Elizabeth Labor
Introduction to Computer Security
20/05/2020

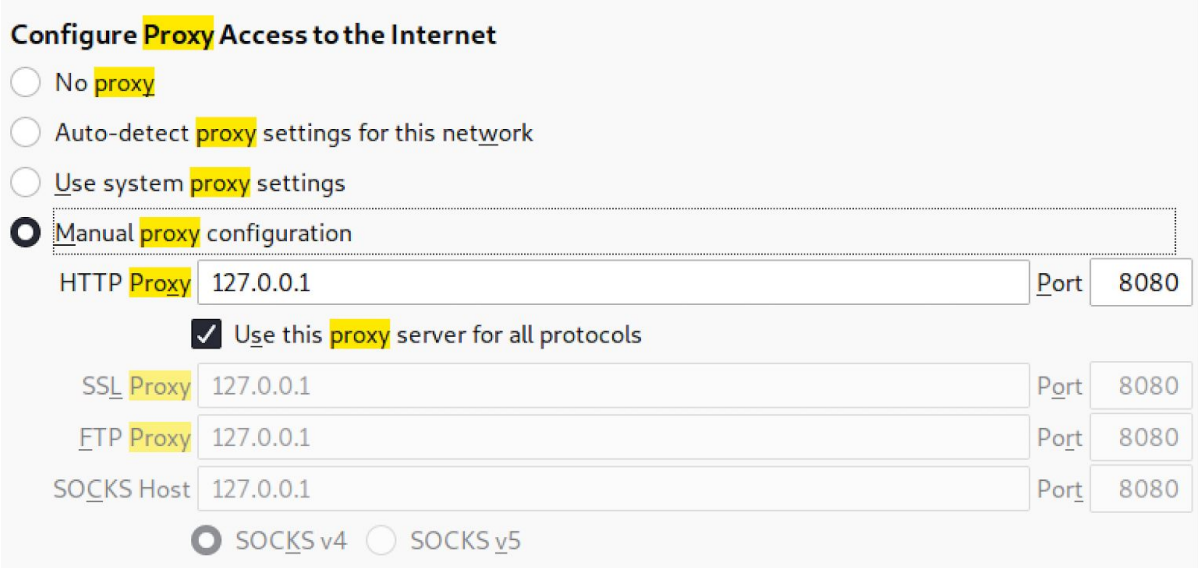
Mock Security

Tools & Setup

This lab takes place on a Kali Linux (2020.1) 64-bit virtual machine in NAT mode (IP address 10.0.2.15). The DVWA is hosted at 127.0.0.1/DVWA/ and attacks are executed locally.

For this lab I use the Burp Suite tool configured with Firefox to intercept, analyze, and modify web traffic to the DVWA. To set up, open Burp Suite with the `burpsuite` command. Go to Target → Scope and add localhost. Under Proxy → Intercept make sure Intercept is turned on. Then click the Options tab to check that the proxy listener is set to 127.0.0.1:8080 and running.

Next, configure Firefox so that all web traffic is routed through the Burp listener for examination. Go to Preferences and search “proxy.” Use the settings shown below.



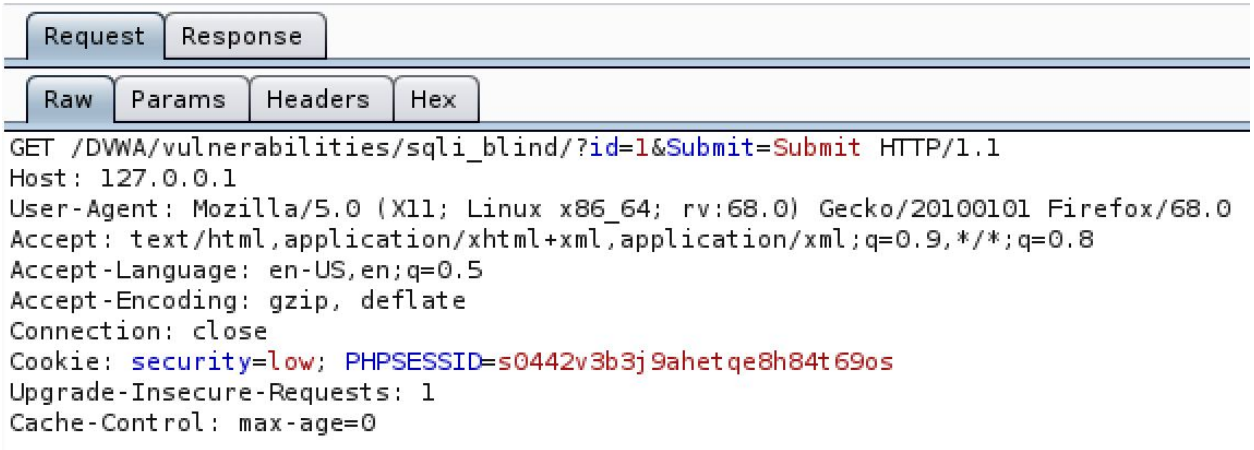
Now add Burp’s certificate to Firefox’s trusted list. In Burp Suite, go to Proxy → Options and click Import / export CA certificate. Export the certificate in DER format and save with the .cer extension. In Firefox, click Preferences → Privacy & Security → View Certificates and import the Burp certificate. Check the “identify websites” box.

I was still getting bad certificate errors with this setup so I also downloaded the FoxyProxy Firefox extension, added 127.0.0.1:8080 as a proxy there, and turned it on. This seemed to fix the issue. Now we are ready to attack!

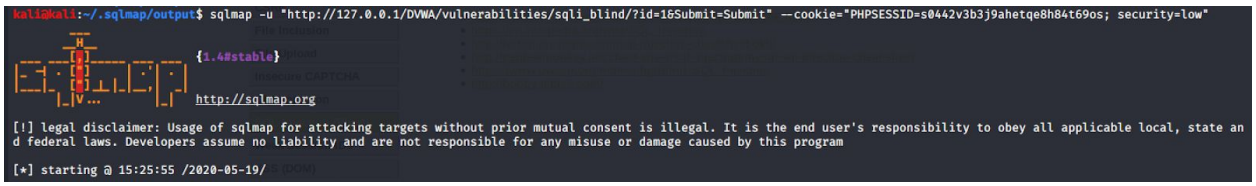
Attack 1: Blind SQL Injection

For the first attack we use a blind SQL injection to steal users' authentication information from the database. The attack is similar to a regular SQL injection except little to no feedback is displayed onscreen, so we have to do more guessing. For this we will use Burp Suite and the SQL injection tool SQLMap.

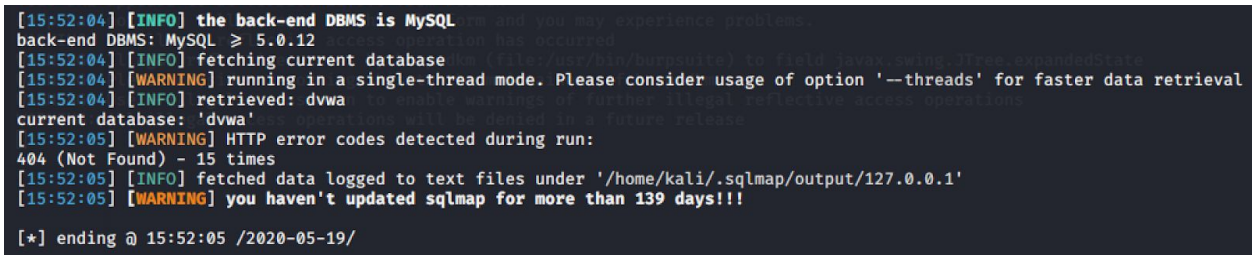
The first step is information gathering. Navigate to the SQL Injection (Blind) tab of DVWA and enter a random number in the box. Observe the web traffic with Burp, where we see an HTTP GET request and some cookies.



With this information, we can now use SQLMap to learn more about the DVWA database. Start by entering `sqlmap --help` and examining the available options. The request URL is our target and after some trial and error it seems the `--cookie=` flag is also necessary. Use the command shown below.



From here, SQLMap will perform a set of automated injection tests and recommend some payloads at the end. Neither produces any results when entered into the box, however, so we will have to do more digging. Add the `--current-db` flag to see what database we're in.



Now use `-D dvwa --tables` to see dvwa's tables. The one called users looks promising.

```
[15:54:12] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL ⩾ 5.0.12
[15:54:12] [INFO] fetching tables for database: 'dvwa'
[15:54:12] [INFO] fetching number of tables for database 'dvwa'
[15:54:12] [INFO] resumed: 2
[15:54:12] [INFO] resumed: guestbook
[15:54:12] [INFO] resumed: users
Database: dvwa
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+
```

Add `-T users --columns` to examine it. Seems like this is where the passwords are.

```
Database: dvwa
Table: users
[8 columns]
+---------------+--------------+
| Column        | Type         |
+---------------+--------------+
| user          | varchar(15)  |
| avatar        | varchar(70)  |
| failed_login  | int(3)       |
| first_name    | varchar(15)  |
| last_login    | timestamp    |
| last_name     | varchar(15)  |
| password      | varchar(32)  |
| user_id       | int(6)       |
+---------------+--------------+
```

Finally, dump the table by replacing `--columns` with `--dump` and let SQLMap crack the passwords.

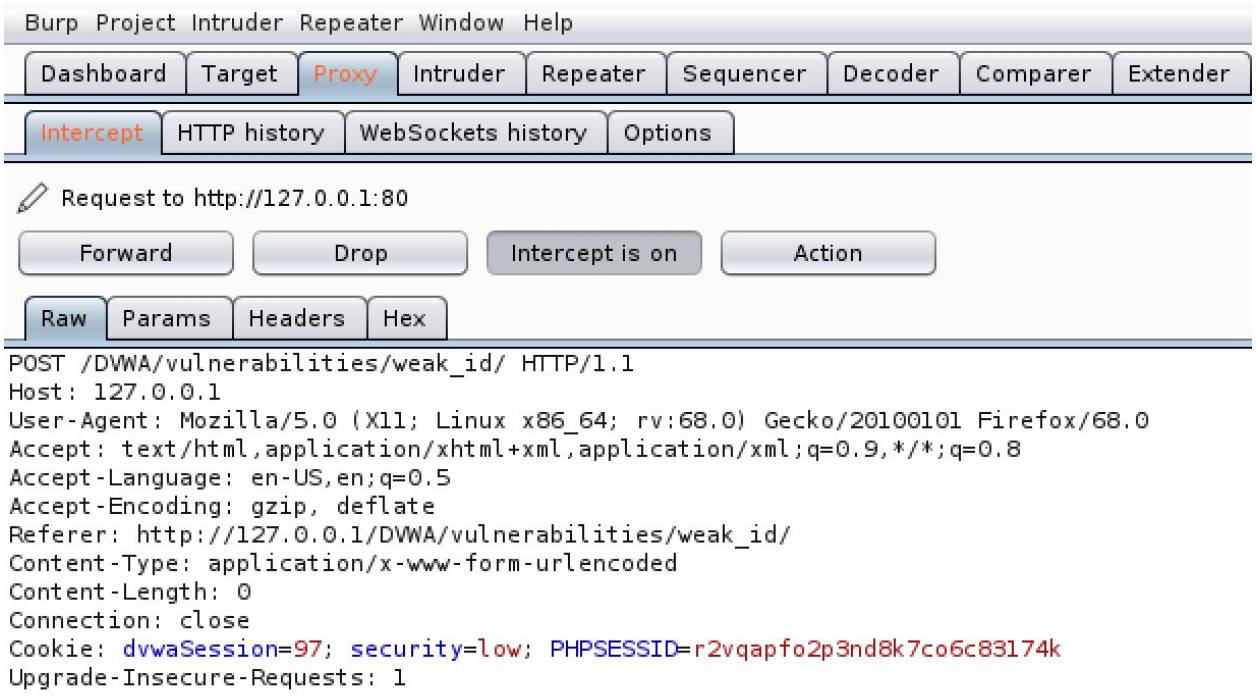
user_id	user	avatar	password	last_name	first_name	last_login	failed_login
3	1337	/DVWA/hackable/users/1337.jpg	8d3533d75ae2c3966d7e0d4fcc69216b (charley)	Me	Hack	2020-05-15 22:05:07	0
1	admin	/DVWA/hackable/users/admin.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	admin	admin	2020-05-15 22:05:07	0
2	gordonb	/DVWA/hackable/users/gordonb.jpg	e99a18c428cb38d5f260853678922e03 (abc123)	Brown	Gordon	2020-05-15 22:05:07	0
4	pablo	/DVWA/hackable/users/pablo.jpg	0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)	Picasso	Pablo	2020-05-15 22:05:07	0
5	smithy	/DVWA/hackable/users/smithy.jpg	5f4dcc3b5aa765d61d8327deb882cf99 (password)	Smith	Bob	2020-05-15 22:05:07	0

We now have the usernames and passwords for all users, including admin, and can access the site for any of our own malicious purposes.

Attack 2: Weak Session IDs

In this attack we exploit DVWA's insecure generation of session IDs. Communicated via cookies, session IDs indicate to the site which user is logged in so that they do not have to continually authenticate themselves. These IDs should be hard to guess and drawn from a large address space so that users' sessions cannot be impersonated.

To start, navigate to the Weak Session IDs tab in the DVWA. Click on the Generate button a few times and observe the intercepts and HTTP history in Burp. In this step we are gathering information about the session IDs to see if there is a pattern we can exploit.



Immediately we notice that the session IDs are simply incrementing by one each time. This means that *any* session ID will be valid and we can break into a session just by picking one. In the next image I've intercepted a request and changed the dvwaSession number to 105.

Original request | Edited request | Response

Raw | Params | Headers | Hex

```
POST /DVWA/vulnerabilities/weak_id/ HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://127.0.0.1/DVWA/vulnerabilities/weak_id/
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
Connection: close
Cookie: dvwaSession=105; security=low; PHPSESSID=r2vqapfo2p3nd8k7co6c83174k
Upgrade-Insecure-Requests: 1
```

Notice that the response still comes back 200 OK. Because the session IDs are predictable, I am able to hijack other users' (including admin's) sessions and potentially steal their information.

Original request | Edited request | Response

Raw | Headers | Hex | HTML | Render

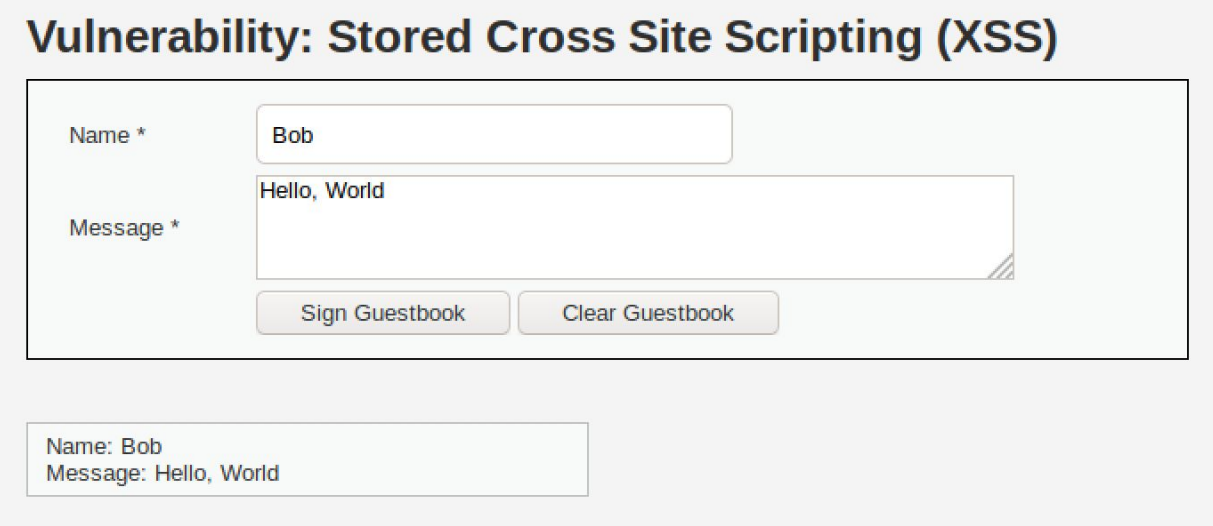
```
HTTP/1.1 200 OK
Date: Tue, 19 May 2020 03:07:01 GMT
Server: Apache/2.4.41 (Debian)
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: dvwaSession=98
Vary: Accept-Encoding
Content-Length: 3517
Connection: close
Content-Type: text/html;charset=utf-8
```

Were the session IDs less obvious or more "rare," we might instead write a script to check many IDs at once for this 200 response. Session IDs must therefore be sufficiently random to avoid these kinds of attack.

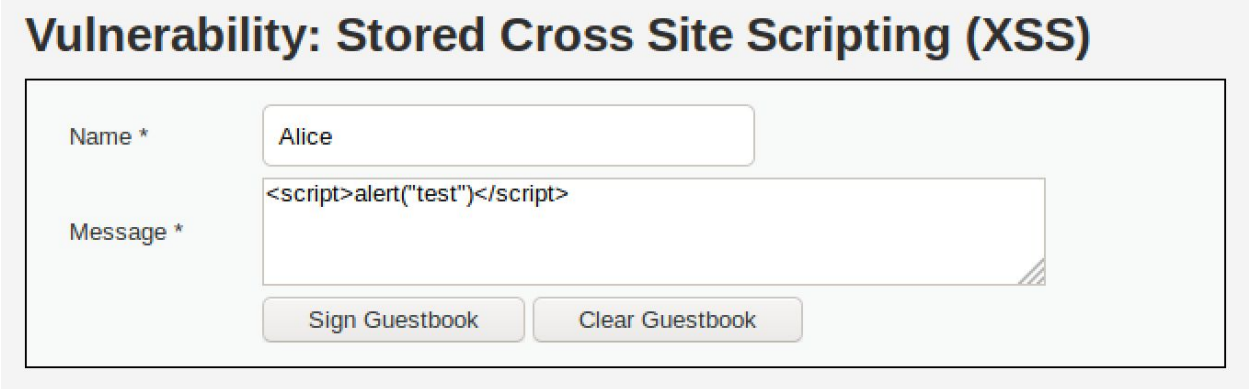
Attack 3: Stored XSS

A cross site scripting attack (XSS) involves injecting a malicious script (typically JavaScript) into an otherwise trusted website so that users of the site unknowingly execute the attacker's code. In a stored XSS attack, the injection persists in the database until manually removed, giving it the capacity for widespread damaging consequences.

Navigate to the XSS (Stored) tab of DVWA and enter a test message to see how the page works. We see that Name and Message are stored in a database and then displayed.



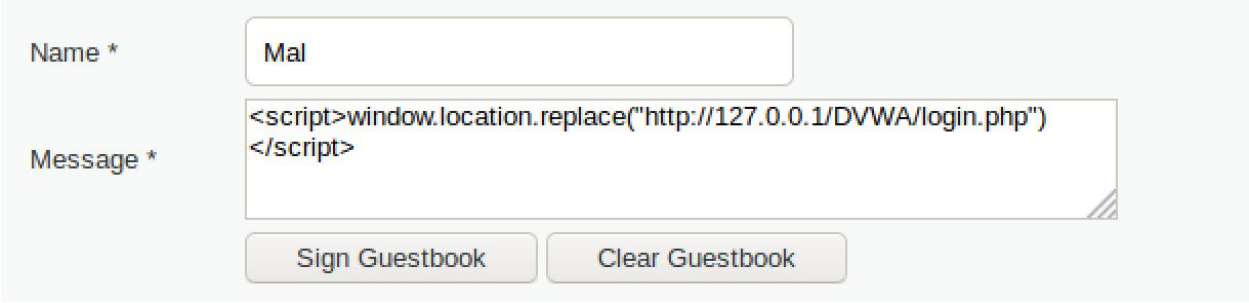
Next, try a simple alert script to check if the page is vulnerable to injection.



Sure enough, an alert box pops up saying "test." Even after we close the box and leave the page, every time we return, the alert appears again. We can see why by inspecting the HTML.

```
<div id="guestbook_comments"> ··· </div>
<div id="guestbook_comments">
  Name: Alice
  <br>
  Message:
  <script>alert("test")</script>
  <br>
</div>
```

The script here is embedded permanently (unless we reset the database) and will execute for everyone who visits the page. Now we can try some more malicious injections, such as redirecting to a different page or site. Use the code below to return to login.



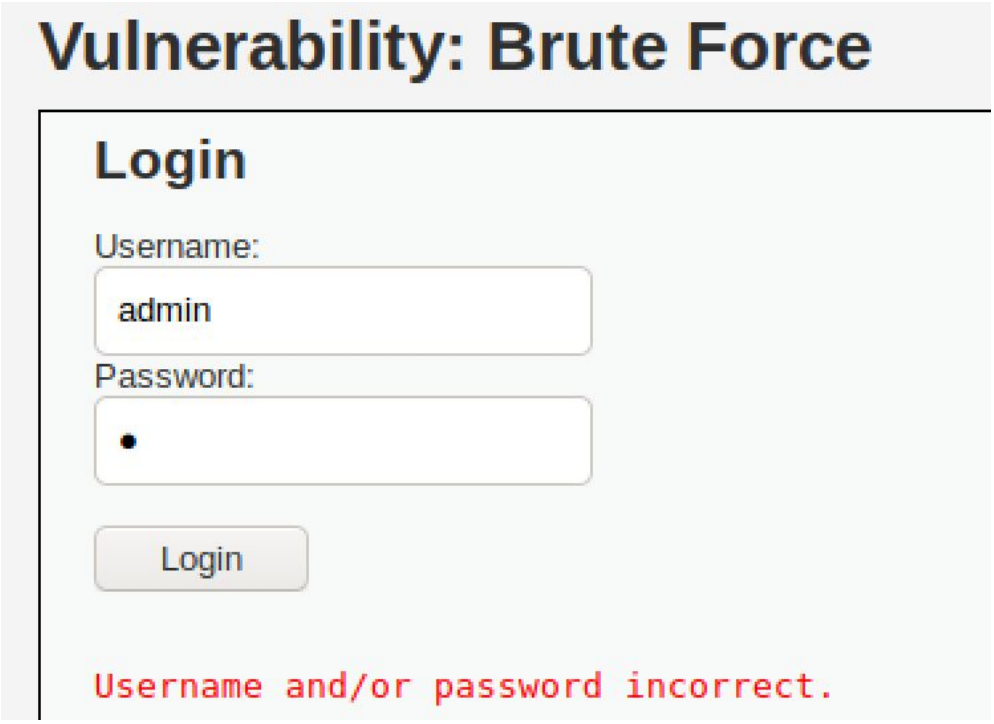
(To do this you will need to inspect the HTML and alter the maximum length for messages.)

```
<td width="100">Message *</td>
<td>
  <textarea name="mtxMessage" cols="50" rows="3" maxlength="1000"></textarea>
</td>
```

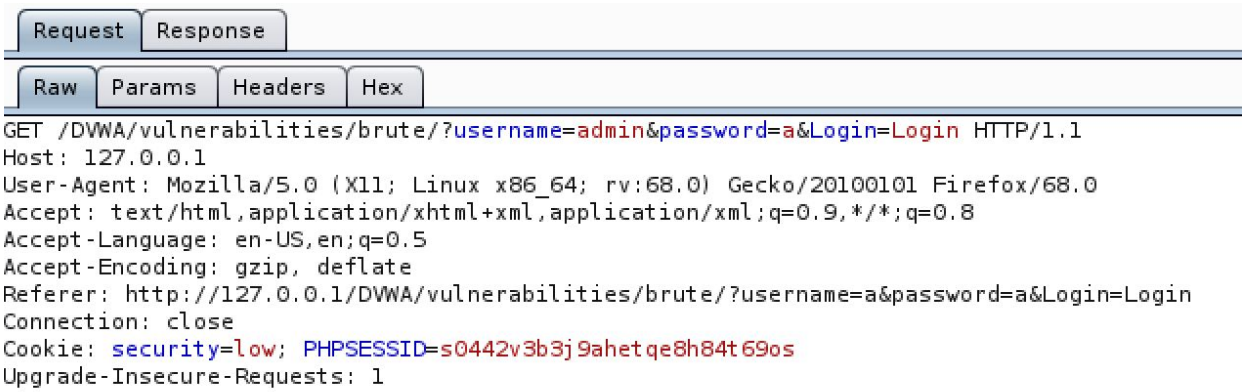
While returning visitors to the login page is relatively harmless, one could easily imagine more damaging redirections, such as to a keylogging spoof page or one embedded with CSRF. With XSS, the attacker gains broad control over the victim's actions without them even knowing.

Attack 4: Brute Force

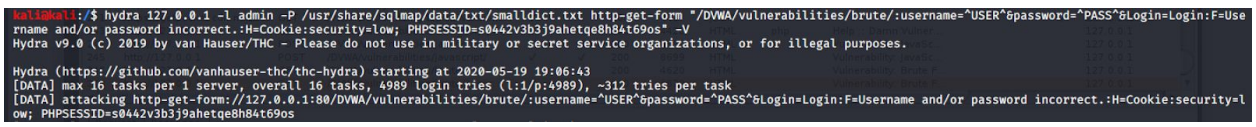
The last attack is to try to guess the admin's password by brute force. This is a relatively unsophisticated strategy but provides the opportunity to work with some new tools. We will use Burp Suite, as before, and Hydra. The first step is of course to gather information. Navigate to the Brute Force tab of the DVWA and try a random password.



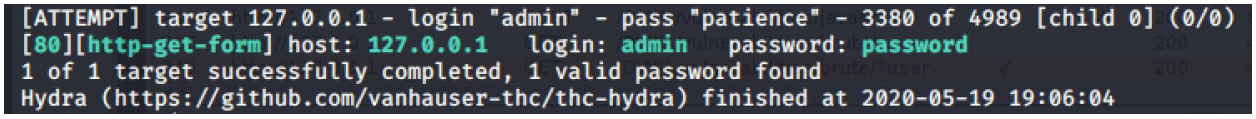
We get it wrong, obviously, and the website prints a message. Notice the traffic in Burp also, which shows the HTTP GET request and cookies.



We now have sufficient information make use of Hydra, a tool for performing brute force dictionary attacks. Start by typing `hydra -h` to see all the options. We know from Burp that the server is 127.0.0.1 and the service is http-get-form. We also have the failure message and the cookie. Choose a password dictionary (I used one included with SQLMap) and compose the command shown here.



Hydra will test every word in the dictionary until it finds a match, which may take a while. Eventually we see that admin's password is password and we can now access the site with administrative privileges.



This attack is made possible by the fact that admin has a very common and easily guessable password. There are also no protections in place against the number of attempts we are making.