

IX Seminarium PLOUG  
Warszawa  
Maj 2004

# **Zabezpieczanie aplikacji internetowych za pomocą mod\_security**

Wojciech Dworakowski

*SecuRing*  
*wojciech.dworakowski@securing.pl*



Aplikacje internetowe – czyli aplikacje udostępniające interfejs użytkownika za pomocą przeglądarki internetowej – to technologia która zrewolucjonizowała sposób tworzenia aplikacji. Niestety – jak z każdą nową technologią, również i z tą wiążą się nowe niebezpieczeństwa (przeгляд technik ataku był prezentowany na zeszłorocznym Seminarium). Do zagrożeń tych możemy zaliczyć m.in.:

- Brak obsługi błędów i sytuacji specjalnych
- Manipulacje parametrami
- Path traversal
- Wykonywanie komend systemu operacyjnego
- SQL injection
- Ataki na sesje (cross-site scripting i session fixation)

W niektórych wypadkach (np. skrypty CGI) aplikacje internetowe mogą być podatne na klasyczne podatności takie jak „buffer overflow” czy „format strings”.

Na domiar złego źródłem zagrożeń może być samo środowisko w jakim działa aplikacja. Zarówno oprogramowanie serwera WWW, oprogramowanie serwera aplikacyjnego jak i moduły pomocnicze służące do szybkiego tworzenia aplikacji webowych (np. Oracle Portal czy PHPNuke) mogą być podatne na różne metody ataku.

Problem polega na tym że protokołem w którym są przenoszone wszystkie te typy ataków jest protokół HTTP. Ze względu na to, tradycyjne środki ochrony (np. firewalles) nie są w stanie zapewnić właściwego poziomu ochrony, gdyż zarówno atak jak i zwyczajne korzystanie z aplikacji są widziane przez nie jako ruch HTTP (lub HTTPS).

Żeby skutecznie chronić się przed atakami na aplikacje, powstało zapotrzebowanie na firewalles aplikacyjne, czyli filtry działające na wysokich warstwach modelu OSI, rozumiejące protokoły aplikacyjne, w szczególności – protokół HTTP.

Na zeszłorocznym Seminarium PLOUG wspominałem m.in. o projekcie Open Source – CodeSeeker. Niestety okazało się, że po uwolnieniu kodu projekt ten przestał być rozwijany. Poza tym posiadał pewne znaczące niedociągnięcia. Z korespondencji z uczestnikami zeszłorocznego Seminarium wiem, że temat filtrów aplikacyjnych spotkał się z zainteresowaniem i kilka osób testowało CodeSeeker (bez sukcesów). W związku z tym – na tegorocznym spotkaniu chcę przedstawić mod\_security. Projekt, który w międzyczasie wyrósł na niekwestionowanego lidera rozwiązań typu firewall aplikacyjny w światku Open Source.

Mod\_security jest modułem integrującym się z serwerem HTTP Apache, który pozwala na szczegółowe filtrowanie ruchu w zależności od zawartości danych przekazywanych w protokole HTTP. Apache jest serwerem HTTP stosowanym przez Oracle Application Server (Oracle HTTP Server), tak więc mod\_security można zintegrować bezpośrednio z rozwiązaniami Oracle. Alternatywnym (i chyba lepszym) rozwiązaniem jest zastosowanie tego modułu na osobnym serwerze pełniącym rolę firewalles aplikacyjnego. Obie te konfiguracje opisane zostaną szczegółowo w dalszej części. Projekt ten jest rozwijany od roku 2002 i jego rozwój przebiega bardzo dynamicznie. Obecnie jest przygotowywana wersja 1.8. Pozwala to ocenić ten projekt jako dojrzały i stabilny – gotowy do stosowania w instalacjach produkcyjnych.

Oprogramowanie jest dostępne na stronie: [www.modsecurity.org](http://www.modsecurity.org)

## 1. Konfiguracja

Mod\_security dodaje do konfiguracji Apache (httpd.conf) kilka dodatkowych dyrektyw. Wszystkie wpisy związane z mod\_security zaczynają się literami „Sec”.

Kilka komend ogólnych:

Włączenie filtra:

```
SecFilterEngine On
```

Włączenie inspekcji zawartości zleceń POST:

```
SecFilterScanPOST On
```

## Sprawdzanie kodowania URL:

```
SecFilterCheckURLEncoding On
SecFilterCheckUnicodeEncoding On
```

## Mod\_security posiada dość rozbudowane możliwości reakcji:

- **deny** – zablokowanie ruchu
- **allow** – przepuszczenie ruchu (nie są rozpatrywane dalsze reguły)
- **status:nnn** – odpowiedź HTTP status nnn (np. 404 – not found)
- **redirect:url** – przekierowanie do innego URL
- **exec:cmd** – wykonanie komendy systemu operacyjnego
- **log** – zapisanie zdarzenia do logów  
Zdarzenie jest traktowane przez Apache jako błąd a więc standardowo jest zapisywane w error\_log.
- **nolog** – brak zapisania w logach (przydatne przy łączeniu reguł)
- **pass** – zignorowanie danej regułki (w ciągu reguł)
- **pause:nnn** – zatrzymanie zlecenia na nnn milisekund.  
Może być przydatne np. przy blokowaniu ataków DDoS ale należy tego używać bardzo ostrożnie gdyż sztucznie wprowadzone opóźnienia mogą spowodować zakłócenia w działaniu aplikacji. Nietypowym zastosowaniem tej właściwości może być symulowanie opóźnień w środowiskach testowych.

Akcje można łączyć. Tzn. do jednej regułki można przypisać wiele działań.

Można zdefiniować reakcję standardową, która będzie podejmowana o ile do danej regułki nie zostanie przypisana inna akcja. Przykład definicji reakcji standardowej:

```
SecFilterDefaultAction "deny,log,status:404"
```

Powyższy zapis oznacza reakcję standardową polegającą na zablokowaniu zlecenia HTTP, zalogowaniu zdarzenia oraz odpowiedzi komunikatem HTTP 404 (not found).

Podstawowe regułki filtrowania można zdefiniować za pomocą składni skróconej:

```
SecFilter SŁOWO_KLUCZOWE [ AKCJA ]
```

Regułka SecFilter działa w ten sposób, że wyszukuje SŁOWO\_KLUCZOWE w pierwszej linii zlecenia HTTP (czyli np. GET /index.jsp HTTP/1.1) oraz w danych zlecenia POST. Jeżeli słowo kluczowe zostanie znalezione to zostanie wykonana przypisana AKCJA (lub akcja standardowa jeśli do regułki nie została przypisana akcja). Tak więc składnię skróconą SecFilter można zastosować do poszukiwania słów kluczowych w ciągu URI oraz parametrach zleceń GET i POST. W przypadkach gdy te możliwości inspekcji zlecenia HTTP są niewystarczające, można użyć składni pełnej:

```
SecFilterSelective ZMIENNA SŁOWO_KLUCZOWE [ AKCJA ]
```

Składnia pełna daje dostęp nie tylko do pierwszej linii zlecenia HTTP i parametrów POST ale również do nagłówek i pozostałych części zlecenia HTTP. Pozwala też na wygodniejsze operowanie parametrami przekazywanymi w zleceniach GET i POST. W stosunku do składni podstawowej mamy tu możliwość zdefiniowania zmiennej która zostanie przeszukana.

Przykładowe zmienne:

```
REMOTE_ADDR – adres IP klienta
REMOTE_HOST – nazwa klienta
REQUEST_METHOD – metoda HTTP (np. GET, POST, HEAD)
REQUEST_URI – ciąg URI
AUTH_TYPE – sposób uwierzytelnienia HTTP
```

SERVER\_NAME – nazwa serwera (przydatne w przypadku serwerów wirtualnych gdy jeden Apache obsługuje wiele serwerów)  
 TIME, TIME\_YEAR, TIME\_MON, TIME\_DAY, itd – czas  
 POST\_PAYLOAD – zawartość zlecenia POST (parametry)  
 ARGS – wszystkie parametry (ze zlecenia POST i po znaku ? w zleceniu GET)

Można również używać zmiennych specjalnych:

HTTP\_nazwa – nagłówek HTTP „nazwa” np. HTTP\_USER\_AGENT – nagłówek User-agent (nazwa przeglądarki)  
 ARG\_nazwa – konkretny argument zlecenia GET/POST. Np. ARG\_UZYTKOWNIK – parametr „uzytkownik” przekazywany z formularza  
 COOKIE\_nazwa – zmienna przekazywana w cookie. Np. COOKIE\_USER\_ID – zmienna user\_id przekazana z cookie użytkownika.

Zmienne można łączyć za pomocą operatora | oraz negować (wprowadzać wyjątki) za pomocą operatora ! Przykładowo: ARGS!ARG\_nazwa spowoduje przeszukanie wszystkich argumentów z wyjątkiem argumentu „nazwa”.

Bardzo dużą zaletą mod\_security jest możliwość definiowania słów kluczowych za pomocą wyrażeń regularnych w składni PCRE (Perl Compatible Regular Expressions). Daje to bardzo rozbudowane możliwości definiowania słów kluczowych. Wyrażenia regularne są standardem wywodzącym się ze świata unixów. Są stosowane jako standard opisu słów kluczowych m.in. w takich narzędziach systemowych jak *grep*, *sed*, *awk* czy *perl*. Również Oracle w wersji 10g wprowadziło wyrażenia regularne do składni SQL i PL/SQL<sup>1</sup>.

Więcej informacji o wyrażeniach regularnych można znaleźć np pod adresami:

<http://www.pcre.org>

<http://etext.lib.virginia.edu/helpsheets/regex.html>

lub w dokumentacji Oracle 10g.

Na koniec tego wprowadzenia do pisania reguł warto jeszcze wspomnieć że reguły mod\_security można łączyć w ciągi. Robi się to za pomocą słówka chain:

```
SecFilterSelective reguła1 chain SecFilterSelective reguła2
```

## 2. Przykładowe reguły filtrowania

Poniżej przedstawiam kilka przykładów reguł podnoszących bezpieczeństwo serwera WWW i chroniących aplikacje.

W poniższych przykładach założono że standardową akcją jest odrzucenie zlecenia, a więc dopasowanie wyrażenia powoduje zablokowanie zlecenia.

- Zabezpieczenie przed próbami uruchamiania komend systemu operacyjnego

```
SecFilter "/bin/*.sh"  
SecFilter cmd.exe
```

- Zabezpieczenie przed atakami typu path-traversal.

```
SecFilter "\\.\."/
```

<sup>1</sup> Writing Better SQL Using Regular Expressions:

[http://otn.oracle.com/oramag/webcolumns/2003/techarticles/rischert\\_regex\\_pt1.html](http://otn.oracle.com/oramag/webcolumns/2003/techarticles/rischert_regex_pt1.html)

Using Regular Expressions:

<http://otn.oracle.com/obe/obe10gdb/develop/regex/regex.htm>

- Zabezpieczenie przed atakami cross-site scripting.

```
SecFilter "<[[:space:]]*script"
```

[[:space:]] w wyrażeniu regularnym oznacza klasę znaków typu „odstęp” tzn.: znak spacji lub inny znak oznaczający odstęp (np. TAB, CR, LF, ASCII-255).

- Zabezpieczenie przed podstawowymi atakami SQL-injection.

```
SecFilter "delete[[:space:]]+from"
SecFilter "insert[[:space:]]+into"
SecFilter "select[[:space:]]+from"
SecFilter "or[[:space:]]+(.)=\1"
```

Ostatnia reguła zabezpiecza przed doklejeniem wyrażenia logicznego które jest zawsze prawdziwe (np.: or 1=1 , lub: or nnn=nnn)

Uwaga: Skonstruowanie uniwersalnych reguł które chroniłyaby przed atakami SQL-injection jest bardzo trudne albo wręcz niemożliwe. Problem polega na tym, że fragmenty doklejanych zapytań będą parsowane przez interpreter SQL a my mamy do dyspozycji wyrażenia regularne. SQL jest językiem bardzo rozbudowanym, przez co ten sam skutek można osiągnąć konstruując inne zapytanie. Przykładowo w ciągu:

```
or `abcde`=' abcde'
```

Można zastosować operator sklejania ciągów znaków:

```
or `abcde`=' abc'+ ' de'
```

Warunek zawsze prawdziwy można też skonstruować w SQL na wiele różnych sposobów. Tak więc w wypadku SQL-injection mod\_security (i każdy inny filtr) daje tylko częściowe zabezpieczenie.

- Jeśli przeglądarka przedstawia się za pomocą cookie, to przepuszczenie wyłącznie tych zleceń w których parametr „SessionID” składa się tylko z cyfr:

```
SecFilterSelective COOKIE_sessionid "!^([0-9]{1,9})$"
```

- Przepuszczenie całego ruchu dla jednego adresu IP (np. stacji administratora):

```
SecFilterSelective REMOTE_ADDR "^10.1.1.3$" nolog,allow
```

Tego typu reguła musi pojawić się przed wszystkimi innymi.

## Zapobieganie atakom na oprogramowanie Oracle

Niektóre wersje oprogramowania Oracle (Apache/Oracle HTTP Server i towarzyszące mu moduły) zawierały błędy lub niebezpieczne ustawienia standardowe, które można wykorzystać do skutecznego ataku na dane lub system operacyjny serwera. Mod\_security można wykorzystać również do ograniczenia możliwości ataków na oprogramowanie Oracle wykorzystujących jako transport protokół HTTP.

- Zabronienie dostępu do skryptów demo standardowo udostępnianych w instalacjach Oracle HTTP Server:

```
SecFilterSelective REQUEST_URI "demo"
```

- Wcześniejsze wersje Oracle HTTP Server posiadały błąd pozwalający na pobranie kodu źródłowego stron JSP. Atak taki polegał na pobraniu źródła strony z katalogu `/_pages`. Katalog ten nie jest potrzebny dla użytkownika więc dostęp do niego można zablokować:

```
SecFilterSelective REQUEST_URI "_pages/"
```

- Zablokowanie dostępu do stron administracyjnych różnych modułów:

```
SecFilterSelective REQUEST_URI "pls\/.+\/admin_"
SecFilterSelective REQUEST_URI "dms0|dms\DMSDump"
SecFilterSelective REQUEST_URI "servlet\Spy"
SecFilterSelective REQUEST_URI "oprocmgr"
```

- Zablokowanie prób pobrania plików konfiguracyjnych np: `plsql.conf`, `XSQLConfig.xml`, `soapConfig.xml`. W niektórych wersjach było to możliwe dzięki wykorzystaniu technik *double-encoding* i *path traversal*:

```
SecFilterSelective REQUEST_URI
    "plsql.conf|XSQLConfig.xml|soapConfig.xml"
```

- Starsze wersje Oracle HTTP Server udostępniały przez interfejs `mod_plsql` wszystkie procedury z pakietu `owa_util`. Umożliwiało to wiele ciekawych ataków na serwer i na aplikacje (np. pobranie kodu źródłowego dowolnej procedury za pomocą `owa_util.showsource`). Zabezpieczenie może polegać na wyblokowaniu zleceń charakterystycznych dla wywołań procedur z pakietu `owa_util`:

```
SecFilterSelective REQUEST_URI "pls\/.+\/owa_util\."
```

### 3. Mod\_security a Oracle

Mod\_security można zintegrować bezpośrednio z Oracle HTTP Server. Sprawdziłem jego współpracę z OHS zarówno będącym elementem bazy jak i 9i Application Server.

Dokumentacja mod\_security opisuje instalację mod\_security ze źródeł lub z wykorzystaniem dostępnych wersji binarnych. Instalacja ze źródeł (oprócz kompilatora gcc) wymaga obecności skryptu `$ORACLE_HOME/Apache/Apache/bin/apxs`, który nie jest obecny we wszystkich instalacjach OHS.

Instalacja z binariów udostępnionych na stronach projektu ([www.modsecurity.org](http://www.modsecurity.org)) jest dość prosta. W przypadku OHS polega ona na:

1. Skopiowaniu pliku `mod_security.so` (w przypadku Linuxa) lub `mod_security.dll` (w przypadku Windows) do katalogu `$ORACLE_HOME/Apache/Apache/modules`
2. Dopisaniu do `$ORACLE_HOME/Apache/Apache/conf/httpd.conf` linii:

```
LoadModule security_module modules/mod_security.dll
AddModule mod_security.c
```

3. Skonfigurowaniu mod\_security. Dla wygody można to zrobić w osobnym pliku, który dołączy się za pomocą polecenia *include* na końcu `httpd.conf`

```
include "C:\ora9ias\Apache\Apache\conf\modsecurity.conf"
```

W przypadku integrowania mod\_security z Oracle Application Server warto przepuścić wszystkie zlecenia płynące bezpośrednio z adresu IP naszego serwera.:

```
SecFilterSelective REMOTE_ADDR "^10.1.1.100$" nolog,allow
```

Jest to spowodowane tym, że niektóre z modułów Application Server-a komunikują się z Oracle HTTP Server w sposób inny niż przeglądarki HTTP. Np. zauważyłem że raz na minutę serwer sprawdza czy serwis HTTP działa poprawnie przez wywołanie metody HEAD. Zlecenie te nie zawiera żadnych dodatkowych nagłówków HTTP.

Oracle HTTP Server nawet w najnowszej wersji 10gR1 nadal wykorzystuje oprogramowanie Apache 1.3.x. Należy o tym pamiętać konfigurując `mod_security`. Największym ograniczeniem dla `mod_security` współpracującego z Apache 1.3 jest brak możliwości analizy odpowiedzi serwera.

#### 4. Jak zbudować firewall aplikacyjny na osobnym serwerze?

Apache 1.3.x nie jest dobrze przygotowany na dołączanie modułów filtrujących ruch. W związku z tym `mod_security` dla Apache 1.3 musi wykorzystywać pewne sztuczki programistyczne. Sam autor `mod_security` pisze w dokumentacji:

*“Apache 1.x does not offer proper infrastructure to intercept requests. Since what mod\_security is doing is a hack,”*

W związku z powyższym nie zalecałbym bezpośredniego integrowania `mod_security` z Oracle HTTP Server w instalacjach produkcyjnych (Apache 1.3 jest stosowany we wszystkich obecnych wersjach Oracle HTTP Server). Nawet gdyby nie byłyby żadnych przeciwskażeń do integrowania `mod_security` z Apache 1.3 to wydaje mi się, że taka ingerencja w oprogramowanie Oracle mogłaby być zbyt ryzykowna.

W związku z tym zalecałbym instalację `mod_security` na osobnym serwerze lub osobnym serwisie HTTP działającym na tym samym serwerze (na innym porcie). Taki serwer HTTP powinien działać jako reverse proxy. Funkcjonalność ta polega na tym że serwer obsługuje wszystkie zlecenia płynące od klientów i przekazuje je do właściwego serwera. W analogiczny sposób są przesyłane odpowiedzi serwera. Na podobnej zasadzie działa m.in. Oracle WebCache.

Do zbudowania serwisu HTTP typu reverse proxy z filtrowaniem `mod_security` najlepiej wykorzystać Apache 2.x. W standardowej konfiguracji Apache należy wprowadzić następujące zmiany:

1. Załadować `mod_security` i moduły niezbędne do obsługi funkcjonalności `reverse_proxy`:

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
LoadModule security_module modules/mod_security.dll
```

2. Wyłączyć obsługę zleceń typu proxy.

```
ProxyRequests Off
```

Zlecenia typu proxy są potrzebne do realizacji funkcjonalności typowego proxy HTTP a nie funkcjonalności *reverse proxy*. Jeśliśmy nie wyłączyli obsługi tych zleceń, to nasz firewall aplikacyjny mógłby być wykorzystywany jako *proxy* przez intruzów, do ukrywania swojej tożsamości przy atakach przy użyciu protokołu HTTP.

3. Spowodować przekierowanie wszystkich zleceń HTTP do chronionego serwera (analogicznie dla odpowiedzi serwera):

```
ProxyPass / http://10.1.1.100/
ProxyPassReverse / http://10.0.0.100/
```

4. Skonfigurować `mod_security`

Naturalnie serwer reverse proxy powinien być dostępny pod adresem IP (i na porcie tcp) na którym chcemy udostępnić chronioną aplikację.