Broken authentication: A complete guide to exploiting advanced authentication vulnerabilities

BY BLACKBIRD-EU · NOVEMBER 30, 2024 · LAST UPDATED ON MAY 11, 2025

Broken authentication vulnerabilities are fun to find as they are impactful by nature and often grant unauthorized users access to various resources with elevated privileges. Even though they are harder to spot, placed just at the 7th position on the OWASP Top 10 list, they still form a significant risk and are of course worth testing for.

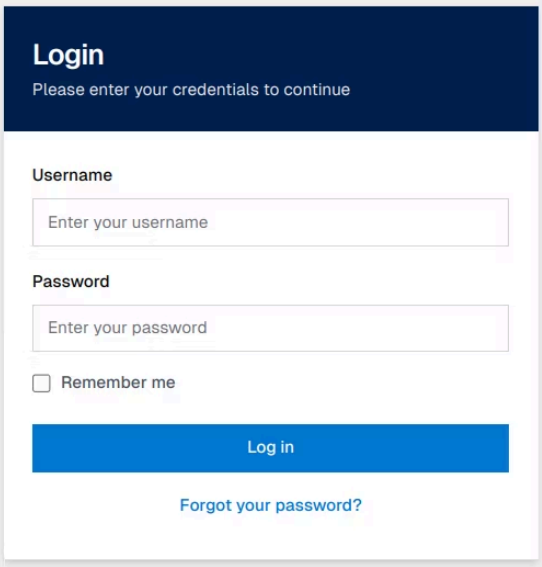
In this article, we will be covering what authentication vulnerabilities are and also help you identify and exploit simple as well as more advanced cases. Please note that this article will not cover security misconfigurations or vulnerabilities found in third-party authentication mechanisms such as OAuth 2.0 or SAML. These will be covered in future articles.

Let's dive in!

What are broken authentication vulnerabilities?

Authentication is the act of validating the user's credentials before granting him/her access to a certain resource. It's the forefront defense layer that is necessary to control access to critical resources and applications that are only intended for a select few authorized users (for example, applications intended for development purposes).

If authentication was not present, it would leave the critical resources (often with elevated privileges) exposed to anyone on the internet, including unauthenticated and unauthorized users.



Example of an authentication panel

A simple authentication process often resolves around the validation of a supplied set of user credentials (email and password). However, various other authentication mechanisms are being applied today to prevent unauthenticated users from accessing controlled resources. And when this authentication method is incorrectly configured, it can open a new attack vector.

Applications vulnerable to broken authentication vulnerabilities fail to validate the user's access and as a result, expose the protected application.

This article will break down the most commonly occurring broken authentication vulnerabilities but before that, let's tackle a common misconception between authentication & authorization first.

The difference between authentication and authorization

Authentication is the process of verifying who someone is—it's about confirming the identity of the user. Authorization, on the other hand, determines what someone is allowed to do—it's about permissions and access rights that the user has.

Both authentication and authorization controls need to be enforced as they both play a huge role in securing an application or system.

Let's take a look at a simple example. At Intigriti, you must first sign in to your account, that way we can determine who you are and return the data that's associated with your account (= authentication). Afterward, we enforce authorization controls to prevent unauthorized bug bounty hunters from accessing private bug bounty programs that they are not invited to (= authorization).

TIP! Want to score some private invites on our platform? Submit some high-quality reports and be active and show off your hacking skills on Intigriti! Bug bounty programs love active hunters and invite them to private programs all the time!

Identifying broken authentication vulnerabilities

Authentication vulnerabilities arise when the authentication mechanism is either weak (such as predictable session tokens) or inadequately developed.

Below, we will be exploring several methods on how to exploit broken authentication vulnerabilities and take advantage of any present logic flaws.

Exploiting basic authentication vulnerabilities

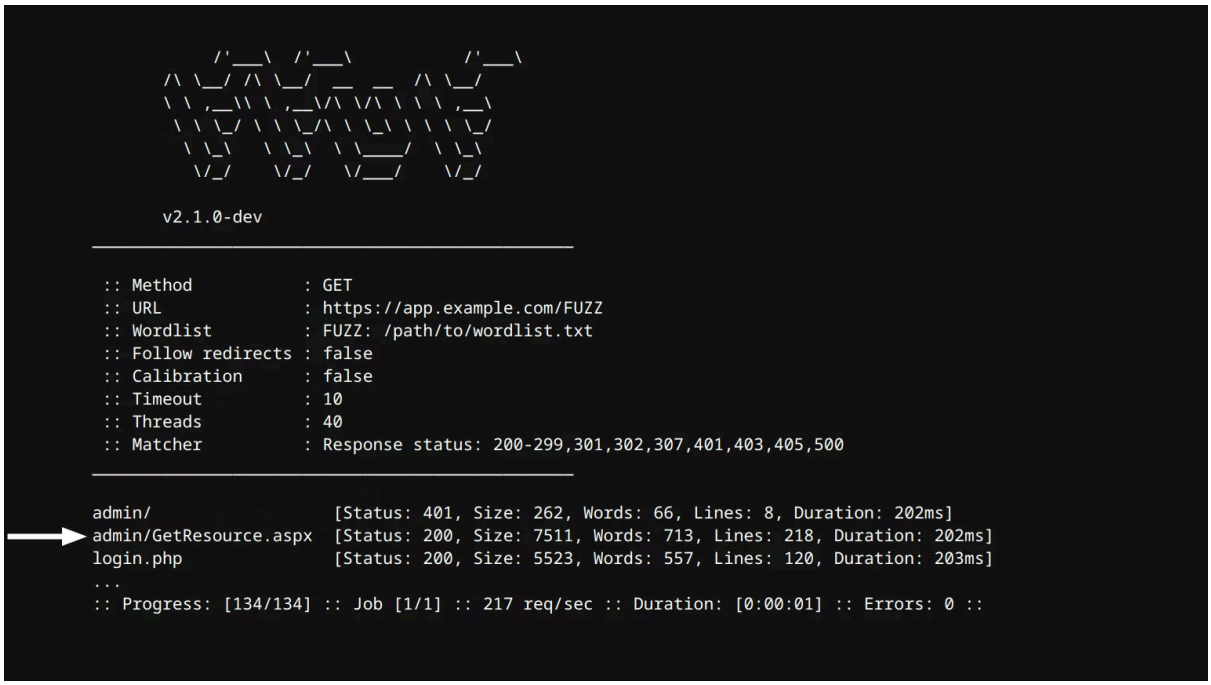
Forced browsing

This exploitation method involves directly requesting the resource behind the authentication wall. In some cases, logic errors or incorrect implementation of authentication mechanisms cause the resources to still be accessible but only by directly requesting them.

This is often due to the lack of authentication verification on the requested resource. A common way to test for this is by performing a bruteforcing attack with a wordlist that is suited for the target type.

The results will allow you to see which resources are accessible without the need for authentication.

Let's take a look at a quick example:



Ffuf content discovery scan

The figure above stems from a content discovery scan made using Ffuf, it revealed that although proper permissions were applied on the **/admin/** directory, these permissions were unfortunately not recursive.

This allowed us to directly request files inside that directory without any authentication, which revealed the presence of the **GetResource.aspx** endpoint!

TIP! When running content discovery scans, try to also change your request method (for example, from `GET` to `POST` or `PUT`)! Some API endpoints or app routes are only programmed to return a valid response when a specific HTTP method is sent in the request!

Default credentials

Some authentication mechanisms are set up to also accept weak or default credentials (such as admin/admin). When developers or web administrators set up a new application or system but have no access to a password manager, they often leave the default credentials or set the login to an easy-to-remember username and password combination. The issue arises when this resource is exposed to the internet and even more when the developer forgets to rotate these credentials.

Here are a few common default credential patterns in the following format `"<username>:<password>"`:

- admin:admin
- administrator:administrator
- <company>:<company>
- <company>:password
- test:test

These are just a few that you could try manually.

TIP! SecLists has a list of over 2.8K documented default credentials for almost all commonly used software vendors! Next time you come across a target that makes use of a third-party software vendor, try to check for default credentials!

Lack of rate limiting (bruteforcing)

Applications that fail to enforce a rate limit on incoming client requests are more susceptible to bruteforcing and credential stuffing attacks. This often allows attackers to bruteforce for weak credentials. Especially if there are response changes that allow you to enumerate existing user accounts.

Various automated tools can help us with bruteforcing logins such as BurpSuite Intruder but also Ffuf! Let's take an example of using Ffuf:

```
ffuf -w "/path/to/wordlist" -X "POST" -d "username=admin&password=FUZZ" -H "Content-Type: application/x-www-form-urlencoded" -u "https://app.example.com/auth/login" -mr "Incorrect password"
```

This command would essentially try out all sorts of weak passwords from your selected wordlist against the admin user. We've also set a match rule to notify us when a potentially valid password is accepted.

Please do note that a wide range of bug bounty programs do not recognize the lack of rate limiting alone to pose a significant risk to the application and often mark it as out of scope as they have other systems and security features put in place. Learn more on aggressive scanning in bug bounty.

Exploiting advanced authentication vulnerabilities

Lack of input validation

Injection vulnerabilities, specifically SQL injections, can cause authentication bypasses. Let's take a look at a simple example of a vulnerable code snippet:

Vulnerable code snippet

As you can see in the figure above, both fields are vulnerable to SQL injection. In this scenario, we can pass a valid username (such as "admin") and break the rest of the SQL query without ever having to provide a password:

```
POST /login.php HTTP/1.1
Host: app.example.com
Content-Type: application/x-www-form-urlencoded; charset=utf-8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/537.3

username=admin&password=xyz' OR 1=1; --
```

Sending the request above would automatically allow us to sign in as the admin user. This is a simple example that demonstrates that lack of input validation can also trigger unwanted behavior that results in partial or complete authentication bypass. Because of this, it is always recommended to test against SQL injections extensively on any authentication endpoint.

Predictable "Remember Me" tokens

Developers often introduce a feature to allow persistent authentication and to keep users logged in when they visit the site at a later time. This token effectively bypasses the entire authentication process, so it is important that this persistent authentication token is unpredictable and stored in a secured place.

A "remember me" token often consists of a unique identifier (such as a username or user ID) and a high-entropy token and is often saved in the client's web browser as a (HTTP Only) cookie.

In some cases, despite all the efforts taken by the developers to generate unique tokens, it is still predictable and can potentially allow anyone to generate his/her token.

It's always recommended to analyze these tokens and see if you can spot any potential flaws that would weaken the token.

Password reset vulnerabilities

Password reset functionalities are always interesting to test as they can result in full account takeovers without requiring any secondary steps from the targeted user account.

Most developers make use of authentication frameworks that come with password reset functionalities out of the box. These frameworks follow best practices and security guidelines and are often well-tested. However, when additional modifications are made or when the password reset functionality is incorrectly implemented, it could result in password reset vulnerabilities.

Predictable reset password tokens

One of them is predictable or weak password reset tokens. An issued reset password token must be a high-entropy token with an expiry date. If the token can be guessed, anyone would be able to generate a token with which you could reset any account's password.

This would effectively bypass the authentication altogether without any secondary required steps from the targeted user.

TIP! Host header injections in password reset implementations are another way to bypass authentication! Make sure to test for these types of vulnerabilities as well!

Trusted IP whitelist bypasses

Some authentication mechanisms revolve around validating the client's public IP address. Servers and applications that make use of this authentication method allow traffic to pass only if the IP matches a whitelist. If the IP can't be matched, the application will either reject all of your requests entirely or ask for other credentials to verify your access.

In certain scenarios, a whitelist for an IP range is defined. This is often done for, example, to allow developers to easily access a resource. This can result in another potential authentication bypass where if the IP whitelist is not strictly defined, you could spoof your IP with a VPN or proxy to match the allowlist and grant you access to the resource.

Another way to bypass IP whitelist is to understand how the client's IP address is derived, and if there's a reverse proxy server between you (the client) and the origin server. If that is the case, it might be possible

that you can spoof your IP by passing the `X-Forwarded-For` request header for example.

Take a look at the following Nginx server configuration file:

```
server {
    listen 80;
    server_name dev-int.stg.example.com;
    root /var/www/dev-int.stg.example.com;

    location /admin {
        if ($http_x_forwarded_for != "127.0.0.1") {
            return 403;
        }

        # Allow access if the incoming request comes from a trusted source
        alias /var/www/html/dev-int.stg.example.com/admin/;
        index index.php;

        autoindex on;
    }

    # Regular site configuration
    location / {
        try_files $uri $uri/ =404;
    }
}
```

Server.conf

From this, we can derive that if the `X-Forwarded-For` HTTP request header matches `127.0.0.1`, it is marked as trusted and it will forward the request. Otherwise, it will reject and return a 403 response status code.

Lack of expiration on magic links

Magic links are used more and more as an authentication mechanism. It's a passwordless authentication process where the user enters his email associated with the account and receives a unique sign-in link that can be used to log in.

The issue arises when the token is predictable or when it never expires. If the token used in the unique sign-in link is weak, it can also be predicted and allow anyone to generate his/her magic links for any user account. Making it possible to completely bypass the authentication layer.

If the magic link never expires, and one of the links is accidentally leaked (via host header injection or indexed by a search engine), it can allow unauthorized users to use the same link later in the future to gain access.

Make sure to test targets that have deployed passwordless authentication implementations.

Conclusion

Authentication vulnerabilities are harder to spot but often carry a significant impact allowing attackers to get access to unauthorized services or components. It's always a good idea to test your targets for broken

authentication vulnerabilities, especially when your target has deployed its unique authentication implementation as it is more prone to design flaws that could be turned into security vulnerabilities.

You've just learned something new about broken authentication vulnerabilities... Right now, it's time to put your skills to the test! Browse through our 70+ public bug bounty programs on Intigriti, and who knows, maybe your next bounty is earned with us!

START HACKING ON INTIGRITI TODAY

REQUEST A DEMO	VISIT THE WEBSITE	GET IN TOUCH
intigriti.com/demo	intigriti.com	hello@intigriti.com