Dear novice hacker,

Welcome to my vulnerable web application 😊

I was able to implement 6 flaws, so let’s start.

First of all, the port is 8090, due a localhost need, I have already stuff running on 8080.

1. A2:2017-Broken Authentication & InsufficientLogging & Monitoring

As said by OWASP top 10, attacker as access to undreed of millions of default password, and guess what, account Customer as one of these!

To crack it:

1. Launch Kali

Download top 10k most used password, let’s assume it’s located in /root/Downloads its name is: passwords.txt

1. Launch a command invite
2. type:

hydra -l Customer -P/root/Downloads/passwords.txt -t 16 [your\_server\_IP] http-post-form “/login:username=^USER^&password=^PASS^&Login:Invalid username and password.” -V -s 8090

[server\_ip] is for me replaced by: 192.168.0.15.

1. Wait, can take a while depending of your cpu, the password is one of the last, tips: you love it to take a break during two work phases.

To solve this, several solutions:

1. Forbid user to take a password in a list of password
2. Activate csrf to make it more complicated to brute force
3. Monitor logging, if you detect a lot of fail, block the account, this may seams it is being attacked.
4. Security Misconfiguration

One of the default log/pass is admin/admin. I also left ted in the place by the way.

To solve it:

Change default setting after installation, take just a few minutes.

1. Broken Access Control

Admin and Users have the same rights!

How to detect it? They have access to the exact same pages. For example, being able to see people password when logged in.

How to fix this:

Add this in MySimpleUrlAuthenticationSuccessHandler>determineTargetUrl, and use proper role, then block access to page when unwanted.

// boolean isUser = false;

// boolean isAdmin = false;

// Collection<? extends GrantedAuthority> authorities

// = authentication.getAuthorities();

// for (GrantedAuthority grantedAuthority : authorities) {

// if (grantedAuthority.getAuthority().equals("ROLE\_USER")) {

// isUser = true;

// break;

// } else if (grantedAuthority.getAuthority().equals("ROLE\_ADMIN")) {

// isAdmin = true;

// break;

// }

// }

// if (isUser) {

// return "/logged";

// } else if (isAdmin) {

// return "/logged";

// } else {

// throw new IllegalStateException();

// }

1. Cross-Site Scripting (XSS)

How to detect it:

On the first page,

Fill each area, Name: Any name

Address: <script>alert('XSS Expoit worked');</script>

Password: any password

Then submit, and look at happen when you come back to this page.

You now have a wonderful 'XSS Expoit worked' pop up. This means that the page is XSS vulnerable.

A malicious pirate can then add whatever he want to steal your cookie. They can even redirect your browser to a malicious server and do a lot of stuff with your data. This kind of attack are very powerful.

How to fix it:

1. In the table code, replace utext by text, this will consider whatever is written as text, and not anything else, but it’s not enough
2. Set up input validation, escape the ‘<’ and ‘>’ and also their equivalent in code, and remove keyword like ‘script’ or any html/php/javascript tag.
3. Sensitive Data Exposure

How to detect it:

1. Launch Owasp Zap.
2. Launch a sniffed browser
3. Go to /login
4. Login using any good combination of USER/PASS
5. Go back to Owasp Zap and check what you see:

username=Customer&password=vacation

1. So anyone can sniff username and password because nothing is crypted.

How to fix it:

Implement https protocol to encrypt every transfer. And activate csrf to make pirate work more complicated.

I tried to implement SQL injection flaw, but I was unable to set up an embedded SQL server to my project, I found several but it takes a lot of time, I’ll try to submit another version of this project later and I do want to add it.

SQL injection is also a very powerful attack, the pirate can really get access to a lot of sensitive data, and once he is inside it’s very hard to spot him.

It’s actually also very easy to block it has we seen before.

Designing this application, I realized something, it’s usually very simple to avoid all these breaches, it’s just a matter of taking time to patch the flow, and close the door.

But we are lazy and want to go faster so we don’t tale take the necessary time to implement all of these. In addition, in huge team, maybe we can forget to speak together, and then create a breach, I think it’s very important to add a pentester in all dev team.

On flaw may be quite tricky, the out of date dependencies, even if you fox it when releasing the application, it will happen later for sure, one has to think about maintaining all application and versioning in order to avoid it to happen. I think this may become one of the worse on internet on the years to come.