ANALYZING GOOGLE CHROME EXTENSION BOTNET EXPLOITS

SELENA FENG, LINDA GONG, HELEN HE

Abstract. Botnets are an increasingly popular way of exploiting Google Chrome Extensions. There are two main phases to executing a botnet via Chrome Extensions. The first phase is user adoption and the second phase is botnet execution. In this paper, we analyze a variety of botnet exploit and assign them a relative level of ease, depending on their phase. Finally, we aggregate our work to determine the level of ease that Chrome itself can be exploited, recommending different types of solutions. Our hope is to increase user awareness of the dangers in popular browser product Google Chrome.

1. Introduction

Google Chrome is quickly becoming the web's most popular browser, boasting about 64.92% of the world's internet traffic. [bro]. Chrome is also a very powerful browser; it's power is closer to that of an operating system, offering a wide variety of user engagement opportunities, including the option of browser extensions which creates additional browser functionality for users. These extensions are surprisingly powerful and can have a great variety of privileges and permissions. With the increasing popularity and power of Google Chrome extensions and the widespread usage of Chrome, botnet exploits have become more and more prevalent. Botnets need two things: an army of users and computation power per user. Given Google Chrome's popularity, it is a natural vehicle to aid the spread of the botnet program. Chrome's computing power also provides a great resource to power botnets. Hence, the general roadmap that malicious botnet Chrome extensions follow is to first get widespread user adoption of the extension, avoid user datection, then use the obtained computing power to accomplish various, often malicious tasks.

Some well-known cases of botnet exploits include Digmine, FacexWorm and Nigelify [BS] [Che] [RS]. Digmine was a cryptocurrency mining botnet that spread through Facebook Messenger via an executable video file, completely bypassing the Google Chrome Web Store. The executable modified Google Chrome configurations and downloaded the malicious extension into Chrome, finally utilizing Chrome's resources to both spread itself further and mine cryptocurrency. FacexWorm operated very similarly by social engineering its userbase via Facebook and conduct a variety of malicious cryptocurrency mining exploits. Nigelthorn, while also a cryptocurrency mining botnet, spread via internet virality. The original extension that the malware piggybacked off, Nigelify, replaced pictures in Chrome with a picture of Nigel Thornburry. Nigelthorn abuses Nigelify by running an executable that modifies the

existing, legitimate extension's code to include malicious JavaScript that mines for cryptocurrency.

Of course, not all existent botnets mine for cryptocurrency. Many perform other attacks, including DDoS services for sale, email spam campaigns, and user credential farming. In this paper, we will be analyzing various flavors of botnet exploits. We separate these exploits into two phases: phase 1 - user adoption, phase 2 - botnet execution. Within each phase, we explore a variety of attacks, assigning each a relative level of ease. The relative levels of ease are inversely proportional to the complexity of the attack. Our methodology will involve analyzing Chrome Extension infrastructure, designing conceptual attacks, and looking at past/existing botnet extension attacks.

2. Chrome Extension Architecture

Most browser extensions follow the Javascript Engine (JSE) model where extensions consist of a set of scripts, usually written in Javascript. Chrome extensions, in particular, consist of three components - the manifest file, event pages, and content scripts. There can also be additional files, e.g., HTML files or style sheets, as long as they are declared in the manifest file. The manifest file specifies which permissions and URLs the extension has access to [PH18]. Event pages are the intermediaries between the manifest file and the content scripts. Event pages access browser contexts using APIs and do not have access to the DOM (Document Object Model, a programming interface that defines the structure of the objects in an HTML or XML document). Content scripts are Javascript files that are injected onto webpages and have unrestricted access to the DOM. Figure 1 shows a diagram of the architecture of Chrome Extensions.

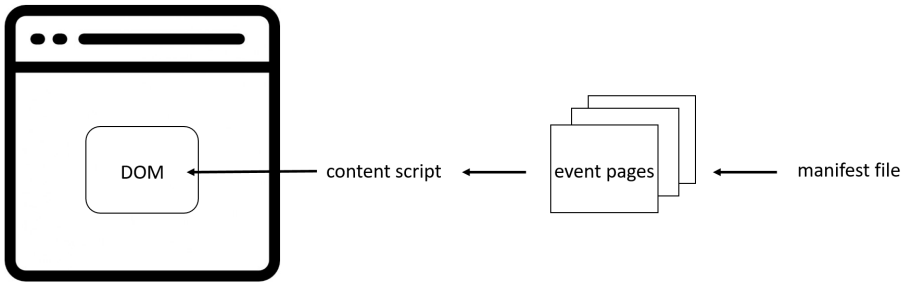

Figure 1. A diagram of the architecture of Chrome Extensions.

2.1. Permissions.

Without even explicitly requesting or stating in the manifest, Chrome Extensions have a powerful number of privileges. Chrome Extensions are, by default, allowed to run background scripts, make XMLHttpRequests, do pageActions (run a script depending on the user's actions on a page), do browserActions (adding

icons and popups in the toolbar), override certain webpages with custom HTML, and message pass from the content script to the parent extension, and autoupdate extensions. [chr] While a developer can't do too much harm without more explicit permisisons, you can certainly write a malicious extension that communicates with a malicious server and causes mischief for the user, e.g. overriding a user's homepage.

As mentioned before, the Chrome Extension manifest file contains the explicit permissions which can give the extension quite a lot of power and cause it to be quite harmful. Figure 2 displays all the possible Chrome Extension permissions and highlights some particularly alarming ones.

```
"activeTab", "alarms", "background", "bookmarks", "browsingData",
"certificateProvider", "clipboardRead", "clipboardWrite", "contentSettings",
"contextMenus", "cookies", "debugger", "declarativeContent",
"declarativeNetRequest", "declarativeWebRequest", "desktopCapture", "displaySource",
"dns", "documentScan", "downloads", "enterprise.deviceAttributes",
"enterprise.hardwarePlatform", "enterprise.platformKeys", "experimental",
"fileBrowserHandler", "fileSystemProvider", "fontSettings", "gcm", "geolocation",
"history", "identity", "idle", "idltest", "management", "nativeMessaging",
"networking.config", "notifications", "pageCapture", "platformKeys", "power",
"printerProvider", "privacy", "processes", "proxy", "sessions", "signedInDevices",
"storage", "system.cpu", "system.display", "system.memory", "system.storage",
"tabCapture", "tabs", "topSites", "tts", "ttsEngine", "unlimitedStorage",
"vpnProvider", "wallpaper", "webNavigation", "webRequest", "webRequestBlocking"
```

FIGURE 2. All possible Chrome Extension permissions. [ext] We have highlighted some permissions that we found particularly alarming.

For example, if an extension has access to the `webRequest` permission, it is able to read user web requests and make web requests on the user's behalf. This is a particularly dangerous permission if we consider the fact that many user passwords are sent via plaintext in a webRequest. Furthermore, the extension is allowed to read users' private messages to each other. Since the extension is already allowed to communicate with outside servers, this allow the extension to farm for user credentials. Another way that an extension can fake user credentials is through the `cookies` permission, which allows the extension to login as the user on various social media websites without even going through the trouble of phishing for user credentials. This could be an easy way for an extension to social engineer more users. Another disturbing permission is the `activeTab` permission which gives the extension almost complete control over the current active tab of the user. This means the extension could block the user's access to the Chrome Extensions management page, which would prevent users from deleting the extension. Indeed, even benign Chrome extensions, such as StayFocusd, which is a productivity extension, is able to block access to the extensions management page.

3. Evaluation Methodology

As mentioned before [1], we will analyze extensions through designing conceptual attacks and examining known botnet extension attacks. We have separated botnet

attacks into two phases: Phase 1 is user adoption and Phase 2 is botnet execution. For each of these phases, we have come up with a list of criteria by which we judge the ease of attack. For each criteria, there are three levels: easy, medium, and hard.

Phase 1: User Adoption Criteria

(1) How many steps a user needs to take to download the malicious extension
(2) Detectability of extension during the downloading process
(3) Speed at which the extension spreads from user to user (an R_0 of sorts)
(4) How easily will extension with exploit get published into Chrome Store

Phase 2: Botnet Execution

(1) How many steps the user (i.e. the one who downloaded the extension) needs to take for exploit to succeed
(2) Detectability of extension during exploit
(3) How frequently can attack be done on user
(4) How many Chrome Extension permissions does exploit require?
(5) How easily extension with exploit will get published into Chrome Store

The final ease of the exploit will be determined by the aggregation of its criteria. We chose the criteria for Phase 1 based on general principles by which determine how easily something spreads. We chose the criteria for Phase 2 based on the best ways we thought of to quantify the complexity of an extension.

4. User Discovery

Perhaps the largest challenge present to malicious actors is to successfully obtain a large victim user base. Malware authors use a number of tactics to acquire users and keep the compromised extension in their browser.

4.1. **Initial Installation.** The lowest-effort way to acquire a large user base is to buy an extension from a legitimate developer. There are a number of examples where a malicious entity purchases an extension to serve adware to the thousands of users who already have the extension installed such as YouTube Queue, Particle for YouTube, Typewriter Sounds, and Twitch Mini Player [Cimb]. Extension developers who don't outright sell their software may sometimes fall victim to phishing attacks of their Chrome Web Store credentials such as in 2017 when over one million browsers were affected by the popular Web Developer extension being compromised [Ped].

Other attackers may choose to build an extension and put it on the Chrome store. While Google does claim to vet all code before it is published, many malicious extensions make it through the initial vetting process - just this February, researchers announced more than 500 malicious extensions on the Chrome Web Store linked to a single malvertising campaign which had been installed by millions of users. One popular attack vector for extension installation is to instruct users to install a Chrome extension so they can view some sort of content, for example a video on a fake YouTube page. Once an extension has been installed, it can phish the social media cookies of its users to log in and send links to the user's friends and acquaintances [RS].

If the user has acquired malware on their machine through some other attack vector like a trojan, the malware can also continuously attempt to install a malicious

Chrome extension in the background [BS]. Even though Google disables non-Web Store extensions in sufficiently recent versions of Chrome, there are still ways to get around this by enabling developer mode or creating registry policies.

4.2. **User Retention.** Depending on the type of exploit, it can be incredibly difficult for the user to notice a compromise has occurred. With the correct permissions, the extension could block access to the extension management tab and make it difficult for the user to get rid of anything installed. The Nigelthorn malware used this tactic (code displayed in Figure 3) to prevent users from uninstalling [RS].

```
if(tab.url.protocol == "chrome:" && tab.url.host == "extensions"){
  blocked(tab);
}
```

FIGURE 3. Blocking access to extension management

With access to the history permission, an extension could selectively delete portions of a user's browsing history to cover up its tracks. Without any visible evidence of activities their browser has been used for, it is unlikely a user would even realize there is a malicious extension installed. While a user could theoretically inspect all outgoing packets, that would require a level of technical effort that we deem unrealistic.

Extensions which exploit social media posts to reach a wider audience can also block the user from URLs which would delete or edit these posts. While this sort of retention tactic is more heavy-handed and noticeable by the user, it may work well on targets who don't realize a browser extension is the source of their issues.

If there is supporting malware in the victim's computer, even after removing the offending Chrome extension the malware may continuously re-install it. In addition, the malware could exploit Google's registry policies to block uninstallation and create additional admin profiles [Abr].

5. BOTNET EXECUTION EXPLOITS

A botnet is a collection of compromised internet-connected devices that allow a botmaster to carry out certain attacks on a large scale. In this section we explore some common attacks and evaluate them with respect to our methodology [3].

5.1. **Cookie Stuffing.** Cookie stuffing is an attack where the browser extension adds information to the user's cookies. There were two Chrome extensions that used cookie stuffing to add a parameter to the user's cookies when they visited certain websites; this addition allowed the attackers to "earn a commission from any payments users made on the sites". [Cima] These extensions masqueraded as ad blocker extensions with similar names to reputable ad blockers, i.e., AdBlock from AdBlock, Inc (versus AdBlock from AdBlock) and uBlock by Charlie Lee, which caused many users to install these extensions (AdBlock had over 800,000 installs and uBlock had over 850,000). However, the extensions stopped working if the user accessed Chrome Developer Tools, and Google removed these extensions from the Chrome Store in September 2019.

Here are the evaluation results.

- How many steps the user needs to take for exploit to succeed: **1** (just need to download it from the Chrome store)
- Detectability of extension during exploit: **Difficult** (the extensions actually do block ads, since they were based on the actual AdBlock code, and the names were very misleading; in addition, users rarely examine their cookies or are notified about the contents of the cookies, which makes cookie stuffing difficult for users to detect)
- How frequently can attack be done on user: **Frequently** (doesn't take much CPU processing power to just add a parameter to a cookie)
- How many Chrome Extension permissions does exploit require?: **2** (activeTab, chrome.cookies)
- How easily extension with exploit will get published into Chrome Store: **Easy** (the extensions look like legitimate ad blockers)

5.2. **Phishing.** Another type of attack that can be carried out using Chrome Extensions is phishing attacks, especially ones based on iframes. It has been shown that it is easy to carry out an iframe phishing attack that substitutes a legitimate HTTPS website's webpage with the attacker's webpage without changing the URL of the actual webpage or changing the lock icon in the browser address bar that indicates the security of the page. [PH18] Attackers were actually able to substitute the Facebook login page with a malicious Phishbook page, as shown in Figure 4, while keeping the Facebook URL; this makes it difficult for users to detect that the webpage they are looking at is illegitimate, which makes it easy for attackers to phish sensitive user information.

FIGURE 4. Example of an iframe based phishing attack using Google Chrome Extensions. The Facebook URL is uncompromised, as is the lock icon in the address bar. [PH18]

Another example of phishing attacks using Chrome Extensions include phishing credit card information by injecting phishing scripts that create an iframe pop-up on a legitimate webpage that asks users for their credit card details. [Gat] Once users submit their information to the form, the information is sent to an "exfiltration server" hosted on a Russian server. [Gat] An example of an iframe pop-up that was used can be seen in Figure 5.

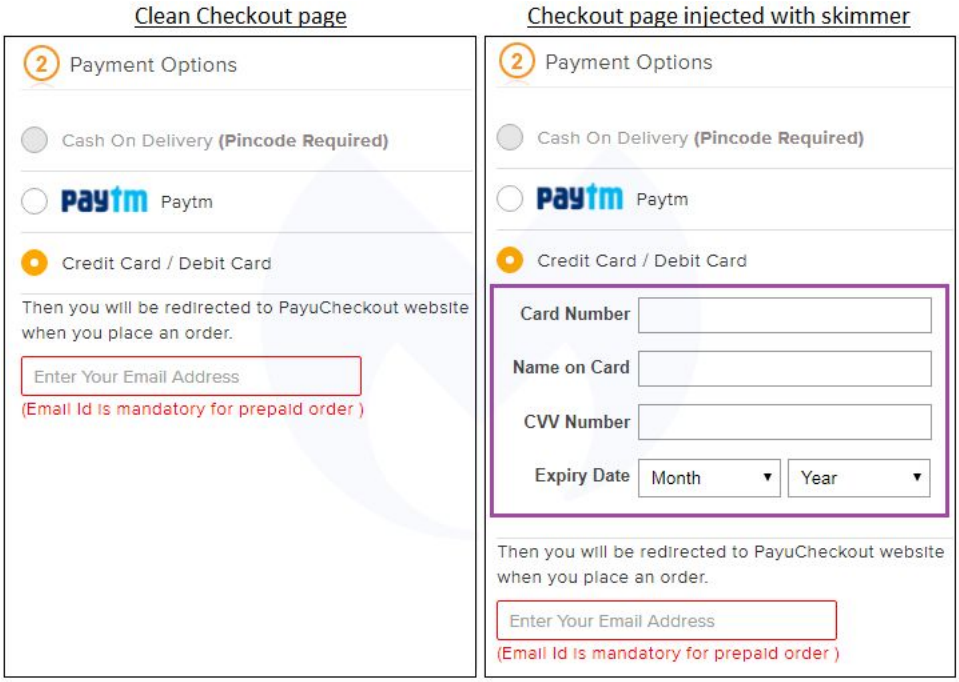

FIGURE 5. Example of an iframe pop-up that tries to phish credit card information. [Gat]

Here are the evaluation results.

- How many steps the user needs to take for exploit to succeed: **1** (just need to download it from the Chrome store)
- Detectability of extension during exploit: **Moderate to difficult** (from examples - legitimate URL and lock icon are uncompromised; however, in the credit card phishing attack, if the user notices that the payment is supposed to show up on a new page, they can avoid the attack)
- How frequently can the attack be done on user: **Frequently** (doesn't take much CPU processing power to have an iframe pop-up that just sends a POST request once a user enters some information)
- How many Chrome Extension permissions does exploit require?: **2-5** (activeTab, webRequest, optional: contextMenus, system.display, chrome.identity)

- How easily extension with exploit will get published into Chrome Store: **Easy to medium** (iframes generally look harmless/normal)

5.3. **DDoS.** DDoS (Distributed Denial of Service) attacks make machines or network resources unavailable to users by flooding a target with Internet traffic. For example, an extension could obtain the URL of the webpage the user is currently using, and then send a flood of HTTP requests to the user and prevent the user from using their browser normally. [LZC] This is quite easy to implement and does not require many permissions; however, is a bit abnormal in that it does not seem to lead to any obvious financial gain for most attackers (unless they directly benefit from disrupting traffic to a particular service).

Some botmasters sell large botnets to buyers who wish to attack other services, so the command and control (C&C) servers need some way of communicating a target to the extension because it would likely not be hard-coded into the extension itself. In many botnets, there are complicated dynamic naming protocols which obfuscate which server is actually giving commands making it difficult for an outside attacker to take down the botnet by targeting C&C servers. However, Chrome extensions have an included `update_url` field which allows for the browser to query some third-party host for updates to the extension - this is a natural channel for C&C servers to send new instructions to the bots.

Here are our evaluation results.

- How many steps the user needs to take for exploit to succeed: **1** (just need to download it from the Chrome store)
- Detectability of extension during exploit: **Moderate to difficult** (making HTTP requests is very cheap for web clients, so this attack doesn't significantly slow down or otherwise impact a user's machine)
- How frequently can attack be done on user: **Frequently** (doesn't take much CPU processing power to spam HTTP requests)
- How many Chrome Extension permissions does exploit require?: **2** (activeTab, webRequest)
- How easily extension with exploit will get published into Chrome Store: **Medium** (the DDoS attack can probably be hidden in the code of a legitimate extension, but is not guaranteed to be unnoticeable - in order to receive a target from the CC server, there would need to be some communication with the server itself)

5.4. **Cryptocurrency Mining.** Google Chrome extensions can also be exploited to mine cryptocurrencies. One example is the cryptocurrency mining bot called Digmine that spread using Facebook Messenger.[BS] The extension would send users a video file that was actually an AutoIt script that would send links to the victims' network via Messenger. The malware would then install a miner module on the user's machine, which would then use the botnet to mine cryptocurrency. FacexWorm was another malicious extension; similar to Digmine, FacexWorm used Facebook Messenger to propagate and install mining modules, but FacexWorm also tried to "hijack transactions in trading platforms and web wallets by replacing the recipient address with the attacker's" [Che].

While mining is traditionally a noticeably CPU-heavy activity, the extension could restrict mining to only-at-night, or even only while the CPU is under a certain threshold if there was additional malware on the machine.

Here are our evaluation results.

- How many steps the user needs to take for exploit to succeed: **2-3** (depends on the attack)
- Detectability of extension during exploit: **Moderate** (it's difficult for the user to figure out that they've been added to a botnet, but they could notice suspiciously high CPU usage compared to the past)
- How frequently attack can be done on user: **Infrequently** (this is a trade-off with detectability, as the user would likely notice if their computer was suddenly mining 24/7)
- How many Chrome extension permissions does exploit require: **2+** (activeTab, webRequest, and potentially others, depending on the nature of the attack)
- How easily extension with exploit will get published into Chrome Store: **Medium to hard** (Chrome banned bitcoin-mining extensions as of June 2019, so any malware which installed a mining module would have to be well hidden)

5.5. **Email Spamming.** Email spamming is when attackers send spam emails to victims, often using botnets that receive spamming commands from the attacker. The bots can send the emails at various rates; a high rate will cause more short-term damage, while sending sporadic emails will make detection more difficult. An example of an email spamming attack using Google Chrome Extensions is to access the DOM of the webpage of websites with email services. When a user composes an email, the attacker can access the editing area that the user is writing the email in; when the email is sent, the email contents are saved in the DOM, so the attacker can add some spam content to the email contents before the email is sent. [LZC] Here are our evaluation results.

- How many steps the user needs to take to download the extensions: **1** (just need to download it from the Chrome store)
- User detectability: **Moderate** (the user won't be directly notified of the change in email contents, but the email can be traced back to the user from the receiver)
- How often it can be done: **Frequently** (doesn't take much CPU processing power to add some lines to an email)
- How many permissions it requires: **2** (activeTab, webRequest)
- How easily it'll get published into the Chrome store: **Medium** (the spam attack can probably be hidden in the code of a legitimate extension, but is not guaranteed to be unnoticeable)

6. Results and Recommendations

We summarize the analyses across attacks in the table below.

Attack	Steps	Detectability	Frequency	Permissions	Publishability
Cookie Stuffing	1	Difficult	Frequent	2	Easy
Phishing	1	Moderate/Difficult	Frequent	2-5	Easy/Medium
DDoS	1	Moderate/Difficult	Frequent	2	Medium
Cryptomining	2-3	Moderate	Infrequent	2+	Medium/Hard
Email Spamming	1	Moderate	Frequent	2	Medium

We find that the large majority of attacks only require the user to install an extension, after which it is fairly difficult to detect the exploit. They often only require a few key permissions, and the majority of required code often looks typical to what a normal extension of that purpose would have, making it no guarantee that a quick skim of the code will cause it to be flagged as malicious. Our conclusion is that for many attacks it is very feasible to get a malicious Chrome extension into the Web Store, either through an existing legitimate extension or published directly by the bad actor, and it is moderately difficult for users to notice malicious actions once the extension has been installed.

We recommend that users be very careful about which extensions they install, especially if it claims to need the `activeTab`/`tabs` and `webRequest` permissions. Legitimate extensions can be turned into malware vectors, so even if a user only installs extensions from well-regarded services or developers, that doesn't necessarily mean they are safe.

7. Future Work

Computer security is always an ongoing battle. There are always more creative exploits and more creative ways to combat these exploits. Our paper reviewed some of the most popular botnet exploits, but there are certainly others that we have not touched upon. Chrome extensions can also be malicious without the constraint that they are botnets. While we have shown the level of ease at which Chrome can be exploited, there are still more things that developers should be cognizant of while developing products to stave off exploitation. We would also be interested in investigating other Chromium-based browsers such as Opera or Brave, as they support Chrome extension installation but have their own policies with regards to browser security.

References

[Abr] Lawrence Abrams. Chrome saying it's managed by your organization may indicate malware. https://www.bleepingcomputer.com/news/software/chrome-saying-its-managed-by-your-organization-may-indicate-malware/. Last Accessed: 2020-5-11.

[bro] Browser market share worldwide. https://gs.statcounter.com/browser-market-share#monthly-201910-201910-bar. Last Accessed: 2020-5-11.

[BS] Lenart Bermejo and Hsiao-Yu Shih. Digmine cryptocurrency miner spreading via facebook messenger. https://blog.trendmicro.com/trendlabs-security-intelligence/digmine-cryptocurrency-miner-spreading-via-facebook-messenger/. Last Accessed: 2020-5-11.

[Che] Joseph C. Chen. Facexworm targets cryptocurrency trading platforms, abuses facebook messenger for propagation. https://blog.trendmicro.com/trendlabs-security-intelligence/facexworm-targets-cryptocurrency-trading-platforms-abuses-facebook-messenger-for-propagation/. Last Accessed: 2020-5-11.

[chr] Chrome developer guide. https://developer.chrome.com/extensions/devguide. Last Accessed: 2020-5-11.

[Cima] Catalan Cimpanu. Google removes two chrome ad blocker extensions caught 'cookie stuffing'. https://www.zdnet.com/article/google-removes-two-chrome-ad-blocker-extensions-caught-cookie-stuffing/. Last Accessed: 2020-5-11.

[Cimb] Catalin Cimpanu. Chrome extension caught hijacking users' search engine results. https://www.zdnet.com/article/chrome-extension-caught-hijacking-users-search-engine-results/. Last Accessed: 2020-5-11.

[ext] chrome.permissions. https://developer.chrome.com/extensions/declare_permissions. Last Accessed: 2020-5-11.

[Gat] Sergiu Gatlan. Hackers steal payment card data using rogue iframe phishing. https://www.bleepingcomputer.com/news/security/hackers-steal-payment-card-data-using-rogue-iframe-phishing/. Last Accessed: 2020-5-11.

[LZC] L. Liu, X. Zhang, and S. Chen. Botnet with browser extensions. *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, pages 1089–1094.

[Ped] Chris Pederick. Web developer for chrome compromised. https://chrispederick.com/blog/2017/08/03/web-developer-for-chrome-compromised/. Last Accessed: 2020-5-11.

[PH18] Raffaello Perrotta and Feng Hao. Botnet in the browser: Understanding threats caused by malicious browser extensions. *IEEE Security Privacy*, 16:66–81, 2018.

[RS] Adi Raff and Yuval Shapira. Nigelthorn malware abuses chrome extensions to cryptomine and steal data. https://blog.radware.com/security/2018/05/nigelthorn-malware-abuses-chrome-extensions/. Last Accessed: 2020-5-11.