# A Protection Architecture for Malicious JavaScripts on Web Browsers

Woung Jang, Jonghun Jung, Myoungsun Noh, Kyungho Jung, and Chaetae Im
Internet Incidents Response Technology Team
Korea Internet & Security Agency
IT Venture Tower, Jungdaero 135, Songpagu, Seoul, Korea.
Email : {jangw2232, jjh2640, nmsnms, khc, chtim }@kisa.or.kr

## ABSTRACT

'Web-based Cyber Attacks' for leaking private information or making target system to denial of service (DoS) are arising. Traditional Web-based cyber attacks hide malwares in the Web pages to make it install and make the client under control. Nowadays, however, these attacks have evolved to 'Script-based Cyber Attacks' which infect the visitors just by accessing a certain Web page, not using malwares. Traditional Web-based cyber attacks can be protected by detecting malwares, but script-based cyber attacks cannot be protected by the former method because no malwares are detected. Therefore, new architecture is necessary to prevent malicious behaviors on Web browsers. In this paper, a protection architecture is introduced to detect the connection requests for URLs which are used by the attacker, to detect Web pages which have malicious scripts, and to prevent the execution of the malicious JavaScript codes.

## KEYWORDS

Script-based Cyber Attacks; Browser-level Protection Architecture; Browser Helper Objects (BHO).

## 1 INTRODUCTION

'Web-based Cyber Attacks' have been continually increased for attackers to leak private information or making target system to denial of service (DoS) by infecting malwares to visitors through web sites. As shown in Fig. 1, however, in case of the 6.25 cyber attack to Web pages of government organizations in South Korea in 2013, a new type of cyber attack called 'Script-based Cyber Attack' happened which makes Distributed DoS (DDoS) attack to the Web pages by the visitors, without infection of malwares. Traditional web-based cyber attacks can be protected by detecting the download of malwares, but the script-based cyber attacks cannot be protected by the former method because of not using malwares. Furthermore, with the spread of HTML5 environments, various types of attacks can appear by using new features supported in the HTML5 standard, such as Web sockets or Web workers. Therefore, a new architecture is necessary to prevent the script-based attack on Web browsers.

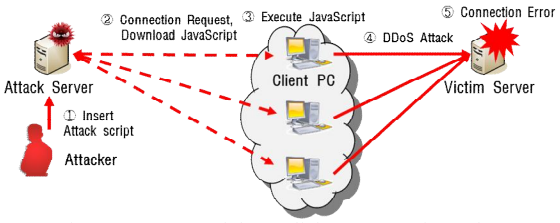


Fig. 1. DDoS Attack by Malicious JavaScript code

Most of the Web browsers support browser extensions to expand some intended functions or modify Web pages. Therefore, with an appropriate browser extension, we can protect the malicious JavaScript codes by scanning the Web pages before it rendered. In this paper, a protection architecture based on the browser extension is introduced to detect the connection requests for URLs which are previously used by attacks, to detect the Web pages which have malicious JavaScript codes, and to prevent the execution of the malicious JavaScript codes by blocking APIs which are essential for the attack.

## 2 BACKGROUNDS & RELATED WORK

The protection methods can be classified into three that is in Fig. 1. The first one is the protection in the first step, blocking the insertion of malicious JavaScript codes to an attack server. The second one is the protection in the second step, blocking the connection to the attack server or the download of malicious JavaScript codes on the network. The last one is the protection in the third step, blocking the execution of the malicious JavaScript codes on the client browser.

The first method should consider the routes of inserting malicious JavaScript codes, or the malicious server made for the attack. However, this method will not be introduced in this paper because of focusing the main topic of this paper.

The second method has some advantages that the protection system on the network can stand without installing the additional agent program on the client, and it can cover the whole users on the network to which the system is applied. This method should consider about an appropriate architecture for detecting malicious code in the user traffic, which include identification an Web session on the network traffic, extraction and analysis of malicious JavaScript codes on the Web page, and notification to the detected user. Furthermore, the system on the network which has this architecture should have high analyzing speed for providing low latency to the users, because the analysis engine on the protection system should analyze huge amount of JavaScript codes on the network, and the collected codes are text string, not just binary stream. And also, the system should consider many exceptions on the network for stable operation.

Moshchuk et al. introduces SpyProxy [1], which is a system to detect malicious behaviors by loading Web pages directly on a browser in a virtual machine. The system made an effort for low latency of users by containing Web cache and proxy, which sends partial safe contents to the clients. But, this system need an additional agent program on the clients. Li et al. proposes WebShield [2], which suggests a method that could run the system not dependent on the additional agent program on the client. But the system uses sandboxes for each client on the network to make content analysis precise, and the whole sandboxes requires huge memory on the system. To this end, the system has limited performance that could analyze the contents of at most 70 users.

The third method, the protection method of blocking the malicious JavaScript codes on the client browser, on the other hand, it makes each user install an additional software which makes the user uncomfortable in some sense. Nevertheless, this approach can detect the browser events and analyze the content easily, and it does not have to deploy additional network equipment on the network. Therefore, we follow this approach in this paper.

There are two approaches to realize this method. One is to detect each connection request by requested URL which has been used previously by the attacker, and the other one is to detect malicious JavaScript codes by analyzing the downloaded contents. The former method is used by Webcheck [3] and SiteAdvisor [4]. This method is intuitive and fast, but have a weakness in case of changing attack site frequently. The latter method is used in JSAND [5] suggested by Agten et al. Specifically, they suggest a method that makes the sandbox for blocking malicious behaviors of third-party JavaScript codes in the contents by monitoring the downloaded Web contents. This method makes it possible to block the third-party JavaScript codes, but if the malicious JavaScript codes are in the main script in the Web page, it is impossible to detect. This paper suggests an architecture that applies two methods at the same time.

## 3 PROPOSED ARCHITECTURE

The method of blocking the malicious JavaScript codes using the browser extension

depends on Web browser. If detection module has a dependency in the type of browser, this architecture have low expendability and reusability in the multi browser environment.

Therefore, we propose the architecture that the detection module take a part in an independent program to communicate each browser extensions.

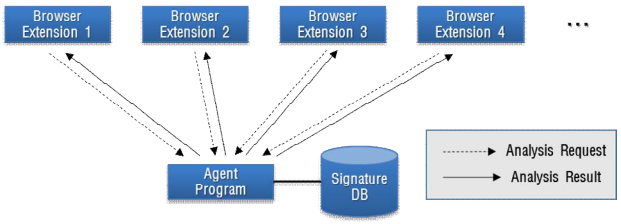


Fig. 2. Proposed architecture (abstract)

In the abstracted architecture in Fig. 2, agent program can communicate each browser extensions. These extensions can request analysis for URLs at the time that the client requested, and JavaScript codes in the Web page before the codes are executed.

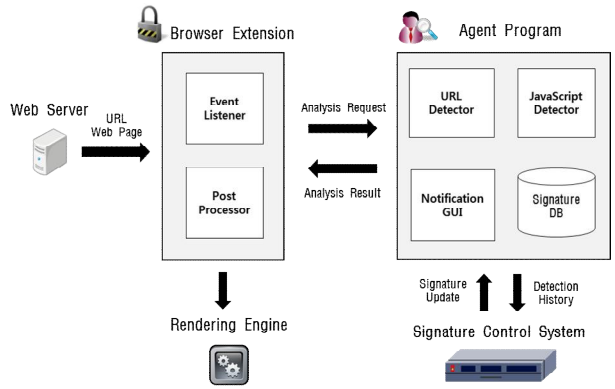


Fig. 3. Proposed architecture (full)

As shown in Fig. 3, browser extensions have two modules, and agent program have four modules. The descriptions about each component are in the next chapters.

### 3.1 Browser Extension

To catch the events on the browser and prevent the execution of malicious JavaScript codes, the browser extension consists of two modules. The event listener catches the browser events, and the post processor modifies JavaScript codes which are defined as malicious code. In Fig. 4, it shows the type of events on Internet Explorer. [6]

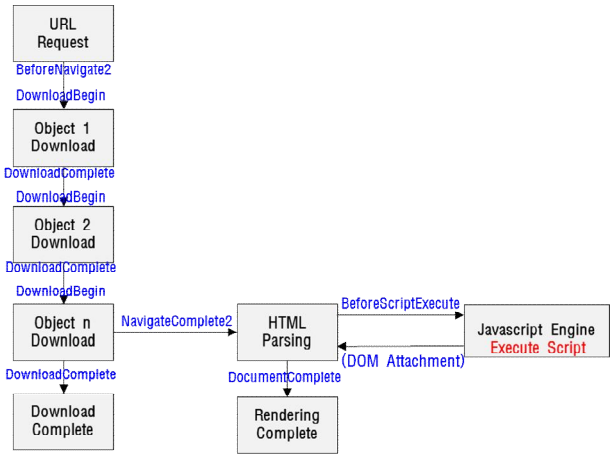


Fig. 4. Browser events (Internet Explorer)

At the time of URL requested, the event listener can catch the browser event named BeforeNavigate2, so the Browser Helper Object (BHO) can request the agent program to identify the URL. At the time of just before executing JavaScript codes in the Web page, the event listener can catch the browser event named BeforeScriptExecute, so the BHO can request the agent program to identify that the JavaScript codes in the page are malicious or not.

At the post processer, it receive the analysis result and take appropriate action. If the requested URL is on the blacklist, the post processor can block the connection, or redirect to an extra page followed by the policy. If the requested Web page has malicious JavaScript codes, it modifies the JavaScript codes follows by the analysis result which has the modified codes not to execute it, or block the whole page or redirect to an extra page followed by the policy. In Fig. 5, it shows the example of notification on detection of the malicious JavaScript codes, and the modified Web page not to execute the JavaScript codes.

ISBN: 978-1-941968-02-4 ©2014 SDIWC

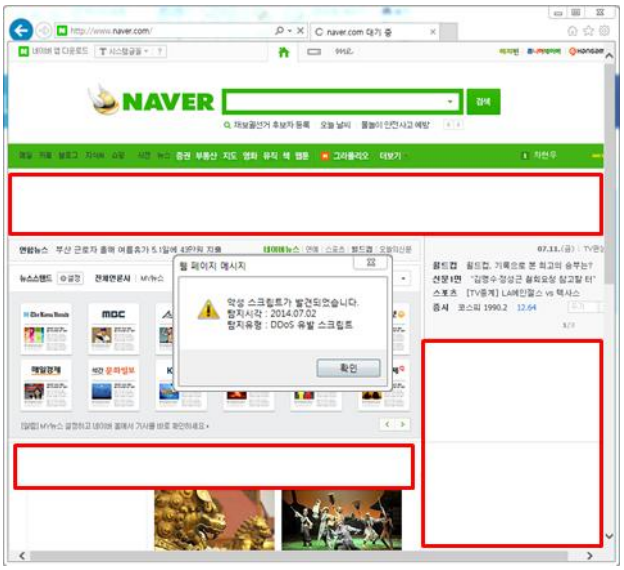

Fig. 5. Example screenshot. (Internet Explorer)

### 3.2 Agent Program

To support various types of browser extensions, the agent program is outside of Web browser. The agent program consists of three modules and one database for detection.

The malicious URL detection module has a role to identify the requested URL on the browser is malicious or not from the analysis request message. When this module find the requested URL is malicious, it responses the result which contains the policy to post process. And it make the notification GUI notify the result to the client user. Blacklist URLs are updated regularly from the external database.

The malicious JavaScript detection module has a role to identify the malicious JavaScript codes in the requested Web page by matching the malicious JavaScript code signature. When this module find the requested Web page is malicious, it responses the result which contains the deletion point on the JavaScript code and the policy to post process. This module finally make the notification GUI notify the result to the client user.

The Notification GUI module has a role to notify the analysis result to client user when it is malicious. This notification contains detected time, detected URL, type of malicious behavior, and processed policy.

## 4 CONCLUSION & FUTURE WORK

In this paper, the prevention architecture is introduced for blocking malicious JavaScript codes on Web browsers by browser extensions and an agent program. Because this architecture depends on the signature matching with the detected signature information, it is important to maintain and update the types of attack signatures. Therefore, it is necessary to identify new attack types by monitoring regularly, and to improve the detection rate of malicious JavaScript codes by extracting signatures on the attack script samples. Furthermore, because experiments on this architecture have not been conducted sufficiently, additional tests should be conducted with optimization of the developed software.

## ACKNOWLEDGEMENT

This work was supported by the ICT R&D Program of MSIP/IITP. [14-912-06-002, The Development of Script-based Cyber Attack Protection Technology]

## REFERENCES

[1] A. Moshchuk, T. Bragin, D. Deville, S. D. Gribble, and H. M. Levy, "SpyProxy: Execution-based Detection of Malicious Web Content," in Proc. of USENIX Security, 2007.

[2] Z. Li, Y. Tang, Y. Cao, V Rastogi, Y. Chen. And B. Liu, "WebShield: Enabling Various Web Defense Techniques without Client Side Modifications," in Proc. of NDSS, 2011.

[3] WebCheck, Korea Internet & Security Agency. Available: http://webcheck.kisa.or.kr

[4] SiteAdviser. McAfee. Available: http://www.siteadvisor.com

[5] P. Agten, S. V. Acker, Y. Brondsema, P. H. Phung, L. Desmet, and F Piessens, "JSand: Complete Client-side Sandboxing of Third-party JavaScript without Browser Modifications," in Proc. of the 28th Annual Computer Security Applications Conference, 2012.

[6] DWebBrowserEvents2 interface, MSDN, Microsoft. Available : http://msdn.microsoft.com/en-us/library/aa768283(v=vs.85).aspx