When Your Browser Becomes the Paper Boy
An Anonymous Browser Network

Juan D. Parra Rodriguez(✉), Eduard Brehm, and Joachim Posegga

University of Passau, Passau, Germany
{dp,eb,jp}@sec.uni-passau.de

Abstract. We present a scenario where browsers' network and computation capabilities are used by an attacker without the user's knowledge. For this kind of abuse, an attacker needs to trigger JavaScript code on the browser, e.g. through an advertisement. However, unlike other Web attacks, e.g. cross-site scripting, the attack can be executed isolated from the Origin of the site visited by the user.

We demonstrate this by forcing common browsers to join an overlay network and perform onion routing for other peers in the network. An attacker can create and tear down such browser networks whenever needed and use them to avoid detection, complicate forensic analysis, and protect his identity. Based on a performance evaluation with real browsers, we ascertain that the network delivers messages in a timely manner under load while remaining unnoticed. From a more constructive point of view, we discuss how the current CSP specification and other mechanisms under discussion can help to protect users against this attack.

Keywords: Web security · Browser abuse
Content Security Policy (CSP)

1 Motivation

The World Wide Web as a de facto standard for modern applications, along with the advent of APIs allowing access to resources on the client-side, e.g. WebWorkers, has pushed the browser to execute sophisticated applications. On the other hand, the Web security community is focused on preventing session- and data-related confidentiality and integrity breaches. Thus, attackers have started to abuse computational resources from the browser to perform malicious activities such as Denial of Service attacks against third parties [23,31], or CryptoJacking, i.e. performing crypto-currency mining without the user's consent [32].

Herein, we study a stealthy, yet resource consuming scenario in which an attacker instructs browsers to join a browser network performing onion routing following Tor's approach [19]. Although there is existent work bridging browsers

L. J. Janczewski and M. Kutyłowski (Eds.): SEC 2018, IFIP AICT 529, pp. 94–107, 2018.
https://doi.org/10.1007/978-3-319-99828-2_7

with the Tor network [22,34], we aim at a different direction. Instead of enabling users to willingly provide an entry point to Tor, we show that browsers can be used as peers, i.e. Web Onion Peers (WOP), to build a browser network used to relay encrypted packets between an attacker and other malicious actors.

Also, from the possible scenarios exemplifying browser resource abuse, onion routing is interesting for several reasons. First of all, the user's consent should be a prerequisite for joining an anonymization network; especially, because they do not obtain benefits in return. Further, hosting parts of an anonymity network can be problematic [20]. Tor's website advises against running exit nodes at home, as this can bring attention from authorities leading to equipment seizures [3].

From the attacker's perspective, the generation and destruction of WOP networks in an ad hoc manner makes browsers an instrumental part for an attacker avoiding detection and eavesdropping. Also, browser's churn makes the network volatile and therefore harder to analyze from a forensic point of view. Last but not least, users can suffer when limited resources are exhausted; for example, the processing and networking functionality used for the WOP network could affect battery lifetime (phones or laptops) as well as Internet quotas.

Our **contributions** can be summarized as follows. First of all, we explain the principle of the attack we encountered. Second, we evaluate the attack's scalability by analyzing our proof of concept implementations (centralized and decentralized) by measuring the time required to build circuits and time needed to deliver messages. Our evaluation confirms that the network delivers messages with acceptable delays (usually around 6 s with a maximum of 10 s under heavy load). Last but not least, we show how the Content Security Policy standard (CSP) and other mechanisms can help to protect users against the anonymous network.

This paper is organized as follows. We describe the attack in Sect. 2. Section 3 describes our proof of concept implementation, followed by the evaluation of the scalability, resource use and stealthiness of the network in Sects. 4 and 5. In Sect. 6, we discuss possible countermeasures. Finally, we cover related work in Sect. 7 and then draw conclusions from our research in Sect. 8.

2 The Attack

In a nutshell, an attacker capable of triggering JavaScript code can force the user's browser to join a WOP network to relay encrypted packets for his benefit, yet without making the user aware of this situation. Unlike common Web-based attacks, this kind of attack succeeds even when it is executed in a context completely isolated by the Same Origin Policy [36], e.g. loaded in an Iframe which executes in an isolated Origin.

Our **attacker model** considers an attacker who can host a server accessible from the internet, i.e. *the directory server* from Fig. 1. Moreover, the attacker needs to trigger the execution of JavaScript code while the user visits a site, i.e. *the affected site* shown in Fig. 1.

To **include the malicious scripts in a regular, benign, website** an attacker can use advertisement networks as it has been shown by research presentations [23] or as it has been observed in the Wild for CryptoJacking [35]. Also, modifying a famous JavaScript library could force a high number of browsers to join the network; a single CMS plugin compromised more than 3000 applications to do CryptoJacking recently [27]. Moreover, previous studies performed by Nikiforakis et al. [28] have shown how an attacker can execute malicious code, e.g. using stale IPs or domains that are still included but forgotten. Specifically, they found out that 56 domains used in 47 websites out of the *Alexa Top 10,000* were available for registration at the time. Thus, whoever obtains these domains, could automatically deliver JavaScript code to sites looking up resources on these stale inclusions. Furthermore, Nikiforakis et al. discovered inclusions pointing to IPs in the Intranet: an attack already discussed by Johns et al. too [24]. Last but not least, a more aggressive attacker could include the abusive script on several websites using cross-site scripting (XSS) vulnerabilities. Lekies et al. have shown that more than 6000 unique XSS vulnerabilities were present just on the 480 websites of the Alexa raking (9.6% of the *Alexa Top 5,000*) [25].

There are several **advantages** for an attacker establishing a WOP network. First of all, he can use other browsers to transport his messages while selecting a multi-hop path using onion routing. The onion routing approach ensures that even highly skilled users whose browsers forward encrypted packets cannot decrypt them. Further, traceability of this communication is very hard because paths established to send one message are likely to be destroyed after a short period of time, i.e. circa 5 min on average [16], due to churn produced by browsers joining and leaving the network. Further, whenever the attacker suspects the current network is under surveillance, he can tear it down and create another one to communicate with his allies over a different WOP network.

In spite of the advantages for an attacker, three **limitations**, mostly inherited by the Tor network's design, also apply. The first one is that, like Tor, the directory server is needed to establish connections; therefore, restricting access to a particular directory server limits the network. Obviously, our proof of concept could be enhanced to distribute the peer index over several servers, but this is not crucial to show the feasibility of the attack exposed in this paper. The second limitation is that, provided that someone is interested in analyzing the traffic of a given WOP network and has enough resources to control a high amount of WOP peers, it is possible to correlate source and destination of messages to individual browsers. The third limitation is that if the attacker's traffic is monitored, it is possible to statistically identify whether the communication is a WebRTC data channel [21]. Nonetheless, none of the limitations breaches the confidentiality of messages.

3 Proof of Concept

Our proof of concept requires a directory server to register all peers in the network, which can either forward messages using onion routing or send messages

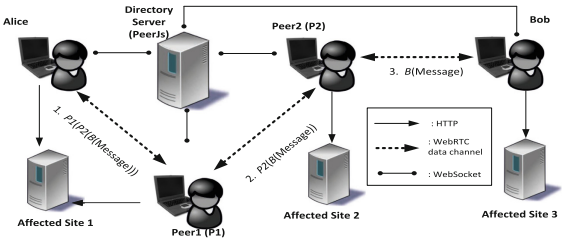

Fig. 1. Proof of concept diagram

to other peers over the WOP network. Data transferred between browsers is sent over a **WebRTC Data Channel**. For simplicity, we have extended the WebRTC signalling server (PeerJS [14]) to behave as a directory server and handle WOP peer registration and lookup to support circuit establishment. A *circuit* determines a list of peers over which the message is relayed. When a message is sent over the network, several layers of encryption are applied recursively. In this way, each peer along the circuit can only partially decrypt the packet and forward the rest of the packet to the next peer. This process is repeated until the encrypted packet reaches the final recipient. Every message is encrypted with end-to-end encryption with an ephemeral key for the recipient; therefore, eavesdropping the communication to learn the message is also impossible. To avoid bothering users and to ensure that heavy computation tasks run in the background all **the cryptographic functions are performed by WebWorkers**.

Section 4 evaluates the performance overhead introduced if a WOP peer chooses freely in comparison to receiving a predetermined circuit from the server. To this end, we have implemented both versions and from now they are termed *decentralized* and *centralized* WOP networks. This term is used because in the centralized case, the circuit paths are specified by a centralized entity, i.e. the directory server. In both cases, every peer can have at most 5 simultaneous WebRTC connections to other peers; thus, each peer can be relay packets for two circuits (4 connections) while sending or receiving messages over a circuit of its own (1 connection)[1].

4 Scalability Evaluation

Set-Up: As shown in Fig. 2, we used the Selenium Grid architecture in which a program orchestrating the whole experiment and browsers' behaviour, i.e. the automation program, interacts with a centralized Selenium component, i.e. the hub. The hub, acting upon requests from the automation program, sends commands to several Selenium nodes and tracks the current load distribution on the

[1] As Tor, we use Elliptic Curve Diffie-Hellman (ECDH) key agreement with a single shared key to encrypt the communication between every pair of peers. In Fig. 1, *P1(P2(B(Message)))*) represents the message encrypted with the keys from Bob, P2, and P1, in this particular order.

different Selenium nodes on the grid. In turn, each node spawns and controls browsers within the VM whenever instructed to do so by the hub. In addition to the Selenium-related programs required for the measurements, we have a Node.js HTTP server shown as the Data Collection Server in Fig. 2, which stores the delays measured by the WOP peers running in each browser.

To build the Selenium Grid, we used 10 VMs running on a VMware ESX Virtualization server. Each VM had 16 GB RAM, 8 cores, 15 GB hard drive and had a fresh installation of Lubuntu 16.4 LTS AMD 64. Selenium V2.47.1 was used either as a hub, or a node. From the 10 VMs, there was one *main VM* in charge of the orchestration of the Selenium Grid, building the WOP network and obtaining the information measured, i.e. delays. Additionally, the rest of the VMS, i.e. *node VMs*, just executed Firefox and a plain Selenium node.

The automation program is a Java program that generates one WebDriver object for each remote browser; however, as shown in Fig. 2, one Selenium node opens more than one browser instance during the experiments. To maximize the number of peers, we have decided to execute several tabs in each browser. This saves resources in comparison to running separate processes[2].

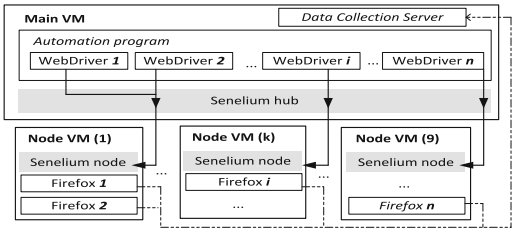

Fig. 2. Set-up overview for the scalability evaluation

In Selenium, the WebDriver must activate a particular tab in the graphical interface before executing the code therein; furthermore, the WebDriver remains blocked while it executes JavaScript code in a browser tab. To overcome performance limitations, we keep a "controller tab" active in the graphical interface at all times which receives instructions, i.e. build a circuit or send a message, for the other so-called "peer tabs". This allows the controller tab to push instructions to a queue (read by other tabs) and return. Thus, we avoid the overhead of switching active tabs; also, the WebDriver is not blocked by the controller tab.

To record information during the experiment, each peer tab reports data to the Data Collection Server shown in Fig. 2 directly. More specifically, before sending or after receiving a message, the participating peer sends the message identifier to the data collection server. A similar approach is followed before attempting a circuit establishment, and after the circuit has been established[3].

[2] A Firefox instance needs 400 MB of memory. Each tab needs 30 MB.

[3] The time required for the action is only based on the Data Collection server's clock.

We detect messages or circuit establishments that failed when the Data Collection server receives an event specifying an attempt with a particular identifier, yet without receiving a notification when the action succeeded.

Experiments: We have classified peers in two categories: active or passive. The former peers actively initiate new WOP circuits and send messages to other WOP peers in the network. The latter neither begins the establishment of new circuits, nor sends messages to other peers; nonetheless, passive peers can receive messages from other peers, they can be used as part of a circuit initiated by other peers, and they can also be used to relay packets in the WOP network. To introduce churn (for realism), passive peers rejoin with new keys every 3 min, i.e. 180 s. This value is realistic since average time spent on websites ranges between 293 and 310 s per visit [16]. Also, active peers follow instructions specified in a set of scripts that are picked randomly. These scripts instruct the active peers to build circuits with peers chosen randomly, send messages, wait for a given time between 0 and 5 s without performing the next instructions, leave and join with the previously existing keys as well as generate new keys and rejoin as completely different peers. Once a script finished executing, the active peer picks another one and starts again. All active circuits are lost when a peer disconnects or rejoins the network.

Our dependent variables, i.e. variables that we measure, are the delays required to establish a circuit or to send a message over the network. In each experiment, we measure the number of successful circuit establishments or message deliveries as well as the number of errors. On the other hand, the constant conditions applied to all the experiments, i.e. controlled variables, include the aforementioned set-up and peer behaviour. Also, every experiment we execute lasts for 30 min, and every message sent across the WOP network has to go through 3 hops before reaching its destination, i.e. 2 intermediate peers excluding the peer generating and receiving the message.

The main independent variables, i.e. parameters modified, are the kind of WOP network (centralized or decentralized), the number of WOP peers spawned for the network, and how many of the deployed peers are active. To determine the total number of WOP peers in the network, we conducted initial experiments to ensure that delays measured were induced by the WOP network implementation instead of the underlying hardware. To this end, we built a centralized WOP network with 50, 75, 100, 125, and 150 peers. In each scenario, we executed the experiment with 50% of active peers and analyzed the number of messages delayed, the number of errors and the message delay. Therein we observed that between 25 and 100 WOP peers, the network delivered more messages as the network size increased. Still, the number of successful messages decreased and the errors increased for 125 and 150 peers. As a consequence, we conclude the different experiments should be executed with *25, 50, 75 and 100 WOP peers.* Also, we chose to execute experiments with *25, 50, 75 and 100%* of active peers.

In total, without including the preliminary measurements, we present the result of 24 (12 for each kind of WOP network) experiments in this Section. All the dependent variables were measured for each experiment run. So, although

we present the circuit establishment and message delivery analysis in separate graphs they were both measured simultaneously.

Results: To present the result of all the experiments concisely, each figure in this section uses the left and the right y-axis. On the left-hand y-axis, the number of successful and failed events are plotted with two tones of grey. The total height of each bar represents the total of events, i.e. successful plus errors, while the right-hand y-axis shows on a white bar the average time taken for an action, i.e. build a circuit or send a message, along with its standard deviation.

To make the differences between each network set-up evident for each kind of network, the x-axis has three groups showing how many peers were executed, namely 50, 75 and 100; furthermore, each group contains 4 cases modifying the percentage of active peers between 25 and 100%.

The ***results for building circuits*** for the centralized and decentralized WOP networks are presented in Figs. 3 and 4 respectively. The success/failure rate, as well as the number of circuits established, is higher for the decentralized WOP network in comparison to the decentralized version. This happens because peers in the decentralized network pick peers for circuits randomly, therefore introducing delays in the circuit establishment with this trial and error approach. Also, the number of failures increases (decentralized) because the probability of a peer dropping the circuit due to a reconnection increases proportionally to the time required to build a circuit.

Regarding the time needed to establish a circuit, the average time, as well as the standard deviation, are very similar when 50, 75 or 100 peers are joining the network. This is not surprising because the time average only includes the circuits that are successfully built. Therefore, even though the centralized approach can produce more circuits, the time required to (successfully) establish each circuit should not change as the algorithm for the ephemeral key exchange is exactly the same. In the case of 100 peers, the centralized approach requires some more time to build the circuits, because the network builds more circuits successfully. The increased efficiency establishing paths for circuits imposes more encryption load on every single peer and requires more time.

Figures 5 and 6 show ***results obtained for sending messages*** between WOP peers in the centralized and decentralized networks. Due to the correlation between circuits and messages, the number of attempts to send messages over the network is lower for the decentralized WOP network. This is a direct consequence of the necessity of a circuit to attempt to send a message.

Inside each group according to the number of peers, e.g. 50, for the centralized network, the number of messages increases monotonically as the number of active peers increases. Nonetheless, for each group based on the number of peers, the number of messages for the decentralized network increases monotonically between 25, and 75% and then decreases for the case of 100% active peers. As in the previous case, this is a consequence of the need to have a circuit in order to send a message. In other words, increase or decrease of attempts to deliver messages in both networks is heavily influenced by the number of successful (only the light grey bar) circuit establishments for each experiment.

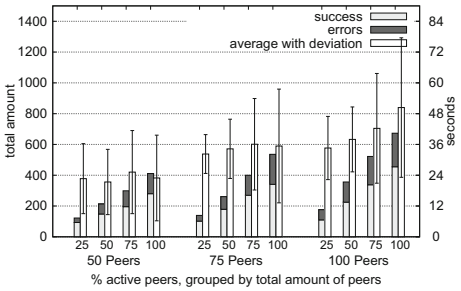

Fig. 3. Circuits and avg. time-centralized

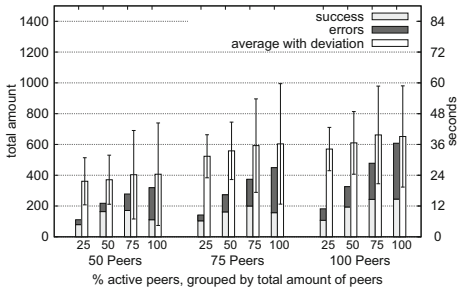

Fig. 4. Circuits and avg. time-decentralized

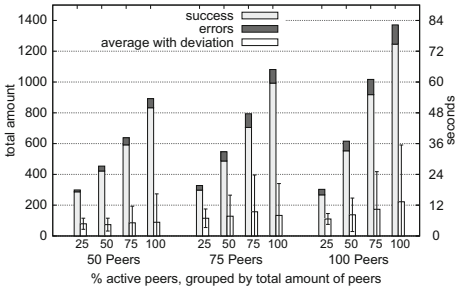

Fig. 5. Messages and avg. time-centralized

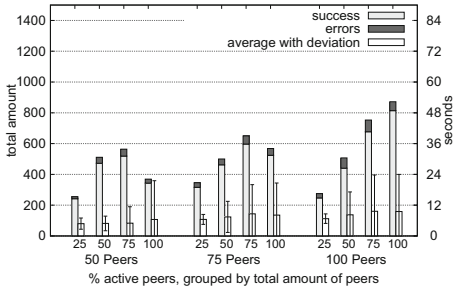

Fig. 6. Messages and avg. time-decentralized

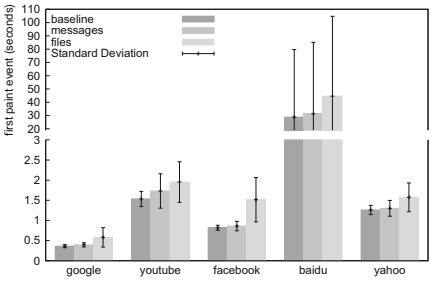

Fig. 7. First paint (Alexa top)-centralized

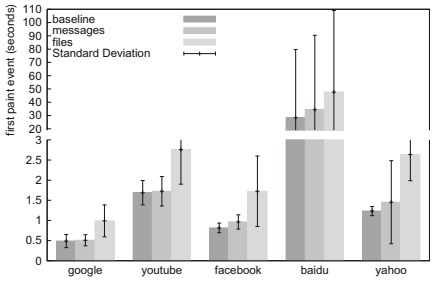

Fig. 8. First paint (Alexa top)-decentralized

All in all, the number of messages successfully delivered is high when compared with the number of failures for message distribution, i.e. light bar is considerably bigger than the dark grey bars on top of the light bars. Also, the message delivery has acceptable delays in both networks, even in the case of high load, i.e. time to deliver a message under high load lies between 4 and 12 s.

Overview: It can be observed that decentralizing the decision regarding how to build a circuit has non-negligible overhead regarding the WOP network's performance; however, an attacker can still achieve his goal without requiring a

decentralized version of the WOP network, since in case he hosts the directory server, there is no need to protect the path information from the directory server. Furthermore, considering that the experiments take 30 min, the centralized WOP network can build up to 15 circuits per minute in average, i.e. 450 connections divided by 30 min. Likewise, the centralized network can send 41 messages in average per minute, i.e. 1250 divided by 30 min. As a result, following a similar argumentation, our prototypical implementation relying on a relatively small directory server can comfortably relay messages using onion routing mechanisms between 15 concurrent users, although each one of them is sending 2.7 messages per minute in average.

5 Stealthiness, Resources and Network Conditions

Experiments described in this section were executed in a Dell XPS15 9550 with 16 GB RAM an i7-6700HQ (2.6 GHz Quadcore) processor.

We assess the **stealthiness** of the attack by measuring the time required until the first render event in Chrome. To achieve this, we execute an experiment 40 times in which a browser automated through the Chrome Debugging Protocol opens two tabs: a tab joining the network, and another tab opening one of the Alexa top 5 sites. The tab joining the network does so in three ways: as the baseline, sending messages, or sending a file. The baseline execution joins the network and does not perform any further actions. The second way is to send messages continuously, waiting 250 ms between messages. The third case is when the tab compresses and splits a 3.1 MB file into multiple chunks (each with 4096 Bytes) and sends them over the network, waiting 250 ms between chunks. To ensure that the first render is executed under heavy load in the last two scenarios and without load for the baseline, the tab joining the network is opened as soon as the browser starts; then, the browser automation program waits one minute (to ensure that circuit establishment is finished) and then opens one of the 5 top Alexa sites in a separate tab. Figures 7 and 8 show the time required for the first paint for all the cases and both networks. From these results, we conclude that the attack has a negligible impact on the user experience of tabs opened in the same browser.

We also measured the **impact of network conditions**, i.e. delay, on the WOP network, using the Selenium set-up in Fig. 2. We used the `tc-netem` command designed to add packet delays (to network interfaces) in the host machine. We executed 10 VMs with only one peer tab for each VM to ensure that tabs communicating with each other use the network interfaces. Each element of Table 1 shows the percentage of successful messages or circuits, and the time overhead, compared to a network without any delay, for the central(ized) and decentral(ized) network. We run the measurements with two common Internet latencies and a high latency: 50 ms, 100 ms and 150 ms respectively [1]. Table 1 shows that increasing the latency only affects marginally the number of circuits and messages successfully sent and the time required them.

Now, we cover the **resource consumption**, i.e. networking and electricity, of the attack. In this setup we executed all the network infrastructure and 3 peers in a laptop. Further, we executed a single browser with a single tab, which contains the first WOP peer of the circuit (the sender excluded) in the target machine, i.e. laptop being analyzed. We measured this particular peer because it uses the most computation and network resources. For the network use, we recorded traffic on the network interface of the target machine while the peer in the target machine was used as part of 10 circuits or while it was used to send 10 messages. After counting the number of IP packets and the bytes present on the captured file, using the python dpkt library, we found out that the target machine sends 23 KB and 130 IP packets per message in average, while each plaintext message had 102 bytes. Furthermore, it takes 70.2 KB and 24.920 IP packets to establish a circuit. The overhead is associated with the directory server, WOP encryption, browser protocol encryption, etc. For the power consumption, we used the same set-up as the network consumption, but instead of sending 10 messages we let a peer send messages constantly through the targe machine, waiting 250 ms between messages. Then, we measured the power consumption of the browser process (Chrome) in the target machine using `powertop` for 10 min. The power consumption for the browser running the intermediate peer in the network was 4.16 mW, which is almost twice of the regular consumption, i.e. 2.38 mW were consumed by a tab opening Google in the same set-up.

6 Countermeasure Discussion

CSP has received significant attention as an in-depth mechanism against script injection and data exfiltration, yet CSP supports developers in other ways, e.g. sand-boxing external resources. Now, we discuss how security-aware developers (or Web administrators) can leverage CSP to avoid the attack.

Script execution can be disabled by specifying a strict list of origins from which scripts can be executed. To this end, the `script-src` keyword, commonly used against XSS attacks, can be used. However, there is one caveat when using the `script-src` directive in real-life applications. Weissbacher et al. [38] have shown that implementing CSP on existing sites is a challenging and error-prone task. They also warned that less than 1% of the sites in the Alexa top 100 implemented CSP; further, at the time, the few of sites including CSP policies

Table 1. Performance of the network with delays on the links between VMs

	% Messages		% Circuits		% Time messages		% Time circuits	
Delay	Central	Decen	Central	Decentral	Central	Decentral	Central	Decentral
50 ms	98	98	97	90	105	103	127	104
100 ms	94	90	95	86	128	105	148	126
150 ms	91	51	86	86	135	147	161	131

were wrongly deployed and therefore not really protecting sites against potential attacks. More recently, Weichselbaum et al. [37] have conducted a renewed experimental analysis on how CSP policies are used. By the time they executed their experiments, 94% of policies can be bypassed and that many sites include insecure hosts. Furthermore, many policies include extremely relaxed wildcards, unsafe origin white-lists or directives such as `unsafe-inline` which allows inline scripting. However, a new implementation to propagate nonces over scripts using the `strict-dynamic` keyword attempts to solve these issues.

Also, the `media.peerconnection.enabled` Firefox property or the WebRTC Control Add-on for Chrome, Firefox and Opera [4] let browser's users block WebRTC connections. Restricting where WebRTC Data channels can connect to as part of CSP has been discussed [6–8]. If this is provided by CSP, developers could restrict the Origins with which WebRTC data channels can be established.

7 Related Work

Although there are several approaches to let users willingly use their browsers for anonymity [2,15,22,34], we explore how an attacker can use browsers to hide his identity, yet without making the browser's user aware of this. From now on, we focus our related work analysis on computational resource abuse from browsers.

From the network perspective, several cases where attacks directed to external network resources through browsers have been described. Johns et al. used JavaScript to access servers and hosts in the internal network, e.g. for fingerprinting purposes [24]. Antonatos et al. performed a theoretical study regarding the throughput of browsers when used to do port-scanning, Denial of Service (DoS) and worm propagation [11]. Athanasopoulos et al. demonstrated how a Facebook application can use a social network to deliver code forcing browser to launch a DoS attack against third-party servers [12]. Grossman et al. presented a how to use browsers to steer DoS attacks through advertisements in a real demonstration [23]. Additionally, Parra et al. showed how an attacker could perform a DoS attack against third-party websites with WebSockets and WebWorkers; also, as part of this work, a previously unknown vulnerability in Chrome was discovered [31]. Regarding abuse of CPU and GPU, browsers have been abused to calculate hashes [9,26] and mine crypto-currencies. In particular, since the beginning of Bitcoin, several JavaScript libraries were developed to monetize browsers from visitors [5]. Even though Bitcoin mining in browsers is no longer an option, new miners have emerged, such as Coinhive for monero [17]. In terms of storage abuse, an attack to fill the user's disk by Feross [10,13]. Parra et al. analyzed an extension of this approach where information is stored on the user's disk, and synchronized with other browsers, without the user's knowledge and analyzed possible CSP countermeasures [29,30]. To the extent of our knowledge, this is the first paper describing the attacker model, implementing a proof of concept and extensively evaluating the feasibility of a browser resource abuse attack where an attacker uses browsers as an anonymous routing network.

8 Conclusion

We describe and show the feasibility of a misuse scenario whereby browsers are forced to join an overlay network to perform onion routing for other peers without asking for the user's consent. Also, we discuss the power of CSP beyond XSS and ClickJacking attacks, and show that it can help against the attack described here. However, CSP has faced several barriers as part of its adoption process. From our analysis we conclude that one of the key aspects making the CSP adoption difficult is that websites' developers and administrators are not fully aware of the code executed by their sites. In particular with simple, yet popular libraries, developers obtain power over the open source software ecosystem[4]. A clear example reflecting browser abuse was observed by the Weather plugin removed from the WordPress marketplace due to CryptoJacking [27]. Thus, as future work, we propose that the Web security model could benefit from emphasizing an approach where the users are more involved in granting permissions per resource, e.g. as Android lets users take back specific API access at any time, or an approach to automatically detect malicious behavior, e.g. similar to machine learning solutions to classify malicious JavaScript code [18,33].

Acknowledgements. This research has been supported by the EU under the H2020 AGILE (Adaptive Gateways for dIverse muLtiple Environments), grant agreement number H2020-688088.

References

1. Global IP network averages. http://ipnetwork.bgtmo.ip.att.net/pws/global_network_avgs.html. Accessed 06 Apr 2018
2. iAnonym. http://www.ianonym.com/. Accessed 08 Jan 2018
3. Tips for running an exit node. https://blog.torproject.org/tips-running-exit-node. Accessed 15 Jan 2018
4. WebRTC Control. https://mybrowseraddon.com/webrtc-control.html. Accessed 06 Apr 2018
5. BitcoinPlus (2011). https://web.archive.org/web/20170103133312/http://www.bitcoinplus.com/miner/embeddable. Accessed 06 Apr 2018
6. WebRTC via 'connect-SRC'? September 2013. https://github.com/aghorler/WebRTC-Leak-Prevent. Accessed 06 Apr 2018
7. CSP for WebRTC, August 2014. https://lists.w3.org/Archives/Public/public-webappsec/2014Aug/0162.html. Accessed 06 Apr 2018
8. WebRTC RTCDataChannel can be used for exfiltration, June 2016. https://github.com/w3c/webappsec-csp/issues/92. Accessed 06 Apr 2018
9. Aboukhadijeh, F.: MD5-Password-Cracker.js (2013). https://github.com/feross/md5-password-cracker.js/. Accessed 06 Apr 2018
10. Aboukhadijeh, F.: The joys of HTML5: introducing the new HTML5 HardDisk-Filler API (2013). https://github.com/PeerCDN. Accessed 06 Apr 2018

[4] A developer broke most packages in npm by removing a left padding module: https://www.theregister.co.uk/2016/03/23/npm_left_pad_chaos/.

11. Antonatos, S., Akritidis, P., Lam, V.T., Anagnostakis, K.G.: Puppetnets: misusing web browsers as a distributed attack infrastructure. ACM Trans. Inf. Syst. Secur. **12**(2), 12 (2008)
12. Athanasopoulos, E., et al.: Antisocial networks: turning a social network into a botnet. In: Wu, T.-C., Lei, C.-L., Rijmen, V., Lee, D.-T. (eds.) ISC 2008. LNCS, vol. 5222, pp. 146–160. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85886-7_10
13. Web Code Weakness allows Data Dump on PCs (2008). http://www.bbc.com/news/technology-21628622. Accessed 06 Apr 2018
14. Bu, M., Zhang, E.: The PeerJS library (2012). https://github.com/peers/peerjs. Accessed 06 Apr 2018
15. Burgstaller, F., Derler, A., Kern, S., Schanner, G., Reiter, A.: Anonymous communication in the browser via onion-routing. In: 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, pp. 260–267, November 2015. https://doi.org/10.1109/3PGCIC.2015.22
16. ClickTales 2013 Web Analytics Benchmarks Report (2013). https://research.clicktale.com/web_analytics_benchmarks.html. Accessed 06 Apr 2018
17. Coinhive JS Crypto Miner (2017). https://coinhive.com/. Accessed 06 Apr 2018
18. Cova, M., Kruegel, C., Vigna, G.: Detection and Analysis of Drive-by-download Attacks and Malicious JavaScript Code. In: Proceedings of the 19th International Conference on World Wide Web, WWW 2010, pp. 281–290. ACM, Raileigh (2010). https://doi.org/10.1145/1772690.1772720
19. Dingledine, R., Mathewson, N., Syverson, P.: Tor: the second-generation onion router. In: Proceedings of the 13th Conference on USENIX Security Symposium-Volume 13, SSYM 2004, pp. 21–21. USENIX Association, Berkeley (2004). http://dl.acm.org/citation.cfm?id=1251375.1251396
20. Russian Tor Exit Node Operator Arrested (2017). https://www.deepdotweb.com/2017/05/01/russian-tor-exit-node-operator-arrested/. Accessed 06 Apr 2018
21. Fifield, D., Epner, M.G.: Fingerprint ability of WebRTC. Technical report, Cornell University Library (2016). http://arxiv.org/abs/1605.08805. Accessed 06 Apr 2018
22. Fifield, D., et al.: Evading censorship with browser-based proxies. In: Fischer-Hübner, S., Wright, M. (eds.) PETS 2012. LNCS, vol. 7384, pp. 239–258. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31680-7_13
23. Grossman, J., Johansen, M.: Million Browser Botnet (2013). https://www.blackhat.com/us-13/briefings.html. Accessed 06 Apr 2018
24. Johns, M., Winter, J.: Protecting the intranet against "JavaScript Malware" and related attacks. In: Hämmerli, B.M., Sommer, R. (eds.) DIMVA 2007. LNCS, vol. 4579, pp. 40–59. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73614-1_3. https://web.sec.uni-passau.de/members/martin/docs/2007_DIMVA_Johns_Winter_Anti_JS_Malware_lncs.pdf
25. Lekies, S., Stock, B., Johns, M.: 25 million flows later: large-scale detection of DOM-based XSS. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS 2013, pp. 1193–1204. ACM, New York (2013). https://doi.org/10.1145/2508859.2516703
26. Matthews, N.: Ravan: JavaScript Distributed Computing System (BETA) (2012). http://www.andlabs.org/tools/ravan.html. Accessed 06 Apr 2018
27. Maunder, M.: WordPress plugin banned for crypto mining. https://www.wordfence.com/blog/2017/11/wordpress-plugin-banned-crypto-mining/. Accessed 15 Jan 2018

28. Nikiforakis, N., et al.: You are what you include: large-scale evaluation of remote JavaScript inclusions. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS 2012, pp. 736–747. ACM, New York (2012). https://doi.org/10.1145/2382196.2382274
29. Parra Rodriguez, J.D., Posegga, J.: CSP & Co. Can save us from a rogue cross-origin storage browser network! But for how long? In: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, CODASPY 2018, pp. 170–172. ACM, New York (2018). https://doi.org/10.1145/3176258.3176951
30. Parra Rodriguez, J.D., Posegga, J.: Local storage on steroids: abusing web browsers for hidden content storage and distribution. In: Proceedings of the 14th International Conference on Security and Privacy in Communication Networks: SecureComm. Springer International Publishing (2018, to appear soon)
31. Rodriguez, J.D.P., Posegga, J.: Why web servers should fear their clients. In: Thuraisingham, B., Wang, X.F., Yegneswaran, V. (eds.) SecureComm 2015. LNICST, vol. 164, pp. 401–417. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-28865-9_22
32. Report, B.: Cryptojacking: 2017 Year-End Review. https://badpackets.net/cryptojacking-2017-year-end-review/. Accessed 15 Jan 2018
33. Rieck, K., Krueger, T., Dewald, A.: Cujo: efficient detection and prevention of drive-by-download attacks. In: Proceedings of the 26th Annual Computer Security Applications Conference, ACSAC 2010, pp. 31–39. ACM, Austin (2010)
34. Snowflake (2016). https://trac.torproject.org/projects/tor/wiki/doc/Snowflake. Accessed 06 Apr 2018
35. Telegraph, T.: YouTube shuts down hidden cryptojacking adverts. http://www.telegraph.co.uk/technology/2018/01/29/youtube-shuts-hidden-crypto-jacking-adverts/. Accessed 15 Jan 2018
36. W3C: Same origin policy. https://www.w3.org/Security/wiki/Same_Origin_Policy. Accessed 20 Mar 2018
37. Weichselbaum, L., Spagnuolo, M., Lekies, S., Janc, A.: CSP Is Dead, Long LiveCSP! On the insecurity of whitelists and the future of content security policy. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS 2016, pp. 1376–1387. ACM, New York (2016). https://doi.org/10.1145/2976749.2978363
38. Weissbacher, M., Lauinger, T., Robertson, W.: Why is CSP failing? Trends and challenges in CSP adoption. In: Stavrou, A., Bos, H., Portokalidis, G. (eds.) RAID 2014. LNCS, vol. 8688, pp. 212–233. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11379-1_11