L-WMxD: Lexical based Webmail XSS Discoverer

Zhishou Tang, Haojin Zhu, Zhenfu Cao, Shuai Zhao

Abstract—XSS (Cross-Site Scripting) is a major security threat for web applications. Due to lack of source code of web application, fuzz technique has become a popular approach to discover XSS in web application except Webmail.

This paper proposes a Webmail XSS fuzzer called L-WMxD (Lexical based Webmail XSS Discoverer). L-WMxD, which works on a lexical based mutation engine, is an active defense system to discover XSS before the Webmail application is online for service. The engine is initialized by normal JavaScript code called seed. Then, rules are applied to the sensitive strings in the seed which are picked out through a lexical parser. After that, the mutation engine issues multiple test cases. Newly-generated test cases are used for XSS test. Two prototype tools are realized by us to send the newly-generated test cases to various Webmail servers to discover XSS vulnerability. Experimental results of L-WMxD are quite encouraging. We have run L-WMxD over 26 real-world Webmail applications and found vulnerabilities in 21 Webmail services, including some of the most widely used Yahoo!Mail, Mirapoint Webmail and ORACLE’ Collaboration Suite Mail.

Keywords—Webmail, XSS, fuzzer, L-WMxD

I. INTRODUCTION

XSS is one of the most popular vulnerabilities in web application. There are three types of known XSS flaws [1–3]: 1) Stored XSS, 2) Reflected XSS, 3) DOM based XSS, among which Stored XSS vulnerability always allows the most powerful attacks. During this type of attack, attack vectors are submitted to web server and stored on the server (in a database, file system or other locations). When other users request this data and attack vectors are displayed on their browser without sufficient checks, an attack may happen. Hackers use XSS vulnerability to gain access to victims’ browser for identification, e-mail or password. Because the malicious script running under the context of current user, firewall, encryption method or IDS (intrusion detection system) are ineffective in preventing this type of attack. As described in OWASP TOP10 security risks (both 2007 [4] and 2010 [5]), XSS is one of the most critical risks.

In order to find vulnerability of web application, two approaches are mainly used: “White Box” and “Black Box”.

The “White Box” approach needs source code of the web application. This approach can be realized manually or by using automatic code analysis tools, such as Fortify SCA [6], Pixy [7]. Since manual code review is time-consuming, error-prone, costly and hard to evaluate, static analysis tools are more preferred. Most static analysis tools use the source code of web application as input, and then traversal all the possible paths through the manipulation of the XSS attack vectors and finally the detection results are presented.

The “Black Box” test is now commonly used by many commercial systems, including AppScan [8], Acunetix [9] and Nessus [10]. These tools inject attack vectors to the web application’s HTML form elements or URL parameters and get feedback immediately. If input validation module of a web application can’t sufficiently filter attack vectors, these tools can discover vulnerability of the test target. This kind of test procedure is fit for testing Reflected XSS, but not Stored XSS. For most Stored XSS, the malicious input is stored in the database and later included in another but not the immediately response. This is an obstacle for current existing tools to make the analysis procedure automatically, so, they principally focus on URL-based XSS like Reflected XSS detection. Furthermore, “Black Box” test can’t take the advantage of the logic of the program, it is not efficient, and can not guarantee the coverage. Fonseca’s [11] study shows that the percentage of false positives of current widely used tools is very high, ranging from 20% to 77%. However, the advantage of “Black Box” test is that it is not sensitive to the implementation programming language of the web application, which makes it ideal for wide deployment.

Both of the “White Box” and “Black Box” methods need to generate test cases to verify the ability of the XSS discoverer. Model checking and symbolic execution are the two methods now used to generate test cases, but these methodologies are also based on source code of web application.

“White Box” test is hard to be performed, because Webmail service vendors are reluctant to release their Webmail source code. Therefore, “Black Box” test approach is finally adopted by us to discover Webmail’s vulnerability. Through comprehensive consideration of different Webmail’s filter mechanism, we make corresponding rules for mutation to generate test cases. The results show that our method is quite effective in finding Webmail XSS Vulnerability.

This paper makes the following contribution:

• Firstly introduce a semi-automatic XSS detector to Webmail. Until now, there is no publicly released Webmail XSS detector. Due to the attribute of Webmail, most of the Webmail XSS discoverer job is now done by labor. We firstly introduce a semi-automatic fuzz technique to Webmail XSS discoverer and achieve a good success rate.

• Deliberately selected seeds and rules. Seeds we used can widely cover most possible usage of JavaScript in HTML. Based on well known documents and experiment, rules are selected deliberately and made by sufficient observation of various kinds of Webmail applications.
They are set on the assumption of the possible defects of a Webmail filter module, which should be responsible for finding scripting commands or meta-characters in untrusted inputs and filtering any such contents before these inputs are processed by the web applications and returned to users’ browser.

- **A lexical-based test case generating engine.** Unlike other XSS fuzz methods which use pre-defined test cases to discover the vulnerability of Web application, we make different rules corresponding to different filter’s strategies of Webmail server. Our Rules work on a lexical-based parser to translate the sensitive areas of the input seeds into different test cases.

The rest of the paper is organized as follows. Section II describes the target we want to defeat and the seeds, rules we used for generating XSS attack vectors. In section III, prototype tools for testing are described. We present experimental results in section IV. In this section, a brief analysis will be followed after the list of vulnerabilities our tools find. Section V presents related work on discovering and detecting XSS attacks. Finally, in section VI, we present our conclusion and future work.

### II. Fundamental Design Principles

In this section, first we give a brief overview of seeds and discussion about rules which are used for generating test cases. Then we will discuss the lexical parser used, through which rules can be employed. The relationship of the Webmail’s filter and rules is presented by Figure 1. On the server side, the filter module is divided into Front-End ($F_x$) and Back-End. The Front-End is designed to transform original user input data to a unified expression, then the Back-End of filter module can apply XSS attacks detection using fix signature. Therefore, if our mutated data can defeat the Front-End of the filter module, it is likely to bypass the Back-End. We design rules ($R_x$), which are applied to the seeds to generate new test cases to bypass the Front-End of a Webmail’s filter module. The filter’s strategies for transforming the original input data and the rules we used will be discussed in detail in the following subsections.

#### A. Seeds selected

After investigating more than 100 XSS samples, which are collected from previous XSS in the wild like Yahoo!Mail and Hotmail XSS, we select a serial of seeds deliberately. Each seed is a standard type of XSS attack vector, such as pure JavaScript, HTML tags, HTML attributes, events, Flash and ActiveX. $S_x$ represents the seed we used and some of the seeds are listed in Table I.

#### B. Rules for mutation

Considering most Webmail systems have their own mechanisms for sensitive string identification, like “JavaScript”, “expression”. We mainly apply rules to sensitive strings in order to make mutation escape capturing by the Webmail filter. Because sensitive strings always have some common features, we collect them in advance and build a database to hold them.

The rules discussed in the following subsection can be applied to the sensitive strings, then we check whether they can bypass the Webmail filter mechanism.

Currently, there are six rules mainly used for making mutation, we present them from $R_1$ to $R_6$.

1) $R_{1x}$, ASCII $\rightarrow$ HEX/DEC transform: This rule is used to transform ASCII encoded string to HEX/DEC encoded string. The HEX/DEC encoding formats are HEX value with semicolons and DEC value without semicolons. Table II lists the different strategies the filter may take when dealing with HEX/DEC encoded string. The strategies are made by assumption. Table III is the corresponding rules we take.

The relation of $F_{1x}$, $R_{1x}$, and the result is:

$$\text{XSS} = (F_{11} \land R_{11}) \lor (F_{12} \land R_{12}) \lor (F_{13} \land R_{13}) \lor (F_{12} \land R_{12}) \lor (F_{13} \land R_{13}) \lor (F_{13} \land R_{13})$$

If server can transform mixed type of encoding value from HEX and DEC to ASCII, our $R_{1x}$ rule will not defeat the filter mechanism.

#### Table I

<table>
<thead>
<tr>
<th>Seed</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td><code>&lt;script&gt;alert('xss')&lt;/script&gt;</code></td>
</tr>
<tr>
<td>$S_2$</td>
<td><code>&lt;img src=&quot;javascript:alert('xss')&quot;&gt;</code></td>
</tr>
<tr>
<td>$S_3$</td>
<td><code>&lt;img style=&quot;xss:expression(alert('xss'))&quot;&gt;</code></td>
</tr>
<tr>
<td>$S_4$</td>
<td><code>&lt;body onload=&quot;alert('xss')&quot;&gt;</code></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#### Table II

<table>
<thead>
<tr>
<th>Filter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_{11}$</td>
<td>The mail server does nothing on the mail.</td>
</tr>
<tr>
<td>$F_{12}$</td>
<td>The mail server employs a weak filter, it transforms HEX value with semicolons to ASCII string and checks sensitive string in it.</td>
</tr>
<tr>
<td>$F_{13}$</td>
<td>The mail server employs a weak filter, it transforms DEC value without semicolons to ASCII string and checks sensitive string in it.</td>
</tr>
</tbody>
</table>

#### Table III

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_{11}$</td>
<td>HEX value with semicolons encoding. Using this rule, we can bypass $F_{11}$ and $F_{12}$.</td>
</tr>
<tr>
<td>$R_{12}$</td>
<td>DEC value without semicolons encoding. Using this rule, we can bypass $F_{11}$ and $F_{12}$.</td>
</tr>
<tr>
<td>$R_{13}$</td>
<td>Mixed encoding with HEX and DEC values. Using this rule, we can bypass $F_{11}$, $F_{12}$ and $F_{13}$.</td>
</tr>
</tbody>
</table>
The transformed string can work well under Internet Explorer. If the mail server's filter can not deal with the inserted new line in seed, a new `ENTER` may help us to escape capturing by the filter. The places where a new line can be inserted have some features in common. For example: `ENTER` can not be inserted into the place between the starting of tags or attributes (characterized by `'<` symbol) and the first letter, but can exist between the last letter and the closure of tags or attributes (characterized by `'>` symbol). There should be no `ENTER` in a string, which is presented by closure of double quotation marks. To identify a string, we need the assistance of a lexical parser which will be discussed later.

4) \( R_4 \), lower→upper case transform: `HTML` language does not require lower or upper case strictly, but JavaScript language is case sensitive. Therefore, the mixed `HTML`, JavaScript and their different requirement for case encoding will make Webmail filter puzzled. For example, "expression" and "EXPRESSION" will all be accepted by `IE`, but if Webmail’s filter do not sensitive to this change, our transformed attack vectors will successfully bypass the Webmail’s filter. So \( R_4 \) rule mainly makes case change on sensitive string of `HTML` language.

5) \( R_5 \), ANSI encoding→Other encoding transform: There are many types of encoding, like ANSI, UNICODE, UNICODE of big endian and so on. `IE` accepts all these kinds of encoding formats. But if the server filter can not deal with these transformation, our attack vectors will reach victim’s browser successfully.

6) \( R_6 \), DBC case→SBC case transform: Lower version of `IE` is likely to accept SBC case character, so we transform DBC case character to SBC case. If Webmail server does not deal with SBC case character, our attack vectors will execute on user’s browser eventually. This means, even sensitive string “JavaScript” is filtered, “JavaScript” may not be captured.

7) Mix multiple rules: Multiple rules can be applied on the same seed, newly generated test cases \( M_1 \) can be used for rotate mutation. Following is the mutation algorithm:

\[
M_{i+1} = R_i(M_i)(i = 1 \sim 6, R \in \{ R_1, R_2, R_3, R_4, R_5, R_6 \}, \\
M_0 = S_j)
\]

To avoid the cases that the Front-End of a Webmail filter does not apply decoding procedure but simply dropping unknown encoded string, the algorithm we design do abundant job. For example, if we apply \( R_1 \) rule to the sensitive string “expression” to get string \( M_{11} \): “&x65;xpression”, and the Webmail filter drops the transformed string for the unrecognized string format, then the next string “&#x65;&#120;pression” based on \( M_{11} \) and \( R_1 \) is also dropped by the server. So, we adjust the sequence of test cases generation to make the test cases take effect. The test case generation procedure can be presented as following:

```java
SeedGeneration (seed) {
    curRule = 1; 
    if (curRule == 1) 
        
    TestCase = Rule1(seed); 
```
C. lexical parser

As described above, rules can not be applied to certain part of seed, so a lexical parser is needed for identifying sensitive strings or tags of HTML or JavaScript language. The parser we used is described in Figure 3.

At end point VT1, a string is identified, almost all rules can be applied to the string; A tag of HTML is found at end point VT2, R3 rule can be applied between the last character of a string and the “>” symbol. Considering certain mail areas do not accept multi-lines input, test cases generated by R3 normally should not be applied to these areas, like “Subject” area.

III. IMPLEMENTATION

We have implemented two prototype tools to send new test cases generated by L-WMxD to Webmail server.

1. An automatic mail sender built on SMTP protocol is developed to make a comprehensive test. This tool has the following characters to support the rules we discussed above:
   - Test areas include not only mail body, subject, but also sender name, receiver name, CC name, etc.
   - Multiple test case encoding types for selection.
   - For most of Webmail server can not tackle multi-line text of subject, sender name, receiver name or CC name areas, these kinds of messages will be rejected at once when received. Therefore, it is programmed as optional for us to decide whether to apply R3 to them or not.
   - Most mail server set anti-spam mechanism and this will interrupt the procedure of testing, we choose gmail as our relay server. Since gmail server needs SSL connection, we add SSL support for this tool.

2. A Paros [12]-based tool is developed to make an interception. Using this tool, we can modify and resend the captured message. Paros is powerful to deal with HTTPS protocol. Using a fake certification, Paros can get plain text of the encrypted data over HTTPS. This tool can also help us to do interception when testing Microblog or web-based online chat service.

IV. EVALUATION

Some special mail systems are hard to register (identification card number needed, etc). In order to test these Webmail systems, we use hacking tool HScan to guess week password. Through this method, we gain password of certain Webmail user account, then we can do the subsequent tests on it.

Another trick is adopted to deal with the case that weak password is not available. We call this trick “blind injection”. For example, we transform the functions of test cases from invoking a message box to downloading an empty text file residing on our server. The test cases are sent to a Webmail server and then the Webmail user interacts with the new coming message, after that, a request may happen on our server. If there is a request for the empty text file, we think XSS attack may affect this kind of Webmail application.

L-WMxD has been applied to 26 real-world Webmail services using our tools and the discovered XSS is verified by the latest web browser: Internet Explorer 8.0.7600.16385. The XSS we found and L-WMxD’s performance will be discussed in the following.

A. Found vulnerabilities

Table VI shows XSS we have discovered by testing. As for Webmail, sending message and receiving the same message are asynchronous. This means that a sequence of interactions are needed when viewing a new coming message, and more labor work is needed when making the audit phase automatic. Till now, we simply check mails one by one after sending all test cases, and manually confirm whether there is a vulnerability (a message box or a desired request appeared through an unmodified Paros). If there is, we classify it as find-xss. As for “blind injection” test method, we are not sure if this request is made by a Webmail user or XSS, this type of request is classified as unknown. Others, test procedures which are terminated in advanced (because there are too much test cases generated) are grouped as none-xss. Test results show that we have found XSS vulnerability in 21 Webmail service and 1 Webmail service is likely to have defects.

B. Case Study

Although we have informed all the vendors, most of the XSS we discovered keeps the state of un-patched. Till now, we are providing supports to the Webmail vendors to fix their problems, ChinaUnionPay for example. Therefore, we will only give a brief description and the test cases will not be presented.

1) Test result using R13 rule: Through R13 rule, we find XSS vulnerability in Mirapoint Webmail 3.10.6, Mirapoint Webmail 3.10.8, mail.126.com, mail.163.com, etc.
<table>
<thead>
<tr>
<th>Webmail</th>
<th>find-XSS</th>
<th>none-XSS</th>
<th>unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>mail.eastday.com</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mail.186sh.com</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mail.263.com</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mail.139.com</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mail.tom.com</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mail.ru</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmail.com</td>
<td></td>
<td>✔️</td>
<td>-</td>
</tr>
<tr>
<td>KOAL secure mail</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>freemail.ru</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>ORACLE’ Collaboration Suite Mail</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>freemail.eyou.com</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Mirapoint 3.10.x</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>mail.qq.com</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>mail.aol.com</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Total</td>
<td>21</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

TABLE VI
Webmail vulnerability that have been discovered by L-WMxD

2) Test result using R22 rule: By using R21 on seed generation we discover XSS vulnerability in mail.186sh.com, mail.139.com, mail.133sh.com, mail.vip.tom.com, etc.

3) Test result using R3 rule: Test result shows that R3 is very effective in discovering XSS vulnerability. More than 10 Webmail services are defective in filtering such kind of attacks, such as mail.wo.com.cn, mail.163.com. Some of them are patched without the public announcement on March 2010.

4) Test result using R4 rule: R4 rule has been used to discover vulnerability in mail.21cn.com.

5) Test result using R5 rule: Using rule R5, we discover XSS in mail.ru and “KOAL secure mail”, it’s likely that they can not deal with message encoded with UNICODE.

6) Test result using R6 rule: Since R6 need to be tested under elder version of IE, we don’t do much test on it.

7) Others: Taking advantage of the tools we developed based on L-WMxD, we can perform a comprehensive test. Through the test procedure, we discover XSS in mail.sohu.com, freemail.eyou.com and so on. For example, when attack cases are applied to “title” area of a message, we discover vulnerability of mail.sohu.com even with pure seed (seed is send directly without transformation). We believe such area of a message is ignored by filter mechanism for years.

Distribution of XSS we have found when using corresponding rule can be presented in Figure 4.

V. RELATED WORK

Nowadays, most XSS detection tools rely on source code of web applications. Techniques employed by such tools include control-flow, data-flow or formal method. Considering the stage that the XSS detection method is employed, we classify current work into “Active Defense” and “Passive Defense”. “Active Defense” means discovering flaws of software in advance before it is publicly released. “Passive Defense” means that discovering mechanism is applied to online software to detect XSS attack in real time. Michelle Ruse [13] has done a good job of collecting latest research about XSS attack detection.

A. Active Defense

MUTEC [3] applies mutation on PHP source code of a product to verify whether a test case can bypass a filter. Pixy [7] uses flow-sensitive, interprocedural and context-sensitive data flow analysis to discover vulnerable points in a PHP program. In addition, alias and literal analysis are employed to improve the correctness and precision of the results. QED [14] is a goal-directed model-checking system (based on Java PathFinder Model Checking system) that automatically generates attacks in standard Java web applications. It uses programmable technique to automatically generate attacks for large web-based applications.

B. Passive Defense

The “Passive Defense” approach needs intervention with original web applications, which will have an impact on robustness and stability of the application potentially. Further, it also specifies the language in which an application is written. “Passive Defense” mainly resides on client side [15]–[18] and most work focuses on “Cross Site Request Forgery (CSRF). Using technique of dynamic data tainting and static analysis, this kind of defending technique can avoid XSS attacks effectively. Since it is hard to deploy such technique to every browser, it’s better to find vulnerability in advance.

Most current methods have difficulty in verifying the true and false of the XSS found (because simulating the real world browser is very difficult), which leads to high false positives [11]. XSSDS [1], XSS-GUARD [19] use Firefox components like rbNarcissus [20] to precisely identify scripts in a web page. Wassermann’s [21] work is based on tainted information flow with string analysis. He also checks whether un-trusted...
parts of the document can invoke the JavaScript interpreter or not. Such a method is a good remedy for the shortcoming of current identification problem. However, for IE, it’s not an open source software and is hard to decide whether certain JavaScript can run on it.

C. Test case generation

Test case generation is commonly considered as a criterion for code coverage. Well generated test cases can test more branches of a program. Dynamic execution is used for attack vectors generation [22]–[25]. For example, Kiezun et al [23] presents a dynamic tool, Ardilla, to create SQL and XSS attacks. This tool uses dynamic tainting, concolic execution for attack-candidate generation and validation. Moreover, Kudzu [25] introduces a string constraint solver for JavaScript to perform symbolic execution in-depth.

VI. CONCLUSION AND FUTURE WORK

An effective test of XSS vulnerability helps to fix implementation early and decrease losses incurred for the end users. Experimental test shows that XSS vulnerability extensively exist in Webmail service and our L-WMxD is effective in discovering it. We will continue the test procedure using tool derived from Paros to discover XSS vulnerability in other web applications like online chat system which is used by Google, AOL, etc.

As noted above, it is hard to evaluate the coverage when using “Black Box” testing. We also notice some attack vectors from existing attack like hotmail XSS (patched):

```<style>p{height:expression((window.rrr==914)?xxx=8:(eval(code.title)==88)||((rrr==914))?,80,180):}!</style><p><img id="code" width=1 height=1 title="#emailkey=‘emailID23456’;var cc=alert(‘x’);eval(cc);"></p>
```

The above attack vector can bypass hotmail filter. When it reaches victim’s browser, IE also accepts such encoding. This is an obstacle for most of XSS discover methods, because it’s difficult for them to generate such a complex test case. In our opinion, the combination of symbolic execution on server side (based on source code of Webmail to bypass filter mechanism) and client side (based on HTML parser to reach the JavaScript interpreter) can generate such kind of attack vector automatically. We will apply symbolic execution to find Webmail XSS vulnerability for future work.

ACKNOWLEDGEMENT

This research is supported by National Natural Science Foundation of China (Grant No. 61003218 and No. 61033014), Doctoral Fund of Ministry of Education of China (Grant No.20100073120065), Natural Science Foundation of Hohai University under Grant No. 2009427811, Science and Technology Commission of Shanghai, (Grant No.10511501503), and Opening Project of Key Lab of Information Network Security of Ministry of Public Security (The Third Research Institute of Ministry of Public Security).

REFERENCES