The Risks of WebGL: Analysis, Evaluation and Detection

Chapter · September 2019

DOI: 10.1007/978-3-030-29962-0_26

Israel Cidon
VMware

THE RISKS OF WEBGL: ANALYSIS, EVALUATION AND DETECTION

A PREPRINT

Alex Belkin
Department of Electrical Engineering
Technion University
Haifa, Israel
belkinalex@gmail.com

Nethanel Gelernter
Department of Computer Science
College of Management Academic Studies
Rishon LeZion, Israel
nethanel.gelernter@gmail.com

Israel Cidon
Department of Electrical Engineering
Technion University
Haifa, Israel
cidon@ee.technion.ac.il

May 1, 2019

ABSTRACT

WebGL is a browser feature that enables JavaScript-based control of the graphics processing unit (GPU) to render interactive 3D and 2D graphics, without the use of plug-ins. Exploiting WebGL for attacks will affect billions of users since browsers serve as the main interaction mechanism with the world wide web. This paper explores the potential threats derived from the recent move by browsers from WebGL 1.0 to the more powerful WebGL 2.0. We focus on two possible abuses of this feature: distributed password cracking and distributed cryptocurrency mining. Our evaluation of the attacks also includes the practical aspects of successful attacks, such as stealthiness and user-experience. Considering the danger of WebGL abuse, as observed in the experiments, we designed and evaluated a proactive defense. We implemented a Chrome extension that proved itself effective in detecting and blocking WebGL. We demonstrate in our experiments the major improvements of WebGL 2.0 over WebGL 1.0 both in performance and in convenience. Furthermore, our results show that it is possible to use WebGL 2.0 in distributed attacks under real-world conditions. Although WebGL 2.0 shows similar hash rates as CPU-based techniques, WebGL 2.0 proved to be significantly harder to detect and has a lesser effect on user experience.

Keywords WebGL · security · distributed-attack · crypto-mining · password-cracking · web-browser

1 Introduction

The rapid evolution of web technologies has delivered new possibilities to billions of users, while at the same time exposing them to new security threats. Browsers are an excellent example of this phenomenon. Their new features are being abused by malicious hackers to create attack vectors and efficiently launch attacks not previously considered a risk.

Some recent examples are the Cache API [1] and the ServiceWorker [2] features, which allow the launch of sophisticated timing side-channel attacks [3, 4]. Another example is the Quota API, which can be used to extract the exact size of cross-site requests [5].

Browsers serve users as the main interaction mechanism with the world wide web. Therefore, security vulnerabilities or browser features that can be exploited for attacks will affect billions of users and close to two billion websites that are being accessed. Previous works by both Van Goethem, Gerlernter [3, 4, 5], and many others [6, 7] offer examples of methods that exploit browser features to attack users. Other browser features, such as web-workers or web-sockets, have been used to effectively launch distributed denial-of-service attacks on websites [8, 9, 10].

This paper explores the risks posed to web users by WebGL 2.0 [11]. Web Graphics Library (WebGL) is a JavaScript API that uses the graphics processing unit (GPU) to render interactive 3D and 2D graphics within any compatible web browser, without the use of plug-ins [12]. WebGL allows users to communicate directly with the graphics hardware. It comes with its own programming language called GLSL. GLSL allows anyone to control the computational power of the GPU, using it as they wish.

Initially, GPUs were designed to accelerate the creation of images intended for output to a display device. However, their highly parallel structure makes them more efficient than general-purpose CPUs for algorithms that process large blocks of data in parallel. This made GPUs rise to prominence in the fields of crypto-mining, deep-learning, and more.

Controlling the GPU via the browser has introduced several new opportunities for attackers. For example, GPUs are used to efficiently break hashes [13] or for Bitcoin mining [14]. The ability to abuse the previous version of this API has already been examined by researchers. Fortunately, WebGL 1.0 is quite limited and does not support 32-bit integers or bitwise operators [15]. This makes it difficult to implement algorithms that efficiently calculate MD5 hashes, and much harder to implement more complex hashing algorithms such as SHA-2. Marc Blanchou's presentation at Black Hat Europe [16] showed that these limitations make the abuse of WebGL 1.0 ineffective, even compared to more naive implementations in JavaScript that use the CPU.

WebGL 2.0 was recently integrated into the popular browsers Google Chrome and Mozilla Firefox. WebGL's new features (e.g., support for 32-bit integers) establish the need to reevaluate the risks posed by this API. Due to the expected danger, it was also essential to develop countermeasures that can detect malicious WebGL 2.0 code and block it. This is a challenge that has not been studied before, even for WebGL 1.0.

Our initial research focused on distributed password cracking, where users' browsers are exploited to crack hashes while the user is surfing web pages controlled by the attacker. In Section 4.2, we describe an experiment that enabled us to compare the effectiveness of password cracking using WebGL 1.0, WebGL 2.0, and CPU techniques. Our results show that although WebGL 1.0 demonstrates a slower hash rate than the CPU hasher, WebGL 2.0 shows significant improvement. Its hash rate is nearly two times faster than the CPU hasher when using best trade-off values as shown in Table 1.

A few months into the research, the crypto-currency market began its rapid growth, and our research on the abuse of browsers for hash cracking became a reality. Coinhive [17] and other similar companies made it easy for every website owner to mine crypto-currency such as Monero [18], on the browsers of people who visit their page. As a result, the browsers of hundreds of millions of web users were abused to mine crypto-currency. In most cases, it was done without the permission or the knowledge of the users [19]. This phenomenon encouraged researchers [20, 21] to perform an in-depth investigation of the landscape and impact of in-browser crypto-currency mining.

Both Hong et al. [20] and Konoth et al. [21] show how prevalent and potentially profitable crypto-currency mining can be for attackers. They explored the distribution of the infected websites containing mining code and demonstrate that no type of website is safe. Both works emphasize the inadequacy of current defense mechanisms, which are based on blacklists, and each suggested their own innovative countermeasure. Konoth et al. [21] managed to identify as many as 20 different active crypto-currency mining campaigns in 0.18% of Alex's Top 1 Million websites.

Other researchers implemented a framework to allow persistent and stealthy bot operation through web browsers without the need to install any software on the client side called MarioNet [22]. The effectiveness of MarioNet is demonstrated by designing a large set of successful attack scenarios where the user's system resources are abused to perform malicious actions including DDoS attacks to remote targets, cryptojacking, malicious/illegal data hosting, and darknet deployment.

At this point, we decided to add a new direction to our research and examine the consequences of WebGL 2.0 abuse for distributed crypto-mining.

1.1 Contributions

The main contributions of this paper are the analysis and evaluation of the risks posed by WebGL 2.0 under real-world conditions, and a prevention method to WebGL 2.0 attacks. To the best of our knowledge this is the first work to analyze the risks associated with WebGL 2.0. We addressed both the computational and user experience aspects of such potential exploits. Specifically, our research studies the following questions:

1. Can WebGL 2.0 be used to launch practical attacks?

2. How effective is WebGL 2.0 for distributed password cracking and crypto-currency mining compared to WebGL 1.0 and CPU-based techniques?
3. What can be done to detect and block WebGL 2.0 attacks?

Evaluating WebGL 2.0 only in terms of the theoretical attacker scenario benefit is not enough. It is also crucial to include different aspects that affect the effectiveness of distributed attacks under real-world conditions, such as user experience and stealthiness. Even if we manage to show a high performance attack under lab conditions, it isn't worth much to the attacker if the user can sense or even detect our attack. We implemented a distributed attack, which was used in several experiments on numerous users, to test all the necessary aspects of a distributed attack. Our results show that when it comes to performance, WebGL 2.0 and Coinhive show approximately the same hash rate as shown in Figure 2. However, for distributed attack aspects, our results in Section 6 demonstrate that WebGL 2.0 is much harder to detect and has a lesser effect on user experience compared to CPU miner. Shown in Figure 3 and the results of Experiment 4, this proves that WebGL 2.0 miner is more suited to cryptocurrency mining distributed attacks than CPU miner.

We further implemented and tested a means to detect such an attack. In Section 7, we implemented a Chrome extension to serve as a means for detecting and blocking the use of WebGL. We performed an experiment with numerous participants over an extended period of time to test the extension's efficiency and collect statistics about the use of WebGL in websites. Our results show that our extension is efficient in preventing WebGL abuse and that WebGL is relatively rare in websites.

Our findings are important and relevant to the Web community, mainly because the attacks studied in this paper have already been launched in different ways in the wild.

1.2 Paper Organization

Section 2 offers relevant background material about hash cracking, browser-based attacks, and the difference between GPU and CPU implementations. Section 3 discusses related work. Sections 4 and 5 analyze the abuse of WebGL 2.0 for password cracking and cryptocurrency mining, correspondingly. In both sections, the analysis is done from the perspective of a single victim and compares similar implementations using CPU and WebGL 1.0. Section 6 evaluates the abuse of WebGL 2.0 in a distributed attack under real-world conditions. Section 7 suggests and evaluates a means of defense, and Section 8 concludes.

2 Background and Motivation

The following section briefly explains concepts that will help the reader acquire a deeper understanding of the issues addressed in this paper. The section reviews hash cracking, browser-based distributed attacks, and the key differences between hash cracking implementation in WebGL and CPU.

2.1 Hash Cracking

This work addresses the challenge of hash cracking in two different scopes: password cracking and cryptocurrency mining.

Password cracking is the process of recovering passwords from the exposed output of a one-way cryptographic hash function, performed on the password. Password cracking can be done for several reasons, but the usual malicious reason is to gain unauthorized access without owner authorization or awareness.

There are dozens of password cracking programs on the market, each with its own special procedure[23]. However, all usually do one or a combination of the following password searches:

1. Create variations from a dictionary of known common passwords
2. Using brute force attack by trying all possible strings

In cryptocurrency mining, hash cracking is needed to ensure the authenticity of the information and to update the blockchain with the transaction [24]. Each time a cryptocurrency transaction is made, the cryptocurrency miners must solve complicated mathematical problems using cryptographic hash functions; these functions are associated with a block containing the transaction data. The mining process validates the calculated hashes on incremental values, called nonce, which are added to the given block data. The hash output must match a certain criterion: it needs to be less than the cryptocurrency's current target value. The current target value is also represented by the cryptocurrency's difficulty. Cryptocurrency difficulty is a measure of how long would it take at a given hash rate to find a block that matches the current target. Higher difficulty means a lower target value. The difficulty increases over time and varies between cryptocurrencies. Mining also involves competing with other cryptocurrency miners. Only the first one to crack the hash is rewarded with small amounts of cryptocurrency.

Advances in hardware and dedicated hash-cracking software have made hash cracking more practical and accessible than it used to be. For instance, a password's hash that would take over three years to crack in the year 2000 took just over two months to crack by the year 2016 [25].

One example of these hash cracking tools is an advanced password recovery tool created by Team Hashcat. Called Hashcat [26], this technology has won a succession of recent "Crack Me If You Can" contests [27] and is described [26, 28] as the world's fastest password cracker. Hashcat enables any user to crack a significant number of different kinds of hash algorithms with ease and speed on multiple platforms, and introduces a variety of advanced features. This led to designated hash cracking rigs that managed to break 300 GH/s on MD5 algorithm, 3493 KH/s on Scrypt algorithm, and more [29]. Furthermore, hash cracking has been tested on cloud GPU systems (e.g., Amazon EC2), which are more accessible to users than dedicated rigs. These tests managed to reach a rate of 2494 MH/s on MD5 hashing algorithm on a single GPU [30].

Browser-based CPU hash crackers written in JavaScript are generally slower than native CPU hash crackers due to their implementation and the browser's overhead. However, in recent years, a new browser-based CPU hash cracker was introduced: WebAssembly hash cracker. WebAssembly (Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for the compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications [31]. Wasm is designed to be faster to parse than JavaScript, as well as faster to execute, enabling very compact code representation [32]. This led to the introduction of Coinhive [17], the first browser-based CPU cryptocurrency miner. Coinhive uses Wasm to increase the hash rate and reduce JavaScript overhead. As a result, the Coinhive Wasm miner outperforms JavaScript miners. Coinhive [17] even states that they are able to reach about 65% of the performance of a native miner.
At the time of writing, we found no efficient browser-based GPU hash crackers.

2.2 Browser-based Distributed Attacks

Distributed computing takes complex computing tasks, such as breaking cryptographic hashes, and splits them up into smaller parts. It then sends them out to many different personal computers or servers to be processed in parallel and return with results. This parallel processing of many smaller parts serves to significantly reduce the time needed to compute each given task. In general, distributed computing uses abundant compute resources, including many CPUs, high network bandwidth, and a diverse set of IP addresses.

A browser-based distributed attack allows the attacker to exploit web users to perform distributed tasks at will. The attack starts when a victim enters a web page that is controlled by the attacker. Opening a web page causes the web browser to initiate a series of background communication messages to fetch and display the requested page. This requires the client's web browser to download and run code that is served from the website on the client's device. The code that gets executed in the client's browser is assumed to be related to the functionality of the site being browsed. Technically, however, there is nothing stopping a website from serving arbitrary code that is not related to the browsing experience. With the absence of any protection, the client's web browser will blindly execute whatever code it downloads from the website.

At this point, a malicious code can run and gain full access to the web browser's API, which presents an increasingly powerful set of web technologies. The code is transient and difficult to detect once the user has navigated away from the website. This gives the attacker access to the compute resources of all concurrent website visitors at any given time. This amount of compute power is especially significant on high-traffic websites. An attacker can take advantage of these opportunities to execute large-scale browser-based distributed attacks. For example, these attacks may exploit the victims' compute resources to perform in-browser distributed hash cracking.

In [33], Dorsey explains how easy it is to execute distributed attacks on the browsers of unsuspecting users. With small effort and funds (spending less than 100$), he managed to reach thousands of victims, using paid advertisements as a means of distribution. This allowed him to freely run the code of his choosing in the clients' browsers. He then demonstrated the feasibility of CPU mining bots, distributed denial of service bots, torrent bots, and more.

2.3 Differences between WebGL and CPU Implementations

As stated in the introduction, WebGL is designed to provide graphics operations, so it is naturally more difficult to implement a GPU hash cracker than a CPU hash cracker. WebGL 1.0 has several limitations that create difficulties in implementing hash crackers. These include inability to return values other than pixel color, lack of dynamic access to arrays, no debugging abilities, lack of bitwise operators, and no 32-bit support. All of these are needed to implement any cryptographic function. WebGL 2.0 makes the browser more vulnerable, as it introduces an improvement in its capabilities and relaxes some of the implementation limitations. It adds support for 32-bit integers and provides implementation for the majority of bitwise operators (not all); however, the rest of the limitations are still present.

GPUs are massively parallel, with hundreds (if not thousands) of stream processors that can simultaneously calculate hashes. Although a CPU core is much faster than a GPU core, the CPU usually limited to only four to eight cores.

Hash cracking is highly suited to parallel computing due to the need to execute the same cryptographic functions on independent data sets. This gives GPUs a tremendous edge in hash cracking over CPUs. The open question we address is whether overcoming WebGL limitations would inflict a heavy cost on performance and eliminate the GPU hardware advantage over the CPU.

3 Related Work

The feasibility of performing stealthy calculations using HTML5 Web Workers is presented by Rushanan [34]. Web Workers are JavaScripts that run in the background, allowing web applications to spawn background workers in parallel to the main thread. Namely, it is possible to perform calculations without blocking the main thread, leaving the web pages' UI loading time unaffected. After the initial loading time, which takes about 3 seconds, Rushanan [34] managed to calculate 500K MD5 hashes per second. Although the web page loading time seems unaffected, there is a rise in the CPU load. The rise in the CPU load might be noticeable if the user performs CPU intensive processes during the attack.

The first attempt to use WebGL 1.0 for distributed hash cracking was presented by MWR Labs [35]. Even after overcoming the hurdles involved, which included packing input into textures, computing using a shader, and retrieving output from images, it became clear to them that this method of attack was not feasible. The authors [35] indicated that this was related to limitations in the WebGL 1.0 shading language, especially the lack of support for 32-bit integers and bitwise operations.

One year later, Marc Blanchou presented his efforts to overcome both of the above challenges [16]. He tried to use a vector with two floats and implemented the bitwise operations. However, this attempt was not an efficient implementation. Similar to the previous work cited [35], he concluded that the use of WebGL 1.0 for hash cracking was not cost effective. Blanchou also presented a comparison to a simple JavaScript CPU hash cracker, and showed that it would be faster. Furthermore, he added that the upcoming support of OpenGL ES 3.0 (upon which WebGL 2.0 is based) is an upgrade and would not have the limitations of WebGL 1.0.

Recently, a new phenomenon known as cryptojacking was discovered. This involves the in-browser mining of crypto-currencies, sometimes even after the browser window is closed, [19, 36, 37, 38, 39], and more. Specifically, this entails the mining of Monero[18] using Coinhive[40, 17] or similar JavaScript CPU mining tools. Several papers in the past year performed a comprehensive analysis and in-depth study of cryptojacking [41, 42, 20, 21]. These papers examine and analyze the phenomenon and its prevalence, each in its own unique way. To overcome the naive detection methods, modern mining tools commonly use evasion techniques such as limiting CPU usage, code obfuscation, and hiding the malicious code in popular third-party libraries. Both Hong et al. [20] and Konoth et al. [21] introduce countermeasures that enable the user to detect the different in-browser CPU mining tools more efficiently than the existing naive methods. Some websites may use such mining tools as an alternative to ad-based financing or offer premium content in exchange for mining. Other websites unknowingly fall victim to attacks that cause them to unwittingly serve mining code that uses the computer resources of its visitors.

Both Hong et al. [20] and Konoth et al. [21] show that detecting miners by means of blacklists, string patterns, or CPU throttling alone is an ineffective strategy, because of both false positives and false negatives. They thoroughly explored the mining attack structure, miner communication, and distribution methods of current in-browser CPU mining tools and provided important insights that allowed them to implement more efficient defense mechanisms. They prove that it is more effective to use their suggested behavioral-based detection methods by using either static analysis [21] or runtime profilers [20].

Both Eskandari et al. [42] and Rüth et al. [41] tried to investigate how often cryptojacking occurs in websites using two straightforward approaches. Eskandari et al. [42] queried the top million sites indexed by Zmap.io and PublicWWW.com to determine which websites contained coinhive.min.js script in their body. Over $30,000$ websites were found. Rüth et al. [41] inspected the .com/.net/.org and Alexa Top 1M domains for mining code, and found mining code in a relatively low percentage 0.08% of the probed sites; however, this still accounts for more than 100,000 websites.

Konoth et al. [21] crawled Alexa's Top 1 Million websites for a week. Using static analysis, they managed to detect 1,735 websites containing cryptojacking code out of 991,513 websites in total, meaning 0.18%.

On the other hand, Hong et al. [20] used their CMTracker detector, which has two runtime behaviour-based profilers, to collect 2,770 cryptojacking samples from 853,936 popular web pages, including 868 among the top 100K in Alexa's list, meaning 0.32%. In addition, according to their findings, 53.9% of these identified samples would have not been identified with current widely used detectors that are based on blacklists.

The increase use of crypto-currency as an alternative means of payment and the rise in performance and compute resources provided by in-browser coding, specifically with the use of WebAssembly, have made cryptojacking very appealing to criminals as a continuous source of income.

We predict that cryptojacking has the potential to be very profitable in high traffic websites, but the potential harm to users introduces ethical problems that must be considered. Some of the problems include higher energy bills, accelerated device degradation, slower system performance, and poor web experience. This forecast led researchers [20, 21] to address the magnitude of the potential harm and investigate potential defense mechanisms against this type of CPU-based in-browser cryptojacking. Some researchers state that the trust model of web, which considers web publishers as trusted and allows them to execute code on the client-side without any restrictions is flawed and needs reconsideration [22]. Furthermore, it is essential to explore other means of in-browser cryptojacking that may attract attackers, alongside effective defense mechanisms where necessary. This increases the importance of our WebGL research.

4 Password Cracking Attack

Although researchers previously implemented hash cracking using WebGL 1.0, they haven't optimized it thoroughly, nor compared hash cracking over WebGL 1.0 to WebGL 2.0. This section reviews the challenges WebGL 1.0 and WebGL 2.0 present in implementing hash functions. It also briefly describes the implementations of password cracking using WebGL 1.0 and WebGL 2.0.

Our work presents new optimizations that improve previous results [35, 16]. We show that password cracking using WebGL 1.0 can be as efficient as the exploit of CPU in computational aspects, while WebGL 2.0 outperforms the CPU. For the sake of comparison to previous works, the first algorithm we evaluate is MD5. We also introduce the improvements WebGL 2.0 brings and compare them to WebGL 1.0 and CPU results.

4.1 Implementing MD5 in WebGL

The MD5 [43] hashing computation conflics with the WebGL limitations presented in Section 2.3. The MD5 hashing algorithm processes a variable-length message into a fixed-length output of 128-bits, while WebGL's maximum output length is 32-bits. The algorithm consists primarily of bitwise operations operating on 32-bit variables, while WebGL 1.0 doesn't support 32-bit variables. These bitwise operations include rotate, XOR, AND, OR, and NOT. WebGL 1.0 doesn't implement bitwise operators at all, and WebGL 2.0 is still missing some of them. This means we had to implement the missing bitwise operators ourselves in WebGL, and define a new unit for WebGL 1.0 to support 32-bit calculations. We also had to efficiently divide the hashing process to adjust it for the reduced output size.

For the password cracking attack, the MD5 hashing algorithm needs to be implemented in WebGL while overcoming all the presented challenges. Unlike WebGL 1.0, WebGL 2.0 has fewer limitations and it would be easier to implement the hashing algorithm. Therefore, we expect WebGL 2.0 to provide a significant performance boost as a result of these improvements.

4.2 Evaluation

Implementing a simple straightforward brute-force MD5 hasher on WebGL is not enough. There are several algorithmic improvements and optimizations that must be evaluated and considered. These optimizations should use multi-threading effectively and maximize the number of calculations done per each draw call.

Furthermore, the evaluation done by Rushanan [34] was performed on an old i5 processor and therefore the results are obsolete. It was essential to get more up-to-date results on how an MD5 hasher performs on a browser using a modern CPU, and to compare these results as well.

Based on a GitHub project of an MD5 brute-force password cracker [44], we implemented a JavaScript password cracker we could evaluate on a modern CPU.

Evaluation Experiment To evaluate the proposed algorithms we devised an experiment on our personal computer to compare the results of WebGL 1.0, WebGL 2.0, and CPU-based techniques. We tested several numbers of threads and HTML workers [45] for both WebGL 2.0 and CPU password crackers. For the WebGL 2.0 miner, our parameters are calculations per thread and the number of threads that combined give us calculations per draw. For the CPU miner, we were able to control the number of HTML workers. We ran each combination in turn for a duration of two minutes. During each one of the combinations, we measured the load on the GPU or CPU accordingly. This allowed us to find the best trade-off values to use, while keeping the highest possible hash rate that would not reduce performance for the user. Before the evaluation process, we tested several parameters for WebGL 1.0 password cracker and found the best trade-off values as well.

We performed one experiment on password cracking as an intro to in-browser attacks in general, and more specifically WebGL attacks. Our goal was to test how well the WebGL password cracker performed compared to a CPU-based cracker. For this test, we didn't evaluate large scale experiments that challenge other aspects of distributed attacks, such as: user experience, stealthiness, and possible defenses.

We did, however, evaluate large scale extended experiments on cryptocurrency mining attacks, as detailed in Section 5. Password cracking attacks don't pose as viable a threat as cryptojacking, and are much less common than cryptocurrency mining attacks that use in-browser techniques.

Evaluation Setup Intel i5 6600K @3.5Ghz, 2X4GB @2400MHz, Radeon HD6870, Chrome Canary 64.0.3282

4.2.1 Experiment 1 : Password Cracking

Goal: Compare the effectiveness of password cracking using WebGL 1.0, WebGL 2.0, and CPU techniques. Also examine when the user starts to notice a password cracking process running in the background while using any of the above techniques.

Process: We ran each of the proposed techniques in turn for a duration of two minutes for each parameter, while the user continued using the web-browser and other programs to simulate practical conditions. We then compared the hash rate achieved by each of the techniques and checked with the user when they noticed any impact on their usual experience.

Results: The best trade-off parameters for the evaluation were the ones that gave us the highest hash rate, while keeping the load around 50%. Although better hash rates than the chosen results in Table 1 can be achieved in WebGL and CPU, these are the best trade-off values that did not affect the user experience.

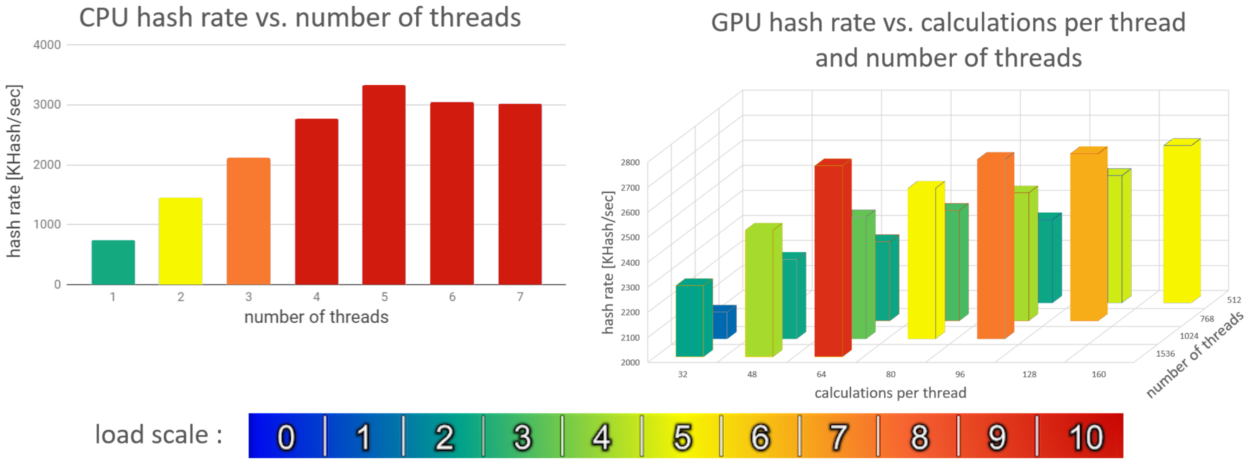

Figure 1: Password Cracking - Experiment 1 Results
Each column in both graphs presents the hash rate achieved and it's corresponding parameter. Each column is colored according to the measured percentage load.

We can see that even our best result in WebGL 1.0 doesn't perform as well as the hash rate we managed to achieve on a modern CPU. However, there is a significant improvement when using WebGL 2.0 compared to WebGL 1.0; it even outperforms the hash rate achieved in our CPU hasher.

Table 1: Comparison between CPU, WebGL 1.0 and WebGL 2.0 hash rate results using best trade-off parameters

	threads	calculations per thread	Hashes in 10 minutes	Rate [hash/sec]
WebGL 1.0	256	16	65212416	108687
WebGL 2.0	512	160	1577615360	2629359
CPU	2	NA	870187800	1459854

As part of the evaluation, we also combined both WebGL and CPU hashers to run simultaneously. This allowed us to gain the full benefit of both GPU and CPU due to the fact that each one of them uses different hardware resources.

This experiment shows the effectiveness of WebGL-based password cracking attacks.

5 Cryptocurrency Mining Attack

The rising popularity of both purchasing and mining cryptocurrency, caused a significant growth of the mining community and the introduction of new cryptocurrencies. This section introduces the two cryptocurrencies we experimented with: Monero [18] and Litecoin [46]. For Monero, we evaluated the effectiveness of WebGL in distributed cryptocurrency mining and compared it to the JavaScript CPU miner, Coinhive[17]. While for cryptomining Litecoin, we only did an initial WebGL evaluation and compared it to a native GPU miner as additional studies. This seemed logical since Monero is more popular than Litecoin.

As noted in Section 6.1, we wanted to compare the miner part of the cryptocurrency mining attack. We do not address the performance of other aspects in distributed cryptocurrency mining, such as mining pools, server side, databases, and more. For these elements we used commercial tools to which we added our WebGL miner; we only measured the mining performance.

5.1 Monero

A new type of cryptocurrency mining was introduced: CPU mining using JavaScript-based miners [19]. These JavaScript mining tools can be injected into popular websites as a source of income, at the expense of users' resources. The most common cryptocurrency mined using these web browser mining tools was Monero [18]. Monero experienced rapid growth in market capitalization and transaction volume during the year 2016. This was partly due to its adoption by the major darknet market AlphaBay, which was later closed down in mid-2017 [47, 48].

Monero's proof of work, the process that must be done to ensure a block is valid before it is added to the blockchain, is based on the underlying CryptoNote protocol [49], called CryptoNight [50]. Unlike Litecoin and other cryptocurrencies that are derivatives of Bitcoin, the CryptoNight proof-of-work hash algorithm is a memory-intense function. Monero's proof-of-work algorithm is designed to be inefficiently computable on GPU, FPGA, and ASIC architectures, which makes it ideal for mining on CPUs. Therefore, the two main features of the algorithm that challenge our WebGL implementation are:

- CryptoNight uses large fast memory to work on 2MB (L2 cache size), which requires a lot of silicon. This is far more than what is needed by the SHA-256 circuitry used for Bitcoin, Litecoin, and other similar cryptocurrency mining algorithms.
- CryptoNight is based on AES (Advanced Encryption Standards, a cryptographic cipher applied by the majority of organizations) [51] and was designed to take advantage of the AES-NI instruction set [52], which uses existing hardware circuitry on modern x86_64 CPUs to speed up AES operations.

The following section describes the challenges presented in implementing the CryptoNight hashing algorithm in WebGL.

5.2 WebGL Monero Implementation Challenges

Implementing cryptocurrency mining on WebGL 1.0 has proven to be inefficient. The performance cost is too high since the use of bitwise operators and 32-bit variables (which we implemented ourselves) is significantly greater compared to the MD5 hashing algorithm. Therefore, we evaluated cryptocurrency mining experiments only with WebGL 2.0.

After the initialization of an AES key from the input using the Keccak hashing algorithm [53], the CryptoNight algorithm consists of three main steps: initializing a 2 MB scratch pad, executing a memory-hard loop, and finalizing the hash output. Each step presents different implementation challenges.

WebGL's memory limit. WebGL is unable to allocate a consecutive 2MB memory array to be used as a scratch pad for the first step. We could split the array into smaller chunks, but it would significantly affect the algorithm's performance.

Potentially long runtime. After initializing the scratch pad, a memory-hard loop of 524288 iterations on non-consecutive array elements is performed; this can take a significant amount of time. WebGL calls are done in the main UI thread, so we want to minimize the time it takes for a WebGL call to return. Long periods would cause the users to feel the page was unresponsive or even lead to a context-lost event for WebGL.

Large shader code. For the CryptoNight output, a hash function is randomly chosen out of four possibilities and applied on the state resulting from the previous steps. The resulting hash is the output of CryptoNight's algorithm. This step presents us with the new challenge of implementing all four possible hash functions: Blake [54], Groestl [55], JH [56], and Skein [57]. This would result in a major increase of the shader code size.

Finally, as we stated before, GPUs are all about parallelism. We needed to make the CryptoNight algorithm parallel, despite the fact that there is no natural way to share data between shader threads.

The question remains whether overcoming the challenges in implementing the CryptoNight algorithm on WebGL would result in a severe cost in performance and make WebGL unusable for Monero mining.

5.3 Experiment 2: Cryptomining Performance and User Experience

We overcame the major challenges presented in each step of the CryptoNight algorithm by using data textures, dividing the data efficiently, and moving some of the large code but non-intensive calculations to the CPU side. After implementing Monero's algorithm in WebGL 2.0, we can proceed to evaluating the major aspects of the Monero mining attack.

In our experiment, we compared our WebGL 2.0 miner to the Coinhive browser-based CPU miner introduced in Section 4. The results compared the performance, user experience, and hardware load locally on our computer.

Evaluation Setup Intel i5 6600K @3.5Ghz, 2X4GB @2400MHz, GeForce GTX1080, Chrome Canary 64.0.3282

Similar to evaluating the password cracking attack, we tested different parameters to determine how the GPU and CPU load are affected. This allowed us to find the best trade-off values for a cryptocurrency mining attack. To evaluate the proposed algorithms, we devised an experiment to compare the results of WebGL 2.0 and CPU-based techniques. We tested several parameters for both WebGL 2.0 and Coinhive Monero miners. For the WebGL 2.0 miner our parameters were: number of threads and time between draws. Shorter times between draw calls means a higher hash rate, but it also leads to a higher GPU load. WebGL 2.0 did not allow us to choose more than 32 threads.

For the CPU miner, we controlled the number of HTML workers [45] in Coinhive. We ran each parameter combination of the GPU and CPU miners in turn for the duration of two minutes. During each of the combinations, we measured the hash rate and the load on the GPU or CPU, accordingly.

Goal: Find the best trade-off evaluation parameters and compare the effectiveness of Monero cryptocurrency mining using WebGL 2.0 and CPU techniques.

Process: We ran each of the proposed techniques and parameters in turn for a duration of two minutes while the user continued using the web-browser and other programs to simulate practical conditions. We then compared the hash rate achieved by each of the techniques and checked with the user when they felt any effect on their usual experience.

Results: The best trade-off parameters for the evaluation were the ones that gave us the highest hash rate, while still not affecting the user experience.

We can see that the best trade-off results would be achieved if we limit Coinhive to using 2 threads, which leads to about 60% CPU usage. For the WebGL 2.0 miner, the best performance was achieved using 32 threads with a 1 second draw delay, resulting in 50% GPU usage. Using these best trade-off parameters, we reached a hash rate of 25 hashes/sec

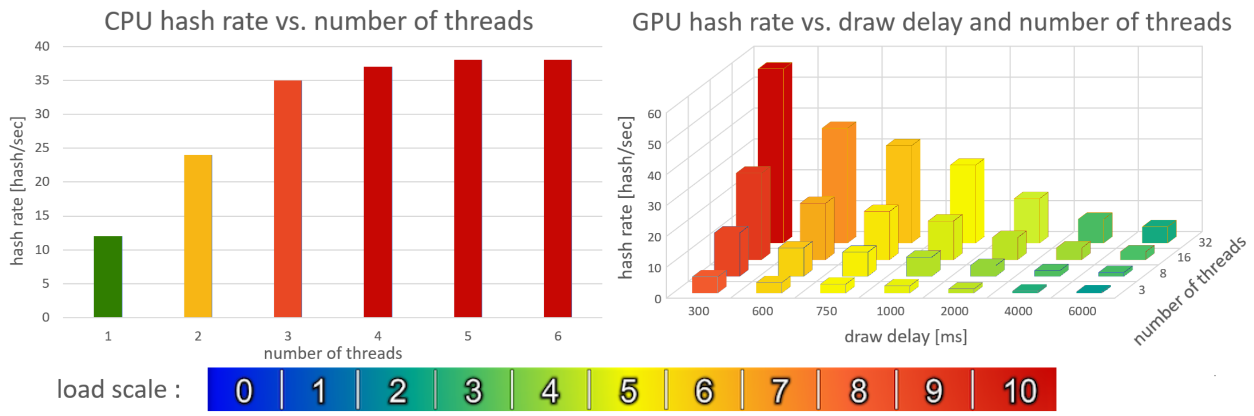

Figure 2: Cryptomining - Experiment 2 Results

Each column in both graphs presents the hash rate achieved and it's corresponding parameter. Each column is colored according to the measured percentage load.

in WebGL and 24 hashes/sec in Coinhive. We observed that the WebGL miner is just as fast as the Coinhive miner. We did not detect any impact on the user experience when we evaluated performance on a single user's machine.

5.3.1 Litecoin

As additional studies we decided to test whether WebGL mining is efficient with another type of cryptocurrency, called Litecoin [46]. Litecoin gained a lot of popularity and shows great promise. It's mining difficulty is lower than that of Bitcoin [58], and the market is not yet saturated with dedicated ASIC miners. Litecoin is a fork of the Bitcoin core client. It differs primarily in requiring less time to mine a block (2 minutes instead of 10 minutes), a higher maximum number of coins, and different hashing algorithm: Scrypt [59] instead of SHA-256 [60].

Implementation and Challenges Cryptocurrencies that use Scrypt are often mined on GPUs, which tend to have significantly more processing power (for some algorithms) compared to the CPU. Although at first the Scrypt algorithm seems to be complicated, implementing Scrypt in WebGL 2.0 is relatively straightforward because it is suitable for GPU computation. The main difficulties in implementing Scrypt in WebGL 2.0 involve:

- Splitting the algorithm to allow the dynamic hash output to be divided between the 32-bit pixel output from each shader.
- Implementing PBKDF2(HMAC-SHA256) [61], which includes SHA-256 implementation with different key lengths, while keeping the shader code small enough so we don't reach the maximum number of instructions per shader program.

Evaluation The initial hash rate we managed to achieve with WebGL 2.0 was around 210 kHash/sec, which is about half the hash rate a native GPU miner (OpenCL[62]) can reach with the same hardware [63, 29].

6 Evaluating the Distributed Attack

The previous two sections analyzed the exploit of WebGL 2.0 from the narrow perspective of single user. An attacker who aims to abuse WebGL for password cracking or cryptocurrency mining, must launch the attack on many web users. This section analyzes the practical aspects of distributing these attacks.

Of the two attacks, we chose to evaluate the distributed attack on cryptocurrency mining for two reasons:

1. Distributed cryptocurrency mining attacks are more common today.
2. We wanted to evaluate distributed attacks from a practical perspective. In cryptocurrency mining attacks, we have other real-world implementations of the attack that can be used for comparison.

Due to the similarity between the attacks, we believe that the findings can also be used to reach the same conclusions for password cracking.

In Section 6.1 we introduce cryptocurrency mining distributed attacks and our implementation. Section 6.2 discusses different aspects of distribution on a large scale and how we handled them.

In Section 6.4 the experiment evaluates the user experience of the attack using our WebGL miner and Coinhive with specific parameters. Section 6.5 describes our experiment to check the stealthiness of the attack. We wanted to examine whether users could locate the attack if they are aware it is running on their computer. Each of the experiments focuses on a different aspect of distributed cryptocurrency mining attacks, while providing additional data about the mining hash rate.

6.1 Implementing the Distributed Attack

Similar to what we introduced in 2, the goal of distributed attacks for cryptocurrency mining is to find input data for which, after applying the appropriate hashing function, the result matches the attacker's desired criterion: it needs to be less than the chosen cryptocurrency's current target value. This would give the mining reward to the attacker based on work done using the victims' resources. The attack is performed by dividing the data search among as many clients as possible.

The process of distributed hash cracking attacks on the victim's browser works as follows:

1. Malicious code reaches a victim as per the distribution methods described in Section 6.2
2. Attacker's code in the victim's browser sends a message to the attack server asking for a target and data range
3. The code calculates the appropriate hash function on the data range using the victim's resources
4. The code compares the crypto hash function's output to the desired target value
5. The results are posted from the victim's browser to the attack server
6. The server sends a new target and data range to run on the victim's browser
7. The process returns to Step 3 until there are no calculations to be done for the attacker

In addition to the attack process on the victims' side as described above, the cryptocurrency mining attack involves several additional elements that an attacker needs to address. Cryptocurrency mining starts by creating a new mining pool or joining an existing one. The mining pool is a group of cooperating miners who agree to share block rewards in proportion to the mining hash power they contribute. After the server has a working mining pool, it is ready to send mining tasks to its member clients.

As stated in Section 4, cryptocurrency mining is a race to find the corresponding hash, so time is of essence. Performance and speed play a vital part in the success of cryptocurrency mining. Even if we are behind the competitors by just a fraction from posting the correct hash, this will mean we miss out on the mining reward. In a distributed cryptocurrency mining attack, the server can divide the data range between its cooperating miners and increase the chances of finding a match to the transaction hash. The potential reward increases with the computational speed.

In the following experiments we evaluated Step 3 of the distributed hash cracking process (described above) using WebGL calculations. We show the effectiveness of WebGL in mining Monero cryptocurrency as compared to an equivalent CPU hasher. This should prove that WebGL can handle mining cryptocurrency that is of significant relevance today.

6.2 Large Scale Distribution Aspects

In this section we describe the different issues an attacker needs to consider before launching a large-scale distributed attack, and how we resolved them for our evaluations.

The browser's GPU can be abused by an attacker to conduct efficient distributed password cracking attacks, cryptocurrency mining, and more. To start spreading the attack, the attacker just needs to somehow inject their JavaScript code into a website that will reach as many users as possible. An attacker can achieve this through several methods, depending on her technical knowledge and resources:

- Come to an arrangement with websites to insert attacker code, for example, by sharing earnings
- Pay an ad company to pop ads containing attacker's code [33], this can be done with or without the ad company's knowledge
- Inject attacker code maliciously into websites
- Come to an arrangement with extension companies, such as AdBlock[64]

- Develop a popular website to lure victims

An attacker who wishes to launch such an attack on many web users for an extended period of time, must consider additional aspects of the attack:

1. *Stealthiness*. The attack must be conducted without arousing the user's suspicion. If the user feels any impact on his browsing experience, he may close the website or even contact the website's owner. This can lead to detection and prevent the attack from running for a prolonged period of time.
2. *Management*. Assuming thousands of browsers run the attack for different periods of time, it is necessary to manage them all to maximize the profit. Duplicate runs need to be avoided because they waste valuable computing power. Moreover, there is a need to keep track of users going offline to prevent computations from being skipped. To accomplish this, the attacker needs an efficient and synchronous control server.

6.3 Implementing Distributed Attack Experiments

In our experiments, we didn't need any of the distribution methods mentioned above because we had volunteers who knowingly entered websites that contained our attacking code.

In Section 5, for CPU mining experiments we used Coinhive, which handled the management of all the users. It also enabled some degree of stealthiness by setting the CPU usage limit to avoid detection and by the use of WASM, which makes it difficult to find the mining code.

For experimental purposes only, we implemented our own naive WebGL mining server to handle our user scale. Then we installed a NodeJS server [65] that served mining jobs to each of the clients. We used a MongoDB [66] to keep track of ongoing work and store the target hashes. To keep things simple and isolate our WebGL miner, we didn't connect it to any active Monero mining pools.

For the client side, we had an iframe with obfuscated code; this received the WebGL mining code from our server and contributed greatly to our miner's stealthiness. We also limited the WebGL performance, using the best trade-off parameters observed in Experiment 2 in Section 5.3 to prevent high GPU usage, with a minimal effect on the user experience.

As stated in Section 6.1, we only planned to evaluate the effectiveness and performance of Step 3 of the in-browser attack process. Consequently, we used commercial tools for the other steps to narrow our measurements to the achieved hash rate and user experience.

6.4 Experiment 3: Distributed Cryptocurrency Mining Performance and User Experience

The following distributed experiment extended the previous one described in Section 5.3 to test several chosen best trade-off parameters on a larger scale. Our GPU was relatively high end and to see how well we could perform on weaker GPUs, we also used some of the parameters that showed only 30% load on our GPU. This enabled us to show the relevance of our local results to a wider range of GPUs and CPUs. Similar to previous experiments, we ran each combination in turn for two minutes. During each of the combinations, the user was asked to state whether he felt any effect on his computer's performance.

Goal: Check whether the user notices the effect of a cryptocurrency mining process running in the background. Further compare the effectiveness of distributed cryptocurrency mining using WebGL 2.0 and CPU techniques.

Methodology and ethics: The experiment was carried out with 100 volunteers. All the volunteers were paid to cover their expenses (primarily electricity), signed a consent form, and used their own computers to simulate a more realistic scenario. Mining can result in high electricity use and even lead to physical damage to less suitable devices (overheating cellphones for instance), so we only ran the experiment on personal computers. To avoid unintentional bias by participants and/or staff, the experiment was 'double blinded'. The users were assigned randomly to one of two sets, (*CPU mining* or *GPU mining*), without either the user of the staff being aware of the assignment. We didn't collect any statistics or personal information from the volunteers so there were no privacy issues.

Process: Users were asked to visit the website that contained our cryptocurrency mining logic and leave it open to run in the background. The users were then instructed to continue using the web-browser and other programs to simulate practical conditions for the duration of 10 minutes, and write down the times when they felt any influence on their browser's performance. There were three possible answers: (1) Significant effect. (2) Minor effect. (3) No effect. We then randomly assigned to each visitor one of the two mining options: *CPU mining* or *GPU mining*. For

CPU mining users, we ran Coinhive Wasm miner in the background. For *GPU mining* users, we ran our WebGL miner in the background.

For each visitor, we saved the hash rate we managed to achieve with each parameter combination, and used it to compare between GPU and CPU mining hash rates.

Results: We observed that most of the users assigned to a CPU miner did notice some effect on their browsing experience.

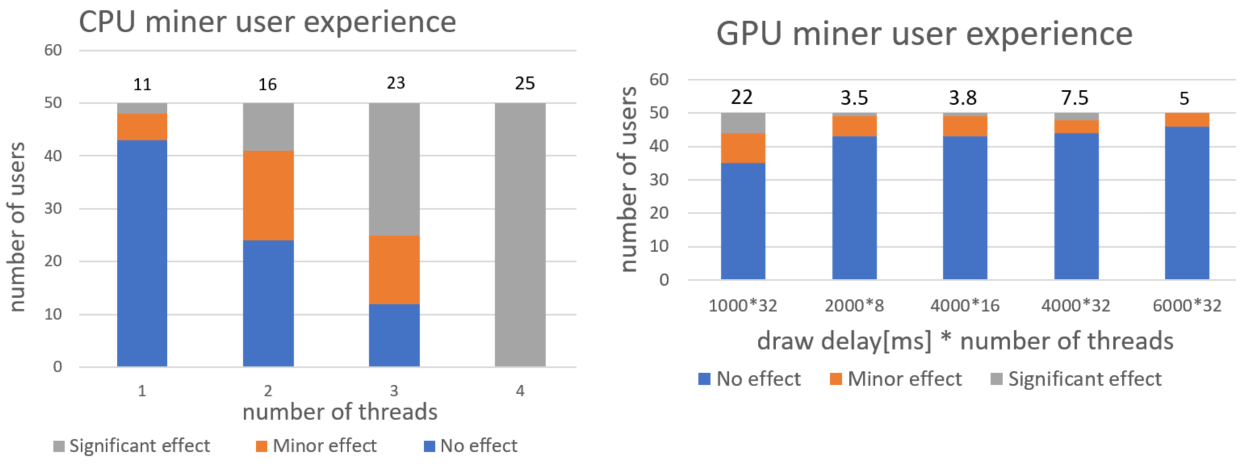

Figure 3: Distributed Cryptomining - Experiment 3 Results
The results show the degree to which users felt their computer's performance was affected, if at all. Above each column is the average hash rate [hash/sec].

A substantial number of users have CPU usage monitoring tools that allowed them to immediately notice the browser's high CPU use. Moreover, some of the users were simultaneously running CPU intensive tasks in the background, which resulted in a significant deterioration of their computer's performance.

On the other hand, only a few users out of the GPU mining group reported noticing any effect. The common ground for the users that did feel some deterioration in the GPU group was that it occurred during the period with a high number of threads and a low draw delay. Moreover, their computers were relatively old and weak, in particular their GPU (more than five years old) or integrated GPUs.

We can see that the best trade-off parameters from the previous experiment gave us a high average hash rate of 22 hashes/sec. However, almost a third of the users did notice some effect on their performance when using this parameter combination of 32 threads with a 1 second draw delay. Perhaps this could have been avoided if we had added adaptation logic in our WebGL 2.0 miner to make it reduce its performance on weaker machines. Until we do, in our opinion, it is better to choose less demanding parameters for a wide range attack.

The best trade-off values for WebGL 2.0 that give us the highest hash rate while having a minimal effect on user experience (about 10%) are: 32 threads and a 4 second draw delay. This gave us the average hash rate of 7.5 hashes/sec for the WebGL 2.0 miner, while the Coinhive miner's average hash rate reached 16 hashes/sec with 2 threads. That said, if we had demanded the same degree of minimal effect on user experience from the CPU miner, we would need to limit Coinhive to using only 1 thread, giving us an average hash rate of 11 hashes/sec. We concluded that, although we could reach a higher hash rate in WebGL 2.0, we should compromise on lower values until we optimize the miner with adaptation logic.

This experiment demonstrated that the CPU hash rate is faster than the GPU hash rate in typical home machines. The results also indicate that the user experience is affected much more during CPU attacks than GPU attacks, making the CPU attacks easier to notice and detect. Sixty percent of the CPU miner group reported degraded performance compared to only 15% on average in the GPU miner group. This corroborates our view that the GPU can be used for effective attacks.

The fact that users notice an intensive task happening in the background doesn't necessarily mean that they can find it. We therefore conducted Experiment 4 to evaluate how difficult it would be for them to discover the attack in each miner type (CPU or GPU).

6.5 Experiment 4: Cryptocurrency Mining Stealthiness

Goal: Using the best trade-off evaluation parameters from previous experiments, observe whether the user can locate our cryptocurrency mining code running in the background.

Methodology and ethics: Similar to the previous experiment to measure the user experience during cryptocurrency mining, the experiment was carried out with 80 volunteers. The volunteers were paid to cover their expenses (primarily electricity), signed a consent form, and used their own computers to simulate a realistic scenario. Mining could lead to physical damage on less suitable devices (overheating cellphones for instance), so we only ran the experiment on their personal computers. To avoid unintentional bias by participants and/or staff, the experiment was 'double blinded.' The users were randomly assigned to one of two sets, *CPU mining* or *GPU mining*, without either the user or staff aware of the assignment. The users were told that we are going to try to steal money from them and they need to find out how. We didn't collect any statistics or personal information from the visitors so there were no privacy issues.

Process: Each user received a list of five different websites. One of them contained our mining logic from the previous experiment; the other websites were 'clean'. The users were instructed to open all of the websites and leave them open in the background. We then randomly assigned each visitor to one of the two mining options: *CPU mining* or *GPU mining*. For *CPU mining* users, we ran Coinhive miner in the background. For *GPU mining* users, we ran our WebGL miner in the background. The users had to detect how we were stealing money from them i.e., find our CPU miner or GPU miner (depending what they were assigned).

There were four possible outcomes to this experiment: 1) Found *CPU mining*, 2) Found *GPU mining*, 3) Didn't find *CPU mining*, 4) Didn't find *GPU mining*.

Results: We observed that when it comes to stealthiness, GPU-based miners are much more effective and harder to detect than CPU-based miners. In the CPU group, 22 out of 40 users were able to find the CPU mining code, representing 55% of the users. Out of 40 users in the GPU group, only 1 user with a high level of technical knowledge in web research was able to find the GPU mining code, representing only 2.5%. Clearly, an attacker who launches a GPU-based attack on a large scale can be less concerned about detection. This strengthens the advantages of GPU miners as compared to CPU miners.

7 Defenses

Today, WebGL is still not widely used by most of the common websites. Most do not harness the power of the GPU to perform any of the rendering on their pages. Apparently, WebGL is primarily being used in online web games or websites containing 3D imaging. Therefore, in our opinion, the most effective way to prevent WebGL attacks would be using an extension that can disable WebGL when it's not supposed to be used.

Similar to web notifications [67], which can send alerts to the user outside the context of a web page, it would be preferable to have WebGL disabled by default. The user would have the option to enable WebGL for each website individually, if it is required.

We considered looking into other directions for defense mechanisms to prevent the abuse of WebGL. For example, trying to detect when the canvas color is posted to a remote server, since it should only be used locally for further rendering. Another example might detect the increased use of bitwise operations inside shader code, which typically characterizes crypto calculations. However, we felt that this would be an overkill and we should consider a more simple solution.

Until a solution is addressed by browsers, our extension could operate as follows:

- Detect that the current website is using WebGL.
- Alert the user that the current website is using WebGL.
- Ask the user if the current website is supposed to use WebGL in any way, say for: online gaming, 3D imaging, augmented reality, complex geometric rendering, and more.
- Allow the user to easily disable WebGL for the current website if none of the above conditions are met.

We could also maintain a blacklist of websites for which WebGL should be disabled by default and reduce the need for user interaction. This extension would enable us to minimize or even entirely eliminate WebGL attacks.

The following experiment allowed us to evaluate how well our extension works and to test our assumption that most of the websites don't use WebGL.

7.1 Experiment 5: Extension Effectiveness

Goal: Collect statistics about the use of WebGL in websites and test the extension's efficiency.

Methodology and ethics: The experiment was carried out with 50 volunteers who installed the extension on their personal computers for 3 weeks. They were instructed to keep their usual browsing habits and not try to test the extension intentionally. They were also asked to report if they noticed anything unusual: slow browsing, pages not loading, crashes, and so forth. The extension reported back statistics on how many websites the user visited, how many of them contained WebGL code, and how many users chose to disable WebGL. We didn't collect any personal information on the visitors so there were no privacy issues.

Process: Each volunteer installed our extension and continued using their personal computer as usual. After three weeks, we instructed them to uninstall the extension and we reviewed the statistics collected by the extension.

Results: None of the volunteers reported any issues regarding the extension.

During our evaluation, the extension encountered 1345 websites in total. Surprisingly, during the experiment period our extension came across very few websites containing WebGL. The extension encountered only 6 websites containing WebGL, and only one instance where the user chose to disable WebGL. We conjecture that some (if not all) of these WebGL website entries derive from the users wanting to challenge our extension, although they were instructed against this.

These results strengthen our assumption that WebGL websites are only a small fraction of existing sites and most websites don't use WebGL rendering. For the average user, disabling WebGL by default in their browsers, probably will not have any effect on their browsing experience and can only enhance their security.

8 Conclusions

Most of the popular web browsers today support WebGL 2.0 and it is enabled by default for all websites. WebGL is completely integrated into the web standards of these web browsers, allowing GPU-accelerated usage of image processing and 3D effects as part of the web page canvas. WebGL allows browsers to communicate directly with graphics hardware, enabling code to harness the GPU's power. Currently, web browsers have not implemented any countermeasures or detection mechanisms for malicious WebGL code or the misuse of WebGL.

Our experiments show that WebGL 2.0 has introduced major improvements over WebGL 1.0 both in performance and in convenience. We also show that WebGL 2.0 can be used in practical attacks, such as exploitation for cryptocurrency mining, where in some cases it even outperforms CPU-based attacks. We demonstrated how difficult it is for a user to detect attacks that use GPU-based techniques compared to similar CPU-based techniques. WebGL allows an attacker to benefit financially by abusing users' resources without their knowledge. Our work also suggests a practical defense mechanism.

We hope this paper will call attention to the problem and help tackle this vulnerability. It is our intention to raise awareness regarding the risk posed by WebGL, and the need for this risk to be addressed by web browsers.

References

[1] Mozilla Developer Network. Cache api. `https://developer.mozilla.org/en-US/docs/Web/API/Cache`.

[2] W3C. Service Workers, June 2015. `https://www.w3.org/TR/service-workers/`.

[3] Tom Van Goethem, Wouter Joosen, and Nick Nikiforakis. The clock is still ticking: Timing attacks in the modern web. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1382–1393. ACM, 2015.

[4] Nethanel Gelernter. Timing Attacks Have Never Been So Practical: Advanced Cross-Site Search Attacks. *Black Hat USA*, 2016.

[5] Tom Van Goethem, Mathy Vanhoef, Frank Piessens, and Wouter Joosen. Request and conquer: Exposing cross-origin resource size. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 447–462, Austin, TX, August 2016.

[6] Sangho Lee, Hyungsub Kim, and Jong Kim. Identifying cross-origin resource status using application cache. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015*.

[7] Nethanel Gelernter, Yoel Grinstein, and Amir Herzberg. Cross-site framing attacks. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 161–170. ACM, 2015.

[8] Jussi-Pekka Erkkilä. Websocket security analysis. *Aalto University School of Science*, pages 2–3, 2012.

[9] Giancarlo Pellegrino, Christian Rossow, Fabrice J Ryba, Thomas C Schmidt, and Matthias Wählisch. Cashing out the great cannon? on browser-based ddos attacks and economics. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, 2015.

[10] VT Lam, Spyros Antonatos, Periklis Akritidis, and Kostas G Anagnostakis. Puppetnets: misusing web browsers as a distributed attack infrastructure. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 221–234. ACM, 2006.

[11] Dean Jackson and Jeff Gilbert. WebGL 2.0 Specification. `https://www.khronos.org/registry/webgl/specs/latest/2.0/`, November 2016.

[12] Mozilla Developer Network. Webgl. `https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API`.

[13] Dan A Alcantara, Andrei Sharf, Fatemeh Abbasinejad, Shubhabrata Sengupta, Michael Mitzenmacher, John D Owens, and Nina Amenta. Real-time parallel hashing on the gpu. *ACM Transactions on Graphics (TOG)*, 28(5):154, 2009.

[14] Michael Bedford Taylor. Bitcoin and the age of bespoke silicon. In *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, page 16. IEEE Press, 2013.

[15] Dean Jackson. WebGL 1.0 Specification. `https://www.khronos.org/registry/webgl/specs/1.0/`, October 2014.

[16] Marc Blanchou. Harnessing GPUs Building Better Browser Based Botnets. *Black Hat Europe*, 2013.

[17] Coinhive – Monero JavaScript Mining. `https://coinhive.com/`.

[18] Home | Monero - secure, private, untraceable, 2014. `https://getmonero.org/`.

[19] Check Point Research Team. Crypto Miners – The Silent CPU Killer of 2017, 2017. `https://blog.checkpoint.com/2017/10/23/crypto-miners-the-silent-cpu-killer-of-2017`.

[20] Hong, Geng & Duan, Haixin & Yang, Zhemin & Yang, Sen & Zhang, Lei & Nan, Yuhong & Zhang, Zhibo & Yang, Min & Zhang, Yuan & Qian, Zhiyun. How You Get Shot in the Back: A Systematical Study about Cryptojacking in the Real World, 2018.

[21] Konoth, Radhesh Krishnan & Vineti, Emanuele & Moonsamy, Veelasha & Lindorfer, Martina & Kruegel, Christopher & Bos, Herbert & Vigna, Giovanni. MineSweeper: An In-depth Look into Drive-by Cryptocurrency Mining and Its Defense, 2018.

[22] Papadopoulos, Panagiotis & Ilia, Panagiotis & Polychronakis, Michalis & P. Markatos, Evangelos & Ioannidis, Sotiris & Vasiliadis, Giorgos. Master of Web Puppets: Abusing Web Browsers for Persistent and Stealthy Computation, 2018.

[23] Keeper Security. How Password Crackers Work and How to Stay Protected - Keeper Blog, September 2016. `https://keepersecurity.com/blog/2016/09/28/how-password-crackers-work/`.

[24] Blockgeeks. What Is Hashing? Under The Hood Of Blockchain, 2017. `https://blockgeeks.com/guides/what-is-hashing/`.

[25] BetterBuys. Estimating Password Cracking Times, 2016. `https://www.betterbuys.com/estimating-password-cracking-times/`.

[26] Team Hashcat. hashcat - advanced password recovery. `https://hashcat.net/hashcat/`.

[27] KoreLogic Security. Crack Me If You Can Contest. `https://contest.korelogic.com/`.

[28] FossBytes. World's Fastest Password Cracking Tool Hashcat Is Now Open Source, 2015. `https://fossbytes.com/worlds-fastest-password-cracking-tool-is -now-open-source/` .

[29] Jeremi M. Gosney. Nvidia GTX 1080 Ti Hashcat Benchmarks, April 2017. `https://gist.github.com/epixoip/ace60d09981be09544fdd35005051505/`.

[30] David Um. GPU Based Password Cracking with Amazon EC2 and oclHashcat, 2015. `http://www.rockfishsec.com/2015/05/gpu-password-cracking-with-amazon-ec2.html`.

[31] WASM. WebAssembly. `https://webassembly.org/`.

[32] WebAssembly High-Level Goals. https://github.com/WebAssembly/design/blob/master/HighLevelGoals.md.

[33] Brannon Dorsey. Browser as Botnet, or the Coming War on Your Web Browser, 2018. https://medium.com/@brannondorsey/browser-as-botnet-or-the-coming-war-on-your-web-browser-be920c4f718 .

[34] Michael Rushanan, David Russell, and Aviel D Rubin. Malloryworker: Stealthy computation and covert channels using web workers. In *International Workshop on Security and Trust Management*, pages 196–211. Springer, 2016.

[35] MWR Labs. Distributed Hash Cracking on the Web, Januray 2012. https://labs.mwrinfosecurity.com/blog/distributed-hash-cracking-on-the-web/.

[36] Josh Grunzweig. The Rise of the Cryptocurrency Miners, 2018. https://researchcenter.paloaltonetworks.com/2018/06/unit42-rise-cryptocurrency-miners/.

[37] Robert Hackett. Popular Google Chrome Extension Caught Mining Cryptocurrency on Thousands of Computers, 2018. http://fortune.com/2018/01/02/google-chrome-extension-cryptocurrency-mining-monero/ .

[38] Michael Nadeau. What is cryptojacking? How to prevent, detect, and recover from it. https://www.csoonline.com/article/3253572/internet/what-is-cryptojacking-how-to-prevent-detect-and-recover-from-it.html .

[39] Jérôme Segura . Persistent drive-by cryptomining coming to a browser near you, 2017. https://blog.malwarebytes.com/cybercrime/2017/11/persistent-drive-by-cryptomining-coming-to-a-browser-near-you/ .

[40] Brian Krebs. Who and What Is Coinhive?, 2018. https://krebsonsecurity.com/2018/03/who-and-what-is-coinhive/.

[41] Jan Rüth, Torsten Zimmermann, Konrad Wolsing, Oliver Hohlfeld. Digging into Browser-based Crypto Mining, 2018.

[42] Shayan Eskandari, Andreas Leoutsarakos, Troy Mursch, Jeremy Clark. A first look at browser-based cryptojacking, 2018.

[43] R. Rivest, MIT Laboratory for Computer Science and RSA Data Security, Inc. RFC 1321 - The MD5 Message-Digest Algorithm, April 1992. https://tools.ietf.org/html/rfc1321.

[44] Feross Aboukhadijeh. GitHub MD5 Password Cracker, 2012. https://github.com/feross/md5-password-cracker.js/.

[45] Mozilla Developer Network. Using web workers. https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers.

[46] Litecoin - Open source P2P digital currency. https://litecoin.org/.

[47] Eduardo Gómez. Monero (XMR) On The Rise Following Its Inclusion In The Darknet Market AlphaBay, 2016. https://themerkle.com/monero-xmr-on-the-rise-following-its-inclusion-in-the-darknet-market-alphabay/ .

[48] JP Buntinx. The Early History of Monero in 500 Words, 2017. https://themerkle.com/the-early-history-of-monero-in-500-words/.

[49] CryptoNote - the next generation cryptocurrency. https://cryptonote.org/.

[50] CryptoNote. CryptoNight Hash Function, 2013. https://cryptonote.org/cns/cns008.txt.

[51] Federal Information Processing Standards Publication 197, NIST. Announcing the Advanced Encryption Standard, 2001. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[52] Leslie Xu, Intel Corporation. Securing the Enterprise with Intel AES-NI, 2010. https://www.intel.com/content/dam/doc/white-paper/enterprise-security-aes-ni-white-paper.pdf.

[53] Keccak Team. https://keccak.team/keccak.html.

[54] Kudelski Security. The BLAKE2 Cryptographic Hash and Message Authentication Code, 2015. https://tools.ietf.org/html/rfc7693.

[55] Groestl Team. Hash function Groestl – SHA-3 candidate. http://www.groestl.info/.

[56] Hongjun Wu. Hash Function JH. http://www3.ntu.edu.sg/home/wuhj/research/jh/index.html.

[57] The Skein Hash Function Family. http://www.skein-hash.info/.

[58] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf.

[59] C. Percival, S. Josefsson. The scrypt Password-Based Key Derivation Function, 2016. https://tools.ietf.org/html/rfc7914.

[60] D. Eastlake 3rd , T. Hansen, Motorola Labs, AT&T Labs. RFC 4634 - US Secure Hash Algorithms (SHA and HMAC-SHA), July 2006. https://tools.ietf.org/html/rfc4634.

[61] B. Kaliski. Password-Based Cryptography Specification, 2000. https://tools.ietf.org/html/rfc2898.

[62] Khronos. OpenCL Overview - The Khronos Group Inc, 2018. https://www.khronos.org/opencl/.

[63] Mining hardware comparison, 2013. https://github.com/atmshop/litecoin/wiki/Mining-hardware-comparison.

[64] AdBlock Extension. https://getadblock.com/.

[65] Node.js. https://nodejs.org.

[66] MongoDB. https://www.mongodb.com/.

[67] W3C. Web Notifications, October 2015. https://www.w3.org/TR/notifications/.