# Authentication

⋮ 04/08/2022

Aug 3, 2022

13 min read
Auth Web-Notes

In this section, we'll look at some of the most common authentication mechanisms used by websites and discuss potential vulnerabilities in them.

We'll highlight both inherent vulnerabilities in different authentication mechanisms, as well as some typical vulnerabilities that are introduced by their improper implementation.

## References

Authentication - OWASP Cheat Sheet Series

GitHub - ngalongc/bug-bounty-reference

GitHub - alexbieber/Bug_Bounty_writeups

Bug-bounty/bugbounty_checklist

HowToHunt/Authentication_Bypass

Login Bypass

Mind-Maps/2FA Bypass Techniques

Web Application Penetration Testing Notes

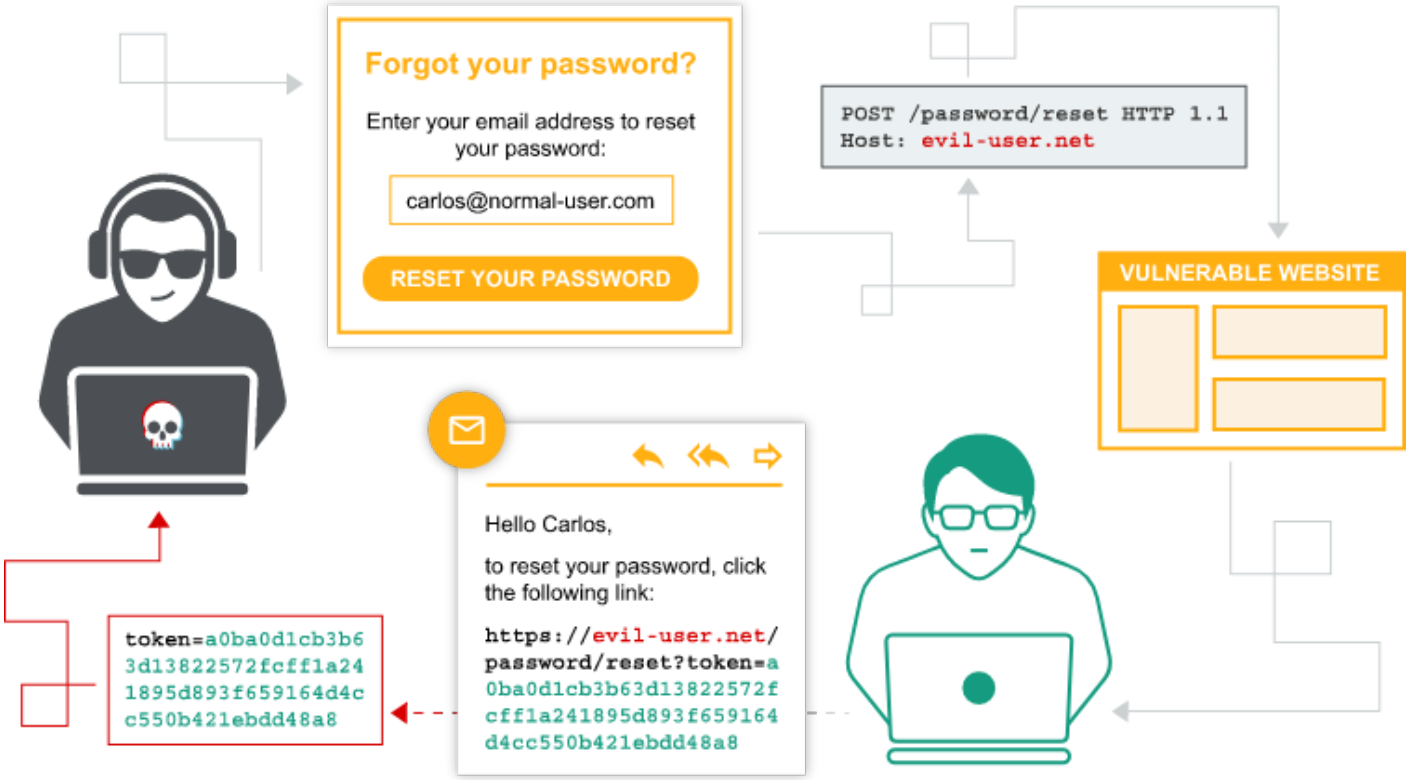
Writeups Bug Bounty hackerone

Authentication Bypass - Pastebin.com

Authentication / Authorization

Bug-bounty/bugbounty_checklist

Dashboard




## What is authentication?

Authentication is the process of **verifying the identity** of a given user or client.

- Something you **know**, such as a password or the answer to a security question. These are sometimes referred to as "**knowledge factors**".
- Something you **have**, that is, a physical object like a mobile phone or security token. These are sometimes referred to as "**possession factors**".
- Something you **are** or do, for example, your biometrics or patterns of behavior. These are sometimes referred to as "**inherence factors**".

Authentication is the process of verifying that a user really **is who they claim to be**, whereas authorization involves verifying whether a user **is allowed to do something.**

> If the User Ahmed want to access to a website he will if he the Owner of this account, Once he entered he will authorize to specific functions only depend on his role.

## How do authentication vulnerabilities arise?

- The **authentication mechanisms are weak because** they fail to adequately protect against brute-force attacks.
- Logic flaws or poor coding in the implementation allow the authentication mechanisms to be bypassed entirely by an attacker. This is sometimes referred to as "broken authentication".

## What is the impact of vulnerable authentication?

- Access to all data functionalities.

- If he compromised a High-Privillage Account ⇒ He Could delete Users and do Even more worse.. etc

# Vulnerabilities in authentication mechanisms

We will look more closely at some of the most common vulnerabilities in the following areas:

Note that several of the labs require you to enumerate usernames and brute-force passwords. To help you with this process, we've provided a shortlist of candidate usernames and passwords that you should use to solve the labs.

---

# Vulnerabilities in password-based login

For websites that adopt a password-based login process, users either register for an account themselves or they are assigned an account by an administrator.

This account is associated with a **unique username and a secret password**, which the user enters in a login form to authenticate themselves.

## Brute-force attacks

This typically involves enforcing passwords with:

- A minimum number of characters
- A mixture of lower and uppercase letters
- At least one special character

For example, if `mypassword`is not allowed, users may try something like `Mypassword1!`or `Myp4$w0rd`instead.

In cases where the policy requires users to change their passwords on a regular basis For example, `Mypassword1!`becomes `Mypassword1?`or `Mypassword2!`.

## Username enumeration

Username enumeration typically occurs either on the login page, for example, when you enter a valid username but an incorrect password, or on registration forms when you enter a username that is **already taken**.

This greatly reduces the time and effort required to brute-force a login because the attacker is able to quickly **generate a shortlist of valid usernames.**

While attempting to brute-force a login page, you should pay particular attention to any differences in:

- **Status codes**: During a brute-force attack, the returned HTTP status code is likely to be the same for the vast majority of guesses because most of them will be wrong. If a guess returns a different status code, this is a strong indication that the username was correct. It is best practice for

websites to always return the same status code regardless of the outcome, but this practice is not always followed.
- **Error messages**: Sometimes the returned error message is different depending on whether both the username AND password are incorrect or only the password was incorrect. It is best practice for websites to use identical, generic messages in both cases, but small typing errors sometimes creep in. Just one character out of place makes the two messages distinct, even in cases where the character is not visible on the rendered page.
- **Response times**: If most of the requests were handled with a similar response time, any that deviate from this suggest that something different was happening behind the scenes. This is another indication that the guessed username might be correct. For example, a website might only check whether the password is correct if the username is valid. This extra step might cause a slight increase in the response time. This may be subtle, but an attacker can make this delay more obvious by entering an excessively long password that the website takes noticeably longer to handle.

Lab: Username enumeration via different responses Web Security Academy

Lab: Username enumeration via subtly different responses Web Security Academy

Lab: Username enumeration via response timing Web Security Academy

## Flawed brute-force protection

The two most common ways of preventing brute-force attacks are:

- Locking the account that the remote user is trying to access if they make too many failed login attempts
- Blocking the remote user's IP address if they make too many login attempts in quick succession

In some implementations, the counter for the number of failed attempts resets if the IP owner logs in successfully.

This means an attacker would simply have to log in to their own account every few attempts to prevent this limit from ever being reached.

Lab: Broken brute-force protection, IP block Web Security Academy

## Account locking

Just as with normal login errors, responses from the server indicating that an account is locked can also help an attacker to enumerate usernames.

Lab: Username enumeration via account lock Web Security Academy

For example, the following method can be used to work around this kind of protection:

1. Establish a list of candidate usernames that are likely to be valid. This could be through username enumeration or simply based on a list of common usernames.

2. Decide on a very small shortlist of passwords that you think at least one user is likely to have. Crucially, the number of passwords you select must not exceed the number of login attempts allowed. For example, if you have worked out that limit is 3 attempts, you need to pick a maximum of 3 password guesses.
3. Using a tool such as Burp Intruder, try each of the selected passwords with each of the candidate usernames. This way, you can attempt to brute-force every account without triggering the account lock. You only need a single user to use one of the three passwords in order to compromise an account.

## User rate limiting

Typically, the IP can only be unblocked in one of the following ways:

- Automatically after a certain period of time has elapsed
- Manually by an administrator
- Manually by the user after successfully completing a CAPTCHA

As the limit is based on the rate of HTTP requests sent from the user's IP address, it is sometimes also possible to bypass this defense if you can work out how to guess multiple passwords with a single request.

Lab: Broken brute-force protection, multiple credentials per request Web Security Academy

### HTTP basic authentication

Although fairly old, its relative simplicity and ease of implementation means you might sometimes see HTTP basic authentication being used. In HTTP basic authentication, the client receives an authentication token from the server, which is constructed by concatenating the username and password, and encoding it in Base64. This token is stored and managed by the browser, which automatically adds it to the `Authorization` header of every subsequent request as follows:

```
Authorization: Basic base64(username:password)
```

Firstly, it involves repeatedly sending the user's login credentials with every request. Unless the website also implements HSTS, user credentials are open to being captured in a man-in-the-middle attack.

In addition, implementations of HTTP basic authentication often don't support brute-force protection. As the token consists exclusively of static values, this can leave it vulnerable to being brute-forced.

---

## Vulnerabilities in multi-factor authentication

Verifying biometric factors is impractical for most websites. However, it is increasingly common to see both mandatory and optional two-factor authentication (2FA) based on **something you know** and **something you have**. This usually requires users to enter both a **traditional password** and a **temporary verification code** from an out-of-band physical device in their possession.

## Two-factor authentication tokens

Many high-security websites now provide users with a **dedicated device** for this purpose, such as the RSA token or keypad device that you might use to access your online banking or work laptop.

It is also common for websites to use a dedicated mobile app, such as **Google Authenticator**, for the same reason.

Some other websites send this code via a Message, this creates a potential for the code to be intercepted. If the attacker have the victim's SIM or mobile phone he would recieve the sms.

## Bypassing two-factor authentication

In some cases the website ask the use to enter the password first and then the verification code, If we have a way to skip the second step we could bypass this verification step.

Lab: 2FA simple bypass Web Security Academy

## Flawed two-factor verification logic

For example, the user logs in with their normal credentials in the first step as follows:

```
POST /login-steps/first HTTP/1.1
Host: vulnerable-website.com
...
username=carlos&password=qwerty
```

They are then assigned a cookie that relates to their account, before being taken to the second step of the login process:

```
HTTP/1.1 200 OK
Set-Cookie: account=carlos

GET /login-steps/second HTTP/1.1
Cookie: account=carlos
```

When submitting the verification code, the request uses this cookie to determine which account the user is trying to access:

```
POST /login-steps/second HTTP/1.1
Host: vulnerable-website.com
Cookie: account=carlos
...
verification-code=123456
```

In this case, an attacker could log in using their own credentials but then change the value of the `account` cookie to any arbitrary username when submitting the verification code.

```
POST /login-steps/second HTTP/1.1
Host: vulnerable-website.com
Cookie: account=victim-user
...
verification-code=123456
```

Lab: 2FA broken logic Web Security Academy

## Brute-forcing 2FA verification codes

Some websites attempt to prevent this by automatically logging a user out if they enter a certain number of incorrect verification codes.

This is ineffective in practice because an advanced attacker can even automate this multi-step process by creating macros for Burp Intruder. The Turbo Intruder extension can also be used for this purpose.

Lab: 2FA bypass using a brute-force attack Web Security Academy

---

## Vulnerabilities in other authentication mechanisms

### Keeping users logged in

A common feature is the option to stay logged in even after closing a browser session. This is usually a simple checkbox labeled something like “Remember me” or “Keep me logged in”.

This functionality is often implemented by generating a “remember me” token of some kind, which is then stored in a persistent cookie

some websites generates a predictable cookie such as `username+timestamp` Once they work out the formula, they can try to brute-force other users’ cookies to gain access to their accounts.

Even using proper encryption with a one-way hash function is not completely bulletproof. If the attacker is able to easily identify the hashing algorithm, and no salt is used, they can potentially brute-force the cookie by simply hashing their wordlists. This method can be used to bypass login attempt limits if a similar limit isn’t applied to cookie guesses.

Lab: Brute-forcing a stay-logged-in cookie Web Security Academy

Even if the attacker is not able to create their own account, they may still be able to exploit this vulnerability. Using the usual techniques, such as XSS an attacker could steal another user’s “remember me” cookie and deduce how the cookie is constructed from that. If the website was built using an open-source framework, the key details of the cookie construction may even be publicly documented.

Hashed versions of well-known password lists are available online, so if the user's password appears in one of these lists, decrypting the hash can occasionally be as trivial as just pasting the hash into a search engine. This demonstrates the importance of salt in effective encryption.

Lab: Offline password cracking Web Security Academy

## Resetting user passwords

As the usual password-based authentication is obviously impossible in this scenario, websites have to rely on alternative methods to make sure that the real user is resetting their own password.

**Sending passwords by email**

Email is also generally not considered secure given that inboxes are both persistent and not really designed for secure storage of confidential information.

Many users also automatically sync their inbox between multiple devices across insecure channels.

## Resetting passwords using a URL

Less secure implementations of this method use a URL with an easily guessable parameter to identify which account is being reset, for example:

```
http://vulnerable-website.com/reset-password?user=victim-user
```

A better implementation of this process is to generate a high-entropy, hard-to-guess token and create the reset URL based on that. In the best case scenario, this URL should provide no hints about which user's password is being reset.

```
http://vulnerable-website.com/reset-password?
token=a0ba0d1cb3b63d13822572fcff1a241895d893f659164d4cc550b421ebdd48a8
```

When the user visits this URL, the system should check whether this token exists on the back-end and, if so, which user's password it is supposed to reset.

This token should expire after a short period of time and be destroyed immediately after the password has been reset.

However, some websites fail to also validate the token again when the reset form is submitted.

In this case, an attacker could simply visit the reset form from their own account, delete the token, and leverage this page to reset an arbitrary user's password.

Lab: Password reset broken logic Web Security Academy

If the URL in the reset email is generated dynamically, this may also be vulnerable to password reset poisoning. In this case, an attacker can potentially steal another user's token and use it change their password.

Lab: Password reset poisoning via middleware Web Security Academy

## Changing user passwords

Password change functionality can be particularly dangerous if it allows an attacker to access it directly without being logged in as the victim user. For example, if the username is provided in a hidden field, an attacker might be able to edit this value in the request to target arbitrary users. This can potentially be exploited to enumerate usernames and brute-force passwords.

Lab: Password brute-force via password change Web Security Academy

---

## Preventing attacks on your own authentication mechanisms

### Take care with user credentials

Although you may have implemented HTTPS for your login requests, make sure that you enforce this by redirecting any attempted HTTP requests to HTTPS as well.

### Don’t count on users for security

A popular example is the JavaScript library `zxcvbn`, which was developed by Dropbox. By only allowing passwords which are rated highly by the password checker, you can enforce the use of secure passwords more effectively than you can with traditional policies.

### Prevent username enumeration

Regardless of whether an attempted username is valid, it is important to use identical, generic error messages, and make sure they really are identical.

You should always return the same HTTP status code with each login request and, finally, make the response times in different scenarios as indistinguishable as possible.

### Implement robust brute-force protection

One of the more effective methods is to implement strict, IP-based user rate limiting. This should involve measures to prevent attackers from manipulating their apparent IP address.

Ideally, you should require the user to complete a CAPTCHA test with every login attempt after a certain limit is reached.

### Triple-check your verification logic

Auditing any verification or validation logic thoroughly to eliminate flaws is absolutely key to robust authentication. A check that can be bypassed is, ultimately, not much better than no check at all.

### Don’t forget supplementary functionality

Remember that a password reset or change is just as valid an attack surface as the main login mechanism and, consequently, must be equally as robust.

## Implement proper multi-factor authentication

**SMS-based 2FA** is technically verifying two factors (something you know and something you have). However, the potential for abuse through SIM swapping, for example, means that this system can be unreliable.

Ideally, **2FA** should be implemented using a **dedicated device or app** that generates the verification code directly. As they are purpose-built to provide security, these are typically more secure.