ModSecurity 1 Day Tutorial

Introduction to ModSecurity, the Apache Security Module

OWASP European Conference 2009 in Kraków, Poland

May 12th, 2009

By Dr. Christian Folini, netnea – christian.folini@netnea.com

Day Program

Lab: Rules I
: Writing ModSecurity rules with instant reward

Architecture I
: The place of a Web Application firewall in an ideal and in a not so ideal environment

Core Rules
: Generic blacklisting saves you from 90% of attacks

Alerts and Logs
: How they look and how you should handle them

HTTP
: A closer look at a well known protocol

Architecture II
: How to use ModSecurity in the Enterprise

Lab: Rules II
: The more complex rules and those you need to check the documentation in order to understand them

Remo and AuditViewer
: When blacklisting is not enough, it's time for whitelisting with the help of Remo. AuditViewer helps reading audit logs.

Who am I Dr. Christian Folini

Historian specialized in German Mysticism and Monasteries
The parallels between a multi tier application and a well built monastery are quite surprising

Some papers on database architecture for Historians
Ever tried to store a historical date in a database when the unix epoch started in 1970?

Some years of experience as System Administrator
You know how to configure procmail? You are just the man we are looking for!

Webserver engineering for multiple online banking sites
That's been more serious projects, so no jokes here.

Linux Desktop project in the Enterprise
... but that one never really came off.

Remo Developer
Kind got stuck with this whitelisting idea and then I just had to do it.

Spare time Burgundian Soldier
See me with the Company of St. George

Basic Setup – Apache and a few commands

Archive

Get the following files from

http://folini.tikon.ch/owasp.tar.gz

Download, untar and move to /apache

File Hierarchy

DocumentRoot is /apache This can be a symlink

There should be the following subfolders and symlinks

logs	for logfiles
modules	a symlink to the apache modules folder
htdocs	where the content lives
conf	where the config files reside
conf/modsecurity	a place for rulesets

Files we will need

htdocs/index.html	Make this a hello world page
htdocs/subfolder/index.html	The same thing
htdocs/errordocuments/error.html	Generic Error page
conf/httpd.conf	

Commands

```
alias startapa="/usr/sbin/apache2 -k start  \
    -f  /apache/conf/httpd.conf"

alias stopapa="/usr/sbin/apache2 -k stop \
    -f /apache/conf/httpd.conf"

alias reloadapa="/usr/sbin/apache2 -k graceful \
    -f /apache/conf/httpd.conf"

curl -v http://localhost/index.html
```

```
SecRuleEngine           On
SecRequestBodyAccess    On
SecResponseBodyAccess   On
SecResponseBodyMimeType (null) text/html text/plain text/xml

SecDebugLog             /tmp/modsecurity_debug.log
SecDebugLogLevel        5

SecAuditEngine          On
SecAuditLogType         Serial
SecAuditLogParts        "ABCEFHKZ"
SecAuditLog             /tmp/modsecurity_audit.log

SecDataDir              /tmp
SecUploadDir            /tmp
SecTmpDir               /tmp
```

DO NOT USE THIS IN PRODUCTION!

... as is it too verbose

Lab: Rules I **Deny access to an individual URI**

Rule 1 `SecRule REQUEST_URI "^/index\.html$" "deny"`

New:
The *command* SecRule
The *variable* REQUEST_URI
The *operator* regular expression
The *action* deny

Test it:
curl ––verbose http://localhost/index.html

→ HTTP STATUS 501 METHOD NOT IMPLEMENTED

Some other frequently used simple variables:

```
QUERY_STRING
REMOTE_ADDR
REMOTE_HOST
REMOTE_PORT
REMOTE_USER
REQUEST_BODY
REQUEST_LINE
REQUEST_METHOD
REQUEST_PROTOCOL
REQUEST_URI
REQUEST_URI_RAW
RESPONSE_BODY
SERVER_ADDR
SERVER_NAME
SERVER_PORT
```

Lab: Rules I **Deny request with a certain query-string parameter**

Rule 2

```
SecRule QUERY_STRING "debug" \
     "id:2,deny,msg:'Request with debug in querystring'"
```

New:
The *action* id
The *action* msg

Test it:
curl –v http://localhost/index.html

→ HTTP STATUS 501 METHOD NOT IMPLEMENTED

Some other frequently used actions

```
allow
auditlog
chain
ctl
drop
noauditlog
phase
pass
redirect
severity
setenv
skip
skipAfter
status
sanitiseArg
sanitiseMatched
sanitiseRequestHeader
sanitiseResponseHeader
```

Lab: Rules I **Deny request with a certain query-string parameter**

Rule 3

```
SecRule ARGS_GET:debug ".*"  \
     "id:3,drop,msg:'Request with debug \
     parameter querystring'"
```

New:
The *collection variable* ARGS_GET

Test it:
curl –v http://localhost/index.html?debug

→ Empty reply from server (No HTTP Status Code)

Some other frequently used collection variables

```
ARGS
ARGS_GET
ARGS_POST
FILES
REQUEST_HEADERS
REQUEST_COOKIES
RESPONSE_HEADERS
```

Lab: Rules I

Deny request if it is not meeting defined value in parameter (if parameter is given)

Rule 4

```
SecRule ARGS_GET:debug "!^limited$"  \
     "id:4,deny,t:lowercase,msg:'Request \
     illegal debug parameter'"
```

New:
Inversion of the *operator* with exclamation mark (!)
→ This is a positive rule!
Transformation action lowercase

Test it:
curl –v http://localhost/index.html?debug=limited
→ HTTP STATUS 200 OK

curl –v http://localhost/index.html?debug=full
→ HTTP STATUS 501 METHOD NOT IMPLEMENTED

Some other frequently used transformation actions
- lowercase
- replaceNulls
- compressWhitespace
- hexDecode
- hexEncode
- htmlEntityDecode
- escapeSeqDecode
- normalisePath
- normalisePathWin
- md5
- sha1
- replaceComments
- urlDecode
- urlDecodeUni
- base64Encode
- base64Decode

Lab: Rules I

Deny request if certain parameter is not given

Rule 5

```
SecRule &ARGS_GET:debug "@eq 0"  \
     "id:5,deny,msg:'Request with missing \
     debug parameter'"
```

New:
The *modifier* “&” counts the parameters in a *collection variable*
The *operator keyword* @eq 0

Test it:
curl –v http://localhost/index.html

→ HTTP STATUS 501 METHOD NOT IMPLEMENTED

curl –v http://localhost/index.html?debug=limited

→ HTTP STATUS 200 OK

Some other frequently used operator keywords

```
beginsWith
contains
endsWith
eq
ge
gt
le
lt
pm
pmFromFile
rx
streq
validateByteRange
within
```

Lab: Rules I **Deny request with multiple submissions of parameter**

Rule 6

```
SecRule &ARGS_GET:debug "@gt 1"  \
     "id:6,redirect:www.example.com,\
     status:301,severity:1,\
     msg:'Request with multiple debug \
     parameters'"
```

New:
Simple use of the previous method
Extensive use of *actions*

Test it:
curl –v “http://localhost/index.html?debug=limited&debug=full”

→ HTTP STATUS 301 MOVED PERMANENTLY

Lab: Rules I

Deny request with certain parameter only in certain Location

Rule 7

```
<Location "/index.html">
     SecRule ARGS_GET:debug ".*"  \
     "id:7,deny,msg:'Request with debug querystring'"
</Location>
```

New:
Limit execution of ModSecurity command on certain URI.
This is a very important technique to (a) handle false positives and (b) improve performance and (c) reduce complexity.

Test it:
curl –v http://localhost/index.html

→ HTTP STATUS 301 MOVED PERMANENTLY

curl –v http://localhost/subfolder/index.html

→ HTTP STATUS 200 OK

Lab: Rules I **Whitelist certain IP address**

Rule 8

```
SecRule REMOTE_ADDR "@streq 127.0.0.1" \
     "id:8,phase:1,allow"
```

New:
The *operator* streq means string-equal
The *action* allow ends the execution of any rules for all phases

Test it:
Leave Rule 7 active and execute the following:

```
curl -v http://localhost/index.html?debug=limited
```

→ HTTP STATUS 301 MOVED PERMANENTLY

Then change the IP Address in Rule 8 and try again.

```
curl -v http://localhost/index.html?debug=limited
```

→ HTTP STATUS 200 OK

Architecture I

The place of a Web Application Firewall in an ideal and in a not so ideal environment

Draw a fairly complete machine-oriented diagram of an example application

Keywords:
Client, Network Firewall, IDS, IPS, Access Layer Loadbalancer, Entry Server / Reverse Proxy, Authentication Server, Directory, Web Application Firewall, Application Layer Firewall, Application Layer Loadbalancer, Application Webserver, Application Server, Database, Cluster, Demilitarized Zone, Access Layer, Application Layer, Database Layer, Secure Zone, Connection Break, Protocol Break, SSL, UFBP (Universal Firewall Bypass Protocol), Portals, Backend Systems, Sessions, Cookies, Multiple Sessions, Replay Attacks, Input Validation, Layered Input Validation

Threat Modelling (a traditional approach)
- System centric approach
- Attacker centric approach

Now place the Web Application Firewall in this setup and explain where it can help

Talk about Blacklisting and Whitelisting

Further reading:

http://www.ranum.com/security/computer_security/editorials/dumb/

http://jeremiahgrossman.blogspot.com/2008/06/why-most-wafs-do-not-block.html

http://www.modsecurity.org/blog/archives/2005/11/positive_securi.html

Blacklisting and Generic Blacklisting

Have a look at the gotroot ruleset
- http://downloads.prometheus-group.com/delayed/rules/

Have a look at the official ModSecurity Core Rules
- http://www.modsecurity.org/download/index.html

Discuss Parallels and Differences
Keywords: Size, complexity, expressivity, usability in production

ModSecurity Core Rules in detail

modsecurity_crs_10_config.conf
modsecurity_crs_20_protocol_violations.conf
modsecurity_crs_21_protocol_anomalies.conf
modsecurity_crs_23_request_limits.conf
modsecurity_crs_30_http_policy.conf
modsecurity_crs_35_bad_robots.conf
modsecurity_crs_40_generic_attacks.conf
modsecurity_crs_45_trojans.conf
modsecurity_crs_50_outbound.conf

Dealing with false positives

```
SecRule REQUEST_URI "^\/subfolder\/index\.html(\?.*)?$"
      "phase:1,pass,ctl:ruleRemoveById=2"
```

or

```
<Location "/subfolder/">
     SecRuleRemoveById   2
</Location>
```

Note that the parameter and the operater are just an example. In reality you would write a rule that matches the request in question as exactly as possible.

Further Reading
http://www.modsecurity.org/blog/archives/2007/02/handling_false.html

This blog entry mentions the ctl:ruleRemoveById action as a feature request. This has been implemented in the meantime.

Successful Debugging thanks to Server Side Includes in error pages

```
LoadModule     include_module  modules/mod_include.so
ErrorDocument 403 /errordocs/403.shtml

<Location /errordocuments/>
     Options +IncludesNoExec
     AddType text/html .shtml
     AddOutputFilter Includes .html
</Location>

File: ./errordocuments/error.html
<html>
<!--#echo var="DATE_LOCAL" -->
<!--#echo var="UNIQUE_ID" -->
</html>
```

Core Rules **Rule IDs**

Reserved ranges

11-99,999	Reserved for local (internal) use. Use as you see fit but do not use this range for rules that are distributed to others.
100,000-199,999	Reserved for internal use of the engine, to assign to rules that do not have explicit IDs.
200,000-299,999	Reserved for rules published at modsecurity.org.
300,000-399,999	Reserved for rules published at gotroot.com.
400,000-419,999	Unused (available for reservation).
420,000-429,999	Reserved for ScallyWhack.
430,000-899,999	Unused (available for reservation).
900,000-999,999	Reserved for the Core Rules project.
11,000,000 and above	Unused (available for reservation).

Debug log

```
SecDebugLog         /path/to/file
SecDebugLogLevel    0-9
```

Levels 1 - 3 are always sent to the Apache error log.
Beware of LogLevels above 3. This is in the Gigabyte range very, very quickly. The Debug Log is not meant to be use on production servers.

Audit Log

Serial audit log

```
SecAuditEngine          RelevantOnly
SecAuditLogType         Serial
SecAuditLogParts        "ABFHKZ"
SecAuditLog              /path/to/file.log
```

Concurrent audit log

```
SecAuditEngine          RelevantOnly
SecAuditLogType         Concurent
SecAuditLogParts        "ABFHKZ"
SecAuditLog              /path/to/file.log
SecAuditLogStorageDir    /path/to/dir
```

File structure under /path/to/dir with concurrent audit log

```
./20080615
...
./20080615/20080615-1646
./20080615/20080615-1646/20080615-164615-lSkAb38AAAEAAAX4A7wAAAAA
./20080615/20080615-1646/20080615-164645-HGf7dfKJHFEDF78HDJXXHDUU
./20080615/20080615-1646/20080615-164657-JHDS7874jJHJFUE888AAGJKL
./20080615/20080615-1647
./20080615/20080615-1647/20080615-164703-KLOI787GFHGDSJtzGF455DGH
./20080615/20080615-1647/20080615-164612-3TZGD5gez677BBGDFATT99H1
...
```

The different log parts in detail

A - audit log header - mandatory

```
[15/Jun/2008:16:50:45 +0200] os1XKX8AAAEAAAZKAssAAAAA \
 192.168.1.10 56754 192.168.1.112 80
```

Format:

```
[Timestamp] Unique-Request-ID \
  Client-IP Client-Port Server-IP Server-Port
```

B - request line and request headers

C - request body
Present only if the request body exists and ModSecurity is configured to intercept it: SecRequestBodyAccess On

D - RESERVED for intermediary response headers
not implemented yet.

E - intermediary response body
present only if ModSecurity is configured to intercept response bodies, and if the audit log engine is configured to record it. SecResponseBodyAccess On

F - final response headers
excluding the Date and Server headers, which are always added by Apache in the late stage of content delivery.

G - RESERVED for the actual response body
not implemented yet.

H - audit log trailer

```
Message: Access denied with redirection to /action.php
using status 307 (phase 2). Pattern match ".*" at
ARGS_GET:debug. [file "/data/custom-apaches/apache-2.2.8-
worker/conf/httpd.conf"] [line "136"] [id "1"] [msg
"Request with debug querystring"]
Action: Intercepted (phase 2)
Stopwatch: 1213541217009775 24311 (316 542 610)
Response-Body-Transformed: Dechunked
Producer: ModSecurity for Apache/2.5.4
(http://www.modsecurity.org/).
Server: Apache
```

I – Meta request body, like C, but without files

J – RESERVED – reserved for future files upload information

K - This part contains a full list of every rule that matched

```
SecRule "REQUEST_METHOD" "@rx POST" \
     "id:7,deny,status:501,msg:'Post method used'"
```

Z - final boundary, signifies the end of the entry - mandatory

A closer look

Talk about the characteristics of the Hypertext Transfer Protocol

Keywords:
RFC 2616, Stateless, Request Methods, Persistent Connections, SSL (HTTPS), Request Line, Request Header, Request Body, Status Line, Response Header, Response Body

The eight standard HTTP methods

HEAD
GET
POST
PUT
DELETE
TRACE
OPTIONS
CONNECT

The most frequent request headers

Host
Referer
User-Agent
Accept
Accept-Language
Accept-Encoding
Accept-Charset
Keep-Alive
Connection
Cookie
If-Modified-Since
If-None-Match
Cache-Control
Via
X-Forwarded-For
From
Content-Length
Content-Type

The usual three content types defining the request body

application/x-www-form-urlencoded
multipart/form-data
text/xml

How does Apache work with HTTP

The Apache Request Cycle

http://stein.cshl.org/~lstein/talks/perl_conference/apache_api/lifecycle.html

Request is disassembled and parsed in multiple stages.
Various modules get into action in various phases.
Traditionally, the body is less interesting to Apache.

Apache Hooks

ModSecurity within the Apache Request Cycle:
http://modsecurity.org/documentation/modsecurity-apache/2.5.5/apache_request_cycle-modsecurity.jpg

ModSecurity Phases

phase 1: header read
phase 2: body process (incl. auth and everything)
phase 3: header write
phase 4: body write
phase 5: logging

Sometimes, you have to experiment with the phases to get the desired result. There are a few pecularities.

The Timestamps within the Audit-Log part H

(It has been mentioned, that these will change in the future)

```
Stopwatch: 1213541217009775 24311 (316 542 610)
```

That is microtimestamp duration \
  (ModSTime1 ModSTime2 ModSTime3)

Microtimestamp: Unix Timestamp in microseconds
Duration: Apache duration
ModSTime1: ~ Before phase 2
ModSTime2: ~ After phase 2, before the application
ModSTime3: ~ Before the response is written

Further Reading:
Ivan Ristić, Apache Security, O'Reilly, 2005, around page 180.

ModSecurity in the Enterprise

Use, PCI 6.6, Community, Commercial Alternatives

Uses of ModSecurity
http://blog.modsecurity.org/2008/03/web-application.html
http://blog.modsecurity.org/2008/03/web-applicati-1.html

OWASP: Use of Web Application Firewalls
https://www.owasp.org/index.php/Category:OWASP_Best_Practices:_Use_of_Web_Application_Firewalls

Web Application Firewall Evaluation Criteria
http://www.webappsec.org/projects/wafec/
New release approaching, hopefully

Threat Modelling or Enummerating Badness?
Know yourself and know your enemy

What is a good deployment?
Transparent for the user and the backend application, easy to understand and simple to operate and maintain (update, debug and monitor)

Updating the rules, monitoring with the console, reporting

Performance issues
Keep an eye on the timestamps
Try to keep ModSecurity below 50 milliseconds. That's what Ivan suggests and he should know.
(This is more or less defined as the difference between ModSTime1 and ModSTime2)

Be careful with big HTTP Responses. If you have responses with a size of several MBs, then rules matching RESPONSE_BODY become very slow. (→ Core Rules: modsecurity_crs_45_trojans.conf, modsecurity_crs_50_outbound.conf)

Example Configuration within Apache

```
# ModSecurity Base config

SecRuleEngine            On          (Start with DetectionOnly)
SecRequestBodyAccess     On
SecRequestBodyLimit      1048576           (this is fairly low)
SecResponseBodyAccess    On
SecResponseBodyMimeType  (null) text/html text/plain text/xml
SecResponseBodyLimit     524288            (this is fairly low)

SecDebugLog              /path/to/modsecurity_debug.log
SecDebugLogLevel         0

SecAuditEngine           RelevantOnly
SecAuditLogType          Concurrent
SecAuditLogParts         "ABFHKZ"
SecAuditLog              /path/to/modsecurity_audit.log
SecAuditLogStorageDir    /path/to/modsecurity_audit_store

SecDataDir               /path/to/modsecurity_data
SecUploadDir             /path/to/modsecurity_upload
SecTmpDir                /path/to/modsecurity_tmp

Include                  /path/to/core-rules/*
Include                  /path/to/shared-enterprise-rules/*

# ModSecurity Service Specific Rules

SecRule REQUEST_METHOD ...
...

<LocationMatch ...>

...

</LocationMatch>
```

Note that it is a smart thing to install start in detection-only mode and examine the behaviour for a while.

Make sure you keep this setup simple.
Make sure you have a well documented upgrade path for the core rules.
Make sure you have an automated testsuite or at least a testing protocol of your application.
Make sure you keep Core Rules, Shared Enterprise Rules and Service Specific Rules seperate.
Make sure you have a good documentation regarding your Service Specific Rules. You absolutely need to know all these rules and why they are there. It is best to test them in your automatic testsuite of the application.
Make sure you have a clear path for doing Virtual Patching and you are experienced to do it in time.

More complex rules

Regexes in the Variable name

```
SecRule ARGS_POST|!ARGS_POST:post_form_freetext ...
SecRule ARGS_POST:'/^post_form_[a-z0-9_]{1,16}$/' ...
```

Looking up remote address in a RBL

```
SecRule REMOTE_ADDR "@rbl xbl.spamhaus.org" \
     "id:8,drop,msg:'client listed at spamhouse'"
```

Persistent Collection

```
SecAction      "phase:1,nolog,pass,\
                    initcol:ip=%{REMOTE_ADDR}"

SecRule REQUEST_URI "^/loginfailed\.html$" \
     "phase:1,chain,setvar:ip.auth_failure_count=+1,\
     expirevar:ip.auth_failure_count=300"
SecRule REQUEST_HEADERS:Referer   "login\.cgi"

SecRule IP:auth_failure_count "@ge 10" \
     "phase:1,drop,msg:'Too many login failures'"
```

Block request when reaching threshold

```
SecAction      "phase:1,id:1,setvar:tx.score=0"

SecRule REQUEST_HEADERS:'user-agent'    "bot" \
     "phase:1,id:2,t:lowercase,pass,setvar:tx.score=+5"

SecRule ARGS_POST:debug  "all" \
     "phase:2,id:3,pass,setvar:tx.score=+5"

SecRule   TX:SCORE  "@ge 10"  \
     “phase:2,id:4,deny,msg:'Threshold reached'”
```

Alternatively, send him to sandbox

```
SecRule    TX:SCORE  "@ge 10"  \
     "phase:2,id:4,proxy:http://sandbox.example.com, \
     msg:'Threshold reached, sending to sandbox'"
```

See the documentation for more examples

Credit Card Number Detection

```
SecRule RESPONSE_BODY "@verifyCC \d{13,16}" \
     "phase:4,sanitiseMatched,log,auditlog,pass,\
     msg:'Potential credit card number'"
```

GeoIP

```
SecRule REMOTE_ADDR "@geoLookup" \
     "chain,deny,msg:'Non-Swiss IP address'"
SecRule GEO:COUNTRY_CODE "!@streq CH"
```

Alternative Variables

COUNTRY_NAME	The full country name.
COUNTRY_CONTINENT	The two character continent that the country is located. EX: EU
REGION	The two character region For US, this is state
CITY:	The city name
POSTAL_CODE:	The postal code
LATITUDE:	The latitude
LONGITUDE:	The longitude

XML

```
SecRule XML "@validateDTD /path/to/xml-scheme.dtd"
```

You can do much more XML stuff, but I lack the experience. See the documentation for more examples.

Remo

Practical Whitelisting with ModSecurity

Remo is RuleEditor for ModSecurity

OWASP Europe 2008 presentation with a bit more time

Playing around with the latest development version:

Model a simple application
Deploy the ruleset
Use the application
Get blocked
Import the audit log
Check it
Update the model of the application
Continue to deploy the ruleset