helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

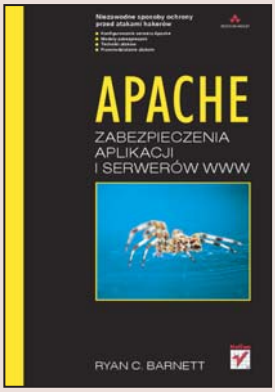
TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

Apache. Zabezpieczenia aplikacji i serwerów WWW

Autor: Ryan C. Barnett
Tłumaczenie: Marek Pałczyńsk
ISBN: 83-246-0505-3
Tytuł oryginału: Preventing Web Attacks with Apache
Format: B5, stron: 544



Internet to nie tylko niezmierzone źródło informacji. To także zagrożenie dla serwerów WWW, aplikacji internetowych i baz danych, które codziennie są atakowane przez komputerowych przestępców, korzystających z dziesiątek technik. Publikowane regularnie raporty o cyberprzestępczości są zatrważające. Liczba ataków na serwery internetowe wzrasta corocznie średnio o 30%. Wśród atakowanych serwerów przeważają te, na których utrzymywane są witryny WWW i aplikacje. Według raportu firmy Symantec, „aplikacje WWW są popularnymi celami ataków z uwagi na ich rozpowszechnienie i fakt, że pozwalają włamywaczom na pominięcie tradycyjnych mechanizmów zabezpieczających, takich jak firewalle". W tym samym raporcie można również przeczytać, że prawie 50% luk w zabezpieczeniach serwerów wiąże się właśnie z aplikacjami WWW.

W książce „Apache. Zabezpieczenia aplikacji i serwerów WWW" znajdziesz informacje o tym, w jaki sposób uchronić przed atakami hakerów aplikacje i witryny WWW kontrolowane przez najpopularniejszy obecnie serwer WWW — Apache. Przeczytasz o tym, jak poprawnie zainstalować i skonfigurować Apache'a i w jaki sposób uruchomić w nim moduły zabezpieczeń. Poznasz techniki ataków hakerskich i dowiesz się, jak im zapobiegać. Znajdziesz sposoby testowania zabezpieczeń swojego serwera za pomocą odpowiednich narzędzi. Nauczysz się także wykrywać próby ataków i reagować na nie odpowiednio wcześnie.

- Czynniki wpływające na bezpieczeństwo sieci
- Instalacja serwera Apache
- Plik httpd.conf — konfiguracja Apache'a
- Instalowanie i konfigurowanie modułów zabezpieczeń
- Klasyfikacja zagrożeń sieciowych WASC
- Metody zabezpieczania aplikacji sieciowych
- Ochrona przed atakami
- Tworzenie serwerów-pułapek

Dzięki tej książce każdy administrator będzie mógł spokojnie spać

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Rozdział 5.

Najważniejsze moduły zabezpieczeń

W poprzednim rozdziale zostały przedstawione dyrektywy Apache, które mają wpływ na bezpieczeństwo serwera. Ten rozdział jest poświęcony dodatkowym modułom, przeznaczonym do zabezpieczenia aplikacji. Mogą one odgrywać istotną rolę w ogólnym zabezpieczeniu serwera Apache, a ich złożoność i zakres działań są znacznie większe niż opisywanych wcześniej dyrektyw. W treści niniejszego rozdziału zostały zamieszczone opisy zarówno procedur instalacyjnych, jak i ogólnych funkcji każdego z modułów. Nie zostały natomiast przedstawione wszystkie dyrektywy modułów, ponieważ będą one omawiane w dalszej części książki, w rozdziałach dotyczących konkretnych sposobów zapobiegania atakom sieciowym.

Protokół SSL

Omawianie najważniejszych modułów zabezpieczeń musi się rozpocząć od prezentacji modułu `mod_ssl` z pewnego szczególnego względu. Istnieje bowiem wielka rozbieżność między jego rzeczywistymi funkcjami a oczekiwaniami wielu użytkowników. Gdyby trzeba było podać motto tego podrozdziału, mogłoby nim być zdanie „Szyfrowanie nie jest równoznaczne z bezpieczeństwem". Chociaż protokół SSL odgrywa bardzo ważną rolę w zabezpieczaniu komunikacji sieciowej, nie jest panaceum na wszystkie problemy związane z bezpieczeństwem. O prawdziwości tego twierdzenia przekonał się autor książki podczas dokonywania zakupów internetowych. Szczegółowy opis zdarzenia został zamieszczony w części pt. „Jesteśmy bezpieczni, ponieważ stosujemy SSL — błędne rozumowanie".

Dlaczego należy korzystać z protokołu SSL?

Stosowanie protokołu SSL pozwala na realizację trzech zadań.

- **Szyfrowanie danych przesyłanych między serwerem i klientem (tj. przeglądarką).** Cecha ta jest najczęściej wykorzystywaną funkcją protokołu SSL. Po zainstalowaniu serwera WWW obsługującego mechanizm SSL administratorzy nie mają większych problemów z wygenerowaniem certyfikatów potrzebnych w komunikacji SSL. Rozwiązanie to natomiast zapewnia poufność wymiany informacji.
- **Weryfikacja tożsamości serwera.** Przeglądarki internetowe analizują certyfikaty SSL przesyłane przez serwer WWW, sprawdzając, czy zostały one podpisane przez zewnętrzne urzędy certyfikacji (ang. *Certificate Authority* — CA), takie jak VeriSign lub Entrust. Jeżeli certyfikat nie jest podpisany, przeglądarka wyświetla okno dialogowe z informacją o błędzie. Prowadzenie serwisu handlu elektronicznego niemal zawsze wiąże się z koniecznością uzyskania podpisanego certyfikatu. Certyfikat ten uwiarygodnia witrynę i eliminuje niedogodności związane z wyświetlaniem komunikatów o błędach.
- **Weryfikacja tożsamości klienta.** Wiadomo, że klient może przeglądać certyfikat SSL serwera. Podobnie istnieje możliwość uwierzytelnienia klienta, o ile w jego przeglądarce został zainstalowany prywatny certyfikat. Mimo iż rozwiązanie to jest znane od wielu lat i ma więcej zalet w porównaniu ze standardowymi mechanizmami uwierzytelniania (wymagającymi podania nazwy użytkownika i hasła), nie jest stosowane na szerszą skalę ze względu na koszty i trudności w zarządzaniu certyfikatami.

Jesteśmy bezpieczni, ponieważ stosujemy SSL — błędne rozumowanie

W lutym 2004 roku zdecydowałem się na zakup przez internet zestawu ziół, które podgrzane w kuchence mikrofalowej pomagają zwalczyć ból mięśni. Gdy wyświetliłem stronę producenta, od razu zostałem przywitany komunikatem „Ta witryna jest bezpieczna! Używamy 128-bitowego szyfrowania SSL". Miał on rozproszyć ewentualne wątpliwości użytkowników. Podczas finalizowania zamówienia postanowiłem sprawdzić parametry SSL połączenia. Kliknąłem dwukrotnie ikonę „kłódki" w prawym dolnym rogu okna przeglądarki i upewniłem się, że nazwa domenowa certyfikatu SSL odpowiada nazwie zawartej w adresie URL. Ponadto certyfikat został podpisany przez zaufany urząd certyfikacji, jakim jest VeriSign, i nadal był ważny. Ponieważ nic nie budziło moich podejrzeń, przeszedłem do strony płatności i wpisałem dane karty kredytowej. Nacisnąłem przycisk przesłania formularza i zobaczyłem komunikat, który wywołał u mnie skurcz żołądka. Był to komunikat przesłany listem elektronicznym. Jego treść znajduje się poniżej. Zmieniłem jedynie dane firmowe i informacje o karcie kredytowej.

Otrzymałem e-mail o następującej treści:

```
Od:nazwa_firmy@aol.com
Do: RCBarnett@email.com
Temat:ONLINE HERBPACK!!!
nazwisko: Ryan Barnett
adres: Nieznana 1234
miejscowość: Gdzieś
stan: Stan
```

```
kod pocztowy: 12345
telefon#:
rodzaj karty : American Express
nazwisko na karcie: Ryan Barnett
numer karty: 123456789012345
data ważności: 11/05
liczba zestawów podstawowych:
liczba poduszek na oczy:
liczba usztywniaczy karku: 1
liczba pasów: 1
liczba zestawów jumbo:
liczba ogrzewaczy stóp: 1
liczba opasek na kolano:
liczba opasek na nadgarstek:
liczba zestawów klawiaturowych:
liczba opasek barkowych:
liczba kapeluszy czarnych:        liczba kapelusz szarych:
liczba kapeluszy niebieskich:        liczba kapeluszy czerwonych:
liczba kapeluszy beżowych:           liczba kapeluszy pomarańczowych:
czy chcesz przesłać paczkę do przyjaciela:
nazwisko:
jego adres:
jego miejscowość:
jego stan:
jego kod pocztowy:

cgiemail 1.6
```

Nie mogłem uwierzyć własnym oczom. Numer mojej karty kredytowej został przesłany otwartym tekstem na konto pocztowe w AOL. Jak to możliwe? Administratorzy witryny byli najwyraźniej dostatecznie zaznajomieni z technologią, żeby wiedzieć, że należy szyfrować dane klienta przesyłane do serwera. Dlaczego zatem nie wykazali się taką samą inteligencją, projektując końcową fazę procedury?

Uznałem, że to zwykła pomyłka. Przeczytałem reklamę zamieszczoną u dołu ekranu, która informowała, że do przetwarzania zamówień wykorzystywane jest oprogramowanie *cgiemail 1.6*. Otworzyłem stronę Google i zacząłem szukać informacji na temat tego produktu. W serwisie Google znalazłem odsyłacz do poradnika administratora programu cgiemail. Szybko przejrzałem jego treść i odszukałem interesujący rozdział pt. „Jakie zabezpieczenia zostały zaimplementowane?”.

> *Istnieje zagrożenie przechwycenia danych przesyłanych przez sieć z przeglądarki do serwera i z serwera do przeglądarki. Do podsłuchu informacji może dojść w każdym punkcie trasy między przeglądarką a serwerem.*
>
> *Ryzyko: Aplikacja cgiemail, jak każdy program przekazujący dane z formularza na pocztę, jest podatna na podsłuch w dowolnym punkcie trasy między serwerem WWW a odbiorcą poczty elektronicznej. Z uwagi na brak mechanizmów szyfrujących nie zaleca się przekazywania poufnych informacji, na przykład numerów kart kredytowych.*

Tak jak przypuszczałem. Resztę dnia spędziłem, próbując poinformować wystawcę karty kredytowej o możliwym ujawnieniu danych karty oraz obserwując saldo rachunku bankowego. Przesłałem również list do firmy prowadzącej sklep internetowy (na adres AOL). Opisałem w nim problemy związane z bezpieczeństwem operacji. Całą sprawę można podsumować jednym zdaniem. Wykorzystanie protokołu SSL nie stanowi o *bezpieczeństwie witryny*.

Jak działa mechanizm SSL?

Mechanizm SSL działa na pograniczu warstwy trzeciej i czwartej modelu TCP/IP (tabela 5.1). Zapewnia funkcje szyfrowania dla wielu usług, wśród których jest również usługa HTTP. W praktyce algorytm SSL może być zaimplementowany w aplikacji osłonowej, tworzącej tzw. tunel SSL, który zapewnia szyfrowany kanał transmisyjny dla dowolnej usługi TCP/IP.

Tabela 5.1. *Sieciowy model odniesienia TCP/IP*

Warstwa aplikacji (warstwa 4.)	HTTP
SSL	Szyfrowanie wszystkich danych warstwy 4.
Warstwa transportowa (warstwa 3.)	TCP
Warstwa internetowa (warstwa 2.)	IP
Warstwa sieciowa (warstwa 1.)	Ethernet

W systemie Solaris jest dostępny monitor sieci *snoop*, za pomocą którego można wizualnie wyróżnić poszczególne warstwy komunikacji sieciowej w standardowej transakcji HTTP. W przykładzie listingu wynikowego programu *snoop* są widoczne dane wszystkich warstw, choć ich kolejność jest odwrotna w porównaniu z wykazem przedstawionym w tabeli 5.1. Wynika to z faktu demultipleksacji pakietu.

```
ETHER: ----- Ether Header -----
ETHER:
ETHER: Packet 4 arrived at 10:47:44.50
ETHER: Packet size = 411 bytes
ETHER: Destination = 0:3:ba:8:1d:d,
ETHER: Source = 0:8:e2:42:b1:fc,
ETHER: Ethertype = 0800 (IP)
ETHER:
IP: ----- IP Header -----
IP:
IP: Version = 4
IP: Header length = 20 bytes
IP: Type of service = 0x00
IP: xxx. .... = 0 (precedence)
IP: ...0 .... = normal delay
IP: .... 0... = normal throughput
IP: .... .0.. = normal reliability
IP: Total length = 397 bytes
IP: Identification = 5914
IP: Flags = 0x4
IP: .1.. .... = do not fragment
IP: ..0. .... = last fragment
IP: Fragment offset = 0 bytes
IP: Time to live = 125 seconds/hops
IP: Protocol = 6 (TCP)
IP: Header checksum = 1b05
IP: Source address = 192.168.1.100, 192.168.1.100
IP: Destination address = 192.168.1.101, 192.168.1.101
IP: No options
IP:
```

```
TCP: ----- TCP Header -----
TCP:
TCP: Source port = 1260
TCP: Destination port = 80 (HTTP)
TCP: Sequence number = 2934191179
TCP: Acknowledgement number = 3769986061
TCP: Data offset = 20 bytes
TCP: Flags = 0x18
TCP: ..0. .... = No urgent pointer
TCP: ...1 .... = Acknowledgement
TCP: .... 1... = Push
TCP: .... .0.. = No reset
TCP: .... ..0. = No Syn
TCP: .... ...0 = No Fin
TCP: Window = 65520
TCP: Checksum = 0x4bb9
TCP: Urgent pointer = 0
TCP: No options
TCP:
HTTP: ----- HyperText Transfer Protocol -----
HTTP:
HTTP: GET / HTTP/1.1
HTTP: Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword,
application/x-shockwave-flash, */*
HTTP: Accept-Language: en-us
HTTP: Accept-Encoding: gzip, deflate
HTTP: User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
HTTP: Host: 192.168.1.101
HTTP: Connection: Keep-Alive
```

Program *snoop* rejestruje dane wszystkich czterech warstw: sieciowej, IP, TCP i HTTP. W sekcji protokołu HTTP jest widoczne całe żądanie przesłane przez jednostkę kliencką, włącznie ze wszystkimi nagłówkami klienckimi. Dla porównania drugi listing wynikowy programu *snoop* został sporządzony podczas przekazywania żądania do serwera WWW z włączoną opcją SSL. Dane czwartej warstwy nie są widoczne, ponieważ aplikacja *snoop* nie mogła rozszyfrować informacji.

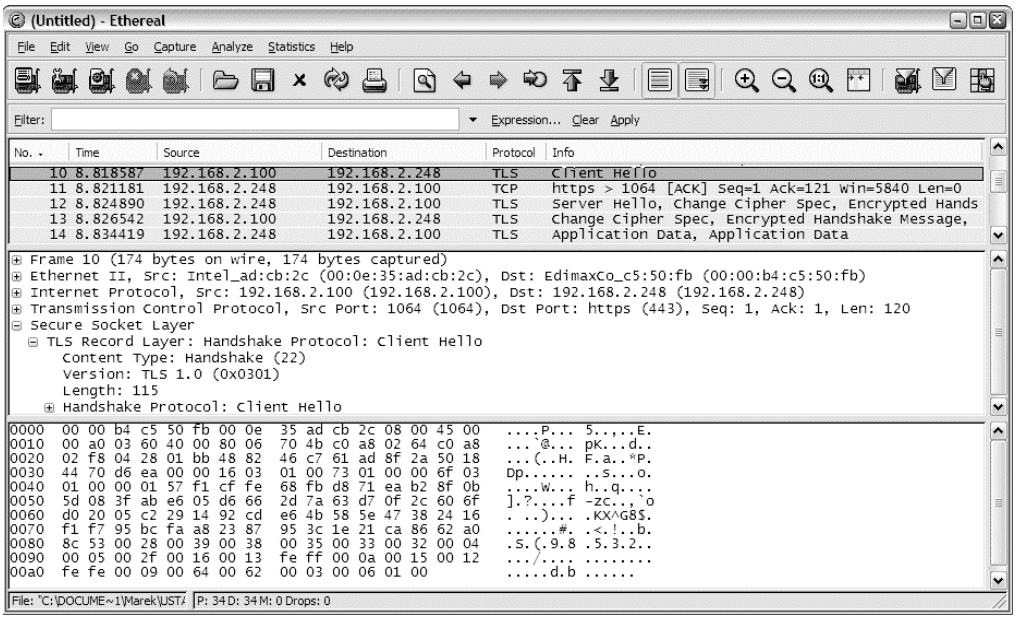
```
ETHER: ----- Ether Header -----
ETHER:
ETHER: Packet 4 arrived at 13:04:27.94
ETHER: Packet size = 156 bytes
ETHER: Destination = 0:3:ba:8:1d:d,
ETHER: Source = 0:8:e2:42:b1:fc,
ETHER: Ethertype = 0800 (IP)
ETHER:
IP: ----- IP Header -----
IP:
IP: Version = 4
IP: Header length = 20 bytes
IP: Type of service = 0x00
IP: xxx. .... = 0 (precedence)
IP: ...0 .... = normal delay
IP: .... 0... = normal throughput
IP: .... .0.. = normal reliability
IP: Total length = 142 bytes
```

```
IP: Identification = 44235
IP: Flags = 0x4
IP: .1.. .... = do not fragment
IP: ..0. .... = last fragment
IP: Fragment offset = 0 bytes
IP: Time to live = 125 seconds/hops
IP: Protocol = 6 (TCP)
IP: Header checksum = 8652
IP: Source address = 192.168.1.100, 192.168.1.100
IP: Destination address = 192.168.1.101, 192.168.1.101
IP: No options
IP:
TCP: ----- TCP Header -----
TCP:
TCP: Source port = 1516
TCP: Destination port = 443 (HTTPS)
TCP: Sequence number = 694787250
TCP: Acknowledgement number = 1952824342
TCP: Data offset = 20 bytes
TCP: Flags = 0x18
TCP: ..0. .... = No urgent pointer
TCP: ...1 .... = Acknowledgement
TCP: .... 1... = Push
TCP: .... .0.. = No reset
TCP: .... ..0. = No Syn
TCP: .... ...0 = No Fin
TCP: Window = 65520
TCP: Checksum = 0xf26c
TCP: Urgent pointer = 0
TCP: No options
TCP:
HTTPS: ----- HTTPS: -----
HTTPS:
HTTPS: ""
HTTPS:
```

Do szyfrowania kanału komunikacyjnego mechanizm SSL wykorzystuje algorytm kryptograficzny bazujący na kluczu publicznym. Z tego względu ustanowienie połączenia SSL wiąże się z pewnym narzutem komunikacyjnym. Przed przesłaniem jakichkolwiek danych HTTP klient i serwer muszą wymienić klucze publiczne, zaakceptować zbiór algorytmów szyfrujących itd. Na rysunku 5.1 została przedstawiona przykładowa transakcja SSL ustanawiająca połączenie między serwerem i klientem, zarejestrowana przez narzędzie monitorujące *Ethereal*.

Wymagane oprogramowanie

Aby zaimplementować mechanizm SSL w działaniu serwera Apache, trzeba najpierw zainstalować pakiet OpenSSL. Ostatnią wersję oprogramowania OpenSSL można pobrać ze strony *www.openssl.org*. W czasie pisania książki była to wersja 0.9.8b. Biblioteka OpenSSL jest niezbędna podczas kompilowania modułu `mod_ssl`, w którym są wykorzystywane różne jej funkcje kryptograficzne.



Rysunek 5.1. *Wymiana danych protokołu SSL*

Poza pakietem oprogramowania OpenSSL system operacyjny musi obsługiwać urządzenie przeznaczone do generowania wartości pseudolosowych. Większość systemów Unix jest wyposażona domyślnie w urządzenie */dev/random*, odpowiednie dla modułu `mod_ssl`. Z kolei aby takie urządzenie było dostępne w systemie Solaris, należy zainstalować odpowiednią nakładkę. W przypadku braku generatora liczb losowych program Apache nie zostanie uruchomiony i spowoduje wygenerowanie komunikatów o błędach.

Instalacja oprogramowania SSL

Osoby korzystające z serwera Apache w wersji 1.3 muszą pobrać kod modułu `mod_ssl` ze strony *www.modssl.org*. Oprogramowanie serwera w wersji 2.0 jest rozpowszechniane wraz z odpowiednią wersją biblioteki `mod_ssl`. Zatem procedura włączenia modułu podczas kompilacji kodu serwera sprowadza się do zastosowania opcji `--enable-ssl`. Niestety podczas kompilowania biblioteki `mod_ssl` jako obiektu DSO (dla Apache 2.0.52) występuje pewien błąd. Sama kompilacja przebiega co prawda poprawnie, ale proces potomny SSL nie obsługuje żadnych żądań i przedwcześnie kończy swoją pracę. Z informacji dostępnych w internecie wynika, że ten problem został zauważony przez wielu administratorów, którzy próbowali wykorzystać moduł `mod_ssl` jako obiekt DSO w połączeniu ze statycznie skompilowanym pakietem OpenSSL. Aby instalacja przebiegła prawidłowo, należy wykonać procedurę opisaną w rozdziale 3.

Tworzenie certyfikatów SSL

W opisywanej procedurze instalacyjnej został wykorzystany moduł `mod_ssl` rozpowszechniany z oprogramowaniem Apache 2.0.52. Do utworzenia certyfikatów nie można zatem wykorzystać tej samej metody, która jest stosowana w odniesieniu do kodu pobranego z witryny *modssl.org*. Choć trzeba przyznać, że procedura generowania certyfikatów dla modułu dostępnego w witrynie *modssl.org* jest znacznie mniej nużąca. Sprowadza się do wykonania poniższego polecenia (w pliku *Makefile* znajduje się odpowiednia reguła).

```
# make certificate TYPE=custom
```

Wykonanie instrukcji powoduje wyświetlenie niewielkiego menu, które prowadzi użytkownika przez całą procedurę konfiguracyjną, związaną z definiowaniem nazwy organizacji, określeniem umiejscowienia serwera, nazwy domeny i wszystkich parametrów zapisywanych w certyfikacie SSL. Niestety wersja oprogramowania `mod_ssl` dostarczana wraz z kodem Apache 2.0.52 nie zawiera interfejsu dla programu *make*. Administratorowi pozostaje więc jedynie wykorzystanie biblioteki OpenSSL. Wynik końcowy jest oczywiście taki sam jak w przypadku zastosowania oprogramowania pobranego ze strony *modssl.org* — tworzony jest certyfikat SSL oraz żądanie podpisania certyfikatu (ang. *Certificate Signing Request* — CSR), przesyłane później do urzędu certyfikującego (ang. *Certificate Authority* — CA), takiego jak VeriSign czy Entrust. Niestety dojście do tego etapu nie jest wcale łatwe. Na kolejnym listingu są przedstawione polecenia, które tworzą katalogi potrzebne do przechowywania plików SSL oraz instrukcje odpowiedzialne za wygenerowanie samego certyfikatu.

```
# mkdir /usr/local/apache/conf/ssl.key
# mkdir /usr/local/apache/conf/ssl.crt
# mkdir /usr/local/apache/conf/ssl.crl
# /usr/local/ssl/bin/openssl req \  -new \  -x509 \  -days 60 \  -keyout
/usr/local/apache/conf/ssl.key/server.key \  -out
/usr/local/apache/conf/ssl.crt/server.crt
Using configuration from /usr/share/ssl/openssl.cnf
Generating a 1024 bit RSA private key
.............++++++
.............++++++
writing new private key to '/tmp/server.key'
Enter PEM pass phrase:******
Verifying password - Enter PEM pass phrase:******
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:PL
State or Province Name (full name) [Some-State]:slaskie
Locality Name (eg, city) []:Gliwice
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Zabezpieczenia Apache
Organizational Unit Name (eg, section) []:Zabezpieczenia WWW
Common Name (eg, your name or your server's hostname) []:www.moja_nazwa.pl
Email Address []:webmaster@moja_nazwa.pl
```

Sprawdzenie wstępnej konfiguracji

Na tym etapie prac warto sprawdzić, czy moduł `mod_ssl` będzie poprawnie działał przy domyślnej konfiguracji. Do uruchomienia serwera należy wykorzystać standardowy skrypt *apachectl*, ale tym razem z opcją `startssl`.

```
# /usr/local/apache/bin/apachectl startssl
Apache/2.0.52 mod_ssl/2.0.52 (Pass Phrase Dialog)
Some of your private key files are encrypted for security reasons.
In order to read them you have to provide us with the pass phrases.

Server www.example.com:443 (RSA)
Enter pass phrase:

Ok: Pass Phrase Dialog successful.

# ps -ef | grep httpd
root      3806     1  0 14:23 ?      00:00:00 /usr/local/apache/bin/httpd -k start
   -DSSL
webserv   3807  3806  0 14:23 ?      00:00:00 /usr/local/apache/bin/httpd -k start
   -DSSL
webserv   3808  3806  0 14:23 ?      00:00:00 /usr/local/apache/bin/httpd -k start
   -DSSL
webserv   3809  3806  0 14:23 ?      00:00:00 /usr/local/apache/bin/httpd -k start
   -DSSL
webserv   3810  3806  0 14:23 ?      00:00:00 /usr/local/apache/bin/httpd -k start
   -DSSL
webserv   3811  3806  0 14:23 ?      00:00:00 /usr/local/apache/bin/httpd -k start
   -DSSL
```

Z analizy listingu wynika, że serwer obsługuje mechanizm SSL, co można stwierdzić na podstawie opcji `-DSSL`. Ostatni etap procedury obejmuje próbę połączenia się z serwerem WWW i sprawdzenie, czy rzeczywiście obsługuje on żądania klienckie. Jeżeli do testu zostanie wykorzystana aplikacja *s_client* pakietu OpenSSL, administrator będzie mógł zweryfikować dane certyfikatów SSL, które sam wcześniej wprowadził. Po ustanowieniu połączenia wystarczy wpisać standardowe polecenie `HTTP HEAD`. Serwer odeśle wówczas typowe nagłówki odpowiedzi HTTP. Przykład zastosowania opisywanej aplikacji znajduje się poniżej.

```
# openssl s_client -connect 192.168.2.248:443
CONNECTED(00000003)
depth=0 /C=PL/ST=slaskie/L=Gliwice/O=Zabezpieczenia Apache/OU=Zabezpieczenia
WWW/CN=www.moja_nazwa.pl/emailAddress=webmaster@moja_nazwa.pl
verify error:num=18:self signed certificate
verify return:1
depth=0 /C=PL/ST=slaskie/L=Gliwice/O=Zabezpieczenia Apache/OU=Zabezpieczenia
WWW/CN=www.moja_nazwa.pl/emailAddress=webmaster@moja_nazwa.pl
verify return:1
---
Certificate chain
 0 s:/C=PL/ST=slaskie/L=Gliwice/O=Zabezpieczenia Apache/OU=Zabezpieczenia
WWW/CN=www.moja_nazwa.pl/emailAddress=webmaster@moja_nazwa.pl
   i:/C=PL/ST=slaskie/L=Gliwice/O=Zabezpieczenia Apache/OU=Zabezpieczenia
WWW/CN=www.moja_nazwa.pl/emailAddress=webmaster@moja_nazwa.pl
---
```

```
Server certificate
-----BEGIN CERTIFICATE-----
MIID8TCCA1qgAwIBAgIBADANBgkqhkiG9w0BAQQFADCBsjELMAkGA1UEBhMCUEwx
EDAOBgNVBAgTB3NsYXNraWUxEDAOBgNVBAcTB0dsaXdpY2UxHjAcBgNVBAoTFVph
YmV6cGllY3plbmlhIEFwYWNoZTEbMBkGA1UECxMSWmFiZXpwaWVjemVuaWEgV1dX
MRowGAYDVQQDFBF3d3cubW9qYV9uYXp3YS5wbDEmMCQGCSqGSIb3DQEJARYXd2Vi
bWFzdGVyQG1vamFfbmF6d2EucGwwHhcNMDYwNjIxMTIyMzEyWhcNMDYwODIwMTIy
MzEyWjCBsjELMAkGA1UEBhMCUEwxEDAOBgNVBAgTB3NsYXNraWUxEDAOBgNVBAcT
B0dsaXdpY2UxHjAcBgNVBAoTFVphYmV6cGllY3plbmlhIEFwYWNoZTEbMBkGA1UE
CxMSWmFiZXpwaWVjemVuaWEgV1dXMRowGAYDVQQDFBF3d3cubW9qYV9uYXp3YS5w
bDEmMCQGCSqGSIb3DQEJARYXd2VibWFzdGVyQG1vamFfbmF6d2EucGwwgZ8wDQYJ
KoZIhvcNAQEBBQADgY0AMIGJAoGBANDGhsVGjya95LkwrMoS4Ukyq4Q2XQd7Ksy5
2xa7bpOqwU9NSyiImESIXNn8ozy+j/u0ilCqf4Ney8+vyH5YLsvpNq321meKIrdy
HIvN5SVqYrhJSRjVogTk5vRs79l29WcW3Y2Kc6UcGfFbtuIEob2n2gnfvJ4b+qgu
/+smCpn5AgMBAAGjggETMIIBDzAdBgNVHQ4EFgQUpv+7RTs7O9n+DByi0rLsb7NH
acgwgd8GA1UdIwSB1zCB1IAUpv+7RTs7O9n+DByi0rLsb7NHacihgbikgbUwgbIx
CzAJBgNVBAYTAlBMMRAwDgYDVQQIEwdzbGFza2llMRAwDgYDVQQHEwdHbGl3aWNl
MR4wHAYDVQQKExVaYWJlenBpZWN6ZW5pYSBBcGFjaGUxGzAZBgNVBAsTElphYmV6
cGllY3plbmlhIFdXVzEaMBgGA1UEAxQRd3d3Lm1vamFfbmF6d2EucGwxJjAkBgkq
hkiG9w0BCQEWF3dlYm1hc3RlckBtb2phX25hendhLnBsggEAMAwGA1UdEwQFMAMB
Af8wDQYJKoZIhvcNAQEEBQADgYEARHdWGAo23vluFths2pIAdG+L1p+ts2QAWh6B
9E6xKUXAICQ62Bh6yFcfc8KkefrEr188pT92SZ+N7kPYcO4enl6SQPfvY7BXm/JO
Ss1RzCvT5/f5rmIDb5d2JhgL3LAHDvjLMs5A51D4RotdbYjZE/VmfGVxCVqbhtwb
CrzXl24=
-----END CERTIFICATE-----
subject=/C=PL/ST=slaskie/L=Gliwice/O=Zabezpieczenia Apache/OU=Zabezpieczenia
WWW/CN=www.moja_nazwa.pl/emailAddress=webmaster@moja_nazwa.pl
issuer=/C=PL/ST=slaskie/L=Gliwice/O=Zabezpieczenia Apache/OU=Zabezpieczenia
WWW/CN=www.moja_nazwa.pl/emailAddress=webmaster@moja_nazwa.pl
---
No client certificate CA names sent
---
SSL handshake has read 1577 bytes and written 340 bytes
---
New, TLSv1/SSLv3, Cipher is DHE-RSA-AES256-SHA
Server public key is 1024 bit
SSL-Session:
    Protocol  : TLSv1
    Cipher    : DHE-RSA-AES256-SHA
    Session-ID: EB220700074951F9D6F874A06D30C6DFC60EF8B8E07D805C10981D89B30BC0C5
    Session-ID-ctx:
    Master-Key:
2E50087D58325FF259F87DF6C9B8FBFDA440A2EA39F96E955A01C720BCB005A9C522863DFA10AE04E4C
8EFACC73D5695
    Key-Arg   : None
    Krb5 Principal: None
    Start Time: 1150893428
    Timeout   : 300 (sec)
    Verify return code: 18 (self signed certificate)
---
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Wed, 21 Jun 2006 12:38:33 GMT
Server: Apache
Last-Modified: Sat, 17 Jun 2006 12:51:49 GMT
ETag: "5b1a6-7-ff18c340"
Accept-Ranges: bytes
```

```
Content-Length: 7
Connection: close
Content-Type: text/html; charset=ISO-8859-1

Closed
```

Doskonale! Aplikacja kliencka OpenSSL pokazuje przebieg negocjacji SSL. Można więc dzięki niej przeanalizować dane zawarte w utworzonym wcześniej certyfikacie SSL (np. określić nazwę domenową itp.) oraz parametry mechanizmu SSL (w tym wersję protokołu i rodzaj algorytmu szyfrującego). Mimo iż nowo uruchomiony serwer HTTPS może już obsługiwać żądania klientów, konieczne jest jeszcze wprowadzenie pewnych zmian w jego domyślnej konfiguracji — podobnie jak w przypadku modyfikacji pliku *httpd.conf*.

Konfiguracja modułu mod_ssl

Moduł `mod_ssl` udostępnia administratorowi wiele dyrektyw konfiguracyjnych. Nie wszystkie jednak zostaną tu opisane. Celem niniejszego punktu jest przedstawienie jedynie tych ustawień, które mają bezpośredni związek z zabezpieczeniami i które służą rozwiązaniu najczęściej występujących problemów. Szczegółowe informacje na temat dyrektyw modułu `mod_ssl` są zamieszczone na stronach dokumentacji Apache 2.0 (*http://httpd.apache.org/docs-2.0/mod/mod_ssl.html*) oraz w witrynie *modssl.org* (*www.modssl.org/docs/2.8*).

Dyrektywa SSLEngine

Dyrektywa `SSLEngine` włącza lub wyłącza pracę mechanizmu SSL/TLS. Zazwyczaj jest ona wymieniana w kontenerze `VirtualHost` dla portu 443. Aby włączyć opcję, należy umieścić w pliku konfiguracyjnym wpis `SSLEngine ON`.

Dyrektywa SSLProtocol

Serwer Apache może pracować z trzema wersjami protokołu SSL.

- **SSLv2**. Jest to pierwotny protokół, opracowany przez firmę Netscape Corporation w 1994 roku. Został on zaprojektowany z myślą o zapewnieniu bezpiecznej komunikacji w internecie. Mimo iż podstawowe założenie zostało spełnione, w samym protokole odkryto wiele nieprawidłowości. Do najważniejszych z nich trzeba zaliczyć możliwość wymuszenia przez jednostkę kliencką stosowania słabszych algorytmów szyfrowania i brak ochrony dla procedury wymiany kluczy. Wady te sprawiły, że rozwiązanie okazało się mniej bezpieczne, niż początkowo sądzono. Słabości protokołu wymiany kluczy pakietu OpenSSL SSLv2 zostały później wykorzystane w kodzie robaka Slapper do łamania zabezpieczeń systemu.
- **SSLv3**. Uaktualnienie protokołu SSL zostało wydane przez firmę Netscape w 1996 roku. Jego celem było wyeliminowanie niedociągnięć wersji SSLv2. Mechanizm ten szybko stał się podstawowym algorytmem szyfrowania danych w komunikacji sieciowej.

- **TLSv1**. Skrót TLS jest akronimem od angielskich słów Transport Layer Security, oznaczających zabezpieczenie warstwy transportowej. Protokół TLS został zdefiniowany przez organizację Internet Engineering Task Force w 1999 roku w dokumencie RFC 2246. Głównym celem prac IETF było utworzenie otwartego standardu SSL oraz przetestowanie i włączenie do standardu niektórych rozwiązań firmy Microsoft z pakietu PCT.

Mając podstawowe informacje na temat różnych wersji protokołu, można przystąpić do konfigurowania jego ustawień. Operowanie parametrami dyrektywy `SSLProtocol` jest zbliżone do ustawień `Options` (opisanych w poprzednim rozdziale). Do włączania i wyłączania konkretnych wersji protokołu stosowane są odpowiednio znaki plus (+) i minus (-). Można również użyć słowa kluczowego `all`, które zastępuje definicję `+SSLv2 +SSLv3 +TLSv1`. Z uwagi na wady protokołu SSLv2 powinien on być wyłączony. Zatem zalecanym ciągiem tekstowym dyrektywy jest *`SSLProtocol all -SSLv2`*.

Dyrektywa SSLCipherSuite

Jeśli chce się mieć naprawdę skuteczne zabezpieczenie kryptograficzne, trzeba spełnić dwa warunki — zagwarantować dostępność silnego algorytmu kryptograficznego i długiego klucza szyfrowania. Istnieje wiele dowodów na to, że nieefektywny mechanizm szyfrowania lub krótki klucz (mniejszy niż 128-bitowy) przy obecnej mocy obliczeniowej komputerów można złamać w bardzo krótkim czasie. Dyrektywa `SSLCipherSuite` pozwala administratorowi na odpowiednie wyznaczenie obydwu parametrów. Wartościami ustawienia są cztery rozdzielane znakiem dwukropka specyfikacje algorytmów szyfrowania, które klient może wykorzystać w trakcie negocjacji. Cztery wspomniane specyfikacje wyznaczają cztery kategorie parametrów — algorytm wymiany kluczy, algorytm uwierzytelniania, algorytm szyfrowania i algorytm skrótu MAC. W ramach każdej kategorii administrator może definiować wartości odpowiadające poszczególnym algorytmom, takim jak RSA, Diffie-Hellman i 3DES. Kolejność wymieniania opcji wyznacza kolejność użycia ich przez jednostkę kliencką. Pewnym ułatwieniem procedury konfiguracyjnej jest możliwość stosowania aliasów zamiast list poszczególnych parametrów. Dyrektywa gwarantująca najwyższy stopień zabezpieczeń powinna mieć następującą treść:

```
SSLCipherSuite HIGH:MEDIUM:!aNULL:+SHA1:+MD5:+HIGH:+MEDIUM
```

Powyższy zapis oznacza, że dozwolone są tylko silne algorytmy szyfrujące (o kluczach dłuższych niż 128 bitów). Ponadto wyłączone zostają algorytmy słabsze i nieszyfrujące danych, wyłączone jest również anonimowe uwierzytelnianie. Algorytm SHA1 ma pierwszeństwo przed MD5 (ze względu na problemy z kolizjami MD5).

Dyrektywa SSLRandomSeed

Innym niezwykle istotnym elementem dobrego algorytmu szyfrowania jest odpowiednie działanie generatora liczb losowych. Użycie losowych wartości jako wartości wejściowej dla funkcji szyfrowania OpenSSL znacznie utrudnia osobie atakującej predykcję przyszłych danych. Jednym z parametrów tej dyrektywy może być parametr `builtin`, który oznacza stosowanie wartości będącej połączeniem aktualnego czasu, identyfikatora uruchomionego procesu oraz przypadkowej wartości pobranej z listy procesów Apache.

Wada tego rozwiązania polega na tym, że przypadkowość wartości pobranej z listy procesów nie jest wystarczająca. Natomiast dwa pozostałe elementy są potencjalnie łatwe do ustalenia.

Najczęściej stosowanymi generatorami liczb losowych na platformach Unix są urządzenia */dev/random* i */dev/urandom*. Obydwa wymienione urządzenia mogą być stosowane jako źródła wartości początkowej dla funkcji szyfrowania. Jednak zalecanym urządzeniem jest */dev/urandom*. Wynika to z faktu, że w przypadku niemożności uzyskania dostatecznej entropii */dev/random* może się zablokować. Oprogramowanie Apache pozwala na wykorzystywanie dyrektywy `SSLRandomSeed` w dwóch różnych kontekstach — startu serwera (`startup`) (analizowana w chwili pierwszego uruchamiania za pomocą skryptu *apachectl*) oraz połączenia (`connect`) (analizowana w chwili inicjowania przez jednostkę kliencką połączenia z procesem potomnym). Zalecanym ustawieniem dyrektywy jest:

```
SSLRandomSeed   startup   file:/dev/urandom   512
SSLRandomSeed   connect   file:/dev/urandom   512
```

Dyrektywy SSLCertificateFile i SSLCertificateKeyFile

Te dwie dyrektywy stanowią dla serwera Apache informację o tym, w których plikach jest zapisany certyfikat i klucz prywatny SSL. Jeżeli został utworzony certyfikat PEM serwera, zawierający jego klucz prywatny, można wykorzystać jedynie dyrektywę `SSLCertificateFile`. W większości przypadków są jednak stosowane dwa niezależne pliki. Wystarczy przejrzeć przedstawione wcześniej polecenie `openssl`, aby zobaczyć, że generuje ono dwa pliki certyfikatów SSL. Konieczne jest zatem wprowadzenie dwóch następujących wierszy:

```
SSLCertificateFile /usr/local/apache/conf/ssl.crt/server.crt
SSLCertificateKeyFile /usr/local/apache/conf/ssl.key/server.key
```

Certyfikaty bez haseł

Klucz prywatny serwera (*server.key*) jest szyfrowany. Z tego względu podczas uruchamiania usługi Apache administrator musi podać hasło, które pozwoli oprogramowaniu rozszyfrować plik i wykorzystać klucz prywatny. Pod względem bezpieczeństwa systemu takie rozwiązanie jest optymalne. Gwarantuje bowiem zachowanie poufności nawet w przypadku, gdy włamywacz zdobędzie plik klucza prywatnego — jeżeli nie będzie znał hasła, nie będzie mógł rozszyfrować treści pliku i użyć klucza. Uniemożliwia zatem obcej jednostce podszycie się pod serwer z przechwyconym certyfikatem SSL.

Wymienione zalety wydają się dostatecznie przekonujące, by uznać stosowanie haseł za najkorzystniejsze rozwiązanie. Wiąże się ono jednak z dużymi utrudnieniami w automatyzacji zadań administracyjnych. Wielu doświadczonych administratorów serwerów uruchamia różnego rodzaju skrypty monitorujące pracę usługi httpd. Gdy skrypt wykryje wyłączenie serwera, automatycznie próbuje ponownie go uruchomić. Metoda ta nie sprawdza się jednak w przypadku zabezpieczenia klucza prywatnego za pomocą hasła. Jednym ze sposobów rozwiązania problemu jest umieszczenie hasła w kodzie skryptu i automatyczne wprowadzenie danych uwierzytelniających na żądanie programu serwera. Koncepcja ta wyklucza jednak wszystkie korzyści związane z bezpieczeństwem i ochroną klucza.

Niewątpliwie jednym z najczęściej występujących pytań na listach FAQ jest pytanie o sposób usunięcia hasła z certyfikatu SSL. Przedstawiane w tej książce rozwiązania bazują na założeniu, że bezpieczeństwo aplikacji WWW zależy od zabezpieczeń danej jednostki. Zgodnie z informacjami zawartymi w rozdziale 2. serwera nie można uznać za bezpieczny, jeżeli brakuje zabezpieczeń samego systemu operacyjnego. Przyjmując ten tok rozumowania, należy stwierdzić, że jeżeli nie można polegać na systemie operacyjnym, to ryzyko przechwycenia klucza prywatnego SSL powinno być najmniejszym zmartwieniem. Zatem usunięcie hasła z pliku klucza prywatnego nie powinno stanowić problemu przy założeniu, że administrator skrupulatnie zrealizował wszystkie procedury zabezpieczenia jednostki. Aby usunąć z pliku klucza prywatnego hasło, trzeba wykonać następujące instrukcje:

```
# cd /usr/local/apache/conf/ssl.key
# mv server.key server.key.old
# openssl rsa -in server.key.old -out server.key
```

Zadanie przedstawionych poleceń polega na utworzeniu kopii zapasowej pliku *server.key* oraz usunięciu szyfrowania RSA z tego pliku. Od chwili wykonania tej operacji serwer Apache nie będzie wymagał hasła podczas uruchamiania usługi w trybie SSL.

Dyrektywa SSLOptions

Moduł SSL pozwala na definiowanie wielu opcji, które umożliwiają precyzyjne parametryzowanie sposobu jego działania w szczególnych sytuacjach (np. gdy trzeba wprowadzić dodatkową zmienną środowiskową dla skryptu CGI lub wykorzystać fragmenty certyfikatu klienta jako dane uwierzytelniające w podstawowej procedurze uwierzytelniania [ang. *Basic Authentication*]). Wśród nich jest opcja, która pozwala na kontynuowanie przetwarzania żądania klienckiego, jeżeli spełni ono jedno z kryteriów. Z punktu widzenia osoby zajmującej się zabezpieczeniami nie jest to rozwiązanie idealne, gdyż daje możliwość zaakceptowania żądania, które zostało odrzucone na podstawie innej opcji. Przedstawiona dyrektywa zapewni ścisłe respektowanie wszystkich reguł.

```
SSLOptions +StrictRequire
```

SSLRequireSSL

Po uwierzytelnieniu klienta w aplikacji WWW jego dane są zapisywane w nagłówku podstawowego uwierzytelnienia lub w formie identyfikatora sesji w pliku cookie. W obydwu przypadkach istotne jest, aby poufne informacje nie zostały przekazane przez internet w formie otwartego tekstu — każda aplikacja monitorujące pracę sieci mogłaby takie dane przechwycić. Problem można rozwiązać kilkoma metodami, a oprogramowanie Apache zapewnia jedną z nich. Zastosowanie dyrektywy `SSLRequireSSL` wymusza na jednostce klienckiej stosowanie protokołu SSL podczas próby dostępu do zasobów o określonym adresie URL. Jeżeli klient prześle żądanie w protokole HTTP, zostanie ono odrzucone z kodem statusowym `403-Forbidden`. Dyrektywa może występować wewnątrz różnych kontenerów, w tym w `Directory`, `Location` i `LocationMatch`. Oto przykład:

```
<LocationMatch /account/login.*>
SSLRequireSSL
AuthType Basic
```

```
AuthUserFile /ścieżka/do/pliku_haseł
Require user test
</LocationMatch>
```

Choć stosowanie opisywanej dyrektywy wydaje się dobrym mechanizmem zabezpieczającym, ma ono pewną wadę, którą trzeba uwzględnić. Stanowi bowiem doskonałą *bramę wejściową*, która wymusza stosowanie protokołu SSL na klientach rozpoczynających pracę z aplikacją wymagającą uwierzytelnienia. Niemniej mechanizm ten nie może funkcjonować jako *brama wyjściowa*. Gdy klient opuszcza witrynę zabezpieczoną protokołem SSL i pobiera stronę z serwera HTTP wymagającego dostarczenia hasła otwartym tekstem, przeglądarka może przesłać dane uwierzytelniające z witryny szyfrowanej w formie niezabezpieczonej. Aby usunąć tę niedogodność, administratorzy serwerów stosują dwa rozwiązania. Pierwsze z nich polega na dołączeniu opcji `secure` do wszystkich danych cookie przesyłanych w ramach aplikacji WWW. Dzięki temu przeglądarka otrzymuje informację, że pliki cookie należy przesyłać jedynie do stron objętych działaniem protokołu SSL. Druga metoda sprowadza się do zaimplementowania mechanizmu wylogowania, który zagwarantuje zamknięcie okna przeglądarki przechowującego dane uwierzytelniające. W praktycznej realizacji opisanej koncepcji pomocny jest kod JavaScript.

```
<INPUT onClick="javascript:window.close()" TYPE="BUTTON" VALUE="Zamknij"
  TITLE="Kliknij, aby zamknąć okno" NAME="CloseWindow" >
```

Gdy użytkownik kliknie przycisk *Zamknij*, na ekranie zostanie wyświetlone okno dialogowe przedstawione na rysunku 5.2.

Rysunek 5.2. *Okno dialogowe JavaScript, które poprzedza zamknięcie okna przeglądarki z danymi uwierzytelniającymi*

Dyrektywy SSLVerifyClient i SSLRequire

Wymienione dyrektywy zazwyczaj występują jednocześnie, ponieważ odpowiadają za weryfikację certyfikatu klienta. Działanie dyrektywy `SSLVerifyClient` nie jest szczególnie skomplikowane. Jeżeli ma ona wartość `require`, klient musi przedstawić certyfikat, aby dostać się do chronionych zasobów. Po przesłaniu certyfikatu przez klienta serwer musi sprawdzić, jakie dane są potrzebne, aby można było zaakceptować żądanie dostępu do wymienionych zasobów. Definiowanie listy potrzebnych informacji należy do zadań parametru `SSLRequire`. Dyrektywa `SSLRequire` jest niezwykle interesująca, chociażby ze względu na elastyczność wyznaczania parametrów. Pozwala na zapisywanie wyrażeń logicznych, bazujących na zmiennych środowiskowych CGI i SSL. Aby klientowi zostało przydzielone prawo korzystania z zasobów, muszą być spełnione wszystkie zdefiniowane wyrażenia. Na poniższym zestawieniu znajduje się wykaz wszystkich zmiennych środowiskowych, które mogą być stosowane w treści dyrektywy `SSLRequire`.

Zmienne standardu CGI/1.0 i serwera Apache:

HTTP_USER_AGENT	PATH_INFO	AUTH_TYPE
HTTP_REFERER	QUERY_STRING	SERVER_SOFTWARE
HTTP_COOKIE	REMOTE_HOST	API_VERSION
HTTP_FORWARDED	REMOTE_IDENT	TIME_YEAR
HTTP_HOST	IS_SUBREQ	TIME_MON
HTTP_PROXY_CONNECTION	DOCUMENT_ROOT	TIME_DAY
HTTP_ACCEPT	SERVER_ADMIN	TIME_HOUR
HTTP:**nazwa_nagłówka**	SERVER_NAME	TIME_MIN
THE_REQUEST	SERVER_PORT	TIME_SEC
REQUEST_METHOD	SERVER_PROTOCOL	TIME_WDAY
REQUEST_SCHEME	REMOTE_ADDR	TIME
REQUEST_URI	REMOTE_USER	ENV:**nazwa_zmiennej**
REQUEST_FILENAME		

Zmienne związane z protokołem SSL:

HTTPS	SSL_CLIENT_M_VERSION	SSL_SERVER_M_VERSION
	SSL_CLIENT_M_SERIAL	SSL_SERVER_M_SERIAL
SSL_PROTOCOL	SSL_CLIENT_V_START	SSL_SERVER_V_START
SSL_SESSION_ID	SSL_CLIENT_V_END	SSL_SERVER_V_END
SSL_CIPHER	SSL_CLIENT_S_DN	SSL_SERVER_S_DN
SSL_CIPHER_EXPORT	SSL_CLIENT_S_DN_C	SSL_SERVER_S_DN_C
SSL_CIPHER_ALGKEYSIZE	SSL_CLIENT_S_DN_ST	SSL_SERVER_S_DN_ST
SSL_CIPHER_USEKEYSIZE	SSL_CLIENT_S_DN_L	SSL_SERVER_S_DN_L
SSL_VERSION_LIBRARY	SSL_CLIENT_S_DN_O	SSL_SERVER_S_DN_O
SSL_VERSION_INTERFACE	SSL_CLIENT_S_DN_OU	SSL_SERVER_S_DN_OU
	SSL_CLIENT_S_DN_CN	SSL_SERVER_S_DN_CN
	SSL_CLIENT_S_DN_T	SSL_SERVER_S_DN_T
	SSL_CLIENT_S_DN_I	SSL_SERVER_S_DN_I
	SSL_CLIENT_S_DN_G	SSL_SERVER_S_DN_G
	SSL_CLIENT_S_DN_S	SSL_SERVER_S_DN_S
	SSL_CLIENT_S_DN_D	SSL_SERVER_S_DN_D
	SSL_CLIENT_S_DN_UID	SSL_SERVER_S_DN_UID
	SSL_CLIENT_S_DN_Email	SSL_SERVER_S_DN_Email
	SSL_CLIENT_I_DN	SSL_SERVER_I_DN
	SSL_CLIENT_I_DN_C	SSL_SERVER_I_DN_C
	SSL_CLIENT_I_DN_ST	SSL_SERVER_I_DN_ST
	SSL_CLIENT_I_DN_L	SSL_SERVER_I_DN_L
	SSL_CLIENT_I_DN_O	SSL_SERVER_I_DN_O
	SSL_CLIENT_I_DN_OU	SSL_SERVER_I_DN_OU
	SSL_CLIENT_I_DN_CN	SSL_SERVER_I_DN_CN
	SSL_CLIENT_I_DN_T	SSL_SERVER_I_DN_T
	SSL_CLIENT_I_DN_I	SSL_SERVER_I_DN_I
	SSL_CLIENT_I_DN_G	SSL_SERVER_I_DN_G
	SSL_CLIENT_I_DN_S	SSL_SERVER_I_DN_S
	SSL_CLIENT_I_DN_D	SSL_SERVER_I_DN_D
	SSL_CLIENT_I_DN_UID	SSL_SERVER_I_DN_UID
	SSL_CLIENT_I_DN_Email	SSL_SERVER_I_DN_Email
	SSL_CLIENT_A_SIG	SSL_SERVER_A_SIG
	SSL_CLIENT_A_KEY	SSL_SERVER_A_KEY
	SSL_CLIENT_CERT	SSL_SERVER_CERT
	SSL_CLIENT_CERT_CHAINn	
	SSL_CLIENT_VERIFY	

Jako przykład wykorzystania obydwu dyrektyw rozważmy aplikację, w której istnieje katalog *rachunki*. Do katalogu *rachunki* powinni mieć dostęp jedynie pracownicy działu *ksiegowosc*. Oto stosowne dyrektywy:

```
<Location /klienci/rachunki>
SSLVerifyClient require
SSLRequire %{SSL_CLIENT_S_DN_O} eq "Firma Iks" \
and %{SSL_CLIENT_S_DN_OU} in {"ksiegowosc"}
</Location>
```

Dyrektywy SSLSessionCache i SSLSessionCacheTimeout

Jak nietrudno się domyślić, konieczność wymiany kluczy publicznych podczas zestawiania połączenia między klientem a serwerem jest przyczyną generowania dużego narzutu transmisyjnego. Aby zwiększyć nieco wydajność mechanizmu SSL, oprogramowanie Apache udostępnia możliwość wykorzystywania pamięci podręcznej sesji dla każdego połączenia. Pamięć podręczna jest wykorzystywana przez jednostki klienckie, które przekazują strumienie żądań w ramach mechanizmu `KeepAlive`. Czas trwania sesji dla danego połączenia wyznacza wartość dyrektywy `SSLSessionCacheTimeout`. Zalecane ustawienie obydwu parametrów zostało przedstawione poniżej.

```
SSLSessionCache dbm:/ścieżka/do/apache/logs/ssl_cache
SSLSessionCacheTimeout 60
```

Podsumowanie mechanizmu SSL

Nie ulega wątpliwości, że na temat modułu `mod_ssl` można napisać oddzielną książkę. Ilość zagadnień związanych z szyfrowaniem danych nie pozwala na opisanie ich wszystkich w tej publikacji. Sam moduł `mod_ssl` odgrywa bardzo istotną rolę w całościowym zabezpieczeniu witryny WWW. Zapewnia poufność transmitowanych danych oraz udostępnia mechanizmy uwierzytelniania serwera i klienta. Z tego względu odniesienia do jego funkcji występują także w innych rozdziałach książki.

Moduł mod_rewrite

Najciekawszą cechą modułu mod_rewrite jest to, że zapewnia takie możliwości konfiguracyjne i elastyczność pracy, jak Sendmail. Największą wadą modułu mod_rewrite jest to, że zapewnia takie możliwości konfiguracyjne i elastyczność pracy, jak Sendmail.

— Brian Behlendotf, Apache Group

Mimo niezliczonych przykładów i instrukcji mod_rewrite to czarna magia. Rewelacyjna czarna magia, ale jednak czarna magia.

— Brian Moor, dem@news.cmc.net

W dokumentacji Apache znajduje się informacja, że moduł mod_rewrite jest tak uniwersalny, jak szwajcarski scyzoryk. Jego zadanie polega na odpowiednim przetwarzaniu ciągów URL. To niezwykle użyteczne narzędzie działa na podstawie wyrażeń regularnych, za pomocą których administrator systemu kontroluje dostęp klientów do stron witryny. Skuteczność mechanizmu wynika z możliwości połączenia funkcji wyrażeń regularnych z takimi danymi, jak zmienne środowiskowe CGI i nagłówki HTTP. Liczba parametrów konfiguracyjnych sprawia, że moduł mod_rewrite jest niezwykle skomplikowany, a jego działanie nie zawsze jest oczywiste dla administratora. Z tego względu w tym podrozdziale zostaną zaprezentowane jedynie podstawowe elementy konfiguracyjne, które pozwolą na zaimplementowanie pewnych szczególnych rozwiązań.

Włączenie modułu mod_rewrite

Pierwszą czynnością, jaką trzeba wykonać, jest sprawdzenie, czy kod mod_rewrite został skompilowany wraz z oprogramowaniem Apache. Ponieważ wszystkie kompilowane wcześniej moduły mają postać obiektów DSO, zadanie sprowadza się do sprawdzenia, czy w pliku *httpd.conf* występuje dyrektywa LoadModule właściwa dla tego modułu.

```
# grep mod_rewrite httpd.conf
LoadModule rewrite_module modules/mod_rewrite.so
```

Wynik polecenia potwierdza włączenie modułu. Można więc przystąpić do zapisania kilku dyrektyw, które w dalszej części książki będą wykorzystywane do zabezpieczenia aplikacji.

Dyrektywa RewriteEngine

Dyrektywa RewriteEngine włącza lub wyłącza mechanizm mod_rewrite podczas uruchamiania serwera. Zagadnienie nie jest jednak tak oczywiste, jak by się mogło wydawać. Zmieniając ustawienie RewriteEngine, warto wziąć pod uwagę dwie sprawy. Po pierwsze, jeżeli w pliku konfiguracyjnym występuje znaczna liczba dyrektyw i zachodzi konieczność wyłączenia ich na czas usuwania problemów w działaniu aplikacji, nie należy poprzedzać wierszy modułu znakiem komentarza. Wystarczy zmienić wartość dyrektywy na off i ponownie uruchomić serwer Apache. Po drugie, moduł mod_rewrite musi być implementowany w poszczególnych serwerach wirtualnych (o ile są zdefiniowane), ponieważ jego funkcje nie podlegają dziedziczeniu. Aby włączyć mechanizm, trzeba zastosować następującą dyrektywę:

```
RewriteEngine On
```

Dyrektywa RewriteLog

Dyrektywa RewriteLog wyznacza pliki dziennika, w których mechanizm mod_rewrite będzie rejestrował wszystkie zdarzenia zachodzące w czasie przetwarzania żądań klienckich. Jeżeli w danej aplikacji prowadzenie dzienników jest zbędne, wystarczy tę dyrektywę poprzedzić znakiem komentarza lub usunąć. Niektórzy administratorzy rozwiązują problem, podając jako plik danych wyjściowych plik */dev/null*. Choć takie ustawienie

rzeczywiście powoduje usunięcie zbędnych informacji, nie wstrzymuje działania algorytmu generowania danych wyjściowych, co obniża wydajność serwera. Zalecana postać dyrektywy to:

```
RewriteLog /ścieżka/do/apache/logs/rewrite.log
```

Dyrektywa RewriteLogLevel

Ustawienie `RewriteLogLevel` jest blisko związane z poprzednią dyrektywą. Wyznacza ilość danych rejestrowanych w pliku określonym za pomocą parametru `RewriteLog`. Wartość `0` oznacza całkowite wyłączenie rejestracji (przy podtrzymaniu działania samego mechanizmu, co zostało opisane w poprzednim punkcie). Z kolei wartość `9` i wyższe zapewniają zapisywanie olbrzymich ilości danych. Podczas normalnej pracy serwera parametr ten powinien mieć wartość z przedziału od `2` do `4`. Dyrektywa ma następującą postać:

```
RewriteLogLevel 3
```

Dyrektywy RewriteCond i RewriteRule

Dyrektywa `RewriteCond` występuje razem z dyrektywą `RewriteRule` i definiuje warunek, którego spełnienie oznacza wyzwolenie funkcji opisanej za pomocą ustawienia `RewriteRule`. Dyrektywa `RewriteCond` ma dostęp do tych samych zmiennych środowiskowych CGI, którymi operuje dyrektywa `SSLRequire` (opisana w poprzednim podrozdziale). Gwarantuje jej to niezwykłą elastyczność w działaniu. Dodatkowo istnieje możliwość łączenia reguł `RewriteCond` z bardziej złożonymi wyzwalaczami warunkowymi. Najlepszym sposobem na wyjaśnienie zasady działania obydwu ustawień jest analiza przykładu. Załóżmy, że celem konfiguracji jest zabezpieczenie witryny przed działaniem niebezpiecznej aplikacji robota, która w polu `user-agent` ma wpisaną wartość `zly-robot`. Aby wykonać zadanie, trzeba użyć następujących dyrektyw `RewriteCond` i `RewriteRule`:

```
RewriteCond HTTP_USER_AGENT ^zly-robot$
RewriteRule .* - [F]
```

W przypadku zestawienia połączenia z serwerem jednostka kliencka o ustalonym wcześniej polu nazwy aplikacji (`user-agent`) odbierze komunikat o błędzie `403-Forbidden`. W pliku dziennika zostanie wówczas zapisana następująca treść:

```
192.168.26.1 - - [20/Apr/2005:15:31:44 --0400]
[localhost.localdomain/sid#80a0d50][rid#81be228/initial] (2) init rewrite engine
with requested uri /
192.168.26.1 - - [20/Apr/2005:15:31:44 --0400]
[localhost.localdomain/sid#80a0d50][rid#81be228/initial] (4)
RewriteCond: input='zly-robot' pattern='^zly-robot$' => matched
192.168.26.1 - - [20/Apr/2005:15:31:44 --0400]
[localhost.localdomain/sid#80a0d50][rid#81be228/initial] (2)
forcing '/' to be forbidden
```

Podsumowanie modułu mod_rewrite

Moduł `mod_rewrite` udostępnia wiele niezwykle elastycznych funkcji. Ich wykorzystanie w określonych sytuacjach związanych z zabezpieczeniem serwera zostanie opisane w kolejnych rozdziałach książki — w części dotyczącej zapobiegania rozpoznaniu sieciowemu i implementacji systemów-pułapek.

Moduł mod_log_forensic

Co spowodowało błąd segmentacji procesu potomnego Apache? Takie pytanie często się nasuwa podczas przeglądania wpisów w dzienniku *error_log*, w którym występują wiersze podobne do pokazanych poniżej.

```
[Sun Apr 24 09:11:02 2005] [notice] child pid 5500 exit signal Segmentation fault (11)
```

Zapis bardzo ogólnikowy, prawda? Ogólnie rzecz ujmując, błąd segmentacji nie jest czymś dobrym. Może być spowodowany błędem w kodzie aplikacji, prowadzącym do natychmiastowego przerwania jej działania lub, co gorsza, próbą wykorzystania błędów serwera, prowadzącą do jego „zawieszenia". Niezależnie od przyczyn wpisy podobne do przedstawionych wymagają dokładniejszej analizy. Największy problem polega jednak na tym, że komunikatowi o błędzie segmentacji nie towarzyszą jakiekolwiek dane na temat żądania klienckiego, które doprowadziło do wystąpienia błędu. Wyeliminowanie tej niedogodności należy do zadań modułu `mod_log_forensic`. Jednak przed jego wykorzystaniem trzeba się upewnić, że w pliku *httpd.conf* znajdują się wpisy włączające obiekty DSO `mod_log_forensic` i `mod_unique_id`.

```
# egrep 'log_forensic|unique_id' /usr/local/apache/conf/httpd.conf
LoadModule log_forensic_module modules/mod_log_forensic.so
LoadModule unique_id_module modules/mod_unique_id.so
```

Dyrektywa ForensicLog

Dyrektywa `ForensicLog` wyznacza jedyny parametr modułu — nazwę pliku dziennika. Stanowi ona informację dla serwera Apache, gdzie należy zapisywać dane wyjściowe modułu `mod_log_forensic`. Wartość parametru może się odnosić do zwykłego pliku lub programu, który będzie pobierał informacje. Oto podstawowa forma zapisu ustawienia:

```
ForensicLog /usr/local/apache/logs/forensic.log
```

Zasada działania modułu jest bardzo prosta. Generuje on plik dziennika, w którym każdemu żądaniu odpowiadają dwa wpisy. Pierwszy odzwierciedla żądanie klienckie i jest oznaczony znakiem plus (+). Drugi wpis, oznaczony znakiem minus (-), zawiera odpowiedź serwera na dane żądanie. W obydwu wpisach występuje ten sam unikalny identyfikator operacji. Przykładowy fragment dziennika dla udanej transakcji żądanie-odpowiedź został przedstawiony poniżej:

```
# tail -2 /usr/local/apache/logs/forensic.log
+cDkrlsCoAWUAAC4xChEAAAAA|GET / HTTP/1.1|Accept:image/gif, image/x-xbitmap,
image/jpeg, image/pjpeg,application/vnd.ms-powerpoint, application/vnd.ms-excel,
```

```
application/msword, application/x-shockwave-flash, */*|Accept-Language:en-
us|Accept-Encoding:gzip, deflate|User-Agent:Mozilla/4.0 (compatible; MSIE 5.5;
Windows NT 5.0)|Host:192.168.1.101|Connection:Keep-Alive
-cDkrlsCoAWUAAC4xChEAAAAA
```

W przypadku wystąpienia błędu segmentacji w pliku dziennika nie zostanie zarejestrowana odpowiedź serwera (ciąg poprzedzony znakiem minus). Kod źródłowy Apache jest rozpowszechniany ze skryptem powłoki *check_forensic*, który ułatwia automatyzację procesu sprawdzania pliku *error_log* i wyszukiwania przypadków błędów segmentacji. Kolejny listing przedstawia wynik uruchomienia programu.

```
# /tools/httpd-2.0.52/support/check_forensic /usr/local/apache/logs/forensic.log
+Ll@PbH8AAAEAAFYcFXkAAAAE|GET / HTTP/1.1|Accept:*\*|Accept-Language:en-us|Accept-
Encoding:gzip, deflate|If-Modified-Since:Sat, 19 Feb 2005 16%3a06%3a07 GMT;
length=1833|User-Agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET
CLR 1.1.4322)|Host:192.168.26.134|Connection:Keep-Alive|Transfer-Encoding:Chunked
+NKqZ6X8AAAEAAFYhFuwAAAAF|GET / HTTP/1.1|Accept:*/*|Accept-Language:en-us|Accept-
Encoding:gzip, deflate|If-Modified-Since:Sat, 19 Feb 2005 16%3a06%3a07 GMT;
length=1833|User-Agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET
CLR 1.1.4322)|Host:192.168.26.134|Connection:Keep-Alive|Transfer-Encoding:Chunked
```

W treści wyniku znajdują się informacje o dwóch żądaniach, które doprowadziły do przedwczesnego zakończenie pracy procesu. Analizując nagłówki przesyłane przez przeglądarki klienckie, można dojść do wniosku, że są one związane z próbą wykorzystania błędu wcześniejszych wersji oprogramowania Apache, opisanego na stronie *www.cert.org/advisories/CA-2002-17.html*. Świadczy o tym informacja o rodzaju kodowania — `Transfer-Encoding: Chunked`.

Moduł mod_evasive

W rozdziale 4. zostały opisane standardowe dyrektywy Apache zapewniające minimalizację negatywnych skutków ataku typu DoS. Wśród stosownych ustawień zostały wymienione `Timeout`, `KeepAlive` i `KeepAliveTimeout`. Mimo iż zwiększają one wydajność pracy serwera Apache i ograniczają wpływ ataków DoS, nie gwarantują aż takiej efektywności, jaką zapewnia moduł zewnętrznej firmy — `mod_evasive`.

Do czego służy moduł mod_evasive?

Moduł `mod_evasive` jest obiektem Apache przeznaczonym do ochrony serwera przed atakami HTTP DoS i (lub) atakami bazującymi na metodach siłowych. Został opracowany przez Jonathana Zdziarskiego i jest dostępny w serwisie *www.nuclearelephant.com*. Ważnym elementem rozwiązania jest funkcja wykonywania poleceń systemowych w przypadku wykrycia ataku. Dzięki temu istnieje możliwość przekazania adresu IP atakującej jednostki do innych aplikacji zabezpieczających, takich jak lokalne firewalle (w celu zablokowania połączeń z danego adresu IP). Rozwiązania zaimplementowane w module `mod_evasive` doskonale sprawdzają się zarówno w atakach

pojedynczych, jak i rozproszonych. Niemniej, jak w przypadku każdego ataku DoS, najistotniejszym czynnikiem jest tu szerokość pasma transmisyjnego oraz zajętość procesora i pamięci RAM.

Instalacja modułu mod_evasive

Zgodnie z informacjami zawartymi w rozdziale 3. implementacja modułu `mod_evasive` w postaci obiektu DSO wymaga użycia skryptu Apache *apxs*. Kod źródłowy jest dostępny w dwóch wersjach — dla Apache 1.3 (*mod_evasive.c*) i dla Apache 2.0 (*mod_evasive20.c*). Wykonanie poniższego polecenia powoduje skompilowanie, zainstalowanie i uaktywnienie modułu.

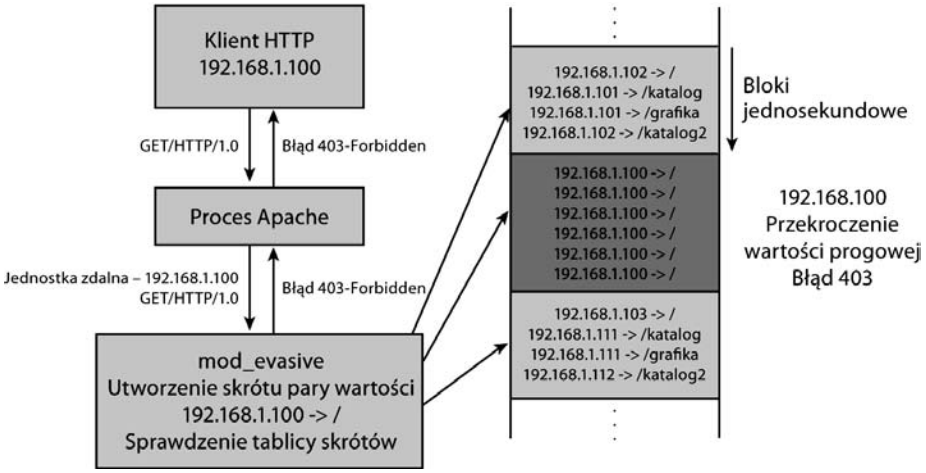
```
# ./apxs -cia /narzedzia/mod_evasive/mod_evasive20.c
/usr/local/apache/build/libtool --silent --mode=compile gcc -prefer-pic  -
DAP_HAVE_DESIGNATED_INITIALIZER -DLINUX=2 -D_REENTRANT -D_XOPEN_SOURCE=500 -
D_BSD_SOURCE -D_SVID_SOURCE -D_GNU_SOURCE -g -O2 -pthread -
I/usr/local/apache/include  -I/usr/local/apache/include   -
I/usr/local/apache/include   -c -o /narzedzia/mod_evasive/mod_evasive20.lo
/narzedzia/mod_evasive/mod_evasive20.c && touch
/narzedzia/mod_evasive/mod_evasive20.slo
...
chmod 755 /usr/local/apache/modules/mod_evasive20.so
[activating module `evasive20' in /usr/local/apache/conf/httpd.conf]
# grep mod_evasive /usr/local/apache/conf/httpd.conf
LoadModule evasive20_module   modules/mod_evasive20.so
```

Jak działa moduł mod_evasive?

Kod modułu tworzy i analizuje dynamiczną tablicę skrótów wyznaczanych na podstawie adresu IP i ciągu URI każdego odebranego żądania. Gdy serwer odbiera nowe żądanie, moduł `mod_evasive` wykonuje następujące operacje.

- Sprawdza, czy adres IP klienta znajduje się na tymczasowej czarnej liście tablicy skrótów. Jeżeli tak, klient otrzymuje informację o odmowie dostępu — błąd `403-Forbidden`.
- Jeżeli adres IP jednostki klienckiej nie znajduje się na czarnej liście, adres IP nadawcy i identyfikator zasobu URI zostają poddane działaniu funkcji skrótu. Uzyskana wartość stanowi klucz w tablicy skrótów. Algorytm sprawdza, czy wyznaczona wartość była wcześniej zapisana w tablicy. Jeśli tak, określa liczbę wystąpień w danym przedziale czasu i porównuje z wartością progową zapisaną w pliku *httpd.conf* w dyrektywach `mod_evasive`.
- Jeżeli sprawdzenie nie prowadzi do odrzucenia żądania, adres IP jednostki klienckiej zostaje wykorzystany do wyznaczenia kolejnego skrótu. Następnie ponownie sprawdzana jest tablica skrótów. Jedyna różnica w porównaniu z poprzednio opisaną procedurą polega na tym, że nie jest uwzględniany ciąg URI. Celem zadania jest sprawdzenie, czy dany klient nie przekroczył dopuszczalnej liczby żądań w jednostce czasu dla całej witryny.

Jeżeli kryteria którejkolwiek z procedur sprawdzających zostaną spełnione, żądanie zostaje odrzucone, a klient otrzymuje standardową informację o błędzie `403-Forbidden`. Po odrzuceniu żądania połączenia z danej jednostki są ignorowane przez ustalony czas (domyślnie 10 sekund). Jeśli w tym okresie jednostka kliencka nie zaprzestanie przesyłania żądań, blokada połączeń zostaje przedłużona. Działanie algorytm `mod_evasive` zostało zilustrowane na rysunku 5.3.



Rysunek 5.3. *Algorytm mod_evasive*

Konfiguracja

Domyślne ustawienia modułu `mod_evasive` pozwalają na uruchomienie rozszerzenia bez potrzeby wprowadzania dodatkowych dyrektyw w pliku *httpd.conf*. W praktyce jednak rozwiązanie to często wymaga dostosowania do określonego środowiska — ustawienia odpowiednich wartości progowych. Z tego względu warto umieścić w pliku *httpd.conf* następujący blok dyrektyw:

```
<IfModule mod_evasive20.c>
   DOSHashTableSize 3097
   DOSPageCount 2
   DOSSiteCount 50
   DOSPageInterval 1
   DOSSiteInterval 1
   DOSBlockingPeriod 10
</IfModule>
```

Każda z wymienionych dyrektyw została opisana w dalszej części podrozdziału. Większość przytaczanych tu informacji pochodzi z pliku *README* pakietu *mod_evasive* i została przygotowana przez twórców modułu.

Dyrektywa DOSHashTableSize

Dyrektywa ta określa maksymalną liczbę węzłów najwyższego poziomu w tablicy skrótów dla każdego procesu potomnego. Zwiększanie wartości powoduje przyspieszenie działania mechanizmu z uwagi na zmniejszenie liczby iteracji, które trzeba wykonać, aby odczytać rekord. Kosztem jest jednak zwiększenie rozmiaru pamięci potrzebnej do przechowywania tablicy. W przypadku mocno obciążonych serwerów WWW wartość powinna być zwiększana. Podana wartość parametru zostanie automatycznie zmieniona na najbliższą liczbę pierwszą spośród zbioru wykorzystywanych liczb pierwszych (lista wykorzystywanych liczb pierwszych jest zapisana w pliku *mod_evasive.c*).

Dyrektywa DOSPageCount

Dyrektywa `DOSPageCount` wyznacza wartość progową dla liczby żądań kierowanych do tej samej strony (lub zasobu o określonym ciągu URI) w jednostce czasu, definiowanej za pomocą dyrektywy `DOSPageInterval`. Gdy liczba żądań generowanych przez daną jednostkę przekroczy wartość tego parametru, adres IP klienta zostanie dodany do listy blokowanych stacji.

Dyrektywa DOSSiteCount

Dyrektywa `DOSSiteCount` wyznacza wartość progową dla liczby żądań wygenerowanych przez jednego klienta i kierowanych do tego samego procesu nasłuchu (niezależnie od strony witryny) w jednostce czasu, definiowanej za pomocą dyrektywy `DOSSiteInterval`. Po przekroczeniu wartości progowej adres IP jednostki klienckiej zostaje dodany do listy blokowanych stacji.

Dyrektywa DOSPageInterval

Dyrektywa ta określa czas oczekiwania na przekroczenie progu `DOSPageCount`. Domyślna wartość odpowiada jednej sekundzie.

Dyrektywa DOSSiteInterval

Dyrektywa określa czas oczekiwania na przekroczenie progu `DOSSiteCount`. Domyślna wartość odpowiada jednej sekundzie.

Dyrektywa DOSBlockingPeriod

Dyrektywa `DOSBlockingPeriod` odpowiada za wyznaczanie czasu (w sekundach) ignorowania żądań stacji po umieszczeniu jej na liście blokowanych jednostek. Wszystkie nadchodzące w tym czasie żądania są odsyłane z informacją o błędzie `403`, a odbiór każdego żądania powoduje rozpoczęcie ponownego zliczania czasu (np. następnych dziesięciu sekund). Z uwagi na każdorazowe zerowanie zegara czas blokowania nie musi być długi. W przypadku ataku DoS zegar będzie ciągle resetowany.

Dyrektywa DOSEmailNotify

Jeżeli dyrektywa `DOSEmailNotify` zostanie zdefiniowana, informacja o każdym przypadku zarejestrowania adresu IP na czarnej liście zostanie przesłana na podany adres IP. Mechanizm blokad (korzystający z katalogu */tmp*) zabezpiecza administratora przed ciągłym wysyłaniem listów e-mail.

Przed skompilowaniem modułu należy sprawdzić, czy zawartej w pliku *mod_evasive.c* (lub *mod_evasive20.c*) stałej `MAILER` zostało przypisane odpowiednie polecenie przesłania poczty. Domyślnie jest to instrukcja `/bin/mail -t %s`, gdzie `%s` odpowiada docelowemu adresowi poczty, definiowanemu w pliku konfiguracyjnym. W przypadku używania innego systemu operacyjnego niż Linux lub stosowania innego programu pocztowego niż */bin/mail* konieczne jest odpowiednie dostosowanie wartości stałej.

Dyrektywa DOSSystemCommand

Jeżeli parametrowi `DOSSystemCommand` zostanie przypisane polecenie systemowe, będzie ono wykonywane automatycznie, po każdorazowym dopisaniu adresu IP do czarnej listy. Funkcja ta pozwala na systemowe odwołania do oprogramowania IP Filter lub do innych narzędzi. Mechanizm blokad (korzystający z katalogu */tmp*) chroni system przed ciągłymi wywołaniami. Symbol `%s` oznacza adres IP dopisany do czarnej listy.

Dyrektywa DOSLogDir

Dyrektywa pozwala na zdefiniowanie alternatywnego katalogu danych tymczasowych. Domyślnie mechanizm blokad wykorzystuje katalog */tmp*. Jeżeli system jest dostępny dla wielu użytkowników powłoki, takie rozwiązanie nie jest najbezpieczniejsze. Warto wówczas utworzyć osobny katalog i przypisać prawa modyfikacji tylko do konta, które jest skojarzone z serwerem Apache. Następnie w pliku *httpd.conf* trzeba umieścić dyrektywę `DOSLogDir` z odpowiednią wartością katalogu.

Dyrektywa WhiteListing

Od czasu opracowania wersji 1.8 modułu adresy IP mogą być również umieszczane na białej liście, która gwarantuje, że jednostki o tych adresach nigdy nie zostaną zablokowane. Opcja jest przeznaczona dla programów, skryptów, lokalnych wyszukiwarek i automatycznych narzędzi pobierających znaczne ilości danych z serwera. Nie należy jej jednak wykorzystywać do dodawania listy użytkowników, gdyż może to narazić serwer na atak. Wyzwolenie funkcji modułu nie jest wcale łatwe i wymaga przeprowadzenia rzeczywistego ataku. Dlatego decyzję o tym, czy dany użytkownik powinien zostać zablokowany, czy nie, bez obaw można pozostawić algorytmowi modułu.

Aby dodać adres lub pulę adresów do białej listy, trzeba wpisać w pliku konfiguracyjnym Apache następujące wiersze:

```
DOSWhitelist 127.0.0.1
DOSWhitelist 127.0.0.*
```

Symbole wieloznaczne mogą zastępować maksymalnie trzy ostatnie oktety adresu. Nic nie stoi na przeszkodzie, żeby w jednym pliku konfiguracyjnym występowało wiele dyrektyw `DOSWhitelist`.

Testowanie

Kod modułu `mod_evasive` jest rozpowszechniany wraz ze skryptem języka PERL o nazwie *test.pl*. Uruchomienie programu powoduje przesłanie stu żądań o inkrementowanych wartościach URL do lokalnego serwera (*localhost*) na port 80. Częstotliwość wysyłania żądań jest dostatecznie duża, by po około dwudziestu próbach moduł `mod_evasive` zablokował połączenia. Treść skryptu została przedstawiona poniżej.

```
#!/usr/bin/perl

# test.pl: small script to test mod_dosevasive's effectiveness

use IO::Socket;
use strict;

for(0..100) {
  my($response);
  my($SOCKET) = new IO::Socket::INET( Proto   => "tcp",
                                      PeerAddr=> "127.0.0.1:80");
  if (! defined $SOCKET) { die $!; }
  print $SOCKET "GET /?$_ HTTP/1.0\n\n";
  $response = <$SOCKET>;
  print $response;
  close($SOCKET);
}
```

Wynik wykonania skryptu powinien być zbliżony do pokazanego na kolejnym listingu.

```
# ./test.pl
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 200 OK
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
```

```
HTTP/1.1 403 Forbidden
HTTP/1.1 403 Forbidden
...
```

Podsumowanie modułu mod_evasive

Modułu `mod_evasive` okazuje się niezwykle skuteczny w zwalczaniu ataków DoS prowadzonych na małą i średnią skalę oraz w obronie przed atakami realizowanym metodami siłowymi. Chroni on system przed nadmiernym zużyciem pasma i uruchamianiem tysięcy procesów CGI (w wyniku ataku). Cecha ta zostanie wykorzystana w kolejnych rozdziałach książki, podczas analizowania zaawansowanych mechanizmów alarmowych. Połączenie funkcji modułu z innymi rozwiązaniami prewencyjnymi, takimi jak blokowanie nadawców na poziomie routera (ang. *router blackholing*), pozwala na walką również z atakami prowadzonymi na dużą skalę.

Jeżeli w danej infrastrukturze nie są zaimplementowane żadne mechanizmy obrony przed innymi rodzajami ataków DoS, narzędzie to może być przynajmniej pomocne w wyznaczeniu całkowitej przepustowości sieci i pojemności serwera. Niemniej bez odpowiednio przygotowanej infrastruktury i planu unikania ataków DoS zmasowany rozproszony atak DoS nadal pozostaje dużym zagrożeniem.

Moduł `mod_evasive` został opisany także w kolejnych rozdziałach książki. Umożliwia bowiem precyzyjne przetestowanie i sparametryzowanie dyrektyw odpowiedzialnych za utrzymanie wysokiej wydajności aplikacji. Poza tym w dalszej części książki zostaną przedstawione pewne modyfikacje kodu modułu, które podnoszą poziom zabezpieczeń serwera.

Moduł mod_security

Gdyby trzeba było wybrać jedno spośród różnych rozwiązań związanych z zabezpieczeniem serwera, wiele osób — w tym autor książki — wybrałoby właśnie moduł `mod_security`. Niezależnie od tego, czy konieczne jest utworzenie systemu wykrywania włamań WWW, czy aplikacji typu firewall WWW, moduł `mod_security` zawsze okaże się odpowiednim narzędziem. Pakiet `mod_rewrite` jest uznawany za niedoścignione rozwiązanie w zakresie przetwarzania ciągów URL. Moduł `mod_security` jest jego odpowiednikiem w dziedzinie zabezpieczania aplikacji WWW.

Kod `mod_security` został napisany przez Ivana Ristica i udostępniony na stronie *www.modsecurity.org*. Firma Ristica Thinking Stone (*www.thinkingstone.com*) oferuje również pomoc o charakterze komercyjnym. W czasie pisania książki biblioteka `mod_security` była dostępna w wersji 1.8.7. W dalszej części podrozdziału zostało przedstawionych wiele dyrektyw modułu, jednak najbardziej aktualnym źródłem informacji na ich temat jest dokumentacja dostępna w serwisie modsecurity *www.modescurity.org/documentation/index.html.*

Instalacja modułu mod_security

Instalacja modułu `mod_security` przebiega tak samo, jak w przypadku biblioteki `mod_evasive` — odpowiada za nią skrypt *apxs*. Aby sprawdzić, czy pakiet został zainstalowany, wystarczy użyć polecenia `grep`.

```
# grep security_module httpd.conf
LoadModule security_module modules/mod_security.so
```

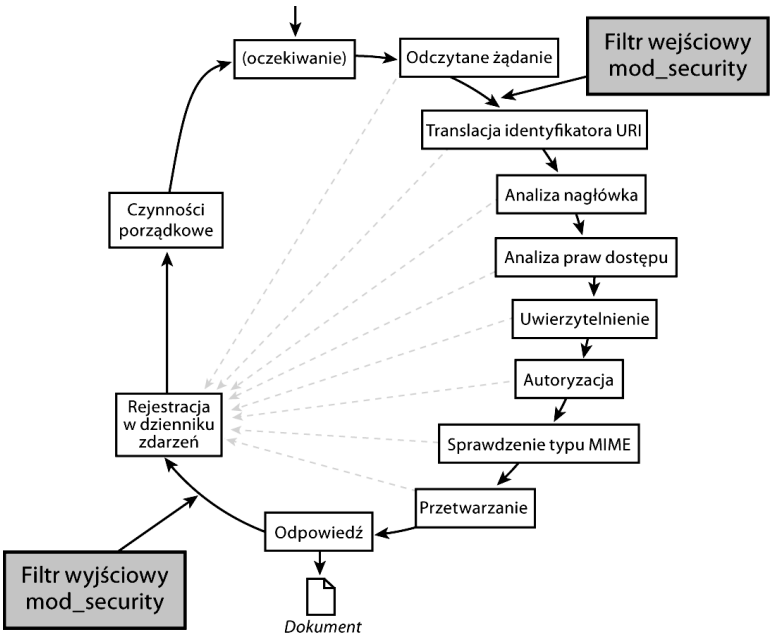
Ogólne informacje na temat modułu

Do najważniejszych cech modułu `mod_security` należy zaliczyć:

- **Filtrowanie żądań**. Nadchodzące żądania są poddawane analizie natychmiast po dostarczeniu do systemu, przed przekazaniem do serwera WWW lub innych modułów.
- **Techniki przeciwdziałania maskowaniu ataków**. W celu uniezależnienia mechanizmu od różnych technik maskowania ataków analiza żądania jest poprzedzana normalizacją parametrów i ścieżek dostępu.
- **Analiza danych protokołu HTTP**. Kod modułu został dostosowany do standardu HTTP, co umożliwia precyzyjne filtrowanie danych przesyłanych w tym protokole.
- **Analiza danych pola ładunkowego metody POST**. Analiza informacji obejmuje również dane użytkownika przekazywane do serwera za pomocą metody POST.
- **Rejestracja przebiegu komunikacji**. Wszystkie przesyłane informacje (wraz z polem metody POST) mogą być rejestrowane z przeznaczeniem do późniejszej weryfikacji.
- **Filtrowanie żądań HTTPS**. Z uwagi na fakt, że moduł jest osadzony w środowisku serwera WWW, ma dostęp do danych HTTPS po ich rozszyfrowaniu.

Funkcje modułu `mod_security` obejmują zadania przechwytywania i analizowania zarówno danych wejściowych (żądań klienckich), jak i wyjściowych (odpowiedzi serwera). Ich celem jest wykrycie przypadków ataku lub niestandardowej wymiany informacji. Umiejscowienie modułu `mod_security` w diagramie przetwarzania żądań serwera Apache zostało przedstawione na rysunku 5.4.

Na szczególną uwagę zasługuje fakt, że kod najnowszych wersji (1.9) modułu `mod_security` może korzystać z zerowego punktu zaczepienia (*hook 0*). Dzięki temu istnieje możliwość analizowania wszystkich żądań, włącznie z tymi, które normalnie są przetwarzane przez moduł jądra Apache. Są to żądania o treści niezgodnej ze standardem, których odbiór powoduje wygenerowanie błędu `400-Bad Request`.



Rysunek 5.4. *Przechwytywanie danych wejściowych i wyjściowych przez moduł mod_security*

Opcje i możliwości modułu mod_security

W module `mod_security` została zaimplementowana obsługa różnych dyrektyw, które odpowiadają za realizację wielu zadań związanych z bezpieczeństwem aplikacji. W dalszej części rozdziału zostanie przedstawionych kilka z najważniejszych. Pozostałe są tematem rozdziału dotyczącego zapobiegania atakom. Opis dyrektyw `mod_security` został w dużej części zaczerpnięty z doskonałego podręcznika, dostępnego w serwisie *modsecurity.org*.

Dyrektywa SecFilterEngine

Ustawienie to odpowiada za włączenie lub wyłączenie modułu. Domyślnie moduł jest wyłączony. Dopuszczalnymi wartościami parametrów są: `On`, `Off` (odpowiednio włączenie i wyłączenie).

```
SetFilterEngine On
```

Dyrektywa SecFilterScanPOST

Funkcja opisywana dyrektywą jest domyślnie wyłączona (`Off`). Jej włączenie (`On`) powoduje uaktywnienie skanowania pola danych HTTP metody POST, jeżeli zostało ono zakodowane zgodnie z jednym z wymienionych typów MIME:

- `application/x-www-form-urlencoded` — transfer danych formularza;
- `multipart/form-data` — przesłanie pliku do serwera.

Zalecana postać dyrektywy to:

```
SecFilterScanPOST On
```

Dyrektywa SecFilterScanOutput

Dyrektywa ta pozwala na włączenie opcji analizowania danych wyjściowych, generowanych po przetworzeniu żądania klienckiego. Ze względu na budowę interfejsu API Apache funkcja ta jest dostępna tylko w gałęzi 2.0 oprogramowania serwera. Mimo iż jej zasada działania jest zbliżona do zadań opcji `SecFilterScanPOST`, definiowane przez administratora filtry danych różnią się od tych, które są stosowane w odniesieniu do informacji wejściowych. Zagadnienie analizowania danych wyjściowych zostało opisane w kolejnym rozdziale. Składnia dyrektywy jest następująca:

```
SecFilterScanOutput On
```

Techniki przeciwdziałania maskowaniu ataków

Osoby przeprowadzające atak często starają się ukryć generowane przez siebie żądania przed sygnaturowymi systemami wykrywania włamań. W tym celu przekształcają w pewien sposób treść żądań. Poniżej zostało opisanych kilka przykładów operacji normalizacyjnych wykonywanych na żądaniach przed porównaniem ich z sygnaturą ataku.

- Usunięcie wielokrotnych znaków ukośnika.

  Przekształcenie ciągu // w /.

- Jednakowe traktowanie znaków ukośnika i odwrotnego ukośnika (tylko w systemie Windows).

  Przekształcenie znaku \ w znak / w systemie Windows.

- Usunięcie cyklicznych odwołań do katalogu.

  Skrócenie ciągu /./ do /.

- Wyszukanie i usunięcie pustych bajtów (`%00`).

  Usunięcie pustych bajtów (`null`) umożliwia przeprowadzenie standardowej analizy danych.

- Dekodowanie znaków zakodowanych w sposób charakterystyczny dla ciągów URL.

  Usunięcie kodowania znaków specjalnych pozwala na zastosowanie filtrów sygnaturowych.

Szczegółowe omówienie technik maskowania ataków HTTP zostało zamieszczone w rozdziale 9.

Specjalne wbudowane procedury sprawdzające

Kod modułu `mod_security` wykonuje kilka operacji weryfikacji kodowania tekstu i akceptuje jedynie znaki z określonych zbiorów. Poszczególne funkcje sprawdzające zostały opisane w dalszej części punktu.

Weryfikacja kodowania URL — dyrektywa SecFilterCheckURLEncoding

Podczas formowania pola identyfikatora zasobu (URI) niektóre znaki muszą zostać zakodowane w określonej formie. Wynika to z tego, że wiele metaznaków ma szczególne znaczenie w danym systemie i może być przyczyną błędów podczas próby interpretowania tekstu. Więcej informacji na ten temat zawiera dokument RFC 2396. Aby zatem znaki specjalne mogły być przekazywane do serwera WWW, trzeba je poddać kodowaniu zgodnie z formatem przewidzianym dla adresów URL. Każdy znak może być zastąpiony przez trzyznakowy symbol `%XY`, gdzie wartości `XY` odpowiadają szesnastkowemu kodowi danego znaku. Liczby szesnastkowe składają się z cyfr `0 – 9` oraz liter `a – f`. Na rysunku 5.5 znajduje się wykaz wszystkich znaków (o kodach od 0 do 255) oraz odpowiadających im kodów URL. Hakerzy i programy robaków często wykorzystują nadmiarowe kodowanie znaków w żądaniach. Dyrektywa `SecFilterCheckURLEncoding` włącza mechanizm sprawdzający, czy zastosowane kodowanie jest poprawne. Powinna ona mieć następującą treść:

```
SecFilterCheckURLEncoding On
```

Weryfikacja kodowania Unicode — dyrektywa SecFilterCheckUnicodeEncoding

Podobnie jak w przypadku opisanej wcześniej procedury sprawdzenia kodowania URL weryfikacja kodowania Unicode ma na celu ustalenie, czy użyte znaki spełniają kilka kryteriów charakterystycznych dla tego kodowania.

- **Niedostateczna liczba bajtów**. Standard UTF-8 pozwala na stosowanie kodowania z wykorzystaniem dwóch, trzech, czterech, pięciu lub sześciu bajtów. Moduł `mod_security` sprawdza, czy nie brakuje jednego lub większej liczby bajtów w kodowaniu danego znaku.
- **Błędne kodowanie**. Dwa najstarsze bity większości znaków powinny mieć taką wartość, aby bajt miał wartość `0x80`. Hakerzy wykorzystują ten fakt do wywoływania błędów w pracy dekoderów Unicode.
- **Zbyt długi kod znaku**. Znaki ASCII są odwzorowywane bezpośrednio w przestrzeni Unicode. Są więc reprezentowane za pomocą jednego bajta. Jednak większość znaków ASCII może być również opisywana dwoma, trzema, czterema, pięcioma lub sześcioma znakami. Takie kodowanie może wywołać błędną pracę dekodera, który może interpretować dany znak jako inną informację (a tym samym nie wykonać odpowiedniego sprawdzenia).

æ	%00	0	%30	`	%60		%90	À	%c0	ð	%f0
	%01	1	%31	a	%61	‘	%91	Á	%c1	ñ	%f1
	%02	2	%32	b	%62	’	%92	Â	%c2	ò	%f2
	%03	3	%33	c	%63	“	%93	Ã	%c3	ó	%f3
	%04	4	%34	d	%64	”	%94	Ä	%c4	ô	%f4
	%05	5	%35	e	%65	•	%95	Å	%c5	õ	%f5
	%06	6	%36	f	%66	–	%96	Æ	%c6	ö	%f6
	%07	7	%37	g	%67	—	%97	Ç	%c7	÷	%f7
Usunięcie znaku z lewej strony	%08	8	%38	h	%68	˜	%98	È	%c8	ø	%f8
Tabulacja	%09	9	%39	i	%69	™	%99	É	%c9	ù	%f9
Nowy wiersz	%0a	:	%3a	j	%6a	š	%9a	Ê	%ca	ú	%fa
	%0b	;	%3b	k	%6b	›	%9b	Ë	%cb	û	%fb
	%0c	<	%3c	l	%6c	œ	%9c	Ì	%cc	ü	%fc
Powrót na początek wiersza	%0d	=	%3d	m	%6d		%9d	Í	%cd	ý	%fd
	%0e	>	%3e	n	%6e	ž	%9e	Î	%ce	þ	%fe
	%0f	?	%3f	o	%6f	Ÿ	%9f	Ï	%cf	ÿ	%ff
	%10	@	%40	p	%70		%a0	Ð	%d0		
	%11	A	%41	q	%71	¡	%a1	Ñ	%d1		
	%12	B	%42	r	%72	¢	%a2	Ò	%d2		
	%13	C	%43	s	%73	£	%a3	Ó	%d3		
	%14	D	%44	t	%74		%a4	Ô	%d4		
	%15	E	%45	u	%75	¥	%a5	Õ	%d5		
	%16	F	%46	v	%76	¦	%a6	Ö	%d6		
	%17	G	%47	w	%77	§	%a7		%d7		
	%18	H	%48	x	%78	¨	%a8	Ø	%d8		
	%19	I	%49	y	%79	©	%a9	Ù	%d9		
	%1a	J	%4a	z	%7a	ª	%aa	Ú	%da		
	%1b	K	%4b	{	%7b	«	%ab	Û	%db		
	%1c	L	%4c	|	%7c	¬	%ac	Ü	%dc		
	%1d	M	%4d	}	%7d		%ad	Ý	%dd		
	%1e	N	%4e	~	%7e	®	%ae	Þ	%de		
	%1f	O	%4f		%7f	¯	%af	ß	%df		
Spacja	%20	P	%50	€	%80	°	%b0	à	%e0		
!	%21	Q	%51		%81	±	%b1	á	%e1		
"	%22	R	%52	‚	%82	²	%b2	â	%e2		
#	%23	S	%53	ƒ	%83	³	%b3	ã	%e3		
$	%24	T	%54	„	%84	´	%b4	ä	%e4		
%	%25	U	%55	…	%85	µ	%b5	å	%e5		
&	%26	V	%56	†	%86	¶	%b6	æ	%e6		
'	%27	W	%57	‡	%87	·	%b7	ç	%e7		
(	%28	X	%58	ˆ	%88	¸	%b8	è	%e8		
)	%29	Y	%59	‰	%89	¹	%b9	é	%e9		
*	%2a	Z	%5a	Š	%8a	º	%ba	ê	%ea		
+	%2b	[	%5b	‹	%8b	»	%bb	ë	%eb		
,	%2c	\	%5c	Œ	%8c	¼	%bc	ì	%ec		
-	%2d	]	%5d		%8d	½	%bd	í	%ed		
.	%2e	^	%5e	Ž	%8e	¾	%be	î	%ee		
/	%2f	_	%5f		%8f	¿	%bf	ï	%ef		

Rysunek 5.5. *Kodowanie URL*

Zalecana treść dyrektywy to:

```
SecFilterCheckUnicodeEncoding On
```

Weryfikacja zakresu wartości bajta — dyrektywa SecFilterForceByteRange

Pojedyncze znaki mogą być zapisywane w różnych standardach — ASCII, w kodowaniu URL, kodowaniu Unicode itd. Dodatkowym sposobem notacji jest zapis dziesiętny, wykorzystujący jeden bajt (dwu- lub trzycyfrową wartość). Zestawienie właściwe dla kodowania URL (przedstawione na rysunku 5.5) można przekształcić w zestawienie znaków kodowanych dziesiętnie (rysunek 5.6).

æ	00	0	48	`	96		144	À	192	ð	240
	01	1	49	a	97	‘	145	Á	193	ñ	241
	02	2	50	b	98	’	146	Â	194	ò	242
	03	3	51	c	99	“	147	Ã	195	ó	243
	04	4	52	d	100	”	148	Ä	196	ô	244
	05	5	53	e	101	•	149	Å	197	õ	245
	06	6	54	f	102	–	150	Æ	198	ö	246
	07	7	55	g	103	—	151	Ç	199	÷	247
Usunięcie znaku z lewej strony	08	8	56	h	104	˜	152	È	200	ø	248
Tabulacja	09	9	57	i	105	™	153	É	201	ù	249
Nowy wiersz	10	:	58	j	106	š	154	Ê	202	ú	250
	11	;	59	k	107	›	155	Ë	203	û	251
	12	<	60	l	108	œ	156	Ì	204	ü	252
Powrót na początek wiersza	13	=	61	m	109		157	Í	205	ý	253
	14	>	62	n	110	ž	158	Î	206	þ	254
	15	?	63	o	111	Ÿ	159	Ï	207	ÿ	255
	16	@	64	p	112		160	Ð	208		
	17	A	65	q	113	¡	161	Ñ	209		
	18	B	66	r	114	¢	162	Ò	210		
	19	C	67	s	115	£	163	Ó	211		
	20	D	68	t	116		164	Ô	212		
	21	E	69	u	117	¥	165	Õ	213		
	22	F	70	v	118	¦	166	Ö	214		
	23	G	71	w	119	§	167		215		
	24	H	72	x	120	¨	168	Ø	216		
	25	I	73	y	121	©	169	Ù	217		
	26	J	74	z	122	ª	170	Ú	218		
	27	K	75	{	123	«	171	Û	219		
	28	L	76	|	124	¬	172	Ü	220		
	29	M	77	}	125		173	Ý	221		
	30	N	78	~	126	®	174	Þ	222		
	31	O	79		127	¯	175	ß	223		
Spacja	32	P	80	€	128	°	176	à	224		
!	33	Q	81		129	±	177	á	225		
"	34	R	82	‚	130	²	178	â	226		
#	35	S	83	ƒ	131	³	179	ã	227		
$	36	T	84	„	132	´	180	ä	228		
%	37	U	85	…	133	µ	181	å	229		
&	38	V	86	†	134	¶	182	æ	230		
'	39	W	87	‡	135	·	183	ç	231		
(	40	X	88	ˆ	136	¸	184	è	232		
)	41	Y	89	‰	137	¹	185	é	233		
*	42	Z	90	Š	138	º	186	ê	234		
+	43	[	91	‹	139	»	187	ë	235		
,	44	\	92	Œ	140	¼	188	ì	236		
-	45	]	93		141	½	189	í	237		
.	46	^	94	Ž	142	¾	190	î	238		
/	47	_	95		143	¿	191	ï	239		

Rysunek 5.6. *Kodowanie dziesiętne — zakres bajtowy 0 – 255*

Sprawdzanie zakresu wartości bajta jest istotne w przypadku wielu ataków bazujących na metodzie przepełnienia bufora. Przesyłane informacje zawierają wówczas dane binarne, których celem jest uruchomienie odpowiedniego kodu powłoki (shellcode'u). W jednym z kolejnych rozdziałów zostało przedstawione zastosowanie tej dyrektywy do obrony przed atakami wykorzystującymi przepełnienie bufora. Zdefiniowanie dyrektywy tak jak w poniższym przykładzie gwarantuje akceptację jedynie znaków z „przedziału" ograniczanego znakami spacji i tyldy (~).

```
SecFilterForceByteRange 32 126
```

Reguły filtrowania

Po wykonaniu wbudowanych procedur sprawdzających moduł mod_security analizuje filtry zdefiniowane przez administratora. Kryteria dopasowania reguły do dowolnej porcji danych są wyznaczane za pomocą wyrażeń regularnych. Sprawdzenie obejmuje nagłówki HTTP wraz z polem danych metody POST. Oto kilka cech charakterystycznych procesu:

- Użytkownik może zdefiniować dowolną liczbę reguł.
- Reguły są zapisywane w formie wyrażeń regularnych.
- Obsługiwane są wyrażenia negowane (odwrotne).
- Każdemu kontenerowi (VirtualHost, Location itp.) można przypisać inne ustawienia.
- Analiza obejmuje nagłówki HTTP.
- Analiza obejmuje dane cookie.
- Analiza obejmuje zmienne środowiskowe.
- Analiza obejmuje zmienne serwera.
- Analiza obejmuje zmienne strony.
- Analiza obejmuje pole danych metody POST.
- Analiza obejmuje dane wyjściowe skryptu.

Zasady filtrowania regulują dwie dyrektywy: SecFilter i SecFitlerSelective. Ustawienie SecFilter odpowiada podstawowemu (ogólnemu) filtrowi. Składnia instrukcji jest następująca:

```
SecFilter Słowo_kluczowe [akcja]
```

Wartość Słowo_kluczowe jest wyrażeniem regularnym. Moduł mod_security dokonuje analizy pierwszego wiersza żądania, poszukując w nim ciągu Słowo_kluczowe. Jeżeli taki ciąg zostanie znaleziony, opcjonalnie może zostać wykonana określona akcja (opisana w kolejnym punkcie). Jeżeli żadna akcja nie zostanie zdefiniowana, program wykorzysta regułę wyznaczającą akcję domyślną SecFilterDefaultAction. Dyrektywę SecFilterSelective cechuje mniejszy obszar poszukiwania. Wymaga ona precyzyjnego zdefiniowania obszaru żądania, który będzie przeszukiwany. Składania instrukcji jest następująca:

```
SecFilterSelective Obszar Słowo_kluczowe [akcja]
```

Parametr Obszar może się składać z kilku nazw pól nagłówka, które powinny zostać sprawdzone. Nazwy pól są zbliżone do tych, które są stosowane w module mod_rewrite. Choć jest również kilka dodatkowych, istotnie zwiększających efektywność modułu. W praktyce bardzo użyteczne okazują się reguły negowane, dlatego kolejny przykład dotyczy sposobu zapisu tego typu reguł. Jest on zbliżony do prezentowanego wcześniej, jednak zapis obszaru przeszukiwania i (lub) słowa kluczowego musi być poprzedzony znakiem wykrzyknika (!).

```
SecFilterSelective HTTP_USER_AGENT "!Przegladarka_testowa123"
```

Zastosowanie takiej reguły oznacza odrzucenie żądania każdego użytkownika, który nie korzysta z aplikacji o nazwie `Przegladarka_testowa123`. Opisywana funkcja jest niezwykle użyteczna podczas implementowania bardzo złożonych mechanizmów zabezpieczających.

Akcje

Jeżeli żądanie spełni kryteria dopasowania filtru, moduł `mod_security` automatycznie wykonuje pewną zdefiniowaną czynność (akcję). Akcje są podzielone na trzy kategorie: podstawowe, wtórne i sterujące.

Akcje podstawowe

Od definicji akcji podstawowej zależy to, czy po spełnieniu kryteriów filtru żądanie będzie dalej przetwarzane, czy zostanie odrzucone. Każdemu filtrowi można przypisać tylko jedną akcję podstawową. Jeśli zostanie zdefiniowana większa liczba akcji podstawowych, obowiązującą będzie ostatnia na liście. Nazwy czterech podstawowych akcji wraz z krótkim opisem zostały wymienione poniżej.

- `Deny` — dyrektywa ta powoduje natychmiastowe przerwanie przetwarzania żądania i wygenerowanie odpowiedzi o kodzie błędu 500, chyba że została zastosowana również dyrektywa statusu.
- `Pass` — pozwala na kontynuowanie procesu przetwarzania żądania i porównywanie go z kolejnymi regułami.
- `Allow` — działa podobnie jak `Pass`, ale nie pozwala na porównywanie żądania z kolejnymi regułami.
- `Redirect` — kieruje żądanie klienckie spełniające kryteria dopasowania pod określony adres URL.

Akcje wtórne

Akcje wtórne są wykonywane niezależnie od akcji podstawowych. Istnieje pięć tego typu akcji.

- `Status` — dyrektywa ta umożliwia nadpisanie domyślnego kodu statusowego o wartości 500. Dzięki niej administrator może przypisać odpowiedzi dowolny kod statusowy. Kod ten będzie również wykorzystany przez dyrektywy `ErrorDocument` zapisane w pliku konfiguracyjnym *httpd.conf*.
- `Exec` — dyrektywa ta pozwala na uruchomienie programu w przypadku zgodności żądania z regułą dopasowania. Definiując program, trzeba podać pełną ścieżkę dostępu do pliku, bez parametrów. Można stosować zmienne środowiskowe CGI.
- `Log` — dyrektywa ta stanowi dla modułu `mod_security` rozkaz rejestrowania żądań w dzienniku błędów określonym za pomocą dyrektywy `ErrorLog`.

- `Nolog` — dyrektywa ta uniemożliwia zarejestrowanie żądania w dzienniku błędów, nawet jeśli zgadza się ono z podanym kryterium.
- `Pause` — dyrektywa wymusza wstrzymanie generowania odpowiedzi na określoną wartość czasu. Okres przerwy jest definiowany w milisekundach, zatem zapis `pause:5000` oznacza wstrzymanie pracy na 5 sekund. Ustawienie to przydaje się do opóźnienia działania skanerów sieciowych. Liczne testy wykazują, że niektóre aplikacje skanerów w ogóle przestają działać, gdy opóźnienie osiąga pewną wartość.

Akcje sterujące

Akcje sterujące określają sposób doboru filtrów dla danego żądania. Podczas normalnej pracy po nadejściu żądania jest ono poddawane normalizacji, czyli sprawdzeniu kodowania URL itp. Następnie kod modułu `mod_security` porównuje treść żądania z kolejnymi regułami filtrów, zgodnie z kolejnością ich występowania w pliku *httpd.conf* lub pliku włączanym. Zatem kolejność definiowania reguł jest bardzo istotna. Aby zredukować liczbę błędnie dopasowanych żądań, reguły ogólne oraz te o najszerszym obszarze zgodności powinny być wymieniane na końcu zestawienia. Filtry o precyzyjnie ustalonych kryteriach dopasowania powinny z kolei znajdować się w początkowej części listy. Koncepcja definiowania reguł jest zbliżona do rozwiązań stosowanych w wielu aplikacjach firewall. Jednak nawet ustalony porządek analizy reguł nie gwarantuje dostatecznej elastyczności mechanizmu. Dlatego funkcje modułu zostały uzupełnione o dyrektywy `chain` i `skipnext`.

- `Chain` — dyrektywa ta umożliwia łączenie reguł w łańcuchy (ang. *chain*) w celu definiowania bardziej złożonych testów, precyzyjnie dopasowanych do określonego rodzaju żądań. Opcja budowania łańcuchów reguł istotnie zmniejsza liczbę przypadków błędnego dopasowania żądań.
- `Skipnext` — dyrektywa ta wymusza pominięcie jednej lub większej liczby kolejnych reguł. Jest to następny sposób, poza budowaniem łańcuchów, na zmniejszenie liczby przypadków błędnego dopasowania żądań. Jeżeli wśród zdefiniowanych reguł występuje taka, której kryteria dopasowania są bardzo ogólne, zawsze można ją poprzedzić filtrem o dokładniej sprecyzowanym wzorcu dopasowania, którego akcja będzie miała wartość `skipnext`. Aby pominąć kilka reguł, należy zastosować składnię pokazaną poniżej.

```
SecFilterSelective Arg_username "jkowalski12" skipnext:2
SecFilterSelective Arg_username "jkowalski1"
SecFilterSelective Arg_username "jkowalski"
```

Dyrektywa SecFilterDefaultAction

Dyrektywa `SecFilterDefaultAction` wyznacza domyślną akcję dla wszystkich reguł. Jeżeli parametr akcji konkretnego filtru nie zostanie nadpisany, obowiązuje akcja domyślna. Ustawienie wykorzystane w kolejnym przykładzie przekazuje do modułu `mod_security` informację o konieczności odrzucenia żądania, zarejestrowania danych żądania w dzienniku błędów i wygenerowania odpowiedzi o kodzie błędu 403.

```
SecFilterDefaultAction "deny,log,status:403"
```

Z kolei akcja zdefiniowana w sposób pokazany poniżej powoduje wstrzymanie pracy na 5 sekund, a następnie przekazanie żądania do innej witryny i uruchomienie programu.

```
SecFilterDefaultAction
"redirect:http://192.168.1.1,pause:50000,exec:/bin/program.sh"
```

Weryfikacja przesyłanych plików

Moduł `mod_security` udostępnia opcję analizowania i zapisywania plików przesyłanych do serwera.

- `SecUploadKeepFiles` — dyrektywa odpowiada za przechwytywanie plików przesyłanych do serwera.
- `SecUploadDir` — dyrektywa odpowiada za zapisanie pliku na dysku.
- `SecUploadApproveScript` — dyrektywa odpowiada za uruchomienie zewnętrznego skryptu, ustalającego, czy plik zostanie zaakceptowany, czy odrzucony (np. przez program antywirusowy).

Rejestrowanie zdarzeń

W module `mod_security` zostały zaimplementowane niezwykle użyteczne funkcje rejestracji zdarzeń. Działanie mechanizmu jest definiowane dyrektywą `SecFilterAuditEngine`, która jest niezależna od dyrektywy `SecFilterEngine`. Dzięki temu istnieje możliwość rejestrowania zdarzeń bez nakładania jakichkolwiek filtrów. Takie rozwiązanie często bywa użyteczne, gdy administrator musi zgromadzić pewną ilość informacji, zanim przystąpi do definiowania filtrów. Działanie mechanizmu rejestracji jest regulowane dwoma poniższymi dyrektywami.

```
SecAuditEngine On
SecAuditLog logs/audyt.log
```

Po ich wprowadzeniu moduł `mod_security` będzie zapisywał dane wszystkich żądań w pliku *audyt.log* w katalogu dzienników pracy serwera. Format wpisów w dzienniku jest następujący:

```
============================================================
Żadanie: adres_IP_zdalny użytkownik_zdalny użytkownik_lokalny [aktualny_czas]
\"treść_żądania\" status liczba_bajtów
Procedura_przetwarzania: cgi-script/proxy-server/null
Błąd: Komunikat o błędzie
------------------------------------------------------------
Wiersz URL żądania
Nagłówki klienckie
Kod statusowy odpowiedzi serwera
Nagłówki odpowiedzi serwera
============================================================
```

Kolejny listing przedstawia przykładowe wpisy z pliku *audyt.log*, w których procedura rejestracji została wyzwolona po spełnieniu kryteriów dopasowania filtra chroniącego plik */etc/passwd*.

```
--------------------------------------------------------------------
UNIQUE_ID: gYV8wH8AAAEAAHYBBFEAAAAA
Request: 192.168.26.1 - - [21/Apr/2005:10:53:34 --0400] "GET /etc/passwd HTTP/1.1"
403
729
Handler: cgi-script
--------------------------------------------------------------------
GET /etc/passwd HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR
1.1.4322)
Host: 192.168.26.134
Connection: Keep-Alive
mod_security-message: Access denied with code 403. Pattern match "/etc/passwd" at
THE_REQUEST.
mod_security-action: 403
HTTP/1.1 403 Forbidden
Content-Length: 729
Keep-Alive: timeout=15, max=500
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1
====================================================================
```

Pozostałe funkcje

Moduł `mod_security` udostępnia administratorom jeszcze wiele bardzo użytecznych funkcji, o których warto wspomnieć, ponieważ będą wykorzystywane w dalszych rozdziałach książki.

Maskowanie serwera — dyrektywa SecServerSignature

Dyrektywa ta pozwala na zmianę informacji zwracanej przez serwer w nagłówku HTTP `Server`. Aby zmodyfikować treść nagłówka, wystarczy dodać odpowiedni wiersz do pliku *httpd.conf*, bez potrzeby edytowania kodu źródłowego i ponownego kompilowania oprogramowania. Oto składnia dyrektywy:

```
SecServerSignature "Microsoft-IIS/5.0"
```

Trzeba jednak pamiętać, że dodanie opcji tylko nieznacznie podnosi poziom zabezpieczeń. W ten sposób można oszukać jedynie aplikacje analizujące „komunikaty powitalne" serwerów. Niemniej mimo iż nie jest to idealny mechanizm zabezpieczający, zapewnia pewną ochronę przed robakami, których działanie bazuje na weryfikacji treści pola `Server`.

Wewnętrzny mechanizm chroot

W rozdziale 2. zostały omówione zagadnienia związane z przygotowaniem oprogramowania Apache do uruchomienia w środowisku chroot. Każdy, kto próbował zaimplementować opisane rozwiązania, wie, że ze względu na konieczność spełnienia wszystkich zależności między programami realizacja zadania jest niezwykłym wyzwaniem.

Na szczęście moduł mod_security udostępnia dyrektywę, która pozwala na uruchomienie serwera Apache w środowisku chroot bez większych komplikacji. Łatwość implementacji wynika z faktu, że funkcja chroot została wbudowana w moduł mod_security i jest wywoływana bezpośrednio przed powołaniem procesów potomnych Apache. Wcześniej serwer ładuje wszystkie niezbędne biblioteki, inicjuje pracę modułów i otwiera pliki dzienników. Oznacza to, że pliki nie muszą być przenoszone do specjalnie wydzielonych katalogów, tak jak ma to miejsce w przypadku standardowego mechanizmu chroot. Zadanie administratora sprowadza się jedynie do włączenia dyrektywy w pliku *httpd.conf*.

```
SecChrootDir /ścieżka/do/chroot
```

Plikami, które muszą być zapisywane w katalogu */ścieżka/do/chroot*, są przede wszystkim pliki dokumentów witryny. Poza nimi w katalogu tym należy przechowywać wszystkie pliki, które mogą w jakikolwiek sposób być wykorzystywane w działaniu witryny — pliki związane z działaniem skryptów CGI czy pliki uwierzytelniania tworzone przez narzędzia htpasswd i htdigest (są one odczytywane podczas realizacji każdego żądania).

Podsumowanie modułu mod_security

Po zapoznaniu się z opisem modułu mod_security nikt chyba nie ma wątpliwości, że jest on niezwykle użyteczny. Gdyby jednak takowe wątpliwości się nasuwały, zostaną rozwiane po przeczytaniu książki. Moduł charakteryzuje się niezwykłą elastycznością identyfikowania ataków i alarmowania o wykrytych atakach. Dla wielu osób odpowiedzialnych za zabezpieczanie serwerów jest to podstawowe narzędzie pracy — gwarantuje odpowiednią ochronę serwera i spokojny sen administratora.

Podsumowanie

Celem rozdziału było przedstawienie kilku dodatkowych modułów Apache, które mogą rozwiązywać pewne problemy związane z bezpieczeństwem serwera. Celowo nie została tu przedstawiona pełna dokumentacja poszczególnych pakietów. Jest ona dostępna na stronach modułów i na tych stronach należy szukać szczegółowych danych.

Niemniej ilość przedstawionych informacji jest wystarczająca, aby Czytelnik mógł zapoznać się z przeznaczeniem i dyrektywami każdego modułu. W dalszej części książki będą opisywane różne rozwiązania systemu zabezpieczeń, bazujące na narzędziach i koncepcjach przedstawionych w tym rozdziale.