

OWASP Technical Editing: “Before” and “After”

This section demonstrates my ability to refine and enhance technical documentation. This sample is a highly technical document from the Open Worldwide Application Security Project (OWASP)’s Cheat Sheet Project. The first part of the sample is the original “Before” un-edited cybersecurity “cheat sheet” for cybersecurity authentication. The second part is my “After” edit of the same “cheat sheet.”

Note that the “After” edit is better written and clearer. It is much easier to read and much more understandable than the “Before” section.

Authentication Cheat Sheet

Introduction

Authentication (**AuthN**) is the process of verifying that an individual, entity, or website is who or what it claims to be by determining the validity of one or more authenticators (like passwords, fingerprints, or security tokens) that are used to back up this claim.

Digital Identity is the unique representation of a subject engaged in an online transaction. A digital identity is always unique in the context of a digital service but does not necessarily need to be traceable back to a specific real-life subject.

Identity Proofing establishes that a subject is actually who they claim to be. This concept is related to KYC concepts and it aims to bind a digital identity with a real person.

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 subsequent requests throughout a transaction. Sessions are maintained on the server by a session identifier which can be passed back and forth between the client and server when transmitting and receiving requests. Sessions should be unique per user and computationally very difficult to predict. The [Session Management Cheat Sheet](Session_Management_Cheat_Sheet.md) contains further guidance on the best practices in this area.

Authentication General Guidelines

User IDs

The primary function of a User ID is to uniquely identify a user within a system. Ideally, User IDs should be randomly generated to prevent the creation of predictable or sequential IDs, which could pose a security risk, especially in systems where User IDs might be exposed or inferred from external sources.

Usernames

Usernames are easy-to-remember identifiers chosen by the user and used for identifying themselves when logging into a system or service. The terms User ID and username might be used interchangeably if the username chosen by the user also serves as their unique identifier within the system.

Users should be permitted to use their email address as a username, provided the email is verified during signup. Additionally, they should have the option to choose a username other than an email address. For information on validating email addresses, please visit the [input validation cheatsheet email discussion](Input_Validation_Cheat_Sheet.md#email-address-validation).

Authentication Solution and Sensitive Accounts

- Do **NOT** allow login with sensitive accounts (i.e. accounts that can be used internally within the solution such as to a back-end / middle-ware / DB) to any front-end user interface
- Do **NOT** use the same authentication solution (e.g. IDP / AD) used internally for unsecured access (e.g. public access / DMZ)

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 through either manual or automated means. The following characteristics define a strong password:

- Password Length
 - **Minimum** length of the passwords should be **enforced** by the application. Passwords **shorter than 8 characters** are considered to be weak ([NIST SP800-63B](<https://pages.nist.gov/800-63-3/sp800-63b.html>)).
 - **Maximum** password length should not be set **too low**, as it will prevent users from creating passphrases. A common maximum length is 64 characters due to limitations in certain hashing algorithms, as discussed in the [Password Storage Cheat Sheet]([Password_Storage_Cheat_Sheet.md#maximum-password-lengths](#)). It is important to set a maximum password length to prevent [long password Denial of Service attacks](<https://www.acunetix.com/vulnerabilities/web/long-password-denial-of-service/>).
- Do not silently truncate passwords. The [Password Storage Cheat Sheet]([Password_Storage_Cheat_Sheet.md#maximum-password-lengths](#)) provides further guidance on how to handle passwords that are longer than the maximum length.
- Allow usage of **all** characters including unicode and whitespace. There should be no password composition rules limiting the type of characters permitted.
- Ensure credential rotation when a password leak occurs, or at the time of compromise identification.
- Include a password strength meter to help users create a more complex password and block common and previously breached passwords
 - [zxcvbn-ts library](<https://github.com/zxcvbn-ts/zxcvbn>) can be used for this purpose.
 - [Pwned Passwords](<https://haveibeenpwned.com/Passwords>) is a service where passwords can be checked against previously breached passwords. You can host it yourself or use the [API](<https://haveibeenpwned.com/API/v3#PwnedPasswords>).

For more detailed information check

Authentication Cheat Sheet

FINAL EDIT

Introduction

****Authentication**** is the process of verifying that an individual, entity, or website is who/what it claims to be. Authentication in the context of web applications is commonly performed by submitting a username 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 subsequent requests throughout a transaction. Sessions are maintained on the server by a session identifier which can be passed back and forth between the client and server when transmitting and receiving requests. Sessions should be unique per user and computationally very difficult to predict. The [Session Management Cheat Sheet](Session_Management_Cheat_Sheet.md) contains further guidance on the best practices in this area.

Authentication General Guidelines

User IDs

Make sure your usernames/user IDs are case-insensitive. User 'smith' and user 'Smith' should be the same user. Usernames should also be unique. For high-security applications, usernames could be assigned and secret instead of user-defined public data.

Email address as a User ID

For information on validating email addresses, please visit the [input validation cheatsheet email discussion](Input_Validation_Cheat_Sheet.md#email-address-validation).

Authentication Solution and Sensitive Accounts

- Do ****NOT**** allow login with sensitive accounts (i.e. accounts that can be used internally within the solution such as to a back-end / middle-ware / DB) to any front-end user-interface
- Do ****NOT**** use the same authentication solution (e.g. IDP / AD) used internally for unsecured access (e.g. public access / DMZ)

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 through either manual or automated means. The following characteristics define a strong password:

- Password Length

FINAL EDIT

FINAL EDIT

- **Minimum** length of the passwords should be **enforced** by the application. Passwords **shorter than 8 characters** are considered to be weak ([NIST SP800-63B](https://pages.nist.gov/800-63-3/sp800-63b.html)).
- **Maximum** password length should not be set **too low**, as it will prevent users from creating passphrases. A common maximum length is 64 characters due to limitations in certain hashing algorithms, as discussed in the [Password Storage Cheat Sheet](Password_Storage_Cheat_Sheet.md#maximum-password-lengths). It is important to set a maximum password length to prevent [long password Denial of Service attacks](https://www.acunetix.com/vulnerabilities/web/long-password-denial-of-service/).
- Do not silently truncate passwords. The [Password Storage Cheat Sheet](Password_Storage_Cheat_Sheet.md#maximum-password-lengths) provides further guidance on how to handle passwords that are longer than the maximum length.
- Allow usage of **all** characters including unicode and whitespace. There should be no password composition rules limiting the type of characters permitted.
- Ensure credential rotation when a password leak occurs, or at the time of compromise identification.
- Include a password strength meter to help users create a more complex password and block common and previously breached passwords
 - [zxcvbn-ts library](https://github.com/zxcvbn-ts/zxcvbn) can be used for this purpose.
 - [Pwned Passwords](https://haveibeenpwned.com/Passwords) is a service where passwords can be checked against previously breached passwords. You can host it yourself or use the [API](https://haveibeenpwned.com/API/v3#PwnedPasswords).

For more detailed information check

- [ASVS v4.0 Password Security Requirements](https://github.com/OWASP/ASVS/blob/master/4.0/en/0x11-V2-Authentication.md#v21-password-security-requirements)
- [Passwords Evolved: Authentication Guidance for the Modern Era](https://www.troyhunt.com/passwords-evolved-authentication-guidance-for-the-modern-era/)

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. Please see [Forgot Password Cheat Sheet](Forgot_Password_Cheat_Sheet.md) for details on this feature.

Store Passwords in a Secure Fashion

It is critical for an application to store a password using the right cryptographic technique. Please see [Password Storage Cheat Sheet](Password_Storage_Cheat_Sheet.md) for details on this feature.

FINAL EDIT