BOSF: BY-OWNER SCRIPT FILTERING

Jungtae Kim[1)], Kyoung-Soo Han[1)], Biao Jiang[1)], Eul Gyu Im[2)]
Dept. of Electronics and Computer Engineering[1)],
Div. of Computer Science and Engineering[2)],
Hanyang University, Seoul, Korea
{ohman21, 1hanasun, dante, imeg}@hanyang.ac.kr

ABSTRACT

Cross-Site Scripting (XSS) attacks are one of the most prevalent security threats all over the world. Although various defense methods have been proposed, most of them lack usability. Motivated by this reason, we present a location-based solution, called By-Owner Script Filtering (BOSF), which determines whether a script on a webpage can be executed or not using either the rules defined in the webpage or the information fetched from our database based on the ownership. BOSF is effective against various XSS attacks with minimal performance overheads, and more importantly, it places little requirements on developers and users.

KEYWORDS

Cross-Site Scripting (XSS) attack, By-Owner Script Filtering (BOSF), website/webpage, URL, custom meta element.

1 INTRODUCTION

Due to the rapid development of Internet, Cross-Site Scripting (XSS) attacks have become one of the most prevalent security threats all over the world [12]. XSS takes advantage of vulnerabilities of web applications to bypass security schemes and execute unauthorized scripts. XSS enables attackers to violate security policies, extract sensitive information, and perform malicious actions.

Because of the prevalence and criticality, XSS has been widely studied. The basic defense against XSS is to validate all untrusted inputs. To assist better implementation of validation, static and dynamic analysis [9][10] and verification [11] tools have been proposed for web applications. However, it is much more difficult than it seems, mainly resulted from different quirk modes provided by different browsers. Quirk modes were intended for backward compatibility, but they have been made use for XSS attacks. A cheat sheet of XSS vulnerabilities of a wide range of browsers is available at [13]. To further mitigate XSS attacks, various advanced solutions are proposed. By the parties involved, they can be categorized into client-side, server-side, and server-client solutions. In terms of client-side solutions, the most popular solutions are white- and/or blacklist-based script filters, such as NoScript [14]. Other examples of client-side solutions include Noxes [6] which works like a firewall, a proxy by Ismail et al. [7], an information flow tracker by Vogt et al. [8]. On the other hand, examples of server-side solutions include SessionSafe [4] and SWAP [5] which is a reverse proxy. The server-client solutions might be the most promising, the underlying philosophy of which is that the server determines the scripting policies and the clients enforce them. The pioneer of this kind is BEEP (Browser Enforced Embedded Policies) [1], which uses one-way hashing and DOM sandbox to mitigate XSS. In addition, Noncespaces [2] generates random namespace prefixes to help clients identify trusted scripts, and Blueprint [3] produces a parse tree to enforce policies.

However, most of the solutions require information security knowledge from developers and/or users of websites; for example, even the simplest NoScript is difficult for ordinary people without special training on network security. This prevents them widespread deployment without specific education. Moreover, some solutions need considerably large modification on servers and/or browsers, which is usually too costly and sometimes impossible.

Motivated by the facts above, we present a location-based solution By-Owner Script Filtering (BOSF), which requires a little security knowledge

from developers and nothing from users, and needs no modification on servers and minimal modification on browsers (by an add-on).
The remainder of the paper is as follows: Section 2 gives an overview of the design of BOSF, and Section 3 briefly discusses technical details of the concept-of-proof implementation. Section 4 provides an evaluation on BOSF, and Section 5 concludes the paper.

2 DESIGN

The design of BOSF is mainly based on an observation:

The owner (i.e. the developer) of a website/webpage usually knows what scripts are trusted and allowed to run on the website/webpage.

As scripts need to be added by their URLs, the locations are known to the owner. Thus, if the list of the locations of allowed scripts is recorded by the owner, it will be a perfect reference for browsers to identify trusted scripts. First, it requires little security knowledge from an owner, as long as the owner is conscious when adding a script. Second, it takes little extra effort to separately record it. Because of the triviality, it is even possible to do the recording automatically.

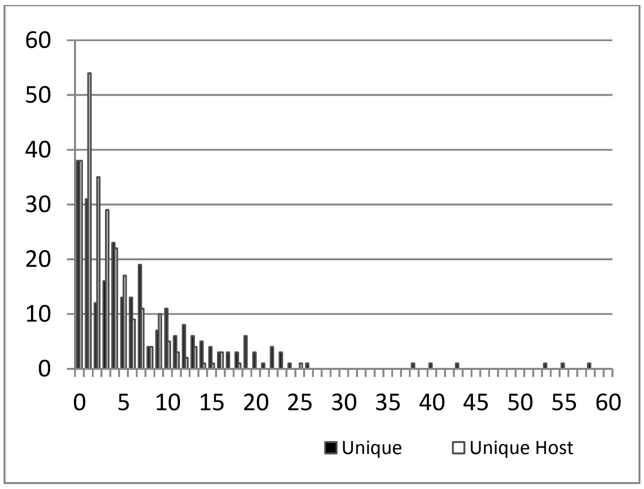

Figure 1. The top 250 sites on Alexa use 8 scripts from 4 hosts on average, 80% of the sites uses less than 12 scripts from less than 6 hosts.

To further reduce the burden on owners, the granularity is flexible, for example, several scripts under the same path can be listed as one. A survey on the top 250 sites on Alexa [17] (Figure 1) shows that the list is likely to be short. Third, the list is established by the owner, it is unlikely to introduce false alarms.
To embed the list into a webpage, a custom meta element [16] is inserted into the head element of the webpage. An example of the custom meta element is shown in Figure 2. Because a webpage is read sequentially, the custom meta is always available before any script is able to be executed.

```
<meta name="trusted" content="URL list"/>
```

Figure 2. A Custom meta Element

It is worth noting that though a custom meta element needs to be inserted into every webpages of a website, in most cases they are all the same and thus will not introduce complexity in the development of the website.
On the other hand, a user only needs to install an add-on on their browser in order to get protected by BOSF.
In short, the owner inserts a list of trusted locations into their webpages, the users install the add-on for their browsers, and all the rest is left to the add-on.
Later, when a user using a browser with the add-on installed visits a BOSF-secured website, the browser will read the custom meta element, extract the list of trusted locations, and decide whether a script is able to run or not based on the list.
To benefit more websites from BOSF, it is extended based on another observation:

A website/webpage and the locations of the scripts on the website/webpage (except scripts from Internet infrastructure such as Google APIs) usually belong to the same owner in reality.

Thus, we can construct a database of trusted locations for different websites based on the ownership in reality. Now, if extended (complete) BOSF does not find the custom meta element in the head element, it further consults the database.
In summary, the complete version of BOSF works in the way illustrated by Figure 3.
In the loading phase, BOSF reads through head element trying to find the custom meta element. If the meta element is found, BOSF constructs a list

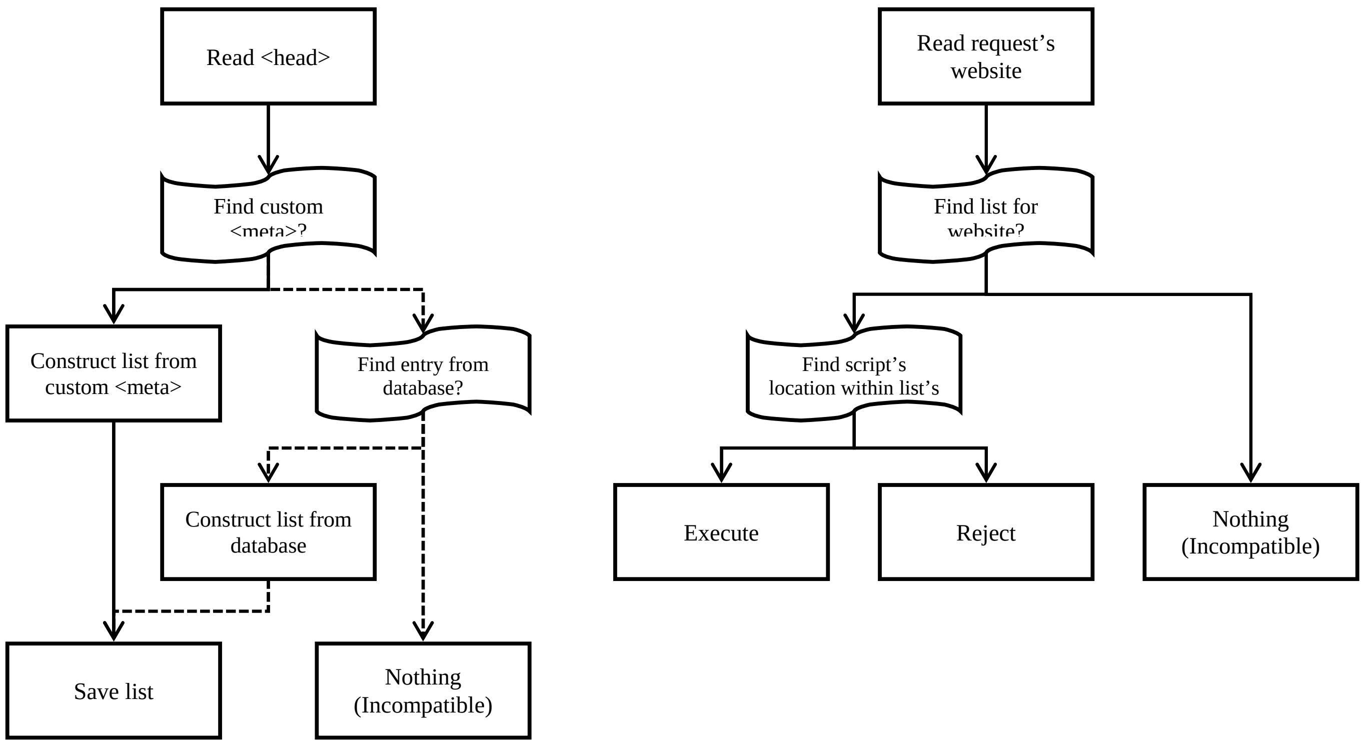

Figure 3. Workflow of BOSF. The left part is for the loading phase and the right part is for the processing phase. For the flow forks below a flag node, the left branch is for "yes" and the right is for "no".

based on the content extracted from the meta element. Otherwise, BOSF communicates with the database to whether there is a record for the request's website/webpage. If a valid record is returned by the database, a list is constructed according to it.

In the processing phase, whenever a script is pending to run, BOSF checks its active lists to see whether there exists one that matches the requester (website/webpage). If a matched list is found, whether the script is able to run is decided by whether the location of the script falls into the range of the list. Otherwise, BOSF just allows the script to run to avoid interruption for incompatible websites/webpages.

3 IMPLEMENTATION

We implement a prototype add-on for Firefox 9.0.1 as a concept of proof. Firefox provides an interface called nsIContentPolicy [15] to control loading and processing of different kinds of resources, including scripts, images, etc. By extending the interface, our modification on the flow of the loading and processing of scripts can be easily achieved. A hashmap is used for collecting all the lists and each list is implemented by an array.

4 EVALUATION

4.1 Effectiveness

According to our tests, BOSF is able to stop unauthorized scripts from processing and thus prevent XSS attacks. As every script needs to go through BOSF in order to run, it is an expectable result.

4.2 Performance

We mainly tested start-up and page-load overhead on the browser side. The test objects include: normal (Firefox with necessary add-ons and plug-ins), +empty (with an extra empty Firefox add-on, which may affects start-up performance), +BOSF-no-ext (with BOSF without database-based extension), +BOSF (with complete BOSF). All the tests were executed 10 times and the results were averaged.

Start-up performance illustrated in Figure 4 shows that the start-up overhead is ignorable. In the figure, the complete BOSF is even slightly better than BOSF without extension. This might be caused by systematic error, which further proves that the start-up overhead is ignorable.

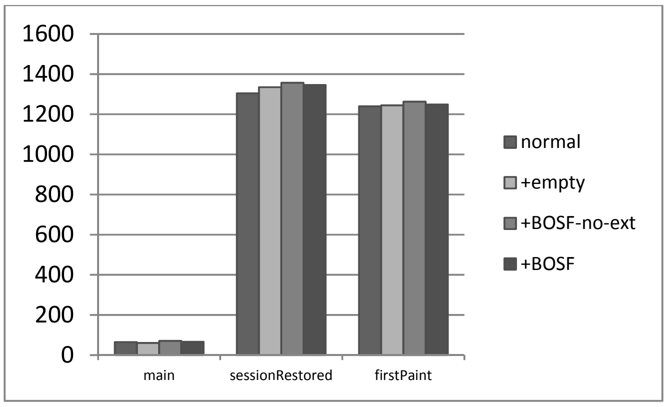

Figure 4. Start-up Performance. "main" means the time Firefox main function is entered; "sessionRestored" means the tabs in last session (the welcome tab) is created (not loaded); and "firstPaint": something is firstly painted in Firefox.

In order to obtain a better result, we crafted a page with 1000 scripts. The time took for 1000 scripts is shown in Figure 5. The approximate overhead was 100 milliseconds for BOSF without extension and 400 milliseconds for BOSF, or 0.1 milliseconds per script and 0.4 milliseconds per script respectively. In other words, for 80% websites (with less than 12 scripts), BOSF's overhead is no more than 5 milliseconds, which is imperceptible.

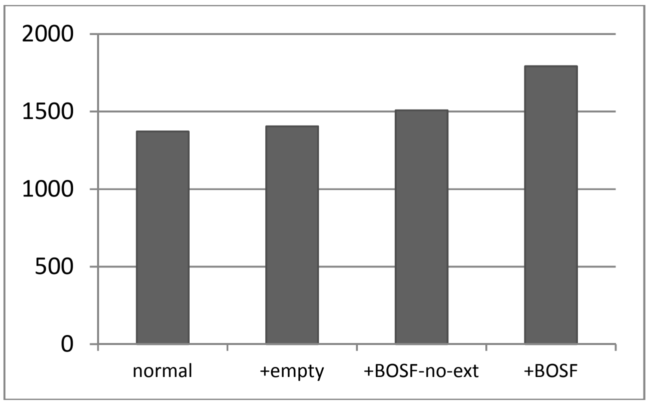

Figure 5. Page-load Performance (on a page with 1000 scripts).

On the server side, it is obvious that there will not be any overhead other than one extra element which usually occupies less than 1 kilobyte.

The transition to upgrade to BOSF-compatible may take some time as every page needs to be updated. However, it is once for all, and automatic upgrade is possible.

5 CONCLUSION

To improve usability of defense against XSS attacks, we invented location-based solution BOSF. BOSF is easy to understand and use for both developers and users, and is effective against XSS with minimal performance overhead

6 ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 20120006492).

7 REFERENCES

1. Jim, T., Swamy, N., Hicks, M.: Defeating Script Injection Attacks with Browser Enforced Embedded Policies. In: Proc. 2007 ACM Conference on World Wide Web, pp. 601--610, ACM Press, New York (2007).
2. Gundy, M., Chen, H.: Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-Site Scripting Attacks. In: Proc. 2009 Network and Distributed System Security Symposium (NDSS), pp. 612--628, Elsevier Press (2009).
3. Louw, M., Vecnkatakrishnan, V.: Blueprint: Robust Prevention of Cross-Site Scripting Attacks for Existing Browsers. In: Proc. 2009 IEEE Symposium on Security & Privacy, pp. 331--346, IEEE Press, Chicago (2009).
4. Johns, M.: SessionSafe: Implementing XSS Immune Session Handling Computer Security. In: Proc. 2006 European conference on Research in Computer Security (ESORICS), pp. 444--460, ACM Press, Berlin (2006).
5. Wurzinger, P., Platzer, C., Ludl, C., Kirda, E., Kruegel, C.: SWAP: Mitigating XSS Attacks Using a Reverse Proxy. In: Proc. 2009 ICSE Workshop on Software Engineering for Secure Systems (IWSESS), pp. 33--39, ACM Press, Washington (2009).
6. Kirda, E., Kruegel, G., Vigna, G., Jovanovic, N.: Noxes: a Client-side Solution for Mitigating Cross-site Scripting Attacks. In: Proc. 2006 ACM Symposium on Applied computing (SAC), pp. 330--337, ACM Press, New York (2006).
7. Ismail, O., Etoh, M., Kadobayashi, Y., Yamaguchi, S.: A Proposal and Implementation of Automatic Detection/Collection System for Cross-site Scripting Vulnerability. In: Proc. 2004 International Conference on Advanced Information Networking and

Applications (AINA), pp. 145--151, ACM/IEEE Press (2004).
8. Vogt, P., Nentwich, F., Nenad, F., Kirda, E., Kruegel, C., Vigna, G.: Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. In: Proc. 2007 Network and Distributed System Security Symposium (NDSS), San Diego (2007).
9. Livshits, V.B., Lam, M.S.: Finding Security Vulnerabilities in Java Applications with Static Analysis. In: Proc. 2005 USENIX Security Symposium (2005).
10. Jovanovic, N., Kruegel, C., Kirda, E.: Pixy: a Static Analysis Tool for Detecting Web Application Vulnerabilities. In: Proc. 2006 IEEE Symposium on Security Privacy, pp. 263--268, IEEE Press (2006).
11. Wassermann, G., Zhendong, S.: Static detection of cross-site scripting vulnerabilities. In: Proc. 2008 International Conference on Software Engineering (ICSE), pp. 171--180, ACM/IEEE Press (2008).
12. The Open Web Application Security Project (OWASP) Top Ten Project, https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.
13. XSS (Cross Site Scripting) Cheat Sheet, http://ha.ckers.org/xss.html.
14. Information Action Open Source Software No Script, http://noscript.net/.
15. Firefox nsIContentPolicy, https://developer.mozilla.org/en/nsIContentPolicy.
16. HTML5 4.2.5 The meta element, http://www.w3.org/TR/html5/semantics.html#the-meta-element.
17. Alexa Top Site, http://www.alexa.com/topsites.