate. Errors resulting from some of the verification steps such as CA's digital signature cause the certificate to be automatically rejected. However, some errors such as the subject field not matching the website's DNS name or the issuer not being trusted by the client computer just causes a warning in the browser. In this case, the user will be made aware of the suspicious (and possibly malicious) situation through a message in the browser, and the decision to proceed will be left to the user. We would like to avoid any kind of certificate error in the browser in order to leave the user with no confusion and doubt. Therefore, CredProxy also generates a self-signed certificate such as the one illustrated in figure 3.5.


Certificate

General | Details | Certification Path

Certificate Information

This certificate is intended for the following purpose(s):
- All issuance policies
- All application policies

Issued to: CredProxy

Issued by: CredProxy

Valid from 15/ 09/ 2012 to 15/ 09/ 2017

Install Certificate... | Issuer Statement

Learn more about certificates

OK

Figure 3.5: Self-signed CredProxy Certificate (Mock-up)

As can be seen in figure 3.5, both the issued-by and the issued-to fields contain the name of CredProxy. This is a self-signed certificate where a certificate is signed by the same entity whose identity it certifies [112]. This is usually the characteristic of root certificates which cannot be attested to by some higher certification authority. The above certificate acts as the root certificate for CredProxy, and CredProxy installs this certificate in the "Trusted Root Certification Authorities" Store. By default, operating systems come with a set of trusted certification authorities' certificates, which are used for verification of websites and other purposes. When a certificate is installed in the store, it means that the computer trusts the issuer, and the certificates signed by that issuer are considered valid. CredProxy adds its root certificate to the store in order for the browser to be able to verify CredProxy-generated website certificates, and in order not to generate any warnings during verification of those certificates. As a result, the tunnel between CredProxy and the browser is established, and the user has not experienced any inconsistency so far.

The aforementioned technique, which we have used in our model, is the same as the SSL Mani-in-the –Middle technique used in the literature [111] [110] but with one difference. The Man-in-the-middle techniques (which we are aware of) all use fake certificates (certificates which are not created and used by intended websites but only by adversaries). In these techniques, attackers obtain a website’s certificate and make an exact copy of it with almost all the fields containing roughly same values as the original one to make it look convincing. However, the issuer field contains a different certification authority and is not usually trusted by the client computer, because it is not a valid certification authority- it is an ad-hoc certification authority created by attackers. Of course, adversaries have to rely on techniques such as social engineering, exploiting browser bugs and defects or abusing naivety of the user in order to carry out this type of attack [111]. However, we do not need to rely on such ill-meant techniques. We have access to the user’s machine as the guardian of the user’s credentials. As a result, we can install the self-signed certificate in the trusted root certification authorities store in order to avoid any sort of warning generation in the browser due to CredProxy-signed certificates. We first saw this technique being used in a commercial product with a proxy-based credential storage scheme, and this idea has been borrowed from that product.

3.5 Shortcoming

While proxy-based password managers can offer outstanding protection against Mani-in-the-Browser attacks, they heavily suffer from extendibility. In the current world of the Internet, almost all login pages grab the values of username and password fields on the login page, and encrypt/hash them before sending them out to their websites. However, the proxy situated in the middle of the communication should have the exact knowledge of this process in order to be able perform the same operation on the real username and password. In other words, without this knowledge, the proxy could not send the real username and password in the format which the website would expect to get. Thus, the knowledge of the technique used on any login page is necessary to be incorporated into the proxy to make it fully operable with an intended website. This can, however, impose a significant drawback on the adaptation of proxy-based password managers, since the user would not be able to use the proxy-based password manager on an ad-hoc basis.

3.6 Chapter Summary

This chapter has introduced CredProxy, and how user credentials could be protected in the browser through it. It has also provided the architecture of CredProxy and discussed how it interacts with the browser and a website through HTTP and HTTPS. Having a proxy as intermediate in a communication can be advantageous, since the user does not have to put his trust in the browser anymore in terms of providing the authentication information to the browser. When CredProxy is used, the user can just input the alias username and password on the browser, and CredProxy will replace it with real credentials if the following conditions are met:

- The user must be logged into the CredProxy via the virtual keypad (discussed in Chapter 6).
- The user should provide the correct alias username and password.
- There must be an entry for that website in the credential vault (discussed in Chapter 5).
- Finally, the certificate of the website should pass all the checking.

If the above conditions are met, the replacement will be carried out. Otherwise, the CredProxy will not allow access to the credential vault. Thus, even if some malicious code were injected in the browser, it would not be able to steal user authentication information. Moreover, phishing is alleviated through the comprehensive checking of website SSL certificates. The next chapter discusses the Function CPGen of CredProxy.

Chapter 4

Function CPGen

In this chapter, we present Function CPGen of CredProxy. This function's task is to securely and automatically generate usernames and passwords for the user to be used for online services. We use only one algorithm for the username generation; however, two different methods will be used for the generation of passwords. In addition, the credential vault is introduced later in this chapter.

4.1 Introduction

In this section, we present Function CPGen of CredProxy. CredProxy is located on the user's computer, and it realizes Function CPGen along with its functionalities. The user's web browser is configured by CredProxy (when CredProxy is launched on the client computer) to automatically forward HTTP and HTTPS traffic to CredProxy, which makes CredProxy the gateway through which all communication is passed. This, in fact, includes all the messages that are exchanged between the user's browser and the server. CredProxy can understand both sides of communication (the server to CredProxy and CredProxy to the browser), since it has participated in both sides' HTTPS negotiation, which leaves it in possession of both communications' symmetric session keys.

Throughout the Internet, text usernames and passwords still maintain their unquestionable dominance as the most popular form of authentication in spite of all the security problems intrinsic to them [7]. Much has been said about user-chosen weak passwords which can make techniques such as brute force attacks fruitful for adversaries. Moreover, forcing the user to choose strong passwords by password policy enforcement techniques may still give rise to more issues. For instance, even in the presence of a good password strength policy, it is not yet very clear how the applied policy might affect the strength of the resulting passwords [113]. To make matters worse, having a strong password policy could make passwords less memorable;

as a direct consequence, this may lead to workarounds by users such as writing passwords on stickers or storing passwords in clear-text format in files, which could result in even more damage.

We have realized that the best way to help the average user is to unburden him from the Herculean task of creating passwords. We believe that the user should be involved as little as possible in the process of password creation. Thus, we put the burden on CredProxy.

4.2 Username Generation

Random usernames which are automatically generated by CredProxy can bring about more anonymity and security for the user. Of course, by anonymity we simply mean that the username should not reveal much about the user's identity. Users usually tend to user their personal information such as first name, last name, etc. as their usernames or at least part of their usernames. Thus, using random strings as username could improve the anonymity of the user. Second, the first step to attacks such as brute-force and dictionary attacks is to find the username of the victim. Florencio et al. [15] showed that a direct brute-force attack on the password of a given online account is difficult, for these attacks are easily detected and blocked. However, instead of searching all possible passwords for one username, an adversary could simply search all possible usernames for a given password, which makes the detection of this attack by far harder than brute-force attacks [15]. Therefore, increasing the strength of the username can also increase the security.

Function CPGen is the component of CredProxy which automates the generation of both usernames and passwords. Of course, CredProxy is flexible with regard to credentials the user might already have; that is, the user always has the choice of either using CredProxy to automatically create usernames and passwords or using his already-created credentials with CredProxy. In either case, credentials will be securely stored in the credential vault, which can be accessed through the alias username and password in the browser when CredProxy is up and running on the client computer. Furthermore, the user can feed his old credentials into the credential vault of CredProxy through the virtual keyboard discussed in chapter 6.

Function CPGen creates usernames as follows:

Min_4	the minimum length of usernames
Max_4	the maximum length of usernames
R	random number generator (between 0 and 1)
σ	Random seed of R
C	set of small alphabetical characters
N	set of digits
S	set of allowed symbols
L	a limited character
L_Max	the maximum number of repetitions of L allowed in an automatically-created username

Table 4.1: Notation Used in Function CPGen (Username Generation)

```
1. Length =$\lfloor Min_4 + (Max_4 - Min_4) * f(\sigma) \rfloor$
2. For (i=1; i< Length; i++):
   2.1. Pick a set from available sets (S, N, C, and L) uniformly at random
   2.2. If (the set! = a limited set) {pick an element from the set uniformly at random and
        attach it to the username} else {if (the set's counter == L_Max) {go to 2.1} else {pick
        the letter from the set and increment the set's counter}
```

Suppose that the user would like to delegate the task of username creation on a website to CredProxy. The username policy on the website allows users choose from alphabetic characters, digits, set of symbols S= {$, #} and also a limited character { _ } with L_Max = 2 in a single username. Thus, through our algorithm, a username will be chosen at random for the user. As a matter of fact, in real life situations, there are usually cases where a character or two are not allowed to be used more than a certain number of times in usernames. For instance, Yahoo! allows letters, numbers and underscores but only one dot in usernames. Therefore, dot will be defined as a limited character and L_Max will be set to 1. We can also accommodate more than one limited character with its maximum counter in our algorithm.

4.3 Password Generation

In this section we describe how we generate random passwords. Our scheme functions in two modes:

- Random password generation mode
- PKI key pair mode

Random password generation mode is used when the user would like to create a password similar in length to everyday passwords but stronger and more secure. We define the guidelines for strong password creation as follows. A strong password

- Should be at least 16 characters
- Should contain uppercase letters
- Should contain lower case letters
- Should contain numbers
- Should contain symbols
- Should not be derived from the user's personal information such as: name, birth date, pet's name, etc.
- Should not be a common word such as: "strong"
- Should not be a pattern of keys on keyboard such as: "edcrfv" or "753951"

However, the most important factor in generating a strong password is that every letter in a password should be chosen uniformly at random. In fact, maximum password strength is achieved when each letter in a password is chosen independently and a uniform distribution is used [114]. That is to say that following the above guidelines in **automatic password generation** could bring about the generation of passwords of lower entropy. Thus, we do not enforce the above guidelines, and base our algorithm only on true randomness. Our proposed algorithm is as follows:

Max_5	maximum length of passwords
R	random number generator (between 0 and 1)
σ	Random seed of R
C_1	set of small alphabetical characters
C_2	set of capital alphabetical characters
N	set of digits
S	set of allowed symbols
MS	Master set containing all sets C_1, C_2, N and S

Table 4.2: Notation Used in Function CPGen (Password Generation)

```
1. Length = $Max_5$
2. For (i=1; i< Length; i++):
    2.1. Choose an element from MS uniformly at random and attach it to the password
```

The second mode of password generation is called Public key pair mode. In this mode, instead of using text passwords, we use PKI key pairs {K_{Public}, $K_{Private}$} for authentication. Function CPGen of CredProxy generates a public key which is stored on the server and a private key which is stored in the credential vault encrypted. Then, this key pair is associated with the website's name and identity before being stored in the credential vault. Of course when PKI key pairs are utilized, the authentication process is not similar to the traditional authentication process; private keys never leave the client computer in any form. Instead, a challenge-response mechanism is usually used. For instance, the server might ask the CredProxy on the client computer to sign a message using the private key, the public key of which is stored on the server. Then, the CredProxy signs the message, and sends it back to the server. Upon successful verification of the signature, the access will be granted to the client computer. The best advantage of this method is that even if the server were hacked and public keys of users were exposed, adversaries would not be able to extract much information out of public keys since public keys could not be used for authentication. For instance, recently Russian hackers claimed that they had hacked into the social networking website LinkedIn [115], and they released a gigantic list containing 6.5 million hashed passwords online [116]. Even some Twitter [117] users have reported that they found their hashed LinkedIn passwords on the list [116].

We believe that the main deterrent to using PKI key pairs has always been their length and their lack of memorability. In our model, the user does not have to memorize passwords; thus, password length does not really distress the user. Through our scheme, we can alleviate the problem of password leakage due to attacks on servers.

In addition, our model is also flexible where the user's already-created passwords are concerned. We want our model to be backward compatible; that is, if a user has signed up for an online service and has credentials for that server, he should still be able to use CredProxy. In order to input the credentials securely into the CredProxy installed on the client computer, the user can use the virtual keyboard discussed in Chapter 6. Our virtual keyboard is an on-screen keyboard that thwarts several known keylogging and screen-grabbing attacks.

4.4 Credential Vault

The credential vault is a file on the client computer system that stores the user credentials in encrypted form. These credentials can be generated using Function CPGen, or entered through the virtual on-screen keyboard when the user signs up on a new website. Already-created user credentials could also be submitted to this repository thorough the virtual on-screen keyboard discussed in Chapter 6. Each entry in the credential vault consists of a username, password and the URL(s) for which this username and password is used. Having the URL, CredProxy can determine which websites it can use the entry for. For instance, if we had an entry for the URL www.example-mail.com, the alias username and password would only be replaced with the username and password in the entry when this URL is visited. Moreover, for each entry more than one URL can be accommodated. This is the case where the user would like to perform authentication in a decentralized manner. In other words, the user might want to perform authentication on a website using an account from another website. Thus, in our scheme this capability has been accounted for to provide more convenience for the user.

4.5 Chapter Summary

This chapter has introduced Function CPGen which automates the generation of usernames and passwords. We have pointed out the significance of strong usernames and passwords, and how they could thwart brute-force guessing attacks. This chapter has also presented the description of the credential vault, and how it can be populated with user credentials. The next chapter focuses on Function CPGuard.

Chapter 5

Function CPGuard

Function CPGuard is the component of CredProxy which is responsible for encrypting and decrypting the credential vault on the client computer. We propose three different protocols for Function CPGuard. The assumptions and threat models for the first two protocols are roughly the same; however, they are different for the third one. Moreover, in the first two protocols a mobile device is used as a token, unlike the third one in which the mobile device is only used to receive one-time codes.

5.1 Introduction

The credential vault is the repository where all the user credentials are stored. After logging into the CredProxy, the user can access the websites for which he has an account in the credential vault through the proxy. However, prior to access being granted, the credential vault must be decrypted. We create a key of adequate entropy to lock the credential vault with. This key is not stored on the client machine, and is stored on the CredServer. CredServer is an additional server, the role of which is to assist us in protecting the credential vault. In our protocols, we create and store the encryption/decryption key to the credential vault on the CredServer. This feature allows two important capabilities which will be discussed in Section 5.5. When the proxy is done with the credential vault (e.g., when the user wants to create, modify or remove entries from the credential vault), the credential vault will be encrypted again.

In the first protocol, the mobile device is used as an out-of band device to perform the two-factor authentication. This protocol best suits situations in which the Internet is available on the cell phone, and connecting the phone to the client computer (in order to use the Internet connection on the client computer), might be cumbersome for the user. On the other hand, the second protocol makes use of the Internet connection on the client computer through which the mobile device can communicate with the CredServer directly. The assumption in both aforementioned protocols is that the mobile device is malware-free. However, we also have the third protocol in which this assumption is not made. Consequently, an additional piece of hardware is used to generate one-time pass response codes which are received from the CredServer. This mechanism is used as the second-factor authentication in the third protocol.

Each protocol consists of two phases: the registration phase and the authentication phase. In the registration phase, the user installs the PC component of CredProxy on the client computer and the mobile component of CredProxy on the mobile device (the mobile component is used only for the first two protocols). Next, the user registers his computer and mobile device on the CredServer. Then, the credential vault is built and encrypted. In the authentication phase, the user uses both devices to get access to the CredServer to decrypt the credential vault, since the credential vault can only be decrypted and used when both devices are present. Table 5.1 describes the notation used in our protocols.

S, P, M, U	CredServer, CredProxy installed on the client computer, a smart phone, the user respectively
Reg_Key	Registration key obtained via an out-of-band channel
H (.)	A cryptographically secure hash function
T (.)	a hardware challenge-response token
$\oplus$	XOR
PIN	Pin selected by the user
S_1	Combination of hardware serial numbers of the mobile device
S_2	Combination of hardware serial numbers of the client computer
ID_1	Identification of the mobile device
ID_2	Identification of the client computer

C	One-time challenge sent by the CredServer
Key	The key used to encrypt/decrypt the credential vault
K_File	The file in which *Key* is stored
Y	A value of arbitrary length
X	A value of arbitrary length
Reg_Req	Request for registration
Auth_Req	Authentication request
Key_Mem	Key used to encrypt/decrypt the credential vault in memory on the client computer
$\{Data\}_{KMS}$	Asymmetric encryption of *Data* using S's public key between M and S
$\{Data\}_{KPS}$	Asymmetric encryption of *Data* using S's public key between P and S
$\{Data\}_{E\ (K)}$	symmetric encryption of *Data* using key *K*
$\{Data\}_{D\ (K)}$	Symmetric decryption of *Data* using key *K*

Table 5.1: Notation Used in Function CPGuard

5.2 Protocol 1 of Function CPGuard

5.2.1 Threat model, Assumptions of Protocol 1

This protocol makes use of the user's mobile device as a token. The mobile device uses an out-of-band channel (e.g. the Internet connection on the mobile device rather than through the user's client computer).

We assume that U has received Reg_Key via an out-of-band channel. The user's computer may have keyloggers and screen grabbers. The browser may also be compromised. DNS cache poisoning may have occurred. Pharming attacks on the client computer may be used. Information shown to the user on the display is not authenticated. We assume that S has a valid certificate, and only accepts SSL connections. On the other hand, P checks S's SSL certificate, and only establishes an SSL session if all the checking passes, and notifies the user of potential threat otherwise. P does not accept the certificates which are signed by CredProxy either (more detail in Section 5.5). The mobile device is malware free. If a user loses his mobile device, he can notify S to disable it and register another one. Denial-of-Service attacks are not discussed.

5.2.2 Registration Process 1

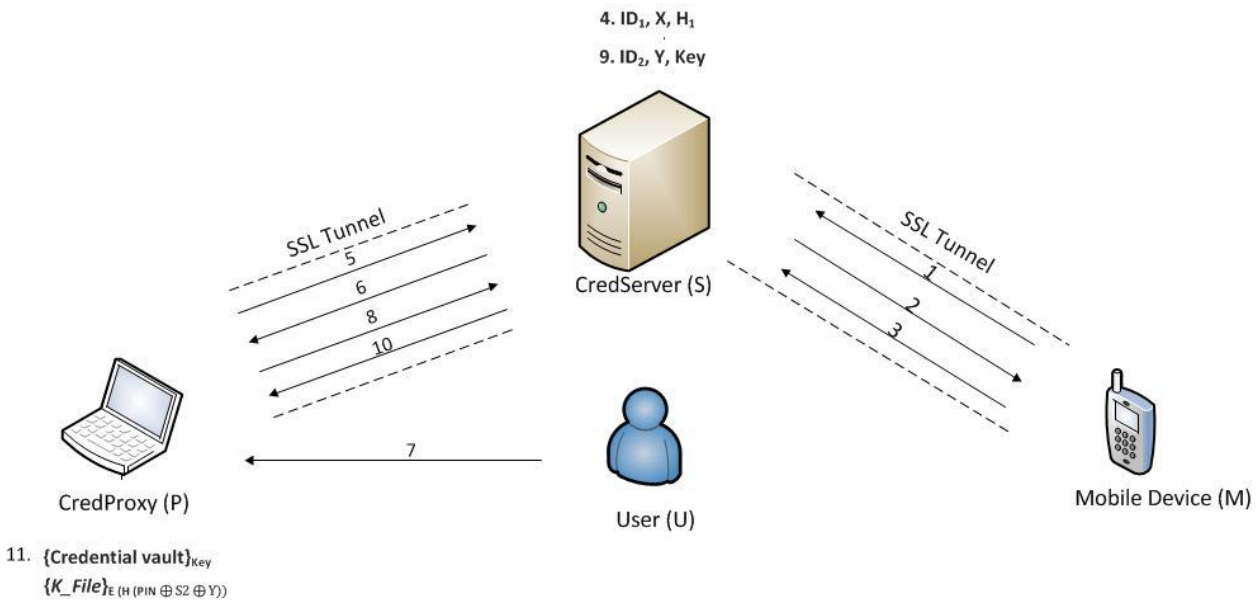


Figure 5.1: Registration Process 1

1. U launches the mobile component M of the CredProxy on the cell phone to initiate the registration process. M establishes an SSL session with the CredServer and asks U to input the registration key which has already been received via an out-of-band channel. U inputs the registration key on M and requests registration. The following message is sent from M to S:

$$S \leftarrow M :\{ Reg\_Req, Reg\_Key\}_{\mathrm{KMS}}$$

2. S sends ID_1 and value X to M in the following message:

$$M \leftarrow S :\{ ID_1, X\}_{\mathrm{KMS}}$$

3. The M stores ID_1, and sends back the following massage to S:

$$S \leftarrow M :\{ H_1 = H (S_1 \oplus X)\}_{\mathrm{KMS}}$$

4. S stores values: ID_1, X, H_1 for M.
5. The user runs the CredProxy which establishes an SSL session with S. P asks the user to input the registration key. Then, P requests registration on S, and sends the registration key to S.
6. S checks to see whether the registration key is valid. Upon verification, it asks P to provide the user's PIN and S_2.
7. U types in the PIN using the secure virtual keypad.
8. P sends back the following message to S:

$$S \leftarrow P :\{ H_2 = H (S_2 \oplus PIN)\}_{\mathrm{KPS}}$$

9. S creates ID_2, Y and Key and associates them with M which has already been registered.
10. Then, it sends back to P the following message:

$$P \leftarrow S: \{ID_2, Y, Key\}_{\mathrm{KPS}}$$

11. P computes the following:

$$\{Credential\ Vault\}_{\mathrm{E\ (Key)}}$$

$$\{K\_File\}_{\mathrm{E\ (H\ (PIN} \oplus \mathrm{S2} \oplus \mathrm{Y))}}$$

5.2.3 Authentication Process 1

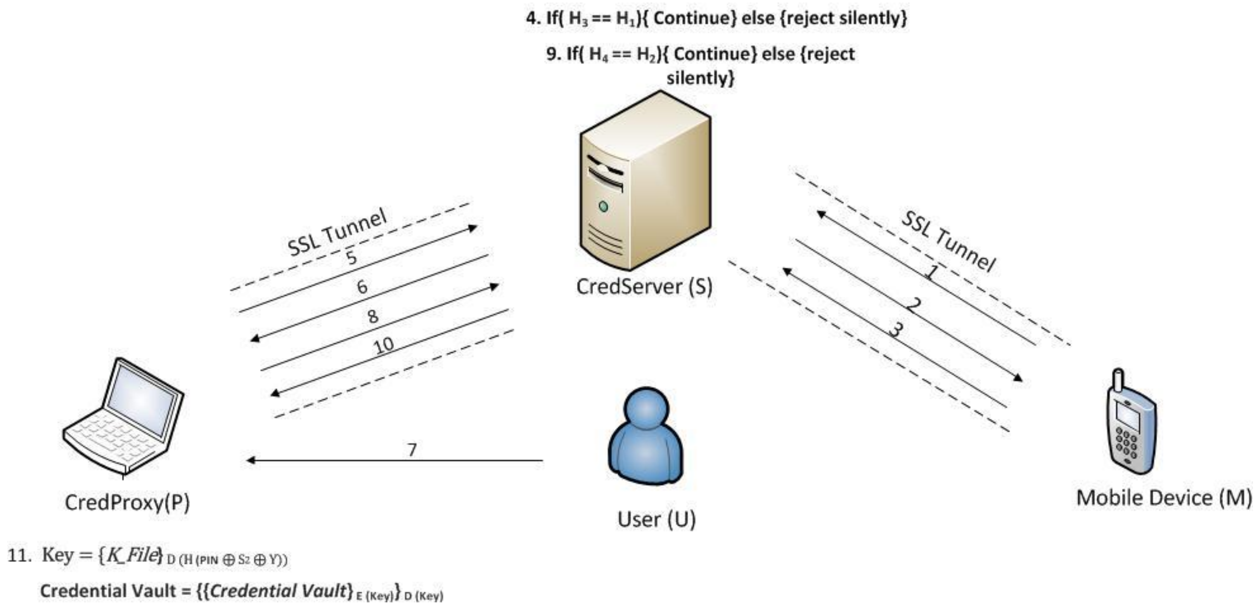

4. If(H3 == H1){ Continue} else {reject silently}
9. If(H4 == H2){ Continue} else {reject silently}
CredServer (S)
SSL Tunnel
5
6
8
10
SSL Tunnel
1
2
3
7
CredProxy(P)
User (U)
Mobile Device (M)
11. Key = {K_File} D (H (PIN ⊕ S2 ⊕ Y))
Credential Vault = {{Credential Vault} E (Key)} D (Key)

Figure 5.2: Authentication Process 1

1. U launches M, and M establishes an SSL session with the server and requests authentication by sending an authentication request and its ID_1 to S.
2. S receives the authentication request and checks ID_1 to see if M has already registered. If M is registered, S sends back the corresponding X to M.
3. M sends the following message to S:

$$S \leftarrow M :\{ H_3 = H (S_1 \oplus X)\}_{\mathrm{KMS}}$$

4. S checks whether H_3 is equal to H_1. If those two hashes match, P can continue with the authentication process; otherwise, the authentication request is rejected silently by S.
5. U runs its CredProxy on the computer, and P establishes an SSL session with S. It then requests authentication by sending the following message:

$$S \leftarrow P: \{Auth\_Req, ID_2\}_{\mathrm{KPS}}$$

6. S checks to see whether the corresponding M has checked in. S also takes into account a time-out delay between the time the phone checks in and the time the CredProxy starts

the authentication process. If the result is positive and the time-out delay has not expired, S will ask P to Proceed with sending the authentication information, and P will be denied access otherwise.

7. U enters the PIN.
8. P sends the following message to S:

$$S \leftarrow P\colon \{H_4 = H\,(S_1 \oplus PIN)\}_{KPS}$$

9. S checks whether H_4 is equal to H_2.
10. If the result is positive, the following will be sent to the CredProxy:

$$P \leftarrow S\colon \{Y,\ Key\_Mem\}_{KPS}$$

11. P uses Y to compute the following:

$$\text{Key} = \{K\_File\}_{D\,(H\,(PIN \oplus S2 \oplus Y))}$$

$$\text{Credential Vault} = \{\{Credential\ Vault\}_{E\,(Key)}\}_{D\,(Key)}$$

12. (Optional) the CredProxy throws away Key and encrypts the credential vault with *key_Mem* and stores it in the memory. Then, it also throws away *key_Mem* as well. From now on, whenever the CredProxy needs to use the credential vault, it needs to contact S through previously established SSL connection, and ask for the *key_Mem*.

At this point, the credential vault has been decrypted and stored encrypted in the memory. The user could access his credentials through CredProxy. Step 12, indeed, is optional, since requesting the *Key_Mem* every time that the CredProxy needs to use the credential vault could have a noticeable impact on the performance of the system. Moreover, *Key_Mem* should be omitted from the message sent in Step 10, if this feature is not used.

In addition, the Value *X* is used for two reasons. First, if a rogue sever impersonated the CredServer, it would not be able to get the S_1, since the mobile device only sends the hashed value of *X* xored with S_1 to the CredServer. Moreover, if the CredServer were compromised, information about the registered mobile devices' hardware properties would not leak out, as they are all xored with *X* and then passed through a hash function. After recovery form the compromise, the CredServer would just have to choose another value *X* for a user, and the user would be able to continue using his smartphone with his account.

5.3 Protocol 2 of Function CPGuard

5.3.1 Threat model and Assumptions of Protocol 2

This protocol also utilizes the use's mobile device as a token. However, the mobile device does not use an out-of-band channel to communicate with S; it uses the client computer.

We assume that U has received Reg_Key via an out-of-band channel. The user's computer may have keyloggers and screen grabbers. The browser may also be compromised. DNS cache poisoning may have occurred. Pharming attacks on the client computer may be used. Information shown to the user on the display is not authenticated. We assume that S has a valid certificate, and only accepts SSL connections. On the other hand, P checks S's SSL certificate, and only establishes an SSL session if all the checking passes, and notifies the user of potential threat otherwise. P does not accept the certificates which are signed by CredProxy either (more detail in Section 5.5). The mobile device is malware free. The communication between the mobile device and the client computer is established through any reasonable means such as Bluetooth and wire. The mobile device cannot be infected by the malware on the client computer (e.g. crossover viruses [118]). If a user loses his mobile device, he can notify S to disable it and register another one. Denial-of-Service attacks are not discussed.

5.3.2 Registration Process 2

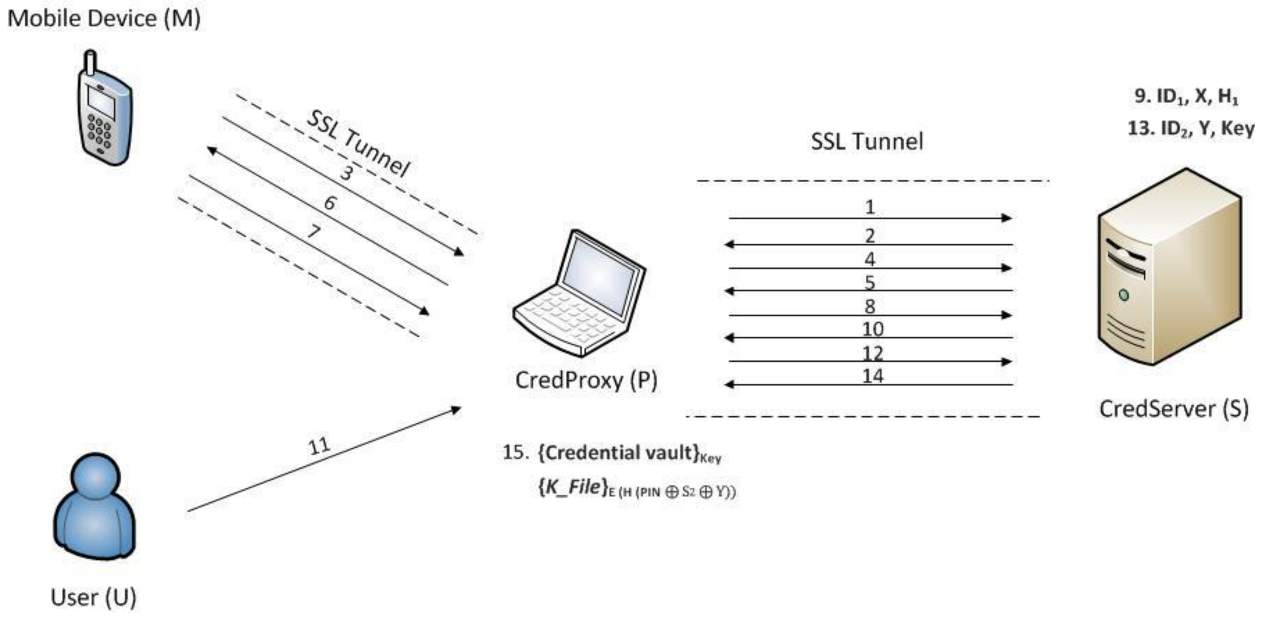


Figure 5.3: Registration Process 2

1. The user runs P which establishes an SSL session with S. P asks the user to input the registration key. The following message is sent to S:

$$S \leftarrow P\text{: } \{ \textit{Reg\_Req}, \textit{Reg\_Key} \}_{KPS}$$

2. S asks U to connect M and initiate the registration process on M.
3. M connects to S and establishes a separate SSL session with S. U also inputs the registration key on M. Then, M sends the following message to P for S:

$$P \leftarrow M\text{: } \{ \textit{Reg\_Req}, \textit{Reg\_Key} \}_{KMS}$$

4. P forwards the message to S:

$$S \leftarrow P\text{: } \{\{ \textit{Reg\_Req}, \textit{Reg\_Key} \}_{KMS}\}_{KPS}$$

5. S decrypts the message, and takes out the data. Then, S asks M to provide S_1 through the following message:

$$P \leftarrow S: \{\{ID_1, X\}_{KMS}\}_{KPS}$$

6. P forwards the message to M.
7. M decrypts the message, stores ID_1 and computes H_1 as follows:

$$P \leftarrow M: \{H1 = H\ (S_1 \oplus X)\}_{KMS}$$

8. P forwards the message to S as follows:

$$S \leftarrow P: \{\{H1 = H\ (S_1 \oplus X)\}_{KMS}\}_{KPS}$$

9. S decrypts the message, and takes out the data. S, then, stores ID_1, X, H_1 for M.
10. S asks P to Proceed.
11. P asks U to enter PIN through the virtual keypad.
12. P sends the following message to S:

$$S \leftarrow P :\{\ H_2 = H\ (S_2 \oplus PIN)\}_{KPS}$$

13. S creates ID_2, Y and Key and associates them with M which has already been registered.
14. Then, it sends back to P the following message:

$$P \leftarrow S: \{ID_2, Y, Key\}_{KPS}$$

15. P computes the following:

$$\{Credential\ Vault\}_{E\ (Key)}$$

$$\{K\_File\}_{E\ (H\ (PIN \oplus S2 \oplus Y))}$$

5.3.3 Authentication Process 2

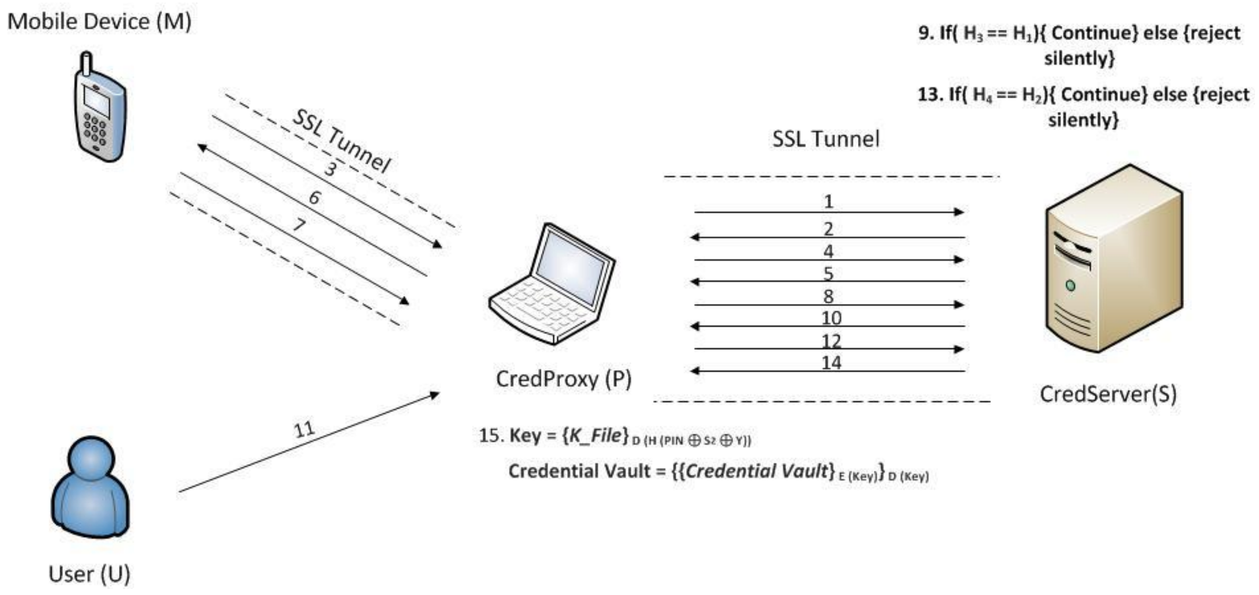

Mobile Device (M)
9. If(H_3 == H_1){ Continue} else {reject silently}
13. If(H_4 == H_2){ Continue} else {reject silently}
SSL Tunnel
3
6
7
SSL Tunnel
1
2
4
5
8
10
12
14
CredProxy (P)
CredServer(S)
11
15. Key = $\{K\_File\}_{D\ (H\ (PIN \oplus S2 \oplus Y))}$
Credential Vault = $\{\{Credential\ Vault\}_{E\ (Key)}\}_{D\ (Key)}$
User (U)

Figure 5.4: Authentication Process 2

1. U runs P on the client computer. P establishes an SSL session with the server and requests authentication by sending an authentication request and its ID_2 to S.
2. S receives the authentication request and checks whether P is registered. Then it asks P to connect the mobile device to the client computer and initiate M.
3. M establishes a separate SSL session with S and sends the following message to P for S:

$$P \leftarrow M :\{ \text{Auth\_Req}, ID_1\}_{\text{KMS}}$$

4. P forwards the message to S:

$$P \leftarrow M :\{\{ \text{Auth\_Req}, ID_1\}_{\text{KMS}}\}_{\text{KPS}}$$

5. S receives and verifies the message. S sends the following message in response:

$$P \leftarrow S :\{\{ X\}_{\text{KMS}}\}_{\text{KPS}}$$

6. P decrypts the message and forwards it to M:

$$M \leftarrow P :\{ X\}_{\text{KMS}}$$

7. M computes the following message for S and sends it to P:

$$P \leftarrow M :\{ H_3 = H\ (S_1 \oplus X)\}_{\text{KMS}}$$

8. P forwards the message to S:

$$P \leftarrow S :\{\{ H_3 = H\ (S_1 \oplus X)\}_{\text{KMS}}\}_{\text{KPS}}$$

9. S verifies the authentication data from M. S also takes into account a time-out delay between the time that the phone checks in and the time the CredProxy starts the authentication process. If the time-out delay has not expired, and If H_3 is equal to H_1, the authentication continues; otherwise, the authentication is rejected.
10. S asks P to continue.
11. U enters the PIN.
12. P sends the following message to S:

$$S \leftarrow P: \{H_4 = H\ (S_1 \oplus PIN)\}_{\text{KPS}}$$

13. S verifies whether H_4 is equal to H_2.
14. If the result is positive, the following will be sent to the CredProxy:

$$P \leftarrow S: \{Y, Key\_Mem\}_{\text{KPS}}$$

15. P uses Y to compute the following:

$$\text{Key} = \{K\_File\}_{\text{D (H (PIN} \oplus \text{S2} \oplus \text{Y))}}$$

$$\text{Credential Vault} = \{\{\textit{Credential Vault}\}_{\text{E (Key)}}\}_{\text{D (Key)}}$$

16. (Optional)The CredProxy throws away Key and encrypts the credential vault with *key_Mem* and stores it in the memory. Then, it also throws away *key_Mem*. From now on, whenever the CredProxy needs to use the credential vault, it needs to contact S through previously established SSL connection, and ask for the *key_Mem*.

Step 16, indeed, is optional, since requesting the *Key_Mem* every time the CredProxy needs to use the credential vault could have a noticeable impact on the performance of the system. Moreover, *Key_Mem* should be omitted from the message sent in step 14, if this feature is not used.

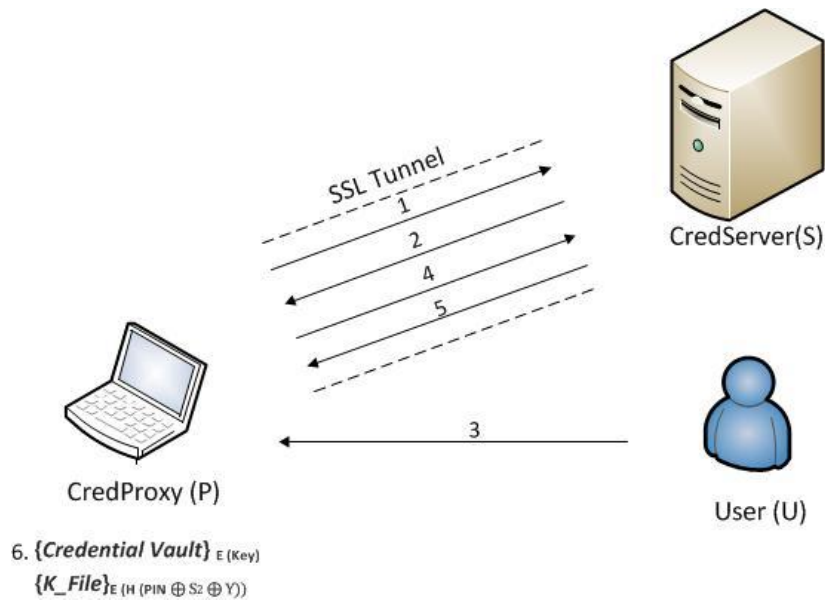
5.4 Protocol 3 of Function CPGuard

5.4.1 Threat model and Assumptions of Protocol 3

In this protocol, we put aside the assumption of malware-free hand held devices. Thus, we do not use the CredProxy mobile component on the mobile device any longer.

The user's computer may be infected with keyloggers and screen grabbers. The browser may also be compromised. DNS cache poisoning may have occurred. Pharming attacks on the client computer may be used. Information shown to the user on the display is not authenticated. We assume that S has a valid certificate, and only accepts SSL connections. On the other hand P checks S's SSL certificate, and only establishes an SSL session if all the checking passes, and notifies the user of potential threat otherwise. P does not accept the certificates which are signed by CredProxy either (more detail in Section 5.5). The user has a cell phone with a registered SIM card. We assume that the user U has received Reg_Key via an out-of-band channel. He also has registered his cell phone number on the CredServer, and the Reg_Key is associated with his phone number. We assume that the only way the user can change his phone number on the CredServer is via an out-of-band channel such as voice phone call. The user can receive SMS messages on his cell phone. SMS messages can be intercepted and changed by adversaries. The user is explicitly advised that only through an out-of band channel (phone call, mail, etc.) and not on the client computer or the cell phone, he should change the cell phone number. Denial-of-Service attacks are not discussed.

5.4.2 Registration Process 3



SSL Tunnel
1
2
4
5
3
CredServer(S)
CredProxy (P)
User (U)
6. {*Credential Vault*} E (Key)
{*K_File*}E (H (PIN ⊕ S2 ⊕ Y))

Figure 5.5: Registration Process 3

1. U launches P on his computer. P contacts S, and establishes an SSL session with S. The user enters the registration key on the virtual keypad and then P sends an authentication initiation message as follows:

   $$S \leftarrow P :\{ \mathit{Reg\_Req}, \mathit{Reg\_Key}\}_{KPS}$$

2. S receives the message, and asks P for identification. The following message is sent to P:

   $$P \leftarrow S :\{ \text{Proceed}\}_{KPS}$$

3. P asks the user to select the PIN and enter it through the virtual keypad.
4. Then, P sends the following message to S:

   $$S \leftarrow P :\{ H_1 = H(S_2 \oplus \text{PIN})\}_{KPS}$$

5. S creates ID_2, Y and Key, and sends to P:

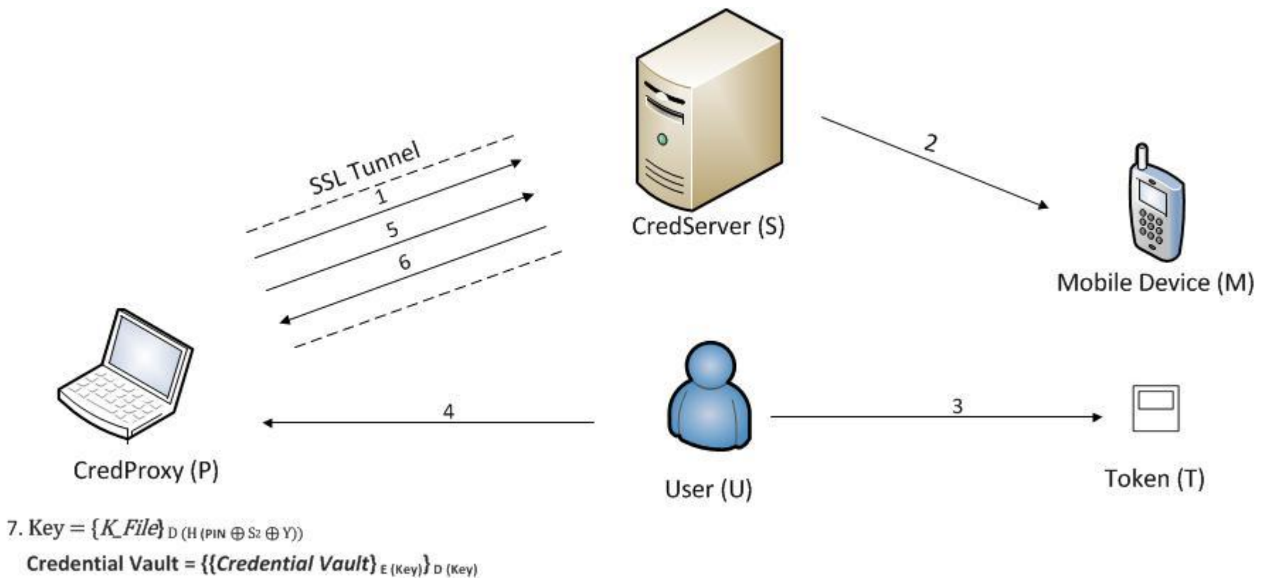
   $$P \leftarrow S: \{ID_2, Y, \text{Key}\}_{KPS}$$

6. P computes the following:

$$\{\textit{Credential Vault}\}_{E\ (Key)}$$

$$\{\textit{K\_File}\}_{E\ (H\ (PIN \oplus S_2 \oplus Y))}$$

5.4.3 Authentication Process 3



SSL Tunnel
1
5
6
2
CredServer (S)
Mobile Device (M)
4
3
CredProxy (P)
User (U)
Token (T)
7. Key = $\{\textit{K\_File}\}_{D\ (H\ (PIN \oplus S_2 \oplus Y))}$
Credential Vault = $\{\{\textit{Credential Vault}\}_{E\ (Key)}\}_{D\ (Key)}$

Figure 5.6: Authentication Process 3

1. U launches P on the computer. P contacts the server and establishes an SSL session. P then requests authentication through the following message:

$$S \leftarrow P: \{\textit{Auth\_Req}, ID_2\}_{KPS}$$

2. S receives the request, looks up ID_1 in its database and sends a one-time challenge C to U's cell phone through SMS.
3. U receives the text message on its cell phone, and inputs it on the token T.
4. U enters the hash response code, along with his PIN on P using the virtual keypad.
5. P sends the following message to S:

$$S \leftarrow P: \{H_4 = H\ (S_2 \oplus PIN) \oplus T\ (C)\}_{KPS}$$

6. P verifies S_2, PIN and T (C). Upon successful verification, the following message is sent:

$$P \leftarrow S: \{ID_2, Y\}_{KPS}$$

7. P computes as follows:

$$\text{Key} = \{K\_File\}_{D\,(H\,(PIN \oplus S_2 \oplus Y))}$$

$$\text{Credential Vault} = \{\{\textit{Credential Vault}\}_{E\,(Key)}\}_{D\,(Key)}$$

5.5 Security and Attack Analysis

We present the security and attack analysis of our Function CPGuard in this section. A number of attacks are discussed, and the security mechanisms and considerations are presented.

CredProxy Private Key Disclosure

As mentioned earlier, the connections between CredProxy and the browser are SSL encrypted. The certificates which are used between the browser and CredProxy are issued by CredProxy. If the private key of CredProxy were revealed, an adversary could decrypt the session between the browser and the CredProxy. Consequently, the alias username and password might be discovered. However, the alias username and password alone would not allow an adversary to use the CredProxy on the client computer; the PIN is also needed. CredProxy also establishes a “three strikes” rule where three unsuccessful login attempts will result in account lockout.

In addition, an adversary could simply sign other certificates with the CredProxy private key, and try to perform phishing on the user. However, as we have already mentioned, the CredProxy does not accept server certificates whose issuer is “CredProxy”, and simply blocks the session. Thus, phishing attempts using CredProxy-Signed certificates would be in vain.

Remote Desktop Attacks

An adversary may try to connect to the client computer through a remote desktop connection after the user has finished the authentication phase. In order to ward off such attacks, the user will be asked to enter his PIN on the virtual keypad every time the user logs on to his operating system, or resumes a session. In such a situation, even if an adversary successfully stole the user's alias username and password and also his operating system login credentials, he would not be able to use CredProxy on the user's computer. In addition, for more security CredProxy can be automatically locked when it is inactive after a certain amount of time (e.g. 10 minutes). The user can then input the PIN again to activate CredProxy on his computer.

Offline Password-guessing Attacks

Our model does not provide complete protection against such attacks. The key to the credential vault is encrypted using a combination of the hardware serial numbers of the client computer, the PIN and a random value of sufficient length and adequate entropy provided by the server; thus, the credential vault is node-locked. An adversary may find a way to steal the credential vault from the user's computer and perform offline dictionary attacks on the credential vault absolutely independently from the file containing the key to the credential vault. However, both the key and the value Y generated on the CredServer are of adequate entropy, which could enhance the security of the credential vault in the face of data theft from the client machine. . In other words, the only approach to increase the security of the credential vault with regards to data theft is to ensure that the encryption algorithm used for protecting the credential vault is very strong, and that it takes a long key of sufficient entropy as the input. In our model, we do not rely on the user to choose master passwords, yet CredProxy picks master passwords of sufficient entropy. In this way, the adversary would be forced to perform a substantial amount of work in order to crack the credential vault; consequently, our model provides a better solution against offline guessing attacks.

Man-in-the-Middle Attacks

An adversary may try to intercept the connections between S and P, P and the browser, and P and a server such as Gmail. However, all these communications are SSL-encrypted. Thus, it is highly unlikely that the attacker could decrypt messages exchanged through these SSL-encrypted tunnels.

Keylogging Attacks

Malware might try to log all the keystrokes of the user. In this way, it would only discover the alias username and the password, which is not enough to decrypt the credential vault. In fact, in order to decrypt and use the credential vault, the malware also needs to steal the PIN which is highly protected by our virtual on-screen keypad. It should be noted that the PIN could not be brute-forced since there is a “three strikes” policy in place.

Session Hijacking

A piece of malware on the client computer may discover the alias username and password, and wait until the authentication phase is complete. A malicious browser can then establish a session with a server through CredProxy using the user’s real credentials for that website. The user can be protected against such attacks by asking the user to type in his PIN every time he wants to use the credential vault. In this manner, every time CredProxy needs to translate the alias username and password to the user’s real credentials for a website, the virtual keypad pops up, and the user enters his PIN. As a direct result, the malware cannot exploit CredProxy after the authentication phase is complete.

Spoofed SMS Messages

An adversary may try to record or change the SMS messages sent by S in the third protocol. If an adversary recorded the SMS, it would not yield any useful information, since the attacker is not in the possession of T. An adversary may try to change SMS messages. This attack will be detected by S, and the user will be notified accordingly as well.

Phishing Attacks

CredProxy blocks phishing attacks by thoroughly examining certificates of the websites for which the user wants CredProxy to replace his alias username and password with the user’s real credentials. When the user signs up on a website or inputs his already-created credentials into the credential vault, the URL of the website along with the credentials will be stored in the credential vault. Therefore, at the time of login, CredProxy checks the certificate and the URL. If there are any problems with the certificate, CredProxy denies access to the website independently from the browser, and does not rely on the user to make the decision in the browser. A friendly notification by CredProxy, then, is displayed in the browser to notify the user of the potential danger.

Generation of the Key to the Credential Vault

As can be seen from our protocols, the key to the credential vault (stored in *K_File*) is generated by the CredServer. This feature can be useful in two ways. First, if the user forgot his PIN or changed his hardware due to any possible reasons, he would not be able to log in and use his credential vault. Storing *Key* on the CredServer can be useful in such situations. Eventually, the user is able to receive the key and use the credential vault again by providing enough proof to the service provider through the phone or any other out-of-band channels that he is the person he claims to be. Second, by generating keys by the CredServer, we make sure that the key is of adequate length and of sufficient entropy. This is of great significance here, since the only defence mechanism we have against offline dictionary attacks is the use of cryptographically secure encryption algorithms and keys of sufficient strength. In case of password managers which use a user-chosen master key, they are more susceptible to offline dictionary attacks (even in the case of using salts), since users do not usually choose passwords of enough entropy. Thus, generating keys on the CredServer can bring about more protection for the credential vault, and the user also does not have to memorize this key, but only the PIN. However, since the key to the credential vault is stored in *K_File*, the credential vault is as secure as the *K_File.* In other words, if an adversary could guess the key to *K_File* through probabilistic methods [70], the credential vault would be decrypted regardless of the strength of the *Key*. As mentioned earlier, the *K_File*, storing the key to credential vault, is encrypted using the PIN, S_2 and Y. An attacker might try to mount a brute-force attack on the CredProxy through the virtual keypad in order to obtain the PIN. However, after three strikes, the attacker would be locked out, and the user would be notified accordingly. An adversary might also be able to extract the PIN and S_2 from the user's device. Since he did not have the smartphone, He would not be able to perform authentication to the CredServer and obtain the key in order to decrypt the credential vault. As a result, he should still try to mount guessing attacks on the credential vault to crack it. However, it is the *Y* which plays an important role in the protection of the credential vault. Thus, the value *Y* should be of sufficient length and entropy.

5.6 Implementation of Function CPGuard

We have developed a prototype of Function CPGuard (the first protocol) using Java, PHP and Android SDK to evaluate our protocols. The client-computer code has been implemented in Java. The code for CredServer has been developed using PHP, and we have also used Microsoft IIS Server as the web server. PHP is a general-purpose scripting language which is much used for server-side scripting; moreover, one good reason we chose PHP and Java is their openness. As for the mobile-client code, we have used Android SDK [119] and Eclipse [120]. We have chosen Eclipse, since the recommended environment for developing Android applications is Eclipse with Android Development Toolkit (ADT) [121]plug-in. The aforementioned technologies are all free for commercial and non-commercial use.

The mobile code and the client code connect to CredServer though the HTTPS Protocol and the port number 443. An X509 certificate is used for SSL authentication and channel encryption between the CredServer and the client/mobile devices. On the server side, in order to generate random numbers we have used standard random number generation functions which are provided by PHP. These functions are seeded with a random value if no seed is provided. However, a good security practice would be to make sure that the random function generator is always seeded with a truly random seed. In fact, this is practicable if we could generate randomness which is not easily predictable. For this purpose, we could use some publicly available services such as Random.org [122]. Random.org is a website which offers true random numbers on the Internet which are generated using atmospheric noise. Random numbers from this site can be used as seeds in order to generate random numbers of more entropy. In addition, for all of the seeds (δ) to our functions used in Function CPGen and Function CPGuard, using random seeds from Random.org would yield more secure and much less predictable results.

In the client code, we have used AES to encrypt the credential vault and the file which holds the key to credential vault. The key size used for AES is 128 bits. We have also used MD5 with 128-bit outputs as the hashing algorithm in all of our code. Table5.2 shows the configuration of the computer we have used to run our tests on.

Processor	1 x Intel® Core™ i7 2620 2 Cores x 2.70 GHZ
Memory	4 Giga Bytes DDR3-1333MHZ
Intel® Smart Cache	4 Mega Bytes
Chipset	Intel® HM67 Express Chipset
Operating System	Microsoft™ Windows 7

Table 5.2: The Machine Configuration

For our testing, we used only one personal computer. The client code, mobile code and the server code were tested on the same computer. The mobile-server code took negligible time (the time was short enough that an average user would not notice it) both for the registration and authentication processes. The client-server code also took negligible time to fulfil its tasks.

5.7 Chapter Summary

This chapter has discussed Function CPGuard. Three different protocols and threat models have been proposed to secure the credential vault on the client computer. This chapter has also provided the security and attack analysis of these protocols and further mechanisms to provide more protection against malware. The chapter has concluded with the implementation data of Function CPGuard. The next chapter focuses on our virtual on-screen keyboard and keypad, and the protection benefits they provide to the user.

Chapter 6

Virtual Keypad and Keyboard

In this chapter, we present our virtual keypad and keyboard. Ordinary virtual on-screen keyboards could fall victim to shoulder surfing. However, we use a novel method in our virtual keypad and keyboard to ward off keylogging and shoulder surfing attacks.

6.1 Introduction

Virtual keyboards or on-screen keyboards are software applications that allow users to input data without using physical keyboards in situations where using a physical keyboard is either inconvenient or insecure. Tablets, as an example, enjoy small sizes and light weights; the attributes which usually characterize tablets. Using a typical physical keyboard might be inefficient or inconvenient for an average user who would like to have a small portable device. On the other hand, malware, particularly keyloggers, have turned the spotlight on PC virtual keyboards. PC virtual keyboards, which are software components, usually operated through the mouse (unlike tablets which usually make use of touch screens), are utilized to help thwart keylogging attacks. Keyloggers usually capture keyboard events based on the keys pressed or mouse clicks by the user. This data is later processed by an attacker, hoping to get some useful information out of it.

Using a virtual keyboard, the user does not have to type anymore; he just clicks the keys on the virtual keyboard. Then, these keys are transferred to the program which has had the focus at that moment. Of course, it is well known that with virtual keyboards there is a big problem. An adversary could easily develop a malicious piece of code which is invoked every time there is a mouse click event. This code would take a screenshot every time there is a mouse left click, which leaves the attacker with an ordered set of screenshots. These screenshots, which illustrate the positions of the virtual keyboard and the mouse when clicked, yielding the keys pressed on the keyboard, are processed by the attacker to extract user sensitive input.

Agarwal et al. [123] proposed a new virtual keyboard in 2011 to protect user input. The keyboard is depicted in figure 6.1.

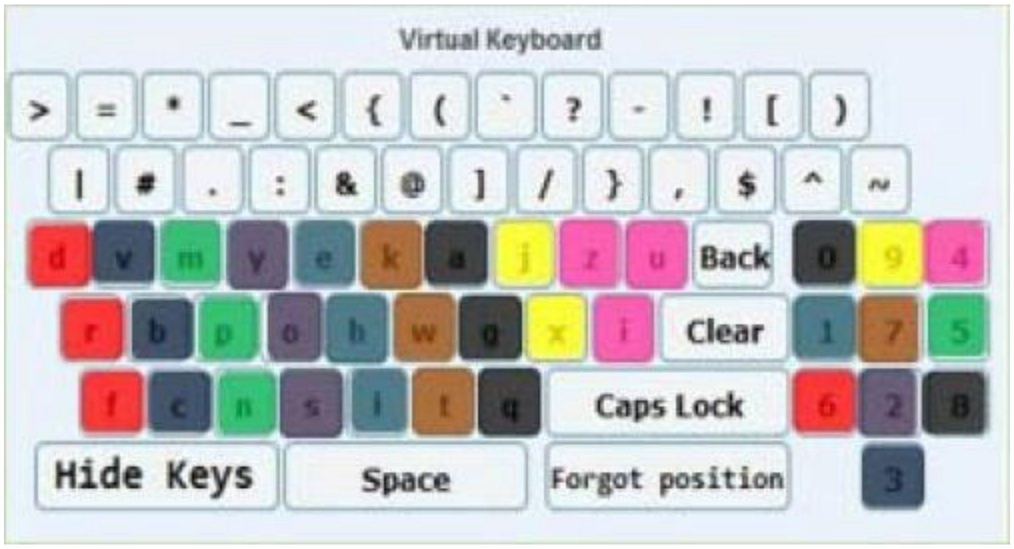

Virtual Keyboard
> = * _ < { (` ? - ! [ )
| # . : & @] / } , $ ^ ~
d v m y e k a j z u Back 0 9 4
r b p o h w g x i Clear 1 7 5
f c n s l t q Caps Lock 6 2 8
Hide Keys Space Forgot position 3

Figure 6.1 [123]**: Agarwal et al.'s Virtual Keyboard**

In this keyboard, when the user wants to click a key on the keyboard, he should first hide the keys. When "Hide keys" key is clicked, the captions of the keys disappear but the colors remain, leaving the user with a keyboard with no layout. Then, the user, who is supposed to memorize the position of the key, presses the appropriate key. If the user forgets the position of an intended key, he can turn on the caption again just by clicking on the "Forgot Position" key, and follow the same procedure.

6.2 Virtual keypad and Keyboard

Our keyboard, however, is more sophisticated. We do not use mouse clicks in order to press keys on the virtual keyboard; we use mouse hover events. A mouse hover or just hover is an event which is triggered when the mouse pointer is placed over a particular pre-defined area of the display. Basically for our virtual keyboard, the user has to hover his mouse pointer within the boundaries of an intended key for a randomized amount of time to have a key clicked. We also make use of automatic random appearance and disappearance of the keyboard layout along with key position shuffling and key color changing. In the following, we will elaborate more on our virtual keypad's functionality.

Figure 6.2 depicts the initial appearance of our keypad on the screen. It should be noted that this keypad is used to enter the pin which identifies the user to CredProxy.

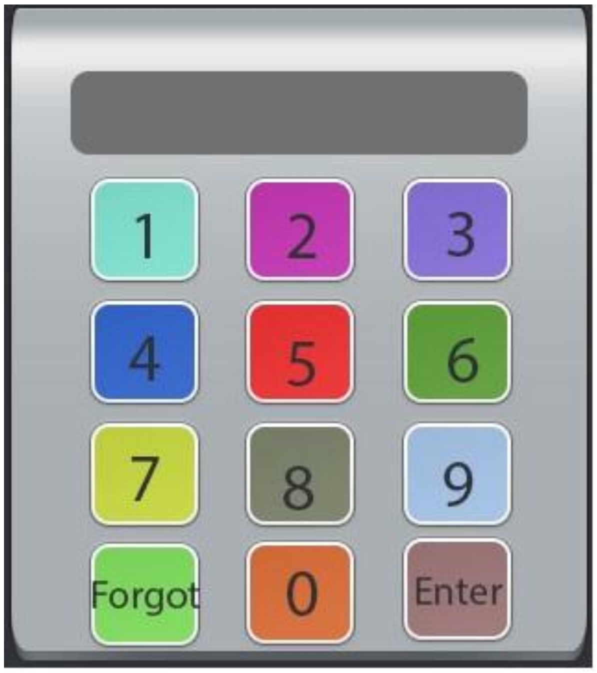


Figure 6.2: Initial Status of the Keypad

As can be seen in figure 6.2, this is the initial layout of our virtual keyboard. We use key coloring to help the user easily remember key positions. Here, the user is allowed to have one key pressed at a time. Suppose that the user needs to enter "123". When the keyboard pops up, the user is given some amount of time to memorize the location of the key. The amount of time T_1 is calculated as follows:

Min_1	Minimum amount of time to keep the keypad layout on
Max_1	Maximum amount of time to keep the keypad layout on
R	Random number generator (between 0 and 1)
σ	Random seed to randomize the random number generator
T_1	Amount of time to keep the layout on

$$T_1 = Min_1 + (Max_1 - Min_1) * R(\sigma)$$

Let suppose that Min = 4, Max = 6, and $(Max_1 - Min_1) * R(\sigma)$ = 1.327. In this case, the keyboard layout disappears after 5.327 seconds. The result can be seen in figure 6.3. Furthermore, the user is not allowed to click a key before the layout disappears. This limitation makes sure that whenever the user wants to have a key clicked on the keypad, the layout has already disappeared; thus, taking a screenshot during the mouse hovering would not yield any useful information to the adversary.

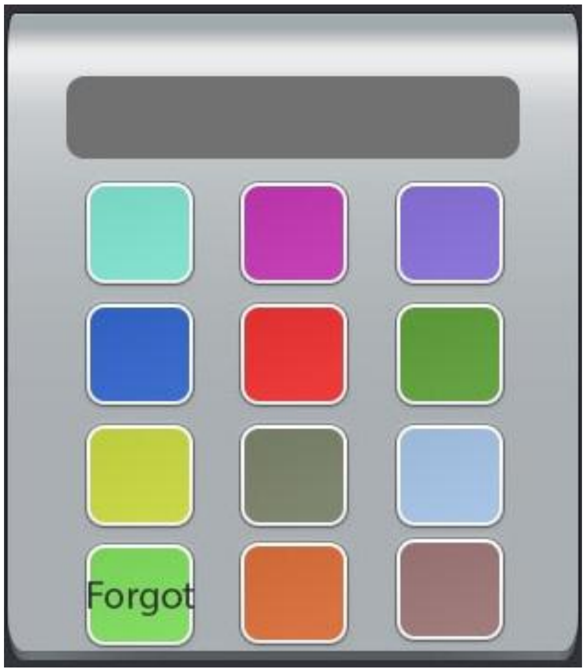


Figure 6.3: Keyboard Layout Disappeared

Now, the user who should remember the color and position of the intended key, does a mouse hover for T_2 amount of time over the key. T_2 is computed as follows:

Min_2	Minimum amount of time to hover the mouse pointer over a key
Max_2	Maximum amount of time to hover the mouse pointer over a key
R	Random number generator (between 0 and 1)
σ	Random seed to randomize the random number generator
T_2	Amount of time needed to hover on the current key in order to have a key clicked

$$T_2 = Min_2 + (Max_2 - Min_2) * R(\sigma)$$

Suppose $Min_2 = 1.8$, $Max_2 = 2.9$, $(Max_2 - Min_2) * R(\sigma) = 0.336$, Then the user has to hover the mouse over the key for $T_2 = 2.136$ seconds in order to have a key clicked. Should the user forget the position of the key he intends to press, he can click the "Forgot" key at any time. Pressing the Forgot key will make the keyboard layout appear; however, every time the layout shows up, the colors and key positions are different. Moreover, "Forgot" key is the only key, the writing on which never disappears, for the user must be able at all times to use this key to make the layout appear again.

After each key is clicked, the layout does not show up immediately. We again compute a random time after which the layout will appear for the user. Figure 6.4 depicts the keypad after each keystroke.

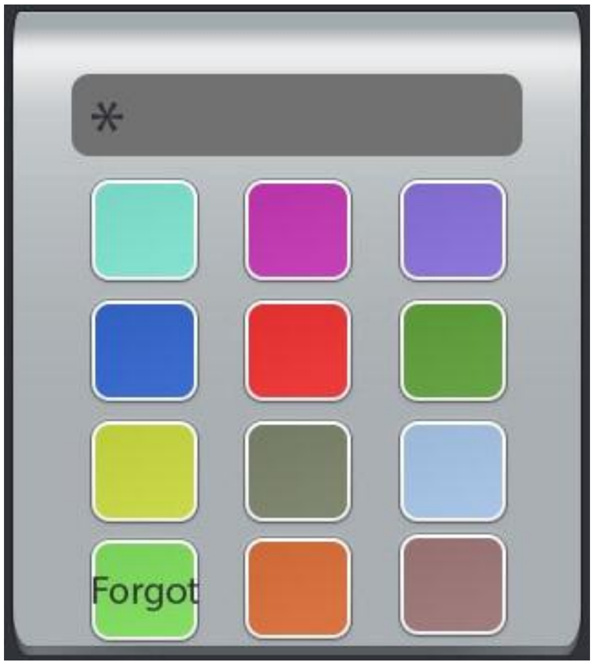


Figure 6.4: Keypad after Each Keystroke

The amount of time needed for the keypad to make the layout appear is computed through T_3 which is as follows:

Min_3 Minimum amount of time after which keypad layout shows up

Max_3	Maximum amount of time after which keypad layout shows up
R	Random number generator (between 0 and 1)
σ	Random seed to randomize the random number generator
T_3	Amount of time after which the keypad layout appears again

$$T_3 = Min_3 + (Max_3 - Min_3) * R(\sigma)$$

Suppose $Min_3 = 1.5$, $Max_3 = 3$, $(Max_3 - Min_3) * R(\sigma) = 0.751$, after $T_3 = 2.251$ seconds the layout appears again, which is depicted in figure 6.5.

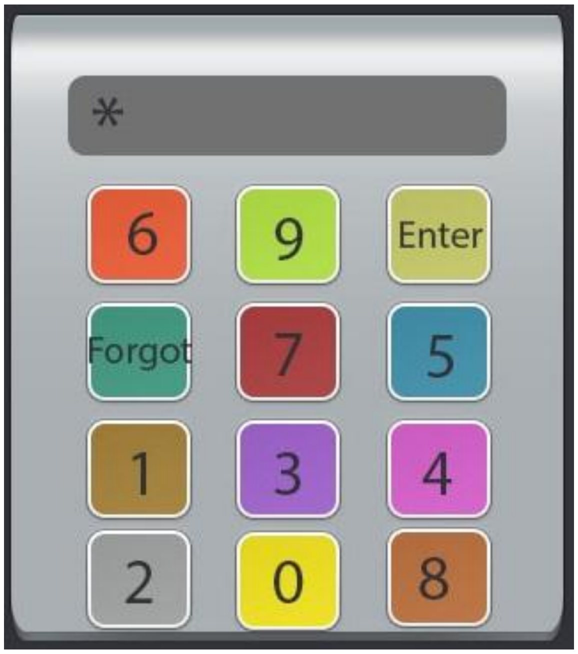


Figure 6.5: Keyboard Layout Appears Again

As can be seen in the figure 6.5, all the key positions and the key colors have changed. We shuffle key positions and change colors in every appearance of the layout. Figure 6.6 depicts the flow chart of our keyboard functionality.

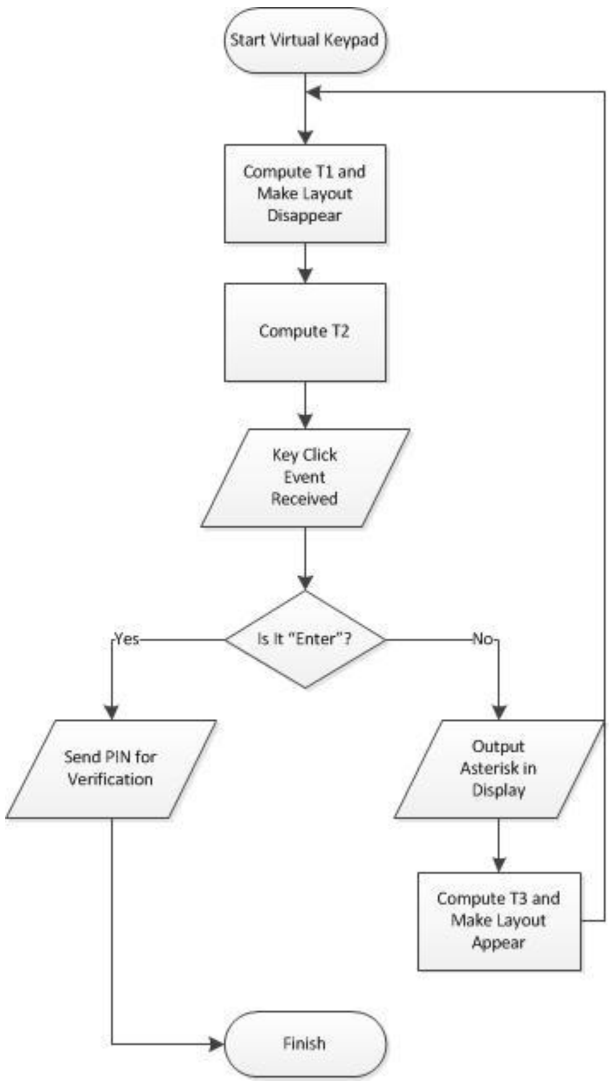

Start Virtual Keypad
Compute T1 and Make Layout Disappear
Compute T2
Key Click Event Received
Is It "Enter"?
Yes
No
Send PIN for Verification
Output Asterisk in Display
Compute T3 and Make Layout Appear
Finish

Figure 6.6: Flow Chart of Virtual Keypad Functionality

We believe our virtual keypad outperforms others in several ways in terms of security. Firstly, we don't use clicks; we use mouse hovers instead. Thus, click-based screen gabbing could be warded off. Second, we use randomized timers to make every key click, layout appearance and layout disappearance random in order to defeat shoulder surfing attacks. Third, we change all the key positions including "Enter" and "Forgot" keys to make it hard for attacks that use co-ordinate positioning techniques [123] to succeed. Lastly, we change key colors every time the keyboard layout changes, which makes it unfeasible for an attacker to use colors to extract the user's pin.

In addition, we have devised a full-fledged virtual keyboard which works the same as our virtual keypad. The layout of the keyboard is depicted in figure 6.7.

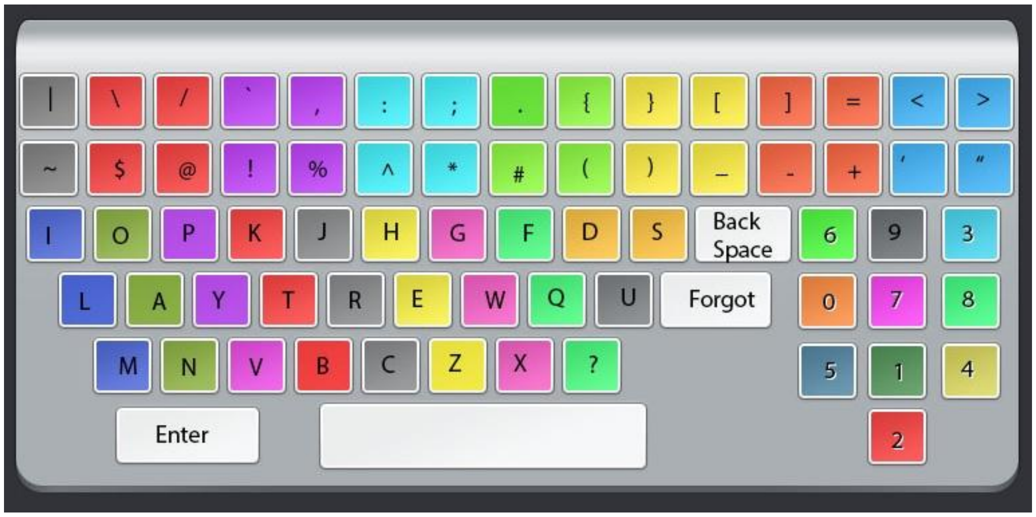


Figure 6.7: Virtual Keyboard

The virtual keyboard starts with a random layout which is highly unlikely to be the standard QWERTY layout. It waits for T_1 for the user to memorize the position and the color of the key. Then, the layout disappears as depicted in figure 12. The user is supposed to do a mouse hover over the boundaries of the intended key for T_2. When the key-click event is triggered, the layout will appear again after T_3. This process goes on until the user has completely input his data.

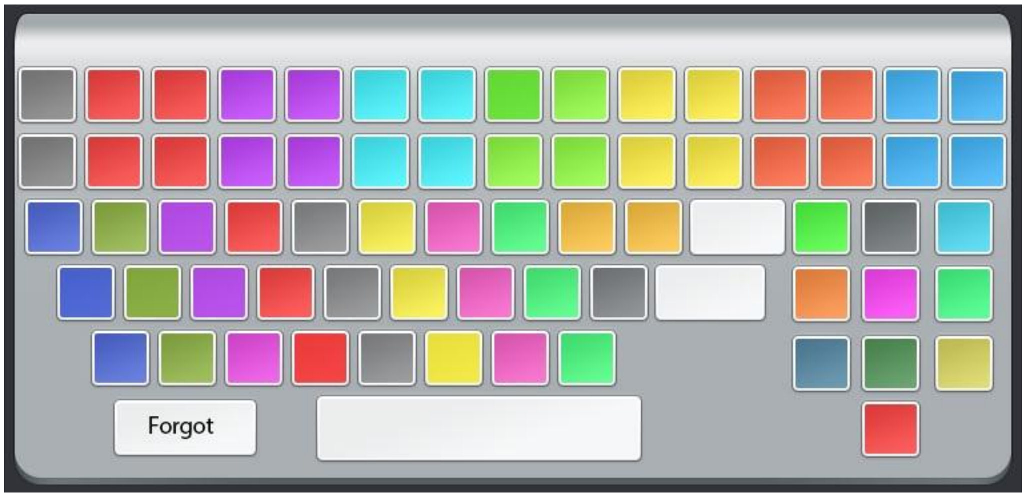


Figure 6.8: Virtual Keyboard without the Layout

As can be seen in figure 6.8, the only key that always keeps its writing on is the “Forgot” key so that whenever the user forgets a position and color, he can have the layout appear again just by having the “Forgot” key clicked.

As can be seen from figure 6.8, we use a specific coloring scheme to make remembering the key positions easier for the user. The symbol keys in the first two rows are colored in groups of four except for the last 2 keys in the top left corner. Moreover, the adjacent letter keys, which form a column with an inclination in the letter key section, are colored the same color in their three rows except for the last two keys in the third row and the last key in the fourth row. It should, however, be noted that although the key positions and colors change, the coloring scheme and key sections always stay the same. For instance, letter keys always appear in the letter key section and symbol keys always appear in symbol key section.

The virtual keyboard can be used to input sensitive data such user credentials into CredProxy credential vault as one of the ways to populate the credential vault. In the next section, we discuss the security analysis of our proposed approach.

6.3 Security and Attack Analysis

Security of the Virtual Keyboard and Keypad

Our virtual keypad and keyboard are not click-based; they are triggered by mouse hover events. Thus, click-based screen-grabbing attacks might be unsuccessful. Moreover, shuffling the keys and their colors on the keyboard and keypad provides our model with more security. An adversary could also take screenshots at certain intervals to extract the user’s PIN. In addition, the attacker may note the coordinates of mouse hovers to predict the sequence of keys clicked [123]. However, randomized timers used for the virtual keyboard and keypad also render these types of attacks ineffective. Our virtual keyboard and keypad, nevertheless, are not resistant to screen recording; that is, if some malware recorded the display while the user were typing in his PIN via the virtual keypad, the malware could steal user’s PIN. The same thing is true about the user credentials input through the virtual keyboard. However, we are unaware of any malware that has ever deployed such an attack.

Shoulder Surfing Attacks

An adversary may try to stand behind or near the user to shoulder-surf him while he is trying to enter the PIN. Our virtual keyboard and keypad would make it very difficult for an attacker to perform such an attack, since for every click on the keypad, the adversary must memorize the layout in order to be able to steal the user's PIN and other data input through the virtual keyboard and keypad. Nonetheless, if an adversary can record the display via a camera on a cell phone, etc., he can extract the user's PIN. Our model does not protect the user against the latter.

6.4 Implementation and Lessons Learned

We have implemented a prototype of our virtual on-screen keypad using Java. In our implementation, we have used standard Java APIs which are provided by Oracle [124]. We have done an informal usability test on our virtual on-screen keypad. For this purpose, we recruited 10 voluntary subjects including 7 engineering students and 3 non-engineering students. There were three six males and four females. All participants were acquainted with conventional on-screen keyboards. The participants were advised to input "159753" as the PIN on our virtual keypad. We also recruited another group consisting of five people. This group's task was to sit next to the participants and try to memorize the sequence of keys clicked by the participants; in other words, we simulated an environment where a shoulder-surfing attack could be performed. The Min/Max times for our algorithms were set to the values illustrated in table 6.1. All the values are in seconds.

T_1	T_2	T_3
$Min_1 = 3$	$Min_2=1.5$	$Min_3=1$
$Max_1 = 4$	$Max_2=2$	$Max_3=2$

Table 6.1: The Setup for T_1, T_2, and T3

Firstly, the participants were instructed how to use the keypad. Second, the participants were asked to use the keypad once in order for them to become more familiar with it. Then, we asked them to punch in the PIN while a participant from the second group was watching. We have measured the times taken by participants to input the aforementioned PIN. The times are illustrated in the graph in figure 6.9. Furthermore, none of the participants of the second group were able to correctly extract the sequence of the 6-digit PIN.

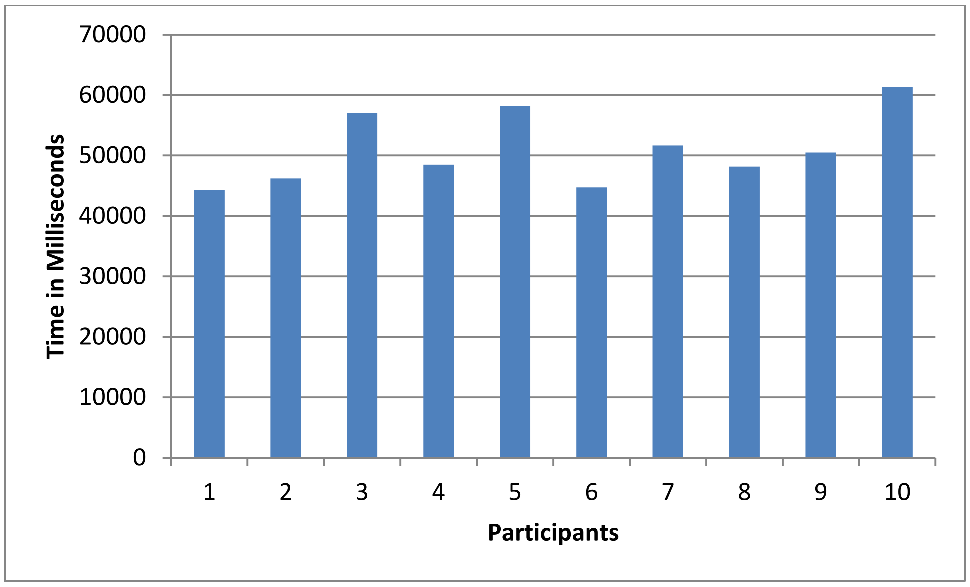


Figure 6.9: The Graph Illustrating the Times Measured

In a short discussion we had after each participant used our keypad, we learned some interesting things from their feedback. First, the participants agreed that after a few times they would probably have no problem using the keypad. Second, after we explained briefly about the security risks which could be thwarted by the keypad, they agreed that more time spent using our keypad would be worthwhile, and that they would like to use it. Lastly, one participant mentioned that alteration of T_2 is annoying, and that he would prefer a more deterministic approach. In other words, the change of T_2 after having every button clicked made it difficult for the participants to expect when a key was clicked and when they could go to the next key. An improvement to resolve this issue could be to calculate T_2 once in every instance of the keypad and make use of the same T_2 for the rest of the mouse hover events during one instance of authentication via the keypad. Another possible improvement could be to minimize the distance between Max_2 and Min_2 so that the alteration of T_2 could go unnoticed from the user's viewpoint. However, determining the Min/Max values in our algorithms would require extensive usability testing, without which the proper and optimal functioning of the keypad and the keyboard would not be feasible.

6.5 Chapter Summary

This chapter has introduced our virtual on-screen keyboard and keypad. The random appearance and disappearance of the layout along with "no-click" policy in our keyboard and keypad can provide protection against keylogging and shoulder-surfing. It has also provided some usability data of the keypad, and mentioned some ways to improve the performance of the keypad. The next chapter focuses on the future work, and concludes.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

We have presented the anatomy of a typical Man-in-the-Browser attack, and have brought to the attention of the reader some possible methods of malware infection and exploitation in the current world of computer technology and the Internet. We have defined the problem of securing user data input, and tried to emphasize some dangerous often-overlooked techniques of data theft both through psychological, otherwise known as social engineering, and technological approaches. Keylogging, phishing and MITB attacks are of great significance to the community of researchers in the field of computer security, since much data and capital have been lost due to these threats, and much to our disappointment, their detection and removal have proved to be a significant challenge. We have also presented a review of the literature which contained approaches to protect sensitive data input and user credentials along with password management approaches. We have briefly analysed some password managers, and have pointed out that password managers, which store the password database encrypted on the client machine, inherently suffer from weakly-chosen master passwords, since user-chosen passwords usually suffer from lack of adequate entropy. We have also pointed out some of their aspects which might be perceived as shortcomings.

Our model, CredProxy, has been presented in chapter 3. CredProxy is a method through which Man-in-the-Browser attacks can be mitigated. Furthermore, keylogging and phishing threats can be drastically alleviated using CredProxy. CredProxy takes a proactive step in performing all the verifications of SSL certificates, and does not rely on the browser and the user to make decisions in the critical situations. Not only does CredProxy offer such security mechanisms, but it also relieves the user from the burden of username/password generation and memorization.

In Chapter 4, we focused on generation of usernames and passwords automatically. We proposed Function CPGen to produce random usernames based on rules which can be tailored to meet the requirements of websites and also random passwords. This function is meant to improve anonymity in the usernames and security of passwords, which, as a result, can alleviates risks associated with weak usernames [15] and passwords.

Function CPGuard has been discussed in Chapter 5. The credential vault which is stored on the client side is guarded via this function. We have proposed three different protocols to protect the credential vault from data theft, the choice of which is highly dependent on the threat model and circumstances. In the first two protocols of Function CPGuard, a hand-held device is utilized as a second-factor authentication with the assumption of malware-free hand-held devices. However, the last protocol has accounted for the closer-to-reality fact that hand-held devices are no longer immune to malware. Therefore, based on the threat model and assumptions, one of these protocols can be selected. The experimental validation of this function has been also presented. The result showed that the time needed for Function CPGuard to perform the registration and authentication processes was negligible. However, we ran this function in our test environment which did not suffer from several real-world issues and delays.

Our proposed virtual on-screen keyboard and keypad have been presented in Chapter 6. Our model enjoys several elements of randomness, which makes current keylogging attacks unproductive. We have presented an informal usability study of our virtual on-screen keypad, and obtained interesting results. Our results also indicate that shoulder-surfing will be more difficult than expected when our keypad is deployed.

7.2 Future Work

The idealized model we presented for CredProxy has yet to be implemented. We have implemented and tested the virtual keypad and the Function CPGen and CPGuard; however, the full CredProxy needs to be implemented. We believe that the implementation of CredProxy, rather than supporting website login pages on an ad-hoc basis, should be customized for intended websites. In other words, due to using several different techniques for online login purposes (most login pages of websites do not send passwords in plain text; they might modify passwords or hash them using different algorithms before sending), CredProxy should be specifically tailored and adjusted to work properly with intended websites so that the replacement of the alias username and password with user credentials could be performed flawlessly. This could be perceived as a big limitation, since the user would not be able to use a website which is not supported by the CredProxy. As a result, future work should address this issue and find a way to mitigate it. Furthermore, CredProxy could be further extended to perform the same replacement function but for credit-card information. In this fashion, some other alias account could be used instead of the user's real credit-card information in the browser which would be identified and replaced by CredProxy when requested by the user.

In addition, the protocols presented in Chapter 5 should be tested in an actual cloud-based environment where the communication between the CredProxy and the CredServer can be analyzed and other security and performance caveats can be identified. Under the test environment, the protocols took negligible time to run; however, they have not been tested in an actual network. Doing so may help in understanding their performance and their usability impacts in practice.

Furthermore, the virtual keyboard and keypad should undergo intensive and extensive usability testing. Although, we have successfully implemented and tested the virtual on-screen keypad, they have to yet be fully tested to minimize the minimum and maximum times and to make it as user-friendly as possible. That is to say, future work should consider altering and perhaps reducing the minimum and maximum times used in Chapter 6 to make the total amount needed to enter the PIN as low as possible, as this could affect users' experience.

7.3 Concluding Remarks

We have explored the security issues of password managers and user sensitive data input, and proposed a model including different features and protocols to protect the user against the wicked world of cyber-crime in the context of password managers. Our model can bring about many features for the user and relieves the user from the traditional burdensome authentication tasks which have been around from the early age of the Internet.

Just one final note. Security is a collective term and concept. Several independent components of a computer system should come together in a complex organized manner for the security to occur. A failure of a component could simply lead to the failure of the whole system in terms of security, and could cause irreversible damages to both people and companies. One security flaw in a component could simply bring about the apocalypse of the whole system. Moreover, the effectiveness of social engineering techniques should not be underestimated; as Bruce Schneier says: “Only amateurs attack machines; professionals target people”.

Bibliography

[1] McAfee Inc, "A Good Decade for Cybercrime: McAfee’s Look Back at Ten Years of Cybercrime," 2010. [Online]. Available: http://www.mcafee.com/ca/resources/reports/rp-good-decade-for-cybercrime.pdf. [Accessed 17 09 2012].

[2] "Zero-day attack," Wekipedia, [Online]. Available: http://en.wikipedia.org/wiki/Zero-day_attack. [Accessed 05 08 2012].

[3] SafeNet, "Man-in-the-Browser Discussion Paper," 2010. [Online]. Available: http://www.paysecure.com.tw/Portals/0/new_files/ManInTheBrowserDiscussion_WP_(A4)_web.pdf. [Accessed 18 09 2012].

[4] Symantec, "Internet Security Threat Report, 2011 Trends, Vloume 17," 2012. [Online]. Available: https://www4.symantec.com/mktginfo/downloads/21182883_GA_REPORT_ISTR_Main-Report_04-11_HI-RES.pdf. [Accessed 18 09 2012].

[5] S. M. Smyth and R. Carleton, "Measuring the Extent of Cyber-Fraud: A Discussion Paper on Potential Methods and Data Sources," August 2011. [Online]. Available: http://www.publications.gc.ca/collections/collection_2011/sp-ps/PS14-4-2011-eng.pdf. [Accessed 21 09 2012].

[6] Symantec, "Symantec Internet Security Threat Report Trends for 2010, Volume 16," 2011. [Online]. Available: https://www4.symantec.com/mktginfo/downloads/21182883_GA_REPORT_ISTR_Main-Report_04-11_HI-RES.pdf. [Accessed 21 09 2012].

[7] C. Herley, P. C. Van Oorschot and A. S. Patrick, "Passwords: If We're So Smart,Why Are We Still Using Them?," in *Financial Cryptography and Data Security, 13th International Conference*, Accra Beach, Barbados, 2009.

[8] S. Gaw and E. W. Felten, "Password management strategies for online accounts," in *Proceeding SOUPS '06 Proceedings of the second symposium on Usable privacy and security*, 2006.

[9] Symantec, "The 11 most common computer security threats… And what you can do to

protect yourself from them.," [Online]. Available: http://www.symantec-norton.com/11-most-common-computer-security-threats_k13.aspx. [Accessed 05 08 2012].

[10] . S. Sagiroglu and G. Canbek, "Keyloggers," *Technology and Society Magazine, IEEE,* vol. 28, no. 3, pp. 10-17, 2009.

[11] Nicolas Falliere; Eric Chien, "Zeus: King of the Bots," 2009. [Online]. Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/zeus_king_of_bots.pdf. [Accessed 21 09 2012].

[12] "Zeus (Trojan horse)," [Online]. Available: http://en.wikipedia.org/wiki/Zeus_(Trojan_horse)#cite_note-13. [Accessed 12 06 2012].

[13] "ZeuS-style banking Trojans seen as greatest threat to online banking: Survey," [Online]. Available: http://www.networkworld.com/news/2010/120810-trojan-bank.html. [Accessed 12 06 2012].

[14] B. Ross, C. Jackson, N. Miyake, D. Boneh and J. C. Mitchell, "Stronger password authentication using browser extensions," in *SSYM'05 Proceedings of the 14th conference on USENIX Security Symposium*, 2005.

[15] D. Florêncio, C. Herley and B. Coskun, "Do strong web passwords accomplish anything?," HOTSEC'07 Proceedings of the 2nd USENIX workshop on Hot topics in security, 2007.

[16] M. Jost, "How to Defeat the Two-factor Authentication-Killing Malware," [Online]. Available: http://www.symantec.com/connect/blogs/how-defeat-two-factor-authentication-killing-malware. [Accessed 13 09 2012].

[17] "W32.Duqu," 2011. [Online]. Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_duqu_the_precursor_to_the_next_stuxnet.pdf. [Accessed 13 09 2012].

[18] N. Falliere, L. O. Murchu and E. Chien, "W32.Stuxnet Dossier," 2011. [Online]. Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf. [Accessed 13 09 2012].

[19] "Code Injection," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Code_injection. [Accessed 17 07 2012].

[20] "JailbreakMe," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/JailbreakMe.

[Accessed 17 07 2012].

[21] "Safari (web browser)," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Safari_(web_browser). [Accessed 17 07 2012].

[22] "iOS," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/IOS. [Accessed 17 07 2012].

[23] "iOS Jailbreaking," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/IOS_jailbreaking. [Accessed 17 07 2012].

[24] J. M. McCune, A. Perrig and M. K. Reiter, "Bump in the ether: a framework for securing sensitive user input," in *ATEC '06 Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, 2006.

[25] C. Herley and D. Florencio, "How To Login From an Internet Caf´e Without Worrying About Keyloggers," in *Symposium on Usable Privacy and Security (SOUPS)*, 2006.

[26] "The Secure Sockets Layer (SSL) Protocol Version 3.0," IETF, [Online]. Available: http://tools.ietf.org/html/rfc6101. [Accessed 24 07 2012].

[27] M. Marlinspike, "New Tricks in Defeating SSL in Practice," [Online]. Available: http://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf. [Accessed 06 08 2012].

[28] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri and L. F. Cranor, "Crying wolf: an empirical study of SSL warning effectiveness," in *Proceeding SSYM'09 Proceedings of the 18th conference on USENIX security symposium*, 2009.

[29] N. L. Clarke and S. M. Furnell, "Authenticating mobile phone users using keystroke analysis," *International Journal of Information Security,* vol. 6, no. 1, pp. 1-14, 2006.

[30] "How Many Mobile Phone Users in the World," [Online]. Available: http://www.howmanyarethere.org/how-many-mobile-phone-users-in-the-world/. [Accessed 17 07 2012].

[31] P. Ruggiero and J. Foote, "Cyber Threats to Mobile Phones," United States Computer Emergency Readiness Team, 2011.

[32] "United States Computer Emergency Readiness Team," [Online]. Available:

http://www.us-cert.gov/. [Accessed 17 07 2012].

[33] McAfee® Labs™, "McAfee Threats Report: Third Quarter 2011," 2011. [Online]. Available: http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q3-2011.pdf. [Accessed 21 09 2012].

[34] M. Mannan and P. C. van Oorschot, "Using a personal device to strengthen password authentication from an untrusted computer," in *FC'07/USEC'07 Proceedings of the 11th International Conference on Financial cryptography and 1st International conference on Usable Security*, 2007.

[35] B. Parno, C. Kuo and A. Perrig, "Phoolproof phishing prevention," in *FC'06 Proceedings of the 10th international conference on Financial Cryptography and Data Security*, 2006.

[36] RSA, "MAKING SENSE OF MAN-IN-THE-BROWSER ATTACKS," 2011. [Online]. Available: http://www.julesbartow.com/Downloads/Making_Sense_of_Man-in-the-Browser_Attacks.pdf. [Accessed 12 06 2012].

[37] Entrust, "Defeating Man-in-the-Browser," 2010. [Online]. Available: http://download.entrust.com/resources/download.cfm/24002/. [Accessed 17 09 2012].

[38] "Google Authenticator," [Online]. Available: http://support.google.com/a/bin/answer.py?hl=en&answer=1037451. [Accessed 12 06 2012].

[39] "Social engineering (security)," [Online]. Available: http://en.wikipedia.org/wiki/Social_engineering_(security). [Accessed 14 06 2012].

[40] "Scareware," [Online]. Available: http://en.wikipedia.org/wiki/Scareware. [Accessed 22 09 2012].

[41] Greg Aaron; Rod Rasmussen; Aaron Routt, "Global Phishing Survey: Trends and Domain Name Use in 2H2011," 2012. [Online]. Available: http://www.antiphishing.org/reports/APWG_GlobalPhishingSurvey_2H2011.pdf. [Accessed 21 09 2012].

[42] F.-H. Hsu, C.-K. Tso, Y.-C. Yeh, W.-J. Wang and L.-H. Chen, "BrowserGuard: A Behavior-Based Solution to Drive-by-Download Attacks," *Selected Areas in Communications, IEEE Journal on,* vol. 29, no. 7, pp. 1461-1468, 2011.

[43] Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinte, L.; Leach, P.; Berners-Lee, T., "Hypertext Transfer Protocol -- HTTP/1.1," IETF, [Online]. Available: http://www.w3.org/Protocols/rfc2616/rfc2616.html. [Accessed 21 09 2012].

[44] E. Rescorla, "HTTP over TLS," IETF, [Online]. Available: http://www.ietf.org/rfc/rfc2818.txt. [Accessed 22 09 2012].

[45] P. Gühring, "Concepts against Man-in-the-Browser Attacks," 2007. [Online]. Available: http://www.cacert.at/svn/sourcerer/CAcert/SecureClient.pdf. [Accessed 22 09 2012].

[46] Fordham University and the Federal Bureau of Investigation, "Cyber War and Cyber Crime: What Your Anti-Virus Software Doesn't See," 06 01 2012. [Online]. Available: http://www.fbi.gov/newyork/press-releases/2012/cyber-war-and-cyber-crime-what-your-anti-virus-software-doesnt-see. [Accessed 14 06 2012].

[47] A. O. Freier, P. Karlton and P. C. Kocher, "The SSL Protocol: Version 3.0," Netscape communications, November 1996. [Online]. Available: http://tools.ietf.org/html/draft-ietf-tls-ssl-version3-00. [Accessed 22 09 2012].

[48] J. Leyden, "ZeuS attacks mobiles in bank SMS bypass scam," 27 09 2010. [Online]. Available: http://www.theregister.co.uk/2010/09/27/zeus_mobile_malware/. [Accessed 27 06 2012].

[49] "ZeuS Mitmo: Man-in-the-mobile (III)," [Online]. Available: http://securityblog.s21sec.com/2010/09/zeus-mitmo-man-in-mobile-iii.html. [Accessed 27 06 2012].

[50] "ZeuS Mitmo Strikes Again: Polish ING Bank," [Online]. Available: http://www.f-secure.com/weblog/archives/00002104.html. [Accessed 27 06 2012].

[51] "Symantec," [Online]. Available: http://www.symantec.com. [Accessed 28 06 2012].

[52] "Zeusbot/Spyeye P2P Updated, Fortifying the Botnet," [Online]. Available: http://www.symantec.com/connect/blogs/zeusbotspyeye-p2p-updated-fortifying-botnet. [Accessed 28 06 2012].

[53] "Trojan.Spyeye," [Online]. Available: http://www.symantec.com/security_response/writeup.jsp?docid=2010-020216-0135-99. [Accessed 28 06 2012].

[54] Harshit Nayyar, "Clash of the Titans: ZeuS v SpyEye," 2010. [Online]. Available: http://www.sans.org/reading_room/whitepapers/malicious/clash-titans-zeus-spyeye_33393. [Accessed 21 09 2012].

[55] "New Russian botnet tries to kill rival," 09 02 2010. [Online]. Available: http://www.computerworld.com/s/article/9154618/New_Russian_botnet_tries_to_kill_rival. [Accessed 28 06 2012].

[56] A. Biryukov, A. Shamir and D. Wagner, "Real Time Cryptanalysis of A5/1 on a PC," in *Fast Software Encryption Workshop 2000*, 2000.

[57] E. Barkan and E. Biham, "Conditional Estimators: An Effective Attack on A5/1," *Selected Areas in Cryptography,Springer LNCS,* vol. 3897, pp. 1-19, 2006.

[58] Trusted Computing Group, "TPM Main Specification Level 2 Version 1.2, Revision 116," March 2011. [Online]. Available: http://www.trustedcomputinggroup.org/resources/tpm_main_specification.

[59] R. Sailer, X. Zhang, T. Jaeger and L. van Doorn, "Design and implementation of a TCG-based integrity measurement architecture," in *SSYM'04 Proceedings of the 13th conference on USENIX Security Symposium*.

[60] "Rootkit," [Online]. Available: http://en.wikipedia.org/wiki/Rootkit. [Accessed 30 06 2012].

[61] W. A. Arbaugh, D. J. Farbert and J. M. Smith, "A secure and reliable bootstrap architecture," in *SP '97 Proceedings of the 1997 IEEE Symposium on Security and Privacy*, 1997.

[62] "Trusted Platform Module," [Online]. Available: http://en.wikipedia.org/wiki/Trusted_Platform_Module. [Accessed 30 06 2012].

[63] "Personal Digital Assistant," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Personal_digital_assistant. [Accessed 01 07 2012].

[64] "Pharming," [Online]. Available: http://en.wikipedia.org/wiki/Pharming. [Accessed 03 07 2012].

[65] H. Wang, Y. Guo, X. Zhao and X. Chen, "Keep Passwords Away from Memory: Password Caching and Verification Using TPM," in *AINA '08 Proceedings of the 22nd International*

Conference on Advanced Information Networking and Applications, 2008.

[66] C. Hargreaves and H. Chivers, "Recovery of Encryption Keys from Memory Using a Linear Scan," in *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, 2008.

[67] "HeapMemView v1.02," [Online]. Available: http://www.nirsoft.net/utils/heap_memory_view.html. [Accessed 03 07 2012].

[68] M. Bellare, R. Canetti and H. Krawczyk, "Keying Hash Functions for Message Authentication," in *CRYPTO '96 Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, 1996.

[69] "Replay attack," [Online]. Available: http://en.wikipedia.org/wiki/Replay_attack. [Accessed 04 07 2012].

[70] M. Dell'Amico, M. Dell'Amico and Y. Roudier, "Password Strength: An Empirical Analysis," in *INFOCOM, 2010 Proceedings IEEE*, Sophia Antipolis, France, 2010.

[71] J. J. Yan, "A Note on Proactive Password Checking," in *NSPW '01 Proceedings of the 2001 workshop on New security* , 1990.

[72] "Message authentication code," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Message_authentication_code. [Accessed 06 07 2012].

[73] W. Diffie and M. Hellman, "New directions in cryptography," in *Information Theory, IEEE Transactions on*, 1976.

[74] R. Rivest, "The MD5 Message-Digest Algorithm," April 1992. [Online]. Available: http://tools.ietf.org/html/rfc1321.

[75] " US Secure Hash Algorithm 1," September 2001. [Online]. Available: http://tools.ietf.org/html/rfc3174.

[76] H.-M. Sun, Y.-H. Chen and Y.-H. Lin, "oPass: A User Authentication Protocol Resistant to Password Stealing and Password Reuse Attacks," *Information Forensics and Security, IEEE Transactions on,* vol. 7, no. 2, pp. 651- 663, 2012.

[77] "Dictionary Attack," [Online]. Available: http://en.wikipedia.org/wiki/Dictionary_attack. [Accessed 09 07 2012].

[78] D. Florencio and C. Herley, "A large-scale study of web password habits," in *WWW '07 Proceedings of the 16th international conference on World Wide Web*, New York, 2007.

[79] L. Lamport, "Password authentication with insecure communication," *Communications of the ACM,* vol. 24, no. 11, pp. 770 - 772, 1981.

[80] H. Gilbert and H. Handschuh, "Security Analysis of SHA-256 and Sisters," *Selected Areas in Cryptography, Springer LNCS,* pp. 175-193, 2004.

[81] B. Ross, C. Jackson, N. Miyake, D. Boneh and J. C. Mitchell , "Stronger password authentication using browser extensions," in *Proceeding SSYM'05 Proceedings of the 14th conference on USENIX Security Symposium*, 2005.

[82] O. Goldreich, S. Goldwasser and S. Micali, "How to construct random functions," *Journal of the ACM (JACM),* vol. Volume 33, no. Issue 4, pp. 792-807, 1986.

[83] "PwdHash," [Online]. Available: https://www.pwdhash.com/. [Accessed 08 09 2012].

[84] J. A. Halderman, B. Waters and E. W. Felten, "A convenient method for securely managing passwords," in *WWW '05 Proceedings of the 14th international conference on World Wide Web*, 2005.

[85] H. Bojinov, E. Bursztein, X. Boyen and D. Boneh, "Kamouflage: loss-resistant password management," in *Proceeding ESORICS'10 Proceedings of the 15th European conference on Research in computer security*, 2010.

[86] D. McCarney, D. Barrera, J. Clark, S. Chiasson and P. van Oorchot, "Tapas: Design, Implementation, and Usability Evaluation of a Password Manager," in *Annual Computer Security Applications Conference (ACSAC)*, 2012.

[87] E. Gabber, P. B. Gibbons, Y. Matias and A. J. Mayer, "How to Make Personalized Web Browising Simple, Secure, and Anonymous," in *FC '97 Proceedings of the First International Conference on Financial Cryptography*, 1997.

[88] "Cyber Crime Costs $114B Per Year, Mobile Attacks on the Rise," PC Magazine, [Online]. Available: http://www.pcmag.com/article2/0,2817,2392570,00.asp. [Accessed 21 07 2012].

[89] "AWPG," [Online]. Available: http://www.antiphishing.org/. [Accessed 21 07 2012].

[90] "DNS Cache Poisoning Used in Brazilian Phishing Attack," [Online]. Available: http://news.softpedia.com/news/DNS-Cache-Poisoning-Used-in-Brazilian-Phishing-Attack-212328.shtml. [Accessed 21 07 2012].

[91] "Santander Bank," [Online]. Available: http://www.santander.com. [Accessed 21 07 2012].

[92] Sans, "Survival Time," [Online]. Available: https://isc.sans.edu/survivaltime.html. [Accessed 21 07 2012].

[93] "Google JavaScript Keylogger," [Online]. Available: http://code.google.com/p/javascript-keylogger/. [Accessed 21 07 2012].

[94] Daniel Hoffman, "Juniper Mobile Security Report 2011 – Unprecedented Mobile Threat Growth," [Online]. Available: http://forums.juniper.net/t5/Security-Mobility-Now/Juniper-Mobile-Security-Report-2011-Unprecedented-Mobile-Threat/ba-p/129529. [Accessed 22 07 2012].

[95] Z. Mao and C. Herley, "A Robust Link-Translating Proxy Server Mirroring the Whole Web," *Newsletter ACM SIGAPP Applied Computing Review,* vol. 11, no. 2, pp. 30-42, 2011.

[96] "Microsoft ISA Server 2006," [Online]. Available: http://technet.microsoft.com/library/bb898433.aspx. [Accessed 22 07 2012].

[97] "List of websites blocked in the People's Republic of China," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/List_of_websites_blocked_in_the_People's_Republic_of_China. [Accessed 22 07 2012].

[98] "BitTorrent," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/BitTorrent. [Accessed 22 07 2012].

[99] "utorrent," [Online]. Available: http://www.utorrent.com/. [Accessed 22 07 2012].

[100] SANS Institute, "Personal Proxy - Online Privacy Protection for Home Users".

[101] "Privoxy," [Online]. Available: http://www.privoxy.org/. [Accessed 22 07 2012].

[102] P. G. Inglesant and M. A. Sasse, "The true cost of unusable password policies: password use in the wild," in *CHI '10 Proceedings of the 28th international conference on Human*

factors in computing systems, 2010.

[103] "HTML 4 — Conformance: requirements and recommendations," W3C, [Online]. Available: http://www.w3.org/TR/html4/conform.html#h-4.2. [Accessed 24 07 2012].

[104] "HTML," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/HTML#cite_note-deprecated-0. [Accessed 24 07 2012].

[105] "W3C," [Online]. Available: http://www.w3.org/. [Accessed 24 07 2012].

[106] "HTTP Over TLS," IETF, [Online]. Available: http://tools.ietf.org/html/rfc2818. [Accessed 24 07 2012].

[107] "The Transport Layer Security (TLS) Protocol," IETF, [Online]. Available: http://tools.ietf.org/html/rfc5246. [Accessed 24 07 2012].

[108] "HTTP Secure," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/HTTP_Secure. [Accessed 24 07 2012].

[109] "Internet X.509 Public Key Infrastructure Certificate and CRL Profile," IETF, [Online]. Available: http://www.ietf.org/rfc/rfc2459.txt. [Accessed 24 07 2012].

[110] T. Chomsiri, "HTTPS Hacking Protection," in *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, 2007.

[111] SANS Institute , "SSL Man-in-the-Middle Attacks," [Online]. Available: http://www.sans.org/reading_room/whitepapers/threats/ssl-man-in-the-middle-attacks_480. [Accessed 25 07 2012].

[112] "Self-signed certificate," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Self-signed_certificate. [Accessed 25 07 2012].

[113] S. Komanduri, R. Shay, P. G. Kelley, M. L. Mazurek, L. Bauer, N. Christin, L. F. Cranor and S. Egelman, "Of Passwords and People: Measuring the Effect of Password-Composition Policies," in *Proceeding CHI '11 Proceedings of the 2011 annual conference on Human factors in computing systems*, New York, NY, USA , 2011.

[114] K. Helkala and E. Snekkenes, "Password Generation and Search Space Reduction," *Journal of Computers,* vol. 4, no. 7, pp. 663-669, 2009.

[115] "LinkedIn," [Online]. Available: http://ca.linkedin.com/. [Accessed 01 08 2012].

[116] "Millions of LinkedIn passwords reportedly leaked online," cnet.com, [Online]. Available: http://news.cnet.com/8301-1009_3-57448079-83/millions-of-linkedin-passwords-reportedly-leaked-online/. [Accessed 01 08 2012].

[117] "Twiter.com," [Online]. Available: https://twitter.com/. [Accessed 01 08 2012].

[118] "Analyzing the Crossover Virus: The First PC to Windows Handheld Cross-infector," [Online]. Available: http://www.informit.com/articles/article.aspx?p=458169. [Accessed 04 08 2012].

[119] "Android SDK," [Online]. Available: http://developer.android.com/sdk/index.html. [Accessed 10 09 2012].

[120] "Eclipse IDE," [Online]. Available: http://www.eclipse.org/. [Accessed 10 09 2012].

[121] "Android Development Toolkit Plugin," [Online]. Available: http://developer.android.com/tools/sdk/eclipse-adt.html. [Accessed 10 09 2012].

[122] "Random.org," [Online]. Available: https://www.random.org/. [Accessed 10 09 2012].

[123] M. Agarwal, M. Mehra, R. Pawar and D. Shah, "Secure authentication using dynamic virtual keyboard layout," in *ICWET '11 Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, 2011.

[124] "Oracle," [Online]. Available: http://www.oracle.com/index.html. [Accessed 09 09 2012].

[125] E. Messmer. [Online]. Available: http://www.networkworld.com/news/2010/120810-trojan-bank.html. [Accessed 11 06 2012].

[126] "Number of Cell Phones Worldwide Hits 4.6B," cbcnews.com, [Online]. Available: http://www.cbsnews.com/2100-500395_162-6209772.html. [Accessed 17 07 2012].

[127] "Hypertext Transfer Protocol -- HTTP/1.1," [Online]. Available: http://tools.ietf.org/html/rfc2616. [Accessed 24 07 2012].

[128] S. Hong, J. Oh, H. Lee and Y. Won, "One Touch Logon: Replacing Multiple Passwords with Single Fingerprint Recognition," in *Computer and Information Technology, 2006. CIT '06. The Sixth IEEE International Conference on*, 2006.

[129] "Get Started Developing for Android with Eclipse," smashingmagazine.com, [Online]. Available: http://coding.smashingmagazine.com/2010/10/25/get-started-developing-for-android-with-eclipse/. [Accessed 10 09 2012].

[130] "OpenSSL PHP," [Online]. Available: http://php.net/manual/en/book.openssl.php. [Accessed 10 09 2012].

[131] D. Recordon and . B. Fitzpatrick, "OpenID Authentication 1.1," May 2006. [Online]. Available: http://openid.net/specs/openid-authentication-1_1.txt. [Accessed 11 09 2012].

[132] "A Look Back at 2011," 2012. [Online]. Available: http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/reports/rpt_a-look-back-at-2011_information-is-currency.pdf. [Accessed 17 09 2012].

[133] PricewaterhouseCoopers LLP, "Fighting Economic Crime in the Financial Services sector," [Online]. Available: http://www.pwc.com/en_GX/gx/economic-crime-survey/pdf/fighting-economic-crime-in-the-financial-services-sector.pdf. [Accessed 22 09 2012].

[134] S. Gaw and E. W. Felten, "Password management strategies for online accounts," in *SOUPS '06 Proceedings of the second symposium on Usable privacy and security*, 2006 .