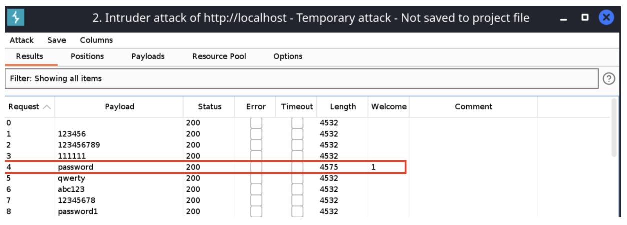
Damn Vulnerable Web Application (DVWA)

1. Brute Force

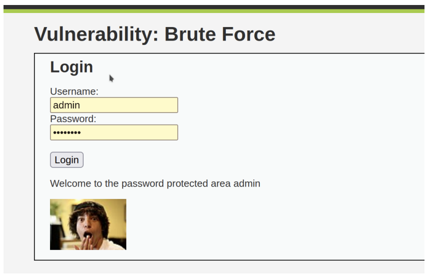
Endpoint: http://localhost/DVWA/vulnerabilities/brute/
Method: Brute force attack



2. Intruder attack of http://localhost - Temporary attack - Not saved to project file

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Welcome	Comment
0		200	☐	☐	4532		
1	123456	200	☐	☐	4532		
2	123456789	200	☐	☐	4532		
3	111111	200	☐	☐	4532		
4	password	200	☐	☐	4575	1	
5	qwerty	200	☐	☐	4532		
6	abc123	200	☐	☐	4532		
7	12345678	200	☐	☐	4532		
8	password1	200	☐	☐	4532		



Vulnerability: Brute Force

Login

Username: admin
Password: ••••••••

Welcome to the password protected area admin

Description:
A brute force attack can take various forms. The most common one involves an attacker specifying specified parameters, sending requests to a server with those numbers, and then evaluating the result. An attacker may utilize a dictionary attack or a standard brute-force attack with certain types of characters such as alphanumeric, special, or case insensitive for efficiency. In this lab, the developer has entirely overlooked all security measures, allowing anyone to attempt to log in to any user as many times as they like without facing any consequences.

Risk score: High
(Exploitability: 3, Weakness prevalence: 3, Weakness detectability: 3, Technical impact: 3)

Impact:
Users and businesses may experience wide-ranging impacts as a result of the brute force attack. They comprise:

1. Identity theft: stealing a person's identity in order to gain access to their accounts, such as bank accounts or credit cards This allows the attacker to use these details to purchase things. Furthermore, information like social security numbers might be sold and used in further cyberattacks.
2. Loss of data: if data is stolen, the company's reputation may suffer. Furthermore, there may be reputational damage as a result of a leak of sensitive client information, which leads to public distrust and unhappiness with the company.
3. Downtime: this term refers to system outages caused by a cyber attack on websites or computer networks. This is costly to the company in terms of lost income, customer happiness, and image.

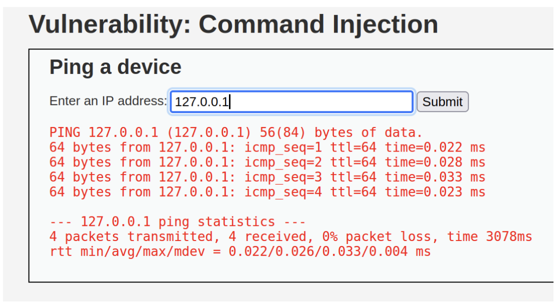
Fix / mitigation method:
The developer should introduce a "lock out" feature, which prevents the locked out user from logging in if there have been a certain number of failed logins at the limit of time (for example 5 failed log in the last 15 minutes).

If the locked-out user attempts to login, even with a valid password, it will report that their username or password is invalid. This makes it impossible to determine whether or not the system has a genuine account with that password and whether or not the account is locked. By repeatedly attempting to login to someone's account, this can result in a "Denial of Service" (DoS). This level would need to be raised by blacklisting the perpetrator (e.g., IP address, country, user agent).

In addition, using defensive tools is another way to detect the use of brute force tools such as WFuzz, OWASP DirBuster, and vulnerability scanners such as Nessus, Nikto, Acunetix, and others on your web servers. This allows you to immediately detect potential probing.

2. Command Injection

Endpoint: http://localhost/DVWA/vulnerabilities/exec/
Method: Command injection attack
Payload: 127.0.0.1; ls; whoami;



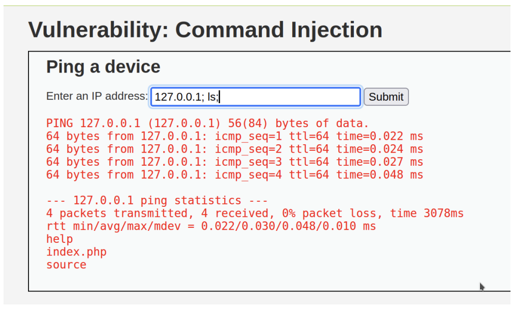
Vulnerability: Command Injection

Ping a device

Enter an IP address: 127.0.0.1

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.022 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.028 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.033 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.023 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3078ms
rtt min/avg/max/mdev = 0.022/0.026/0.033/0.004 ms
```



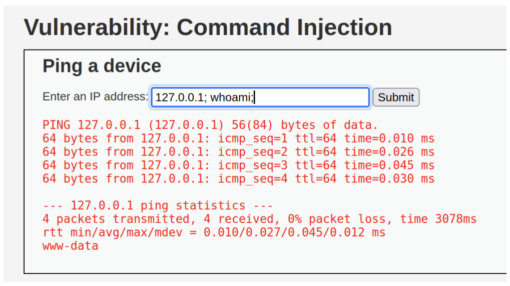
Vulnerability: Command Injection

Ping a device

Enter an IP address: 127.0.0.1; ls;

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.022 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.024 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.027 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.048 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3078ms
rtt min/avg/max/mdev = 0.022/0.030/0.048/0.010 ms
help
index.php
source
```



Vulnerability: Command Injection

Ping a device

Enter an IP address: 127.0.0.1; whoami;

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.010 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.026 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.045 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.030 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3078ms
rtt min/avg/max/mdev = 0.010/0.027/0.045/0.012 ms
www-data
```

Description:
The purpose of a command injection attack is to execute arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are conceivable when an application sends hazardous user-supplied data (forms, cookies, HTTP headers, etc.) to a system shell. In this attack, the hacker enables direct input into any of the several PHP functions that will run OS commands. It is possible to perform accidental acts and evade the designed command. Inadequate input validation makes command injection attacks viable.

Risk score: High
(Exploitability: 3, Weakness prevalence: 3, Weakness detectability: 3, Technical impact: 3)

Impact:
The following are some of the consequences of the command injection vulnerability:

1. An attacker can run arbitrary code on the target system, resulting in the system's complete compromise.
2. An attacker can access sensitive data stored on the target system.
3. By introducing malicious commands that consume all available resources, an attacker can cause a denial of service (DoS) to the target system.
4. An attacker can modify data stored on the target system, resulting in inaccurate results or data loss.
5. An attacker can intercept and access confidential data by redirecting network traffic from the target system to another system under their control.
6. An attacker can pivot and attack other systems on the same network as the susceptible system by using command injection.
7. Security mechanisms such as firewalls and intrusion detection/prevention systems (IDS/IPS) can be

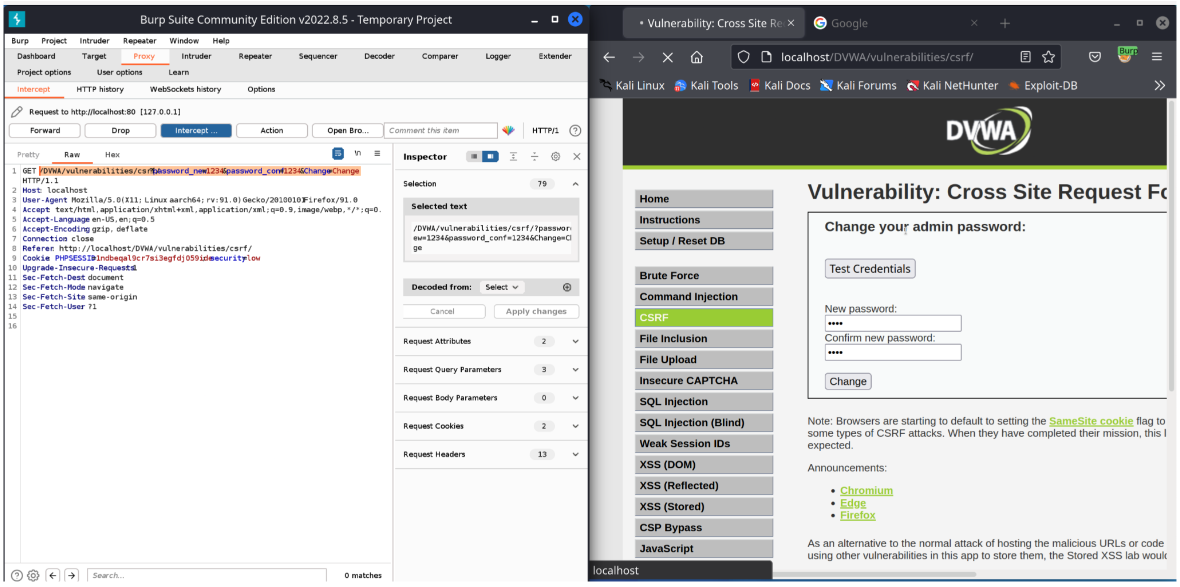
Fix / mitigation method:
The code should be rewritten to accept only extremely specific input. If the input does not match and does not yield a specific outcome, it will not be allowed to run. "white listing" should be used (only allowing certain values) instead of "black listing" screening (allowing any input and deleting the undesired).

Furthermore, run-time policy enforcement can be utilized in an allowlist form to block the execution of any unapproved instructions during the operation phase.

Lastly, the developer should grant the software system permissions that prohibit the user from accessing/opening privileged files.

3. Cross Site Request Forgery (CSRF)

Endpoint: http://localhost/DVWA/vulnerabilities/csrf/
Method: CSRF attack
Payload: ?password_new=555&password_conf=555&change=change#


Burp Suite Community Edition v2022.8.5 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender
Project options User options Learn
Intercept HTTP history WebSockets history Options
Forward Drop Intercept ... Action Open Bro...
Pretty Raw Hex
Inspector
Vulnerability: Cross Site Re
Google
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB
DVWA
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)
CSP Bypass
JavaScript
Test Credentials
Change
localhost

Request to http://localhost:80 [127.0.0.1]

```
GET /DVWA/vulnerabilities/csrf/?password_new=1234&password_conf=1234&Change=Change HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://localhost/DVWA/vulnerabilities/csrf/
Cookie: PHPSESSID=1ndbeqal9cr7si3egfdj059id; security=low
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
```

Selection 79
Selected text: /DVWA/vulnerabilities/csrf/?password_new=1234&password_conf=1234&Change=Change
Request Attributes 2
Request Query Parameters 3
Request Body Parameters 0
Request Cookies 2
Request Headers 13

localhost/DVWA/vulnerabilities/csrf/

Vulnerability: Cross Site Request Fo

Change your admin password:

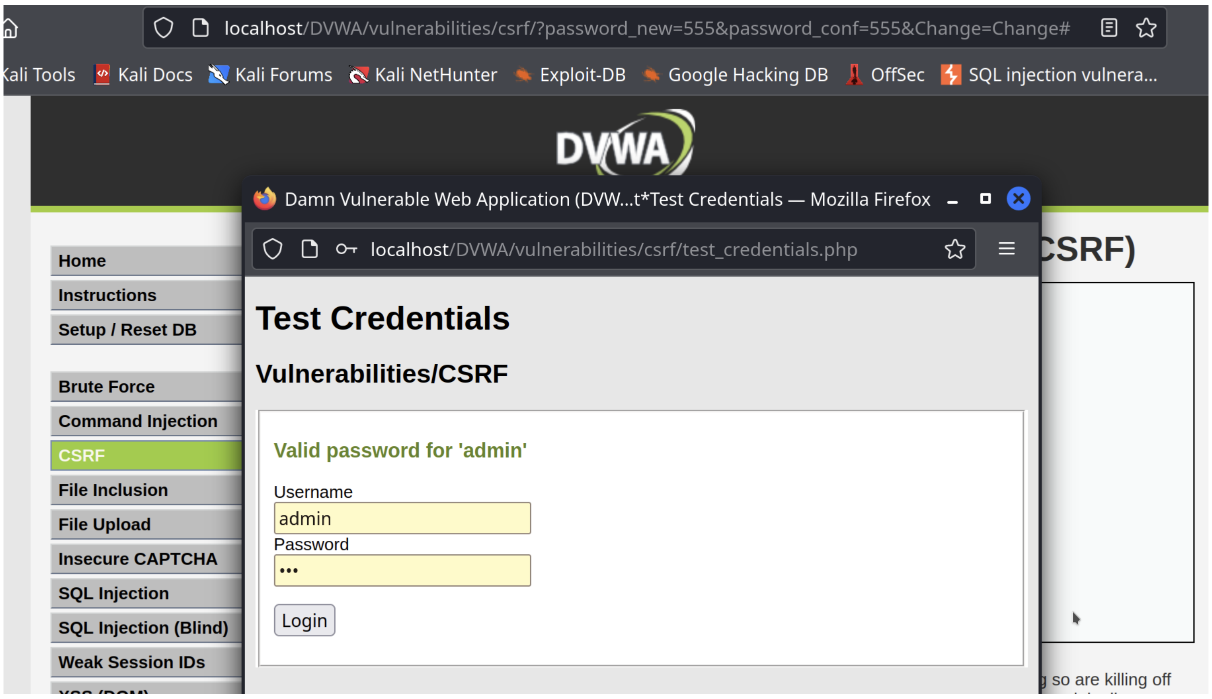
New password: ••••
Confirm new password: ••••

Note: Browsers are starting to default to setting the SameSite cookie flag to some types of CSRF attacks. When they have completed their mission, this l expected.

Announcements:

- Chromium
- Edge
- Firefox

As an alternative to the normal attack of hosting the malicious URLs or code using other vulnerabilities in this app to store them, the Stored XSS lab woulc


localhost/DVWA/vulnerabilities/csrf/?password_new=555&password_conf=555&Change=Change#

Damn Vulnerable Web Application (DVW...t*Test Credentials — Mozilla Firefox

localhost/DVWA/vulnerabilities/csrf/test_credentials.php

Test Credentials

Vulnerabilities/CSRF

Valid password for 'admin'

Username: admin

Password: •••

Description:
A cross-site request forgery (CSRF) attack forces an end user to perform undesirable actions on a web application in which they are currently authenticated. An attacker can force users of a web application to perform actions of the attacker's choosing with the use of social engineering (such as delivering a link via email/chat).

In the event of a typical user, a successful CSRF exploit can jeopardize end user data and operation. If the targeted end user is the administrator account, the entire online program may be jeopardized. In this lab, there are no safeguards in place to prevent this attack. This means that a link can be designed to perform a specific action (in this case, change the current users password). Then, using basic social engineering, have the target click the link (or simply visit a specific page) to initiate the action.

Risk score: High
(Exploitability: 3, Weakness prevalence: 3, Weakness detectability: 3, Technical impact: 3)

Impact:
The severity of a CSRF attack is determined by the targeted user and their privileges within an application.

For normal users, a successful CSRF attack will often include state-changing requests, such as altering their password or email address, transferring funds to another account, or making transactions using their credentials.

If a user with greater privilege, such as an administrative account, is successfully targeted, a CSRF may result in a full-fledged system breach. This is because such an account can submit requests for a different order.

Fix / mitigation method:

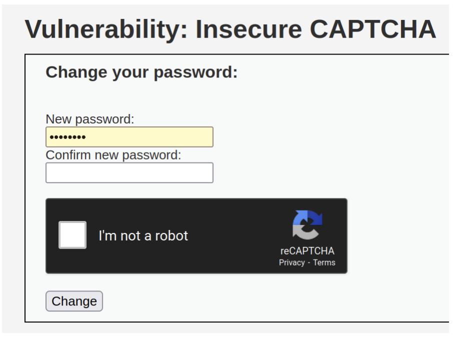
The website should require the user to enter both their current and new passwords. Because the attacker is unaware of this, the site is safe from CSRF-style attacks.

Moreover, all forms should include a hash (session id, function name, server-side secret) and a per-request nonce to assist against this type of attack.

In addition, CSRF attacks can be avoided by checking the referrer header in the client's HTTP request. By ensuring that the HTTP request originated from the original site, assaults from other sites are rendered ineffective. Due to memory constraints, referrer header checks are frequently employed on embedded network hardware.

4. Insecure CAPTCHA

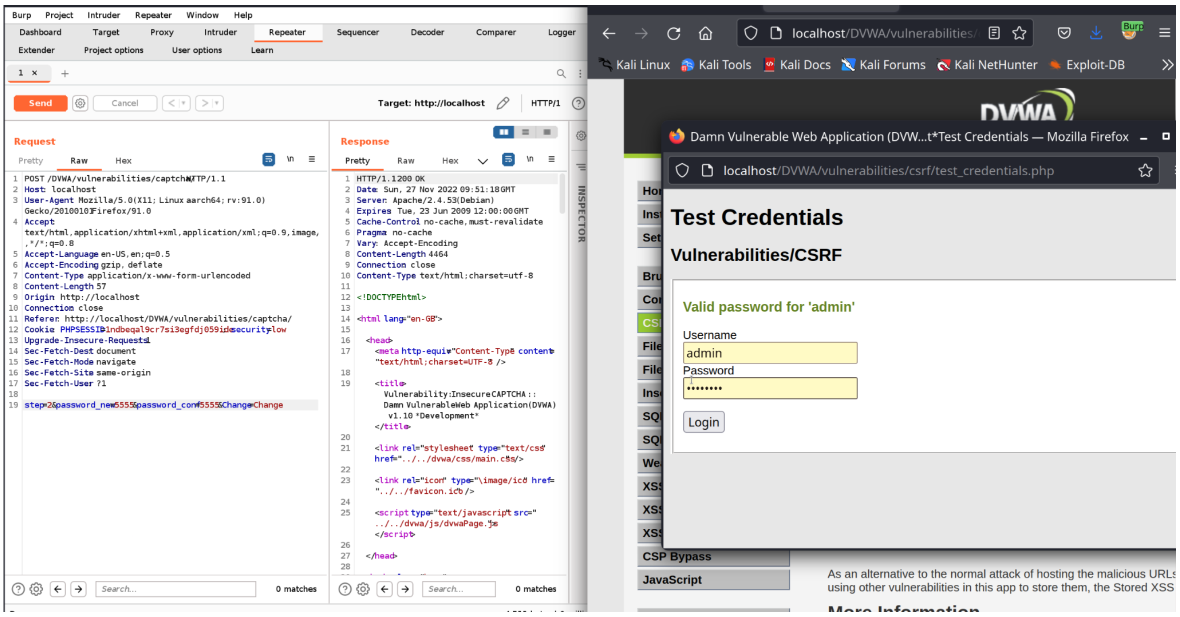
Endpoint: http://localhost/DVWA/vulnerabilities/captcha/
Method: Insecure CAPTCHA attack
Payload: ?step=2&password_new=5555&password_conf=5555&Change=Change


Vulnerability: Insecure CAPTCHA

Change your password:

New password: ••••••••

Confirm new password:


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger
Extender Project options User options Learn
Send Cancel
Target: http://localhost HTTP/1
Request
Pretty Raw Hex
Response
Pretty Raw Hex
INSPECTOR
0 matches

```
POST /DVWA/vulnerabilities/captcha/HTTP/1.1
Host: localhost
User-Agent Mozilla/5.0(X11; Linux aarch64; rv:91.0)
Gecko/20100101Firefox/91.0
Accept
text/html,application/xhtml+xml,application/xml;q=0.9,image,
,*/*;q=0.8
Accept-Language en-US,en;q=0.5
Accept-Encoding gzip, deflate
Content-Type application/x-www-form-urlencoded
Content-Length 57
Origin http://localhost
Connection close
Referer http://localhost/DVWA/vulnerabilities/captcha/
Cookie PHPSESSID=1ndbeqal9cr7si3egfdj059id;security=low
Upgrade-Insecure-Requests 1
Sec-Fetch-Dest document
Sec-Fetch-Mode navigate
Sec-Fetch-Site same-origin
Sec-Fetch-User ?1

step=2&password_new=5555&password_conf=5555&Change=Change
```

```
HTTP/1.1 200 OK
Date: Sun, 27 Nov 2022 09:51:18 GMT
Server: Apache/2.4.53(Debian)
Expires Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control no-cache, must-revalidate
Pragma no-cache
Vary: Accept-Encoding
Content-Length 4464
Connection close
Content-Type text/html;charset=utf-8

<!DOCTYPE html>

<html lang="en-GB">

  <head>
    <meta http-equiv="Content-Type" content=
    "text/html;charset=UTF-8" />

    <title>
      Vulnerability: Insecure CAPTCHA ::
      Damn Vulnerable Web Application (DVWA)
       v1.10 *Development*
    </title>

    <link rel="stylesheet" type="text/css"
    href="../../dvwa/css/main.css"/>

    <link rel="icon" type="\image/ico" href=
    "../../favicon.ico" />

    <script type="text/javascript" src="
    ../../dvwa/js/dvwaPage.js">
    </script>

  </head>
```

localhost/DVWA/vulnerabilities/
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB
DVWA
Damn Vulnerable Web Application (DVW...t*Test Credentials — Mozilla Firefox
localhost/DVWA/vulnerabilities/csrf/test_credentials.php
CSP Bypass
JavaScript

Test Credentials

Vulnerabilities/CSRF

Valid password for 'admin'

Username: admin

Password: ••••••••

As an alternative to the normal attack of hosting the malicious URLs using other vulnerabilities in this app to store them, the Stored XSS

More Information

Description:
A CAPTCHA is a program that determines if the user is human or computer. Many websites employ CAPTCHAs to prevent misuse from "bots," or automated programs designed to generate spam. Bots cannot navigate sites protected by CAPTCHAs because no computer program can interpret distorted language as well as humans. CAPTCHAs are frequently used to prevent automated bots from accessing sensitive functions. User registration and updates, password changes, and content submission are examples of such capabilities. In this lab, the CAPTCHA protects the user account's password changing capability. This provides only little protection against CSRF attacks and automated bot guesses.

The problem in the lab with this CAPTCHA is that it is easily circumvented. The developer has assumed that all users will proceed to screen 1, complete the CAPTCHA, and then proceed to the next screen where the password will be updated. The user can avoid the CAPTCHA system by sending the new password directly to the change page.

Risk Score: High
(Exploitability: 3, Weakness prevalence: 2, Weakness detectability: 3, Technical impact: 3)

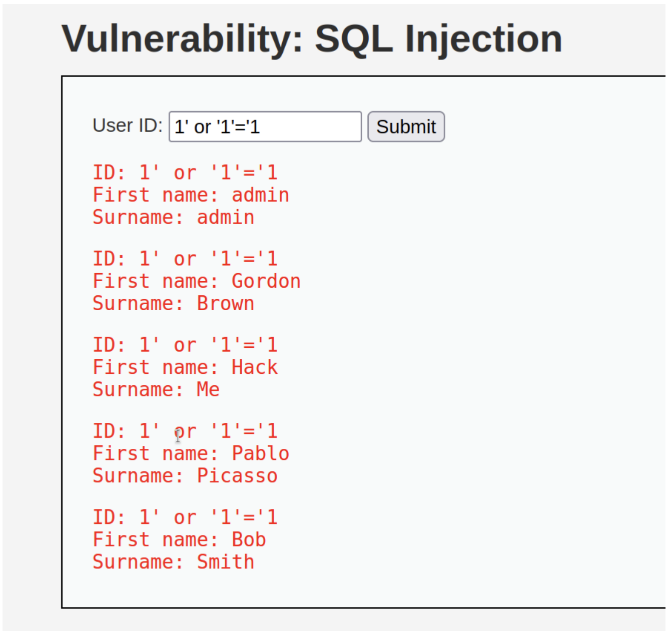
Impact:
Bypass security mechanisms to perform automated attacks

Fix / mitigation method:

To address the issue, the developer should close all possible entry points. The procedure has been streamlined so that data and CAPTCHA verification can be completed in a single step. Alternatively, the developer might have transferred the state variable from the medium level to the server side, preventing the user from changing it.

5. SQL Injection

Endpoint: http://localhost/DVWA/vulnerabilities/sqli/
Method: SQL Injection
Payload: 1‘or’1‘=’1



```
ID: 1' or '1'='1
First name: admin
Surname: admin

ID: 1' or '1'='1
First name: Gordon
Surname: Brown

ID: 1' or '1'='1
First name: Hack
Surname: Me

ID: 1' or '1'='1
First name: Pablo
Surname: Picasso

ID: 1' or '1'='1
First name: Bob
Surname: Smith
```

Description:
A SQL injection attack involves inserting or "injecting" a SQL query into the program via the client's input data. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), perform database administration operations (such as shutting down the DBMS), recover the content of a given file on the DBMS file system, and in some cases issue commands to the operating system. SQL injection attacks are a sort of injection attack in which SQL commands are injected into data-plane input to influence the execution of predetermined SQL commands.

Risk Score: High
(Exploitability: 3, Weakness prevalence: 3, Weakness detectability: 3, Technical impact: 3)

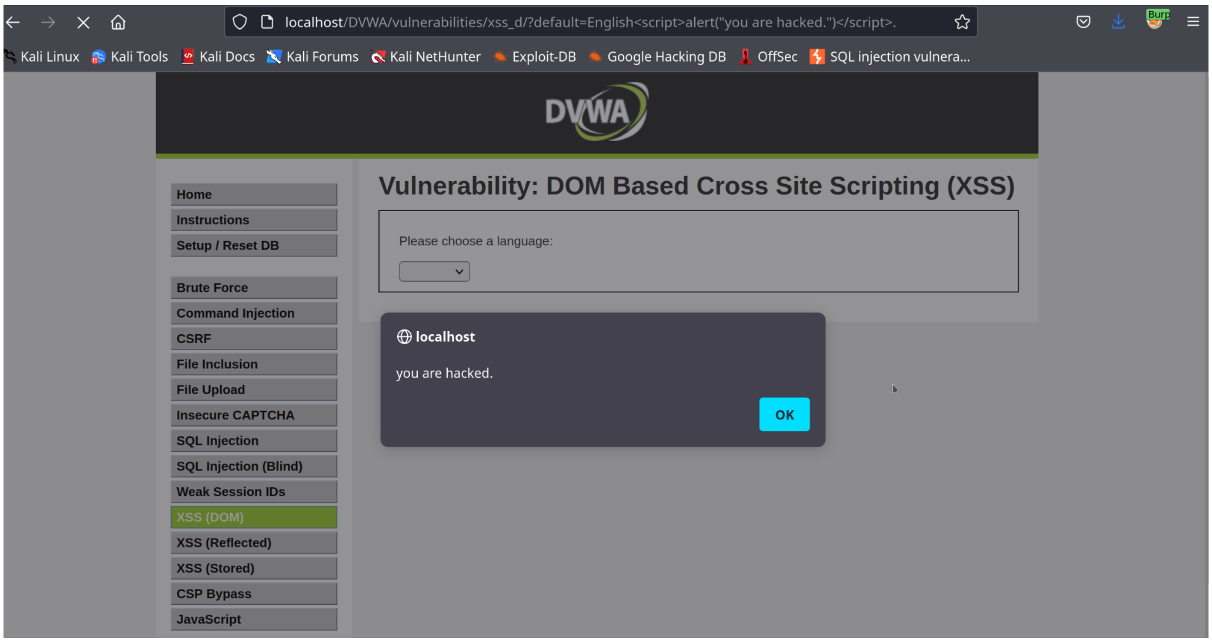
Impact:
SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and gain administrative access to the database server.

Fix / mitigation method:

The queries should be parameterized queries (rather than being dynamic). This means the query has been defined by the developer, and has distinguish which sections are code, and the rest is data. Other options are use of properly constructed stored procedures, the allow-list input validation, and escaping all user supplied input.

6. DOM Based Cross Site Scripting (XSS)

Endpoint: http://localhost/DVWA/vulnerabilities/xss_d/
Method: XSS attack
Payload: ?default=English<script>alert(“you are hacked.”)</script>



Vulnerability: DOM Based Cross Site Scripting (XSS)

Please choose a language:

localhost

you are hacked.

Description:
Cross-Site Scripting or XSS attacks are a sort of injection in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker utilizes a web application to transmit malicious code to a separate end user, typically in the form of a browser side script. The flaws that allow these attacks to succeed are extremely common, and they occur whenever a web application includes user input within the output it generates without verifying or encoding it.

DOM Based XSS is an XSS attack in which the attack payload is performed as a result of manipulating the DOM "environment" in the victim's browser utilized by the original client side script, causing the client side code to run in a "unexpected" manner. That is, the page itself (the HTTP response) remains same, but the client side code contained within the page runs differently as a result of the malicious DOM alterations.

In this lab, the website does not validate the requested input before incorporating it in the result text, which can cause the XSS attack.

Risk Score: High
(Exploitability: 3, Weakness prevalence: 2, Weakness detectability: 3, Technical impact: 3)

Impact:
Although DOM-based cross-site scripting vulnerabilities are not widespread, the repercussions of a successful attack can be just as severe as those of other reflected XSS assaults. Based on the short example described previously, these are some actions that a black-hat hacker might take.

An attacker may launch a phishing campaign and send millions of emails including a malicious link carrying a payload that redirects visitors to a phishing page meant to impersonate your online application. As a result, millions of users may have their credentials stolen and blame your web service, badly harming your reputation.

An attracker could develop a payload that redirects the user to a malicious page that looks like the login page for your application. They might then distribute this malicious URL to your internal users, including your CEO. If even one of your users falls for the ruse, the attacker will gain their credentials and utilize them to further expand the attack. This could eventually allow criminal actors to get access

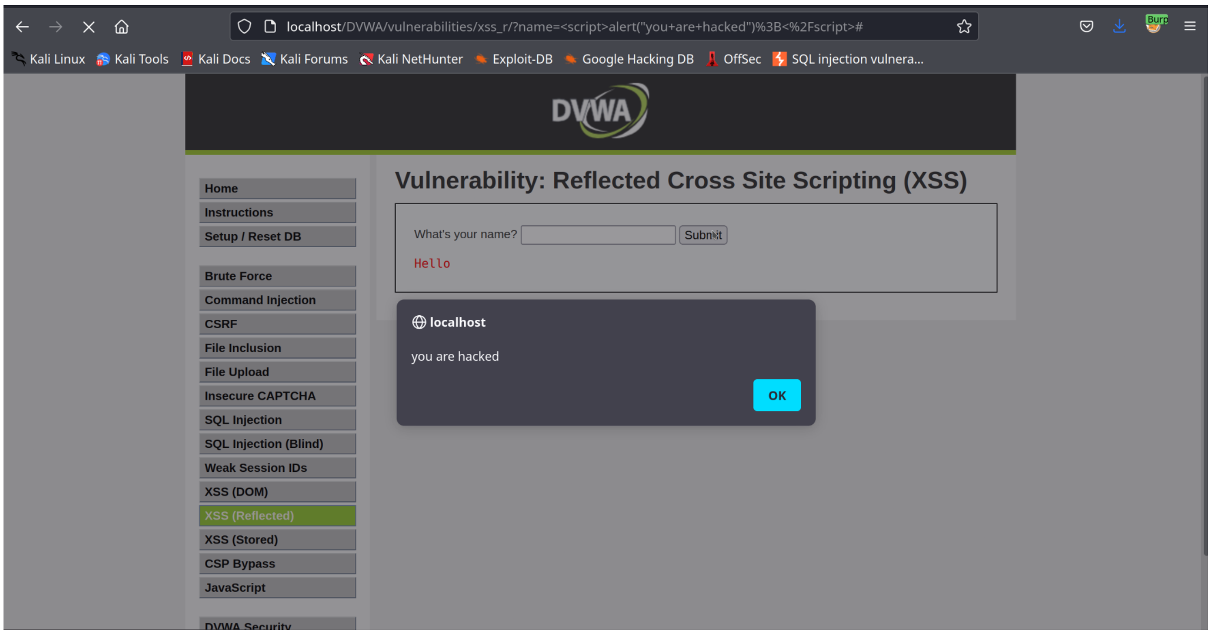
Fix / mitigation method:
The simplest way to entirely avoid DOM-based XSS vulnerabilities in your JavaScript code is to utilize the correct output technique (a safe sink). If you wish to write into an element, don't use innerHtml. Instead, use innerText or textContent.

It should be noted that not all DOM elements offer a safe output mechanism. There are times when you must just avoid using untrusted data. For example, you should never write untrusted data to sinks like eval or execScript.

The developer should apply standard XSS prevention techniques (filtering and escaping) to JavaScrip. Unfortunately, there are no universal JavaScript libraries to help you filter and escape data, as there are for server-side languages, therefore developers must design and maintain such functionality themselves. Proper filtering and escaping for DOM-based XSS is a complicated subject that is detailed in a special article.

7. Reflected Cross Site Scripting (XSS)

Endpoint: http://localhost/DVWA/vulnerabilities/xss_r/
Method: XSS attack
Payload:?name=<script>alert(“you are hacked”)</script>



```
localhost/DVWA/vulnerabilities/xss_r/?name=<script>alert("you+are+hacked")%3B<%2Fscript>#
```

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello

localhost

you are hacked

Description:
Reflected XSS attacks occur when the injected script is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input supplied to the server as part of the request. Reflected attacks are provided to victims via another channel, such as an email message or another website. When a user is misled into clicking on a malicious link, submitting a specially crafted form, or simply navigating to a malicious site, the injected code travels to the susceptible web site and reflects the attack back to the user's browser. The browser then executes the code because it originated from a "trusted" server.
In this lab, the website does not check the requested input before including it in the result text, which might lead to an XSS attack.

Risk Score: High
(Exploitability: 3, Weakness prevalence: 3, Weakness detectability: 3, Technical impact: 3)

Impact:
If an attacker can manipulate a script that is performed in the victim's browser, they can usually totally compromise that user. Among other things, the attacker can:

1. Perform any action that the user has access to within the program.
2. View any information that the user is able to see.
3. Alter any information that the user has the ability to modify.

4. Initiate interactions with other program users, including harmful assaults that appear to come from the first victim user.

There are several ways for an attacker to persuade a target user to create a request that they control in order to deliver a reflected XSS attack. These include inserting links on the attacker's website or another website that permits material to be created, as well as sending a link in an email, tweet, or other communication. The attack may be directed directly against a known user, or it could be directed at all users of the program.

Fix / mitigation method:
There is no single solution to XSS. To avoid XSS, the correct combination of defensive mechanisms must be used.

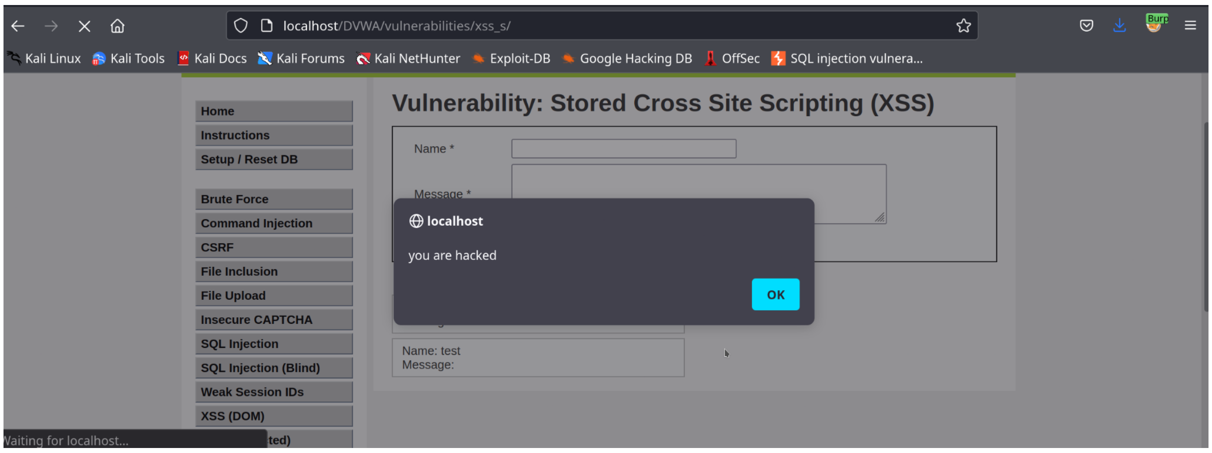
XSS issues are less common in applications built with contemporary web frameworks. These frameworks guide developers toward appropriate security practices and assist prevent XSS through the use of templating, auto-escaping, and other techniques. Learn how your framework avoids XSS and where it falls short. There will be occasions when you will need to act outside of the framework's protection. Output Encoding and HTML Sanitization are critical at this point.

Each variable in a web application must be secured. Perfect injection resistance is achieved by ensuring that all variables are validated and then escaped or sanitized. Any variable that is not subjected to this method is a potential flaw. Frameworks make it simple to validate variables and escape or sanitize them.

When you need to reliably display data exactly as a user typed it in, Output Encoding is advised. Variables should not be treated as code rather than words. This section discusses each type of output encoding, when to use it, and when to avoid using dynamic variables altogether.

8. Stored Cross Site Scripting (XSS) Store

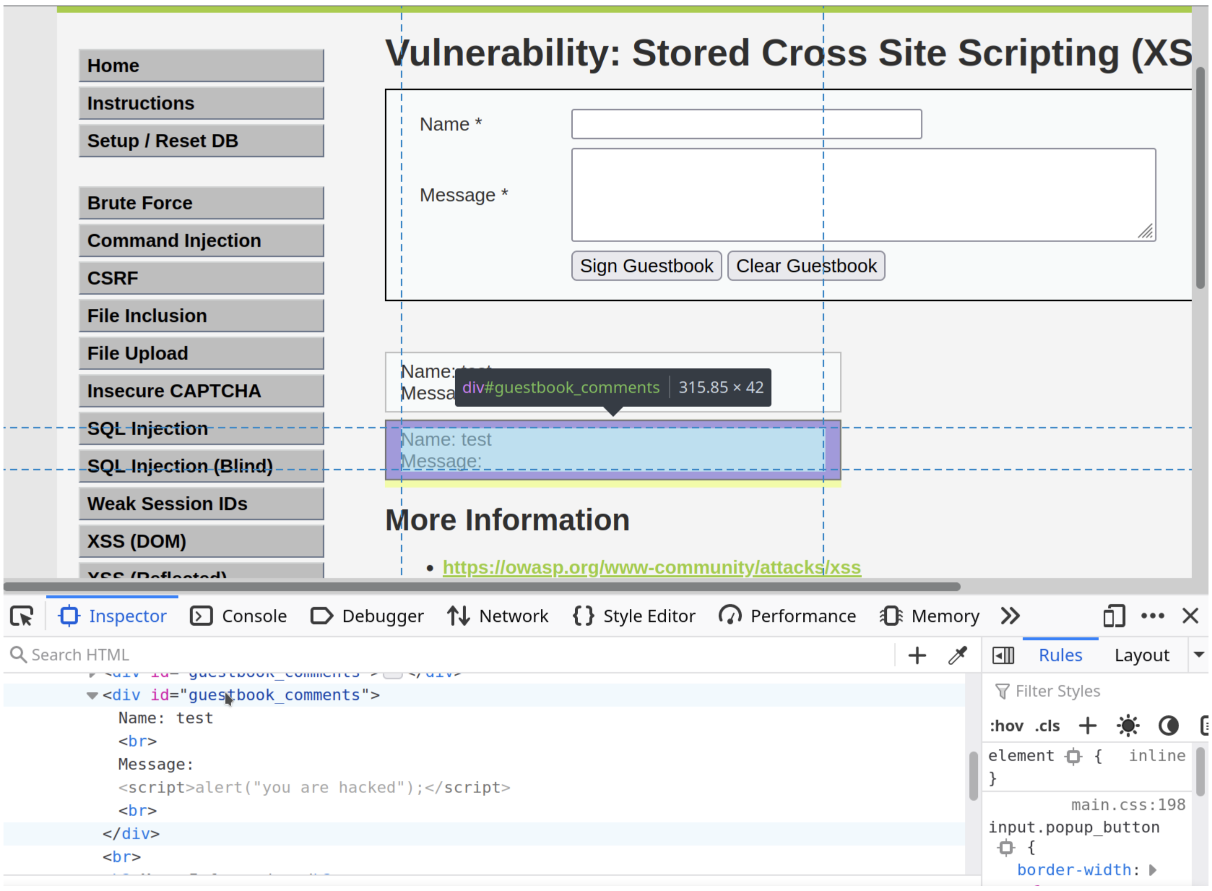
Endpoint: http://localhost/DVWA/vulnerabilities/xss_s/
Method: XSS attack
Payload: message field: <script>alert(“you are hacked”)</script>



localhost

you are hacked

Name: test
Message:



Name: test
Message:

```
<div id="guestbook_comments">
  Name: test
  <br>
  Message:
  <script>alert("you are hacked");</script>
  <br>
</div>
<br>
```

```
element {      inline
}
               main.css:198
input.popup_button
  {
  border-width:
```

Description:

Stored XSS (also known as persistent or second-order XSS) occurs when an application obtains data from an untrusted source and includes that data in an unsafe manner in subsequent HTTP responses.

The data in question could be submitted to the program via HTTP requests, such as comments on a blog post, user nicknames in a chat room, or customer purchase contact information. In other circumstances, the data could come from an untrustworthy source, such as a webmail program displaying SMTP messages, a marketing application displaying social media posts, or a network monitoring application displaying packet data from network traffic.

Risk Score: High
(Exploitability: 3, Weakness prevalence: 3, Weakness detectability: 3, Technical impact: 3)

Impact:
If an attacker can manipulate a script that is performed in the victim's browser, they can generally completely compromise that user. The attacker can perform any of the operations that are relevant to the impact of reflected XSS vulnerabilities.

The main distinction between reflected and stored XSS vulnerabilities in terms of exploitability is that a stored XSS vulnerability allows for self-contained attacks within the application itself. The attacker does not need to discover an external technique to persuade other users to send a specific request containing their vulnerability. Rather, the attacker embeds their vulnerability inside the program and merely waits for users to come across it.

The self-contained nature of cached cross-site scripting vulnerabilities is especially important when an XSS vulnerability affects just users who are currently signed in to the application. If the XSS is reflected, the attack must be well-timed: a user who is convinced to perform the attacker's request while not logged in will not be compromised. In contrast, if the XSS is cached, the user is assured to be logged in at the time the attack is encountered.

Fix / mitigation method:

There is no single solution to XSS. To avoid XSS, the correct combination of defensive mechanisms must be used.

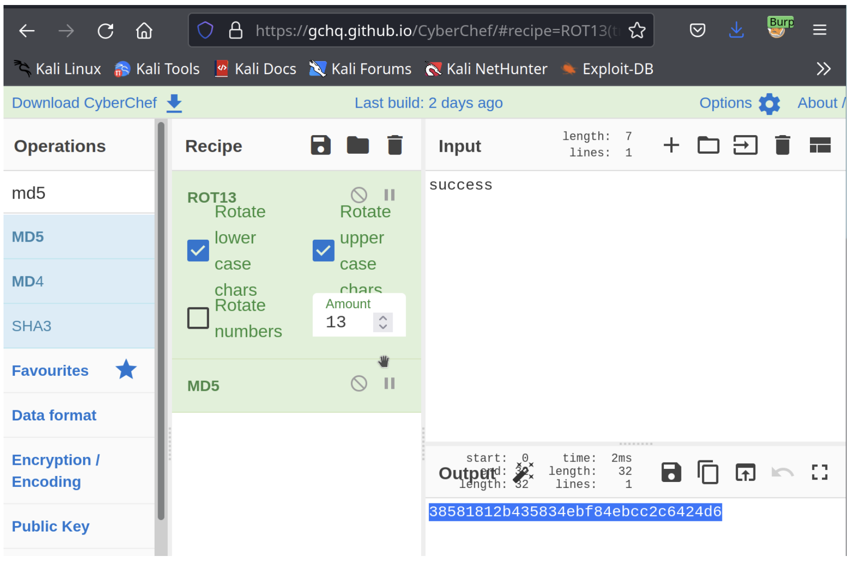
XSS issues are less common in applications built with contemporary web frameworks. These frameworks guide developers toward appropriate security practices and assist prevent XSS through the use of templating, auto-escaping, and other techniques. Learn how your framework avoids XSS and where it falls short. There will be occasions when you will need to act outside of the framework's protection. Output Encoding and HTML Sanitization are critical at this point.

Each variable in a web application must be secured. Perfect injection resistance is achieved by ensuring that all variables are validated and then escaped or sanitized. Any variable that is not subjected to this method is a potential flaw. Frameworks make it simple to validate variables and escape or sanitize them.

When you need to reliably display data exactly as a user typed it in, Output Encoding is advised. Variables should not be treated as code rather than words. This section discusses each type of output encoding, when to use it, and when to avoid using dynamic variables altogether.

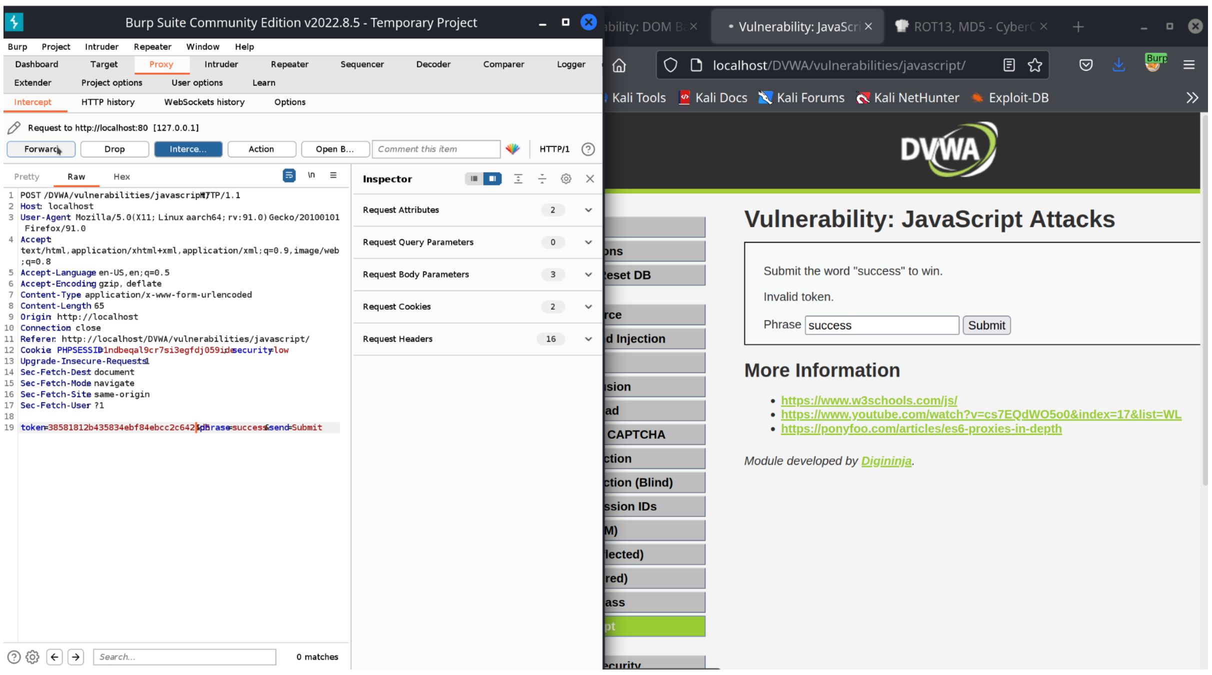
9. JavaScript Attacks

Endpoint: http://localhost/DVWA/vulnerabilities/javascript/
Method: Change the phrase to success and then use the function generate_token() to update the token



```
success
```

```
38581812b435834ebf84ebcc2c6424d6
```



Burp Suite Community Edition v2022.8.5 - Temporary Project
Request to http://localhost:80 [127.0.0.1]
Forward
Drop
Interce...
Action
Open B...
Pretty
Raw
Hex
Inspector
localhost/DVWA/vulnerabilities/javascript/

```
POST /DVWA/vulnerabilities/javascript/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:91.0) Gecko/20100101
 Firefox/91.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/web
;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 65
Origin: http://localhost
Connection: close
Referer: http://localhost/DVWA/vulnerabilities/javascript/
Cookie: PHPSESSID=1ndbeqal9cr7si3egfdj059ide; security=low
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1

token=38581812b435834ebf84ebcc2c6424d6&phrase=success&send=Submit
```

Vulnerability: JavaScript Attacks

Submit the word "success" to win.

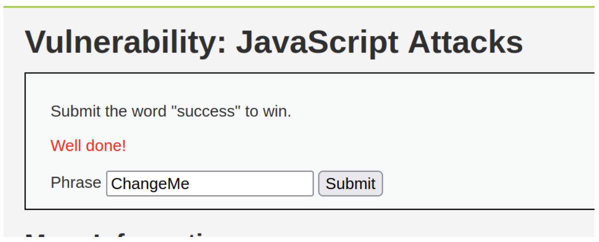
Invalid token.

Phrase success Submit

More Information

- https://www.w3schools.com/js/
- https://www.youtube.com/watch?v=cs7EQdWO5o0&index=17&list=WL
- https://ponyfoo.com/articles/es6-proxies-in-depth

Module developed by Digininja.



Vulnerability: JavaScript Attacks

Submit the word "success" to win.

Well done!

Phrase ChangeMe Submit

Description:

The attacks in this section are intended to teach you about how JavaScript is utilized in browsers and how it may be abused. The attacks could be carried out by just analyzing network traffic, but that isn't the goal, and it would also be far more difficult.
The page contains all of the JavaScript. Read the source to determine which function is being used to generate the token required to match with the phrase, and then manually execute the function.

Risk Score: High
(Exploitability: 3, Weakness prevalence: 3, Weakness detectability: 3, Technical impact: 3)

Impact:

1. Loss of Sensitive Customer Information: A JavaScript injection attack can result in the loss of sensitive PII such as credit card information, as well as name and address. It may also entail the stealing of permission cookies, which might result in user impersonation.

2. Profit loss: JavaScript injection assaults can ruin a company's reputation, which can hurt earnings.

3. Issues of Regulatory and Compliance: Government and industry standards, such as the Payment Card Industry Data Security Standards (PCI DSS) and the General Data Protection Regulations (GDPR), can subject organizations to litigation and

Fix / mitigation method:

You can never trust the user and must always assume that any code supplied to the user can be modified or circumvented.