

Filtering Malicious Routines in Web Browsers Using Dynamic Binary Instrumentation

Jae Won Min *, Sung Min Jung *, Tai Myoung Chung **

* Department of Electrical and Computer Engineering, Sungkyunkwan Univ, 300 Cheoncheon-dong, Jangan-gu, Suwon-si, Gyeonggi-do, Korea

** School of Information Communication Engineering, Sungkyunkwan Univ, 300 Cheoncheon-dong, Jangan-gu, Suwon-si, Gyeonggi-do, Korea

jwmin@imtl.skku.ac.kr, smjung@imtl.skku.ac.kr, tmchung@ece.skku.ac.kr

Abstract—Extension of web browser capability has introduced variety of security problems. Attackers exploit vulnerabilities found in internals of web browsers or plugins to compromise the system and execute arbitrary code. In this paper, we present a filtering method which blocks malicious routines from being executed. Filtering is done by keeping a blacklist of dangerous routines and arguments separately to compare with the routines being called by the web browser at runtime. To show the concept, we built a prototype by using dynamic binary instrumentation framework called Pin. Pin provides rich API that allows us to build a custom tool that inserts instrumentation codes in the program. Using this tool, we are able to block malicious instructions from being executed.

Keywords— Web Browser, Security, Sandbox, Vulnerability, Dynamic Binary Instrumentation

I. INTRODUCTION

Unlike text based web browsers of the past, modern web browsers provide rich user interface and various multimedia services. As web browsers kept extending functionalities and became more complex over the years, it introduced many software bugs. Attackers exploit vulnerabilities found in parsers, browser plug-ins and other places to execute arbitrary code in the victim's system.

To defend from common attacks, web browsers these days have their own security architectures and policies to protect themselves from attacks. Most web browsers use sandbox as default or provide as an extension to restrict privileges of the process so that attacker's code cannot do much harm.

However, attackers constantly find a way to escape the sandbox and execute chain of exploits to finally escalate privilege to root. Moreover, plug-ins are not usually sandboxed due to compatibility problems, so vulnerabilities found in widely used plug-ins like Adobe Flash may lead to major security incidents.

Therefore, instead of concentrating on securing the application, blocking what adversaries want to execute in the final stage of attack seems more reasonable. We propose a routine filter that monitors routines executed by the web browser for malicious codes. It uses blacklist of malicious instructions for detection.

Extending our previous paper [3], which proposed the general model of the filter, we explain specific details of the

filter implementation. Our implementation uses dynamic binary instrumentation framework to insert filtering code in the web browser. The blacklist consists of two separate databases of routine names and arguments.

The remainder of this paper is organized as follows. In section 2, we explain architectures of major web browsers and their malware protection techniques in section 3. In section 4, we describe common type of attacks and present the implementation of the routine filter in section 5. Finally, we discuss related work in section 6 and conclude the paper in section 7.

II. BACKGROUND

A. Chrome

Google Chrome is implemented in a modular architecture. It has two main modules which are the rendering engine and the browser kernel. Rendering engine module does converting of HTTP responses and user input events into rendered bitmaps. The browser kernel is for interacting with the operating system.

The rendering engine is the only module that is sandboxed. Therefore even if rendering engine becomes compromised by an adversary, he or she is unable to give harm to the system because of its low privilege. Components such as the HTML parser, Java virtual machine are allocated inside the rendering engine because they are highly vulnerable to attacks. Figure 1 is a diagram describing the Chrome architecture.

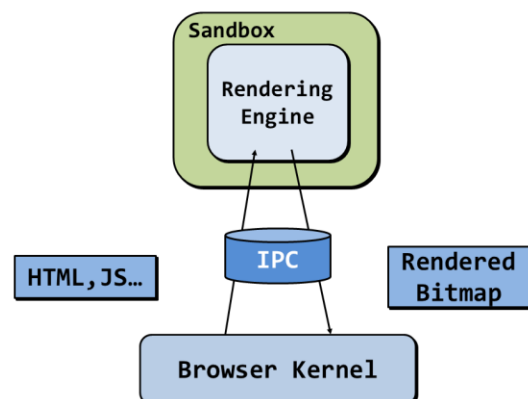


Figure 1. Architecture of Chrome

The browser kernel implements APIs to interface with the system and instances of the rendering engine. State information about the privileges of the rendering engines are stored to implement security policies based on them that restricts interaction with the user system.

Plug-ins are executed in separate host processes because it is too risky to place them in the browser kernel and placing inside the rendering engine will result multiple instance of the same plug-in. Moreover, plug-ins are not sandboxed because restricting its privilege might result compatibility problems[1].

B. Firefox

Firefox is an open source web browser developed by Mozilla. Firefox implements a layered architecture which makes maintaining code easy. Figure 2 describes architecture of Firefox.

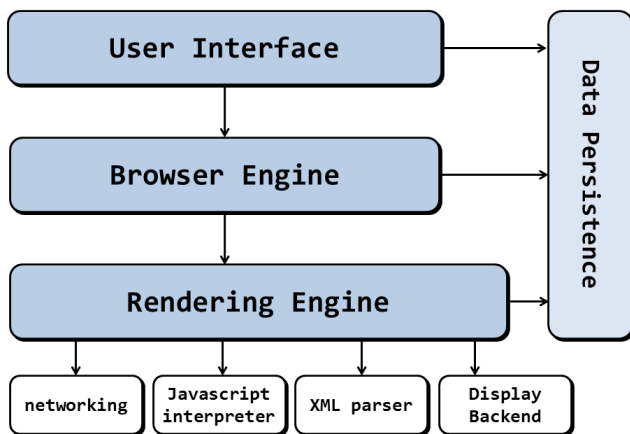


Figure 2. Architecture of Firefox

User interface layer is an interface between the user and the browser engine. The browser engine is responsible for downloading modules, parsing and applying CSS styles. Table below shows sub-components of the browser engine.

TABLE 1. SUB-COMPONENTS OF THE BROWSER ENGINE

Name	Description
Layout	Manage the layout of the DOM for rendering
Rendering	Translates the loaded DOM and style information
Style	Apply styles to the parsed html
DOM	Abstraction layer above the underlying components representing web site content
HTML Parser	Parse incoming content and load DOM

Rendering engine is sub-component of the browser engine which mainly produces visual output of the web page. Javascript interpreter, display backend, XML parser and networking component are used by the rendering engine.

Data persistence layer consist of browser persistence and user data. MozStorage which is a database built on top of SQLite is used store browser information. On the other hand,

user data are saved in cookies, DOM storage and flash local storage[2].

C. Internet Explorer

Internet Explorer is built in modules so each of them can be reused and enhanced by developers. Below figure depicts the main modules of Internet Explorer.

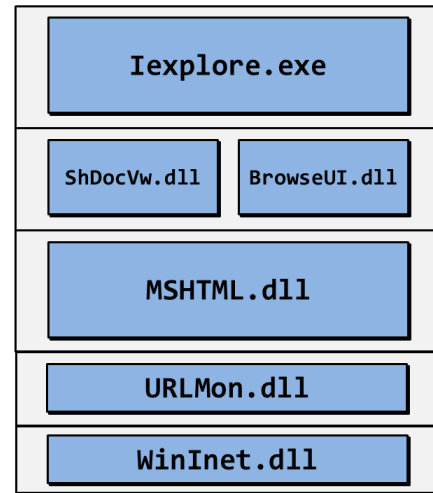


Figure 3. Architecture of Internet Explorer

IExplore.exe does rendering and implements protocols. It has dependency on other modules inside the web browser. BrowseUI.dll provides user interface such as menu and toolbar. ShDocVw.dll is the core component of Internet Explorer and supports Microsoft programs as well as non-Microsoft programs. MSHTML.dll parses HTML and Cascading Style Sheet (CSS) and renders web pages. Lastly, URIMon.dll is for downloading and MIME handling while WinInet.dll handles Windows Internet protocols [4].

III. MALWARE PROTECTION

A. Chrome

Google crawls the web looking for new malicious sites and add them to the list when found. Furthermore Google test the malicious looking sites inside the virtual machine to see if it is actually malicious. It is explained in more detail in [7]

B. Firefox

Built-in phishing and malware protection keeps lists of reported phishing and malware sites and compared every time user browses web sites. Lists are automatically updated every 30 minutes [5]

C. Internet Explorer

Internet Explorer 9 provides protection against social engineering, browser extension attacks and web attacks. Feature called “Application Reputation” warns the users of potentially dangerous files. Moreover, Internet Explorer 9 provides filters and defences that protect users from common web attacks and block untrusted web sites or browser extensions[6]

IV. TYPE OF ATTACKS

A. Remote code execution

If a browser or its plug-ins have software bugs, malicious codes can exploit those bugs and cause the web browser to run arbitrary code[9]. These types of exploits are very common. Comex, an iOS hacker, exploited Freetype Type 1 font parser in the mobile Safari to unlock iOS device by crafting a special PDF file.

B. Drive by download

Drive by download is a remote code execution attack which downloads files to the user system without any notification. When a user connects to a malicious site, the site collects information about victim's web browser and executes suitable exploit code to launch the attack. Once downloaded, files can do malicious activities[13]. Webkit is known to have bugs which are vulnerable to drive by download attacks.

C. Clickjacking

Clickjacking is attack technique that access confidential information or control victim's computer[14]. It embeds attack code in the script without user knowing the existence of the code. Attack is triggered by events like user clicking a button.

V. MALICIOUS ROUTINES FILTER

A. Architecture

If all the routines executed by the web browser are monitored, we are able to filter out attempts like accessing important system directories or actions which are not normal for web browsers. Following figure explains the concept of our filtering mechanism.

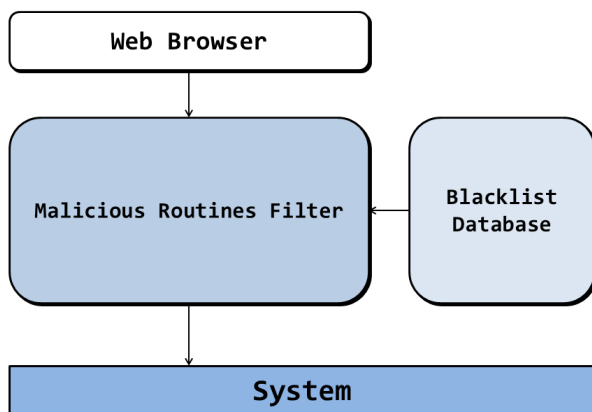


Figure 4. Concept of filtering malicious routines

The web browser runs as an independent process. When the web browser needs to upload a file for example, it needs to access the user file system. Between the web browser and the system, there is a filter that checks the issued instructions with the blacklist of potential threats. If the instruction about to be executed has a match in the blacklist, it considered as

malicious and is rejected. Consider a cookie stealing attack. To steal the cookie, attacker has to access the directory where user cookies are stored. If directory where the cookie is saved is included in the blacklist, every attempt to access to cookie will be discarded.

The advantage of this model is that it gives full functionality to the web browser and its plug-ins. It does not restrict the browser or sandbox the plug-in. The filter just traces issued instructions. Therefore compatibility of plug-in is maintained. When the attacker tries to exploit plug-in bugs and issue arbitrary commands to the system, it is caught before accessing the user system and is rejected. Therefore user system is safe.

For this filter to be most effective, design of the blacklist is important. The blacklist consists of routines and parameters. Intercepting every routines and checking whether it is malicious gives us high overhead. Therefore, routines that must be intercepted are included in routines field of the blacklist. In the parameters field, parameters that can be exploited for attacks are listed. These can be a certain system directory path or code.

B. Implementation

To implement the filter, we used a dynamic binary instrumentation (DBI) framework called Pin[11]. Using Pin, we can inject arbitrary C or C++ codes in the executable which suits our needs. Moreover, it provides rich API which allows us to write to write custom instrumentation code. Figure below depicts the architecture of the routine filter.

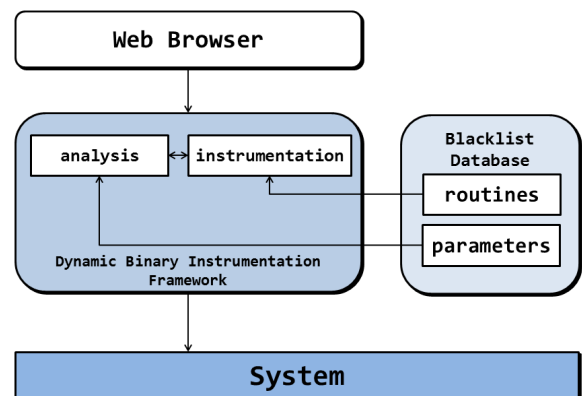


Figure 5. Architecture of malicious routines filter

Our implementation used Pin version 2.8-45467 and was tested in Fedora 16. As explained in section 5, the blacklist consists of parameters and routines. After Pin is loaded and initialized, instrumentation codes are injected for the routines which are specified in the blacklist. Then, the web browser is executed under the control of Pin. When the web browser is started, Pin intercepts the first trace of instructions and the Just-In-Time compiler incorporates instrumentation code and compiles to make new instructions[15]. Every time instrumented routines are executed, analysis module compares parameters with the ones from the blacklist. If there is a match,

that certain routine is considered malicious and blocked from being executed. Below figure describes the process.

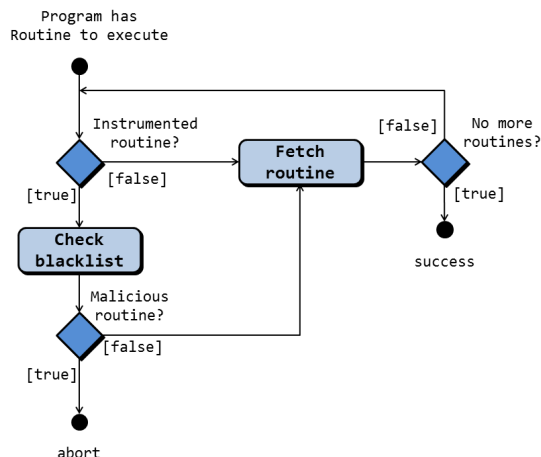


Figure 6. Flowchart of detecting malicious routines

C. Example Attack

We wrote a simple C program to test whether our tool can successfully detect shell execution. Below figure shows the code of the C program.

```

#include <stdio.h>
int main(void)
{
    setuid(0);
    setgid(0);
    system("/bin/sh");
    return 0;
}

```

Figure 7. Test C Program

We inserted execve function in the routine database and “/bin/sh” in the parameter database. C program was executed by Pin with our tool running. As a result, by examining the trace log, the program called execve function and was successfully detected by our filter. Therefore we expect our routine filter to successfully defend against web browser attacks.

VI. RELATED WORK

Various methods of detecting malicious codes were already proposed by researchers. Vladimiri Kiriansky et al. [12] proposed program shepherding approach to detect malicious codes by monitoring all control transfers and applying security policy. However, this approach is complex and arguments are bound to the instruction which means that to block two different instructions with same argument, the specific arguments must be stated twice in the policy. Our approach does not require same argument to be stated multiple times because arguments and instructions are not bound to each other.

Google patent by Song et al. [10] proposed a detecting method for malicious codes. Architecture consists of two main modules: system call interposition module and analysis module. System call interposition module intercepts all system calls and related arguments. Then, analysis module analyses

system calls. If real execution of the code and result of analysis have same behaviour, code is rejected. Drawback of this technique is that still there are possibilities of incorrect results. Moreover it has to analyse before knowing the code is malicious although some code is obviously malicious. However our method rejects obviously malicious codes without any analysis.

VII. CONCLUSION

To protect from attacks, web browsers implement sandboxes and other security mechanisms. However, attackers constantly find security holes and exploit them. Filtering malicious codes by monitoring critical routines stops the final stage of the attacks which makes the previous stages useless. Because the blacklist database is maintained in a separate file, there is a possibility that it can be put on cloud services so clients share same blacklist made by the service provider. For further research, we plan to test our tool with latest web browser exploits and optimize the detecting algorithm and the blacklist format so that time overhead for detecting malicious code and accessing data in the blacklist decreases.

ACKNOWLEDGMENT

This work was supported by the IT R&D program of MKE/KEIT. [KI001810039260, Integrated dev-environment for personal, biz-customized open mobile cloud service and Collaboration tech for heterogeneous devices on server]

REFERENCES

- [1] Adam Barth, Collin Jackson, Charles Reis, Google Chrome Team, “The Security Architecture of the Chromium Browser”, 2008
- [2] Andre Campos, Bryan Lane, Neal Clark, Sunpreet Jassal, Stephen Hitchner, “Conceptual Architecture of Firefox”, June 2007
- [3] Jae Won Min, Sung Min Jung, Tai Myoung Chung, “Secure Interface Layer for Web Browsers”, CUTE 2011
- [4] Dhruvajita Devi, Dhruvajyoti Pathak, Sukumar Nandi, “Vulnerabilities in Web Browsers”, Indian Institute of Technology, Guwahati, India
- [5] (2011) Phishing and Malware Protection [Online], Available : <http://www.mozilla.org/en-US/firefox/phishing-protection/>
- [6] (2011) Microsoft Safety & Security Center [Online]. Available: <http://www.microsoft.com/security/pc-security/ie9.aspx>
- [7] (2008) The Chromium Blog [Online], Available: <http://blog.chromium.org/2008/11/understanding-phishing-and-malware.html>
- [8] Alan Grosskurth, Michael W. Godfrey, “A Reference Architecture for Web Browser”s, Univ. of Waterloo
- [9] (2011) Browser Exploit [Online] Available: http://en.wikipedia.org/wiki/Browser_exploit
- [10] Chen-Hwa Song, Ying-Yuan Huang, Kaohsiung County “Detecting method and architecture thereof for malicious codes”, US Patent 7,665,138 B2, Feb 2010
- [11] (2011) Pintool – A dynamic binary instrumentation tool [Online] Available: <http://www.pintool.org>
- [12] Vladimir Kiriansky, Derek Bruening, Saman Amarasinghe, “Secure Execution Via Program Shepherding”, USENIX Security ’02
- [13] Manuel Egele, Engin Kirda, Christopher Kruegel “Mitigating Drive-by Download Attacks: Challenges and Open Problems”, INETSEC 2009
- [14] (2011) Clickjacking [Online] Available: <http://en.wikipedia.org/wiki/Clickjacking>
- [15] Michalis Polychronakis, Angelos D. Keromytis “ROP Payload Detection Using Speculative Code Execution”, MALWARE 2011