Testing For Common Attacks

Course Introduction

Alexis Ahmed

Senior Penetration Tester @HackerSploit
Offensive Security Instructor @INE

aahmed@ine.com
@HackerSploit
@alexisahmed

Course Topic Overview

- HTTP Method & Authentication Testing
- Sensitive Data Exposure
- Broken Authentication Attacks (Attacking Login Forms, Bypassing Authentication etc)
- Session Security Testing (Session Hijacking, Session Fixation & CSRF)
- Injection & Input Validation Attacks (Command Injection, Code Injection)
- Testing For Security Misconfigurations
- Exploiting Vulnerable & Outdated Components

Prerequisites

- Basic familiarity with HTTP
- Basic Familiarity With Burp Suite/OWASP ZAP
- Basic familiarity with Linux

Learning Objectives:

- You will be able to perform HTTP method tampering.
- You have the ability to attack sites using Basic HTTP Authentication and HTTP Digest Authentication.
- You will have an understanding of what causes sensitive data exposure vulnerabilities.
- You will have the ability to perform authentication testing in the form of attacking login forms and bypassing authentication.
- You will have an understanding of how session management works and the role tokens and cookies play in session management and security
- You will be able to identify and exploit session hijacking, session fixation and CSRF vulnerabilities.
- You will be able to identify and exploit command injection and code injection vulnerabilities.
- You will be able to identify and exploit security misconfigurations in web servers and other outdated and vulnerable components.

Let’s Get Started!

HTTP Method Tampering

HTTP Method Tampering

- HTTP method tampering, also known as HTTP verb tampering, is a type of security vulnerability that can be exploited in web applications. It occurs when an attacker manipulates the HTTP request method used to interact with a web server.

- HTTP requests typically use methods like GET, POST, PUT, DELETE, etc., to perform specific actions on a web application.

HTTP Methods

- GET: Used for retrieving data from the server. It should not have any side effects on the server or the application.
- POST: Used for submitting data to the server, often for actions that modify data on the server, like submitting a form.
- PUT: Used for updating a resource on the server with a new representation. It should be idempotent, meaning multiple requests should have the same effect as a single request.
- DELETE: Used for removing a resource from the server.
- OPTIONS: Used to query the server about the communication options and requirements for a specific resource, such as a URL or endpoint.

HTTP Method Tampering Process

- HTTP method tampering occurs when an attacker modifies the HTTP method being used in a request to trick the web application into performing unintended actions. For example:
  - Changing a GET request to a DELETE request: If the application doesn't properly validate the method used, it might inadvertently delete data when it should only be retrieving it.
  - Changing a POST request to a GET request: This could expose sensitive data that should only be accessible via a POST request.
  - Changing a GET request to a POST request: This might lead to unintended data modification if the application doesn't validate the method and payload correctly.

Demo: HTTP Method Tampering

Attacking Basic HTTP Authentication

Basic HTTP Authentication

- Basic HTTP authentication is a simple authentication mechanism used in web applications and services to restrict access to certain resources or functionalities.
- It's considered "basic" because it's uncomplicated and relies on a straightforward username and password combination. However, it's important to note that basic HTTP authentication is not secure on its own when used over an unencrypted connection (HTTP).
- It should only be used over HTTPS to ensure that the credentials are transmitted securely.

How Basic HTTP Authentication Works

- Client Request: When a client (usually a web browser) makes a request to a protected resource on a server, the server responds with a 401 Unauthorized status code if the resource requires authentication.

- Challenge Header: In the response, the server includes a WWW-Authenticate header with the value "Basic." This header tells the client that it needs to provide credentials to access the resource.

How Basic HTTP Authentication Works

- Credential Format: The client constructs a string in the format username:password and encodes it in Base64. It then includes this encoded string in an Authorization header in subsequent requests. The header looks like this:

```
Authorization: Basic base64-encoded-credentials
```

- For example, if the username is "user" and the password is "pass," the header would be:

```
Authorization: Basic dXNlcjpwYXNz
```

How Basic HTTP Authentication Works

- Server Validation: When the server receives the request with the Authorization header, it decodes the Base64-encoded credentials, checks them against its database of authorized users, and grants access if the credentials are valid.

- Access Granted or Denied: If the credentials are valid, the server allows access to the requested resource by responding with the resource content and a 200 OK status code. If the credentials are invalid or missing, it continues to respond with 401 Unauthorized.

Demo: Attacking Basic HTTP Authentication

Attacking HTTP Digest Authentication

HTTP Digest Authentication

- HTTP Digest Authentication is an authentication mechanism used in web applications and services to securely verify the identity of users or clients trying to access protected resources.
- It addresses some of the security limitations of Basic Authentication by employing a challenge-response mechanism and hashing to protect user credentials during transmission.
- However, like Basic Authentication, it's important to use HTTPS to ensure the security of the communication.

How HTTP Digest Authentication Works

- Client Request: When a client (usually a web browser) makes a request to a protected resource on a server, the server responds with a 401 Unauthorized status code if authentication is required.

- Challenge Header: In the response, the server includes a WWW-Authenticate header with the value "Digest." This header provides information needed by the client to construct a secure authentication request.

How HTTP Digest Authentication Works

Example of WWW-Authenticate header:

```
WWW-Authenticate: Digest realm="Example", qop="auth",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093", opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

How HTTP Digest Authentication Works

- realm: A descriptive string indicating the protection space (usually the name of the application or service).
- qop (Quality of Protection): Specifies the quality of protection. Commonly set to "auth."
- nonce: A unique string generated by the server for each request to prevent replay attacks.
- opaque: An opaque value set by the server, which the client must return unchanged in the response.

How HTTP Digest Authentication Works

- Client Response: The client constructs a response using the following components:
  - Username
  - Realm
  - Password
  - Nonce
  - Request URI (the path to the protected resource)
  - HTTP method (e.g., GET, POST)
  - cnonce (a client-generated nonce)
  - qop (the quality of protection)
  - H(A1) and H(A2), which are hashed values derived from the components.

How HTTP Digest Authentication Works

- It then calculates a response hash (response) based on these components and includes it in an Authorization header.

- Example Authorization header:

```
Authorization: Digest username="user", realm="Example",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093", uri="/resource", qop=auth, nc=00000001,
cnonce="0a4f113b", response="6629fae49393a05397450978507c4ef1",
opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

How HTTP Digest Authentication Works

- Server Validation: The server receives the request with the Authorization header and validates the response hash calculated by the client. It does this by reconstructing the same components and calculating its own response hash.
- If the hashes match, the server considers the client authenticated and grants access to the requested resource.

Demo: Attacking HTTP Digest Authentication

Sensitive Data Exposure Vulnerabilities

Sensitive Data Exposure

- Sensitive data exposure vulnerabilities refer to security flaws in a system that lead to the unintended exposure of confidential or sensitive information.

- These vulnerabilities can have serious consequences, including data breaches, privacy violations, and financial losses.

Sensitive Data Exposure Examples

- Weak Password Storage: Storing passwords in plaintext or using weak hashing algorithms without salting, making it easier for attackers to retrieve user passwords from a compromised database.
- Information Disclosure in Error Messages: Revealing sensitive data, such as system paths, database details, or user credentials, in error messages or logs that could aid attackers in exploiting the system.
- Directory Traversal: Allowing users to manipulate file paths in requests to access files and directories outside their intended scope, potentially exposing sensitive files.
- Unencrypted Backups: Storing backups of sensitive data without encryption or proper access controls, making them vulnerable if they are stolen.

Demo: Sensitive Data Exposure Vulnerabilities

Attacking Login Forms With Burp Suite

Demo: Attacking Login Forms With Burp Suite

Attacking Login Forms With OTP Security

OTP Security

- OTP (One-Time Password) security is a two-factor authentication (2FA) method used to enhance the security of user accounts and systems.
- OTPs are temporary, single-use codes that are typically generated and sent to the user's registered device (such as a mobile phone) to verify their identity during login or transaction processes.
- The primary advantage of OTPs is that they are time-sensitive and expire quickly, making them difficult for attackers to reuse.

OTP Security Methods

- Time-Based OTPs (TOTP): TOTP is a widely used OTP method that generates codes based on a shared secret key and the current time. These codes are typically valid for a short duration, often 30 seconds.

- SMS-Based OTPs: OTPs can be sent to users via SMS messages. When users log in, they receive an OTP on their mobile phone, which they must enter to verify their identity.

- Rate Limiting and Lockout: Implement rate limiting and account lockout mechanisms to prevent brute force attacks on OTPs. Lockout accounts after a certain number of failed OTP attempts.

OTP Rate Limiting

- OTP rate limiting is a security mechanism used to prevent brute force attacks or abuse of one-time password (OTP) systems, such as those used in two-factor authentication (2FA).
- Rate limiting restricts the number of OTP verification attempts that can be made within a specified time period.
- By enforcing rate limits, organizations can reduce the risk of attackers guessing or trying out multiple OTPs in quick succession.

Demo: Attacking Login Forms With OTP Security

Introduction To Session Management

Session Management

- Session management in web applications refers to the process of securely handling and maintaining user sessions.
- A session is a period of interaction between a user and a web application, typically beginning when a user logs in and ending when they log out or their session expires due to inactivity.
- During a session, the application needs to recognize and track the user, store their data, and manage their access to different parts of the application.
- Effective session management is crucial for security, user experience, and maintaining the state of a web application.

Session Management Components

- Session Identifier: A unique token (often a session ID) is assigned to each user's session. This token is used to associate subsequent requests from the user with their session data.

- Session Data: Information related to the user's session, such as authentication status, user preferences, and temporary data, is stored on the server.

- Session Cookies: Session cookies are small pieces of data stored on the user's browser that contain the session ID. They are used to maintain state between the client and server.

Importance of Session Management

- User Authentication: Session management is critical for user authentication. After a user logs in, the session management system keeps track of their authenticated state, allowing them to access protected resources without repeatedly entering credentials.
- User State: Web applications often need to maintain state information about a user's activities. For example, in an e-commerce site, the session management system keeps track of the items in a user's shopping cart.
- Security: Proper session management is essential for security. If not implemented correctly, it can lead to vulnerabilities such as session fixation, session hijacking, and unauthorized access.

Session Management - PHP

- In PHP, session management is relatively straightforward and is handled using built-in functions. Here's how it generally works:

- Session Start: To start a session, you use the session_start() function. This function initializes the session and generates a unique session ID for the user.

- Session Data: You can store and retrieve session data using the $_SESSION superglobal array.

- For example, $_SESSION['username'] = 'john_doe'; stores the username in the session.

Session Management - PHP

- Session Timeout: The session timeout is defined in the PHP configuration file (php.ini) using the session.gc_maxlifetime setting.

- Session ID Management: By default, PHP handles the generation of session IDs and their association with users.

Session Management Testing

- Session management testing is a crucial component of web application security testing.
- It involves assessing the effectiveness and security of how a web application manages user sessions.
- Proper session management testing helps identify vulnerabilities and weaknesses in session handling that could lead to security breaches, unauthorized access, or data leakage.

Session Management Testing

- Session Fixation Testing: Test for session fixation vulnerabilities by attempting to set a known session ID (controlled by the tester) and then login with another account. Verify if the application accepts the predefined session ID and allows the attacker access to the target account.
- Session Hijacking Testing: Test for session hijacking vulnerabilities by trying to capture and reuse another user's session ID. Tools like Wireshark or Burp Suite can help intercept and analyze network traffic for session data.
- Session ID Brute-Force: Attempt to brute force session IDs to assess their complexity and the application's resistance to such attacks.

Session IDs & Cookies

Session IDs & Cookies

- In the context of web application penetration testing, understanding session IDs and cookies is crucial, as these components play a significant role in user authentication and session management.

Session IDs

- Session IDs (Session Identifiers) are unique tokens or strings generated by web applications to identify and track user sessions. They are essential for maintaining stateful communication between the client (user's browser) and the server.
- Session IDs are typically used to associate requests from a user with their session data stored on the server.
- For example, suppose you're conducting a penetration test on an e-commerce website. After a user logs in, the server generates a session ID (e.g., "Session12345") and associates it with the user's session.
- This session ID is then sent to the user's browser as a cookie.

Cookies

- Cookies are small pieces of data (usually text) that a web server sends to the user's browser, which stores them locally.
- Cookies serve various purposes, such as session management, user tracking, and personalization. In the context of session management, session cookies are commonly used to store the session ID, allowing the server to recognize and maintain the user's session.
- For example, during a penetration test, you discover that the website uses cookies for session management. When a user logs in, the server sends a cookie named "sessionID" with the value "Session12345" to the user's browser. On subsequent requests, the browser includes this cookie, allowing the server to identify and associate the user's requests with their session.

Session Hijacking & Session Fixation

Session Hijacking

- Session hijacking, also known as session theft, is a security attack where an attacker illegitimately takes over a user's active session on a web application.
- In this type of attack, the attacker gains unauthorized access to the user's session token or identifier, allowing them to impersonate the victim and perform actions on their behalf.
- Session hijacking is a severe security threat because it can lead to unauthorized access to user accounts, sensitive data, and potential misuse of the hijacked session.

Session Hijacking - Token Acquisition

- Session Prediction: Predicting or guessing the session token, especially if it's predictable or lacks sufficient randomness.
- Session Sniffing: Intercepting the session token as it's transmitted over an unsecured network, such as an open Wi-Fi hotspot.
- Cross-Site Scripting (XSS): Exploiting a vulnerability in the web application to inject malicious JavaScript into a victim's browser, which can steal the session token.

Session Hijacking - Impersonation

- Once the attacker has the session token, they can impersonate the victim by presenting this token during requests to the web application.

- The application, unaware of the hijacking, treats the attacker as the authenticated user.

Session Hijacking - Impact

- Data Theft: Access and steal the victim's sensitive data, such as personal information, financial details, or confidential documents.
- Account Takeover: Change the victim's account settings, passwords, or email addresses, effectively locking the victim out of their account.
- Malicious Transactions: Conduct unauthorized transactions, make purchases, or manipulate the victim's data.
- Data Manipulation: Modify or delete the victim's data or settings.

Session Fixation

- Session fixation is a web application security attack where an attacker sets or fixes a user's session identifier (session token) to a known value of the attacker's choice.

- Subsequently, the attacker tricks the victim into using this fixed session identifier to log in, thereby granting the attacker unauthorized access to the victim's session.

Session Fixation - Token Acquisition

- The attacker obtains a session token issued by the target web application. This can be done in several ways, such as:
- Predicting or guessing the session token: Some web applications generate session tokens that are easy to predict or lack sufficient randomness.
- Intercepting the session token: If the application doesn't use secure channels (e.g., HTTPS) to transmit session tokens, an attacker may intercept them as they travel over an insecure network, such as an open Wi-Fi hotspot.

Session Fixation - Impersonation

- With a session token in hand, the attacker sets or fixes the victim's session token to a known value that the attacker controls. This value could be one generated by the attacker or an existing valid session token.

- The attacker lures the victim into using the fixed session token to log in to the web application. This can be accomplished through various means:
  - Sending the victim a link that includes the fixed session token.
  - Manipulating the victim into clicking on a specially crafted URL.
  - Social engineering tactics to convince the victim to log in under specific circumstances.

Session Fixation - Hijacking

- Once the victim logs in with the fixed session token, the attacker can now hijack the victim's session.

- The web application recognizes the attacker as the legitimate user since the session token matches what is expected.

Session Hijacking Via Cookie Tampering

Demo: Session Hijacking Via Cookie Tampering

Introduction To Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF)

- Cross-Site Request Forgery (CSRF) is a type of web security vulnerability that occurs when an attacker tricks a user into performing actions on a web application without their knowledge or consent.
- This attack takes advantage of the trust that a web application has in the user's browser.
- In the context of web application penetration testing, understanding CSRF is crucial for identifying and mitigating this security risk.

CSRF Attack Methodology

- In a CSRF attack, the attacker crafts a malicious request and tricks a user into unknowingly sending that request to a vulnerable web application.
- Web applications typically trust that requests coming from a user's browser are legitimate. However, CSRF exploits this trust.
- Most web applications use cookies for user authentication. When a user logs in, they receive a session cookie that identifies them during their session. This cookie is automatically sent with every request to the application.

CSRF Attack Methodology

- The attacker crafts a malicious request (e.g., changing the user's email address or password) and embeds it in a web page, email, or some other form of content.
- The attacker lures the victim into loading this content while the victim is authenticated in the target web application.
- The victim's browser automatically sends the malicious request, including the victim's authentication cookie.
- The web application, trusting the request due to the authentication cookie, processes it, causing the victim's account to be compromised or modified.

CSRF Impact

- CSRF attacks can have serious consequences:
  - Unauthorized changes to a user's account settings.
  - Fund transfers or actions on behalf of the user without their consent.
  - Malicious actions like changing passwords, email addresses, or profile information.

Advanced Electron Forum CSRF

Demo: Advanced Electron Forum CSRF

Command Injection

Command Injection

- Command injection vulnerabilities in the context of web application penetration testing occur when an attacker can manipulate the input fields of a web application in a way that allows them to execute arbitrary operating system commands on the underlying server.

- This type of vulnerability is a serious security risk because it can lead to unauthorized access, data theft, and full compromise of the web server.

Command Injection - Causes

- User Input Handling: Web applications often take user input through forms, query parameters, or other means.
- Lack of Input Sanitization: Insecurely coded applications may fail to properly validate, sanitize, or escape user inputs before using them in system commands.
- Injection Points: Attackers identify injection points, such as input fields or URL query parameters, where they can insert malicious commands.

Command Injection - Exploitation

- Malicious Input: Attackers craft input that includes special characters, like semicolons, pipes, backticks, and other shell metacharacters, to break out of the intended input context and inject their commands.
- Command Execution: When the application processes the attacker's input, it constructs a shell command using the malicious input.
- The server, believing the command to be legitimate, executes it in the underlying operating system.

Command Injection - Impact

- Unauthorized Execution: Attackers can execute arbitrary commands with the privileges of the web server process. This can lead to unauthorized data access, code execution, or system compromise.
- Data Exfiltration: Attackers can exfiltrate sensitive data, such as database content, files, or system configurations.
- System Manipulation: Attackers may manipulate the server, install malware, or create backdoors for future access.

Demo: Command Injection

PHP Code Injection

PHP Code Injection

- PHP code injection vulnerabilities, also known as PHP code execution vulnerabilities, occur when an attacker can inject and execute arbitrary PHP code within a web application.

- These vulnerabilities are a serious security concern because they allow attackers to gain unauthorized access to the server, execute malicious actions, and potentially compromise the entire web application.

PHP Code Injection - Exploitation

- Malicious Input: Attackers craft input that includes PHP code snippets, often enclosed within PHP tags (<?php ... ?>) or backticks (`).
- Code Execution: When the application processes the attacker's input, it includes the injected PHP code as part of a PHP script that is executed on the server.
- This allows the attacker to run arbitrary PHP code in the context of the web application.

Demo: PHP Code Injection

RCE Via MySQL

Demo: RCE Via MySQL

Testing For Common Attacks

Course Conclusion

Learning Objectives:

- You will be able to perform HTTP method tampering.
- You have the ability to attack sites using Basic HTTP Authentication and HTTP Digest Authentication.
- You will have an understanding of what causes sensitive data exposure vulnerabilities.
- You will have the ability to perform authentication testing in the form of attacking login forms and bypassing authentication.
- You will have an understanding of how session management works and the role tokens and cookies play in session management and security
- You will be able to identify and exploit session hijacking, session fixation and CSRF vulnerabilities.
- You will be able to identify and exploit command injection and code injection vulnerabilities.
- You will be able to identify and exploit security misconfigurations in web servers and other outdated and vulnerable components.

Thank You!

EXPERTS AT MAKING YOU AN EXPERT