Obrona

Apache jako Reverse Proxy

Radosław „Goblin” Pieczonka

stopień trudności

W środowisku firmowym często można na serwerach intranetowych znaleźć oprogramowanie potencjalnie podatne na różnego rodzaju ataki. Co gorsza, zdarza się, iż od publikacji informacji o dziurze w zabezpieczeniach do wydania odpowiedniej łatki mija niemało czasu.

Jedną z takich aplikacji jest Microsoft OWA (*Outlook Web Access*), wyśmienita aplikacja typu webmail, która staje się coraz popularniejszym sposobem dostępu do poczty w środowisku serwerów Microsoft, a co za tym idzie, znajduje się ona również coraz częściej na celowniku sieciowych wandali.

Z różnych powodów serwery tego typu aplikacji znajdują się często wewnątrz firmowego intranetu, co stanowi świetne rozwiązanie, o ile nie zachodzi potrzeba zdalnego dostępu. Z reguły jednak, prędzej czy później zachodzi potrzeba uzyskania dostępu do aplikacji spoza sieci lokalnej. Aby zapewnić taki dostęp, należy rozważyć kilka opcji. Pierwszym z możliwych rozwiązań jest zestawianie połączeń wirtualnych sieci prywatnych, co jednak wymaga m.in. dodatkowej konfiguracji komputerów klienckich, czego najczęściej chcielibyśmy uniknąć.

Inną możliwością jest kolokacja serwerów aplikacji w serwerowni zewnętrznego operatora. Istotną zaletą takiego rozwiązania jest brak konieczności nadzorowania infrastruktury i serwerów przez naszego administratora i pozostawienie tych obowiązków pracownikom providera. Często okazuje się jednak taka opcja mniej elastyczną, zaś koszta utrzymania takiej infrastruktury mogą być wyższe niż w przypadku innych rozwiązań. Co ważniejsze, w niektórych sytuacjach ważne z punktu widzenia polityki bezpieczeństwa jest przechowywanie wrażliwych danych wewnątrz infrastruktury firmy, nie zaś u zewnętrznego operatora.

Aby pozostawić dane wewnątrz naszej sieci, najczęściej lokalizujemy serwery w strefie zdemilitaryzowanej, tak by pozostały pod naszą kontrolą, ale jednocześnie były dostępne z Internetu poprzez korporacyjny firewall z filtrowaniem pakietów, systemem IPS i innymi zabezpieczeniami. Choć w tym wypadku mamy pełną kontrolę nad serwerami, to samo rozwiązanie z punktu widzenia sieci nie różni się znacząco od poprzedniego. Informacje, zarówno w sieci we-

Z artykułu dowiesz się

- Czym jest reverse proxy?
- Jak skonfigurować Apache2 dla reverse proxy?
- Jak zabezpieczyć publikowane zasoby?

Co powinieneś wiedzieć

- Podstawy konfiguracji Apache2

Listing 1. *Podstawowa konfiguracja vhosta w Apache2*

```
<VirtualHost *>
ServerAdmin admin@foo.bar
ServerName intra.company.com
ErrorLog /var/log/apache2/owa.foo.bar-error.log
CustomLog /var/log/apache2/owa.foo.bar-access.log common
ProxyPass / http://owa.lan/
ProxyPassReverse / http://owa.lan/
ProxyRequests Off
<Proxy *>
   Order deny,allow
   Allow from all
</Proxy>
</VirtualHost>
```

wnętrznej, jak i zewnętrznej, przetwarzane są w podobny sposób, serwery mają publiczne adresy IP czy nazwy domenowe, a komunikacja szyfrowana poprzez SSL wymaga osobnych certyfikatów dla serwera.

Aby pominąć ograniczenia, jakie stawia przed nami lokowanie serwerów w strefie zdemilitaryzowanej oraz jeśli chcemy w jeszcze większym stopniu podnieść bezpieczeństwo, możemy postawić na maszynie pełniącej rolę firmowego firewalla/nata lub na dedykowanym serwerze w DMZ usługę reverse proxy. Ten sposób dostępu do zasobów może wydawać sie najtrudniejszym i najbardziej zaawansowanym, jest jednak niezmiernie elastyczny i daje nam wiele nowych funkcji.

Weźmy jako przykład sytuację, gdy mamy w sieci serwer Microsoft Exchange z aplikacją webmail OWA, intranetowy serwer z webową bazą wiedzy i aplikacją CRM, routing i NAT realizowany jest na dedykowanym routerze sprzętowym lub linuksowym oraz serwer linuksowy zlokalizowany w strefie zdemilitaryzowanej. Tym, co chcielibyśmy osiągnąć, jest umożliwienie przeglądania bazy wiedzy oraz poczty z Internetu oraz zachowanie dostępu do systemu CRM tylko z sieci lokalnej.

Po pierwsze, musimy umożliwić serwerowi w DMZ dostęp do intranetowych serwerów OWA i bazy wiedzy, co w skrócie sprowadza się do:

- utworzenia reguł umożliwiających dostęp do OWA na MS IIS z IP serwera w DMZ,
- utworzenia reguł umożliwiających dostęp do strony bazy wiedzy z IP serwera w DMZ,
- utworzenie odpowiednich reguł firewalla i NAT-a, aby umożliwić ruch pomiędzy wybranymi serwerami intranetowymi oraz serwerem w DMZ.

Jeśli możemy przeglądać zawartość stron z publicznego serwera, oznacza to, że czas opublikować je do Internetu. Aby to osiągnąć, wykorzystamy serwer http Apache2 wraz z modułami *mod_proxy* i *mod_proxy_http*. Listing 1 prezentuje podstawową konfigurację vhosta dla publikowania wewnętrznego serwisu dla klientów łączących się do publicznego serwera.

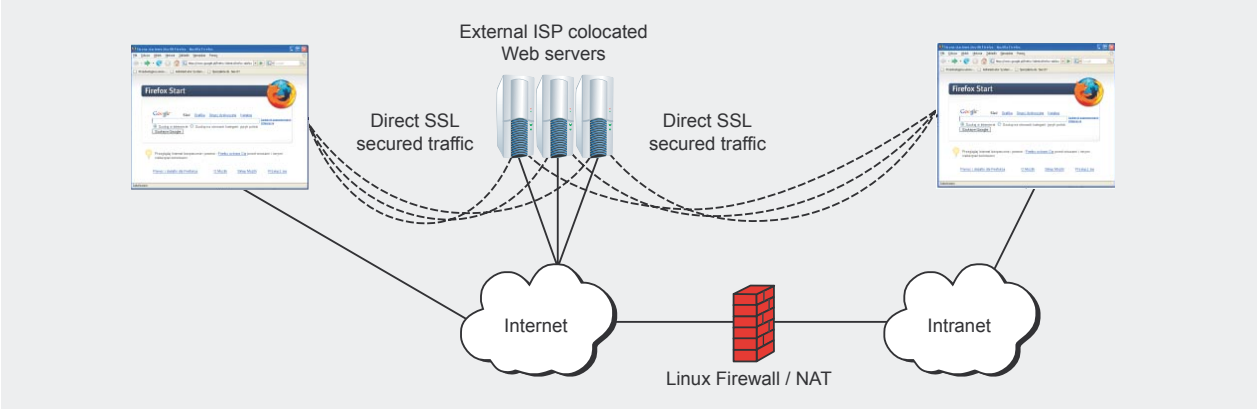
Przy takiej konfiguracji Apache'a strona *http://owa.lan* powinna być dostępna z Internetu pod adresem *http://intra.company.com*. Trudno jednak stwierdzić, aby takie rozwiązanie podnosiło w sposób znaczący bezpieczeństwo, co więcej, przy takich ustawieniach nadal publikujemy zasoby tylko jednego wewnętrznego serwera i to wszystko to, co jest na nim dostępne. Co możemy zrobić w celu poprawienia zaprezentowanego reverse proxy?

Przede wszystkim warto nieco bardziej szczegółowo określić, jakie zasoby mają być publikowane i pod jakimi nazwami, zmieńmy więc:

- *ProxyPass/ http://owa.lan/,*
- *ProxyPassReverse http://owa.lan/.*

na szczegółowy wykaz publikowanych folderów webowych:

- *ProxyPass/exchange http://owa.lan/exchange,*
- *ProxyPassReverse /exchange http://owa.lan/exchange,*
- *ProxyPass /exchweb http://owa.lan/exchweb,*
- *ProxyPassReverse /exchweb http://owa.lan/exchweb,*
- *ProxyPass /public http://owa.lan/public,*
- *ProxyPassReverse /public http://owa.lan/public,*



Rysunek 1. *Diagram dostępu w wariancie kolokacji*

- *ProxyPass /knb http://webserver.lan/knowledgebase,*
- *ProxyPassReverse /knb http://webserver.lan/knowledgebase.*

Ciphered connections

Dzięki powyższej zmianie, dostęp jest ograniczony jedynie do wybranych katalogów, zaś zależnie od wywołanego adresu, prezentowana będzie zawartość odpowiedniego serwera intranetowego. Warto jednak pamiętać o tym, że choć do ruchu wewnątrz sieci lokalnej mamy zazwyczaj spore zaufanie, ruch w Internecie to przypadek przeciwny. W związku z tym, powszechną i wskazaną praktyką jest, by takie połączenia były szyfrowane. Aby pozostać w zgodzie z ową przyjętą procedurą, zaimplementujmy w naszym rozwiązaniu szyfrowanie poprzez dodanie wpisów odpowiadających za SSL do konfiguracji serwera:

```
Listen 443
```

oraz

```
NameVirtualHost *:443
```

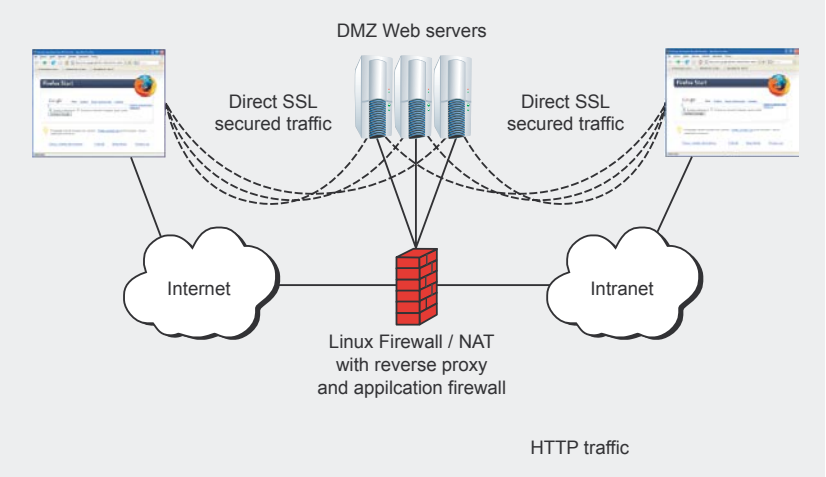
Gdy Apache nasłuchuje już na domyślnym porcie HTTPS, możemy zaktualizować naszą konfigurację vhosta do stanu zaprezentowanego w Listingu 2 i Rysunku 1, gdzie korzystamy również z dyrektywy *SSLProxyEngine*, która umożliwia publikowanie zasobów z wykorzystaniem protokołu HTTPS.

Jeśli publikowane zasoby mają być dostępne jedynie dla pracowników firmy oraz innych zaufanych podmiotów, warto rozważyć implementację dodatkowej warstwy autoryzacji klienta poprzez certyfikaty. Serwer proxy może wymagać od przeglądarki klienta przedstawienia prawidłowego certyfikatu, zanim połączenie zostanie ustanowione.

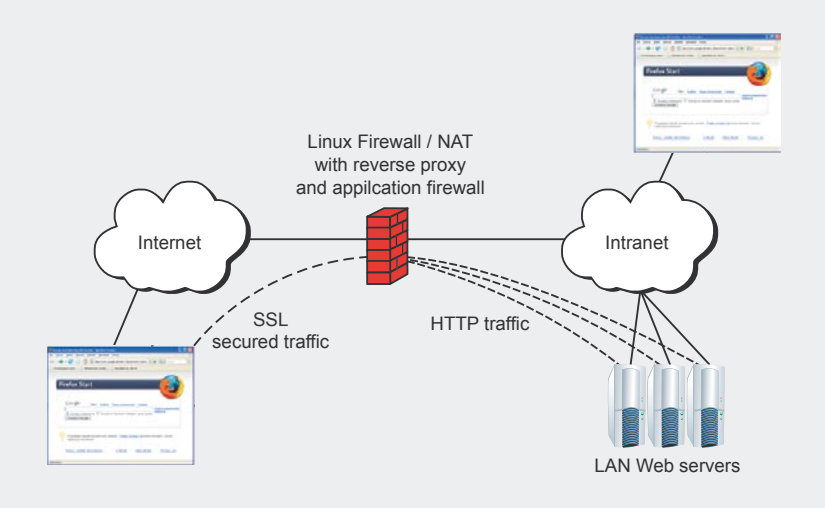
Innym aspektem implementacji *mod_proxy* jako reverse proxy jest rozkładanie ruchu pomiędzy serwery celem równoważenia obciążenia oraz przeniesienie odpowiedzialności za szyfrowanie ruchu z serwera aplikacji na serwer proxy. Gdy publikujemy wewnętrzne zasoby HTTP z uruchomionym szyfrowaniem SSL na publicznie dostępnym serwerze reverse proxy, mamy możliwość publikacji zasobów wielu serwerów pod jedną nazwą domenową, a co za tym idzie, z wykorzystaniem jednego certyfikatu dla SSL. Może to stanowić dla firmy znaczące udogodnienie oraz skutkować obniżeniem kosztów utrzymania certyfikatów nabywanych u komercyjnych dostawców. Ciekawym sposobem wykorzystania reverse proxy w kontekście szyfrowania jest również instalacja pojedynczego akceleratora kryptograficznego dla wszystkich serwerów firmy, co daje nam możliwość znacznego uproszczenia i optymalizacji infrastruktury klucza publicznego stosowanej w firmie.

Specyficzne problemy

Pomimo iż opisywane rozwiązanie jest bardzo dobre, może stać się ono powodem niespodziewanych problemów w wypadku niektórych aplikacji webowych. Na przykład w wypadku publikacji wspominanego kilkakrotnie *Outlook Web Access*, konfiguracja domyślna serwera IIS i Exchange powoduje drobny, acz przykry problem, którym jest autentykacja z wykorzystaniem NTLM czyli tzw. *Windows Integrated Authentication*. Zgodnie z oficjalnymi publikacjami Microsoft, autentykacja *NTLM authentication* nie działa poprzez serwery proxy. Po dokładniejszym sprawdzeniu okazuje się, iż działa ona w przeglądarkach, innych niż Microsoft producentów, takich jak Mozilla Firefox czy Opera, ale nie funkcjonuje, gdy próba otworzenia tej samej strony wykonywana jest z Internet Explorera. Wiedząc, gdzie leży problem, możemy wyłączyć autentykację opartą o NTLM po stronie serwe-



Rysunek 2. *Diagram dostępu w wariancie DMZ*



Rysunek 3. *Diagram dostępu przez reverse proxy*

ra IIS (w *Exchange Management Console*) lub, jeśli chcemy zachować możliwośc takiej autentykacji w sieci LAN, możemy za pomocą *mod_headers* dla Apache zmodyfikować nagłówki tak, by serwer proxy rozgłaszał jedynie w podstawowy sposób, jak poniżej:

```
Header unset WWW-Authenticate
Header set WWW-Authenticate ↩
 "Basic realm=webmail.company.com"
```

Firewall warstwy aplikacji

W wielu sytuacjach dynamicznie generowana zawartość strony przygotowywana jest za pomocą PHP, ASP czy innej podobnej technologii. Rozwiązania takie są potencjalnie podatne na pewne rodzaje nadużyć. Nie zdarza się raczej, by firma dysponowała zasobami koniecznymi do weryfikacji każdego fragmentu kodu źródłowego, a często w wypadku zamkniętych rozwiązań, nawet do kodu dostępu. Jeśli w sieci intranet do pewnego stopnia można dopuścić takie zagrożenie, to już w wypadku zasobów publikowanych do Internetu, konieczne jest, by stosowane rozwiązania były jak najbezpieczniejsze.

Gdy implementujemy reverse proxy, nie tylko możemy wykorzystać je do prostego publikowania z dodatkowym zabezpieczeniem w formie szyfrowania SSL. Dzięki modułowi *mod_security2* możemy utworzyć całkowicie nową warstwę filtrowania ruchu dla naszych rozwiązań.

Jednym z najpopularniejszych zagrożeń, na jakie narażone są aplikacje webowe, są ataki wstrzykujące kod jak *SQL injection.* Najczęściej powodowane przez niewłaściwą obsługę zmiennych, mogą zostać załatane poprzez modyfikacje kodu aplikacji.

Wyobraźmy sobie sytuację, w której używana jest spora aplikacja webowa, utworzona dłuższy czas temu i nigdy niemodyfikowana. Sporo wydarzyło się od czasu, kiedy ją napisano, PHP nie powinno juz np. wykorzystywać zmiennych globalnych (*register_globals*) do obsługi wywołań *GET/POST;* obecnie kładzie się również znacznie większy nacisk na bezpieczeństwo. Jednak dla stworzonego kiedyś skomplikowanego oprogramowania, aktualizacja do bieżących standardów może okazać się koszmarem. Często zdarza się wręcz, iż lepiej jest napisać nowy system w miejsce starego, gdyż może to być tańsze od łatania słabo udokumentowanej aplikacji.

Nawet jeśli zapadnie decyzja o tworzeniu nowego rozwiązania, nie stanie sie to w ciągu dni czy nawet tygodni, a my nie możemy sobie pozwolić na publiczny dostęp do dziurawego oprogramowania. Jeśli system jest podatny na wstrzykiwanie kodu, łatwo sobie wyobrazić jak tragiczne w skutkach może być wpisanie do przeglądarki w polu adresu *http://company.com/buggy.php?username=jonny;DROP%20TABLE%20users*

Aby uniknąć podobnych nadużyć, można wykorzystać *mod_security2* do blokowania typowych ataków wstrzykiwania kodu.

Jeśli dana dystrybucja zawiera bieżącą wersję *mod_security2*, mamy szczęście, jednak w większości wypadków konieczna będzie kompilacja i instalacja tego modułu ze źródeł.

Na szczęście instalacja tego modułu jest dość prosta, i nie powinna stanowić żadnego problemu. Poza instalacją pakietów z odpowiednimi bibliotekami jak *libxml2*, trzeba dokonać drobnej zmiany w *Makefile,* gdzie trzeba prawidłowo zadeklarować ścieżkę do katalogu instalacji Apache (np. */usr/share/apache2* Debianie), a następnie po prostu wykonać polecenia *make* i *make install.*

Po zainstalowaniu modułu *mod_security2* dla Apache2, należy jeszcze zmodyfikować konfigurację serwera, tak by ładował on odpowiednie

Listing 2. *Konfiguracja vhosta z SSL i wybranymi zasobami*

```
<VirtualHost *:443>
ServerAdmin admin@foo.bar
ServerName intra.company.com
ErrorLog /var/log/apache2/ssl-intra.company.com-error.log
CustomLog /var/log/apache2/ssl-intra.company.com-access.log common
SSLEngine on
SSLCertificateFile /etc/ssl/certs/ssl-cert-intra.company.com.pem
SSLCertificateKeyFile /etc/ssl/private/ssl-cert-intra.company.com.key
SSLProxyEngine on
ProxyPass /exchange https://owa.lan/exchange
ProxyPassReverse /exchange https://owa.lan/exchange
ProxyPass /exchweb https://owa.lan/exchweb
ProxyPassReverse /exchweb https://owa.lan/exchweb
ProxyPass /public https://owa.lan/public
ProxyPassReverse /public https://owa.lan/public
ProxyPass /knb http://webserver.lan/knowledgebase
ProxyPassReverse /knb http://webserver.lan/knowledgebase
ProxyRequests Off
<Proxy *>
   Order deny,allow
   Allow from all
</Proxy>
</VirtualHost>
```

Listing 3. *Skrypt vulnerable.php*

```
This site is vulnerable to javascript injection
<?
if(!isset($_GET['username'])) $_GET['username'] = "no username supplied";
echo $_GET['username']
?>
```

Listing 4.

```
[Thu Apr 12 13:09:21 2007] [error] [client xxx.xxx.xxx.xxx] ModSecurity:
 Access denied with code 409 (phase 2). Pattern match "<script" at ARGS:
username. [hostname "company.com"] [uri "/vulnerable.php?username=
noob<script>alert(\\"I%20am%20so%20vulnerable...\\")</script>"]
 [unique_id"kiwv4Fdi6FkABAVRAMsABBAAA"]
```

pliki, co można najprościej osiągnąć poprzez dodanie poniższych wpisów w odpowiednim miejscu plików konfiguracyjnych:

```
LoadFile /usr/lib/libxml2.so
LoadModule security2_module ↩
 /usr/lib/apache2/modules/ ↩
mod_security2.so
LoadModule unique_id_module ↩
 /usr/lib/apache2/modules/ ↩
 mod_unique_id.so
```

mod_unique_id powinien być dostępny w domyślnej instalacji Apache2 i jest wymagany do poprawnej pracy *mod_security2*.

Gdy *mod_security2* jest już dostępny i załadowany do Apache, pozostaje włączyć jego obsługę dla całego serwera, wybranych lokalizacji czy vhostów, w naszym zaś przypadku dla vhosta odpowiedzialnego za reverse proxy, do czego służy dyrektywa *SecRuleEngine On*. Dodatkowo możemy ustalić, jakie operacje zostaną wykonane domyślnie w wypadku wykrycia potencjalnego ataku za pomocą *SecDefaultAction*. Stwórzmy więc taką konfigurację, która odrzuci potencjalnie szkodliwe wywołania z błędem 409 oraz zaloguje je w celu późniejszej analizy. Możliwe jest również modyfikowanie zachowania filtra dla konkretnych reguł, o czym napiszemy za chwilę.

Warto zdawać sobie sprawę z faktu, iż *mod_security2* domyślnie wykonuje pewne operacje w celu niedopuszczenia do ukrycia ataku przed wzorcami wykorzystywanym do jego wykrycia. Wywołanie przekazane przez klienta zostaje doprowadzone do postaci kanonicznej, usunięte zostaną wielokrotne slashe, wskazania do bieżącego katalogu, a także zdekodowane zostaną kody znaków w URLu, co znacząco ogranicza możliwości obejścia zabezpieczeń.

Najczęściej spotykaną w konfiguracji *mod_security2* dyrektywą jest *SecRule*. Za pomocą tych wpisów tworzymy nasze reguły bezpieczeństwa, które będą wywoływać wcześniej zadeklarowane domyślne zachowanie. *Idea* działania reguł *SecRule* jest prosta i sprowadza się do wykonania określonych czynności, gdy zadeklarowany wzorzec zostanie wykryty w przychodzącym od klienta wywołaniu. Ekipa odpowiedzialna za *mod_security2* była na tyle miła, by wraz z kodem źródłowym rozpowszechniać predefiniowany zestaw reguł. Umieszczone tam wpisy pokrywają się z najczęściej spotykanymi w sieci problemami zabezpieczeń aplikacji webowych; intuicyjnie podzielone i dobrze skomentowane, znane są pod nazwą *Core Rules*. Zachęcam do zapoznania się z zawartością tego zbioru regułek, gdyż dają one wgląd w wiele metod ataków na aplikacje webowe oraz wzorców, które pozwalają tego typu zachowania zidentyfikować i zablokować.

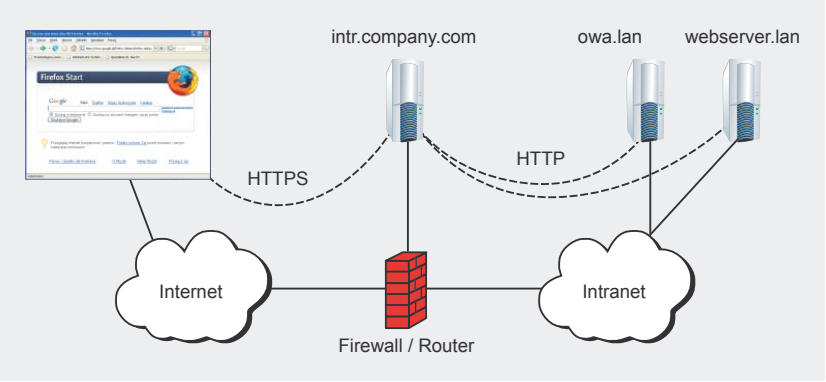
Jak można zauważyć w *Core Rules*, każda reguła *SecRule* może przyjmować dwa lub trzy argumenty. Pierwszy wskazuje na miejsce, w którym mamy poszukiwać wzorca, a kolejny to tenże wzorzec. Jako trzeci, opcjonalny argument, możemy zadeklarować, jakie działania mają być podjęte w wypadku wystąpienia łańcucha znaków zgodnego z wzorcem – operacja ta będzie miała pierwszeństwo nad działaniami deklarowanymi jako domyślne.

Aby zademonstrować, w jaki sposób *mod_security2* może chronić stronę przed nadużyciem, stworzymy mały, dziurawy skrypt *vulnerable.php*, który będzie dostępny na słabo skonfigurowanym serwerze, co w efekcie spowoduje podatność na atak typu *script injection*.

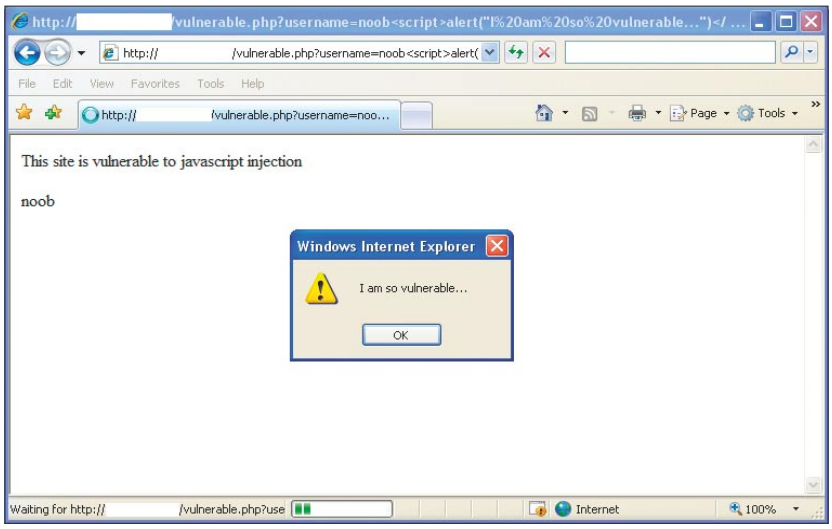
Gdy otwieramy taką stronę, możemy wykorzystać podatność na wstrzyknięcie kodu javascript poprzez wklejenie go jako części nazwy użytkownika, jak na przykład:

```
username=noob<script>alert ↩
 2("I am so vulnerable...")</script>,
```

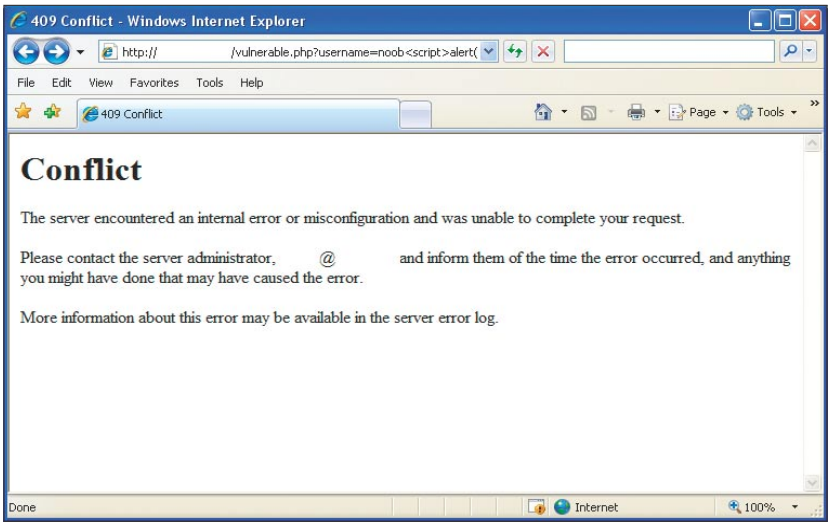
czego efekt widać na Rysunku 5.



Rysunek 4. *Diagram dostępu w wariancie reverse proxy w DMZ*



Rysunek 5. *Przykładowy efekt udanego wstrzyknięcia skryptu*



Rysunek 6. *Wywołanie zablokowane przez ModSecurity*

Problem ten można naprawić po stronie kodu samej aplikacji bądź serwera, na którym jest ona uruchomiona. W tym wypadku zademonstrujemy, jak możemy poradzić sobie z takim problemem, nie posiadając nawet dostępu do serwera, na którym jest ona uruchomiona czy tym bardziej uprawnień administracyjnych lub dostępu do kodu aplikacji. Jeśli zdecydujemy się na taką modyfikację infrastruktury, by serwer chroniony był przez oparty na Apachu reverse proxy z aktywnym *mod_security2*. Możemy na przykład zablokować wszystkie wywołania zawierające w zmiennych łańcuch *<script>* oraz zwracać w takiej sytuacji wybrany kod błędu.

Aby to osiągnąć, wykorzystamy trzy proste wpisy w pliku konfiguracyjnym:

```
SecRuleEngine On
SecDefaultAction log,auditlog, ←
 deny,status:409,phase:2,t:none
SecRule ARGS "<script>"
```

Po takiej modyfikacji sieciowy wandal otrzyma przy próbie przełamania zabezpieczeń naszej aplikacji komunikat o błędzie 409 (Rysunek 6), a co ważniejsze, zostanie on odnotowany w logach bezpieczeństwa w sposób podobny do przedstawionego na Listingu 4.

Pracując nad *ModSecurity* dla Apache, ważne jest, by pamiętać, iż ustawienie zbyt restrykcyjnych reguł zapory poziomu aplikacyjnego dla naszych rozwiązań webowych może skutkować poważnym zakłóceniem ich pracy czy nawet całkowitym ich zablokowaniem. W związku z tym, realizacja tego typu zabezpieczeń powinna być związana z dokładnym testowaniem funkcjonalności aplikacji, którą będziemy chronić. W przeciwnym wypadku może nas czekać niezadowolenie użytkowników, gdy nasze zmiany negatywnie wpłyną na komfort ich pracy.

MS ISA Server 2004

Jak już wspominaliśmy wcześniej, podobną funkcjonalność można spotkać na przykład w *Microsoft Internet and Security Acceleration Server 2004*. Produkt ten ze względów finansowych jest raczej kierowany do dużych korporacji, nie zaś do niewielkich odbiorców z segmentów SOHO i SME. Microsoft nadał tej funkcjonalności nazwę reguł publikowania (*Server/Web Publishing Rules*).

Jeśli mówimy o publikowaniu zasobów wewnętrznych serwisów webowych, to w grę wchodzą *Web Publishing Rules* i *Secure Web Publishing Rules*, przy czym te drugie odpowiadają za publikowanie zasobów z wykorzystaniem szyfrowania SSL. Ruch SSL może być wykonywany w trybie *SSL tunelling* lub *SSL bridging*. W pierwszym wypadku ISA przekazuje pakiety bezpośrednio do serwera wewnętrznego, zaś w drugim, ruch zostaje odszyfrowany i ponownie zaszyfrowany, co pozwala na weryfikację zawartości pakietów.

Na konfigurację reguł publikowania składają się następujące opcje:

- *Action* – włącza lub wyłącza dostęp,
- *Name* – fqdn lub adres IP serwera, który będzie publikowany,
- *Users* – uprawnienia dostępu do witryny na poziomie użytkowników,
- *Traffic source* – obiekty sieciowe, z których możliwy będzie dostęp do witryny,
- *Public name* – URL, na który będzie odpowiadać ISA,
- *Web listener* – obiekt definiujący sposób nasłuchiwania przez serwer ISA,
- *Path mappings* – selektywne mapowanie serwerów wewnętrznych na podstawie ścieżki w wywołanym URL-u,
- *Bridging* – sposób przekierowywania pakietów, może wykorzystywać HTTP, SSL lub FTP,
- *Link translation* – definiuje sposoby modyfikowania linków zawartych w odpowiedziach na żądania klienta.

Podsumowanie

Samo utworzenie reguł publikowania wykonywane jest, jak to w produktach MS zwykło bywać, przez przyjazny użytkownikowi kreator.

Tak skonfigurowane publikowanie zasobów można rozszerzyć o filtry webowe (*Web Filters*), które są funkcjonalnością zbliżone do omówionego wcześniej *ModSecurity2*.

Jak więc mogliśmy się przekonać, reverse proxy może być wykorzystane do znacznego podniesienia bezpieczeństwa publikowanych zasobów, z użyciem otwartego i darmowego oprogramowania. Rozwiązanie oparte o system GNU/Linux i serwer Apache2 może stanowić wyśmienitą alternatywę dla takich komercyjnych produktów jak choćby *Microsoft ISA Server* i jego funkcja publikowania zasobów. ●

O Autorze

Autor jest inżynierem ds. zabezpieczeń w dziale Research&Development w firmie IF Research / Wallix.
Kontakt z autorem: *goblin@pentex.pl*