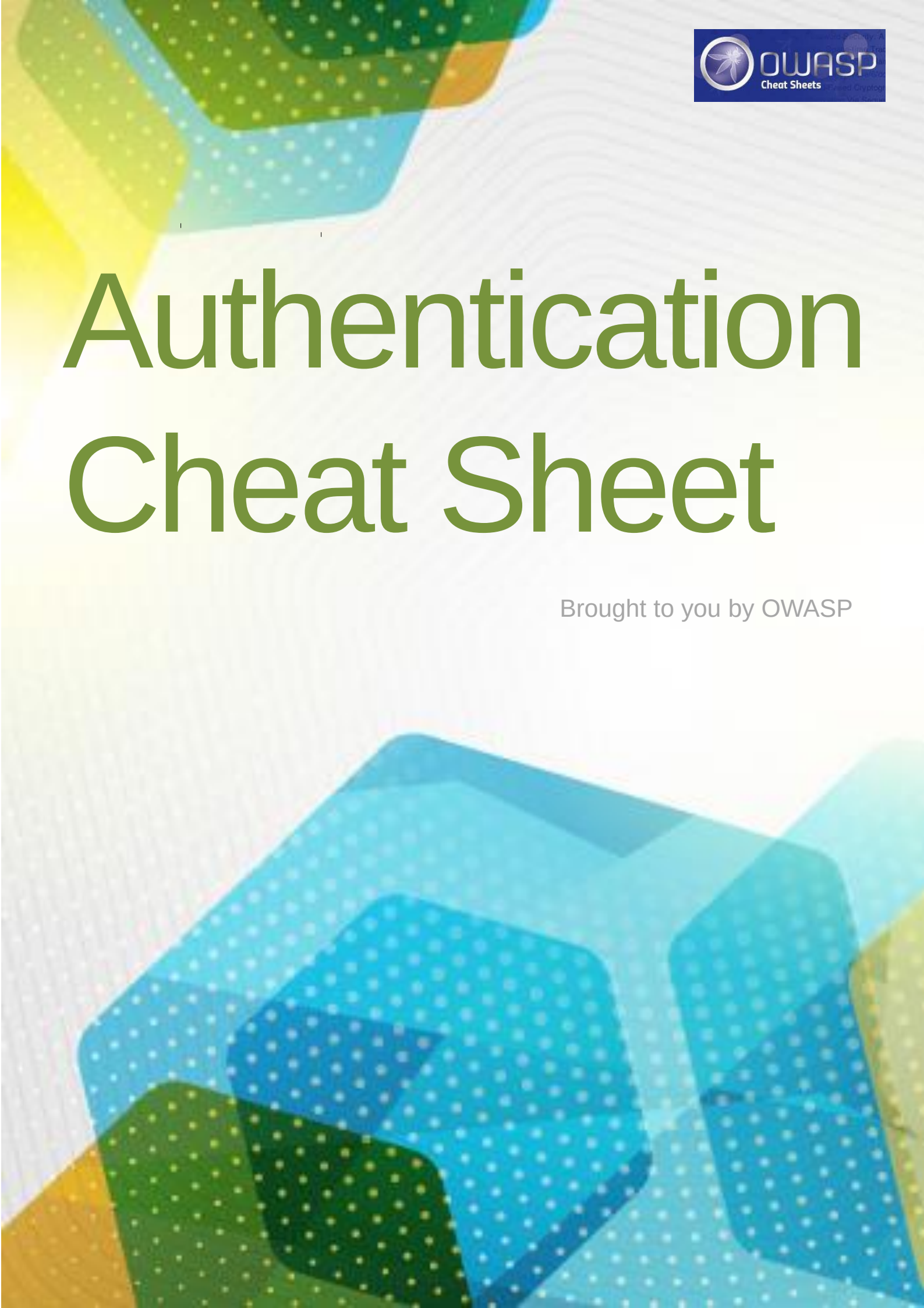

Authentication Cheat Sheet

Brought to you by OWASP

Authentication Cheat Sheet

Brought to you by OWASP Cheat Sheets

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 subsequent 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 transmitting and receiving requests. Sessions should be unique per user and computationally very difficult to predict.

Authentication General Guidelines

User IDs

Make sure your usernames/userids are case insensitive. Many sites use email addresses for usernames and email addresses are already case insensitive. Regardless, it would be very strange for user 'smith' and user 'Smith' to be different users. Could result in serious confusion.

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

Longer passwords provide a greater combination of characters and consequently make it more difficult for an attacker to guess.

- **Minimum** length of the passwords should be **enforced** by the application.
    - Passwords **shorter than 10 characters** are considered to be weak ([1]).

While minimum length enforcement may cause problems with memorizing passwords among some users, applications should encourage them to set *passphrases* (sentences or combination of words) that can be much longer than typical passwords and yet much easier to remember.

- **Maximum** password length should not be set **too low**, as it will prevent users from creating passphrases. Typical maximum length is 128 characters.

    - Passphrases shorter than 20 characters are usually considered weak if they only consist of lower case Latin characters.

- Every character counts!!
    - Make sure that every character the user types in is actually included in the password. We've seen systems that truncate the password at a length shorter than what the user provided (e.g., truncated at 15 characters when they entered 20).
    - This is usually handled by setting the length of ALL password input fields to be exactly the same length as the maximum length password. This is particularly important if your max password length is short, like 20-30 characters.

Password Complexity

Applications should enforce password complexity rules to discourage easy to guess passwords. Password mechanisms should allow virtually any character the user can type to be part of their password, including the space character. Passwords should, obviously, be case sensitive in order to increase their complexity. Occasionally, we find systems where passwords aren't case sensitive, frequently due to legacy system issues like old mainframes that didn't have case sensitive passwords.

The password change mechanism should require a minimum level of complexity that makes sense for the application and its user population. For example:

- Password must meet at least 3 out of the following 4 complexity rules
    - at least 1 uppercase character (A-Z)
    - at least 1 lowercase character (a-z)
    - at least 1 digit (0-9)
    - at least 1 special character (punctuation) — do not forget to treat space as special characters too
- at least 10 characters
- at most 128 characters
- not more than 2 identical characters in a row (e.g., 111 not allowed)

As application's require more complex password policies, they need to be very clear about what these policies are.

- The required policy needs to be explicitly stated on the password change page
    - be sure to list every special character you allow, so it's obvious to the user

Recommendation:

- Ideally, the application would indicate to the user as they type in their new password how much of the complexity policy their new password meets
    - In fact, the submit button should be grayed out until the new password meets the complexity policy and the 2nd copy of the new password matches the 1st. This

will make it far easier for the user to understand and comply with your complexity policy.

Regardless of how the UI behaves, when a user submits their password change request:

- If the new password doesn't comply with the complexity policy, the error message should describe EVERY complexity rule that the new password does not comply with, not just the 1st rule it doesn't comply with

Changing passwords should be EASY, not a hunt in the dark.

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 for details on this feature.

Store Passwords in a Secure Fashion

It is critical for a application to store a password using the right cryptographic technique. Please see Password Storage Cheat Sheet for details on this feature.

Transmit Passwords Only Over TLS

See: 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.

Require Re-authentication for Sensitive Features

In order to mitigate CSRF and session hijacking, it's important to require the current credentials for an account before updating sensitive account information such as the user's password, user's email, or before sensitive transactions, such as shipping a purchase to a new address. Without this countermeasure, an attacker may be able to execute sensitive transactions through a CSRF or XSS attack without needing to know the user's current credentials. Additionally, an attacker may get temporary physical access to a user's browser or steal their session ID to take over the user's session.

Utilize Multi-Factor Authentication

Multi-factor authentication (MFA) is using more than one authentication factor to logon or process a transaction:

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

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. A number of hardware tokens suitable for MFA are available in the market that allow good integration with web applications. See: [2].

SSL Client Authentication

SSL Client Authentication, also known as two-way SSL authentication, consists of both, browser and server, sending their respective SSL certificates during the TLS handshake process. Just as you can validate the authenticity of a server by using the certificate and asking a well known Certificate Authority (CA) if the certificate is valid, the server can authenticate the user by receiving a certificate from the client and validating against a third party CA or its own CA. To do this, the server must provide the user with a certificate generated specifically for him, assigning values to the subject so that these can be used to determine what user the certificate should validate. The user installs the certificate on a browser and now uses it for the website.

It is a good idea to do this when:

- It is acceptable (or even preferred) that the user only has access to the website from only a single computer/browser.
- The user is not easily scared by the process of installing SSL certificates on his browser or there will be someone, probably from IT support, that will do this for the user.
- The website requires an extra step of security.
- It is also a good thing to use when the website is for an intranet of a company or organization.

It is generally not a good idea to use this method for widely and publicly available websites that will have an average user. For example, it wouldn't be a good idea to implement this for a website like Facebook. While this technique can prevent the user from having to type a password (thus protecting against an average keylogger from stealing it), it is still considered a good idea to consider using both a password and SSL client authentication combined.

For more information, see: [3] or [4]

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.

Authentication Responses

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

Incorrect Response 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 userID 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 generic error page is shown to a user, the HTTP response code may differ which can leak information about whether the account is valid or not.

Prevent Brute-Force Attacks

If an attacker is able to guess passwords without the account becoming disabled due to failed authentication attempts, the attacker has an opportunity 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 have a logical weakness. An attacker that undertakes a large number of authentication attempts on known account names can produce a result that locks out entire blocks of user 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 period of time (e.g., 20 minutes). This significantly slows down attackers, while allowing the accounts to reopen automatically for legitimate users.

Also, multi-factor authentication is a very powerful deterrent when trying to prevent brute force attacks since the credentials are a moving target. When multi-factor is implemented and active, account lockout may no longer be necessary.

Use of authentication protocols that require no password

While authentication through a user/password combination and using multi-factor authentication is considered generally secure, there are use cases where it isn't considered the best option or even safe. An example of this are third party applications that desire connecting to the web application, either from a mobile device, another website, desktop or other situations. When this happens, it is NOT considered safe to allow the third party application to store the user/password combo, since then it extends the attack surface into their hands, where it isn't in your control. For this, and other use cases, there are several authentication protocols that can protect you from exposing your users' data to attackers.

OAuth

Open Authorization (OAuth) is a protocol that allows an application to authenticate against a server as a user, without requiring passwords or any third party server that acts as an identity provider. It uses a token generated by the server, and provides how the authorization flows most occur, so that a client, such as a mobile application, can tell the server what user is using the service.

The recommendation is to use and implement OAuth 2.0, since the first version has been found to be vulnerable to session fixation.

OAuth 2.0 is currently used and implemented by API's from companies such as Facebook, Google, Twitter and Microsoft.

OpenId

OpenId is an HTTP-based protocol that uses identity providers to validate that a user is who he says he is. It is a very simple protocol which allows a service provider initiated way for single sign-on (SSO). This allows the user to re-use a single identity given to a trusted OpenId identity provider and be the same user in multiple websites, without the need to provide any website the password, except for the OpenId identity provider.

Due to its simplicity and that it provides protection of passwords, OpenId has been well adopted. Some of the well known identity providers for OpenId are Stack Exchange, Google, Facebook and Yahoo!

For non-enterprise environment, OpenId is considered a secure and often better choice, as long as the identity provider is of trust.

SAML

Security Assertion Markup Language (SAML) is often considered to compete with OpenId. The most recommended version is 2.0, since it is very feature complete and provides a strong security. Like with OpenId, SAML uses identity providers, but unlike it, it is XML-based and provides more flexibility. SAML is based on browser redirects which send XML data. Unlike SAML, it isn't only initiated by a service provider, but it can also be initiated from the identity provider. This allows the user to navigate through different portals while still being authenticated without having to do anything, making the process transparent.

While OpenId has taken most of the consumer market, SAML is often the choice for enterprise applications. The reason for this is often that there are few OpenId identity providers which are considered of enterprise class (meaning that the way they validate the user identity doesn't have high standards required for enterprise identity). It is more common to see SAML being used inside of intranet websites, sometimes even using a server from the intranet as the identity provider.

In the past few years, applications like SAP ERP and SharePoint (SharePoint by using Active Directory Federation Services 2.0) have decided to use SAML 2.0 authentication as an often preferred method for single sign-on implementations whenever enterprise federation is required for web services and web applications.

Session Management General Guidelines

Session management is directly related to authentication. The **Session Management General Guidelines** previously available on this OWASP Authentication Cheat Sheet have been integrated into the Session Management Cheat Sheet.

Password Managers

Password managers are programs, browser plugins or web services that automate management of large number of different credentials, including memorizing and filling-in, generating random passwords on different sites etc. The web application can help password managers by:

- using standard HTML forms for username and password input,
- not disabling copy and paste on HTML form fields,
- allowing very long passwords,
- not using multi-stage login schemes (username on first screen, then password),
- not using highly scripted (JavaScript) authentication schemes.

Authors and Primary Editors

Eoin Keary eoinkeary[at]owasp.org