

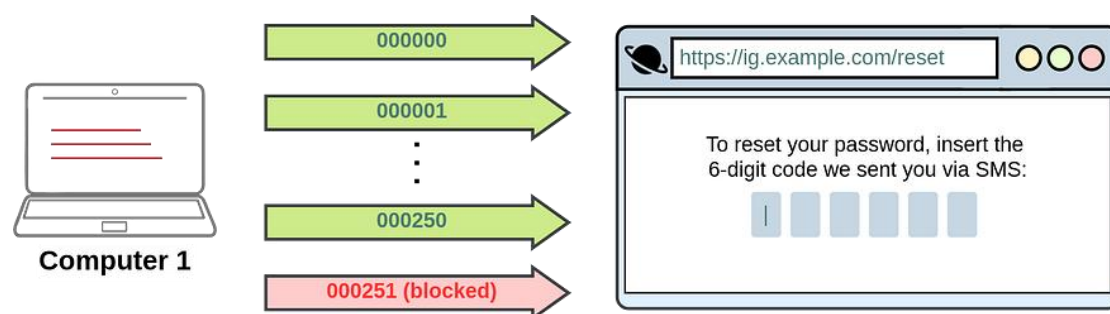
Task 11 : 4. Insecure Design

Insecure Design

Insecure design refers to vulnerabilities which are inherent to the application's architecture. They are not vulnerabilities regarding bad implementations or configurations, but the idea behind the whole application (or a part of it) is flawed from the start. Most of the time, these vulnerabilities occur when an improper threat modelling is made during the planning phases of the application and propagate all the way up to your final app. Some other times, insecure design vulnerabilities may also be introduced by developers while adding some "shortcuts" around the code to make their testing easier. A developer could, for example, disable the OTP validation in the development phases to quickly test the rest of the app without manually inputting a code at each login but forget to re-enable it when sending the application to production.

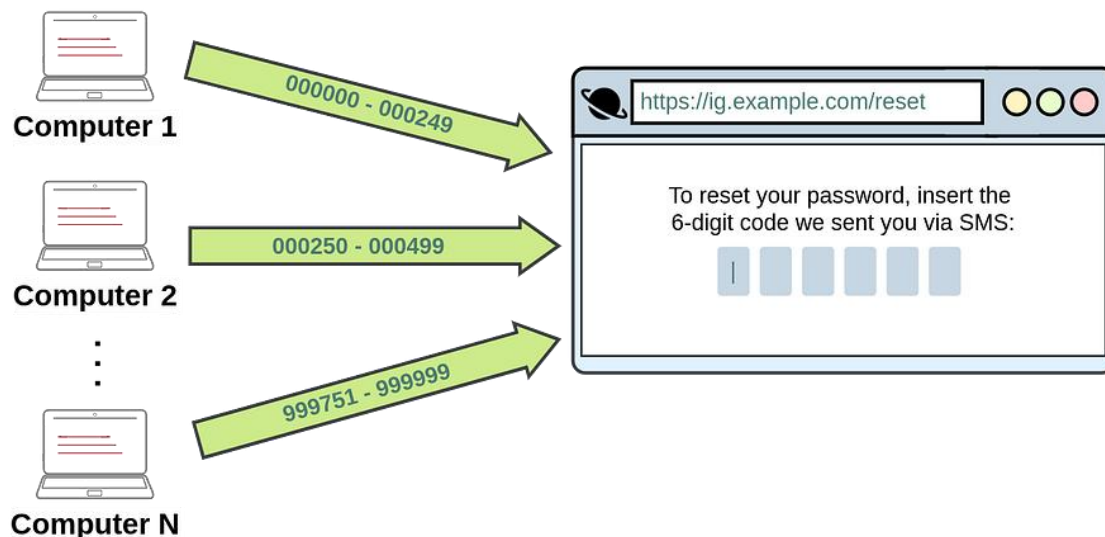
Insecure Password Resets

A good example of such vulnerabilities occurred on Instagram a while ago. Instagram allowed users to reset their forgotten passwords by sending them a 6-digit code to their mobile number via SMS for validation. If an attacker wanted to access a victim's account, he could try to brute-force the 6-digit code. As expected, this was not directly possible as Instagram had rate-limiting implemented so that after 250 attempts, the user would be blocked from trying further.



However, it was found that the rate-limiting only applied to code attempts made from the same IP. If an attacker had several different IP addresses from where to send requests, he could now try 250 codes per IP. For a 6-digit code,

you have a million possible codes, so an attacker would need $1000000/250 = 4000$ IPs to cover all possible codes. This may sound like an insane amount of IPs to have, but cloud services make it easy to get them at a relatively small cost, making this attack feasible.



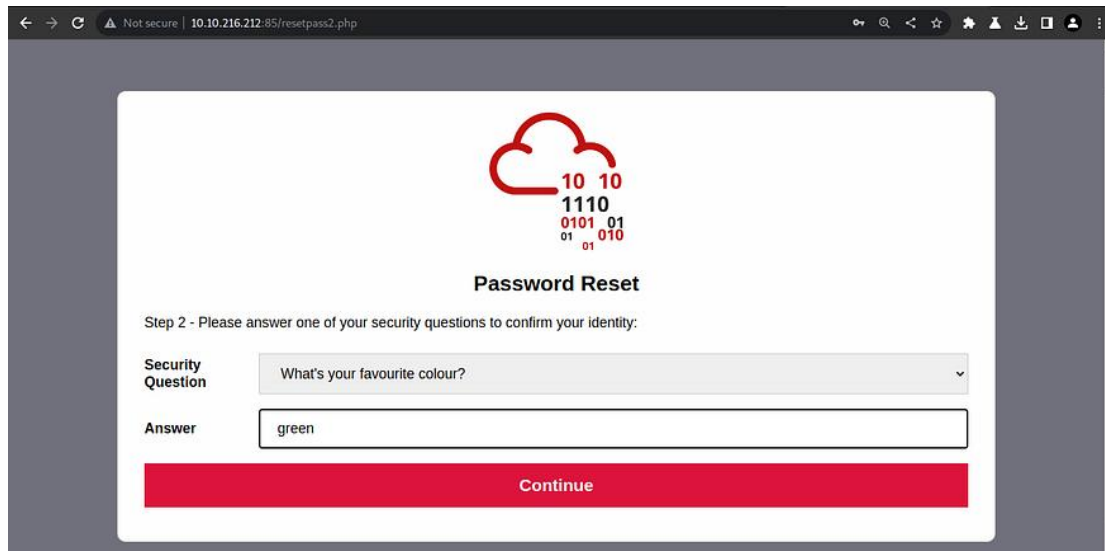
Notice how the vulnerability is related to the idea that no user would be capable of using thousands of IP addresses to make concurrent requests to try and brute-force a numeric code. The problem is in the design rather than the implementation of the application in itself.

Since insecure design vulnerabilities are introduced at such an early stage in the development process, resolving them often requires rebuilding the vulnerable part of the application from the ground up and is usually harder to do than any other simple code-related vulnerability. The best approach to avoid such vulnerabilities is to perform threat modelling at the early stages of the development lifecycle. To get more information on how to implement secure development lifecycles, be sure to check out the SSDLC room.

Practical Example

Navigate to <http://machine-ip:85> and get into joseph's account. This application also has a design flaw in its password reset mechanism. Can you figure out the weakness in the proposed design and how to abuse it?

Now go to password reset page of **joseph** and try to give the answer of Security question. Make a wild guess just like i did and we get to know his favourite colour as **green**.



10 10
1110
0101 01
01 010

Password Reset

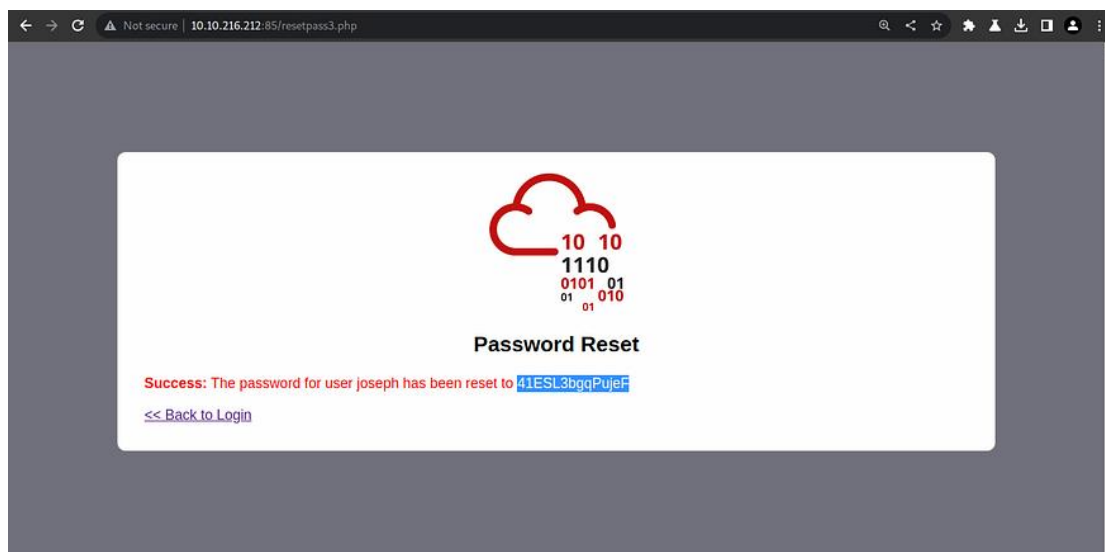
Step 2 - Please answer one of your security questions to confirm your identity:

Security Question: What's your favourite colour?

Answer: green

Continue

And finally we got his password so, try to login with this new password.



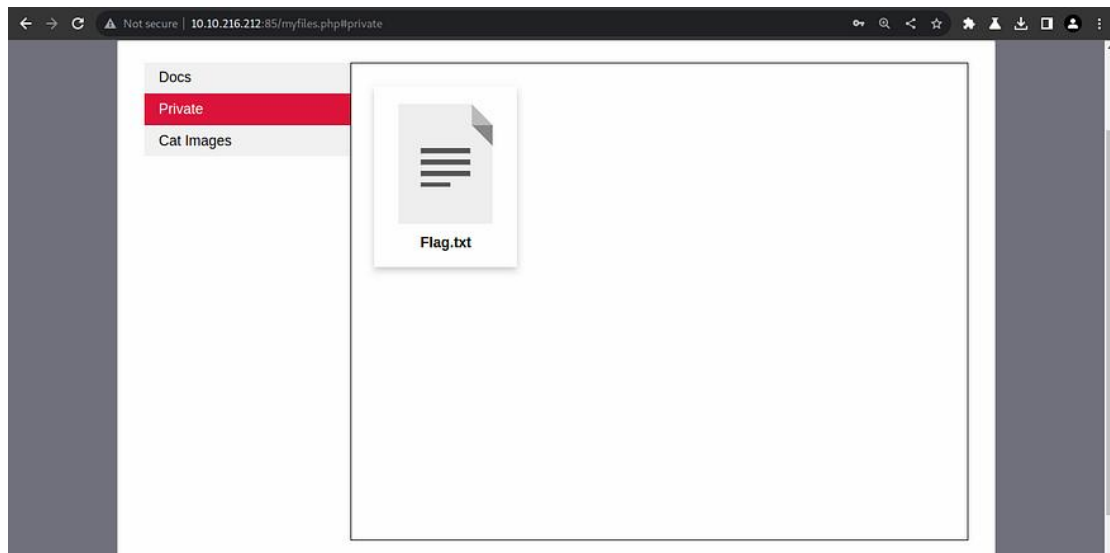
10 10
1110
0101 01
01 010

Password Reset

Success: The password for user joseph has been reset to [41ESL3bggPujef](#)

[<< Back to Login](#)

We got successfully logged in & the only thing which is left is to open the flag.txt file to get the flag.



Answer the questions below :

1. Try to reset joseph's password. Keep in mind the method used by the site to validate if you are indeed joseph.

A. No answer needed

2. What is the value of the flag in joseph's account?

A. THM{Not_3ven_c4tz_c0uld_sav3_U!}