Week-1 Assignment

What is OWASP?

OWASP stands for the "Open Web Application Security Project." It is a nonprofit organisation focused on improving the security of software applications, particularly web applications. OWASP provides resources, tools, and knowledge to help developers, security professionals, and organisations build secure and robust applications.
The primary goal of OWASP is to educate and raise awareness about the importance of web application security. The organisation gathers security experts and practitioners from around the world to collaborate on identifying and addressing the most critical security risks facing web applications.

OWASP 2021 REPORT’S TOP 10

OWASP periodically updates its resources, projects, and reports. It's recommended to visit the official OWASP website (https://owasp.org/) for the most up-to-date information on the OWASP Top Ten and other OWASP projects released after September 2021.

A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
A10:2021-Server-Side Request Forgery (SSRF)*

vulnerability Documentation

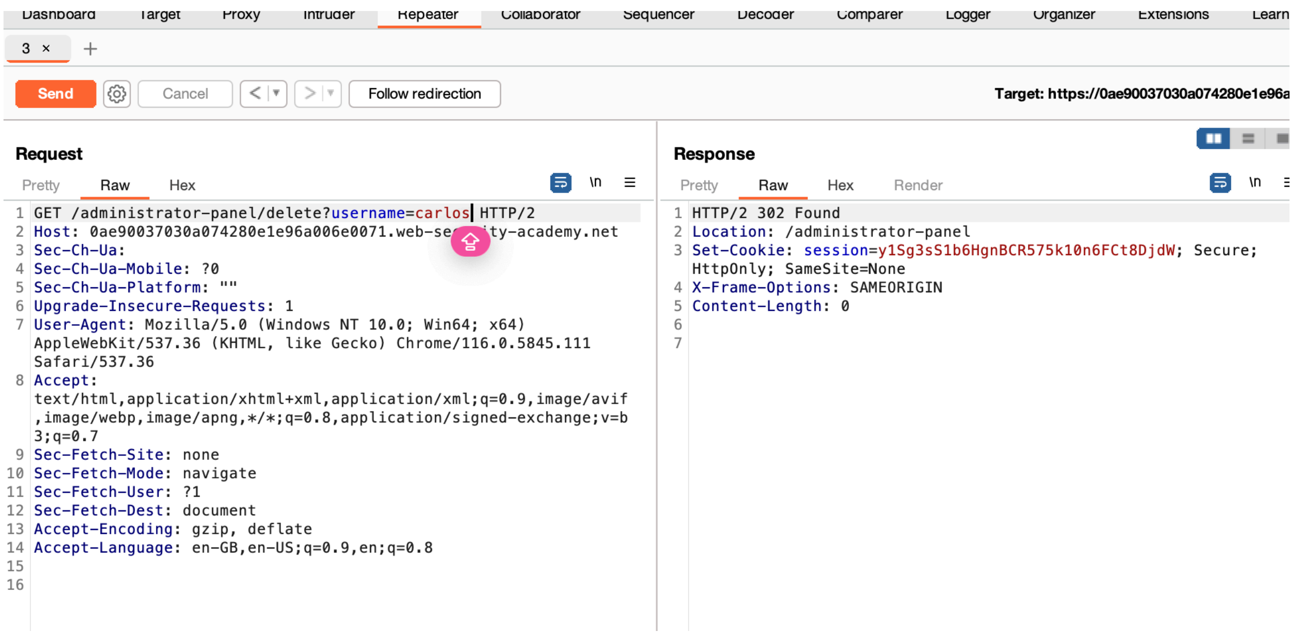
1. Broken Access Control

Broken Access Control refers to a security vulnerability that occurs when an application's access controls are improperly implemented or enforced. Access controls are mechanisms that dictate what actions a user, whether authenticated or not, can perform within an application. These actions might include viewing certain data, modifying settings, or performing administrative tasks.

CONSEQUENCES

When access controls are broken, attackers can gain unauthorized access to functionalities or data that they shouldn't have permission to access. This can lead to a range of security breaches and compromise the confidentiality, integrity, and availability of the application and its data.

ACCESS CONTROL USING BURP SUITE

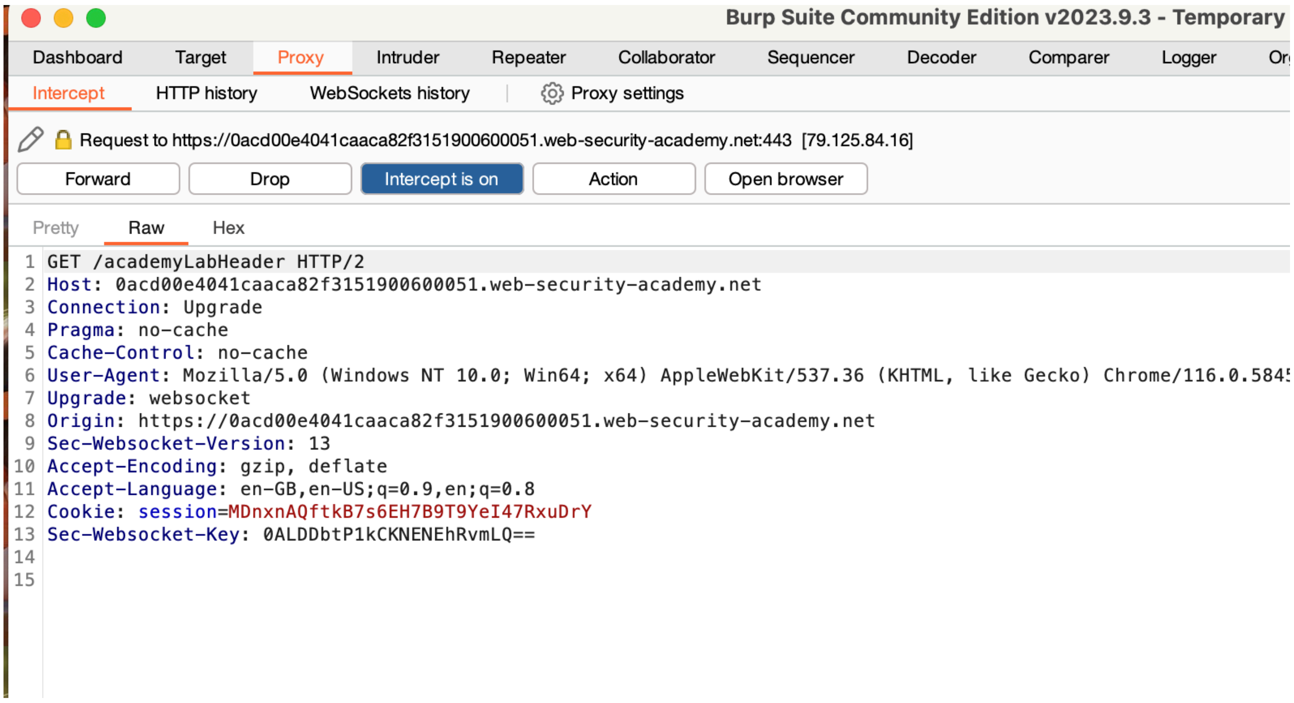


```
GET /administrator-panel/delete?username=carlos HTTP/2
Host: 0ae90037030a074280e1e96a006e0071.web-security-academy.net
Sec-Ch-Ua:
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: ""
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5845.111
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b
3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
```

```
HTTP/2 302 Found
Location: /administrator-panel
Set-Cookie: session=y1Sg3sS1b6HgnBCR575k10n6FCt8DjdW; Secure;
HttpOnly; SameSite=None
X-Frame-Options: SAMEORIGIN
Content-Length: 0
```

2. SQL INJECTION

SQL injection is a type of cybersecurity vulnerability that occurs when an attacker can manipulate an application's input to execute unintended SQL (Structured Query Language) commands on a database. SQL injection attacks can have serious consequences, as they can allow attackers to access, modify, or delete data in a database, bypass authentication mechanisms, and execute arbitrary commands on the database server.



Request to https://0acd00e4041caaca82f3151900600051.web-security-academy.net:443 [79.125.84.16]

```
GET /academyLabHeader HTTP/2
Host: 0acd00e4041caaca82f3151900600051.web-security-academy.net
Connection: Upgrade
Pragma: no-cache
Cache-Control: no-cache
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.584
Upgrade: websocket
Origin: https://0acd00e4041caaca82f3151900600051.web-security-academy.net
Sec-Websocket-Version: 13
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Cookie: session=MDnxnAQftkB7s6EH7B9T9YeI47RxuDrY
Sec-Websocket-Key: 0ALDDbtP1kCKNENEhRvmLQ==
```

Cryptographic Failures:
"CWE" stands for Common Weakness Enumeration, and it is a community-developed list of common software security weaknesses. For cryptographic failures, there are several CWEs (Common Weakness Enumerations) associated with different aspects of cryptographic vulnerabilities. Here are a few relevant CWEs related to cryptographic failures:

1. **CWE-310: Cryptographic Issues**: This category encompasses a range of cryptographic weaknesses, including improper encryption, insecure use of cryptographic functions, and issues with key management.

2. **CWE-327: Use of a Broken or Risky Cryptographic Algorithm**: This refers to vulnerabilities that arise when a weak or compromised cryptographic algorithm is used for encryption, decryption, or hashing.

3. **CWE-328: Reversible One-Way Hash**: This is about using a reversible cryptographic hash when the hash function is meant to be one-way (non-reversible).

4. **CWE-329: Not Using a Random IV with CBC Mode**: In CBC (Cipher Block Chaining) mode of operation, using a non-random or predictable Initialization Vector (IV) can lead to vulnerabilities.

5. **CWE-330: Use of Insufficiently Random Values**: Cryptography often relies on randomness for security. Insufficiently random values can weaken cryptographic mechanisms.

6. **CWE-331: Insufficient Entropy**: This is about insufficient entropy when generating keys, random numbers, or other cryptographic values, making them more predictable.

7. **CWE-332: Insufficient Entropy in PRNG**: Similar to the previous entry, this relates specifically to weaknesses in Pseudo-Random Number Generators (PRNGs).

8. **CWE-345: Insufficient Verification of Data Authenticity**: This involves failing to adequately verify the authenticity of data, leading to potential cryptographic weaknesses.

9. **CWE-347: Improper Verification of Cryptographic Signature**: When cryptographic signatures are not properly verified, attackers can forge signatures and impersonate trusted entities.

10. **CWE-350: Reliance on Reverse Engineering of a Cryptographic Algorithm**: Relying on the secrecy of an algorithm (security through obscurity) can lead to vulnerabilities if the algorithm is reverse-engineered.

These are just a few examples of CWEs related to cryptographic failures. Each CWE provides a description of the vulnerability, potential consequences, and guidance on how to prevent or mitigate the vulnerability. It's important to consult the official CWE website or other trusted sources for the most up-to-date and comprehensive list of cryptographic-related weaknesses.

Server-Side Request Forgery (SSRF)

Server-Side Request Forgery (SSRF) is a type of security vulnerability that occurs when an attacker tricks a web application into making unauthorized requests to internal or external resources on behalf of the application itself. In an SSRF attack, the attacker can manipulate input that the application uses to initiate requests, leading to potential data exposure, remote code execution, and other malicious actions.

Here's how an SSRF attack generally works:

1. **Input Manipulation**: The attacker provides input to the web application, typically in the form of URLs or network endpoints.

2. **Server-Side Execution**: The application processes the provided input and makes requests to the specified URLs or endpoints, often without proper validation or security checks.

3. **Unauthorized Requests**: If the input manipulation is successful, the application sends requests to resources that it shouldn't have access to. These resources might include internal services, sensitive files, or external systems.

4. **Potential Exploitation**: Depending on the attacker's goals and the application's behavior, the consequences of an SSRF attack can vary. Attackers could extract sensitive data, retrieve internal files, perform port scanning, or even execute remote code.

To prevent SSRF vulnerabilities, developers should follow best practices such as:

1. **Input Validation**: Thoroughly validate and sanitize all user-supplied input, especially when it's used to form URLs or make requests to remote resources.

2. **Whitelisting**: Implement a whitelist of allowed domains or IP addresses that the application is allowed to access. This can help prevent requests to unauthorized resources.

3. **Firewall Rules**: Configure network firewalls to restrict outgoing connections from your application's server. This can help prevent unauthorized access to external resources.

4. **Use of Trusted Libraries**: When performing remote requests, use trusted libraries and APIs that provide security features to mitigate SSRF risks.

5. **Avoid Blind Trust**: Don't blindly trust user-provided URLs or parameters. Instead, validate and sanitize inputs thoroughly before using them in requests.

6. **Restrict Network Access**: If possible, restrict the application's network access so that it can only communicate with necessary and trusted resources.

By implementing these security measures, developers can significantly reduce the risk of SSRF vulnerabilities and protect their applications from potential exploits.