Temat numeru

Czy da się oszukać fingerprinting warstwy aplikacji?

Piotr Sobolewski

stopień trudności

Istnieje wiele narzędzi pozwalających określić, jaka usługa działa na danym porcie i jakie oprogramowanie ją obsługuje. Spróbujemy zrozumieć, jak one działają, a następnie zastanówmy się, czy byłoby możliwe (i czy byłoby łatwo) je oszukać.

Jeśli macie pod ręką komputer, wejdźcie na stronę *http://www.netcraft.com.* W okienko what's that site running? wpiszcie adres jakiejś popularnej strony, na przykład www.allegro.pl. Wciśnijcie [enter] – pojawi się informacja o tym, na jakim serwerze postawiona jest ta strona. W naszym przypadku dowiedzieliśmy się, że stronę *www.allegro.pl* obsługuje Apache w wersji 1.3.34 (postawiony na Debianie, z PHP w wersji 4.4.2-1 – patrz Rysunek 1).

Ciekawe, prawda? W dodatku bardzo przydatne. Taka wiedza na pewno przyda się intruzowi chcącemu zaatakować badany sajt (albo osobie przeprowadzającej audyt bezpieczeństwa). Wiedząc, z jaką wersją Apacza ma do czynienia, intruz może poszukać w Internecie informacji o błędach bezpieczeństwa występujących w tej wersji. Jeśli jakieś luki się znajdą, intruz może przystąpić do ataku.

Do czego to się przyda?

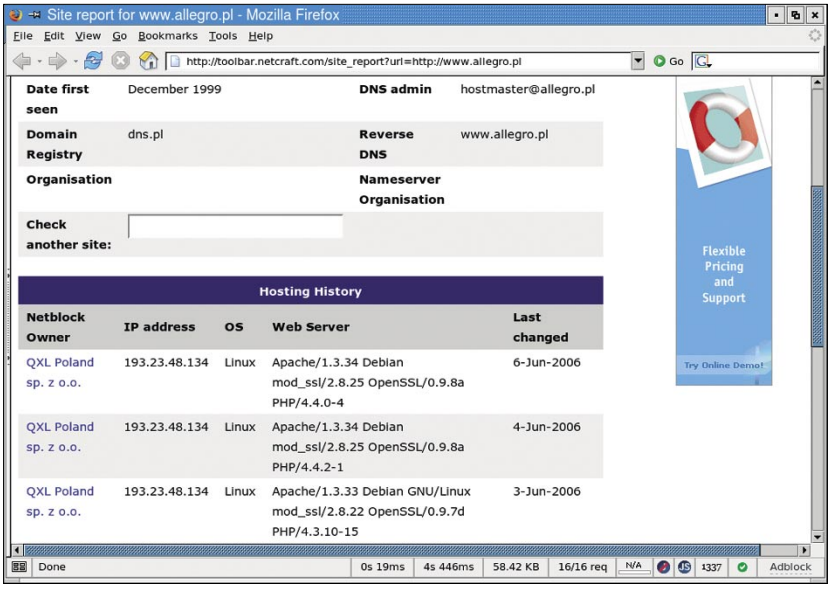
To, co zrobiliśmy przed chwilą – rozpoznanie, jakie oprogramowanie obsługuje daną usługę na zdalnym komputerze – nazywa się fachowo fingerprintingiem warstwy aplikacji (ang. *application level fingerprinting*). Czasem przybiera ono nieco inną postać. Bywa, że nie tylko nie wiemy, jakie oprogramowanie obsługuje daną usługę (Apacz czy IIS, w jakiej wersji), ale nie wiemy nawet, jaka usługa działa na danym otwartym porcie. Po prostu – podczas skanowania portów dowiadujemy się, że na komputerze 192.168.22.33 jest otwarty port 175 i tyle. Może ktoś postawił tam serwer ftp, może www,

Z artykułu dowiesz się...

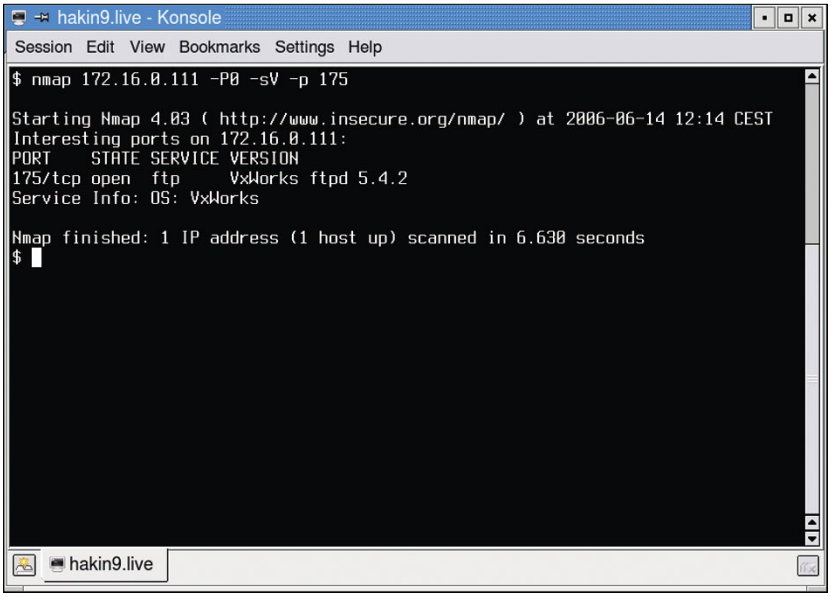
- co to jest application level fingerprinting;
- jakimi technikami się posługuje;
- jakich narzędzi możesz użyć do przeprowadzenia fingerprintingu warstwy aplikacji;
- jakie techniki stosują te narzędzia, a co za tym idzie;
- na ile wiarogodne są wyniki dostarczone przez poszczególne narzędzia;
- na ile trudne (i czy w ogóle możliwe) jest oszukanie poszczególnych narzędzi.

Powinieneś wiedzieć...

- zakładam, że czytelnik dysponuje ogólną wiedzą informatyczną, rozumie jak działa Internet, przyda się - choć nie jest niezbędnie konieczne - podstawowe obycie z linią poleceń Linuksa.



Rysunek 1. *www.netcraft.com informuje nas, jaki serwer obsługuje stronę www.allegro.pl*



Rysunek 2. *Przy pomocy nmapa sprawdzamy, jaka usługa działa na danym porcie i jakie oprogramowanie ją obsługuje (w naszym przypadku usługa to ftp, a oprogramowanie to VxWorks ftpd 5.4.2)*

Listing 1. *Żądanie dokumentu http://www.icm.edu.pl/festiwal/2005/program.html*

```
GET /festiwal/2005/program.html HTTP/1.1
User-Agent: Opera/8.51 (X11; Linux i686; U; en)
Host: www.icm.edu.pl
Accept: text/html, application/xml;q=0.9, application/xhtml+xml, image/png,
                    image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1
Accept-Language: pl,en;q=0.9
Accept-Charset: iso-8859-2, utf-8, utf-16, iso-8859-1;q=0.6, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
Connection: Keep-Alive
(pusta linijka)
```

może jeszcze coś innego? W takiej sytuacji poznany przed chwilą serwis *http://www.netcraft.com* nic nam nie pomoże. Ale jeśli macie komputer z Linuksem (na przykład zabootowany z *hakin9.live*), wystarczy, że wydacie polecenie:

```
$ nmap <adres_IP> -P0 -sV -p <numer_
                        portu>
```

a dowiecie się, jaka usługa działa na tym porcie i jakie oprogramowanie ją obsługuje (Rysunek 2).

Leniwy czytelnik mógłby w tym momencie zakończyć lekturę. Wiemy, jak sprawdzić, jaka usługa działa na tym porcie, umiemy sprawdzić, jakie oprogramowanie ją obsługuje – czego chcieć więcej? Ale dociekliwy czytelnik zacznie się zastanawiać nad kilkoma sprawami:

- Czy to oznacza, że każdy może dowiedzieć się, na jakim serwerze stoi administrowana przeze mnie strona? czy naprawdę nie da się tego jakoś ukryć?
- Na ile wiarygodne są otrzymane przez nas wyniki? Kiedy będę przeprowadzał audyt bezpieczeństwa, to czy mogę ufać, że *netcraft* i *nmap* mówią mi prawdę? Których spośród narzędzi służących do fingerprintingu warstwy aplikacji powinienem używać, a które są niewiarygodne?
- Jak to działa? Inteligentny człowiek potrzebuje rozumieć narzędzia, których używa!

Przyjrzyjmy się więc metodom stosowanym przez różne narzędzia. Spróbujemy zrozumieć, jak one działają, a następnie zastanowimy się, czy byłoby możliwe (i czy byłoby łatwo) je oszukać. Skupimy się przy tym głównie na rozpoznawaniu wersji serwera WWW, a szczególnie blisko przyjrzymy się sprawom związanym z Apaczem.

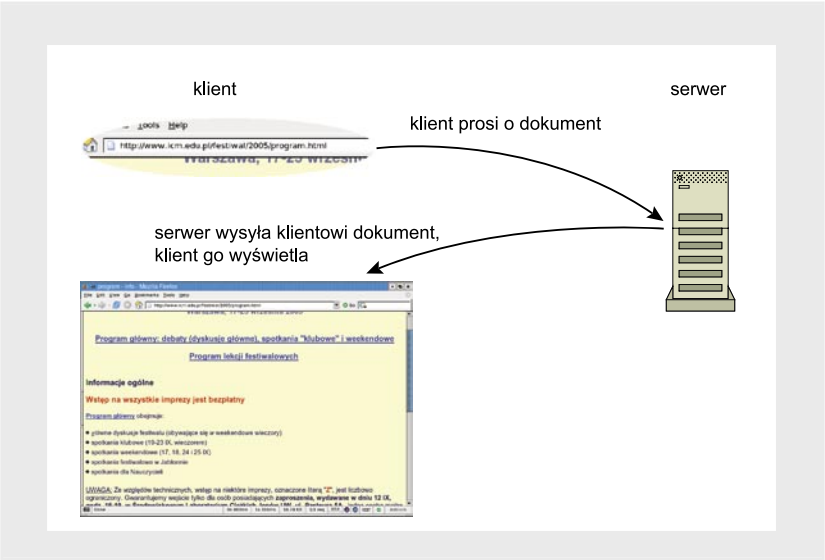
Najprostsza metoda – po prostu spytaj

Zacznijmy od przypomnienia sobie, jak działa protokół HTTP. Otóż kiedy odwiedzamy stronę *http:*

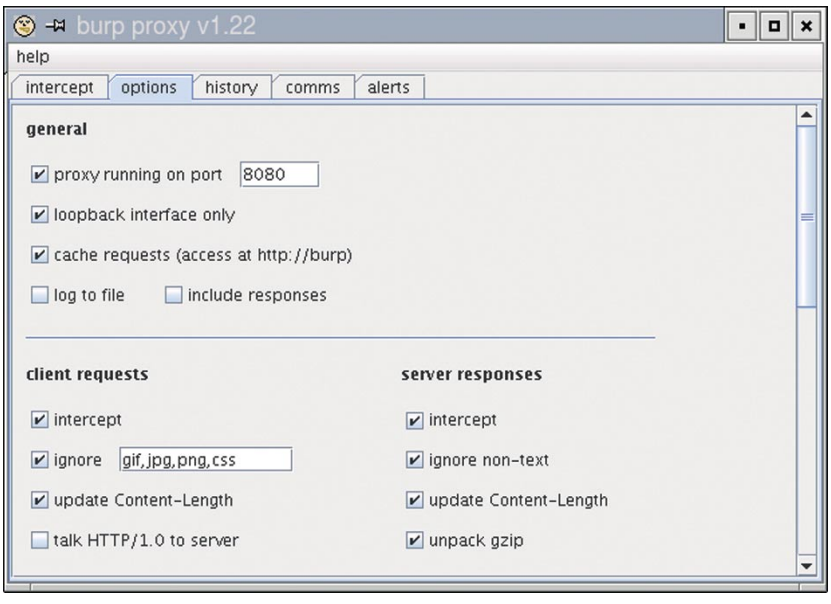
//www.icm.edu.pl/festiwal/2005/program.html, dzieją się trzy rzeczy (Rysunek 3):

- przeglądarka internetowa wysyła do serwera www.icm.edu.pl żądanie o treści: proszę o przesłanie mi dokumentu /festiwal/2005/program.html
- serwer wysyła przeglądarce żądany dokument
- przeglądarka wyświetla go na ekranie

Chcecie zobaczyć, jak wygląda takie przesłanie żądań? Nic prostszego. Możemy użyć przyjemnego narzędzia diagnostycznego o nazwie *burpproxy*. Jest to prosty serwer proxy WWW, który wyświetla przechodzące przez niego żądania i odpowiedzi. Wystarczy, że uruchomimy *burpproxy* na naszym komputerze i skonfigurujemy przeglądarkę tak, by korzystała z serwera proxy na lokalnym komputerze, a cała komunikacja przeglądarki z serwerami WWW będzie przechodzić przez burpproxy, który je wyświetli. Dokładny sposób użycia *burpproxy* podany jest w ramce *Jak korzystać z burpproxy*.



Rysunek 3. *Przeglądarka pobiera z serwera stronę http://www.icm.edu.pl/festiwal/2005/program.html*



Rysunek 4. *Uruchamiamy i konfigurujemy burpproxy*

Spróbujmy podejrzeć przy pomocy *burpproxy*, jak wygląda komunikacja między przeglądarką a serwerem podczas odwiedzin na stronie *http://www.icm.edu.pl/festiwal/2005/program.html* (zachęcam Czytelnika, żeby sam spróbował obejrzeć tę komunikację). Jak widać, żądanie wysyłane przez przeglądarkę wygląda jak na Listingu 1. Pierwsza linijka tego żądania oznacza: chcę pobrać (GET) dokument */festiwal/2005/program.html*. Używam protokołu HTTP w wersji 1.1. Kolejne linijki żądania zawierają dodatkowe informacje (ich znaczenie opisane jest w Tabeli *Żądania i odpowiedzi HTTP – znaczenie*). Żądanie zakończone jest pustą linijką.

Jak korzystać z burpproxy

burpproxy (znajdziesz go na płycie CD dołączonej do pisma) jest napisany w Javie, więc możesz uruchomić go pod Linuksem, Windowsami i Mac OS X. Aby przy pomocy burpproxy przyjrzeć się komunikacji między przeglądarką Mozilla Firefox a serwerami WWW (Rysunek 4):

- uruchom *burpproxy*,
- w *burpproxy* wejdź w zakładkę *options* i zaznacz *server responses -> intercept*,
- w Firefoksie wejdź w ustawienia (*edit -> preferences*), tam wejdź w *general -> connection settings*, następnie jako serwer proxy (pole *HTTP proxy*) ustaw *127.0.0.1*, port *8080*,
- w *burpproxy* wejdź w zakładkę *intercept* – to tu pojawiać się będzie treść żądań i odpowiedzi,
- wróć do Firefoksa i w zwykły sposób przeglądaj strony WWW; za każdym razem, kiedy *burpproxy* przechwyci żądanie albo odpowiedź serwera, wyświetli jej treść w zakładce intercept i będzie czekał z przesłaniem jej dalej, dopóki nie wciśniesz przycisku *forward*.

Jeśli korzystasz z przeglądarki innej niż Firefox, we właściwy dla niej sposób skonfiguruj ją tak, by korzystała z serwera proxy działającego pod adresem 127.0.0.1 na porcie 8080.

Tabela 1. *Żądania i odpowiedzi HTTP – znaczenie*

kolejne wiersze żądania	znaczenie
GET /festiwal/2005/program.html HTTP/1.1	przeglądarka chce pobrać (*GET*) dokument */festiwal/2005/program.html*, używa protokołu HTTP w wersji 1.1
User-Agent: Opera/8.51 (X11; Linux i686; U; en)	przeglądarka przedstawia się jako Opera 8.51
Host: www.icm.edu.pl	żądanie skierowane jest do serwera *www.icm.edu.pl* (ma to znaczenie, kiedy jedna maszyna obsługuje kilka wirtualnych serwerów)
Accept: text/html, application/xml;q=0.9, application/xhtml+xml, image/png, image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1	przeglądarka deklaruje, jakie dokumenty (jakiego typu MIME) gotowa jest przyjąć
Accept-Language: pl,en;q=0.9	przeglądarka deklaruje, w jakich językach dokumenty gotowa jest przyjąć (najchętniej w języku polskim, mniej chętnie w angielskim)
Accept-Charset: iso-8859-2, utf-8, utf-16, iso-8859-1;q=0.6, *;q=0.1	przeglądarka deklaruje, w jakich kodowaniach dokumenty gotowa jest przyjąć
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0	przeglądarka deklaruje, jak skompresowane dokumenty gotowa jest przyjąć
Connection: Keep-Alive	przeglądarka prosi, żeby serwer nie zrywał połączenia po odesłaniu odpowiedzi – dzięki temu wysłanie ewentualnych kolejnych żądań pójdzie szybciej
(pusta linijka)	żądanie kończy się pustą linijką
kolejne wiersze odpowiedzi serwera	**znaczenie**
HTTP/1.1 200 OK	żądanie zostało obsłużone prawidłowo, przesyłam żądany dokument
Date: Wed, 05 Oct 2005 12:46:18 GMT	data wysłania odpowiedzi
Server: Apache/1.3.33 (Debian GNU/Linux) PHP/4.3.10-15 mod_ssl/2.8.9 OpenSSL/0.9.6c mod_perl/1.29	serwer przedstawia się jako Apache 1.3.33
X-Powered-By: PHP/4.3.10-15	na serwerze zainstalowane jest PHP w wersji 4.3.10-15
Connection: close	serwer chce zamknąć połączenie TCP po przesłaniu tej odpowiedzi
Content-Type: text/html	przesyłany dokument jest typu text/html
(pusta linijka)	pusta linijka oznacza koniec nagłówków HTTP i początek właściwego dokumentu
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"> <html> <head> (...)	dalsza część odpowiedzi zawiera dokument, którego żądała przeglądarka

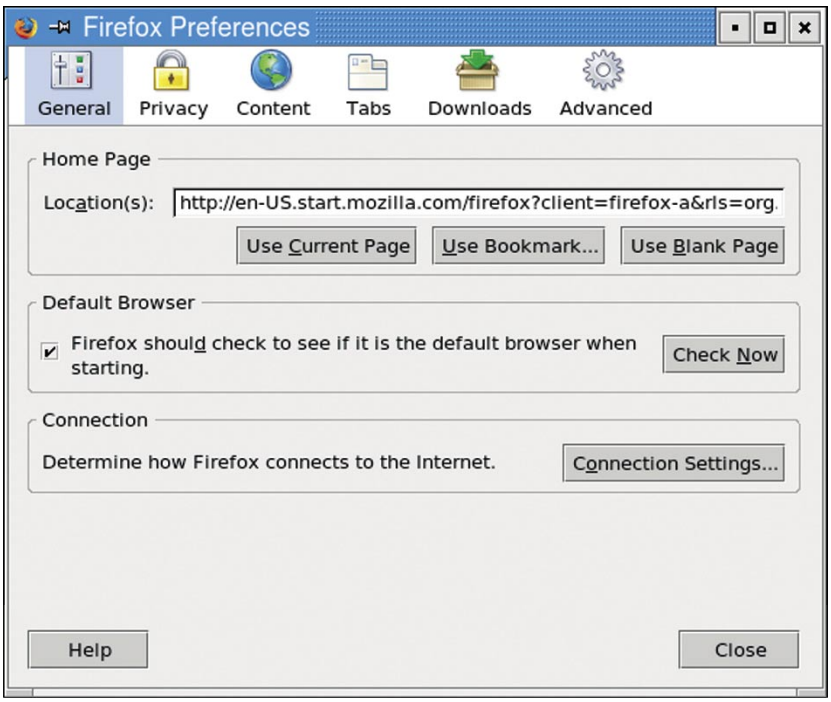
W odpowiedzi serwer wysyła przeglądarce odpowiedź przedstawioną na Listingu 2. Pierwsza linijka tej odpowiedzi oznacza: *używam protokołu HTTP w wersji 1.1 (HTTP/1.1)*, *przesyłam ci żądany dokument* (200 to kod błędu oznaczający *wszystko w porządku*). Kolejne linijki zawierają dodatkowe informacje (szczegółowo opisane w Tabeli *Żądania i odpowiedzi HTTP – znaczenie*). Następnie widzimy pustą linijkę, po której następuje właściwa treść przysłanego dokumentu.

Dla nas szczególnie interesująca jest jedna linijka odpowiedzi serwera:

```
Server: Apache/1.3.33 (Debian GNU
/Linux) PHP/4.3.10-15 mod_ssl/2.8.9
 OpenSSL/0.9.6c mod_perl/1.29
```

W linijce tej serwer przedstawia się przeglądarce: podana jest nazwa serwera (*Apache*), wersja (1.3.33) i inne szczegóły.

Mamy więc sposób na proste dowiedzenie się, z jakim serwerem mamy do czynienia! Jeśli używanie przeglądarki w połączeniu z *burpproxy* wydaje ci się zbyt skomplikowane, możesz po prostu nawiązać połą-



Rysunek 5. *Konfigurujemy Firefoksa*

czenie TCP z serwerem przy pomocy *netcata* i żądanie wpisać z palca. W tym celu wydaj polecenie:

```
$ nc www.icm.edu.pl 80 -v
```

Wydane polecenie oznacza: *połącz się z komputerem www.icm.edu.pl, z portem 80*. Opcja *-v* powoduje, że kiedy połaczenie zostanie nawiązane, program nas o tym poinformuje. Kiedy pojawi się informacja, że połączenie zostało nawiązane (*krankenschwester.icm.edu.pl [212.87.0.40] 80 (www) open*), wpisz:

```
GET / HTTP/1.0
(pusta linijka)
```

Jest to bardzo proste żądanie HTTP, zawierające tylko najpotrzebniejsze rzeczy. Należy rozumieć je tak: *chcę pobrać (GET) dokument / (czyli stronę główną), używam protokołu HTTP w wersji 1.0*. Nie zapomnij o pustej linijce kończącej żądanie!

Przyjrzyj się odpowiedzi, którą otrzymałeś od serwera – prawdopodobnie jest ona podobna do tej, którą widzieliśmy w *burpproxy*:

```
HTTP/1.1 200 OK
Date: Wed, 05 Oct 2005 12:46:18 GMT
Server: Apache/1.3.33 (Debian GNU/
Linux) PHP/4.3.10-15 mod_ssl/2.8.9
 OpenSSL/0.9.6c mod_perl/1.29
X-Powered-By: PHP/4.3.10-15
(...)
```

Widzicie? Właśnie dowiedzieliśmy się, że rozmawiamy z serwerem Apache w wersji 1.3.33.

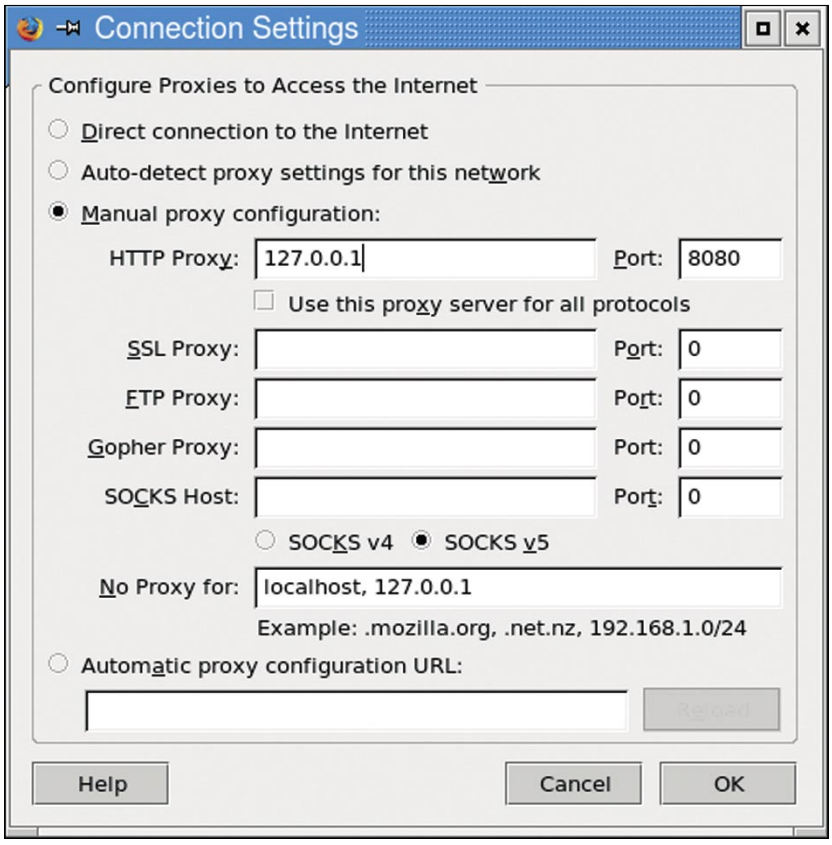
Jak widzicie, sposób jest prosty – zachęcam Czytelników do samodzielnego przebadania kilku sajtów.

To samo dla FTP

Również inne usługi mają zwyczaj się przedstawiać. Kiedy łączymy się z serwerem FTP, ten zazwyczaj podaje nam swoją nazwę i numer wersji:

```
$ nc sunsite.icm.edu.pl 21 -v
(...)
220 SunSITE.icm.edu.pl FTP server (
      Version wu-2.6.2(9) Fri Jun 17
      21:45:54 MEST 2005) ready.
```

Jak widać, w przypadku usługi FTP jest nawet łatwiej niż z WWW. Serwer FTP od razu po nawiązaniu połączenia przysyła nam informację o sobie. Taką informację nazywamy banerem usługi, a to, co właśnie robimy, fachowo nazywa się badaniem banerów (ang. *banner grabbing*).



Rysunek 6. *Konfigurujemy serwer Proxy dla Firefoksa*

Wady tej metody

Metoda, którą przed chwilą poznaliśmy, ma jedną wadę: serwer może skłamać. Kwestią praktyczną – czyli jak na przykład skonfigurować Apacza, żeby podawał się za IIS – zajmiemy się za chwilę, na razie tylko zauważmy, że nie ma żadnego powodu, żeby serwer miał mówić prawdę. Apache może powiedzieć, że jest IIS-em – i skąd możemy dowiedzieć się, że kłamie?

Dlatego powstały inne metody rozpoznawania wersji serwera WWW (i ogólniej: wersji oprogramowania obsługującego daną usługę) – metody, które trudniej jest oszukać.

Różnice w implementacji standardu

Protokół HTTP jest dokładnie opisany i zdefiniowany w odpowiednich dokumentach RFC. Określone jest, jak powinno wyglądać żądanie, co oznacza który kod błędu itp. Z konieczności jednak nie wszystko zostało określone do ostatniego szczegółu, nie każda możliwa sytuacja została opisana.

Przykładowo: protokół HTTP ma kilka wersji – między innymi wersję 0.9 i wersję 1.0. W wersji 0.9 żądanie wyglądało po prostu tak:

Tabela 2. *Jak czytać output nmapa*

output nmapa	znaczenie
Interesting ports on gecew (127.0.0.1): PORT STATE SERVICE VERSION 21/tcp open ftp vsFTPd 2.0.1	otwarty port 21, usługa to ftp, chodzi tam demon vsFTPd w wersji 2.0.1
Interesting ports on gecew (127.0.0.1): PORT STATE SERVICE VERSION 8443/tcp open ssl/http Apache httpd	tu nmap znalazł otwarty port 8443, na nim chodzi SSL, a za SSL rozpoznany został Apache
Interesting ports on gecew (127.0.0.1): PORT STATE SERVICE VERSION 8443/tcp open ssl/ftp vsFTPd 2.0.1	za SSL mogą też być inne usługi – w tym przypadku FTP
Interesting ports on gecew (127.0.0.1): PORT STATE SERVICE VERSION 8080/tcp open http-proxy?	tak wygląda, kiedy nmap nie rozpozna demona; o tym, jaka to usługa, wnioskuje na podstawie numeru portu – ponieważ zaś jest to niepewne, więc przy nazwie usługi jest pytajnik

Listing 2. *Odpowiedź serwera*

```
HTTP/1.1 200 OK
Date: Wed, 05 Oct 2005 12:46:18 GMT
Server: Apache/1.3.33 (Debian GNU/Linux) PHP/4.3.10-15 mod_ssl/2.8.9 OpenSSL/
                      0.9.6c mod_perl/1.29
X-Powered-By: PHP/4.3.10-15
Connection: close
Content-Type: text/html
(pusta linijka)
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<!-- saved from url=(0049)http://www.icm.edu.pl/festiwal/2005/program0.html
                      -->
  <title>program - info</title>
(...)
```

Listing 3. *Przykładowe wyrażenie regularne ze wzorcem odpowiedzi serwera z pliku konfiguracyjnego vmapa (fragment pliku z katalogu http/wo/server_name)*

```
HTTP/1\.1 302 Found\+
Date: .*\+
Server: .*\+
X-Powered-By: .*\+
X-Accelerated-By: .*\+
Set-Cookie: .*; path=/\+
Expires: .*\+
(...)
```

```
GET /index.html
(empty line)
```

Jak widać, pierwsza linijka żądania ma budowę:

```
<metoda> <ścieżka do dokumentu>
```

W wersji 1.0 analogiczne żądanie wygląda w ten sposób:

```
GET /index.html HTTP/1.0
(empty line)
```

Tu pierwsza linijka żądania ma budowę:

```
<metoda> <ścieżka do dokumentu>
      <HTTP/wersja.protokołu>
```

Jak widać, łatwo odróżnić, którą wersję protokołu stosuje klient. Chyba że wyśle on takie żądanie:

```
TRALALA
(empty line)
```

Nie jest to ani prawidłowe żądanie protokołu HTTP w wersji 0.9, ani w wersji 1.0. A jednak serwer musi się na coś zdecydować! Bo jeśli rozmawiamy protokołem 0.9, to również odpowiedź serwera powinna być właściwa dla tej wersji protokołu. W HTTP/0.9 serwer po prostu wysyła

Listing 4. *Przykładowy raport httprinta*

```
Finger Printing on http://127.0.0.1:80/
Finger Printing Completed on http://127.0.0.1:80/
--------------------------------------------------
Host: 127.0.0.1
Derived Signature:
Microsoft-IIS/6.0
9E431BC86ED3C295811C9DC5811C9DC5050C5D32505FCFE84276E4BB811C9DC5
0D7645B5811C9DC5811C9DC5CD37187C11DDC7D7811C9DC5811C9DC58A91CF57
FCCC535B6ED3C295FCCC535B811C9DC5E2CE6927050C5D336ED3C2959E431BC8
6ED3C295E2CE69262A200B4C6ED3C2956ED3C2956ED3C2956ED3C295E2CE6923
E2CE69236ED3C295811C9DC5E2CE6927E2CE6923
Banner Reported: Microsoft-IIS/6.0
Banner Deduced: Apache/2.0.x
Score: 140
Confidence: 84.34
------------------------
Scores:
Apache/2.0.x: 140 84.34
Apache/1.3.[4-24]: 132 68.91
Apache/1.3.27: 131 67.12
Apache/1.3.26: 130 65.36
Apache/1.3.[1-3]: 127 60.28
TUX/2.0 (Linux): 123 53.90
Apache/1.2.6: 117 45.20
Agranat-EmWeb: 86 14.44
Stronghold/4.0-Apache/1.3.x: 77  9.25
Com21 Cable Modem: 70  6.16
(...)
```

żądany dokument, bez żadnych nagłówków:

```
<html>
<head>
  <title>program - info</title>
(...)
```

Natomiast w HTTP/1.0 – jak widzieliśmy poprzednio – odpowiedź serwera zaczyna się od nagłówka HTTP, na przykład:

```
HTTP/1.1 200 OK
Server: Apache/1.3.33
Content-Type: text/html
<html>
<head>
  <title>program - info</title>
(...)
```

Spróbujcie połączyć się z kilkoma serwerami (tak jak poprzednio, przy pomocy *netcata*) i wysłać żądanie TRALALA. Zwróćcie uwagę na różne odpowiedzi udzielane przez różne serwery. Najlepiej przeprowadźcie trochę prób na serwerach, co do których wiecie, jakie oprogramowanie jest na nich zainstalowane. Szybko przekonacie się, że Apache zwykle nasze bezsensowne żądanie traktuje jako HTTP/0.9, a IIS - jako HTTP/1.0.

Sposób wydaje się sprytny, prawda? O ile – nadal bez wchodzenia w szczegóły techniczne – wydaje się, że zmiana konfiguracji Apacza tak, żeby w nagłówkach HTTP przedstawiał się jako IIS, powinna być prosta, to spowodowanie, że serwer inaczej będzie traktował nasze bezsensowne żądanie (o treści *TRALALA*), wygląda na bardziej skomplikowane zadanie – prawdopodobnie wymagające zmian w kodzie serwera.

Inne różnice

Jest dużo takich drobnych różnic w implementacji protokołu HTTP. Inny przykład: mało znana metoda protokołu HTTP o nazwie DELETE. Niemal żaden z popularnych serwerów w domyślnej konfiguracji jej nie obsługuje. Ale jeśli spróbujemy wysłać do kilku serwerów żądanie:

```
DELETE / HTTP/1.0
(pusta linia)
```

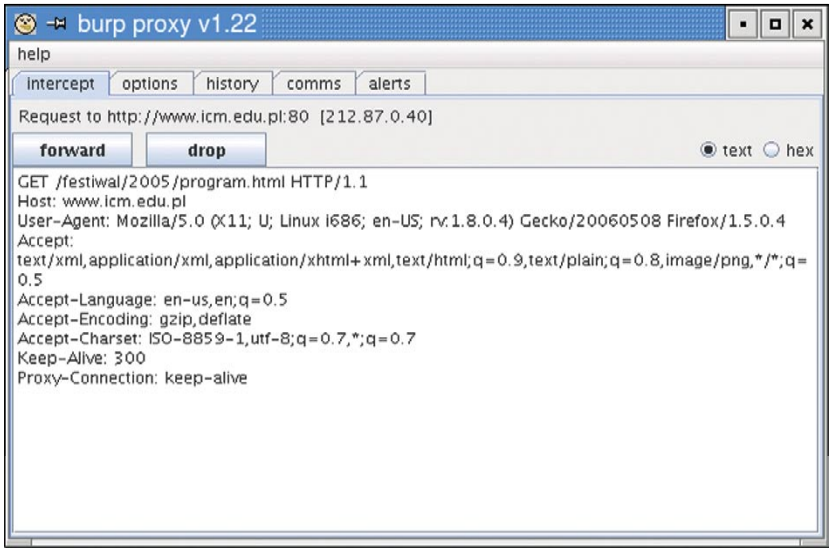
to zobaczymy (zachęcam was, Czytelnicy, żebyście sami spróbowali), że Apache zwykle odpowiada:

```
HTTP/1.1 405 Method Not Allowed
(...)
```

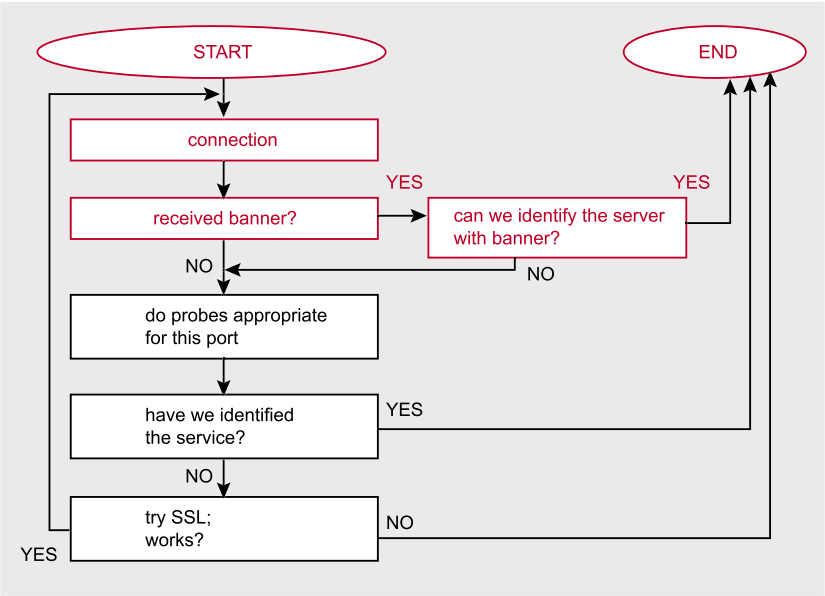
Z kolei serwery IIS-y informują nas:

```
HTTP/1.1 404 Not Found
(...)
```

Jak widać w obu przypadkach nasze żądanie nie jest obsłużone, ale za każdym razem zwracany jest inny kod błędu.



Rysunek 7. *Burpproxy wyświetla przechwycone żądanie i czeka, aż pozwolimy przesłać je dalej*



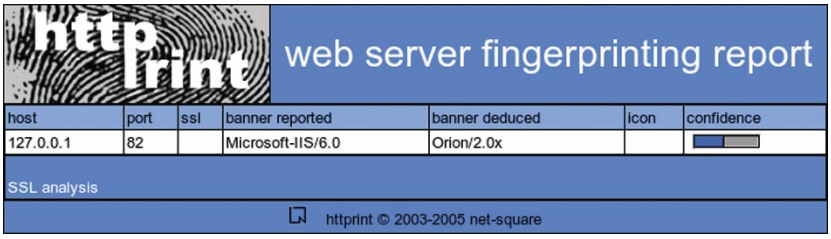
Rysunek 8. *W ten sposób nmap rozpoznaje usługę i obsługujące ją oprogramowanie*

Listing 5a. *Przykładowy raport hmapa*

```
                                    matches : mismatches : unknowns
Apache/2.0.44 (Win32)                 110 :   5 :   8
Apache/2.0.40 (Red Hat 8.0)           110 :   4 :   9
IBM_HTTP_Server/2.0.42 (Win32)        108 :   6 :   9
Apache/1.3.12 (Win32)                 108 :   8 :   7
Apache/1.3.14 (Win32)                 108 :   8 :   7
Apache/1.3.17 (Win32)                 108 :   8 :   7
Apache/1.3.22 (Win32)                 108 :   8 :   7
Apache/1.3.9 (Win32)                  107 :   8 :   8
Apache/1.3.27 (Red Hat 8.0)            90 :  25 :   8
Apache/1.3.23 (RedHat Linux 7.3)       89 :  26 :   8
```

Tabela 3. *Jak czytać output amapa*

wynik działania amapa	znaczenie
Protocol on 127.0.0.1:25/tcp matches smtp Unidentified ports: none	amap rozpoznał, że na porcie 25 chodzi SMTP
Protocol on 66.249.85.99:443/tcp matches ssl Protocol on 66.249.85.99:443/tcp over SSL matches http Unidentified ports: none	na porcie 443 jest SSL, a za nim HTTP
Unidentified ports: 127.0.0.1:12300/tcp (total 1)	port 12300 – usługa nierozpoznana



Rysunek 9. *Wygenerowany przez httprinta raport w HTML-u*

Narzędzia

Można by więc pracę zorganizować sobie tak: przygotować tabelę zawierającą różne drobiazgi pozwalające odróżnić Apache od IIS, IIS od serwera LiteSpeed, Apache w wersji 1.3 od Apache w wersji 1.4 itp. Następnie należałoby usiąść do *netcata*, pracowicie wykonywać test po teście, zapisywać wyniki, porównywać z tabelką, analizować... Oczywiście nikt tego w ten sposób nie robi. Jak już widzieliśmy, są narzędzia, które same przeprowadzą odpowiednie testy. Przyjrzyjmy się kilku popularniejszym.

nmap

nmap, bardzo popularny skaner portów, potrafi również rozpoznawać, jaka usługa działa na danym porcie i jakie oprogramowanie ją obsługuje – wystarczy uruchomić go z opcjami *-sV*; na przykład:

```
# nmap -sV -p 80 www.google.com
```

W odpowiedzi *nmap* wypisze informację o tym, co kryje się za badanym portem. Informacje te są dosyć zrozumiałe, w razie wątpliwości zajrzyj do Tabelki *Jak czytać output nmapa*.

Jak działa nmap?

Jak widać *nmap* całkiem sprawnie rozpoznaje różne usługi. Jego sposób działania jest opisany w dokumentacji (*http://www.insecure.org/nmap/vscan/*). Jeśli przeanalizujemy ten opis, otrzymamy schemat przedstawiony na Rysunku 5.

Jak widać, po nawiązaniu połączenia *nmap* czeka na otrzymanie banera (czyli czeka, aż usługa sama się przedstawi). Jeśli otrzyma baner, porównuje go z listą znanych banerów. Jeśli baner jest znany, usługa jest rozpoznana. Jeśli baner jest nieznany, albo *nmap* wcale go nie otrzymał, *nmap* wykonuje próby właściwe dla tego portu (a więc na przykład w przypadku portu 80 – zwyczajowo używanego dla usługi WWW – *nmap* wykona omawiane wcześniej próby związane z różną implementacją protokołu HTTP przez różne serwery). Jeśli i w ten sposób nie uda się rozpoznać usługi, nmap sprawdza,

Listing 5b. *Plik apache.1.3.12.win32 (fragment)*

```
{'LEXICAL': {'200': 'OK',
             '400': 'Bad Request',
             '403': 'Forbidden',
             '404': 'Not Found',
             '405': 'Method Not Allowed',
             '406': 'Not Acceptable',
             '414': 'Request-URI Too Large',
             '501': 'Method Not Implemented',
             'SERVER_NAME': 'Apache/1.3.12 (Win32)'},
 'SEMANTIC': {'LARGE_HEADER_RANGES': [(1, '200'),
                                      (8176, '200'),
                                      (8177, '400'),
                                      (10000, '400')],
              'LONG_DEFAULT_RANGES': [(1, '200'),
                                      (201, '200'),
                                      (202, '403'),
                                      (8177, '403'),
                                      (8178, '414'),
                                      (10000, '414')],
              'LONG_URL_RANGES': [(1, '404'),
                                  (210, '404'),
                                  (211, '403'),
                                  (8176, '403'),
                                  (8177, '414'),
                                  (10000, '414')],
              'MALFORMED_000': 'NO_RESPONSE_CODE',
              'MALFORMED_001': 'NO_RESPONSE_CODE',
              'MALFORMED_002': '400',
              'MALFORMED_003': '200',
              'MALFORMED_004': '200',
              'MALFORMED_005': '200',
              'MALFORMED_006': '200',
              'MALFORMED_007': '200',
              'MALFORMED_008': '200',
              'MALFORMED_009': '200',
              'MALFORMED_010': '200',
              'MALFORMED_011': '200',
              'MALFORMED_012': '400',
              'MALFORMED_013': '400',
              'MALFORMED_014': '400',
              'MALFORMED_015': '400',
              'MALFORMED_016': '200',
              'MALFORMED_017': '200',
              'MALFORMED_018': '200',
              'MALFORMED_019': 'NO_RESPONSE_CODE',
              'MALFORMED_020': 'NO_RESPONSE_CODE',
              'MALFORMED_021': 'NO_RESPONSE_CODE',
              'MALFORMED_022': 'NO_RESPONSE_CODE',
              'MALFORMED_023': 'NO_RESPONSE_CODE',
              'MALFORMED_024': 'NO_RESPONSE_CODE',
              'MALFORMED_025': 'NO_RESPONSE_CODE',
              'MALFORMED_026': '200',
              'MALFORMED_027': '200',
              'MALFORMED_028': '200',
              'MALFORMED_029': '200',
              'MALFORMED_030': '200',
              'MALFORMED_031': '200',
              'MALFORMED_032': '400',
              'MALFORMED_033': '400',
              'MALFORMED_034': '400',
              'MALFORMED_035': '400',
              'MALFORMED_036': '200',
              'MALFORMED_037': '200',
              'MALFORMED_038': '501',
```

czy aby na tym porcie nie ma SSL. Jeśli tak jest, nmap łączy się przez SSL i cały cykl sprawdzania zaczyna się od nowa.

Jak oszukać nmapa

Przyjrzyjmy się jeszcze raz dokładnie Rysunkowi 5. Widzimy że choć *nmap* umie przeprowadzać sprytne próby związane z różną implementacją protokołów przez różne serwery, to w sytuacji, kiedy rozpozna otrzymany od usługi baner, prób tych nie wykonuje (ścieżka zaznaczona na Rysunku 5 na czerwono). Wniosek prosty: wystarczy spowodować, aby nasz serwer FTP *vsftpd* przedstawiał się jako jakiś inny znany *nmapowi* serwer, a *nmap* uwierzy w to. Spróbujmy przeprowadzić to w praktyce.

Listę znanych *nmapowi* banerów znajdziemy w pliku konfiguracyjnym *nmap-service-probes* (na przykład */usr/share/nmap/nmap-service-probes*). Wśród wielu znajdujących się tam banerów serwerów FTP znajdujemy na przykład taki: *VxWorks* (5.4.2) FTP server ready.

Aby nakazać serwerowi vsftpd przedstawianie się takim banerem, musimy do jego pliku konfiguracyjnego (/etc/vsftpd.conf) dopisać linijkę:

```
ftpd_banner=VxWorks (5.4.2) FTP server
                        ready
```

Jeśli zrestartujemy serwer FTP (`# /etc/init.d/vsftpd restart`), a następnie zbadamy go *nmapem*, ten powie nam:

```
Interesting ports on gecew (127.0.0.1):
PORT    STATE SERVICE VERSION
21/tcp open  ftp     VxWorks ftpd 5.4.2
Service Info: OS: VxWorks
```

Udało nam się oszukać *nmapa!*

Uwaga: zauważmy, że gdybyśmy nakazali serwerowi FTP, żeby przedstawiał się jako – na przykład – *some ftp server, to nmap* będzie rozpoznawał go jako:

```
Interesting ports on gecew (127.0.0.1):
PORT    STATE SERVICE VERSION
21/tcp open  ftp     vsftpd or WU-FTPD
```

Listing 5c. *Plik apache.1.3.12.win32 (fragment)*

```
            'MALFORMED_039': '200',
            'MALFORMED_040': 'NO_RESPONSE_CODE',
            'MALFORMED_041': '400',
            'MALFORMED_042': '200',
            'MALFORMED_043': '200',
            'MALFORMED_044': 'NO_RESPONSE_CODE',
            'MALFORMED_045': '200',
            'MALFORMED_046': '400',
            'MALFORMED_047': '400',
            'MALFORMED_048': '400',
            'MALFORMED_049': '400',
            'MALFORMED_050': '200',
            'MALFORMED_051': '400',
            'MALFORMED_052': 'NO_RESPONSE_CODE',
            'MALFORMED_053': 'NO_RESPONSE_CODE',
(...)
```

Jak więc widać, jeśli serwer FTP przedstawia się banerem nieznanym nmapowi, ten przeprowadza dodatkowe testy i całkiem sensownie rozpoznaje *vsftpd*.

To samo w przypadku Apache

Jak widać, z serwerem FTP poszło łatwo. Trudniej jest zmusić Apache, żeby przedstawiał się jako IIS. W oficjalnej dokumentacji (*http://httpd.apache.org/docs/1.3/misc/FAQ.html#serverheader*) czytamy:

How can I change the information that Apache returns about itself in the headers?

(...)

The answer you are probably looking for is how to make Apache lie about what what it is, ie send something like:

```
Server: Bob's Happy HTTPd Server
```

In order to do this, you will need to modify the Apache source code and rebuild Apache. This is not advised, as it is almost certain not to provide you with the added security you think that you are gaining. The exact method of doing this is left as an exercise for the reader, as we are not keen on helping you do something that is intrinsically a bad idea.

A więc twórcy Apache oficjalnie odradzają stosowanie takich sztuczek i mówią, że w zwykły sposób (przez edycję plików konfiguracyjnych) nie da się przekonfigurować tego serwera tak, żeby przedstawiał się jako IIS. Jedyne co można zrobić, to nakazać Apaczowi, żeby nie podawał swojej wersji. W tym celu w pliku konfiguracyjnym *(httpd.conf*) musimy znaleźć linijkę o treści:

```
ServerSignature On
```

i zmienić ją na:

```
ServerSignature Off
```

Spowoduje to, że na stronach wygenerowanych przez serwer (informacje o błędach itp) nie będzie dodawana stopka z nazwą serwera. Następnie poniżej tej linijki dopisujemy:

```
ServerTokens Prod
```

To spowoduje, że Apache w banerze nie będzie podawał numeru wersji.

Jeśli używamy PHP, trzeba zwrócić uwagę na komendę *expose_php* w pliku konfiguracyjnym PHP *(php.ini*) – powoduje ona, że PHP informuje o tym, że jest zainstalowane na tym serwerze WWW (na przykład przez podanie swoich informacji w nagłówkach HTTP), a raczej tego nie chcemy.

Jeśli teraz zrestartujemy Apacza (`# /etc/init.d/httpd restart`), a następnie zbadamy go *nmapem*, to dowiemy się, że:

```
Interesting ports on gecew (127.0.0.1):
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd
```

Jak widać jest już lepiej – niepożądany gość nie dowie się, jakiej wersji serwera używamy – ale nadal nie udało nam się podszyć pod IIS. Na szczęście każda potrzeba powoduje, że w końcu znajdzie się ktoś, kto ją zaspokoi. Istnieje do Apache moduł *mod_security*, który wśród wielu funkcjonalności posiada możliwość zmiany banera. Aby to zrobić, trzeba do pliku konfiguracyjnego *httpd.conf* dopisać:

```
<IfModule mod_security.c>
  SecFilterEngine On
  SecServerSignature
        "Microsoft-IIS/6.0"
</IfModule>
```

Jeśli teraz zrestartujemy Apacza (`# /etc/init.d/httpd restart`), a następnie zbadamy go nmapem, to usłyszymy, że:

```
Interesting ports on gecew (127.0.0.1):
PORT   STATE SERVICE VERSION
82/tcp open  http    Microsoft IIS
                       webserver 6.0
```

Udało nam się więc po raz kolejny oszukać *nmapa*!

amap i vmap

Jeśli spróbujecie kiedyś poszukać w Internecie informacji o fingerprintingu warstwy aplikacji, natkniecie się na pewno na parę narzędzi: *amap* i *vmap*. *amap* rozpoznaje, jaka usługa działa na danym porcie (na przykład HTTP, FTP, SSH), *vmap* zaś usiłuje rozpoznać, jakie działa tam oprogramowanie. Aby więc przeprowadzić pełną próbę, musimy najpierw uruchomić *amapa*:

```
$ amap 127.0.0.1 80
```

Opcje są proste – podajemy adres IP badanego komputera i numer interesującego nas portu. Wynik działania *amapa* powinien być zrozumiały, w razie wątpliwości zajrzyj do Tabelki *Jak czytać output amapa*.

Kiedy już wiemy, jaka to usługa, możemy uruchomić vmapa, żeby rozpoznał wersję demona:

```
./vmap -P 80 127.0.0.1 http
```

vmap przeprowadza testy i mówi, jakie – jego zdaniem – oprogramowanie tam działa.

Jak działa vmap

Zasada działania *vmapa* jest prosta. Podczas badania serwera WWW wysyła on sześć nietypowych żądań mających na celu wykrycie charakterystycznych niuansów w implementacji protokołu HTTP. Żądania te są zdefiniowane w pliku *commands/http* i są to:

```
HEAD / HTTP/1.0
OPTIONS / HTTP/1.0
TRACE / HTTP/1.0
CONNECT / HTTP/1.0
GET bogus.http / HTTP/1.0
OPTIONS * / HTTP/1.0
```

Otrzymane odpowiedzi są porównywane z wyrażeniami regularnymi z plików konfiguracyjnych z katalogu *http/wo/server_name* (przykładowe wyrażenie regularne przedstawia Listing 3). Na tej podstawie *vmap* określa, z jakim serwerem ma do czynienia.

Przy korzystaniu z *vmapa* należy uważać – aktualna wersja *vmapa*, nadal będąca (w czasie powstawania tego artykułu) zamieszczona na stronie twórców, zawiera błąd. W katalogu *http/wo/server_name* znajduje się ukryty plik (jego nazwa zaczyna się od kropki) o nazwie *.Microsoft-ISS-6.0.swp*, zawierający binarne śmieci. Przez to bardzo często różne serwery są omyłkowo rozpoznawane jako *.Microsoft-ISS-6.0.swp*. Aby móc korzystać z vmapa, należy najpierw skasować ten plik.

Jak oszukać vmapa

Jak wspominaliśmy, *vmap* wykonuje tylko sześć prostych prób – w porównaniu z innymi narzędziami, które zaraz poznamy, to bardzo mało. W połączeniu ze starą bazą fingerptintów efekt jest taki, że rzadko udaje się przy jego pomocy prawidłowo rozpoznać wersję serwera. Dlatego nie będziemy nawet próbowali go oszukać – w praktyce i tak stosowanie go nie ma sensu.

Listing 6. *Nieprawidłowe żądania wysyłane przez hmapa – plik hmap.py, od wiersza 264 (fragment)*

```
def malformed_method_line(url):
  malformed_methods = ( 'GET', #0    TODO: repeat all these with HEAD and
                        OTHER
    'GET /',#1
    'GET / HTTP/999.99',
    'GET / HHTP/1.0',
    'GET / HTP/1.0',
    'GET / HHTP/999.99',
    #'GET / HHTP/1.0',
    'GET / hhtp/999.99',
    'GET / http/999.99',
    'GET / HTTP/Q.9',
    'GET / HTTP/9.Q',
    'GET / HTTP/Q.Q', #10
    'GET / HTTP/1.X',
    'GET / HTTP/1.10',
    'GET / HTTP/1.1.0',
    'GET / HTTP/1.2',
    'GET / HTTP/2.1',
    'GET / HTTP/1,0',
    #r'\GET / HTTP/1.0' or '\\GET / HTTP/1.0'
    #'GET / HTTP\1.0',
    #'GET / HTTP-1.0',
    #'GET / HTTP 1.0',
    'GET / HTTP/1.0X',
    'GET / HTTP/',
    #'get / http/1.0',
    #'qwerty / HTTP/1.0'
    #'GETX / HTTP/1.0'
    #' GET/HTTP/1.0',
    'GET/HTTP/1.0' ,
    'GET/ HTTP/1.0' ,#20
    'GET /HTTP/1.0' ,
    'GET/HTTP /1.0' ,
    'GET/HTTP/1 .0' ,
    'GET/HTTP/1. 0' ,
    'GET/HTTP/1.0 ' ,
    'GET / HTTP /1.0', #etc....
    'HEAD /.\\ HTTP/1.0', # indicates windows??
    'HEAD /asdfasdfasdfasdfasdf/../ HTTP/1.0',
    'HEAD /asdfasdfasdfasdfasdf/.. HTTP/1.0',
    'HEAD /./../../../../../../../../../../ HTTP/1.0',#30
    'HEAD /./../../../../qwerty/../../../../../../ HTTP/1.0',
    #'HEAD ../ HTTP/1.0',
    'HEAD /.. HTTP/1.0',
    'HEAD /../ HTTP/1.0',
    'HEAD /../../../../../ HTTP/1.0',
    'HEAD .. HTTP/1.0',
    #'HEAD . HTTP/1.0',
    'HEAD\t/\tHTTP/1.0',
    'HEAD //////////// HTTP/1.0',
    'Head / HTTP/1.0',
    '\nHEAD / HTTP/1.0',
    ' \nHEAD / HTTP/1.0',#40
    ' HEAD / HTTP/1.0',
(...)
```

Specjalizowane narzędzia do rozpoznawania wersji serwera WWW

Dotąd przyglądaliśmy się narzędziom służącym do ogólnego fingerprintingu poziomu aplikacji. Zwykle jednal bardziej sprytne są specjalizowane narzędzia do rozpoznawania wersji serwera WWW – poznajmy kilka z nich.

netcraft

Pierwsze z tych narzędzi już widzieliśmy: to *www.netcraft.com* –strona, na której wystarczy wpisać nazwę domeny, a dowiemy się, jaki serwer tam działa. Niestety, *netcraft* rozpoznaje serwer na podstawie banera. Z tego powodu bardzo łatwo jest go oszukać (wystarczy podmienić baner – co, jak widzieliśmy, jest całkiem proste) i nie można mu ufać.

httprint

Naprawdę poważnym narzędziem do rozpoznawania wersji serwera WWW jest *httprint*. Jest to narzędzie darmowe, ale o zamkniętym kodzie źródłowym, stworzone przez firmę *Net Square*. Używa ono sprytnych metod, analizuje różnice w implementacji protokołu HTTP, przez co nie da się go oszukać przez zwykłą podmianę banera. Używamy go tak:

```
$ ./httprint -h 127.0.0.1:82 -s
 signatures.txt -P0
```

Jak widać podajemy IP badanego hosta i numer portu. Opcją `-s` wskazujemy położenie pliku zawierającego sygnatury (jest on dostarczony razem z programem, nazywa się *signatures.txt*). Możemy – jak w powyższym przykładzie – dodać opcję `-P0`, co spowoduje, że *httprint* nie będzie zaczynał od spingowania badanego serwera.

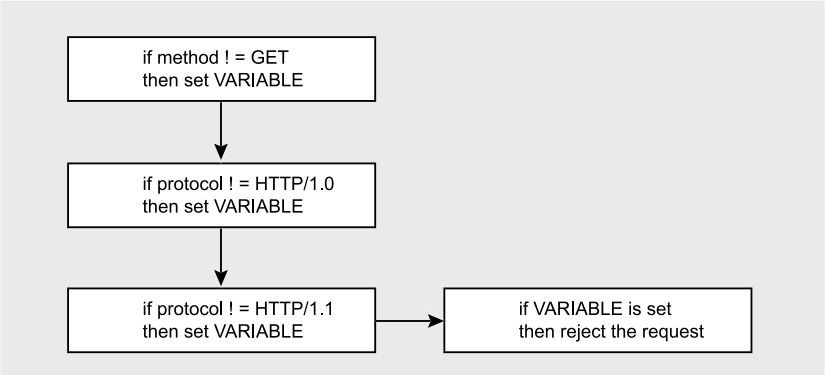
Po uruchomieniu *httprint* zaczyna przeprowadzać testy (wysyła kilkaset żądań – porównajcie to z sześcioma żądaniami *vmapa*!). Wyniki prób są porównywane ze znanymi narzędziu wynikami prób dla znanych serwerów, na ich podstawie podliczana jest punktacja (to znaczy, że najbardziej pasujący serwer dostaje najwięcej punktów) oraz poziom zaufania. Poziom zaufania mówi nam, na ile możemy zaufać otrzymanemu wynikowi – bywa bowiem, że choć badany serwer jest bardziej podobny do *Apacza 2.0.x* niż do czegokolwiek innego (a więc jako wynik zostanie podane *Apache 2.0.x*), to jednak wyniki prób będą o tyle inne od czegokolwiek znanego, że nie powinniśmy zanadto sugerować się otrzymanym wynikiem.

Listing 7. *Próba skonfigurowania modułu mod_security tak, by odrzucał nieprawidłowe żądania*

```
<IfModule mod_security.c>
  SecFilterEngine On
  SecFilterDebugLog logs/modsec_debug_log
  SecFilterDebugLevel 4
  SecFilterDefaultAction "deny,log,status:406"
  SecFilterSelective REQUEST_METHOD "!^(GET|HEAD|PUT)$"
  SecFilterSelective SERVER_PROTOCOL "!(HTTP/1.0|HTTP/1.1)"
</IfModule>
```

Listing 8. *Konfiguracja mod_setenvif*

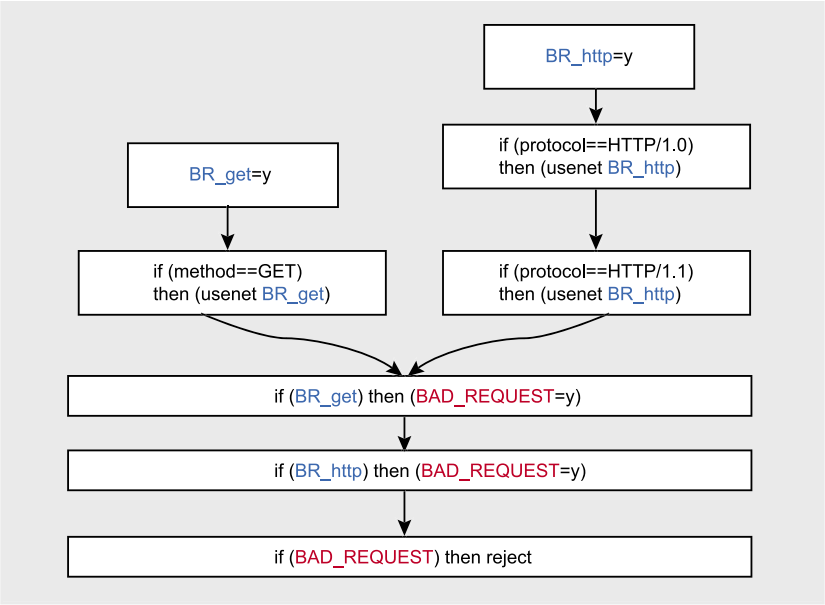
```
SetEnvIf Request_Method . BR_http=y
SetEnvIf Request_Method . BR_get=y
SetEnvIf Request_Protocol HTTP\/1\.0$ !BR_http
SetEnvIf Request_Protocol HTTP\/1\.1$ !BR_http
SetEnvIf Request_Method GET !BR_get
SetEnvIf BR_http y BadRequest=y
SetEnvIf BR_get y BadRequest=y
<Directory />
    Options FollowSymLinks
    AllowOverride None
    Order Deny,Allow
    Deny from env=BadRequest
</Directory>
```



Rysunek 10. *Pomysł na skonfigurowanie mod_setenvif*

Po zakończeniu prób *httprint* wypisuje raport na standardowe wyjście, zapisuje też w bieżącym katalogu ładny raport w HTML-u (plik *report.html*). Raport wypisany na *stdout* jest nieco bardziej szczegółowy, więc najpierw przyjrzyjmy się jemu – patrz Listing 4.

Jak widać raport stwierdza, że zbadano serwer *http://127.0.0.1:80*/, przedstawiający się jako *Microsoft-IIS/6.0*. Dalej następuje fingerprint tego serwera. Potem czytamy, że serwer w banerze przedstawił się jako *Microsoft-IIS/6.0,* ale z testów wynika, że mamy do czynienia z serwerem *Apache/2.0.x*. Ten wynik (serwer to *Apache/2.0.x*) uzyskał 140 punktów, poziom zaufania to 84.34. Poniżej następuje lista innych wyników, które są mniej wiarygodne, ale również otrzymały dużo punktów. Widzimy tu, że choć najprawdopodobniej badany serwer to Apache, to może to być również – choć z dużo mniejszym prawdopodobieństwem – TUX 2.0, Agranat-EmWeb itp. Raport w postaci HTML – Rysunek 6 – zawiera podobne informacje, nie ma w nim tylko listy mniej praw-



Rysunek 11. *Kolejny pomysł na skonfigurowanie mod_setenvif*

dopodobnych wyników. Jak widać *httprint* nie jest tak naiwny jak *nmap* – nie uwierzył bezkrytycznie w podmieniony baner...

hmap

Narzędziem pod wieloma względami podobnym do *httprinta* jest *hmap* – narzędzie, które powstało jako praca dyplomowa Dustina Lee. W przeciwieństwie do *httprinta*, *hmap* jest to open source. Jego zasada działania jest bardzo podobna do *httprinta,* więc kiedy będziemy próbować oszukać te narzędzia, testy będziemy przeprowadzać na obu jednocześnie (co zadziała na *hmapa*, zadziała też zwykle na *httprinta*), *hmapa* uruchamiamy tak:

```
$ python hmap.py -v -c 10
      http://www.somehost.com:80
```

Opcja `-v` nakazuje wypisywanie większej ilości komunikatów (chcemy dokładniej wiedzieć, co robi program), opcja `-c 10` powoduje, że wypisanych zostanie dziesięć najbardziej prawdopodobnych wyników.

Po uruchomieniu *hmap* zaczyna wykonywać próby (już na oko widać, że jest wolniejszy od *httprinta*). Następnie porównuje wyniki z bazą fingerprintów, po czym wypisuje raport (Listing 5). Jak widać, również *hmap* nie miał problemów z rozpoznaniem naszego Apacza ukrytego za banerem IIS-a.

Spróbujmy oszukać hmapa

Aby oszukać *hmapa*, musimy przyjrzeć się dokładniej jak działa. Aby odgadnąć, z jakim serwerem mamy do czynienia, *hmap* porównuje wyniki przeprowadzonych testów ze znanymi sobie wynikami dla różnych serwerach zapisanymi w plikach umieszczonych w *katalogu known.servers* (plik o nazwie *apache.1.3.12.win32* zawiera wyniki dla Apache 1.3.12 na win32, *apache.1.3.14.win32* – dla Apache w wersji 1.3.14 itp). Przykładową zawartość takiego pliku przedstawia Listing 6.

Jak widać, plik *apache.1.3.12.win32* zawiera informacje o ponad stukilkudziesięciu próbach. Z tego ponad sto to próby o nazwie *MALFORMED_000 ... MALFORMED_104*. Są to – jak wskazuje nazwa – próby polegające na wysłaniu do serwera nieprawidłowego (niezgodnego ze standardem) żądania. Jak dokładniej wyglądają takie nieprawidłowe żądania, zobaczymy w źródłach hmapa – patrz plik *hmap*, od linijki 264 (Listing 6).

To dobra wieść. Skoro większość prób wykonywanych przez *hmapa* polega na wysyłaniu nieprawidłowych (ang. *malformed*) żądań, to dobrym pomysłem będzie ich zablokowanie po stronie serwera. Nie ma żadnego powodu, aby obsługiwać takie żądania (zwykły klient ich nie wyśle), a zablokowanie ich może pomieszać szyki *hmapowi*. A więc zróbmy to i zobaczmy, czy w ten sposób uda nam się oszukać *hmapa* (a przy okazji może również *httprinta*).

Zablokować nieprawidłowe żądania można na przynajmniej kilka sposobów:

- możemy spróbować odpowiednio skonfigurować serwer,

Listing 9. *hmap próbuje rozpoznać przekonfigurowanego Apacza*

```
                                        matches : mismatches : unknowns
Apache/1.3.26_3 (FreeBSD 4.6.2-RELEASE)     65 :  47 :  11
Apache/1.3.26 (Solaris 8)                   65 :  47 :  11
Apache/1.3.27 (Red Hat 8.0)                 65 :  47 :  11
Apache 1.3.27 (FreeBSD 4.7)                 65 :  48 :  10
Apache/2.0.44 (Win32)                       64 :  48 :  11
Apache/1.3.23 (RedHat Linux 7.3)            64 :  48 :  11
Apache/1.3.27 (FreeBSD 5.0)                 64 :  48 :  11
Apache/1.3.27 (Mac 10.2.4)                  64 :  49 :  10
Apache/2.0.40 (Red Hat 8.0)                 63 :  48 :  12
Apache/1.3.27 (Mac 10.1.5)                  63 :  49 :  11
Apache/1.3.12 (Win32)                       63 :  50 :  10
Apache/1.3.14 (Win32)                       63 :  50 :  10
Apache/1.3.17 (Win32)                       63 :  50 :  10
Apache/1.3.22 (Win32)                       63 :  50 :  10
IBM_HTTP_Server/2.0.42 (Win32)              62 :  49 :  12
Apache/1.3.9 (Win32)                        62 :  50 :  11
HP-Web-Server-2.00.1454 (Solaris 8)         32 :  77 :  14
thttpd/2.23beta1 26may2002 (FreeBSD 4.6-    32 :  77 :  14
NCSA/1.3 (Ultrix 4.4)                       32 :  79 :  12
thttpd 2.23beta1 26may2002 (RedHat 7.3)     28 :  80 :  15
```

Listing 10. *httprint próbuje rozpoznać przekonfigurowanego Apacza*

```
Finger Printing on http://127.0.0.1:82/
Finger Printing Completed on http://127.0.0.1:82/
-------------------------------------------------
Host: 127.0.0.1
Derived Signature:
Microsoft-IIS/6.0
811C9DC5E2CE6920811C9DC5811C9DC5050C5D32505FCFE84276E4BB811C9DC5
0D7645B5811C9DC52A200B4CCD37187C811C9DC5811C9DC5811C9DC5811C9DC5
E2CE6920E2CE6920E2CE6920811C9DC5E2CE6927811C9DC5E2CE6925811C9DC5
E2CE6920E2CE69202A200B4CE2CE6920E2CE69206ED3C295E2CE6920E2CE6923
E2CE6923E2CE6920811C9DC5E2CE6927E2CE6923

Banner Reported: Microsoft-IIS/6.0
Banner Deduced: Orion/2.0x
Score: 75
Confidence: 45.18
------------------------
Scores:
Orion/2.0x: 75 45.18
AssureLogic/2.0: 73 40.87
Apache/2.0.x: 72 38.82
Apache/1.3.27: 72 38.82
Apache/1.3.26: 72 38.82
Apache/1.3.[4-24]: 72 38.82
Apache/1.3.[1-3]: 67 29.55
TUX/2.0 (Linux): 62 21.82
Apache-Tomcat/4.1.29: 60 19.13
Microsoft-IIS/6.0: 59 17.87
Agranat-EmWeb: 59 17.87
Netscape-Enterprise/3.5.1G: 57 15.50
Apache/1.2.6: 57 15.50
Com21 Cable Modem: 49  7.98
thttpd: 49  7.98
Oracle Servlet Engine: 49  7.98
```

- możemy napisać własny moduł Apacza, który będzie blokował takie żądania,
- możemy napisać specjalne proxy, które będzie blokować takie żądania, a pozostałe przepuszczać,
- możemy użyć systemu zapobiegania włamaniom (IPS).

Każde z podejść ma swoje wady i zalety. Zabawy z konfigurowaniem Apacza są najprostsze, ale ten sposób ma (jak zobaczymy) swoje ograniczenia – nie wszystko da się zrobić konfiguracją. Pisanie własnego modułu Apacza to poważna sprawa, ponadto ryzykujemy, że i tu – jak w przypadku konfigurowania serwera – natrafimy w końcu na ograniczenia (wynikłe ze sposobu, w jaki Apache traktuje moduły), które spowodują, że nie wszystkie nieprawidłowe żądania będziemy mogli zablokować. Pisanie własnego proxy ma tę zaletę, że możemy zrobić z żądaniem dokładnie co chcemy, jednak sposób ten i tak nie nada się raczej do stosowania w środowisku produkcyjnym – prawdopodobnie takie proxy pogorszy wydajność serwera. Bardzo dobrym rozwiązaniem wydaje się zastosowanie IPS-a - jedyną wadą jest to, że trzeba mieć go pod ręką.

Ponieważ zależy nam na sposobie, który można wypróbować szybko, spróbujmy po prostu skonfigurować serwer tak, by blokował wszystkie żądania oprócz tych, dla których użyta metoda to *GET a protokół to HTTP/1.0 lub HTTP/1.1.*

Jak tego nie zrobimy

Jest kilka ślepych uliczek, w które można wpaść próbując odpowiednią konfiguracją zmusić Apacza do odrzucania nieprawidłowych żądań. Można by na przykład spróbować skorzystać z modułu mod_security – ma on możliwość odrzucania żądań spełniających zadany warunek. Można by więc użyć konfiguracji przedstawionej na Listingu 7 – powoduje ona, że odrzucane są wszystkie żądania typu innego niż *GET, HEAD* i *PUT* i używające protokołu innego niż *HTTP/1.0 i HTTP/1.1*.

Niestety, jeśli spróbujemy tej konfiguracji, okaże się, że nieprawidłowe żądania nie są odrzucane. To dlatego, że moduł *mod_security* dostaje żądania za późno, przez co te *malformed* są obsłużone (odmownie) przez Apacza, zanim trafią do *mod_security.*

Konfiguracja, która działa

Dobrym natomiast pomysłem będzie użycie modułu *mod_setenvif* . Pozwala on ustawić jakąś zmienną środowiskową w zależności od pewnego warunku. Przykładowo, poniższy wpis w pliku konfiguracyjnym:

```
SetEnvIf Request_Method GET Variable=y
```

oznaczać będzie: *jeśli metoda użyta w żądaniu to GET, ustaw zmienną środowiskową Variable na wartość y.*

Następnie – przez odpowiedni wpis w *httpd.conf* – możemy nakazać, aby żądanie było odrzucane, jeśli dana zmienna środowiskowa jest ustawiona. Na przykład w ten sposób:

```
<Directory />
    (...)
    Deny from env=Variable
</Directory>
```

Możemy więc spróbować skonfigurować serwer tak, aby po otrzymaniu żądania (patrz Rysunek 7):

- sprawdzane było, czy metoda użyta w żądaniu to *GET*, jeśli nie – ustawiamy pewną zmienną środowiskową,
- sprawdzane było, czy protokół użyty w żądaniu to *HTTP/1.0* lub *HTTP/1.1*, jeśli nie – ustawiamy tę samą zmienną środowiskową,

- jeśli zmienna środowiskowa jest ustawiona, żądanie odrzucamy.

Niestety, jeśli schemat z Rysunku 7 spróbujemy zapisać w pliku konfiguracyjnym, napotkamy na problem. Jak widzieliśmy, możemy skonfigurować *mod_setenvif* tak, aby pewna zmienna środowiskowa była ustawiana, jeśli użyta metoda to *GET*:

```
SetEnvIf Request_Method GET Variable=y
```

Niestety, składnia nie uwzględnia takiej możliwości, aby zmienna środowiskowa była ustawiana, jeśli użyta metoda to nie *GET*:

```
SetEnvIf Request_Method !GET Variable=y
```

Możemy obejść ten problem. Składnia konfiguracji *mod_security* pozwala nie tylko ustawiać (ang. *set*) zmienne środowiskowe, ale również je wyłączać (ang. *unset*). Zamiast więc ustawiać zmienną, jeśli metoda jest inna niż *GET* (czego zrobić nie możemy), wystarczy zrobić tak:

- ustawiamy pewną zmienną
- jeśli metoda to GET, wyłączamy tę zmienną

Ostatecznie więc nasza konfiguracja będzie zbudowana tak, jak to przedstawia Rysunek 8. Jak widać, w konfiguracji biorą udział trzy zmienne. Zmienna *BR_get* oznacza żądanie inne niż *GET* – jest ona początkowo ustawiana na wartość *y*, jeśli okaże się, że użyta metoda to *GET*, zmienna jest wyłączana. Zmienna *BR_http* oznacza żądanie inne niż *HTTP/1.0* lub *HTTP/1.1*, jest ona ustawiana w podobny sposób. Jeśli któraś z tych zmiennych jest ustawiana, włączana jest zmienna *BAD_REQUEST*. Na koniec jeśli *BAD_REQUEST* jest ustawiona, żądanie jest odrzucone.

Schemat z Rysunku 8 zapisany w postaci gotowej do wklejenia do *httpd.conf* przedstawiony jest na Listingu 8. Po wklejeniu go i zrestartowaniu serwera WWW możemy spróbować, jak teraz rozpozna go hmap i httprint.

Najpierw sprawdźmy, co o tak skonfigurowanym serwerze powie *hmap*:

```
$ python hmap.py -v -c 20 http://
                              127.0.0.1:80
```

Wynik działania programu przedstawia Listing 9. Jak widać, *hmap* jest znacznie mniej pewien otrzymanych wyników – porównajcie ten wynik z wynikiem z Listingu 5. Jak widać, o ile poprzednio aż 110 prób wskazało na będącego na czele listy Apacza, a tylko 5 prób nie pasowało i 8 dało wynik nierozstrzygnięty, to teraz tylko 65 prób dało wynik pasujący, 47 – niepasujący i 11 nierozstrzygnięty. Niestety – nasz serwer nadal został rozpoznany jako Apache, choć wersja nie została już rozpoznana prawidłowo. Nasz cel – ukrycie tożsamości serwera – osiągnęliśmy więc tylko połowicznie. Zobaczmy, czy lepiej pójdzie nam z *httprintem*:

```
$ ./httprint -h 127.0.0.1:82 -s
 signatures.txt -P0
```

Wynik działania *httprinta* przedstawia Listing 10. Jak widać, udało nam się go oszukać! Za najbardziej prawdopodobny serwer uznany został Orion/2.0x, na drugim miejscu znalazł się AssureLogic/2.0 i dopiero na trzecim – Apache/2.0.x.

Wnioski

Podsumujmy zdobytą wiedzę. Jak widać, jeśli występujemy w roli intruza i próbujemy rozpoznać, z jakim serwerem mamy do czynienia, nie możemy równym zaufaniem obdarzać wszystkich narzędzi. Oprócz *vmapa* (którego nie bierzemy pod uwagę – jest całkiem niewiarygodny) możemy wyróżnić dwie grupy narzędzi: te, które dają się nabrać na podmieniony baner (*nmap* i *netcraft*) i te, które się na to nie nabierają. Oczywiście w stosunku do wyników podawanych przez te pierwsze powinniśmy być nieufni – jak widzieliśmy, podmiana banera jest bardzo prosta.

Większym zaufaniem możemy obdarzać te programy (*hmap* i *httprint*), które nie ufają bezkrytycznie podmienionemu banerowi. Jak widzieliśmy, i je da się oszukać (choć *hmap* okazał się nieco odporniejszy), jednak wymaga to dużo większego zachodu i można sądzić, że mało kto będzie stosował aż tak wymyślne metody maskowania się.

Jeśli z kolei patrzymy z punktu widzenia ofiary potencjalnego ataku, to warto najpierw zastanowić się, czy na pewno chcemy ukrywać przed światem, jakiego serwera WWW używamy. Niektórzy uważają, że bezpieczeństwo należy osiągać przez łatanie dziur, a nie ich ukrywanie – a więc podmiana banera nie poprawi naszego bezpieczeństwa, a tylko da nam złudne poczucie spokoju. Tym bardziej, że ani robaki, ani script kiddies nie będą prawdopodobnie sprawdzać wersji naszego serwera, tylko po prostu spróbują użyć exploita (zadziała – dobrze, nie zadziała – próbujemy zaatakować kolejny serwer). Z drugiej jednak strony, jeśli będziemy czuli się lepiej wiedząc, że byle kto nie może w kilka sekund dowiedzieć się, której wersji Apacza używamy, powinniśmy przynajmniej podmienić baner (albo chociaż uczynić go mniej szczegółowym) – jak widzieliśmy, nawet tak popularne i poważane narzędzie jak *nmap* da się nabrać na ten prosty trick.

Zaś najambitniejsi czytelnicy mogą pokusić się o poprawienie nmapa tak, by przeprowadzał dokładne testy nawet wtedy, kiedy baner badanej usługi wygląda znajomo. ●

O autorze:
Piotr Sobolewski (www.piotrsobolewski.w.pl), z zawodu programista, interesuje się nietypowym bezpieczeństwem systemów informatycznych i wykorzystywaniem nowych technologii.