Case 1:

OTP Bypass via Deep URL
Lab: 2FA Simple Bypa

We have both my credentials and my friend’s credentials.

1. First, I log in using my own credentials (username and password). After that, the system asks for an OTP. I enter the correct OTP and successfully log in.

Before login, the URL is standard.

After successful login, the URL includes my username (e.g., /dashboard/myusername).

2. Now, I try to log in using my friend's credentials (username and password). The system prompts for an OTP—but I do not have the OTP for my friend’s account.

3. To bypass this, I take advantage of the deep URL:

I stay logged in with my account (which has already passed OTP verification).

I manually modify the URL, replacing my username in the URL with my friend's username (e.g., /dashboard/friendsusername).

I navigate to that modified URL.

4. Since I'm already authenticated with my own credentials, the system does not revalidate the OTP for the new user and grants access—effectively bypassing OTP verification.

Case 2:

OTP Brute-Force (0000 to 9999)
2FA Broken Logic (Lab Scenario)

You have your own credentials and your friend's username.

1.First, log in using your own username and password. After logging in, you'll be asked to enter an OTP.

2.In the OTP page URL, you can see your username. Try changing it to your friend's username and observe if it still works.

3.If that doesn't work, try again by logging in with your credentials. This time, capture the request using Burp Suite after logging in.

4.In the captured request body, look for the parameter verify= and replace its value with your friend's username.

5.Forward the request. It will now ask for the OTP.

6.Enter 0000 and capture this OTP verification request again. Send it to Repeater.

7.In the Repeater tab, again change the verify= value from your username to your friend's username.

8.Highlight the OTP field and select it as the target position for brute-force.

9.Go to the Payloads tab:

Payload type: Numbers

From: 0000 to 9999

Step: 1

Number format settings:

Base: Decimal

Minimum integer digits: 4

Maximum integer digits: 4

Minimum fraction digits: 0

Maximum fraction digits: 0

10.Start the attack. Based on response length or status code, identify the correct OTP.

Solution to Prevent This Vulnerability:

The application should lock the account after 3 to 4 failed OTP attempts to prevent brute-force attacks.

OTP and 2FA Security Considerations

1. OTP Length

OTP (One-Time Password) should be exactly 6 digits, ranging from 000000 to 999999.

2. Predictability

OTPs must not be predictable. They should be randomly generated using a secure algorithm to prevent attackers from guessing them.

3. Expiry Time

OTPs should be valid for less than 5 minutes to minimize the attack window.

Each OTP should be single-use only—once used, it should immediately expire.

4. Retry Limit

After 3 failed OTP attempts, the OTP should be locked or invalidated, and the user should be blocked from retrying for a short time or require additional verification.

5. OTP Flooding Protection

Prevent OTP flooding by implementing rate limiting—restrict the number of OTP requests a user can make within a short period (e.g., max 3 requests in 5 minutes).

Lab and Exploit Scenarios

6. 2FA Bypass via Burp Suite

Scenario: Log in with valid credentials, capture the OTP request using Burp Suite.

Go to the Intruder tab, set the OTP field as the target.

Use null payloads (blank OTPs) and generate 100 payloads.

Launch the attack to check for responses that incorrectly validate blank or missing OTPs.

Note: This was tested on platforms like Flipkart by analyzing OTP endpoints.

Common OTP Implementation Flaws

7. Use of Dummy OTPs

Common weak or hardcoded OTPs like 0000, 000000, 1234, 123456, 1111 may sometimes be accepted due to developer oversight.

These should be explicitly blocked during validation.

8. OTP Leakage in Source Code

If the OTP is visible in the HTML source, JavaScript, or network response, it’s a critical vulnerability.

9. Accepting Partial OTPs

If the system accepts partial input of an OTP (e.g., accepting 123 when 123456 was expected), it's vulnerable.

10. OTP Bypass via JSON Array Injection

If the OTP is sent in a JSON payload, test for bypasses using an array of payloads (e.g., sending multiple OTPs in one request).

Password and Authentication Weaknesses

11. Accepting Partial Passwords

If partial strings of the password (e.g., "sha" from "shadow123") allow login, this is a critical flaw.

12. Password in URL

Passwords or OTPs should never be sent via URLs (GET parameters), as they can be logged or cached.

13. Login Without OTP

If a user can log in without entering OTP, the 2FA flow is bypassable.

14. Login Without Password

If login is possible without providing a password, the system is fundamentally insecure.

15. Default Credentials

If default credentials like admin:admin are accepted and not changed on first login, this is a major risk.

Advanced Bypass Techniques

16. OTP Redirect via X-Forwarded-Host

Improper handling of the X-Forwarded-Host header may allow redirection or manipulation of the flow, especially in multi-tenant applications.

17. Response Manipulation Bypass

After submitting a wrong OTP, intercept the response.

If the server returns 400 Bad Request with a failure message like "Failed to verify OTP", try changing the response to 200 OK and the message to "Success".

Forward the request. In some misconfigured systems, this can trick the frontend into granting access.

UI Security Note

Mobile numbers must be masked on the UI (e.g., *****1234) to protect user privacy.