CYBR 350

Week # 4

WebGoat Part -1

Abby Gopal

General:

Exercise Title and Objective: HTTP Basics

The objective is to grasp the fundamental concepts of HTTP, focusing on requests, responses, methods, and status codes within a web context.

Methodology:

1. Understanding the Hypertext Transfer Protocol (HTTP) and its role in web communication.
2. Learning about various HTTP methods (GET, POST, PUT, DELETE) and their purposes.
3. HTTP Responses: Analyzing different status codes (200, 404, 500) and their meanings in web interactions.
4. Using tools like Burp Suite to intercept and analyze HTTP requests and responses.
5. Experiment with modifying request headers and parameters to observe changes in server responses.

Steps Taken:

To view the HTTP requests made by your browser:

1. Right-click on a webpage and select "Inspect" or press Ctrl + Shift + I (Windows/Linux) or Cmd + Option + I (Mac) to open Developer Tools.
2. In Developer Tools, go to the "Network" tab.
3. Refresh the webpage. You'll see a list of network requests made by the browser.
4. You can filter the requests by file type (e.g., All, XHR, Documents) or search for specific files in the search bar.
5. Clicking on a request will display more details, including request headers, response headers, and the data sent.
6. You can view the request method (GET, POST, etc.), request URL, request headers, parameters (for POST requests), and response data within the request details.

Results and Findings:

1. Grasped the essence of HTTP, including requests, responses, methods, and status codes.
2. Gained hands-on experience using tools like Burp Suite to intercept and modify HTTP traffic.
3. Observed changes in server responses upon altering HTTP request parameters.

Analysis and Conclusions:

The exercise emphasized the critical role of HTTP in web communication, highlighting the significance of understanding request methods, status codes, and traffic interception for security and debugging purposes.

Recommendations:

1. Continuously practice intercepting and analyzing HTTP traffic using tools like Burp Suite to deepen understanding.
2. Stay updated with HTTP specifications and evolving web standards for effective web development and security practices.
3. Reflection:
4. The exercise underscored the foundational importance of HTTP in web interactions, highlighting its relevance in web development, debugging, and security analysis.

References:

HTTP specifications documentation, Burp Suite tutorials, online resources on HTTP methods, and status codes.

A screenshot of a computer

Description automatically generatedA screen shot of a phone

Description automatically generated

HTTP Proxies:

Exercise Title and Objective:

Exercise Title: Intercepting and Modifying HTTP Requests with Burp Suite **Objective:** Gain hands-on experience in intercepting and modifying GET and POST HTTP requests using Burp Suite, focusing on understanding the impact of altering request methods, headers, and parameters.

Methodology:

Approach**:** Utilize Burp Suite to intercept and modify an HTTP GET request, change the method to GET, add a specific header, and modify input values. Repeat the process for an HTTP POST request. Tools Used:

* Burp Suite

Steps Taken:

1. Enable Intercepting:
   * Turn on intercepting in Burp Suite.
   * Trigger an HTTP GET request by submitting a form or interacting with a web page.
2. Access Captured GET Request:
   * View the intercepted GET request within Burp Suite.
3. Edit GET Request Method:
   * Change the HTTP GET request method from GET to POST.
4. Add a Header to GET Request:
   * Introduce a new header with a specific ID and set its value to "truth."
5. Change GET Input Value:
   * Modify the input value within the GET request parameters (e.g., changing "me" to "requests\_add\_tampered\_easily").
6. Forward the Modified GET Request:
   * Use the "forward" button in Burp Suite to send the modified GET request to the server.
7. Observation (GET Request):
   * Observe the confirmation message indicating successful tampering of the GET request.
8. Trigger an HTTP POST Request:
   * Perform an action that triggers an HTTP POST request.
9. Access Captured POST Request:
   * View the intercepted POST request within Burp Suite.
10. Edit POST Request Method:

* Change the HTTP POST request method from POST to GET.

1. Add a Header to POST Request

* Introduce a new header with a specific ID and set its value to "truth."

1. Change POST Input Value:

* Modify the input value within the POST request parameters.

1. Forward the Modified POST Request:

* Use the "forward" button in Burp Suite to send the modified POST request to the server.

1. Observation (POST Request):

* Observe the confirmation message indicating successful tampering of the POST request.

Analysis and Conclusions:

Implications of Findings:

* The ability to modify both GET and POST requests raises awareness of potential security risks in web applications.

Insights on Security Posture:

* The exercise highlights the importance of validating and securing server-side request handling.

Recommendations:

* Emphasize input validation and implement proper security controls.
* Regularly test and monitor for unauthorized request tampering.

Reflection:

* Learned practical skills in intercepting and modifying both GET and POST requests.
* Future approaches might involve exploring different attack scenarios and advanced techniques.
* Illustrates the importance of securing web applications against various forms of request tampering.

References:

* Burp Suite documentation and tutorials.

Developer Tool:

Exercise Title and Objective:

Exercise Title: JavaScript Function Call and Input Manipulation **Objective:** Call the JavaScript function **webgoat.customjs.phoneHome()** using the console in the developer tools, retrieve the response, and submit the generated random number to a text field as part of an assignment.

Methodology:

Approach: Use the browser's developer tools console to call the JavaScript function **webgoat.customjs.phoneHome()**. Capture and analyze the response. Copy the generated random number and submit it to a text field as required. **Tools Used:**

* Browser Developer Tools Console

Steps Taken:

1. Access Developer Tools:
   * Open the browser's developer tools and navigate to the console tab.
2. Call JavaScript Function:
   * Enter the command **webgoat.customjs.phoneHome()** in the console and execute it.
3. Capture Response:
   * Capture and note the response generated by the JavaScript function in the console.
4. Copy Random Number:
   * Extract the random number from the response obtained in step 3.
5. Submit Random Number:
   * Navigate to the targeted text field or input area where the random number needs to be submitted.
6. Paste and Submit:
   * Paste the copied random number into the text field and submit the form.
7. Observation:
   * Observe the web application's behavior and any confirmation message upon successful submission.

Analysis and Conclusions:

Implications of Findings:

* The exercise demonstrates the ability to interact with JavaScript functions in a web application.

Recommendations:

* Ensure secure handling of client-side functions and inputs to prevent potential security risks.

Reflection:

* Learned the process of calling JavaScript functions in the browser console.
* Future approaches might involve exploring more complex JavaScript interactions.
* Illustrates the importance of secure client-side scripting.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

CIA Triads:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Exercise Title and Objective: A1- Broken access control

A1-Session Hijacking

This exercise aims to understand the concept of session hijacking, exploring different techniques to take over a user's session and gain unauthorized access to the application.

Methodology:

1. Utilized the WebGoat application hosted on a Kali machine.
2. Burp Suite Community Edition was employed as a proxy to intercept and analyze HTTP requests and responses.
3. Explored the Session Hijacking module within WebGoat to understand the vulnerabilities associated with session management.

Steps Taken:

1. Log into WebGoat with the created username and password.
2. Intercepted the HTTP request/response using Burp Suite while logging into the application.
3. Identified and copied the session tokens associated with active sessions.
4. Attempted to replace the session token with the token of another active session.
5. Executed the session hijacking by swapping the tokens in the intercepted request and sending it to the server.
6. Monitored the response to check for successful session hijacking.

Results and Findings:

Successfully intercepted and replaced session tokens between two active sessions.

Achieved session hijacking by substituting the session token of one user with that of another, gaining unauthorized access to the victim's account.

Discovered the potential risks associated with weak session management and the importance of securing session tokens.

A computer screen with a white and black text

Description automatically generated with medium confidence

Analysis and Conclusions:

Session hijacking can lead to severe security breaches by granting unauthorized access to sensitive user accounts.

Emphasizes the critical need for robust session handling mechanisms, including encryption, randomization, and short session lifetimes.

Recommendations:

Implement secure session management practices, such as session token encryption and validation.

Employ mechanisms like rotating session IDs and employing secure HTTP-only cookies to mitigate session hijacking risks.

Reflection:

This exercise provided practical insights into the vulnerabilities associated with session management. In future assessments, I will focus on exploring various mitigation strategies and implementing best practices to secure sessions effectively.

References:

Burp Suite Community Edition - <https://portswigger.net/burp/communitydownload>

Exercise Title and Objective: A1-Insecure Direct Object References

This exercise aims to identify and exploit insecure direct object references within the WebGoat application, allowing unauthorized access to resources.

Methodology:

1. Accessed the WebGoat application through a web browser.
2. Utilized Burp Suite Community Edition as a proxy to intercept and analyze HTTP requests and responses.
3. Explored the Insecure Direct Object References module within WebGoat to understand the vulnerabilities associated with access control.

Steps Taken:

1. Logged into WebGoat and navigated to the Insecure Direct Object References module.
2. Identified the vulnerable URLs and attempted to manipulate the parameters to access unauthorized resources.
3. Intercepted HTTP requests using Burp Suite and analyzed parameters associated with accessing user data or files.
4. Modified the URL parameters to access sensitive resources belonging to other users or roles.
5. Attempted to access files and user data by incrementing or changing parameters within the URL.
6. Observed responses to confirm successful access to unauthorized resources.

Results and Findings:

Successfully exploited insecure direct object references, gaining access to resources that should have been restricted.

Accessed sensitive user data and files belonging to other users or roles by manipulating parameters within the URL.

Demonstrated the significance of robust access control mechanisms and the risks associated with improper implementation.

A computer screen shot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Analysis and Conclusions:

Insecure direct object references pose severe threats by allowing unauthorized access to sensitive resources.

Emphasizes the critical need for proper access controls, authorization checks, and validated input to prevent such vulnerabilities.

Recommendations:

Implement proper access control mechanisms, including strict authorization checks based on user roles and permissions.

Utilize indirect references or identifiers that do not expose sensitive information in URLs to mitigate insecure direct object reference vulnerabilities.

Reflection:

This exercise highlighted the dangers of insecure direct object references and the importance of stringent access controls. In future evaluations, I will focus on refining my understanding of access control vulnerabilities and implementing secure practices.

References:

Burp Suite Community Edition - <https://portswigger.net/burp/communitydownload>

Exercise Title and Objective: A1- Missing Function Level Access Control

The objective of this exercise is to identify and exploit vulnerabilities related to missing function level access controls within the WebGoat application.

Methodology:

* Start the module and identify hidden elements in the webpage.
* Use browser inspection tools (like right-clicking and selecting "Inspect") to explore the HTML and CSS of the webpage.
* Within the inspection view, locate elements marked as hidden or with classes/IDs that suggest concealment.
* Look for specific elements like anchor tags or lists that might contain restricted data.

Steps taken:Retrieving Password Hash

* Use Burp Suite (or any intercepting proxy tool) to intercept requests sent by the webpage.
* Identify endpoints like "user-hash" or similar, which may reveal hashed passwords.
* Experiment with different request formats (e.g., x-www-form-urlencoded, application/json) and methods (GET, POST) to access data.
* Extract user hashes from the database by specifying appropriate parameters in the request.

Dealing with Broken Sections

* Examine the webpage and source code for missing functionalities.
* Analyze Java source code files to understand the password encryption process.
* Write code in Java to replicate the encryption process using various salts.
* Obtain the hashed password and submit it within WebGoat to complete the module.

Results and Findings:

Discovered instances where user roles lacked proper access controls, allowing unauthorized access to sensitive functionalities or data.

Successfully accessed restricted sections of the application, demonstrating a lack of proper access control enforcement.

Analysis and Conclusions:

Identified the critical importance of implementing robust access control mechanisms to prevent unauthorized access to sensitive functionalities.

Highlighted the risks associated with missing function level access control, potentially leading to unauthorized data exposure or system manipulation.

Recommendations:

Suggested implementing granular access controls based on user roles to restrict functionalities as per authorization levels.

Advised conducting regular security audits to ensure proper access controls are in place and functioning effectively.

Reflection:

This exercise offered insights into the significance of robust access control measures. For future engagements, I aim to explore more complex access control scenarios and their potential impact on application security.

References:

OWASP Access Control Cheat Sheet

Web Application Security Best Practices - Documentation and Guides

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA computer screen with a message

Description automatically generated

Exercise Title and Objective: A1-Spoofing an Authentication Cookie

The objective of this exercise is to simulate the exploitation of a vulnerability related to authentication cookies within the WebGoat application. The aim is to gain a practical understanding of how an attacker could manipulate authentication cookies to gain unauthorized access.

Methodology:

The methodology involved using browser developer tools and interception proxies like Burp Suite. These tools aided in inspecting cookies, intercepting and modifying requests, and manipulating the authentication cookie value. The exercise focused on altering the cookie content to impersonate a legitimate user.

Steps Taken:

1. Accessed the "Spoofing an Authentication Cookie" module within WebGoat.
2. Studied the basics of authentication cookies and their significance in web sessions.
3. Inspected cookies using browser developer tools to understand the authentication cookie's structure.
4. Intercepted HTTP requests and responses using Burp Suite.
5. Located the authentication cookie within the intercepted requests.
6. Edited the intercepted request to modify the authentication cookie value.
7. Replaced the original cookie value with a manipulated or spoofed value.
8. Forwarded the modified request containing the altered cookie value.

Challenges Encountered and Addressed:

One challenge was identifying the correct cookie to manipulate among multiple cookies. This was overcome by carefully inspecting and analyzing each cookie's content and purpose.

Results and Findings:

1. Vulnerabilities discovered: The exercise demonstrated the vulnerability of improperly handled authentication cookies, allowing unauthorized access.
2. Gained access: Successfully gained access to parts of the application by spoofing the authentication cookie.
3. Unexpected outcomes: None were observed during this exercise.
4. Analysis and Conclusions:
5. The findings underscore the criticality of securely handling authentication cookies. Exploiting this vulnerability could lead to unauthorized access, compromising user accounts and sensitive data. It highlights the importance of robust cookie management and session handling in securing web applications.

Recommendations:

Implement secure cookie practices, including encryption and validation.

Employ session tokens instead of directly using user credentials in cookies.

Regularly update and validate session IDs to prevent spoofing attacks.

Reflection:

This exercise provided practical insights into cookie-based authentication vulnerabilities. It emphasized the need for stringent security measures to safeguard against cookie spoofing. Future approaches might involve exploring advanced cookie manipulation techniques and experimenting with cookie attributes for enhanced security.

References:

Burp Suite for intercepting requests.

Browser developer tools for inspecting cookies.

A computer screen with a message

Description automatically generated

A2 cryptographic failures

Exercise Title and Objective: A2-Crypto Basics - Understanding Cryptography in Web Applications

This exercise aims to familiarize oneself with fundamental cryptographic concepts and their implementation within web applications. The focus is on comprehending encryption, hashing, and related practices used to secure data.

Methodology:

This exercise involved studying cryptographic techniques in web applications, specifically within the WebGoat environment. The exercise delved into understanding encryption and hashing algorithms, their significance, and their application in securing sensitive information.

Steps Taken:

I embarked on a step-by-step journey to grasp the fundamentals of encoding, encryption, hashing, and signatures. Before delving into the core content, I familiarized myself with critical Unix commands like echo, su, ls, cd, cat, and rm while also getting an introduction to encoding techniques such as Base64, XOR encoding, and URL encoding. Each section after that is involved.

In the Base64 Encoding section, I explored the concept of Base64 encoding and its relevance in HTTP authorization headers. Decoding an authorization header revealed a username and password using an online tool. Moving to Other Encoding, I delved deeper into different encoding types, focusing on XOR encoding. Employing an online tool decoded an XOR Cipher, unveiling the string "database password. "The Plain Hashing section revolved around investigating hashes without explicit hash type indications. Utilizing a hash identifier tool and online resources, I identified MD5 and SHA-256 hashes and decrypted the provided hashed values. In the Signatures section, I gained insights into symmetric and asymmetric encryption's connection to signatures. I followed instructions to decode an RSA private key, derived a modulus, and created a signature using a bash script and Java program. Finally, Collecting the Hidden Key involved Docker exploration to access a hidden RSA private key within a container. Manipulating file permissions facilitated root access, allowing retrieval of a password from a file. This extracted password was used to decrypt a hidden message.

Challenges Encountered and Addressed:

Understanding the nuances of various cryptographic algorithms posed a challenge, but it was mitigated by referring to additional documentation and resources available within the WebGoat platform.

Results and Findings:

Insights into encryption and hashing: Gained a better understanding of how encryption and hashing work in web applications.

Importance of secure practices: Realized the significance of robust cryptographic measures to protect sensitive data.

Analysis and Conclusions:

The exercise highlighted the critical role of cryptography in securing data within web applications. It emphasized the need for strong encryption algorithms and proper key management practices to safeguard against data breaches.

Recommendations:

Implement robust encryption standards such as AES for data protection.

Employ secure hashing algorithms like SHA-256 for password storage.

Regularly update encryption keys and algorithms to stay ahead of evolving threats.

Reflection:

References:

WebGoat platform for hands-on exercises.

Online resources and documentation on encryption and hashing techniques.

A screen shot of a computer

Description automatically generated

Find the secret



Go into the Docker with the help of bash



Read the default secret.





A7 Identity and Authorization Failure

Exercise Title: A7-Authentication Bypasses in Web Applications

Objective: This exercise aims to identify, understand, and exploit authentication vulnerabilities within web applications. By examining different authentication methods, the goal is to uncover weaknesses that could potentially allow unauthorized access.

Methodology:

For this exercise, I utilized WebGoat, an intentionally vulnerable web application, to simulate real-world scenarios of authentication bypasses. Employed strategies included testing for weak password controls, manipulating authentication tokens, and examining session management flaws. The tools used encompass Burp Suite for intercepting and analyzing web traffic, SQL injection techniques, and manual URL tampering.

Steps Taken:

This exercise involves an authentication bypass in WebGoat's Section 2 of A7. To start, I turned on Intercept and Burp Suite. Then, I navigated to the WebGoat page, scrolled down, and submitted any value for both questions below. After submitting, I returned to Burp Suite, where I found my request with parameters for question 0 and question 1.

The backend checks for specific values in these parameters. By altering the values from 0 to 'A' and 1 to 'B,' I satisfied the checks we discussed earlier. I forwarded the modified request, and this action exploited a realistic authentication bypass.

Results and Findings:

Discovered multiple authentication vulnerabilities. Successfully gained unauthorized access to restricted areas of the application.

Analysis and Conclusions:

These vulnerabilities could lead to severe consequences in a real-world scenario, such as unauthorized access to sensitive data or accounts. This exercise emphasized the criticality of robust authentication measures in preventing unauthorized access.

Recommendations:

Implement robust input validation and output encoding to prevent SQL injection attacks. Enhance session management by employing secure session handling practices. Enforce stringent access controls to restrict unauthorized access to sensitive functionalities.

Reflection:

This exercise highlighted the need for continuous vigilance in securing authentication mechanisms within web applications. Future approaches would involve more extensive testing scenarios and a deeper exploration of advanced authentication bypass techniques.

References:

WebGoat platform for hands-on exercises

Burp Suite for web traffic interception and analysis

A screenshot of a computer

Description automatically generated

Exercise Title and Objective: A7- Insecure Login

The objective is to exploit an insecure login functionality within WebGoat to gain unauthorized access by bypassing authentication controls.

Methodology:

1. Open WebGoat and navigate to the insecure login section.
2. Study the login mechanism to understand its vulnerabilities.
3. Turn on intercept in a tool like Burp Suite.
4. Submit any username and password to log in.
5. Review the intercepted request in Burp Suite to understand the parameters being sent.
6. Look for potential weaknesses in the authentication process.

Steps Taken:

1. Enabled intercept in Burp Suite.
2. Logged into the insecure login section of WebGoat.
3. Inspected the intercepted request in Burp Suite.
4. Noticed that the application checks credentials for authorization.
5. Copy the username and password from the intercept request and paste them into the WebGoat insecure login as credentials.
6. Logged in successfully without valid credentials.

Results and Findings:

Vulnerabilities Discovered: Identified an insecure login mechanism allowing unauthorized access.

Access Gained: Successfully logged in without legitimate credentials.

Implications: Highlighted the critical security loophole in the login process, posing a significant risk of unauthorized access.

Analysis and Conclusions:

The exercise demonstrated the vulnerability of the insecure login mechanism, emphasizing the importance of robust authentication controls in web applications.

Recommendations:

Implement stronger authentication measures, such as multi-factor authentication.

Validate user input thoroughly to prevent authentication bypasses.

Regularly conduct security assessments and penetration tests to identify and address vulnerabilities.

Reflection:

This exercise underscored the ease of exploiting insecure login methods and emphasized the need for stringent authentication protocols to mitigate such risks.

References:

WebGoat platform, Burp Suite interception tool.

A screenshot of a computer

Description automatically generated

Exercise Title and Objective: A7-Secure passwords

The objective is to enhance password security within a web application, emphasizing strong encryption and storage practices for user passwords.

Methodology:

1. Assess the existing password storage mechanism.
2. Employ robust encryption algorithms like bcrypt or Argon2 for password hashing.
3. Add unique salts to each password before encryption to enhance security.
4. Ensure secure storage of encrypted passwords to prevent unauthorized access.
5. Verify the effectiveness of password security measures by testing various authentication scenarios.

Steps Taken:

1. Evaluated the current password storage method to identify weaknesses.
2. Updated password storage by implementing bcrypt encryption and salting.
3. I reviewed the backend to ensure the secure storage of encrypted passwords.
4. Conducted various authentication tests to validate the updated password security.

Results and Findings:

Implemented bcrypt encryption with unique salts for password storage.

Ensured passwords are securely stored, minimizing the risk of unauthorized access.

Tested authentication mechanisms to verify the effectiveness of the updated security measures.

Analysis and Conclusions:

The exercise highlighted the significance of employing strong encryption techniques and secure storage practices to safeguard user passwords within a web application.

Recommendations:

Continue using strong encryption algorithms like bcrypt or Argon2.

Regularly update and review password security practices to align with evolving threats.

Conduct comprehensive security audits to identify potential vulnerabilities in password storage.

Reflection:

The exercise underscored the importance of implementing robust password security measures to protect user data, emphasizing the need to improve security practices continuously.

References:

Documentation on bcrypt or Argon2 encryption methods, best practices for secure password storage in web applications.

A screenshot of a computer

Description automatically generated