XSSPro: XSS Attack Detection Proxy to Defend Social Networking Platforms

Pooja Chaudhary[1], B. B. Gupta[1(✉)], Chang Choi[2], and Kwok Tai Chui[3]

[1] Department of Computer Engineering, National Institute of Technology Kurukshetra, Kurukshetra, India
pooja.ch04@gmail.com, gupta.brij@gmail.com
[2] Gachon University, Seongnam-si, Republic of Korea
enduranceaura@gmail.com
[3] The Open University of Hong Kong, Kowloon, Hong Kong, China
jktchui@ouhk.edu.hk

Abstract. Social Platforms transpired as the fascinating attack surface to explode multitude of cyber-attacks as it facilitates sharing of personal and professional information. XSS vulnerability exists approximately in 80% of the social platforms. Hence, this paper presents an approach, XSSPro, to defend social networking platforms against XSS attacks. XSSPro operates through isolating the JavaScript code in the external file and performs decoding operation. The context of each injected JS code is identified and then similar scripts are grouped together to optimize the performance of XSSPro. Finally, extracted scripts are matched against the XSS attack vector repository to detect XSS attack. If matched then it is refined by using XSS APIs, otherwise, the response is XSS free and sent to the user. Experimental results revealed that XSSPro achieved an accuracy of 0.99 and is effective against thwarting XSS attack triggered using new features of the built-in code language with low false alarm rate.

Keywords: Cross site scripting (XSS) · Social networking platforms (SNPs) · XSS API · Code injection vulnerability · Malicious JS code

1 Introduction

In this era, everything and everyone around the globe is connected through the internet. This brings out the platform of opportunities and businesses to prosper and grow. This scenario is fueled with the inception of social networking platforms (SNPs) as it facilitates sharing of information that is publically visible to everyone on the network. As per a report, about 80% of daily active internet users visits their social accounts on a daily basis. SNP [1] offers a digital place to internet users where they own their social accounts, initiate new connection with other users on the network, post their personal or any other information that is shared with the connected ones. Fascinating social platform includes Facebook with around 2.7 billion active users [2], YouTube, WhatsApp, Instagram, Twitter to name a few. Since it is a treasure trove of useful information,

S. Chellappan et al. (Eds.): CSoNet 2020, LNCS 12575, pp. 411–422, 2020.
https://doi.org/10.1007/978-3-030-66046-8_34

hence the most enthralling platform for the attackers to abuse latent vulnerabilities [3] such as Cross Site Scripting (XSS). XSS [4, 5] comes from the family of code injection vulnerabilities in which attacker inserts malign script code into any web application. At the time when any user access the application then this script is rendered by the browser, resulting in XSS attack.

XSS attack has 3 distinct classes [5]: Stored XSS, Reflected XSS and DOM based XSS attack. In Stored XSS, attacker permanently stores the malign script code into the web applications with a goal to infect large number of users. Reflected XSS initiates when attacker send a crafted link with malign code to victim so that when he clicks the link then server reflects back the malicious code in the response which gets processed by the browser. DOM based XSS attack is trigged using scripts with hidden malicious code that makes illegitimate modifications in the DOM tree of the web page. This attack may be triggered with an intent to steal sensitive credentials of the victim or it may initiated as the initial step to launch more advanced and sophisticated cyber-attacks such as Distributed Denial-of-Service (DDoS) attack. Therefore, there is a need to emphasize on the solutions to alleviate the XSS attack on the social media platforms.

In this article, we design a XSS attack detection and mitigation approach, XSSPro that acts as the proxy between the browser and server to detect and prevent social media applications from XSS attack. To achieve this task, we extract and isolate the vulnerable/malign JavaScript (JS) code into external file. To bypass the traditional XSS filters, attackers use encoding of JS scripts, hence, we perform decoding followed by the identification of scripts background. It might be possible that attacker injects similar type of malicious scripts at the vulnerable points in the application; therefore, instead of handling each class of similar scripts individually, we implements script grouping using Levenshtein distance so that processing overhead can be reduced. Additionally, the decoded and grouped scripts are compared with the blacklisted XSS attack vectors. If match is found then, code refining is performed. Otherwise, HTTP response is XSS free and sent it to the user.

1.1 Literature Review

In this section, the key contributions of the various researchers have been highlighted briefly. Pelizzi [6] proposed a client-side XSS filter, XSSFilt, which could discover non-persistent XSS vulnerabilities. This filter identifies and thwarts portions of address URL from giving an appearance in web page. Galán et al. [7] designed a multi-agent system to defend against stored XSS attack through perform automatic scanning of the web applications. Gupta et al. [4] design a client-server framework to alleviate all kinds of XSS attack through performing runtime tracking and string matching operations of malicious scripts. However, it incurs processing overhead and time consumption. To mitigate DOM based XSS attack, variance between the expected web ape and received web page is identified to secure applications in cloud computing environment in [8].

Zhang et al. [9] design an approach using GMM models that are modeled using the dataset containing features corresponding to normal and injected payload web page. Rao et al. [10] proposed a technique named as XSSBuster, that mitigate all kinds of XSS attack through processing HTML and JavaScript code separately. However, still there are prevalent issues these techniques such as high processing and computational

overhead, high rate of false positive in attack detection, requiring major portion of code modifications at the browser and/or server side, and are not competent enough to handle DOM based XSS attack. Moreover, less attention is focused towards securing the social media platforms against XSS attack. Hence, to overcome some of the major challenges, authors have designed a proxy called as XSSPro to defend social networking platforms against XSS attack.

The layout of rest of the article is: Sect. 2 comprehensively describes the modules comprising our proposed approach. Section 3 highlights the implementation setup detail and provides details of performance assessment of proposed approach. Finally, Sect. 4 concludes our article.

2 Proposed Work

This section comprehensively describes the key modules of the XSS detection approach. The novelty of this approach lies in the fact that it not only identify and neutralizes the effects of simple XSS attack but it also recognized the encoded or obfuscated JS attack vectors. Next sub-section presents the abstract view of the proposed approach.

2.1 Abstract Design Outline

Authors have designed an approach named as XSSPro (XSS Proxy) to alleviate the XSS attack on the real world social networking platforms. The key objective of XSSPro is to transform the vulnerable HTTP response web page such that the new modified version maintains the application logic by moving entire JavaScript code to a separate file. Then, it analyzes these files to uproot latent attack pattern from untrusted input locations. Figure 1 depicts the abstracted version of XSSPro.

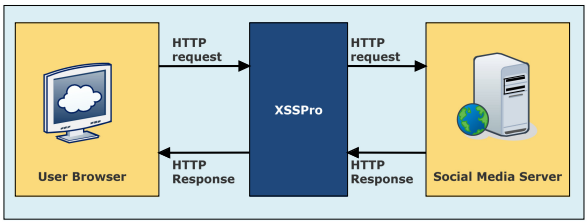

Fig. 1. Abstract design outline of XSSPro

XSSPro operates as the proxy between the client and server side to ignore the modifications at both side. It mainly performs following operations: a) parsing and extraction of JS code into separate external file; b) decoding of extracted JS code to identify partial script injection and obfuscated malign attack vectors; c) determining background information and grouping of extracted scripts; d) testing and refining of code. The entire working procedure of XSSPro is illustrated in the following section.

2.2 Detailed Design Outline

This section furnishes the comprehensive architectural detail of XSSPro. It discuss the sophisticated working of each module, how they process web page to produce the output in required format & how they support other modules to accomplish the respective goal. Figure 2 shows the detailed design overview of XSSPro. To accomplish the desired goal, XSSPro comprises following modules:

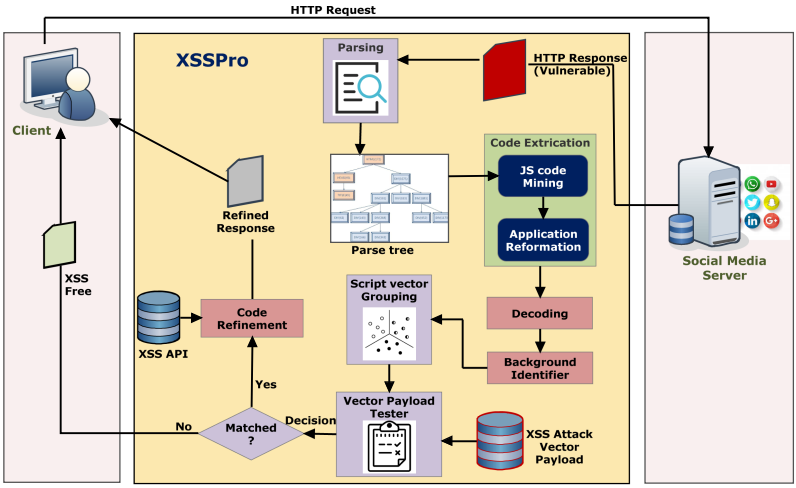

Fig. 2. Detailed design outline of XSSPro

Parsing. This module is the first component to receive the vulnerable HTTP response web page. It is responsible for construction of the Parsed Tree (PT) corresponding to that web page. It is to ensure that the browser renders the web page correctly. For instance, consider the code snippet in Table 1. Here, untrusted user input is applied at

Table 1. Sample Code snippet

<html>
<body>
Click Me!!!
<script>
function my() {
document.getElementByName(“val”).innerhtml = “hello” + “$_GET(‘name’)” + “you are” + “$_GET(‘age’)” + “years old”;}
</script>
</body></html>

S_GET('name') and $_GET('age'). The parse tree generated for the above code snippet is shown in Fig. 3. Each node of the tree represents HTML tags or text. This tree will be processed to determine script node embedded in to the web page.

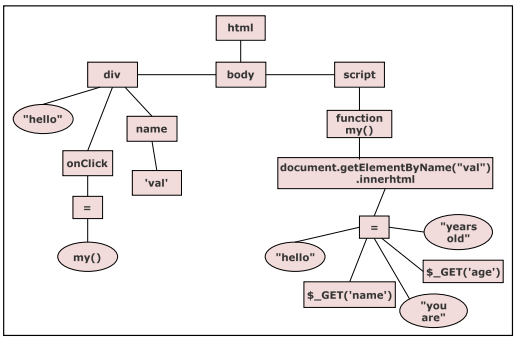

Fig. 3. Parsed tree of above code snippet

Code Extrication. In this module, the extracted parsed tree is processed for isolating the vulnerable JS code into the external file. This task is performed using 2 components: JS code mining and application reformation.

- *JS code mining:* This component is responsible for extracting the JS code from the parsed tree. To ease this task, we perform Depth First Search (DFS) between the nodes having value <script> and </script> . Each identified path denotes the JS code and is then forwarded to the next component which performs the application code rectification.
- *Application reformation:* This component performs the code reformation to achieve the JS code separation task. It receives the extracted JS code from JS code mining component. This component aims at shifting this code to a separate file say, JS_file. It then forwards this file and modified HTTP response web page to the next module. Algorithm 1 is implemented for code extrication.

Decoding. This module is responsible for applying the decoding operation on the extracted JS code. To bypass the traditional XSS filter, attacker employs smarter ways to inject malign code. In most of the cases, attacker use encoding of the attack vectors to forge deployed filters. Therefore, to detect such ambiguous, partially crafted and obfuscated attack vector, we perform decoding operation with respect to the determined encoding method. For instance, to hide `<script>` tag attacker may encode it as `&#60script&#62`. So to reverse this we conduct decoding.

Background Identifier. This component is responsible for the determination of the background information of the vulnerable source. It accepts the decoded vulnerable JS code and then uses it to determine the theme of the injected location in the web page. Algorithm 2 is implemented to identify the background information.

Script Vector Grouping. This module carries out the grouping of similar scripts. Attacker might inject the similar scripts at multiple locations.

Algorithm 1: Code Extrication

Input: Parse Tree (V, E)
Output: JS code external file and modified HTTP response

```
Start
JS_rep ← NULL;
For Each v ∈ V
  If (v.value == "<script>") then
       p ← DFS(v);
       While (p.value!= "</script>")
            JS_rep ← p.value ;
       End while
  End If
End For
For Each output statement
  X ← Check if output string is an HTML Tag;
  If (X) then
       For each (<script>… </script>)I pair ∈ JS_rep
         JS_file ← Content between (<script>… </script>);
            H_RES ← Include pointer to JS_file;
       End For Each
  End If
End For Each
Return JS_file, H_RES
End
```

Therefore, to reduce the time for code refining we conduct grouping of similar scripts. This component implements algorithm 3 for grouping the extracted attack vectors payloads depending on their similarity ratio. Consequently, a template is generated that describes the attack vectors in compressed form. Consider the example as shown below:

```
<script> alert(48a$bc); </script>
<script> alert(48xv&ez); </script>
```

Then the compressed template will be `<script> alert(48-S-); </script>` where S is the placeholder.

Vector Payload Tester . This module is responsible for identification of the XSS attack. It receives clustered JS attack vectors and it compares them with the externally available XSS attack vector payload repository. This repository contains the blacklisted attack vectors to trigger the XSS attack. If any match is identified between the blacklisted XSS vectors and extracted JS code then it indicates the presence of malicious attack vector and denote XSS attack. In this case, the modified response is sent to the code refinement module. Otherwise, the received response is XSS free and sent to the user without any modifications.

Algorithm 2: Background Identifier

Input: decoded JS code file
Output: Background information of each untrusted source.

Start
Background identifier: $BI_1 | BI_2 | \ldots | BI_N$;
$JS_{Dec} \leftarrow$ List of decoded JS code;
B_log $\leftarrow$ NULL;
For Each $S_I \in JS_{Dec}$
    $C_I \leftarrow BI(S_I)$;
    B_log $\leftarrow C_I \cup$ B_log ;
End For Each
For Each $C_I \in$ B_log
    If ($S_I \in$ String) **then**
      $f : C_I \mapsto$ String;
    Else if ($S_I \in$ Numeric) **then**
      $f : C_I \mapsto$ Numeric;
    Else if ($S_I \in$ Regular expression) **then**
      $f : C_I \mapsto$ RegExp;
    Else if ($S_I \in$ Literal) **then**
      $f : C_I \mapsto$ Literal;
    Else if ($S_I \in$ Variable) **then**
      $f : C_I \mapsto$ Variable;
    End If
End For Each
B_log $\leftarrow$ newvalue(C_I);
Return B_log
End

Code Refinement. This module accepts the identified malicious attack vectors as its input. It applies filtering to the malign script code to accomplish code refinement with the help of XSS Filtering APIs. This is done to halt the execution of injected malicious string and triggers malicious effects.

Next section elaborates in detail the implementation details and observed experimental results followed by the performance evaluation.

3 Experimental Outcomes and Assessment

In this section, we elaborate the implementation details and discuss the experimental evaluation of our proposed approach on different social networking platforms.

3.1 Implementation Layout

We have implemented XSSPro in java using Apache Tomcat server [11] as the backend, for mitigating the effect of XSS vulnerabilities. In this work, we have tested the efficacy

of the XSSPro on four distinct social networking platforms i.e. Oxwall [12], HumHub [13], Elgg [14], and Ning [15]. Initially, we verified the performance of the proposed approach against five open source available XSS attack repositories [16–20], which includes the list of old and new XSS attack vectors. Very few XSS attack vectors were able to bypass our proposed approach. We utilize HtmlUnit [21] HTML parser to create the parse tree for extraction of malicious JS code. To avoid the modifications at the client and server side, the XSSPro is designed to be implemented as the XSS detection proxy between browser and server. The experiment background is simulated with the help of a normal desktop system, comprising 3.2 GHz processor, 8 GB DDR RAM and Windows 7 operating system.

Algorithm 3: JS Attack payload vectors grouping

Input: JS payload vector with identified background information
Output: Grouped Template of Attack Vector Payloads

Threshold (δ):= 0;
Start
JS_file ⟵ list of traversed attack vectors;
G_Rep ⟵ NULL;
$V_l \longleftarrow 0$
For Each attack vector $A_l \in$ JS_file
    Equate(A_l, A_{l+1});
    $V_l \longleftarrow$ Levenshtein_distance(A_l, A_{l+1});
    If ($V_l >$ **δ)**
        Approve (A_l, A_{l+1});
        Create group template GT ∈ (A_l, A_{l+1});
        G_Rep ⟵ GT ∪ G_Rep;
    Else
        Abandon (A_l, A_{l+1});
        Select next pair (A_{l+1}, A_{l+2});
    End If
End For Each
Return G_Rep
End

3.2 Experimental Results

For the XSS detection, we inject the XSS attack vector payload at the injection points in the tested social platforms. This is achieved for evaluating the XSS attack detection capability on such platforms of applications. In terms of accuracy, we estimate what percent of "unsurprisingly arising" XSS attack vectors are alleviated by XSSPro. In terms of performance, we evaluate the performance-related issues of executing our framework on a variety of web page-loading and JavaScript standards. Figure 4 highlights the observed results of XSSPro on the four social networking platforms.

We have also calculated the XSS detection rate of XSSPro by dividing the number of attacks detected (i.e.# of True Positives) to the number of XSS attack vectors injected on each tested social platform. Figure 5 highlights the detection rate of all four applications.

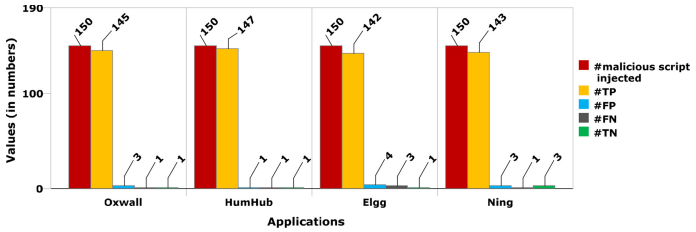

Fig. 4. Observed outcomes of XSSPro on different social platforms

Here, it is clearly reflected that the proposed approach is achieving the highest detection rate of 0.98 on HumHub social networking platform followed by the Oxwall and Ning platform with a detection rate of 0.97. Moreover, XSSPro is achieving a lowest detection rate of 0.95 on Elgg platform. In the next sub-section, we have evaluated the performance analysis of XSSPro through F-measure.

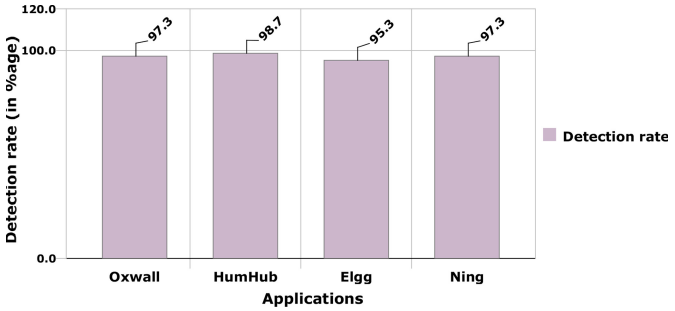

Fig. 5. Detection rate of XSSPro on different social platforms

3.3 Performance Assessment Using F-Measure

This section describes the performance evaluation of XSSPro using F-measure as the statistical analysis method. F-measure generally analyzes the performance of system by calculating the harmonic mean of precision and recall. The analysis conducted reveals that our approach exhibits high performance as the observed value of F-measure in all the social platforms is 0.9. Therefore, XSSPro exhibits 90% success rate in all the four social networking applications. Figure 6 displays the observed results of precision, recall, and F-measure.

$$precision = \frac{TruePositive(TP)}{TruePositive(TP) + FalsePositive(FP)} \tag{1}$$

$$recall = \frac{TruePositive(TP)}{TruePositive(TP) + FalseNegative(FN)} \tag{2}$$

$$F - measure = \frac{2 * precison * recall}{precision + Recall} \tag{3}$$

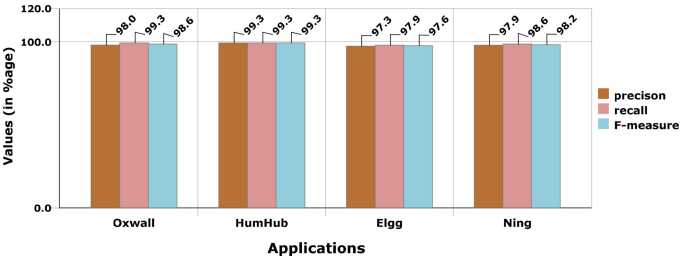

Fig. 6. Performance assessment evaluation outcomes

$$FPR = \frac{FalsePositive(FP)}{FalsePositive(FP) + TrueNegative(TN)} \tag{4}$$

Figure 7 represents the observed False Positive Rate (FPR) of XSSPro on each tested social networking platform. Although, XSSPro is achieving the similar False Positive rate on HumHub and Ning social platform, nevertheless, it achieves the highest detection rate of 0.99 on Humhub and lowest on Ning i.e. 0.97.

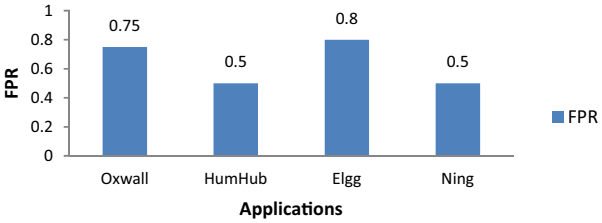

Fig. 7. Observed FPR on different tested platforms

3.4 Comparative Assessment

Here, authors have compared the XSSPro (proposed approach) with the other approaches stated in the literature. Table 2 shed highlights the comparative study on the basis of some pre-defined parameters: XSS Class defended (XCD indicates XSS class defended- S (stored), R (Reflected), and D (DOM-based), Partial injection detection (PID indicates partial malicious code injection detection), Obfuscated code detection (OCD indicates detection of ambiguous script code injection), code amendments (CA indicates any kind of alterations at client or server side), script background identification (SBI indicates identification of background of injected malicious code before refining the response).

Table 2. Comparative analysis of our approach with existing literature

Parameters →	XCD			PID	OCD	CA		SBI
Approaches ↓	S	R	D			Client	Server	
[6]	×	√	×	×	×	×	√	×
[7]	√	×	×	×	√	×	√	√
[4]	√	√	×	√	√	√	√	×
[8]	×	×	√	√	×	√	×	√
[9]	√	√	×	×	×	√	×	×
[10]	√	√	×	√	×	√	×	×
XSSPro	√	√	×	√	√	×	×	√

4 Conclusion

Indeed, Social networking platforms are treasure troves of personal and professional information. Thereby, it is one of the most captivating attack surfaces for the adversaries to stimulate multiple attacks such as XSS. In this article, authors have designed XSSPro, to alleviate XSS attack on real world social networking sites. It performs extraction and decoding of malicious JS Code and then resolves the background details of these scripts. Additionally, it implements scripts grouping based on the Levenshtein distance to reduce the time for code refining in HTTP response. Finally, it equates extracted JS code with the blacklisted XSS attack vectors. If attack vector identified then it executes code refinement with the help of XSS API otherwise, response is XSS free and forwarded to the user. The experimental results has proclaimed the efficiency of XSSPro in detecting new XSS attack vector payload without requiring any kind of alterations at the browser or server side. As a part of our future work, we will attempt to integrate XSSPro for mitigating the effects of XSS against applications in the mobile cloud computing environment and moreover we will enhance the capabilities of XSSPro to defend against DOM based XSS attack.

References

1. Fire, M., Goldschmidt, R., Elovici, Y.: Online social networks: threats and solutions. IEEE Commun. Surv. Tutorials **16**(4), 2019–2036 (2014)
2. Gupta, B.B., Gupta, S., Gangwar, S., Kumar, M., Meena, P.K.: Cross-site scripting (XSS) abuse and defense: exploitation on several testing bed environments and its defense. J. Inf. Priv. Secur. **11**(2), 118–136 (2015)
3. Sahoo, S.R., Gupta, B.B.: Classification of various attacks and their defence mechanism in online social networks: a survey. Enterp. Inf. Syst. **13**(6), 832–864 (2019)
4. Gupta, S., Gupta, B.B., Chaudhary, P.: A client-server JavaScript code rewriting-based framework to detect the XSS worms from online social network. Concurr. Comput. Pract. Exper. **31**(21), e4646 (2019)

5. Rodríguez, G.E., Torres, J.G., Flores, P., Benavides, D.E.: Cross-site scripting (XSS) attacks and mitigation: a survey. Comput. Netw. **166**, 106960 (2020)
6. Pelizzi, R., Sekar, R.: Protection, usability and improvements in reflected XSS filters. In: Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, Seoul, Korea (2012)
7. Galán, E., Alcaide, A., Orfila, A., Blasco, J.: A multi-agent scanner to detect stored-XSS vulnerabilities. In: 2010 International Conference for Internet Technology and Secured Transactions pp. 1–6. IEEE November 2010
8. Chaudhary, P., Gupta, B.B., Yamaguchi, S.: XSS detection with automatic view isolation on online social network. In: 2016 IEEE 5th Global Conference on Consumer Electronics, pp. 1–5. IEEE October 2016
9. Zhang, J., Jou, Y.T., Li, X.: Cross-site scripting (XSS) detection integrating evidences in multiple stages. In: Proceedings of the 52nd Hawaii International Conference on System Sciences January 2019
10. Rao, K.S., Jain, N., Limaje, N., Gupta, A., Jain, M., Menezes, B.: Two for the price of one: a combined browser defense against XSS and clickjacking. In: 2016 International Conference on Computing, Networking and Communications (ICNC), pp. 1–6. IEEE February 2016
11. Apache tomcat server. https://tomcat.apache.org/download-80.cgi
12. Oxwall social networking platform. https://developers.oxwall.com/download
13. Humhub social networking site. https://www.humhub.org/en
14. Elgg social networking engine. https://elgg.org
15. Ning: social networking platform. https://www.ning.com/
16. Rsnake. XSS Cheat Sheet 2008. http://ha.ckers.org/xss.html
17. HTML5 Security Cheat Sheet. http://html5sec.org/
18. XSS vectors available. http://xss2.technomancie.net/vectors/
19. Gupta, S., Gupta, B.: PHP-sensor: a prototype method to discover workflow violation and XSS vulnerabilities in PHP web applications. In: Proceedings of the 12th ACM International Conference on Computing Frontiers, pp. 1–8 (2015)
20. @XSS Vector Twitter Account. https://twitter.com/XSSVector
21. HtmlUnit parser. https://sourceforge.net/projects/htmlunit/files/htmlunit/