Assessing the Threat of Web Worker Distributed Attacks

Yao Pan, Jules White
Dept. of Electrical Engineering & Computer Science
Vanderbilt University
Nashville, TN, USA
{yao.pan, jules.white}@vanderbilt.edu

Yu Sun
Dept. of Computer Science
California State Polytechnic University, Pomona
Pomona, USA
yusun@cpp.edu

***Abstract*—In this paper, we identify and evaluate the potential of new distributed attacks launched through web browsers using the HTML5 Web Workers API. Web worker attacks rely on the new multi-threading capabilities of Web Workers, which can allow malicious JavaScript code to run in the background of a web page without impacting foreground JavaScript performance or user experience. These background computing tasks can be used to launch application-layer DDoS attacks or offload computationally intensive attack tasks, such as password cracking, to the browsers of users visiting a compromised website. These attacks do not harm the compromised users directly but offer a potential path for attackers to gain control of large pools of computing resources, similar to botnets. We evaluate the feasibility of using online advertisement services to gain access to such computing pool and quantitatively evaluate the economics of these attacks and point out the key factors affecting the cost effectiveness of launching attacks through Web Workers in comparison with cloud computing or rented botnets.**

I. Introduction

As web browsers have become ubiquitous in our daily life, various attacks have been developed to target web browsers and web interactions. The majority of the existing research on web security and browser safety focuses on traditional attacks, such as trojans [1], malwares [2], phishing [3], and identity theft [4]. These attacks utilize vulnerabilities in the browsers or network protocols and threaten compromised users' property and privacy directly.

In this paper, we analyze a new potential attack vector – *browser-based distributed attacks based on background JavaScript Web Worker tasks*. These attacks are performed by embedding malicious JavaScript code in the webpages delivered to the users in order to gain access to background processing power on their machines. While users are surfing the Internet, these background JavaScript tasks allow the attackers to use victims' browsers to execute computational tasks. The key differentiator between these attacks and prior JavaScript attacks without Web Workers [5] is that they can run in the background and do substantial computational work without impacting foreground JavaScript performance or user experience [6].

What's new? Lam et al. [5] have considered the possibilities of misusing browsers for distributed attacks. However, the attacks assessed were limited to single-threaded web pages performing tasks that were not computationally intensive, such as worm propagation, click fraud, etc. Complex attacks, such as distributed password cracking, were not feasible before because of their significant impact on foreground JavaScript performance, which would alert users or make them leave the page, but are now possible due to three major JavaScript advances in recent years. 1) JavaScript used to be based on a single-threaded architecture and intensive computation would significantly slow down the page and draw users' attention. With the Web Workers API introduced in HTML5, computation can happen in a background thread. Our experiments have shown that Web Worker computation in the background has no discernible impact on the foreground user experience for a multicore CPU. 2) Popular browsers compete fiercely to optimize their JavaScript engines and there has been a significant increase in JavaScript computational speed. Modern browsers are able to complete complex and computationally intensive tasks such as 3D rendering and image processing [7]. Further, techniques such as asm.js [8] and NativeClient [9] can further improve JavaScript performance to near-native levels. 3) As a major UI evolution, tabs instead of windows in browsers, lead to parallel browsing behavior of users and web pages that are left open for long periods of time. Users tend to keep more tabs open and each tab open for longer, which gives background Web Worker tasks associated with a particular web page a longer period to operate. As we will show in our analysis, the combination of background computing tasks, significant improvements in JavaScript performance and long-lived tabs make Web Workers a potentially economically attractive attack vector for some attacks compared to cloud computing or rented botnets.

"Great Canon", a large-scale DDoS attack against Github [10] in 2015 has shown the immense potential that browser-based distributed attacks can tap. The JavaScript code in the analytics services of Baidu, China's largest search engine, were hijacked to include javaScript code that instructed visitor's browsers to send HTTP requests to a series of victim Github pages. The result was the largest DDoS attack in Github's history and caused Github to be intermittently unavailable for 5 days. The attack drew massive public attention as it only leveraged the browsers of unsuspecting website visitors and was not due to compromised machines or unpatched vulnerable application versions.

Open Question $\Rightarrow$ Are Web Workers a Significant

Threat Vector?

Kuppan [11] first proposed the idea of using Web Worker as a potential attack vector for DDoS attack. And several prototype browser-based distributed computational engines have been built including QMachine [12] and CrowdProcess [13]. However, as far as we know, none of the existing work has discussed the cost model of browser-based distributed computing attacks and whether or not they are a significant attack vector compared to botnets or other distributed attack approaches. If Web Workers are an economically viable attack vector for attackers, which types of attacks are threatening when launched from this type of platform?

To launch Web Worker attacks, a large amount of browsers need to visit a compromised webpage. One way of doing this is to hack a high-traffic website and inject malicious JavaScript code. However, popular websites are usually well-managed and protected by strict security measures, therefore not easy to compromise. Alternatively, attackers can create their own websites and propagate the sites through online advertisement services. Online advertisement is a mature business model on the Internet. Various Ad providers provide platforms for businesses to display banners or image Ads on websites to promote marketing messages. In this paper, we examine two types of online advertisement providers. The first representative is Google AdWords, which operates on a pay-per-click (PPC) model. Businesses will pay based on the number of clicks to their advertised websites. Another type of provider we examine is Hitleap [14], which allows you to exchange or purchase traffic to your website. The scary aspect of the online advertisement approach is that there is no need to exploit any vulnerabilities, the attacker can simply buy background Web Worker time via an Ad provider. As long as you have enough money, you can purchase a large Web Workers botnet for use.

In past work [6], cost models have been developed for legitimate use of Web Worker background computing, which is termed *Gray Computing*. In this paper, we enhance these cost models and apply them to better understand the attack economics of using Web Workers compared to rented botnets and other alternative attack vectors. Further, we analyze the practical aspects regarding connection limits, JavaScript performance, and other issues that impact Web Worker attack feasibility.

The remainder of this paper is organized as follows: Section II introduces the emerging trends that make Web Worker attacks possible; Section III describes the key research questions investigated in this paper; Section IV formalizes the cost model used for attack economic analysis; Section V presents an analysis of several potential attacks and answers the research questions for each attack; Section VIII presents concluding remarks and future work.

II. Motivation

In this section, we describe several factors that motivate our interest in Web Worker distributed attacks and how these factors make the Web Worker distributed attack threat more significant than in the past.

A. *Web Workers: Background Computing Threats?*

Web Workers are a new feature introduced in HTML5, which allows JavaScript to create separate threads in the background to process additional work. Web Workers are supported by all major browsers. A key new capability is that foreground tasks and user experience are not affected by computation happening in the background, as shown in prior work [6]. This ability to silently execute intensive computations in the background provides a foundation for attackers to unknowingly offload computing tasks to website visitors. A in-depth comparison of Web Worker and non Web Worker attacks in terms of influence on user experience and attack efficiency is provided in Section V.

B. *JavaScript Performance: New Computing Power for Attacks?*

JavaScript is not usually considered a language for high performance computing, based on early experiments and benchmarks [15]. However, as the Internet evolved and webpages became increasingly complex and interactive, there has been a growing demand for high-performance JavaScript engines to run tasks, such as 3D rendering, machine learning, or image processing. The performance of JavaScript is critical to the user experience on complex sites and directly affects page load and response time. Major browsers such as Chrome, Firefox, and Safari have made great efforts to optimize their JavaScript engines and have achieved significant performance boosts.

In programming languages benchmark game [16], the speed difference between JavaScript and C on various benchmarked computing tasks is at most 5X. Moreover, JavaScript shows absolute advantages over popular programming languages, such as PHP.

There are several projects aiming to further improve JavaScript performance to near-native levels. Emscripten [8] and NativeClient (NaCl) [9] are the most successful projects. Both Emscripten and NativeClient aim to allow code written in languages like C and C++ to be run in a browser. Emscripten compiles C/C++ code into JavaScript, and NativeClient compiles C/C++ code into an LLVM-like format which is then run in a special NativeClient runtime. As a consequence, Emscripten's generated code can run on all web browsers, since it is standard JavaScript, while NativeClient's generated code requires the NativeClient runtime to be installed (available in Google Chrome only). However, the advantage of NativeClient is it does run at near-native speed while Emscripten's code is close to, but generally slower than NativeClient.

TABLE I: Execution time comparison of native C++, native(handwritten) JS, and ported JS (from C++). JavaScript is running with Chrome 50.0.

Benchmark	C++	Native JS	Emscripten	NativeClient
Nbody	6.6s	28.1s	10.7s	7.8s
fasta	1.3s	11.4s	2.1s	1.6s

Table I shows a preliminary performance comparison of the ported JavaScript with native C++/JavaScript on Nbody (floating-point computation) and fasta(string and integer computation) tasks. To further examine JavaScript efficiency, we also benchmarked the distributed computing attacks we examine in this paper in Section V. The results confirm the conclusion above that ported JavaScript has comparable performance with native C/C++ code for some attack activities.

III. Research Questions

Research Question 1: Is the attack feasible via a Web Worker?

Research Question 2: How does the browser-based Web Worker attack compare to the non-browser-based attack?

Research Question 3: Are browser-based Web Worker attacks economically attractive to attackers?

IV. Cost Model

In this section, we describe the cost model for attacking with Web Workers, rented botnet and cloud computing. There are two types of costs associated with each attack: the acquisition cost, which is the cost to set up the attack infrastructure, and the maintenance cost, which is the cost for performing the attack. A comparison of the cost breakdown for different attack vectors is shown in Table II.

For attacks with cloud computing, there is no acquisition cost as you can purchase as many computing resources as possible from cloud service providers such as AWS or Azure. The maintenance cost for cloud computing would be the price the service providers charge you for actual usage.

For attacks with botnets, there are 2 ways to acquire a botnet. One can either grow their own botnet by exploiting vulnerabilities and propagating malwares or they can rent or buy a botnet in an underground market. It is hard to quantify the cost to grow a botnet as it depends on the expertise of the attacker and the vulnerabilities being used. But there is quantifiable pricing for renting a botnet that we can compare with. For maintaining a botnet for an attack, the attacker has to rent a VPN server to send command and control messages to the bots.

TABLE II: Cost Breakdown

Attack vector	Acquisition Cost	Maintenance Cost
Web Worker	Create own websites or hack high-traffic websites	Online ads costs (PPC or visiting time), VPN server for C&C, bandwidth costs
Botnet	Grow own botnet by exploits or rent botnet from underground market	VPN server for C&C, bandwidth costs
Cloud computing	No cost	Paid to service providers for usage

A Cost Model for Web Worker Attacks. Since hacking a website is difficult and the cost is hard to estimate, we focus on the cost analysis of using online advertisement to launch Web Worker attacks. The acquisition cost $AC_{browser}$ in Web Worker attacks is the cost of setting up a website (purchasing a VPN server and domain name). The maintenance cost $MC_{browser}$ is the cost of coordinating an attack with Web Workers. The maintenance cost consists of three primary parts: 1) the cost of the computing time to run a command and control server to coordinate an attack and 2) the cost of the data transfer to/from the command and control infrastructure 3) the cost to attract visitors through online advertisement services.

Because Web Worker attacks require the coordination of a larger numbers of clients to be effective, they place much higher demands on the command and control infrastructure that must be considered from a cost perspective. Take the distributed password cracking as an example, there has to be a coordinating server which is responsible for dividing the data into subtasks and distributing the data to browsers. The coordinating server needs to instruct the browser clients to work on different tasks and ensure that clients report results before they are terminated (*e.g.,* the user leaves the website). The cost to maintain a server for attack coordination C_{server} equals the unit price of the server multiplied by the time it is in use, which we express as $C_{server} = P_{server} * t$.

Bandwidth cost $C_{bandwidth}$ is the cost to distribute data from the attack coordination server. Take distributed rainbow table generation as an example, the coordinating server needs to assign the browser clients the range of plaintext hashes to compute, and the generated partial rainbow table from each client has to be sent back to server. The hashing algorithm, coded in JavaScript, also needs to be sent to the clients. These costs can be modeled as:

$$C_{bandwidth} = k * I * P_{toclient} + O * P_{toserver} + C_{code} \quad (1)$$

The actual data transfer to clients is almost always larger than the size of the inputs to the clients I because some clients will inevitably leave before their assigned computation is complete. Therefore the data transferred to these clients is wasted. We use μ to represent the successful task completion ratio. The actual volume of data transferred out will be k times the data needed for one copy of the task, where k is equal to $1/\mu$:

C_{code} is the bandwidth cost to distribute the processing code, such as hashing algorithms, to the browser clients. n is the total number of user visits to the website. During each visit, the client may request and complete multiple tasks but only one piece of processing code needs to be transferred. I_{code} is the size of the processing code in JavaScript. $P_{toclient}$ is the unit price of bandwidth cost to transfer data from server to client. $C_{code} = n * I_{code} * P_{toclient}$.

C_{ads} is the cost to attract visitors to attackers' websites through online advertisement services. Depends on the online Ad providers chosen, C_{ads} can be computed as $C_{ads} = n * C_{CPC}$ where C_{CPC} is the average cost per click. Or $C_{ads} = T_{visit} * C_{unit}$ for website like Hitleap where T_{visit} is the total visiting time and C_{unit} is the unit cost of buying the viewing time.

In summary, the cost to coordinate an attack with Web Workers is given by:

$$C_{browser} = C_{server} + C_{bandwidth} + C_{ads} \quad (2)$$

A Cost Model for Cloud-based Attacks. Another resource attackers can utilize to launch an attack is cloud computing services, such as Amazon EC2. Attackers can rent massive EC2 virtual machines for password cracking. Cloud computing provides elastic computing resources at competitive prices, as low as few cents per hour. There is no acquisition cost for cloud-based attack. The cost to run the tasks in the cloud is given by MC_{cloud}:

$$MC_{cloud} = T_{cloud} * I * P_{cloud} \tag{3}$$

where P_{cloud} is the cost per hour of a virtual compute unit. T_{cloud} is the computing time to process 1 unit of data with one virtual compute unit in the cloud.

A Cost Model for Botnet-based Attacks. The maintenance cost of botnet-based attack MC_{botnet} is similar to Web Worker attacks. However, the bots will not remain active forever. Attacks will lose control over some bots as time goes by due to the awareness of users or anti-virus scanning. We define s as the average percentage of bots lost each day. To maintain the botnet of same size, attacks will have to pay an additional cost $C_{loss} = s * P_{botnet} * t$.

$$MC_{botnet} = P_{server} * t + C_{bandwidth} + C_{loss} \tag{4}$$

The acquisition cost of botnet-based attack AC_{botnet} is the cost to purchase the botnet. $AC_{botnet} = P_{botnet} * n$. P_{botnet} is the unit price to rent or buy bot. n is the number of bots needed to launch an equivalent attack to Web Worker attacks.

V. Web Worker Attack Analyses

In this section, we describe three attack vectors that could be launched with Web Workers: distributed password cracking, cryptocurrency mining and DDoS attack. Research questions proposed in Section III are analyzed and answered for each attack.

A. Experiment Environment

In order to evaluate the attack economics quantitatively and to provide direct comparison between different attack approaches, we set up experiment environments consisting of browser clients and cloud instances to facilitate the computations and measurements needed for the cost models described in Section IV. For cloud computing, Amazon AWS was used as benchmark platform. For browser client, we used a desktop computer for our experiments with an Intel Core i5-3570 CPU clocked at 3.5GHz and 8GB of memory. For cloud instance, we used an Amazon EC2 m3.medium instance for CPU and g2.2xlarge for GPU to benchmark cloud data processing performance. As of April 2016, the price is \$0.067/Hr for m3.medium and \$0.65/Hr for g2.2xlarge. The system image used was Ubuntu 14.04 64bit.

B. Potential Attack 1: Distributed Password Cracking

Password cracking is a computationally intensive task. Cracking involves applying a hash algorithm to a candidate plaintext password and comparing the result to the ciphertext. Although the cracking time can often be insurmountable for a relatively strong password, there are always weak passwords exist that can be cracked in a reasonable amount of time.

TABLE III: Peak task distribution throughput and cost of different EC2 instance types.

EC2 Inst. Type	Throughput	Hourly Cost	Price/Throughput
m3.medium	3300 tasks/s	\$0.067/Hr	\$5.64E-9
m3.large	8100 tasks/s	\$0.133/Hr	\$4.56E-9

Distributed computing is an attractive approach to satisfy the computing demands that attackers need for password cracking. To distribute a brute-force cracking attack across a set of compromised hosts is quite simple. A range of the plaintext is sent to each client(e.g. 'aaaaa' to 'azzzz'). The client computes the hash of each plaintext and returns the ciphertext to the server to see if a match has been found.

Three cracking modes are examined in our experiment: SHA1, PBKDF2-SHA512 and rainbow table generation. PBKDF2 is a key derivation function that repeats certain cryptographic hash function many times on the input to make brute force cracking more difficult. We choose SHA512 and an iteration number of 1000 in our experiment. Rainbow Table are precomputed tables for reversing cryptographic hash functions. The idea is to use a time-space tradeoff and compute the hashes of all the possible plaintext within a length range and store the rainbow table chain.

1) Answering Research Question 1: Web Worker Password Cracking Feasibility: For non-browser based cracking, professional tools are used. John the Ripper (JtR) is one of the most famous password recovery program and was used as a benchmark in various of areas. JtR supports a variety of hash algorithms including MD5, SHA, etc. OclHashcat is a GPU-accelerated password recovery tool. We use these two tools to run on the Amazon EC2 to compare against the cracking speed of Web Workers in the browser. For Web Worker cracking, we ported the SHA1, PBKDF2-SHA512 and rainbow table generation algorithms from their C/C++ versions to JavaScript with Emscripten and NativeClient.

For distributed password cracking, a coordinating server is needed to dispatch tasks to all the clients. This server accepts requests from clients and returns the brief task information such as the task ID and task input data. Obviously, the more concurrent clients there are, the higher the throughput the server needs to support.

To assess the load produced by this command and control architecture and predict the cost of this infrastructure, we conducted an experiment to estimate the cost of hosting these task distribution servers in Amazon EC2. The task distribution server that we implemented exposes an HTTP API to handle HTTP GET requests and responds with a JSON-based string containing the task ID and input data for each task. Each client calls the API once before working on an individual cracking task.

The result in Table III shows that a moderate EC2 instance can handle thousands of concurrent command and control requests from Web Worker password cracking clients. An m3.large instance is used in the following cost analysis as the coordinating command and control server.

2) Answering Research Question 2: Comparison of Web Worker and Traditional Password Cracking: In this section, we compare the features of Web Worker and non-browser based platforms to launch the password cracking. A high-level comparison of the Web Worker and traditional attack variants can be found in Table VII. Although the individual computing capabilities of browser clients is lower than bots or cloud instances due to the limitations of JavaScript, the Web Worker attack can scale easily with the traffic of the websites at low additional cost. Therefore, the overall computing power of Web Worker attack is tremendous. Another advantage of Web Worker attacks over botnet-based attacks is the set-up cost. Because JavaScript is a platform neutral language and it can run in any hardware architectures and operating systems, the attackers can use same piece of JavaScript code for password cracking on any device. However, it would be much more difficult to do the same with botnet because of the heterogeneity of botnet composition: windows and Linux cannot share the same executable program (Interpreted languages such as Java work but are much slower, while C/C++ are platform dependent). Installing and configuring the cracking program on every bot will take tremendous efforts and make it hardly practical.

a) Comparison of Web Worker and non Web Worker JavaScript attack: To test whether there are differences in launching a password cracking attack in the background with Web Workers versus in the foreground without Web Workers, we conducted experiments evaluating the impact on user experience of these two ways of attack.

We used Tampermonkey, a Chrome plugin which allows users to install scripts that make on-the-fly changes to web page content. We injected computation tasks into popular websites and observed the influence on page responsiveness. We chose the auto-completion feature of search boxes to track page responsiveness because it is a pervasive and time-critical feature on many websites. Our goal was to test if the generation of search suggestions would be slowed down by the browser's participation in a password cracking computation.

Three scrips were written in Tampermonkey. The first script was the baseline which injected a search term into the search box and recorded the time needed for the web page's foreground to generate the search result HTML elements. The second script measured the performance and user impact of a Web Worker background task and created a Web Worker to perform cracking computation and also record the search suggestion generation time. The third script measured the performance and user experience impact of a traditional foreground JavaScript password cracking computation embedded in the head section of the HTML file and did not use Web Workers.

TABLE IV: Average search box suggestion generation time

Website	Baseline	Web Worker	Without Web Worker
Gmail	100%	104%	158%
Amazon	100%	103%	243%
YouTube	100%	100%	337%

TABLE V: Hashing rate with and without Web Workers.

	Web Worker	Without Web Worker
Chrome	0.61M hash/s	0.60M hash/s
Safari	0.47M hash/s	0.47M hash/s

50 different keywords were used as search inputs and for each keyword we ran 100 trials and averaged the generation times. As the results shown in Table IV illustrate, there was no discernible difference in the search suggestion load time when adding a Web Worker password cracking attack script. But the suggestion generation time significantly increased when the non Web Worker password cracking attack script was injected in the webpage. In terms of computation speed, there is no significant difference between the case of Web Worker and non-Web Worker, as shown in Table V. The results show the advantage of a Web Work based attack over a non-Web Worker based attack – there is no difference in computation speed but a decrease in visibility to the user.

Table VI shows the performance comparison of different browsers. Scripts ported with NativeClient are the fastest but only work under Chrome. For scripts ported with Emscripten, popular browsers such as Chrome, Firefox and Safari show similar performance while IE is more than 50% slower. Although mobile devices such as smartphones or tablets contribute increasing traffic to websites, online advertisement providers provide the options to restrict page view to desktops only. So we focus on desktop browsers only and leave the discussion of mobile devices performance in future work.

To make a more convincing estimation, we use the statistic of browser market share data obtained from StatCounter [17] and compute the weighted speed taking into account the performance variation of different browsers. As of March 2016, Chrome(60.1%), Firefox(15.7%), IE(13.7%), Safari(4.6%) are the most used desktop browsers. The weighted MD5 cracking speed is about 4.4M hash/s.

3) Answering Research Question 3: Web Worker Password Cracking Economic Analysis: To understand whether password cracking with Web Worker is economically attractive to attackers, we compare the cost of password cracking with Web Worker to cloud computing because cloud computing is considered cost-efficient and applicable to large-scale cracking tasks.

To evaluate attacking with Web Worker, we set up a website using WordPress software and embed cracking script in a Web Worker loaded in the main page of the website. We also set up a coordinating server in node.js to record performance information. We promote our website through both Google AdWords and Hitleap.

For Hitleap, 1 million minutes of viewing time can be bought for $625 [14]. We registered our website on Hitleap and used the bought traffic to make other users to view our website. These viewing time will be consumed by other users with a 20s viewing slot. We assume Hitleap can help us attract 100,000 minutes viewing time a day. The daily cost is $62.5.

For AdWords, we create a text ads including a URL of our

TABLE VI: Cracking speed for different browsers.

Browser	MD5	SHA1
Chrome (NativeClient)	6.0M/s	3.6M/s
Chrome (Emscripten)	2.78M/s	0.83M/s
Firefox (Emscripten)	3.1M/s	0.85M/s
Safari (Emscripten)	2.91M/s	0.67M/s
IE11.0 (Emscripten)	1.51M/s	0.09Ms

TABLE VII: Feature analysis of password cracking attack with browsers, botnet and cloud computing.

	Web Worker	Botnet-based	Cloud computing
Scale	Depends on the traffic of the websites	From hundreds to thousands	Purchased as needed, almost infinite
Speed	Low	Low, majority from underdeveloped countries	High
Set-up cost	Low, JavaScript runs everywhere	High, needs customized building for different environments	Low, homogenious system images

TABLE VIII: Password cracking speed (hash/sec) for different hash algorithms on different platforms.

Algorithm	JtR (m3.medium)	oclHashcat (g2.2xlarge)	Native JS	Emscripten JS	NativeClient
SHA1	6.1M/s	485M/s	0.16M/s	0.83M/s	3.6M/s
PBKDF2-SHA512	203/s	4512/s	112/s	123/s	152/s
Rainbow Table	2232/s	N/A	451/s	1503/s	1921/s

TABLE IX: Cost comparison for different hash algorithms on different platforms.

Algorithm	Web Worker	Cloud
SHA1	\$67/day	\$8.42/day
PBKDF2-SHA512	\$67/day	\$38.17/day
RainbowTable	\$67/day	\$975/day

website. The ads received 12000 impressions (views) and 29 clicks in 24 hours. The cost per click is around \$0.1. For those clicks, the average time people stay on our website is 81s. To attract equivalent 100,000 minutes viewing time as Hitleap, the daily cost will be \$7400. Obviously AdWords is not suitable for deploying this attack. To make AdWords economically attractive, we need to create very addictive websites that can hold visitors to stay more than 2 hours for each visit, which is extremely difficult.

To distribute brute force password cracking with Web Workers, the cost is $C_{browser} = C_{server} + C_{bandwidth} + C_{ads}$. For C_{server}, we choose an m3.large instance as coordinating server, $P_{server} = \$0.133/Hour$. For the brute-force attack, the only data that needs to be sent to clients is the start and end of the range of the plaintext, which is negligible. And the client only replies to the server with whether the password has been found. So the data volume transferred from clients to the server $I \approx 0$. For rainbow table generation, the generated table needs to be sent back to server. However, AWS does not charge for data volume transferred from the server to clients. So $P_{toserver} = 0$.

The hashing algorithm coded in JavaScript is 50KB. $I_{code} = 50$KB. The number of unique visits to the website is $n = 10^5 * 60/20 = 3 * 10^5$, which means the hashing algorithm needs to be transferred $3 * 10^5$ times from server to clients. $C_{bandwidth} = 3 * 10^5 * 50 * 10^{-6} GB * 0.09 = \$1.35$. The cost of running a Web Worker attack on this website per day is: $C_{browser} = 0.133 * 24h + 1.35 + 62.5 = \$67$.

To calculate the attack cost using rented cloud computing resources, we use Amazon's Elastic MapReduce Service (EMR) to construct a cluster with GPU instances because GPUs have a better performance/cost ratio than CPU instances as shown in Table VIII. The unit price for a GPU instance (g2.2xlarge) is $P_{unit} = 0.65 + 0.2 = \$0.85$. As we have discussed in Formula , not all the website visitors' browsing time can be utilized. The effective computing time should consider successful task completion ratio μ. Suppose $\mu = 0.8$, the effective computing time of the compromised website is $10^5/60 * 0.8$ hours. Using the cloud to compute the equivalent number of SHA1 hashes, the rented virtual machine time would be $1333 * 3.6/485 = 9.9h$. The cost of cracking SHA1 with cloud computing is $C_{cloud} = 0.85 * 9.9 = \$8.42$. C_{cloud} is \$38.17 for PBKDF2-SHA512 and \$975 for rainbow table generation. Table IX shows the cost comparison of Web Worker cracking and cloud computing cracking for different hash algorithms.

Some insights we can obtain from the calculations are: 1) Web Worker attacks are only cost effective than cloud computing for more complex hashing algorithms. This is understandable because the cost to distribute the computation is fixed. Complex hashing algorithms demand more computational resources and thus create more value to the attacker. 2). The cost savings of the Web Worker approach versus CPU instance in cloud computing is huge, but GPU instances in the cloud greatly narrows the gap. The inability to use GPU resources limits the attractiveness of the Web Worker attack variant. As GPU APIs for JavaScript become more prevalent, Web Worker attacks will become more attractive. 3). The Web Worker attack costs are directly affected by the traffic features of the website where the attack was deployed. Longer viewing times of website visitors produce a more stable computation pool and yields large cost savings for the attacker.

Figure 1 shows the relationship between average viewing time of a website and cost savings for a Web Worker password cracking attack. We use the costs of rainbow table for demonstration, for a given amount of computation, the cloud computing cost is fixed. The Web Worker computing cost drops as average page view time increases. We can see the Web Worker approach is actually not cost-effective for the attacker when the average dwell time on the website is below 20 seconds. This means websites with short visits are not ideal target for attacks with Web Workers. These websites still generate values for their computation but the opportunity cost is too large for the attacker. However, there are websites where users do linger longer, such as sites with videos that average several minutes in length, online gaming, online education, etc.

Web Worker attacks, if they become prevalent, will be biased against these types of websites.

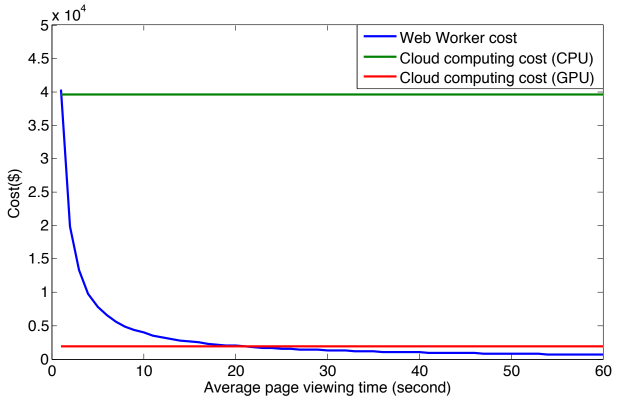

Fig. 1: The relationship between average viewing time of website and cost saving.

C. Potential Attack 2: Cryptocurrency Mining

Cryptocurrencies are digital currencies which use cryptography to secure the transactions and to control the creation of new units. Some of the well-known cryptocurrencies include Bitcoin, Litecoin, etc. Usually cryptocurrencies are mined by people who have installed and run clients such as cpuminer and cgminer on their computers in their spare time. However, the attackers could migrate the mining algorithms to JavaScript and embed them to regular webpages as Web Worker tasks. When visitors view the compromised websites, they became slaves mining for the attackers unwittingly and making profits for the attackers. In this section, we analyze how much profit Web Worker mining could produce for attackers.

1) Answering Research Question 1: Web Worker Cryptocurrency Mining Feasibility: We experimented with two cryptocurrencies: Bitcoin and Litecoin. They have the top 2 market capitalizations and represent two families of cryptography algorithms. The mining of Bitcoin relies on repeated SHA256 hashing while Litecoin uses the scrypt algorithm. For Web Worker mining with Bitcoin, we use BitcoinPlus, which is a web service that provides APIs for people to embed a JavaScript miner on their own websites. The JavaScript miner can connect to any mining pools via the HTTP stratum proxy using the account information attackers registered in the pool. In this way, the payout will be credited directly to the attackers. For Litecoin, the same framework was used. We ported the core algorithm 'scrypt' from C code to JavaScript with Emscripten and constructed our own JavaScript Litecoin miner.

2) Answering Research Question 3: Attack Economics Analysis: As of April 17, 2016, the difficulty factor of mining for Bitcoin is 178,678,307,672 and the price in the market is \$431 per coin. In our experiment, our JavaScript miner reaches a maximum speed of 0.5MHash/s, the time needed to generate a block (25 bitcoins) is $difficulty * 2^{32}/hashrate = 1.5 * 10^{15}$ seconds [18]. Suppose an attacker compromised a website of a million visits per day with an average visiting time of 20 minutes. $10^6 * 20 * 60$ seconds of computing power can be used each day. The attackers can earn only $10^6 * 1200/(1.5 * 10^{15}) * 25 * 431 = \$0.009$ a day on average.

For Litecoin, our JavaScript Litecoin miner on Chrome can run with a speed of 0.25KHash/s. Using the payout data from f2pool, one of the largest mining pools in the world, 0.00985851 LTC coin can be made for 1MHash/s per day. The price of Litecoin as of April 17,2016 is \$3.27. The attackers can earn $0.00985851 * 0.25 * 10^3 * 10^6 * 20 * 60/87600/10^6 * 3.27 = \$0.11$ a day. For both Bitcoin and Litecoin, the reward is extremely low even for a high traffic website.

In summary, it is not worthwhile to do Bitcoin or Litecoin mining attack with Web Workers. Partially because the increasing difficulty of mining and partially because the huge performance gap between the specialized mining hardware used by some mining operations and the commodity computers [19].

D. Potential Attack 3: DDoS attack

Distributed Denial-of-service (DDoS) attacks have been known for many years. Attackers usually inject slave or "bot" computers with remote command and control services. The compromised computers are used to send requests simultaneously to the target services. DDoS attacks have been the most prominent threat for website owners in recent years. Web Workers could be a serious new threat for DDoS attacks, since they do not rely on compromising a host, but instead getting people to simultaneously visit a specific web page.

1) Answering Research Question 1: Feasibility of a Web Worker DDoS Attack: There are various ways attackers can make client browsers send requests to target services. For example, attackers can embed a src address in an image tag: <img src="http://targetURL.com"/> pointing to the victim service. Or an attacker can send an AJAX GET or POST request from JavaScript. In the past, sending AJAX requests to external domains was impossible due to the restrictions of the "same-origin policy" enforced by most websites. The "same-origin policy" permits scripts running on a webpage to access the DOM of the same domain only. However, with the Cross Origin Resource Sharing (CORS) capabilities introduced with HTML5, scripts are allowed to make cross-origin AJAX requests to other domains. The attackers can send multiple HTTP GET or POST requests to any website in a background loop of a Web Worker. Although the request needs the website to return a HTTP header with 'Access-Control-Allow-Origin: *' for the request to fetch contents, the attackers are not really interested in reading the response. As long as the requests have been sent, the server needs to spend resources and create threads to handle incoming requests.

One concern for Web Worker DDoS attack is when sending the first request to the target URL, if the response does not contain the 'Access-Control-Allow-Origin' header with a "*" value, the browser will refuse to send subsequent requests to the this URL. However attackers can bypass this restriction by making every request unique (e.g. adding a random query-string parameter).

Another concern is CORS requests will leave an "Origin" header pointing to the website sending the CORS requests. The server being attacked can use rule-based filter to block requests with specific "Origin" header. However, in our experiment, we found only XMLHttpRequest will leave an "Origin" header while embedding URL in image source does not add "Origin" header in the request.

In our experiment, a Web Worker in a browser was able to send 10,000-16,000 HTTP requests per minute. The attack speed is primarily dependent on the network bandwidth and speed of the host displaying the attack web page. When multiplied by the concurrent visits of high traffic websites, the number of requests that can be produced is substantial.

2) Answering Research Question 2: Comparison of Web Worker and Traditional DDoS Attacks: One characteristic of Web Worker DDoS attacks is that the attacks happen in the application layer. Traditional DDoS attacks come in a number of forms. Some attacks happen in the network layer, such as UDP floods or SYN floods [20]. Some attacks utilize amplifications, such as DNS amplification attacks [21]. However, JavaScript cannot access the network layer, therefore only application layer attacks are possible to launch from Web Workers. The size of a Web Worker DDoS attack depends on the traffic attackers are able to draw. Using online advertisement service, theoretically very large amount of concurrent visitors can be attracted if no limit is put on cost. In comparison, a botnet typically has a few thousand bots and unique IP addresses.

Given the severe impact of DDoS attacks, many vendors provide DDoS protection and mitigation solutions to website operators. The solution usually consists of a set of techniques including blacklists, IP filtering, etc. The key is to differentiate legitimate human traffic from botnet traffic. The detection and mitigation of Web Worker DDoS attacks will be much more difficult than Botnet-based DDoS attacks because network flood attacks, such as UDP floods and SYN floods, can be resolved at the hardware level but application layer attacks of Web Worker come from real users with real IP addresses. It is even possible to simulate real application transactions through a series of HTTP requests. The attacker looks just like a legitimate visitor. Other features also make Web Worker DDoS attacks potentially harder to detect. Unlike a botnet with a fixed pool of IP addresses that can be blacklisted, the client composition of Web Worker DDoS are continually changing as visitors come and go from a compromised website. The transience also makes forensics harder because a user becomes a bot only when they are visiting a compromised website.

3) Answering Research Question 3: Economics Analysis of Web Worker DDoS Attacks: To understand whether DDoS attacks with Web Workers are economically attractive to attackers, we compare the cost of a Web Worker attack with rented botnet DDoS attacks. A recent trend in cybercriminal is the popularization of "attack as a service". People with little expertise of hacking can rent existing botnets from hackers in the black market. The price to rent botnets depends on factors such as the scale and quality of the botnet. For 1 hour of time with 1,000 bots, the price is $25 on average.

Google AdWords supports various Ad forms including text, image and even HTML5 format. We uploaded a legitimate html design file of an Ad. Inside the html file, we included a request to a target server that we set up for test. The difference to password cracking attack is the users does not need to click the Ad. The requests will be sent while users are viewing the website that displays the Ads. Because AdWords operates on a pay-per-click model, we only need to pay for the clicks but not the views. If the clickthrough rate (the number of clicks the ad receives divided by the number of times the ad is shown) is low enough, we will be able to get a large amount of pageviews while paying for a little money. Note that this is exact the opposite of what normal businesses would do, as they usually try to increase their clickthrough rate to attract more customers.

We promote our website through Google AdWords and Hitleap. For AdWords, we received 12000 impressions (views) and 29 clicks in 24 hours. The average cost per click is $0.1. The clickthrough rate (CTR) in our case is 0.24%. It is possible to further lower this rate by making your Ads less attractive and choose unrelated keywords. The average time people see our Ads is 32s. Assume a browser is able to send 10000 HTTP request a minute. If we have more budget and add more keywords for our Ads, we can scale up our impression. Suppose we were able to get 10^6 impressions a day. The average HTTP request rate will be $10^6 * 32/86400 * 10000 = 3.7 * 10^6/min$. The daily cost is $0.1 * 10^6 * 0.0024 =$ $240. For Hitleap, the price for buying traffic is $0.0375/hour [14]. The daily cost to generate $3.7 * 10^6$ HTTP request per minutes is $0.0375 * 370 * 24 =$ $333. Therefore, AdWords is more favorable than Hitleap for DDoS attack.

For botnet-based DDoS attack, a bot with a HTTP DDoS tool can send an average of 20,000 requests in a minute in our experiment. Assume the rented botnet has an online ratio of 50 percent, to generate the same number of requests with botnet, 370 bots are needed. In reality, the purchased botnet will gradually decrease in size due to the awareness of users or anti-virus scanning. An additional cost $C_{loss} = s * P_{botnet} * t$ should be considered. The average percentage of bots lost each day is estimated to be around 6.3% [22]. The cost for botnet-based DDoS C_{botnet} is therefore about $370/1000 * 25 * 24 * (1 + 0.063) =$ $236. The result shows Web Worker DDoS attack has comparable cost to botnet-based DDoS attack. The key factor determining the cost for Web Worker attack is the clickthrough rate of the Ads. There are potentials for attackers to adjust their Ad strategy to reduce the clickthrough rate to make Web Worker DDoS cost saving than botnet.

VI. Attack Limitation and Countermeasures

One possible countermeasure to Web Worker attacks is to enforce Content Security Policy (CSP). Content Security Policy is a new standard to prevent XSS attacks by defining a field in HTTP header so that a server can specify the domains that browsers are allowed to make requests to. If the server restricts the white-list domains to only itself, then a Web Worker will not be able to make DDoS requests or contact with a coordinating server for password cracking. However, very rare websites now actually implement this feature because websites are relying heavily on external resources such as images, Ads, or services. Maintaining a white list of legitimate domains requires tremendous efforts.

For Web Worker DDoS attacks, one way to mitigate would be to enforce more restricted limits on the browsers in terms of the number of concurrent requests, number of references to non-local objects, and sustained usage of concurrent requests over time. For example, it may make sense to send a burst of 100-200 requests to several different hosts when a web page first loads, but a sustained request rate of 10,000 requests per minute is not indicative of legitimate behavior. However, these restrictions need to be carefully balanced with the needs of legitimate websites where it is common for webpages to grab resources from different domains for displaying advertisements, accessing third-party APIs, etc.

VII. Related work

The web browser, as the gateway to the Internet, has been under the spotlight for attackers as well as security experts. There have been extensive studies around browser-based attacks such as browser interrogation, login detection, cross-site request forgery [23], [1]. The attack we assessed in this paper is also launched via browsers. However, the primary difference is that the attack does not exploit any vulnerabilities but simply utilizes the standard APIs within the browser and steals computing resources. It is necessary and inevitable for browsers to offload computing tasks to users. Therefore, this kind of attack is more hidden and difficult to prevent.

Some researchers have noticed the potential of browser-based distributed computing systems and applied Web Worker infrastructure to volunteer computing. QMachine [12] is a web service incorporating volunteer web browsers worldwide together for bioinformatics computing tasks. CrowdProcess [13] is a similar web service but aims for more general problems. Pellegrino et al. [24] present some preliminary analysis on browser-based DDoS. None of the existing work has discussed the cost model of browser-based distributed computing and whether there is really a monetary incentive for attackers to perform such attacks. And if yes, what are the factors that affect the profitability of the attack.

Botnets or zombie networks, are a major threat in network security. Various attacks can be made through botnets such as DDoS attacks [25], spam [26], phishing, or theft of confidential information [27]. Various methodologies and tools have been developed to measure and detect attacks initiated with botnets. Most detection techniques rely on horizontal correlation which is observing the behavioral similarities of multiple bots of the same botnet. Botsniffer [28] uses statistical algorithms to detect botnets based on crowd-like behaviors. BotMiner [29] proposes a detection framework to perform clustering on Command and Control (C&C) communication traffic.

VIII. Conclusion

In this paper, we evaluated the potential of a new attack, which uses browsers and Web Workers as a computing medium to launch botnet-like distributed attacks. The Web Worker attack is carried out by embedding malicious JavaScript code in a webpage delivered to normal website visitors. Based on our quantitative analysis of the cost of launching these attacks in comparison with cloud computing and rented botnet, we found that these Web Worker attacks can be economically feasible and have a number of properties that could be attractive to attackers building new botnet types.

References

[1] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, N. Modadugu *et al.*, "The ghost in the browser analysis of web-based malware," in *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, 2007, pp. 4–4.

[2] M. Johns, "On javascript malware and related threats," *Journal in Computer Virology*, vol. 4, no. 3, pp. 161–178, 2008. [Online]. Available: http://dx.doi.org/10.1007/s11416-007-0076-7

[3] S. Garera, N. Provos, M. Chew, and A. D. Rubin, "A framework for detection and measurement of phishing attacks," in *Proceedings of the 2007 ACM workshop on Recurring malcode*. ACM, 2007, pp. 1–8.

[4] T. Jim, N. Swamy, and M. Hicks, "Defeating script injection attacks with browser-enforced embedded policies," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 601–610.

[5] V. Lam, S. Antonatos, P. Akritidis, and K. G. Anagnostakis, "Puppetnets: misusing web browsers as a distributed attack infrastructure," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 221–234.

[6] Y. Pan, J. White, Y. Sun, and J. Gray, "Gray computing: An analysis of computing with background javascript tasks." in *International Conference on Software Engineering*, 2015.

[7] "Unreal engine 4 in firefox," https://blog.mozilla.org/blog/2014/03/12/mozilla-and-epic-preview-unreal-engine-4-running-in-firefox/.

[8] "Emscripten," http://kripken.github.io/emscripten-site/index.html.

[9] "Nativeclient," https://developer.chrome.com/native-client.

[10] "Ddos on github," http://arstechnica.com/security/2015/04/02/ddos-attacks-that-crippled-github-linked-to-great-firewall-of-china/.

[11] L. Kuppan, "Attacking with html5," *Presentation at Black Hat*, vol. 2010, 2010.

[12] S. R. Wilkinson and J. S. Almeida, "Qmachine: commodity supercomputing in web browsers," *BMC bioinformatics*, vol. 15, no. 1, p. 176, 2014.

[13] "Crowdprocess," https://crowdprocess.com/.

[14] "Hitleap," https://hitleap.com/.

[15] F. Smedberg, "Performance analysis of javascript," p. 145, 2010.

[16] "Computer language benchmarks game," http://benchmarksgame.alioth.debian.org/u32/benchmark.php?

[17] http://gs.statcounter.com/.

[18] "Bitcoin difficulty," https://en.bitcoin.it/wiki/Difficulty.

[19] "Bitcoin mining speed," https://en.bitcoin.it/wiki/Mining_hardware_comparison.

[20] H. Wang, D. Zhang, and K. G. Shin, "Detecting syn flooding attacks," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 2002, pp. 1530–1539.

[21] G. Kambourakis, T. Moschos, D. Geneiatakis, and S. Gritzalis, "Detecting dns amplification attacks," in *Critical Information Infrastructures Security*. Springer, 2008, pp. 185–196.

[22] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. J. Dietrich, and H. Bos, "Sok: P2pwned-modeling and evaluating the resilience of peer-to-peer botnets," in *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 2013, pp. 97–111.

[23] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 75–88.

[24] G. Pellegrino, C. Rossow, F. J. Ryba, T. C. Schmidt, and M. Wählisch, "Cashing out the great cannon? on browser-based ddos attacks and economics," in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, 2015.

[25] E. Alomari, S. Manickam, B. Gupta, S. Karuppayah, and R. Alfaris, "Botnet-based distributed denial of service (ddos) attacks on web servers: classification and art," *arXiv preprint arXiv:1208.0403*, 2012.

[26] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov, "Spamming botnets: signatures and characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 171–182, 2008.

[27] M. Bailey, E. Cooke, F. Jahanian, Y. Xu, and M. Karir, "A survey of botnet technology and defenses," in *Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*. IEEE, 2009, pp. 299–304.

[28] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," 2008.

[29] G. Gu, R. Perdisci, J. Zhang, W. Lee *et al.*, "Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection." in *USENIX Security Symposium*, 2008, pp. 139–154.