Broken authentication: 7 Advanced ways of bypassing insecure 2-FA implementations

BY BLACKBIRD-EU · DECEMBER 7, 2024 · LAST UPDATED ON MARCH 6, 2025

Two-factor authentication (2FA) has become the go-to solution for strengthening account security. More and more companies are deploying 2FA implementations, and some even enforce them on their users to keep them secure against unauthorized access. But what if 2FA wasn't correctly implemented? In this article, we are exploring 7 ways of bypassing 2FA implementations, including some advanced exploitation methods!

Before we start, if you're interested in learning more about bypassing authentication vulnerabilities, make sure to read our latest article, Broken authentication: A complete guide to exploiting advanced authentication vulnerabilities!

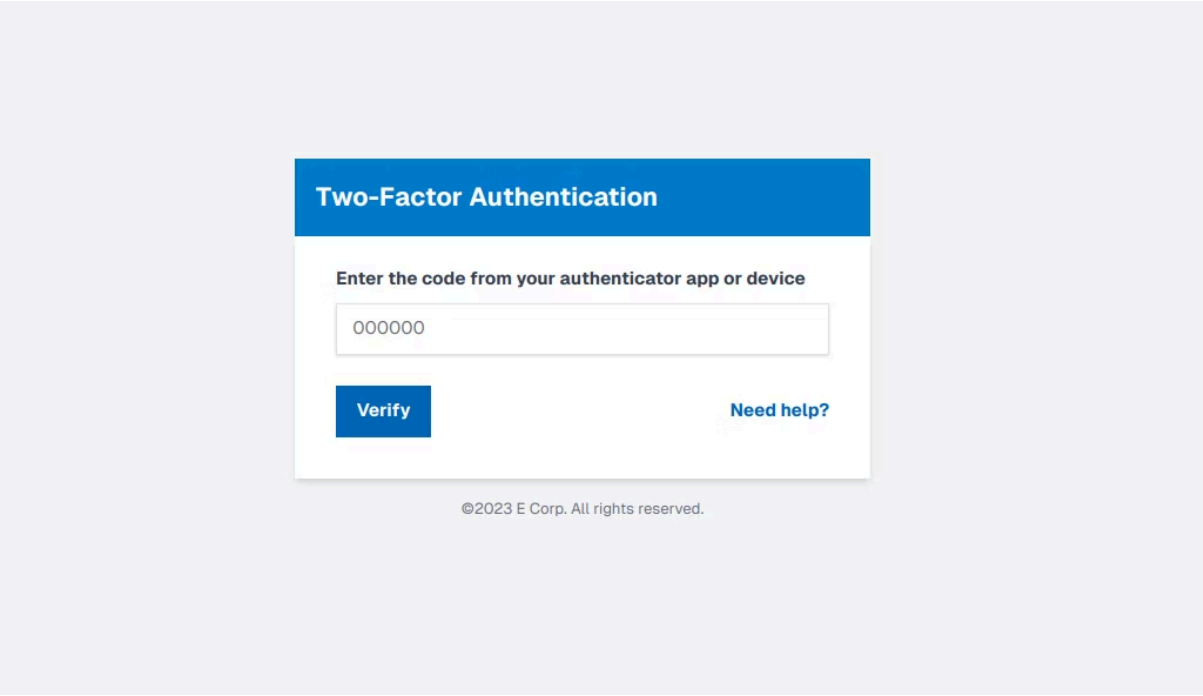
Let's dive in!

What is 2-factor authentication (2FA)?

2FA (Two-Factor Authentication, also referred to as multi-factor authentication or MFA) is a security method that requires users to provide two or more different types of proof to verify their identity when logging into an account.

Traditionally, websites use a combination of email and password to authenticate the user and verify their access. However, weak passwords or just insecure authentication implementations in general often make this a less secure approach to the overall security of the application.

2FA is an additional authentication method that is based on a temporary credential that the user must have and provide, and can't just remember. This approach is considerably much more secure as an additional layer of security prevents bad actors from gaining unauthorized access to someone's user account.



Example of a 2FA implementation

What are 2FA vulnerabilities?

Unfortunately, not all 2FAs are correctly implemented according to security standards and best practices. Some implementations can be entirely bypassed, making it again prone to authentication vulnerabilities. Mainly because companies roll out their own 2FA implementation, while others like to rely on third-party services. And when additional modifications are made or when security best practices are neglected, all sorts of 2FA vulnerabilities can arise that could allow anyone to bypass any mandatory multi-factor authentication.

Identifying 2FA vulnerabilities

2FA vulnerabilities arise when the 2FA implementation is either weak (such as predictable tokens) or inadequately developed (logic errors). There are also other indirect ways we can use to bypass 2FA (for example, with CSRF or IDOR vulnerabilities).

Below, we will be exploring several methods on how to exploit 2FA vulnerabilities and take advantage of any present logic flaws in their insecure implementations.

Exploiting basic 2FA vulnerabilities

Forced browsing

Some 2FA implementations fail to tie your verification token to your current session state. In practice, this looks something like the following:

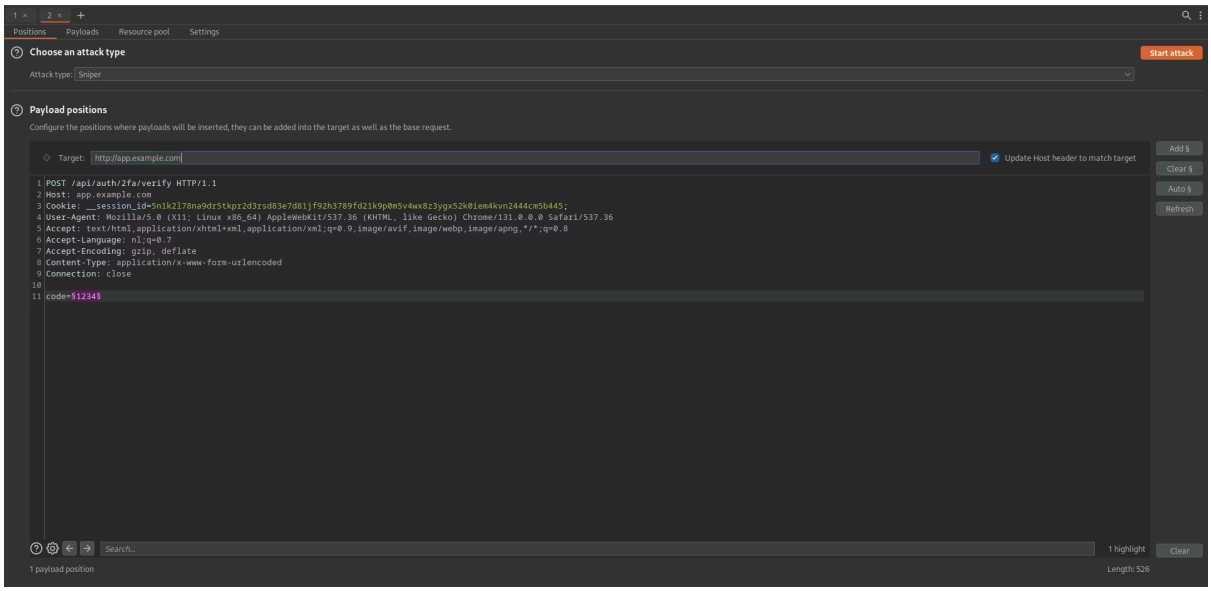
1. The user signs into his account
2. The user's session state is now set to "authenticated"

3. The user is then redirected to the multi-factor authentication page where he/she is requested to supply the 2FA token.

This implementation approach is flawed because any attacker can effectively just skip the third step and still gain access to the account. The malicious user would just try to request any other page that's only accessible to authenticated users, such as the profile page and he/she would gain access to it without ever having to supply the 2FA token.

Bruteforcing

The lack of rate limiting in combination with predictable and/or short tokens makes any 2FA implementation susceptible to bruteforcing attacks. Some applications make use of 4-character 2FA tokens, have no rate-limiting set in place and allow enough time before they expire. These misconfigurations can help us guess the unique token using automated tools like BurpSuite or ZAProxy to bypass the 2FA entirely!



Example of bruteforcing with Burpsuite

Weak 2FA tokens

Another simple way to bypass 2FA implementations is by examining the token and checking if:

- You can re-use the same old token or provide none at all
- You can re-use any of the backup tokens
- The token is not tied to the session
- The token is anywhere exposed in the HTTP response
- Any tokens for testing and development purposes such as "0000" or "123456" are still accepted in production

Re-usable 2FA tokens

Check if the token that you've used previously is still accepted when you log in the second time. 2FA tokens should be temporary and valid only for a fixed period. The token must also expire after it has been used for the first time.

Backup 2FA tokens are also subject to the same rule. Additionally, it is important to also test for logic errors that would skip 2FA validation when no token is supplied. Try sending a request without a token and even one without a parameter.

2FA not tied to your session

If the 2FA is not tied to your session, it likely means that it has been saved in a database under a collection of accepted tokens. This approach is flawed and allows you as an attacker to use your 2FA token to unlock another account.

The best way to test for this edge case is to set up 2 test accounts. Log in with the first and save the 2FA token, next login with the second token and use the 2FA token that you saved from your first test account.

2FA token exposed in the HTTP response

Some applications are designed to validate the 2FA token on the client side, for that to happen, it will need the valid 2FA token. Examine the HTTP response and check if the token is reflected in an HTTP cookie or hidden input field parameter.

Accepted testing tokens

Developers will often resort to static 2FA tokens during development to avoid having to enter a unique 2FA token each time they sign in. However, if these tokens are also accepted in production environments, they can help you entirely bypass the 2FA implementation.

We recommend trying out common 2FA testing tokens such as "0000", "1111" or even "123456", depending on the format and type of token that the application accepts.

Exploiting advanced 2FA vulnerabilities

We've now covered the most simple bypasses, let's dive into the more advanced ones.

Cross-site request forgery (CSRF)

Cross-site request forgery (or even clickjacking) can also help bypass 2FA implementations by crafting a proof of concept that would disable 2FA. This attack vector does require the end-users action and you must make sure that:

1. CSRF is possible (read our detailed article on exploiting CSRF vulnerabilities for more information)
2. The endpoint responsible for disabling 2FA doesn't require a password or any other unique credential

When these conditions apply it is possible to bypass 2FA.

app.ts

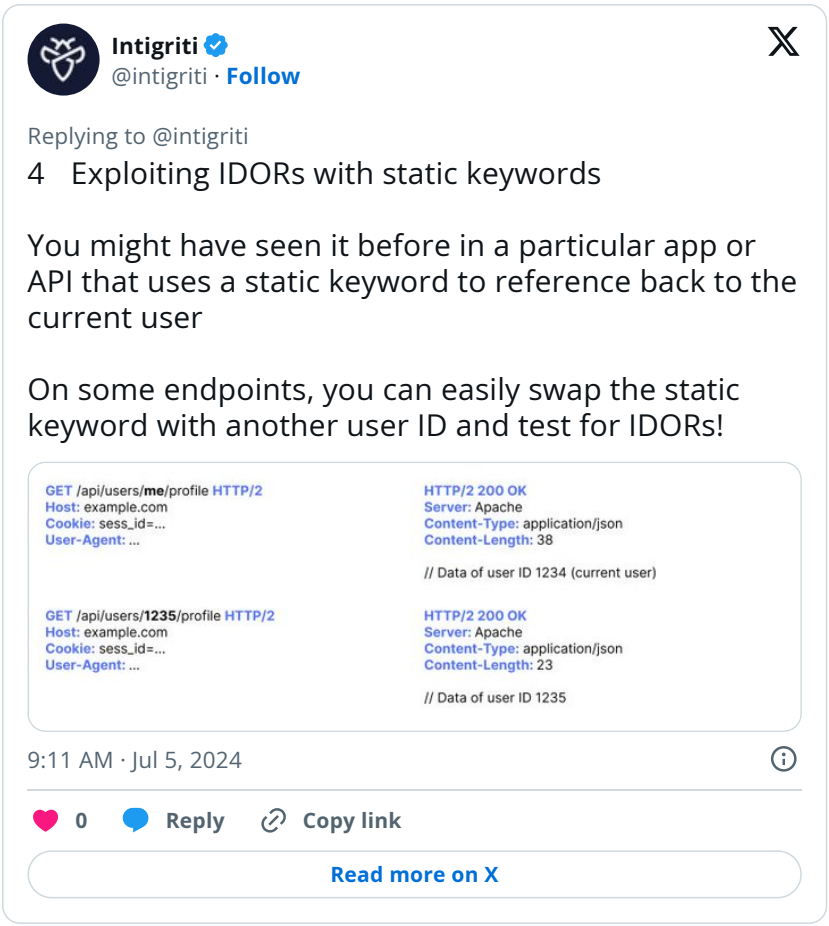
```
const express = require('express');
const cookieparser = require('cookie-parser');

...

const app = express.Router();
app.use(cookieparser());

app.post('/api/account/security/2fa/disable', async (req, res) => {
    try {
        // Assume we have authentication middleware that sets req.session
        const userId = req.session.userId;
        const { token } = req.body;

        if (token) {
            const valid = await Security.verifyToken(token)
            if (!valid) {
                return res.json({ success: false, message: 'Invalid 2FA token!' });
            }
        };

        // Disable 2FA for the user
        const success = await User.update({
            twoFactorEnabled: false,
            twoFactorSecret: null
        }, {
            where: { id: userId }
        });

        return res.json({ success: success, message: '2FA has been disabled' });
    } catch (error) {
        return res.status(500).json({ success: false, error: 'Failed to disable 2FA' });
    };
});
```

Can you spot where the developer made a mistake?

Insecure direct object reference (IDOR)

If the endpoint that is responsible for disabling 2FA is vulnerable to IDOR, we can effectively also disable the victim's 2FA and bypass this security implementation entirely as long as the endpoint doesn't require the previous 2FA token.

TIP! Sometimes, indicators of IDOR vulnerabilities are well hidden inside HTTP requests! Whenever you see a static keyword that refers to an account (such as "current"), try to change it to your user ID and re-send the request! More on this:



Password reset

Password reset functionalities are there to help users who accidentally have forgotten their password and can't get access to their account again. However, in some cases, when the user requests a new password, this functionality also automatically disables 2FA for that particular account. This can also be done even before a new password has been set.

Just initiating a new password reset action would disable 2FA, allowing us to bypass multi-factor authentication.

Second-order 2FA bypass via path traversal

Second-order 2FA bypass via path traversals are quite rare cases of 2FA bypasses in complex applications that occur due to insufficient input validation. The application maintains a flawed validation approach that allows us to bypass 2FA for an account.

Let's take a look at a quick example of a vulnerable code snippet to better understand this case even more!

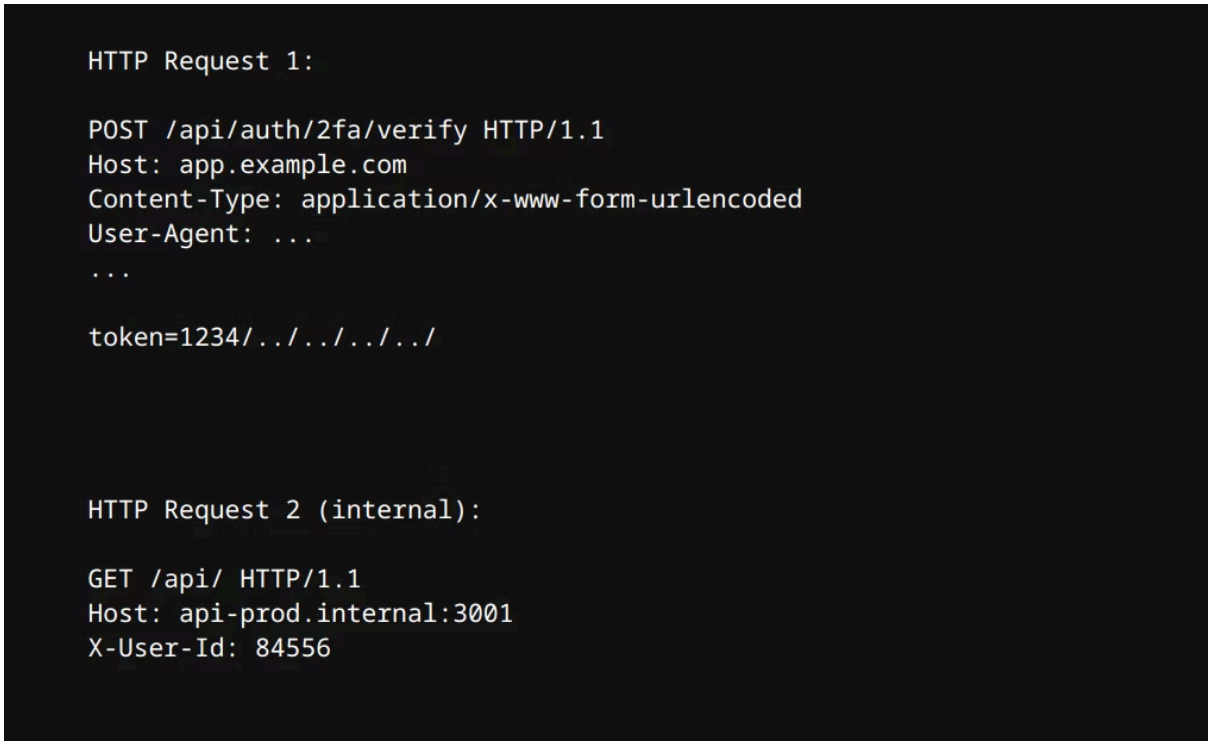
Example of a vulnerable code snippet

As you can see in the figure above, the application consists of 2 services, the backend API and an internal validation API. The backend API forwards our 2FA token without performing any type of validation on our input. Furthermore, the API only checks if the returned status code of the internal validation API is equal to 200 OK.

This allows us to send a 2FA with a path traversal payload to successfully bypass the multi-factor authentication:

```
1234/../../../../
```

In practice, this would look like the following:



HTTP request diagram

It's always recommended to include all sorts of payloads in your testing as you never know how the underlying code is using or transforming your input.

Conclusion

The impact of a 2FA bypass should never be neglected, and we always recommend testing for them, especially on targets with critical assets or applications that store sensitive information.

You've just learned something new about bypassing 2FA implementations... 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