Brute-force login and Bypass
Account lockout on elabFTW 1.8.5

samguy, krastanoel[at]gmail[dot]com

2021-07-08
# Contents

1 Introduction  
   1.1 What is elabFTW ................................. 1  
   1.2 Affected version ............................... 1  
   1.3 Mitigation ..................................... 1  

2 Technical analysis  
   2.1 Lockout process ............................... 2  
   2.2 Attack process ................................. 3  

3 Exploitation ..................................... 4  

4 Conclusion ....................................... 11  

5 Appendix A: Account Enumeration ................. 12
# Introduction

This paper will explain how to bypass brute-force protection or account lockout on *elabFTW 1.8.5*. During a penetration test, I came across this specific version and it has an authenticated remote code execution vulnerability so it needs valid accounts and credentials. The protection bypass was found using code review and then leveraged that to automate the attacks using *Burp Suite Intruder*.

## 1.1 What is elabFTW

eLabFTW is a free and open source electronic laboratory notebook for researchers.\(^1\) Once installed on a server, it allows researchers to track their experiments, but also to manage their assets in the lab (antibodies, mouse, siRNAs, proteins, etc.).

## 1.2 Affected version

The attacks have been specifically tested against versions **1.8.5** and it has been fixed in version **4.1.0**.\(^2\)

## 1.3 Mitigation

1. create `readFailedLoginByIp` function on `app/models/Logs.php` to execute query where `user` field is `REMOTE_ADDR` and the `body` is **Failed login attempt**.
2. Invoke `readFailedLoginByIp` function on `login.php` to validates if the count is reach the failed attempt limit and banned.

---

\(^1\)https://github.com/elabftw/elabftw

\(^2\)https://github.com/elabftw/elabftw/releases/tag/4.1.0
2 Technical analysis

This section will explain how the lockout process works by testing the login page while also reviewing the source code and then making an attack process.

2.1 Lockout process

Assuming the administrator email is already known as “administrator@elabftw.local” with a wrong password submitted in the login form will produce a failed login message. See Appendix A to enumerate valid email accounts.

From the flash messages above, failing 3 times will result in being banned for 1 hour. Let’s find out where in the source code these messages are triggered.
From the grep result above there are two files triggering the error messages: LoginController.php and login.html. Upon further inspection in LoginController.php file at line 74 there is an if-else validation for login failed attempt.

```php
if (!$Session->has('failed_attempt')) {
    $Session->set('failed_attempt', 1);
} else {
    $n = $Session->get('failed_attempt');
    $n++;
    $Session->set('failed_attempt', $n);
}
```

The code above will set failed_attempt key with value 1 in $Session variable if it's not exist or, increment the value if it does. Because PHP handles and tracks $Session variable using PHPSESSID in a Cookie request header, which is controlled by user, the bypass is very obvious. Simply using random value in PHPSESSID or, completely removing the Cookie header on each request to login.php will force the application to create a new session and the failed_attempt key will always be set to 1.

When inspecting the login page a hidden input called formkey was found and it’s required along with email and password as a data submitted to LoginController.php

### 2.2 Attack process

This section will assemble what was found when identifying how the lockout process works.

1. Make a GET request to login.php
   - Extract PHPSESSID from the response header
   - Extract formkey from the response body
2. Make a POST request to LoginController.php with PHPSESSID and formkey from step 1 included and, use valid email address and wordlists for password on data field
3. Follow url redirections from step 2 response location header
   - If url redirect location is login.php, automatically remove the Cookie header
   - If url redirect location is not login.php, the attack is succeed
3 Exploitation

The exploitation will use Burp Suite's Intruder tool to automate the attack process. First step is to extract PHPSESSID and formkey from the login.php assuming the request was already made from the browser through Burp Suite Proxy. Navigating to Proxy > HTTP History, right-clicking on the GET request /login.php and select Send to Intruder:

Now navigate to Intruder window and choose Positions tab and remove a Cookie header if it exists:

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines

https://portswigger.net/burp/documentation/desktop/tools/intruder/using
Next, choose Options tab and scroll down to Grep Extract. Tick Extract the following items from responses and set Maximum capture length to 150:

Click Add button then Refetch response and notice Set-Cookie header is being set. Select PHPSESSID value and click OK:

Click Add button again and do the same for formkey value:
Define the location of the item to be extracted. Selecting the item in the response panel will create a suitable configuration automatically. You can also modify the configuration manually to ensure it works effectively.

Select **Always** option on **Redirections**:

- **Redirections**
  - These settings control how Burp handles redirections when performing attacks.
  - **Follow redirections:**
    - **Never**
    - **On-site only**
    - **In-scope only**
    - **Always**
  - **Process cookies in redirectors**

Navigate to **HTTP History** tab on **Proxy** window then select **POST /app/controllers/LoginController.php** and copy the raw request:
Go back to `Positions` tab on `Intruder` window and paste copied raw request in the editor and then click `Add $` button to set a mark on these fields: `PHPSESSID`, `password`, `formkey` and set `Attack type` to `Pitchfork`:

Next, clicking the `Payloads` tab to set 3 payloads. `Payload set 1` for `PHPSESSID` cookie value using “Recursive Grep”:
Brute-force login and Bypass Account lockout on elabFTW 1.8.5

Payload Sets
You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab.

Payload set: 1
Payload count: unknown
Payload type: Recursive grep
Request count: 0

Payload Options [Recursive grep]
This payload type lets you extract each payload from the response to the previous request in the attack. It is useful in some scenarios.

Select the "extract grep" term from which to derive payloads:

From ['PHPSESSIONID'] to {, path=}
From ['formkey' value=''1'/"]In <in...

Payload set 2 for password and set Payload type to "Simple list" then click Load to choose a small wordlist file from /usr/share/wordlists/wfuzz/others/common_pass.txt:

Payload set: 2
Payload count: 52
Payload type: Simple list
Request count: 0

Payload Options [Simple list]
This payload type lets you configure a simple list of strings that are used as payloads.

Paste
Load ...
Remove
Clear

samguy, krstanoe[AT]gmail[DOT]com
Brute-force login and Bypass Account lockout on elabFTW 1.8.5

Payload set 3 for formkey using “Recursive Grep”:

Using Mitmproxy command-line mitmdump as upstream proxy to automatically remove Cookie header when following a redirect location request to /app/controllers/../../login.php.

Navigate to Project options > Connections > Upstream Proxy Servers, toggle Override user options and click Add button. Specify the Destination host to target domain elabftw.local, Proxy host to 127.0.0.1 and Proxy port: 8081 and click OK.

Upstream Proxy Servers

These settings are configured within user options but can be overridden here for this specific project.

Enabled Destination host Proxy host Proxy port Auth type Username

Add

Edit

samguy, krastanoel[at]gmail[dot]com
Go back to *Intruder* window and start the attack by clicking on *Start attack* button:

See image above the *Payload1* values are always changing and these values are taken from *PHPSESSID* columns for the next request. Forcing the application to create a new session so the failed_attempt key will always be set to 1. The lockout process successfully bypassed.

Notice in the highlighted request, the *Length* size is bigger than others and in the *Response 1* tab, the *Location* header is pointed at “.../.../experiments.php” meaning the attack is successful.
4 Conclusion

Brute-force attacks are often used to break through an application or services to reach the intended goal. The goal is usually to extract confidential data or in this case is to gain administrative access to the application control panel to be able to exploit another critical vulnerability to fully take control of the system.

The brute-force protection on elabFTW can be bypassed because the attacker has control over the request header sent to the application. Meanwhile the application itself has to track failed login attempts through the request header and then stored in the session that was created by the default framework function.

Saving data in the sessions are common practices in web developments however, for this specific case it’s better to have more strict implementation. Looking at the source code elabFTW maintainer opt out to block the request by ip and instead, use a combination of remote ip-address concat with browser user-agent and encrypted using md5. This implementation is also obvious to bypass just by changing the browser user-agent because the attacker has control over the browser.

The elabFTW already has what it needs to prevent this attack, again looking at the source code the application is already logging the failed attempt in the database. By adding a function to query and extract the log based on the ipaddress and the failed attempt message, the result can be counted and validated if it’s already reached the attempt limit to be banned.
5 Appendix A: Account Enumeration

Valid email account is needed to perform successful brute-force attacks on this application. Surprisingly the reset password form page response can be used to determine if the email exists using \textit{wfuzz}.

```bash
kali@kali:~$ wfuzz -c -L -u
-> "https://elabftw.local/app/controllers/ResetPasswordController.php" -d
-> "email=FUZZ@elabftw.local&Submit=" -b "PHPSESSID=FUZZ" -w names.txt --hs 'Email not found'
```

```
* Wfuzz 2.4.5 - The Web Fuzzer *
******************************************************************************
Target: https://elabftw.local/app/controllers/ResetPasswordController.php
Total requests: 23

<table>
<thead>
<tr>
<th>ID</th>
<th>Response</th>
<th>Lines</th>
<th>Word</th>
<th>Chars</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000004:</td>
<td>200</td>
<td>126 L</td>
<td>372 W</td>
<td>4447 Ch</td>
<td>&quot;administrator&quot;</td>
</tr>
<tr>
<td>000000013:</td>
<td>200</td>
<td>126 L</td>
<td>372 W</td>
<td>4448 Ch</td>
<td>&quot;adrian&quot;</td>
</tr>
<tr>
<td>000000018:</td>
<td>200</td>
<td>126 L</td>
<td>372 W</td>
<td>4448 Ch</td>
<td>&quot;benjamin&quot;</td>
</tr>
<tr>
<td>000000023:</td>
<td>200</td>
<td>126 L</td>
<td>372 W</td>
<td>4448 Ch</td>
<td>&quot;caitlyn&quot;</td>
</tr>
</tbody>
</table>
```

Total time: 1.766739  
Processed Requests: 23  
Filtered Requests: 19  
Requests/sec.: 13.01833