

Discovering Vulnerabilities in Web Browser Extensions Contained by Google Chrome

Chapin A. Johnson
Department of CSIT
Saint Cloud State University
St. Cloud, Minnesota, 56301
cajohnson5@go.stcloudstate.edu

Sharveen Paramiswaran
Department of CSIT
Saint Cloud State University
St. Cloud, Minnesota, 56301
sharveen.paramiswaran@go.stcloudstate.edu

Akalanka B. Mailewa
Department of CSIT
Saint Cloud State University
St. Cloud, Minnesota, 56301
amailewa@stcloudstate.edu

Abstract

In today's world, web browsers are used by most everyone daily. Many of us take this incredible functionality for granted, and don't recognize the potential risks that are involved with simple internet use. These risks exist in both the browsers and their extensions. Google Chrome has cemented itself as one of the go-to browsers for both commercial and everyday consumer use. In fact, Google Chrome is currently the most used web browser in the world. But with such heavy global use, comes a feeling of false security. Just like many applications and browsers that are widely available and free to use, Google Chrome has its weaknesses and vulnerabilities. These exist within the browser itself, but for the purposes of this research, Chrome's extensions will be the main focus. This research explores the potential risks that are involved in utilizing browser extensions for Google Chrome. We will also look at a variety of attacks that extensions can execute, and exactly how they work. These attacks aren't guaranteed to cause malicious behavior, but we will also discuss ways of increasing user safety when operating a web browser, and specifically browser extensions. The objective of this research is to test a variety of different attacks using browser extensions on Google Chrome. By researching and implementing a variety of different attacks, authors plan to find where Chrome is susceptible to allowing malicious extensions. This research will help to inform browser users of the dangers that exist when using extensions, and how threat actors may be deceiving them to perform malicious activities on their computers. This research also shows the different vulnerabilities that exist within browsers, and demonstrates the privileges that browsers have on a computer. It is expected the research output to show that browser vulnerabilities aren't a one size fits all type of attack and also expected some websites to have lower levels of protection allowing for poor security against malicious extensions.

Keywords: Vulnerabilities; Security; Risk; Google-Chrome; Attacks; DOM; Passwords; Browser-Extensions

1 INTRODUCTION

The problem that we have identified in this research is the significant vulnerabilities that browser extensions can exploit [1][2]. The threats we intend to identify are applicable to any chrome user who utilizes extensions. According to Google's 2020 statistics, most users have at least one extension installed on their browser, and with over 60% of internet users saying they prefer Chrome as their browser of choice, this has massive implications [3]. Most users who download an extension assume that they are safe, though this is far from the truth. Extensions have significant access to the browser itself, along with limited access to the computer they're installed on [4]. In the world of technology, it's important to stay vigilant when browsing online as there is a constant threat affecting users like school students, all the way up to business executives. In order to better protect ourselves when using Google Chrome, we must first understand how Chrome can be vulnerable to attacks, and what steps we can use to better protect ourselves [5][6].

With the security of Chrome's extension coming under fire, several questions need to be answered. Initially, we need to know what kind of privileges Chrome provides to its extensions. If a malicious extension is installed, the privileges given by Chrome dictate how much damage can be done. There's also the possibility that Chrome could further limit the privilege of these extensions to better secure the privacy of its users. Knowing the privileges will also give us better insight as to what attacks could be executed with an extension alone. But security isn't all up to the browser [7]. That duty also falls to the websites that user's access. Therefore, websites can choose options to better secure their webpages, and decrease the chances that users with malicious extensions could be affected. Knowing which popular websites display low levels of security is important to increasing the safety of users. Finally, it's important to understand Google's policy regarding extensions, and discover what more can be done for the benefit of user security.

The objective of this project is to test a variety of different attacks using browser extensions on Google Chrome. By researching and implementing a variety of different attacks, we can find where Chrome is susceptible to allowing malicious extensions [8]. The scope would entail any type of extension that could be installed in Chrome, with malicious intent. These types of extensions can contain a variety of different attack vectors and target many different types of websites or machines. Understanding the weaknesses of Chrome is important to maintaining a safer browsing experience. As the world of technology is always changing, and staying up to date in this field is paramount to security. This research will help to inform browser users of the dangers that exist when using extensions, and how threat actors may be deceiving them to perform malicious activities on their computers. This research also shows the different vulnerabilities that exist within browsers, and demonstrates the privileges that browsers have on a computer [9]. We expect the research to show that browser vulnerabilities aren't a one size fits all type of attack. In order to successfully exploit a machine, a certain set of circumstances must first be met before an exploit is successful [10]. We also expect some websites to showcase lower levels of protection allowing for poor security against malicious extensions.

2 BACKGROUND

Chrome's extension architecture is based on component isolation and privilege separation. When it comes to Chrome extensions, it's a zipped bundle of files that include various formats like HTML, CSS, JavaScript, and many more [11]. When talking about extensions for Google Chrome, it has three types of components. One of them being content scripts that directly interact with web pages. The other being an extension core that interacts with the browser. Lastly, an optional native binary that interacts with the operating system [12]. The extension core becomes active when either the browser starts or after a user logs into their computer provided that the extension has background permission. With an extension, it can inject content scripts into web pages loaded by the browser and each page has its own instance of an extension's content scripts. Each content script runs in the same process as the web page into which it's injected. A content script is a part of the extension that runs in the context of a particular web page. Background scripts can access all the WebExtension JavaScript APIs, but they can't directly access the content of web pages [13]. So, if your extension needs to do that, you need content scripts. Just like the scripts loaded by normal web pages, content scripts can read and modify the content of their pages using the standard DOM APIs [14]. There is only one instance of the extension core per extension and the extension's native binary each run in a separate process. Throughout this research experiment, we will implement code for an extension and edit the variables using Firebase. Firebase is a product from Google that enables developers to build, manage, and grow their apps. No programming is required on the firebase side which makes it easy to use its features more efficiently. This real-time database enables users to sync-up application data in the cloud and make it available across all devices [15].

With a browser like Google Chrome, it provides more than 40 API's for chrome extensions [16]. API is known as an application programming interface and it enables companies to open up their applications' data and functionality to external third-party developers, business partners, and internal departments within their companies. This allows services and products to communicate with each other and leverage each other's data and functionality through a documented interface. Through these APIs, extension cores would be able to get real-time status of the browser [17]. An example of this can be seen as the list of tabs and running extensions or apps. We also would be able to access and modify user's data, update browser components, hijack or modify arbitrary web requests, and send messages to other extensions. Another concept to remember are iframes which would be applied later in our experiments. An iframe or inline frame is a HTML element that loads another HTML page within the document. It basically puts another webpage within the parent page and is usually used for ads, embedded videos, interactive content, and web analytics. When the web browser encounters an iframe element, it creates a new HTML document environment to load the content within. It takes the code from the referenced src or srcdoc and renders it as its own website that is then put entirely within the parent browsing page. It is called an inline frame because to the user it is all one web page. The child iframe is a complete browsing environment within the parent frame. It can load its own JavaScript and CSS separate from the parent. They can also be refreshed and loaded asynchronously from the parent site [18].

One of the major security features of Google Chrome's extension architecture is that the capabilities of components are limited based on their type and permissions granted to them. One of the high risks associated with extensions are content scripts and how they can be exploited by malicious websites because they directly interact with web pages [19][20]. Because of this reason, content scripts have the lowest privilege and can only use the APIs provided to web pages which are called browser APIs. Browser APIs include JSON, HTML5, and XMLHttpRequest APIs. When talking about content scripts, it is only accessible through the subset of Chrome APIs that support messaging between an extension and its content scripts (chrome.extension API). As previously mentioned the capabilities of the extension is limited by its permissions, so for Chrome APIs, browser APIs, and access to the web pages are guarded by these permissions [21]. The user is notified of these permissions by declaring them in its manifest file. When dealing with permissions, there are two main types which are API permissions and host permissions. Host permissions specify which pages an extension can inject content scripts and are basically a set of URLs. An example of this is when a password manager extension has host permission for one site, then it cannot access any other site. When it comes to API permissions, the extension core can only access it when it is guarded by permissions if it has the corresponding permissions in its manifest [22][23][24]. To add on, gaining access to certain Chrome extension APIs and browser APIs are constrained by host permissions. An example of this is if an extension does not have host permission to a website like <http://www.facebook.com> or an encompassing permission like `"*://*.**"`, then it can't make an XMLHttpRequest to <http://www.facebook.com> or even block a web request to www.facebook.com even if it has API permissions `webRequest` and `webRequestBlocking` [25].

3 METHODOLOGY

In regard to Google Chrome and the extensions that it uses, theft and forgery of user data can be a highly rated risk associated with the extensions. Users that use Google Chrome have sensitive data like usernames, passwords, social security numbers, and credit card numbers which are normally communicated through different web pages [26][27]. There are different attack vectors associated with stealing user data. As explained before, extensions that are granted permissions to access the pages that contain private and sensitive data can also easily steal the data. One of the flaws in Chrome extensions that bleed into different attack vectors is the abuse of the `'http://*/**'` host permission. The most common type of permission that extensions are given are the permissions to inject content scripts into the websites of Google Chrome. This permission enumerates the pages an extension is allowed to access [28][29]. In most cases, the extension's content scripts are allowed to run on any page that is browsed by the user. This permission is denoted with `'http://*/**'` and the injected content scripts can read any content on the page, and this includes sensitive data from user input, extensions like password managers, and the browser itself with its built-in auto form filler. An example of this is when a user visits a web page, we can create a malicious extension that would inject scripts into the specified page and since they are running in the page's environment, the scripts would have the ability to read from the DOM the password that the user enters. For this malicious extension to execute, it needs to be installed and active in the browser at the time the user accesses

the page. Additionally, finding the specific information to extract in the DOM of the target website is usually page specific [30][31]. Throughout this experiment, we will focus on three main attack vectors which are stealthy attacks using background tabs, stealthy attacks using iframes, and forging user data from specific extensions. Some of these attack vectors are difficult for users to detect and some require fewer permissions which may be difficult to attack as well.

3.1 STEALTHY ATTACKS USING BACKGROUND TABS

In relation to stealthy attacks using background tabs, there are at least two different methods in which to open or redirect a tab to target websites that do not require additional permissions beyond the “http://*/*” host permission. This means that the tab permission is not required. The first method in using background tabs is to redirect an inactive tab to the target web page and from there the extension can steal the sensitive data and later, redirect the tab to the original website. As shown in the figure 1, in detail, by calling ‘chrome.tabs.query’ which is the queryInfo where the queryInfo’s active flag is set to false, an extension can get the list of inactive tabs [32].



Figure 1: QueryInfo’s Active Flag Set to False

From there, the query can then be further restricted to tabs that are open in the background windows by setting the queryInfo’s currentWindow field to false. From there, the extension may be able to use ‘chrome.tabs.update’ to redirect the tab. A user may be able to do this because the tab API methods are not considered sensitive to Google Chrome, and this allows the extension to not claim the tab permission in its manifest. This is considered a stealthy attack because the only way of noticing the attack is when the tab icon redraws when a different page is loaded [33]. The other method in using the ‘chrome.tabs.query’ to find out whether or not a tab is visible when using the Windows API. The Windows API can determine which browser windows are currently focused on. For example, there could be a browser window open at the foreground or have the pointer hovering over the windows browser which would enable a malicious extension to launch attacks only when the user is using an application other than the browser. This is sort of a workaround in not requiring the extension to have any permissions. The implementation of stealthy attacks in stealing information can be seen later when it comes to forging user data.

3.2 STEALTHY ATTACKS USING I-FRAMES

Another stealthy attack vector when dealing with Chrome extensions is to use iframes. This is done by loading extensions into iframes which could be placed in background tabs or to make them unnoticeable or hidden to users by making the iframes fully transparent or displaying them at a very small size. Stealthily attacking web pages with iframes can be done in two methods [34][35]. The first method deals with modifying the DOM and the autofill feature. For example, when an extension's content script is running on a page, the extension can modify the DOM of that page to create a new iframe by executing `document.write("<iframe src='http://target.com'> </iframe>");`. After executing, the page (target.com) is then loaded in the iframe and the autofill feature or a password manager extension would automatically fill in the credentials or content needed for 'target.com'. In order to read the content inside the iframe, an extension needs to have host permission to the iframed page as well as the 'all_frames' option specified in its manifest. With the addition of the 'all_frames' option, it causes no warning to be shown to the user on installation.

The other method that deals with iframes takes an even stealthier approach [36]. In this example, by having host permission to the specific page loaded in the tab, an extension has a content script running in the same tab that doesn't contain the target page. With this, the extension can then create an iframe and load the target page similar to the previous method. This then allows Chrome to use its autofill functionality or a password manager's ability to fill content for the target page. After this, the extension would be able to take a screenshot of the page that is running and it would include the auto filled content. This method is used to steal information in plain text like credit card numbers, usernames, date of births, etc. In order to make the iframe stealthier than the previous attack, we could make the iframe meticulously transparent. Another option instead of making the iframe transparent is to change the size of the iframe in making it really small, so that the user would not be able to notice it. We can make the iframe show a single character at a time and move the field of view character by character until the data has been captured by using `frame.contentWindow.scrollTo(xcoord,ycoord)`. As shown below in figure 2 and figure 3, we were able to hide an iframe on a target user's reddit page with the help of the malicious extension and it provided us with the login credentials which would help us find more information like addresses.



Figure 2: Reddit Page with Transparent iframe on the Bottom



Figure 3: Reddit Page with Visible iframe on the Bottom

Occasionally, sensitive information like login credentials are only available after the user has logged into the target page. In order to get sensitive information, the extension has to mount the attack after the user has logged in but before the user’s session expires. One of the ways that a malicious extension can detect the user has recently logged in is in the case where extensions have host permission to the designated page because it can observe the loading of the login page [37][38]. To add on, other permissions given to the extension like viewing the history of the browser and web Requests also lets the extension know that the user had recently logged in. Some of the limitations for these attack methods to work is that the data has to be in plain text. Furthermore, the websites that would be able to implement in an iframe does not include some of the top websites like Facebook, Twitter, and Amazon. Another limitation in these two methods dealing with iframes is that the user must enable or click on the malicious extension’s icon in Google Chrome in order for the malicious extension to run or execute. Moreover, the user must already be on the website that the malicious extension plans to extract information.

3.3 FORGING USER DATA

The last attack vector in dealing with Chrome extensions relates to forging user data or web requests from the extensions itself so that it would appear it came from the user [39][40]. From a malicious extension, it can gather information on the user. It could even log into the targeted website and access the information like addresses or transaction history provided that the user is logged in as well. The amount of information or data that can be extracted also depends on the implementation of the website’s login process itself. In most cases, it would be difficult if the website has two factor authentication. Figure 4 shows an example of how an extension can take the autofill information and translate it for us.

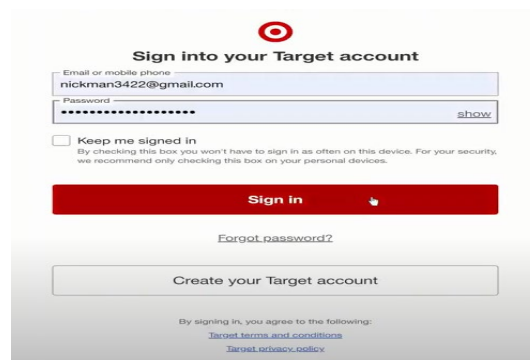


Figure 4: Target Login Page

```
INFO: root:POST request,
Path: /
Headers:
Host: 18.208.130.122:8080
Connection: keep-alive
Content-Length: 103
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36
Content-Type: application/json
Accept: */*
Origin: chrome-extension://koigcpflaploaajelalfepmlfaggbmgg
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

Body:
Personal Information:

target:
username: nickman3422@gmail.com
password: ThisisAPassword1234

undefined

129.21.125.105 - - [19/Nov/2021 19:47:32] "POST / HTTP/1.1" 200 -
```

Figure 5: Console View of Captured Credentials

As seen in the figure 5 above, the malicious extension was able to log in to “Target.com” with the identity of the user whose username and password have been auto-filled by the browser or a password-manager extension. With this malicious extension, more damage can be done than just taking the username and password. One of them includes data integrity attacks when changing passwords since we can modify the extension to log in with an incorrect password forcing the user to be locked out. To add on, we can further the damage by making the user reset the password since the old password wouldn’t work and in turn, it would create an opportunity to get the answers for the password reset questions which could be then used for other websites of the user. Lastly, we would also be able to change banking operations such as transferring money with the login credentials. Another implementation of forging user data is through page captures where it would show us captured information of all the sites the user would visit. This is considered a big threat because whatever website the user visits, the attacker would be able to gain information and use it for malicious purposes. Figure 6 illustrates an example of the implemented code and how the extension would capture the webpage and provide the attacker the information.

```
chrome.tabs.onActivated.addListener(function(activeInfo) {
  chrome.tabs.captureVisibleTab(null, {}, function (img) {
    var http = new XMLHttpRequest();
    var url = "https://evil-123.firebaseio.com/capture.json";
    http.open("POST", url, true);
    http.send(JSON.stringify({"capture": img}));
  });
});
```

Figure 6: Extension Code for Screen Capturing

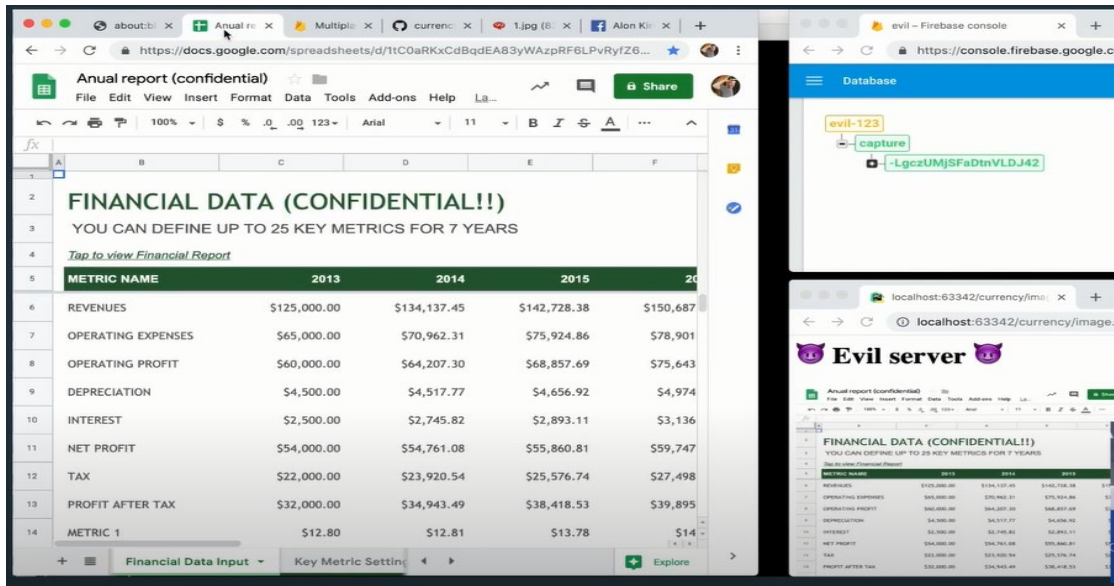


Figure 7: Application of Screen Capture with Malicious Extension

According to the figure 7 above, the page on the right shows the target’s page and the websites that we would want to capture, whereas the page on the top right shows the Firebase console and the page on the bottom right shows a small gallery that is built on top of the Firebase console. With this, we can see that whatever website the target user visits, we would be able to capture the information and use it for malicious purposes as mentioned before.

4 RESULTS

From the attacks that we tested, there was a variety of success across the board. This was expected however, as we originally hypothesized that attacks would vary in their success rate and method. For the results, we’ll go through each of the three attacks that we previously mentioned to see how successful each one was.

As far as the background tab attack, we expected this to be a go-to attack because of how straight forward it is. The development is simple as extensions using this attack have been around for a long time and aren’t anything groundbreaking. Because of this, many of the extensions we wanted to test had already been banned due to being reported for this type of malicious activity. However, several extensions still exist out there that contain one of the attacks we plan to discuss in this project. Regarding our first kind of attack, the attack is pulled off by first checking a series of requirements for performing the attack. The attack takes place by finding tabs that Chrome has running in the background, and then gathering data like login information from those tabs without the user knowing. The malicious extension must first determine if Chrome has the “queryInfo active” flag. If the flag returns false, then the attack can continue. If the flag returns true, then the background tabs won’t update, showing the information that the attacker is looking for.

```
pref_cookie":true,"kevlar_clear_non_displayable_url_params":true,"kevlar_client_save_subs_pre  
f" autocorrect="off" hidden name="search_query" tabindex="0" type="text" spellcheck="false">
```

Figure 8: Malicious Extension with “QueryInfo Active” Flag

As shown on the figure 8, just by looking through the source information on a given website, we can see critical information about the options for that site. For most of the testing, we used popular websites, in this case Youtube.com. By checking the query flags, we can see that search queries on this website are hidden. This helps to hide information like search results from other tools like extensions. In this case, an extension without having a direct view of the window, wouldn't be able to collect search queries on this site.

The downside of attacks using background tabs is the stealth element. The extension must first look to see if the tab is in the background, and then choose to start updating the tab using requests. This method is based on assuming the user can't see the tab since it's being covered by something else on the screen. However, obviously this isn't always the case, and stealth is a major downside for this kind of attack since in some cases it's easy to get caught. Because of that, we regard this attack as one of the most popular because of its simplicity and widespread use, however far from the most effective.

The next attack we investigated, was a far stealthier attack using iframes. The major advantage here is the ability to access information in a similar fashion with far less chances of the user noticing. This is done using iframes, which resemble small windows that can be opened separate from the original Chrome tab. The advantage here is the flexibility that these windows have. Just like a regular tab, these frames can request web pages and display data. However, these iframes can be altered to be extremely small, so that the user doesn't see them, or even transparent. Because of this increased functionality, this kind of attack can be much more effective than utilizing background tabs. The methods of how this attack works remain relatively the same. Within this iframe, the attacker can use the extension to take screenshots and record information shown in these frames. Strangely enough, the success rate for this attack was the same for our previous type of attack. In fact, these use similar attack vectors making the attacks almost interchangeable. This would explain why the simple background tab attack is so rare now, since it's been replaced by something with higher stealth, and the same success rate.

Looking at our final attack utilizing forged user information, we saw different results than expected. This type of attack has a guaranteed success rate if completed correctly, but it must be completed under specific circumstances. For this type of attack, the attacker simply uses stolen data to access restricted websites using login credentials. The extension is only a piece of spyware designed to look at what the user is doing, and any chrome extension you download would have this possible functionality. After seeing the user login, the attack would record the inputs they saw, and mimic those inputs later when they wanted to gain access. After digging through website HTML, and doing research on internet security, we were unable to find any flag options that would prevent this. The downside is the length of time an attack might have to wait before the user decides to login to something personal like social media or a bank account. Finding an existing chrome extension that performs this kind of attack was simple. There is plenty of documentation online giving links to malicious extensions for research purposes. Below was one that we found almost instantly.

VK Tik-Tok Instagram video downloader [Add to Chrome](#)

poqdev.com

★★★★★ 10 | Productivity | 2,000+ users

Overview Privacy practices Reviews Support Related

Report

tandarts Procreate

Charli damelio te #procreate #fory

original sound - Cleopatra

19.5K 548 Share to

https://www.tiktok.com/@tandarts..procreate/video/691... COPY LINK

Extensions

Full access
These extensions can see and change information on this site.

VK Tik-Tok Instagram video d... Pin extension

Manage Extensions

Overview

Compatible with your device

VK Tik-Tok Instagram video downloader also vimeo, facebook, yandex music, dailymotion, deezer, spotify

Very easy extension that allows you to:

- * Download video from VK, Tik-Tok, Instagram, vimeo, facebook
- * Download music from yandex music, dailymotion, deezer, spotify

Additional Information

[Website](#) [Report abuse](#)

Version 1.0.0


Updated February 1, 2021

Size 288KIB

Figure 9: Chrome Extension with VK Tik-Tok Instagram Downloader

Overview **Privacy practices** Reviews Support Related

Privacy practices



The publisher has disclosed that it will not collect or use your data


To learn more, see the publisher's [privacy policy](#).

This publisher declares that your data is:

- Not being sold to third parties, outside of the [approved use cases](#)
- Not being used or transferred for purposes that are unrelated to the item's core functionality
- Not being used or transferred to determine creditworthiness or for lending purposes

Figure 10: Chrome Privacy Practices

As you can see in the figure 9, the extension describes itself as a tool that can download videos from Tik-Toc and Instagram. And upon examining the privacy practices shown in the figure 10, we can see that it claims not to sell or collect any personal data for malicious purposes. However, looking at every other Chrome extension available on the Google store, we found that every single one has this exact same private policy page. This suggests that in order to even publish an extension, the contents of this page must be true. However, extensions are added to Chrome all the time, and Google doesn't have the resources to filter through each one, so developers can simply lie and claim their product to be harmless. Upon looking into the reviews, we found following as shown in the figure 11.



if i could give this zero stars i would. immediately got my instagram that ive had for over seven years banned, cant even get a phone number confirmation code. DO NOT DOWNLOAD THIS APP

Was this review helpful? Yes No [Reply](#) | [Mark as spam or abuse](#)

Figure 11: User Review

It's clear that this was a malicious extension, intended to steal password data for websites. The comment warns other users not to download this, as their accounts could get banned. These extensions exist all over the Google store, and when installing extensions, it's important to understand what you're downloading, before you download it.

5 CONCLUSION

Google Chrome’s browser extensions can be either useful or malicious depending on the extension downloaded and applied. We have seen so far the capability of malicious Chrome extensions and the extent to what is available in terms of attack vectors in Google Chrome. We had tested out different ways in which a malicious extension can steal users’ sensitive data, track their behavior, and forge their input. Even though these malicious extensions are not as popular as other extensions and would not be easily installed by a user willingly, many published extensions still have sufficient privileges to carry out these attacks. A benefit to our approach is that we were able to extract more information than expected from targeted websites. Furthermore, we were able to showcase different attack vectors with simple code to steal user information. Lastly, we were also able to steal this information discreetly without the awareness of the targeted user. The downside to our approach is that we were testing the implementations of these malicious extensions by installing them ourselves and executing them. If it were to apply to practical situations, then we would have to figure out a method in getting the user to install the extension willingly by themselves or we would have to install it on their Chrome browser. To add on, the implications of the extension would be stated on the extension’s details and if the user were to read it and have an understanding, then they would be neglected to install the extension.

When installing an extension on Google Chrome or any browser, it is always suggested or required to read the extension details and see the permissions that are allowed for the extension. Moreover, Chrome also verifies and declares what privacy practices the extension would follow, and it would be necessary to read them if one wanted their browsing secure with the installed extensions. Figure 12 shows an example of the privacy practices for the extension Google Translate.

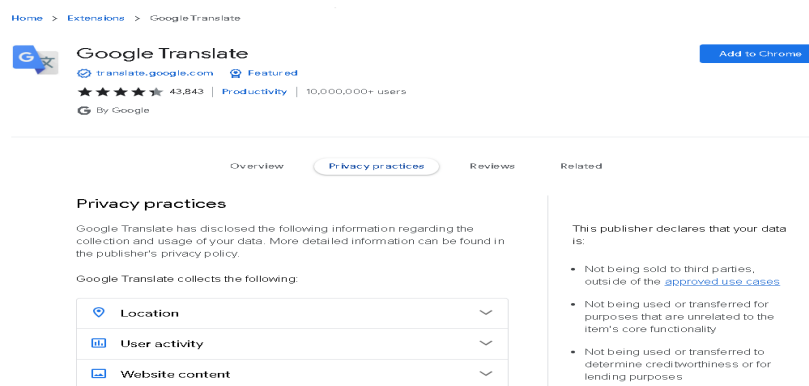


Figure 12: Privacy Practices of Google Translate Extension

Another method in ensuring that no malicious extension is installed is to enable Google Chrome’s Enhanced Safe Browsing feature which is easily enabled by going to the browser’s security settings, where one would discover three different degrees of protection being enhanced protection, standard protection, and no protection.

6 FUTURE WORKS

In the future we would attempt to see if the same attack vectors can be used across different web browsers like Mozilla Firefox and Safari. Through research, we found that different browsers have different methods in implementing their extensions. With this, we would like to test out how the attack vectors would work for these specified browsers. We would also delve into more attack vectors for Google Chrome itself. There are several different attacks that we could still attempt in order to steal user information. Some of them include implementing a key-logger or cross-site scripting. In addition to discovering more attack vectors, we would also like to figure out a method in making the user download or install the malicious extension willingly and without full awareness of its capabilities even after reading the details of the extension.

References

- [1] Fass, Aurore, Dolière Francis Somé, Michael Backes, and Ben Stock. "Doublex: Statically detecting vulnerable data flows in browser extensions at scale." In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 1789-1804. 2021.
- [2] Dissanayaka, Akalanka Mailewa, Susan Mengel, Lisa Gittner, and Hafiz Khan. "Security assurance of MongoDB in singularity LXC: an elastic and convenient testbed using Linux containers to explore vulnerabilities." Cluster Computing 23 (2020): 1955-1971.
- [3] Picazo-Sanchez, Pablo, Lara Ortiz-Martin, Gerardo Schneider, and Andrei Sabelfeld. "Are chrome extensions compliant with the spirit of least privilege?." International Journal of Information Security 21, no. 6 (2022): 1283-1297.
- [4] Pantelaios, Nikolaos, Nick Nikiforakis, and Alexandros Kapravelos. "You've changed: Detecting malicious browser extensions through their update deltas." In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 477-491. 2020.
- [5] Dissanayaka, Akalanka Mailewa, Susan Mengel, Lisa Gittner, and Hafiz Khan. "Vulnerability prioritization, root cause analysis, and mitigation of secure data analytic framework implemented with mongodb on singularity linux containers." In Proceedings of the 2020 the 4th International Conference on Compute and Data Analysis, pp. 58-66. 2020.
- [6] Zhang, Mingming, Xiaofeng Zheng, Kaiwen Shen, Ziqiao Kong, Chaoyi Lu, Yu Wang, Haixin Duan, Shuang Hao, Baojun Liu, and Min Yang. "Talking with familiar strangers: An empirical study on https context confusion attacks." In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 1939-1952. 2020.
- [7] Kariryaa, Ankit, Gian-Luca Savino, Carolin Stellmacher, and Johannes Schöning. "Understanding users' knowledge about the privacy and security of browser extensions." USENIX, 2021.
- [8] Picazo-Sanchez, Pablo, Lara Ortiz-Martin, Gerardo Schneider, and Andrei Sabelfeld. "Are chrome extensions compliant with the spirit of least privilege?." International Journal of Information Security 21, no. 6 (2022): 1283-1297.
- [9] Dissanayaka, Akalanka Mailewa, Susan Mengel, Lisa Gittner, and Hafiz Khan. "Dynamic & portable vulnerability assessment testbed with Linux containers to ensure the security of MongoDB in Singularity LXCs." In Companion Conference of the Supercomputing-2018 (SC18). 2018.

- [10] Sapkota, Bhumika, and Akalanka B. Mailewa. "A Scalable Framework to Detect, Analyze, and Prevent Security Vulnerabilities in Enterprise Software-Defined Networks." Journal homepage: www.ijrpr.com ISSN 2582: 7421. (DOI:10.55248/gengpi.2022.3.2.1)
- [11] Agarwal, Shubham, and Ben Stock. "First, Do No Harm: Studying the manipulation of security headers in browser extensions." In Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb). <https://doi.org/10.14722/madweb>. 2021.
- [12] Mailewa, Akalanka, and Jayantha Herath. "Operating Systems Learning Environment with VMware" In The Midwest Instruction and Computing Symposium. Retrieved from http://www.micsymposium.org/mics2014/ProceedingsMICS_2014/mics2014_submission_14.pdf. 2014.
- [13] Hiremath, Panchakshari N., Jack Armentrout, Son Vu, Tu N. Nguyen, Quang Tran Minh, and Phu H. Phung. "MyWebGuard: toward a user-oriented tool for security and privacy protection on the web." In Future Data and Security Engineering: 6th International Conference, FDSE 2019, Nha Trang City, Vietnam, November 27–29, 2019, Proceedings 6, pp. 506-525. Springer International Publishing, 2019.
- [14] Iqbal, Junaid, Ratinder Kaur, and Natalia Stakhanova. "PoliDOM: Mitigation of DOM-XSS by detection and prevention of unauthorized DOM tampering." In Proceedings of the 14th International Conference on Availability, Reliability and Security, pp. 1-10. 2019.
- [15] Shetty, Roshan Ramprasad, Akalanka Mailewa Dissanayaka, Susan Mengel, Lisa Gittner, Ravi Vadapalli, and Hafiz Khan. "Secure NoSQL based medical data processing and retrieval: the exposome project." In Companion Proceedings of the 10th International Conference on Utility and Cloud Computing, pp. 99-105. 2017.
- [16] Xie, Mengfei, Jianming Fu, Jia He, Chenke Luo, and Guojun Peng. "JTaint: finding privacy-leakage in chrome extensions." In Information Security and Privacy: 25th Australasian Conference, ACISP 2020, Perth, WA, Australia, November 30–December 2, 2020, Proceedings 25, pp. 563-583. Springer International Publishing, 2020.
- [17] Solomos, Konstantinos, Panagiotis Ilia, Nick Nikiforakis, and Jason Polakis. "Escaping the Confines of Time: Continuous Browser Extension Fingerprinting Through Ephemeral Modifications." In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 2675-2688. 2022.
- [18] Bian, Yan, Dechao Ma, Qing Zou, and Weirui Yue. "A Multi-way Access Portal Website Construction Scheme." In 2022 5th International Conference on Artificial Intelligence and Big Data (ICAIBD), pp. 589-592. IEEE, 2022.
- [19] Jairu, Pankaj, and Akalanka B. Mailewa. "Network Anomaly Uncovering on CICIDS-2017 Dataset: A Supervised Artificial Intelligence Approach." In 2022 IEEE International Conference on Electro Information Technology (eIT), pp. 606-615. IEEE, May 2022. (DOI:10.1109/eIT53891.2022.9814045)
- [20] Fass, Aureole, Dolière Francis Somé, Michael Backes, and Ben Stock. "Doublex: Statically detecting vulnerable data flows in browser extensions at scale." In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 1789-1804. 2021.
- [21] Wang, Xinyu, Yuefeng Du, Cong Wang, Qian Wang, and Liming Fang. "Webenclave: protect web secrets from browser extensions with software enclave." IEEE Transactions on Dependable and Secure Computing 19, no. 5 (2021): 3055-3070.
- [22] Kaja, Durga Venkata Sowmya, Yasmin Fatima, and Akalanka B. Mailewa. "Data integrity attacks in cloud computing: A review of identifying and protecting techniques." Journal homepage: www.ijrpr.com ISSN 2582 (2022): 7421. (DOI:10.55248/gengpi.2022.3.2.8)

- [23] Mailewa, Akalanka, and Kyle Rozendaal. "A Novel Method for Moving Laterally and Discovering Malicious Lateral Movements in Windows Operating Systems: A Case Study." *Advances in Technology* (2022): 291-321, ISSN 2773-7098. (DOI:10.31357/ait.v2i3.5584)
- [24] Diamantaris, Michalis, Elias P. Papadopoulos, Evangelos P. Markatos, Sotiris Ioannidis, and Jason Polakis. "Reaper: real-time app analysis for augmenting the android permission system." In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, pp. 37-48. 2019.
- [25] Razali, Muhammad Amirrudin, and Shafiza Mohd Shariff. "Cmblock: In-browser detection and prevention cryptojacking tool using blacklist and behavior-based detection method." In *Advances in Visual Informatics: 6th International Visual Informatics Conference, IVIC 2019, Bangi, Malaysia, November 19–21, 2019, Proceedings 6*, pp. 404-414. Springer International Publishing, 2019.
- [26] Hajli, Nick, Farid Shirazi, Mina Tajvidi, and Nurul Huda. "Towards an understanding of privacy management architecture in big data: an experimental research." *British Journal of Management* 32, no. 2 (2021): 548-565.
- [27] Mailewa Dissanayaka, Akalanka, Roshan Ramprasad Shetty, Samip Kothari, Susan Mengel, Lisa Gittner, and Ravi Vadapalli. "A review of MongoDB and singularity container security in regards to hipaa regulations." In *Companion Proceedings of the 10th International Conference on Utility and Cloud Computing*, pp. 91-97. 2017.
- [28] Sjösten, Alexander, Steven Van Acker, Pablo Picazo-Sanchez, and Andrei Sabelfeld. "Latex Gloves: Protecting Browser Extensions from Probing and Revelation Attacks." In *NDSS*. 2019.
- [29] Khan, Muhammad Maaz Ali, Enow Nkongho Ehabe, and Akalanka B. Mailewa. "Discovering the Need for Information Assurance to Assure the End Users: Methodologies and Best Practices." In *2022 IEEE International Conference on Electro Information Technology (eIT)*, pp. 131-138. IEEE, May 2022. (DOI:10.1109/eIT53891.2022.9813791)
- [30] Ghosal, Sandip, and R. K. Shyamasundar. "Preventing Privacy-Violating Information Flows in JavaScript Applications Using Dynamic Labelling." In *Information Systems Security: 18th International Conference, ICISS 2022, Tirupati, India, December 16–20, 2022, Proceedings*, pp. 202-219. Cham: Springer Nature Switzerland, 2022.
- [31] Khan, Saad, and Akalanka B. Mailewa. "Discover Botnets in IoT Sensor Networks: A Lightweight Deep Learning Framework with Hybrid Self-Organizing Maps." *Microprocessors and Microsystems* (2023): 104753. (DOI: <https://doi.org/10.1016/j.micpro.2022.104753>)
- [32] Gao, Yun, Kai Luo, Chongrong Fang, and Jianping He. "Fragility-Aware Stealthy Attack Strategy for Multi-Robot Systems against Multi-Hop Wireless Networks." In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pp. 4827-4832. IEEE, 2022.
- [33] Luo, Yukui, Cheng Gongye, Shaolei Ren, Yunsi Fei, and Xiaolin Xu. "Stealthy-shutdown: Practical remote power attacks in multi-tenant fpgas." In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pp. 545-552. IEEE, 2020.
- [34] Mailewa, Akalanka, Susan Mengel, Lisa Gittner, and Hafiz Khan. "Mechanisms and techniques to enhance the security of big data analytic framework with mongodb and Linux containers." *Array* 15 (2022): 100236. (DOI:10.1016/j.array.2022.100236)
- [35] Chinpruthiwong, Phakpoom, Jianwei Huang, and Guofei Gu. "{SWAPP}: A New Programmable Playground for Web Application Security." In *31st USENIX Security Symposium (USENIX Security 22)*, pp. 2029-2046. 2022.

- [36] Tramèr, Florian, Pascal Dupré, Gili Rusak, Giancarlo Pellegrino, and Dan Boneh. "Adversarial: Perceptual ad blocking meets adversarial machine learning." In Proceedings of the 2019 ACM SIGSAC conference on computer and communications security, pp. 2005-2021. 2019.
- [37] Gannis, Steven, Matthew VanderLinden, and Akalanka Mailewa. "Analyzing Data Encryption Efficiencies for Secure Cloud Storages: A Case Study of Pcloud vs OneDrive vs Dropbox." *Advances in Technology* (2022): 79-98. (DOI:10.31357/ait.v2i1.5526)
- [38] Lin, Xu, Panagiotis Ilia, and Jason Polakis. "Fill in the blanks: Empirical analysis of the privacy threats of browser form autofill." In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 507-519. 2020.
- [39] Olaosebikan, Ayodeji, Thivanka PBM Dissanayaka, and Akalanka B. Mailewa. "Security & Privacy Comparison of NextCloud vs Dropbox: A Survey." In Midwest Instruction and Computing Symposium (MICS). 2022.
- [40] Calzavara, Stefano, Alvisè Rabitti, Alessio Ragazzo, and Michele Bugliesi. "Testing for integrity flaws in web sessions." In *Computer Security—ESORICS 2019: 24th European Symposium on Research in Computer Security*, Luxembourg, September 23–27, 2019, Proceedings, Part II 24, pp. 606-624. Springer International Publishing, 2019.