Authentication Cheat Sheet

From OWASP

Contents

Introduction

Authentication is the process of verification that an individual or an entity is who it claims to be. Authentication is commonly performed by submitting a user name or ID and one or more items of private information that only a given user should know.

Session Management is a process by which a server maintains the state of an entity interacting with it. This is required for a server to remember how to react to subsiquent requests throughout a transaction. Sessions are maintained on the server by a session identifier which can be passed back and forward between the client and server when mtransmitting and receiving requests. Sessions should be unique per user and computationally very difficult to predict.

For more information on Authentication, please see the OWASP Guide to Authentication page.

Authentication General Guidelines

Implement Proper Password Strength Controls

A key concern when using passwords for authentication is password strength. A "strong" password policy makes it difficult or even improbable for one to guess the password either by using manual or automated means. The following characteristics define strong a strong password:

Password Length

The longer the password the more combinations possible combinations of characters exist and is hence more difficult to guess.

Important applications: Minimum of 6 characters in length.

Critical applications: Minimum of 8 characters in length. (consider multi-factor authentication)

Highly critical applications: Consider multi-factor authentication

Password complexity

Example
Passwords should be checked for the following composition or a variance of such:

- at least: 1 uppercase character (A-Z)

- at least: 1 lowercase character (a-z)
- at least: 1 digit (0-9)
- at least one special character (!"£$%&...)
- a defined minimum length (e.g. 8 chars)
- a defined maximum length (as with all external input)
- no contiguous characters (e.g. 123abcd)
- not more than 2 identical characters in a row (1111)

Implement Secure Password Recovery Mechanism

It is common for an application to have a mechanism that provides a means for a user to gain access to their account in the event they forget their password. Password recovery systems are difficult to secure against abuse, circumvention. The best way is to keep them as simple as possible.

Secret questions

Weaknesses with secret questions are common;

Weakness may be that the security question is too easy to guess or find an answer to E.g. Car, Date of Birth (DOB), Colour Such question have a a finite sample space and are easily brute forced/guessed.

Delivery Mechanisims / Implementation weakness

Common issues with a reset password delivery are

- Delivery:

Email: Setting the email address to that which is not the email address of the user for which the password is to be reset.

- Password Reset Rate

Limiting the rate at which password reset requests are processed in order to limit hijacking and brute force attempts.

- Authentication: Password recovery mechanism relies only on something the user knows and not something the user has; email account, token, multifactor.

Utilize Multi-Factor Authentication

Multifactor factor authentication is using more than one of:

- Something you know (account details or passwords)
- Something you have (tokens or mobile phones)
- Something you are (biometrics)

to logon or process a transaction.

Authentication schemes such as One Time Passwords (OTP) implemented using a hardware token can also be key in fighting attacks such as CSRF and client-side malware.

Authentication and Error Messages

Incorrectly implemented error messages in the case of authentication functionality can be used for the purposes of user ID and password enumeration.

An application should respond (both HTTP and HTML) in a generic manner which is not unique to the error condition or authentication failure.

Authentication responses

An application should respond with a generic error message regardless if the user ID or password was incorrect. It should also no give any indication to the status of an account if it exists.

Incorrect responses examples

- "Login for User foo: invalid password"
- "Login failed, invalid user ID"
- "Login failed; account disabled"
- "login failed; this user is not active"

Correct response example

- "Login failed; Invalid user ID or password"

The correct response does not indicate if the user ID or password is the incorrect parameter and hence inferring a valid user ID.

Error Codes and URL's

The application may return a different HTTP Error code depending on the authentication attempt response. It may respond with a 200 for a positive result and a 403 for a negative result. Even though a generc error page is shown to a user the HTTP response code may differ which can indicate a signature.

Transmit Passwords Only Over TLS

See: "Transport Layer Protection Cheat Sheet"

http://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet

"The login page and all subsequent authenticated pages must be exclusively accessed over TLS. The initial login page, referred to as the "login landing page", must be served over TLS. Failure to utilize TLS for the login landing page allows an attacker to modify the login form action, causing the user's credentials to be posted to an arbitrary location. Failure to utilize TLS for authenticated pages after the login enables an attacker to view the unencrypted session ID and compromise the user's authenticated session."

Implement Account Lockout

If an attacker is able to guess passwords without the account becoming disabled due to failed authentication attempts, this provides the opportunity of the attacked to continue with a brute force attack until the account is compromised.

Automating brute-force/password guessing attacks on web applications is a trivial challenge. Password lockout mechanisms should be employed that lock out an account if more than a preset number of unsuccessful login attempts are made.

Password lockout mechanisms do have a logical weakness, however. It is possible for an attacker to attempt a large number of authentication attempts on known account names resulting in locking out entire blocks of application users accounts.

Given that the intent of a password lockout system is to protect from brute-force attacks, a sensible strategy is to lockout accounts for a number of hours. This significantly slows down attackers, while allowing the accounts to be open for legitimate users.

See also "Reverse Brute-force" Reverse_Brute_Force

Session Management General

Guidelines

Transmit Session ID's Only Over TLS

Session ID's should be transmitted over a secure TLS channel.

A browser directive, the secure flag, for the cookie exists which instructs the browser at the set-cookie action only to send over a secure channel.

This prevents attacks such as Man-in-the-Middle (MitM) and surf jacking attacks.

Ensure Session ID's are Cryptographically Strong and Random

Session ID's should be large and random enough such that they are not easily predicted. This may lead to session hijacking and identity theft.

Be sure the session ID used has the following traits at a minimum:

- Session identifiers should be at least 128 bits long to prevent brute-force session guessing attacks.
- All session identifiers should be sufficiently randomized so as to not be guessable.
- All session identifiers should use the largest character set available to it.

Also see: http://www.owasp.org/index.php/Insufficient_Session-ID_Length http://www.owasp.org/index.php/Session_Management

Implement Idle And Absolute Timeout

Session Idle Timeout

All browser sessions should implement an inactivity timeout.

An inactivity timeout closes/invalidates a session upon a period of inactivity (Server has not received a request for a given session ID) has been reached.

Session Absolute Timeout

Regardless of session/user activity is is recommended to enforce a session timeout "maximum time limit" if possible. This maximum session time invalidates

the users session and forces them to log into the application again by re-authentication. If a session was hijacked via cookie theft, an absolute timeout may limit exposure to the user.

Cookie Security

Secure Flag

If the site requires operation over SSL ensure the secure flag is set upon the set-cookie directive sent the to the browser.

HTTP Only

The HTTP only directive, also sent to the browser can project the cookie from JavaScript manupulation. This is a compensating control against XSS cookie attacks.

HTTP GET Vs POST

Sending any sensitive data over a query string, using HTTP GET can give rise to information leakage at points through the traffic path or at either end of the SSL tunnel:

E.g. Logs (firewall, Router, TLS tunnel terminator, App/web server).

Query strings can also be cached in a browser's history.

Authentication over Query String regardless of the use of SSL is not recommended.

Related Articles

Other Articles in the OWASP Prevention Cheat Sheet Series

- **Authentication Cheat Sheet**
- Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet
- Forgot Password Cheat Sheet
- Cryptographic Storage Cheat Sheet
- SQL Injection Prevention Cheat Sheet
- Transport Layer Protection Cheat Sheet
- XSS (Cross Site Scripting) Prevention Cheat Sheet
- DOM based XSS Prevention Cheat Sheet

Authors and Primary Editors

Eoin Keary eoinkeary[at]owasp.org

Categories: How To | Cheatsheets | OWASP Document | OWASP Top Ten Project

- Powered by MediaWiki