# Digital Forensic Analysis of Web-Browser Based Attacks

**Sally M. Mohamed**[1], **Nashwa Abdelbaki**[1], **and Ahmed F. Shosha**[1]
[1]School of Communication and Information Technology, Cairo, Egypt

**Abstract**— *In recent years, attacks that target browsers' vulnerabilities have increased significantly. An innocent user may lure to access untrusted website and malicious content passively downloaded and executed by his/her web browser; this attack vector known as, Drive-by-Download attack. Systems and security researchers addressed this attack from different perspectives. Several techniques and tools were introduced to detect and prevent Drive-by-Download attack; however, few research addresses the browser forensics perspectives to (1) identify traces (2) reconstruct the executed events of a downloaded malicious content, to assist the digital forensic investigation process. In this paper, digital forensic method is introduced to investigate a web browser subject to Drive-by-Download attack. A Proof-of-Concept implementation based on Firefox browser-extension was developed to inspect and analyze malicious URLs that host malicious executable. The developed system was tested using 55 malicious web pages and successfully identified the digital evidence of the attack. 77% of the identified evidence were artifacts that we believe it could assist forensic investigator to determine if web-browser or a system subject to examination is compromised or not, and the indications of compromises.*

**Keywords:** Web Browser forensics, JavaScript based attack, Drive-By-Download, Malware, Postmortem Analysis

## 1. Introduction

Nowadays, users and corporates are more and more connected to the web. A user can access his sensitive business/non-business applications using a web-browser. The web browsers, however, are complex software that developed using various technologies and have to process different file formats and contents that may be vulnerable or contain malicious code. On the other hand, cybercriminals understand that the user is the weakest link in the security chain, and a higher possibility to a successful attack. That's why attackers are trying to exploit vulnerabilities in web-browsers or luring users to visit malicious websites. In a typical Drive-by-Download attack, an innocent user is redirected to malicious web page, commonly denoted as (landing site[1]) [1]. This page contains code (often written in JavaScript[2]), that exploits a browser's vulnerability; browser's installed plug-ins or insecurely designed APIs. If succeeded, the exploits will download a malware from a malicious site[3] into the victim's machine. Usually a Drive-by-Download attack is developed for a specific vulnerability in a specific browser's version, so a common initial activity in this attack vector is reconnaissance and fingerprinting the web-browser metadata. An embedded script will attempt to collect information about the browser type, version, language, installed plug-ins and the installed operating system. Based on the collected information a malicious shell code will download the appropriate exploit or it may behave in a completely benign manner if, i.e. an analysis environment detected [2].

In order to understand the anatomy of the attack, the infected machine's web browser has to be forensically examined. *The Browser Forensics* is an emerging topic of the digital forensic science that refers to the process of extracting and analyzing the web-browsers artifacts and the user's browsing activities for forensic investigation purposes [3]. It is a technology-dependent domain that focuses on the most popular and commonly used web-browsers, i.e. Chrome browser developed by Google, Mozilla's Firefox browser and Internet Explorer by Microsoft, Safari browser developed by Apple Inc; some less commonly used browsers may also be considered, such as Text-Based web-browser i.e. Lynx Viewer. These browsers store a significant amount of data about the user's activities over the Internet if it's used in its normal mode and less data may also be collected if user opt to browse in the private browsing mode [6][7]. The data provided by the web-browsers are not exactly the same [8]. In order to collect these data, one has to consider the following:

- How a web-browser stores data and in which format?
- What are the minimum basic information that can be found in each browser?
- What are the additional and relative information provided by each browser?

All the major browsers would contain information about the browsing history, web applications' cache, web cookies,

[1]**Landing site:** page contains the shellcode (small binary payload) that will exploit vulnerability in the user's browser/plug-in.

[2]**JavaScript:** is a portable language(once written, it can be executed on any browser with JavaScript support). JavaScript attracts both developers and attackers by its dynamic and flexible features, that's why it's so popular and most Drive-by-Download attacks are implemented using it.

[3]**Malicious site:** page contains the malware downloaded by the shellcode. Often the browser will be targeted by a chain of redirection operations before getting to the malicious site.

user bookmarks, form completion data, stored passwords and many more. For example, Mozilla's Firefox and Google's chrome use a SQLite database to store these data, while Microsoft's IE uses files (like index.dat, cache and cookies files) to store it. Additionally, a considerable amount of data can also be found in the installed browser's extensions[4].

Although these browsers store a lot of data about the user's activities over the Internet, still a digital forensic investigation process is required to reconstruct the browser activities (i.e. executed code from a URL, downloaded resources, etc) resulted after accessing a malicious URL. To reconstruct the attack executed events and analyze its actions, we developed a system using Mozilla Debugger API to monitor, log and debug the details of an executed malicious JavaScript code subject to investigation. Output data after analyzing a list of digital forensics evidence are produced based on the following categorization:

- **Volatile Evidences:** non long lasting digital evidences that are residual in a system CPU, processor caches, and/or system memory.
- **Non-volatile Evidences:** digital evidences that are residual on the file system.

In summary, the contributions of this paper are defined as follows:

1) Digital forensic analysis methodology is proposed to allow forensic examination of Web-browsers artifacts, mostly enabled by executing malicious JavaScript code embedded in a malicious Web page.
2) A Proof-of-Concept Implementation of a system based on Firefox browser extension that outputs a digital forensic trace file for the executed JavaScript code. That includes, a detailed information about the volatile and non-volatile forensic evidences resulted from the malicious JavaScript code execution.

The remainder of this paper is structured as follows: Section Two presents the related work. Section Three introduces the proposed forensic analysis methodology. Section Four presents the preliminary analysis and results. Finally, Section Five concludes the paper and defines the possible future work.

## 2. Related Work

A Drive-by-Download attack could be defined as malicious content downloaded to a user's system without his/her consent using the web-browser. This content may be in different file format, i.e. it can be a malicious Flash file or embedded action script code [9], malicious pdf with embedded JavaScript code [10] or obfuscated JavaScript code in a web page [11] that exploits a vulnerability in the user's system. These downloads can be triggered by different actions, i.e. opening, scrolling or hovering a mouse cursor over a malicious web page or iframe. Academic and professional researches are commonly focusing on the detection and prevention techniques of this attack vector. The currently proposed techniques are mainly based on either analyzing the properties of a malicious web page URL [1] or analyzing the JavaScript code (or any other present code) contained in the web page using a (1) static, (2) dynamic, or (3) a combination of the both, aka, hybrid analysis, as follows:

- **Static analysis:** it uses a set of predefined features to determine that a malicious pattern or code exists in particular web page without code execution; several machine learning techniques and approaches may also be integrated in the static analysis to (1) define the set of features required for the analysis, (2) cluster, classify, and/or determine malicious web pages out of benign web pages [4] [12]. In this analysis approach, a low processing overhead may be required; however, the static analysis generally can be impeded if an obfuscation and/or encryption methods are employed [13].
- **Dynamic/semi dynamic analysis:** It uses a controlled environment, commonly called Sandboxing, to execute a subset or all of the possible execution paths for the embedded code to detect the presence of a malicious behavior. In this approach, a malicious code can be monitored accurately through the execution process. However, additional processing resources may be required. Attackers are also developing their malicious code to detect the analysis environment and to execute a legitimate code to mislead a human analyst or suppresses the execution and attempts to self-delete the code [4] [14].
- **Hybrid analysis:** It uses a combination of both static and dynamic analysis for the embedded JavaScript code to detect Drive-by-Download attacks, and to avoid the drawbacks associated with each approach. Typically, a static analysis technique is used as an initial filter to define the web pages that require a dynamic analysis, i.e. to determine the code that may defeat the dynamic analysis. Applying a hybrid analysis may guarantee accurate detection with minimum resources [4][15].

Other researches focus on analyzing exploit kits that are used to launch a Drive-by-Download attack [5]. *Exploit kit* is a malicious toolkit that exploits security flaws found in software applications. Using an exploit kit requires no proficiency or software development background, and it is equipped with different detection-avoidance methods. This justifies the notable wide-usage of exploit-kits not only by skilled cybercriminal, but also, by a non-skilled malicious users to achieve their purposes. In [20], the authors focuses on the server side part of a Drive-by-Download attack. They analyzed the source code for multiple exploit kits using Pexy, which is a system for bypassing the fingerprinting of an

[4] **A browser extension** is a program that extends the browser functionality and adds extra features to it.

exploit kit and get all of its possible exploits by extracting a list of possible URL parameters and user agents that can be used.

In a recent study presented in [16], the authors propose a system using Chrome JavaScript Debugger to detect browser's extensions that inject malicious ads into a web page. The study revealed that 24% of ad networks domain bring malicious ads. These ads will redirect the user to a landing page, which will finally download a malware/malicious executable into the user's machine. On the forensics side, researches alike [6][7][17] focus on private/portable browsing and how to collect/find the remaining evidences from the memory and the file system. In [18], the authors talked about the importance of making an integrated analysis for different browsers at the same time to understand what happened. They also propose a tool for constructing a timeline for the user's activities. There are a lot of commercial/free browser forensics tools, that give investigators an insight into user's browsing history but none of them deals with the browser memory.

# 3. The Proposed Digital Forensics Methodology

In this section, we propose our digital forensic methodology to investigate a malicious web page that is suspected to download and further execute malicious code using a web browser in a system subject to investigation. A typical scenario would be: a user notices uncommon activities occur in his system such as suspicious web advertisements appeared while surfing the web. Admins in a corporate network system may notice unusual network traffic inbound or outbound from the compromised system or visiting a web server known to host malicious contents. In these cases, a forensic analyst would perform an examination to the system to determine indication of a compromise, such as searching for a URL to malicious web pages in web-browsing history, a cookie file or a temp file in the Internet storage directory. If identified, it is crucial for the forensic investigation to determine what other resources had been downloaded and executed into the browser from this malicious website.

To approach a forensic analysis for web-browsers subject to any variant of web-based attacks, our proposed methodology consists of the following sequential procedures:

1) **Data Gathering:** it is the process of accessing the malicious URL in a setting similar to a compromised system, to lure the malicious URL to download the set of resources (content, code, and exploit payload) similar to those have been downloaded in the system subject to investigation.
2) **Data Analysis:** it is the process of executing, debugging the code downloaded from the malicious URL and producing a forensic trace file for objects, operations, functions created and/or called from the code. The trace file is further analyzed to extract all of the possible forensic evidences that would assist the investigation.
3) **Data Classification:** itis the process of classifying the forensic traces into subsets that could or could not assist the investigation. For example, traces would be classified into volatile, non-volatile forensic evidence, and traces that would not support the investigation, such as script files embedded by Google-ads that are almost exist in all websites. These script files may be downloaded along the malicious content but it is not relevant to the attack and cannot be considered in the forensic analysis process. The output from the classification process not only can be used to identify evidences in the compromise system, but also, can be used to develop a signature for the attack to locate attack traces in other systems in the network (postmortem analysis).

Figure 1, depicts a visual description for the proposed forensic methodology.

## 3.1 Data Gathering

The process of data collection requires simulating the settings of a compromised system subject to investigation to avoid downloading and executing code that has never been executed in the original system subject of the incident. In this scenario, an assumption has been made that the user was running a Firefox web-browser. As such, Firefox Browser Extension was developed to monitor, log, and debug the downloaded resources after accessing a malicious web page with a particular attention to the executed embedded JavaScript code. In the Proof-of-Concept implementation, Mozilla Debugger API[5] was used to develop a browser extension that outputs a detailed trace file. This trace file logs and lists the code executed from accessing the page subject to the forensic investigation. The trace file generated in a JSON[6] file format that includes objects created/accessed/-modified on the system with details about the stack frames of the executed code and the execution timestamps. The JSON object contains the following items:

- Type: a string describing the executed frame ('call', 'eval', global', 'debugger')
- Class: a string describing the ECMAScript [7] class of the referent
- Function name: the name of the function whose application created this frame (null if this is not a 'call' frame)

[5]**Mozilla Debugger API:** is a debugging interface provided by Mozilla JavaScript engine "SpiderMonkey", which enables JavaScript code to observe and manipulate the execution of other JavaScript code.

[6]**JSON:** JavaScript Object Notation

[7]**ECMAScript** is a trademarked scripting language specification standardized by ECMA International in ECMA-262 and ISO/IEC 16262.

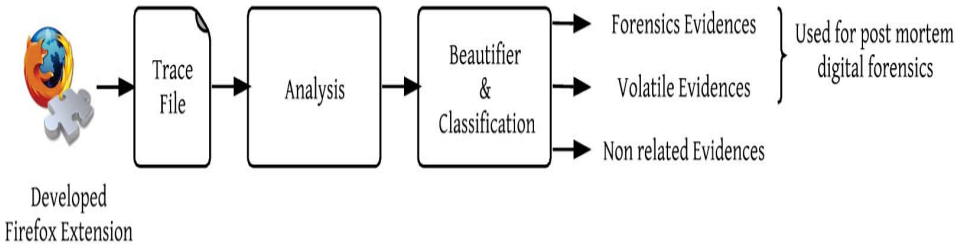


Fig. 1: A visualization for a web-browser forensics system

- Parameters: An object with the name/value pairs of the passed parameters to the called function
- URL: The url of the page in which the function have been called
- Script: The script being executed in this frame

We tested the browser extension using number of real world malicious URLs (55 malicious web site were analyzed) collected from public malware databases[8].

## 3.2 Data Analysis

The analysis for the generated trace files requires a detailed examination for the extracted JavaScript code. We developed an analyzer using NodeJS[9] to search for specific patterns using regular expressions. We search for patterns of obfuscation events, encoding/decoding events, checking for vulnerability events, URL redirection events, downloading external resources, creating local files on the system, etc. There are different well-known and commonly used techniques for cybercriminals to use a JavaScript code to perform malicious actions. For example, to download an external resource, an attacker may employ one of the following methods:

- Create a script tag and set the source attribute to the required downloadable file:

```
(function() {
var as =
document.createElement('script');
as.type = 'text/javascript';
as.async = true;
as.src =
 "xxxd31qbv1cthcecs.
 cloudfront.net/atrk.js ";
var s =
 document.
  getElementsByTagName('script')[0];
s.parentNode.insertBefore(as, s);
})();
```

- Create an image tag with source to a malicious URL:

```
var tempImage = new Image();
tempImage.src =
smf_prepareScriptUrl(smf_scripturl)
+ 'action=keepalive;time=' + curTime;
```

These different methods were taken into consideration during the analysis process. The output from the analyzer shows the number of occurrence or usage for each event. After this we transformed the extracted code into a human readable format, this is by utilizing a web-based services named JavaScript beautifier[10]. The extracted code was further analyzed to get a closer look into each evidence and manually extract the common patterns between various URLs.

## 3.3 Data Classification

To avoid providing a forensic analyst with a significant amount of irrelevant information, data classification and analysis procedure is a crucial activity for eliminating data that is not relevant to the case subject to investigation. As stated in [19], a forensic evidence has to characterize the following:

1) Admissible in the court
2) Authentic: the evidence proves a specific action in a specific incident
3) Complete/Exculpatory: can be used to support or refute a user action
4) Reliable: the procedure used for collecting and analyzing the evidence must guarantee the evidence validity and authenticity

The above characteristics and the aforementioned properties of the Drive-by-Download attack were considered in the evidences classification process. If evidence is related to the examined attack type, it will then be classified as relative and is further classified into (1) Volatile or (2) Non-volatile forensic evidence. Other data will be considered as none related evidences. For example:

- Volatile evidence: i.e. in memory shell-code, encoding/encryption code.

[8] www.malwaredomainlist.com/mdl.php, http://www.malwareurl.com/

[9] NodeJS is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.

[10] http://jsbeautifier.org, http://codebeautify.org/jsviewer

Table 1: Properties list example

| Evidence Name | Properties |
|---|---|
| URL redirection | domain name, path |
| Vulnerability check | branches depth, vulnerabilities name |
| String manipulation | string operations, string value and length |
| Downloaded resource | resource name, source url, resource type |
| Created file | file name, file path, file type |

- Non-volatile evidence: Created file on the system, downloaded resource, URL redirection with a trace of the URL in the browser history, etc.

The main reason behind this classification is to ensure that a forensic analyst would know that a volatile data related to the investigation may present but not necessary can be recovered nor reconstructed, and only the identified non-volatile forensics can be further used to develop an attak signature. The forensic analyst can then use the generated attack signature to detect if there is an attack on other systems.

After manually classified identified evidences, a list of properties will be extracted based on the evidence type, to reveal more information about it. Table 1 lists the possible properties for each evidence based on its type.

In addition, the following code sample for example shows a cookie file created from one of the analyzed JavaScript files.

```
f.cookie = e + "=; expires=
Thu, 01 Jan 1970 00:00:01 GMT;
path=/ " + (t ? "; domain= " + (l("msi ") ?
 " " : ". ") + "xxxaddthis.com " : " ")
```

Most of the analyzed URLs were also checking vulnerabilities and were trying to create an ActiveX object or shockwave flash which is an Adobe's Flash Player built directly into the browser as shown below.

```
d = "ShockwaveFlash ";
...d = b[c], -1 < d[r][q]("Shockwave Flash ")
&& n(e = d.description[y]("Shockwave Flash ")
[1]);
....
try {
  c = new ActiveXObject(d + ".7 "), e =
  c.GetVariable("$version ")
     }catch (f) {} if (!e)
 try {
  c = new ActiveXObject(d + ".6 "), e =
  "WIN 6,0,21,0 ",
c.AllowScriptAccess = "always ",
....
```

# 4. Experiments and Results

In this section, we present the preliminary results of our introduced web-browser forensic analysis method. Table 2 demonestrates the categorization for the 55 analyzed URLs.

Table 2: Categorization for the analyzed URLs

| Description | Number of URLs |
|---|---|
| Compromised site leads to Angler Exploit Kit | 15 |
| Directs to Exploits | 14 |
| Compromised site leads to Exploit Kit | 10 |
| Mass | 12 |
| Script.Exploit | 2 |
| Exploit Kit | 1 |
| Trojan | 1 |

Table 3: Distribution for the downloaded resources

| Category | Number of files |
|---|---|
| JavaScript | 249 |
| PHP | 42 |
| Images | 124 |
| CSS | 15 |

To generate the required trace files we accessed each of the 55 URLs separately using a virtual machine. The generated trace files were then passed to our developed analyzer. Figure2 demonstrate the output from the analyzer, it shows the number of occurance for some searched event like:

1) Vulnerability checking using ActiveXObject.
2) Vulnerability checking using Shockwave.
3) Downloading resources by assigning the src attribute.
4) Downloading resources using the iframe tag.
5) Created cookie files.
6) Encoding
7) Browser fingerprinting
8) Number of URL redirection

The x-axis represents the analyzed URLs and the y-axis represents the number of occurance for the searched events. One notable observation that most of the forensically analyzed URLs are profiling the users' behavior and fingerprinting their browsers. URL redirection, downloading external resources and creating cookie files are the most common events which are forensic evidences and can be used in our post mortem analysis.

After beautifying the extracted code we get a closer look for example into the type of downloaded files and the way used to download it. Table3 demonstrates the distribution for the founded files. A visualization for the file distribution is shown in figure 3

Our experiment showed that we can get a detailed trace file for any executed malicious JavaScript and if this JavaScript file tries to execute a php file or load a malicious ad for example, by creating an iframe and setting the source attribute to that page we can find traces for that file as shown in the code below.

```
iframe.src = iframe_url;
iframe.style.display = "none ";

document.write('<div id= "slide_up ">
<div id= "close_btn_noCookie ">X
```

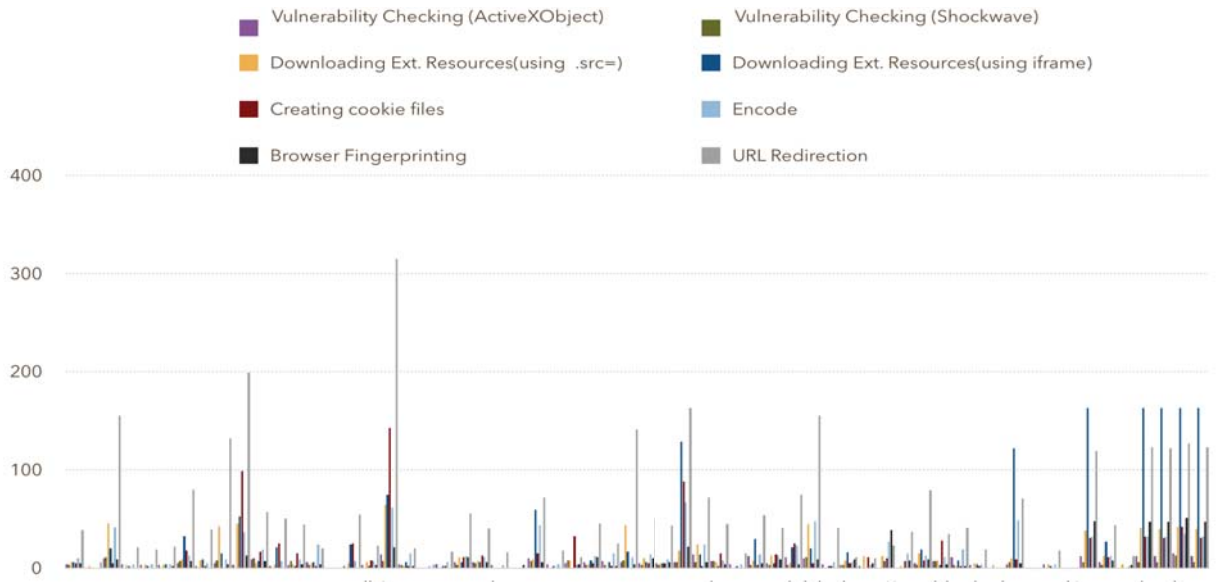


Fig. 2: The Distribution for the searched events

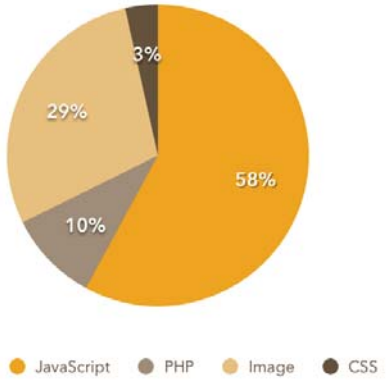


Fig. 3: Files Distribution

```
</div><iframe id= "su_frame " src=
"xxxpcash.imlive.c../releasese/...
```

We also use jsunpack-n[11] program, which emulates browser functionality to detect malicious code and we compare the list of JavaScript files analyzed by our Firefox extension with the files analyzed by it. We consider this comparison as a benchmark for our gathered data. For 36 URL, our extension has successfully detected 61% of the JavaScript files detected by Jsunpack-n and detected 24% of the JavaScript files for the rest of the analyzed URLs. It also detected and analyzed 54 JavaScript files which were not detected or mentioned by Jsunpack-n. Our explanation is that the developed extension only extract script files whose functions were loaded and executed in memory and those script files have no executed functions and were not loaded in memory while loading the page especially that most of these files are responsible for UI interactions like draggable.min.js, menu.min.js, mouse.min.js and so on.

[11] **Jsunpack-n:** is a command-line Javascript unpacker that has more or less the same features as the web version of Jsunpack

## 5. Conclusion & Future Work

Web-based attacks are gaining an increasing momentum and attention from cyber criminal and security researchers; alike, i.e. Drive-by-Download attack vector analysis, detection and prevention. In this paper, we introduce a forensic analysis method to examine web-browsers artifacts produced by accessing malicious URLs. A Proof-of-Concept Firefox Browser extension is developed to enable getting detailed information about the attack, techniques used to evade the detection tools, downloaded malicious resources and executed malicious code. Digital evidence trace file is output for each examined URL that contains a set of volatile and non-volatile forensic evidences that would assist a forensic analyst in the investigation. Our approach gives a closer look at the real code executed from the client side. In the future work, the introduced implementation will be extended to the different browsers, i.e. Google Chrome and IE.

## References

[1] Zhang, J., Seifert, C., Stokes, J.W. and Lee, W., 2011, March. Arrow: Generating signatures to detect drive-by downloads. In Proceedings of the 20th international conference on World wide web (pp. 187-196). ACM.

[2] Egele, M., Kirda, E. and Kruegel, C., 2009. Mitigating drive-by download attacks: Challenges and open problems. In iNetSec 2009âĂŞOpen Research Problems in Network Security (pp. 52-62). Springer Berlin Heidelberg.

[3] Ligh, M., Adair, S., Hartstein, B. and Richard, M., 2010. Malware analyst's cookbook and DVD: tools and techniques for fighting malicious code. Wiley Publishing.

[4] Jayasinghe, G.K., Culpepper, J.S. and Bertok, P., 2014. Efficient and effective realtime prediction of drive-by download attacks. Journal of Network and Computer Applications, 38, pp.135-149.
[5] Kotov, V. and Massacci, F., 2013. Anatomy of exploit kits. In Engineering Secure Software and Systems (pp. 181-196). Springer Berlin Heidelberg.
[6] Ohana, D.J. and Shashidhar, N., 2013. Do private and portable web browsers leave incriminating evidence?: a forensic analysis of residual artifacts from private and portable web browsing sessions. EURASIP Journal on Information Security, 2013(1), pp.1-13.
[7] Aggarwal, G., Bursztein, E., Jackson, C. and Boneh, D., 2010, August. An Analysis of Private Browsing Modes in Modern Browsers. In USENIX Security Symposium (pp. 79-94).
[8] Sonntag, M., Automating Web History Analysis. na.
[9] Van Overveldt, T., Kruegel, C. and Vigna, G., 2012. FlashDetect: ActionScript 3 malware detection. In Research in Attacks, Intrusions, and Defenses (pp. 274-293). Springer Berlin Heidelberg.
[10] Laskov, P. and Åărndiäğ, N., 2011, December. Static detection of malicious JavaScript-bearing PDF documents. In Proceedings of the 27th Annual Computer Security Applications Conference (pp. 373-382). ACM.
[11] Cova, M., Kruegel, C. and Vigna, G., 2010, April. Detection and analysis of drive-by-download attacks and malicious JavaScript code. In Proceedings of the 19th international conference on World wide web (pp. 281-290). ACM.
[12] Curtsinger, C., Livshits, B., Zorn, B.G. and Seifert, C., 2011, August. ZOZZLE: Fast and Precise In-Browser JavaScript Malware Detection. In USENIX Security Symposium (pp. 33-48).
[13] Canali, D., Cova, M., Vigna, G. and Kruegel, C., 2011, March. Prophiler: a fast filter for the large-scale detection of malicious web pages. In Proceedings of the 20th international conference on World wide web (pp. 197-206). ACM.
[14] Ratanaworabhan, P., Livshits, V.B. and Zorn, B.G., 2009, August. NOZZLE: A Defense Against Heap-spraying Code Injection Attacks. In USENIX Security Symposium (pp. 169-186).
[15] Rieck, K., Krueger, T. and Dewald, A., 2010, December. Cujo: efficient detection and prevention of drive-by-download attacks. In Proceedings of the 26th Annual Computer Security Applications Conference (pp. 31-39). ACM.
[16] Xing, X., Meng, W., Lee, B., Weinsberg, U., Sheth, A., Perdisci, R. and Lee, W., 2015, May. Understanding Malvertising Through Ad-Injecting Browser Extensions. In Proceedings of the 24th International Conference on World Wide Web (pp. 1286-1295). International World Wide Web Conferences Steering Committee.
[17] Choi, J.H., Lee, K.G., Park, J., Lee, C. and Lee, S., 2012. Analysis framework to detect artifacts of portable web browser. In Information Technology Convergence, Secure and Trust Computing, and Data Management (pp. 207-214). Springer Netherlands.
[18] Oh, J., Lee, S. and Lee, S., 2011. Advanced evidence collection and analysis of web browser activity. digital investigation, 8, pp.S62-S70.
[19] Kozushko, H., 2003. Digital evidence. online], http://infohost. nmt. edu/ sfs/Students/HarleyKozushko/Papers/DigitalEvidencePaper. pdf.
[20] De Maio, G., Kapravelos, A., Shoshitaishvili, Y., Kruegel, C. and Vigna, G., 2014. Pexy: The other side of exploit kits. In Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 132-151). Springer International Publishing.