ModSecurity

Lance Buttars @Nemus801

Updated Slides @

http://www.obscuritysystems.com

Nemus@dc801.org

What is a (Web Application Firewall) or WAF

An application firewall is a form of firewall that controls input, output, and/or access from, to, or by an application or service. It operates by monitoring and potentially blocking the input, output, or system service calls that do not meet the configured policy of the firewall. The application firewall is typically built to control all network traffic on any OSI layer up to the application layer. It is able to control applications or services specifically, unlike a stateful network firewall, which is - without additional software - unable to control network traffic regarding a specific application. There are two primary categories of application firewalls, network-based application firewalls and host-based application firewalls - Wikipeda

Quote from Infosecinstitute

“In today’s world, over 70% of all attacks carried out are done so at the web application level, so we need to implement security at multiple levels, as organizations need all the help they can get in making their systems secure. Web application firewalls are deployed to establish an external security layer that increases security and detects and prevents attacks before they reach the web application”

http://resources.infosecinstitute.com/configuring-modsecurity-firewall-owasp-rules/

What can ModSecurity Do?

- Real-time application security monitoring and access control.
- Virtual Patching.
- Full HTTP traffic logging.
- Continuous passive security assessment.
- Web Application Hardening.

https://modsecurity.org/about.html

Install ModSecurity on Centos

```
sudo yum install epel #Fedora third party repo

sudo yum install mod_security

sudo httpd -M | grep security
```

Download OWASP Core Rule Set

```
sudo mkdir /etc/httpd/crs

cd /etc/httpd/crs

sudo wget https://github.com/SpiderLabs/owasp-modsecurity-crs/tarball/master

sudo tar -xvf master

sudo mv SpiderLabs-owasp-modsecurity-crs-* owasp-modsecurity-crs
```

Enable Rules

```
cd owasp-modsecurity-crs/

sudo cp modsecurity_crs_10_setup.conf.example modsecurity_crs_10_setup.conf

sudo vim /etc/httpd/conf/httpd.conf

sudo cp -r /etc/httpd/crs/owasp-modsecurity-crs/.
/etc/httpd/modsecurity.d/activated_rules/.

Sudo cp /etc/httpd/crs/owasp-modsecurity-crs/modsecurity_crs_10_setup.conf
/etc/httpd/modsecurity.d/.

sudo systemctl restart httpd
```

Test ModSecurity

curl -i http://127.0.0.1/etc/shadow -A “”

tail -n 1 /var/log/httpd/error_log

[Wed Oct 12 00:27:43.861556 2016] [:error] [pid 10655] [client 127.0.0.1] ModSecurity: Access denied with code 403 (phase 2). Pattern match "^[\\\\d.:]+$" at REQUEST_HEADERS:Host. [file "/etc/httpd/modsecurity.d/activated_rules/modsecurity_crs_21_protocol_anomalies.conf"] [line "98"] [id "960017"] [rev "2"] [msg "Host header is a numeric IP address"] [data "127.0.0.1"] [severity "WARNING"] [ver "OWASP_CRS/2.2.9"] [maturity "9"] [accuracy "9"] [tag "OWASP_CRS/PROTOCOL_VIOLATION/IP_HOST"] [tag "WASCTC/WASC-21"] [tag "OWASP_TOP_10/A7"] [tag "PCI/6.5.10"] [tag "http://technet.microsoft.com/en-us/magazine/2005.01.hackerbasher.aspx"] [hostname "127.0.0.1"] [uri "/etc/shadow"] [unique_id "V-27v7vMEWFWKKJC8VWrrQAAAAA"]

Configuration Options mod_security.conf

https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual

https://github.com/SpiderLabs/ModSecurity/blob/master/modsecurity.conf-recommended

SecRuleEngine

If you have a pre-existing site run ModSecurity in detection mode first and read the log files to look for false positives.

SecRuleEngine On|Off|DetectionOnly

Default Content Type of “Text/xml”

```
SecRule REQUEST_HEADERS:Content-Type "text/xml" \
"id:'200000',phase:1,t:none,t:lowercase,pass,nolog,ctl:requestBodyProcessor=XML"
```

- Additionally, if you are using other types of HTTP content to send and parse data, you’ll need to define a parser to interpret the data.
- You will want to replace this with the following.

Enable XML request body parser.
Initiate XML Processor in case of xml content-type

```
SecRule REQUEST_HEADERS:Content-Type "(?:text|application)/xml" \
"id:'200000',phase:1,t:none,t:lowercase,pass,nolog,ctl:requestBodyProcessor=XML"
```

Enable JSON request body parser (not turned on by default).

Initiate JSON Processor in case of JSON content-type; change accordingly if your application does not use 'application/JSON.'

```
SecRule REQUEST_HEADERS:Content-Type "application/json" \
\"id:'200001',phase:1,t:none,t:lowercase,pass,nolog,ctl:requestBodyProcessor=JSON"
```

HTTP Body

SecRequestBodyAccess On

- Allows ModSecurity to access the request body of HTTP requests. If it's not enabled ModSecurity won't be able to see any POST parameters

SecRequestBodyLimit 13107200 #LIMIT_IN_BYTES

- Anything over the limit will be rejected with status code 413 (Request Entity Too Large). There is a hard limit of 1 GB.

SecRequestBodyNoFilesLimit 131072 #NUMBER_IN_BYTES

- Maximum request body size we will accept for buffering. If your application support sfile uploads then the value given on the first line has to be as large as the largest file you want to accept. The second value refers to the size of data, with files excluded. You want to keep that value low if possible.

Store up to 131072 bytes of request body data in memory. When the multipart parser reaches this limit, it will start using your hard disk for storage. That is slow but unavoidable.

SecRequestBodyInMemoryLimit 131072

What to do if the request body size is above our configured limit.
Keep in mind that this setting will automatically be set to ProcessPartial when SecRuleEngine is set to DetectionOnly mode to minimize disruptions when initially deploying ModSecurity.

ProcessPartial Means it will parse and process the first 131072 bytes

SecRequestBodyLimitAction Reject

Verify that we've correctly processed the request body.
As a rule of thumb, when failing to process a request body
you should reject the request (when deployed in blocking mode)
or log a high-severity alert (when deployed in detection-only mode).

```
SecRule REQBODY_ERROR "!@eq 0" \

"id:'200002', phase:2,t:none,log,deny,status:400,msg:'Failed to parse request body.',logdata:'%{reqbody_error_msg}',severity:2"
```

By default be strict with what we accept in the multipart/form-data request body. If the rule below proves to be too strict for your environment consider changing it to detection-only. You are encouraged _not_ to remove it altogether.

```
SecRule MULTIPART_STRICT_ERROR "!@eq 0" \
"id:'200003',phase:2,t:none,log,deny,status:400, \
msg:'Multipart request body failed strict validation: \
PE %{REQBODY_PROCESSOR_ERROR}, \
BQ %{MULTIPART_BOUNDARY_QUOTED}, \
BW %{MULTIPART_BOUNDARY_WHITESPACE}, \
DB %{MULTIPART_DATA_BEFORE}, \
DA %{MULTIPART_DATA_AFTER}, \
HF %{MULTIPART_HEADER_FOLDING}, \
LF %{MULTIPART_LF_LINE}, \
SM %{MULTIPART_MISSING_SEMICOLON}, \
IQ %{MULTIPART_INVALID_QUOTING}, \
IP %{MULTIPART_INVALID_PART}, \
IH %{MULTIPART_INVALID_HEADER_FOLDING}, \
FL %{MULTIPART_FILE_LIMIT_EXCEEDED}'"
```

ModSecurity encounters what feels like a boundary, but it is not. Such an event may occur when evasion of ModSecurity is attempted.

```
SecRule MULTIPART_UNMATCHED_BOUNDARY "!@eq 0" \
"id:'200004',phase:2,t:none,log,deny,msg:'Multipart parser detected a possible unmatched boundary.'"
```

PCRE Tuning We want to avoid a potential RegEx DoS condition.

```
SecPcreMatchLimit 1000
SecPcreMatchLimitRecursion 1000
```

Some internal errors will set flags in TX and we will need to look for these.
All of these are prefixed with "MSC_". The following flags currently exist:
MSC_PCRE_LIMITS_EXCEEDED: PCRE match limits were exceeded.

```
SecRule TX:/^MSC_/ "!@streq 0" \
        "id:'200005',phase:2,t:none,deny,msg:'ModSecurity internal error flagged: %{MATCHED_VAR_NAME}'"
```

Allow ModSecurity to access response bodies. You should have this directive enabled to identify errors and data leakage issues. Do keep in mind that enabling this directive does increase both memory consumption and latency.

SecResponseBodyAccess On

Which response MIME types do you want to inspect? You should adjust the configuration below to catch documents but avoid static files.

SecResponseBodyMimeType text/plain text/html text/xml application/json

Buffer response bodies of up to 524288 bytes in length.

SecResponseBodyLimit 524288

What happens when we encounter a response body larger than the configured limit? By default, we process what we have and let the rest through.
That's somewhat less secure but does not break any legitimate pages.

SecResponseBodyLimitAction ProcessPartial

The location where ModSecurity stores temporary files (for example, when it needs to handle a file upload that is larger than the configured limit).
This default setting is chosen due to all systems have /tmp available, however, this is less than ideal. It is recommended that you specify a location that's private.

SecTmpDir /tmp/

The location where ModSecurity will keep its persistent data. This default setting is chosen due to all systems have /tmp available, however, it too should be updated to a place that other users can't access.

SecDataDir /tmp/

```
# The location where ModSecurity stores intercepted uploaded files. This
# location must be private to ModSecurity. You don't want other users on
# the server to access the files, do you?

SecUploadDir /opt/modsecurity/var/upload/

# By default, only keep the files that were determined to be unusual
# in some way (by an external inspection script). For this to work you,
# will also need at least one file inspection rule.

SecUploadKeepFiles RelevantOnly

# Uploaded files are by default created with permissions that do not allow
# any other user to access them. You may need to relax that if you want to
# interface ModSecurity to an external program (e.g., an anti-virus).

SecUploadFileMode 0600
```

Messages at levels 1–3 are always copied to the Apache error log. Therefore you can always use level 0 as the default logging level in production if you are very concerned with performance. Having said that, the best value to use is 3. Higher logging levels are not recommended in production because of load issues.

SecDebugLog /opt/modsecurity/var/log/debug.log
SecDebugLogLevel 0

- 0: no logging
- 1: errors (intercepted requests) only
- 2: warnings
- 3: notices
- 4: details of how transactions are handled
- 5: as above, but including information about each piece of information handled
- 9: log everything, including very detailed debugging information

```
# Log the transactions that are marked by a rule, as well as those that trigger a
server error (determined by a 5xx or 4xx, excluding 404,  level response statuses

SecAuditEngine RelevantOnly
SecAuditLogRelevantStatus "^(?:5|4(?!04))"

# Log everything modsecurity knows about a transaction.
SecAuditLogParts ABIJDEFHZ

# Use a single file for logging. This is much easier to look at, but
assumes that you will use the audit log only occasionally.
SecAuditLogType Serial
SecAuditLog /var/log/modsec_audit.log

# Specify the path for concurrent audit logging.
SecAuditLogStorageDir /opt/modsecurity/var/audit/
```

A: Audit log header (mandatory).

B: Request headers.

C: Request body (present only if the request body exists and ModSecurity is configured to intercept it. This would require SecRequestBodyAccess to be set to 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. Intercepting response bodies requires SecResponseBodyAccess to be enabled). Intermediary response body is the same as the actual response body unless ModSecurity intercepts the intermediary response body, in which case the actual response body will contain the error message.

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.

I: This part is a replacement for part C. It will log the same data as C in all cases except when multipart/form-data encoding in used. In this case, it will log a fake application/x-www-form-urlencoded body that contains the information about parameters but not about the files. This is handy if you don’t want to have (often large) files stored in your audit logs.

J: This part contains information about the files uploaded using multipart/form-data encoding.

K: This part contains a full list of every rule that matched (one per line) in the order they were matched. The rules are fully qualified and will thus show inherited actions and default operators. Supported as of v2.5.0.

Z: Final boundary, signifies the end of the entry (mandatory).

Use the most commonly used application/x-www-form-urlencoded parameter separator. There's probably only one application somewhere that uses something else so don't expect to change this value.

```
SecArgumentSeparator &
```

Settle on version 0 (zero) cookies, as that is what most applications use. Using an incorrect cookie version may open your installation to evasion attacks (against the rules that examine named cookies).

```
SecCookieFormat 0
```

Specify your Unicode Code Point.
This mapping is used by the t:urlDecodeUni transformation function to properly map encoded data to your language. Properly setting these directives helps to reduce false positives and negatives.

SecUnicodeMapFile unicode.mapping 20127

Improve the quality of ModSecurity by sharing information about your current ModSecurity version and dependencies versions.
The following information will be shared: ModSecurity version, Web Server version, APR version, PCRE version, Lua version, Libxml2 version, Anonymous unique id for a host.

SecStatusEngine Off # By default it is off you can turn it on if you like.

Configuring CRS

/etc/httpd/modsecurity.d/modsecurity_crs_10_setup.conf

OWASP CRS Notes

- CRS does not configure ModSecurity features such as the rule engine, the audit engine, logging etc. This task is part of the ModSecurity initial setup.
- Use recommend configuration from previous slides for initial setup.
  - https://github.com/SpiderLabs/ModSecurity/blob/master/modsecurity.conf-recommended
- SecDefaultAction in CRS will redirect to your local domain when an alert is triggered. Beware of this as it can cause redirect loops.
- Decide what you want ModSecurity it do when it triggers an activity such as:
  - Drop the request.
  - Return a http status 403.
  - Go to a custom warning page.

CRS Modes

Anomaly mode is where the complete chain of rules is evaluated during each phase of request processing, and an overall score is generated to decide whether to block the request or not. If you want to run the rules in Anomaly Scoring Mode then set The Core Rule Set is best used in a**nomaly scoring mode**.

```
SecDefaultAction pass
```

Traditional mode is the alternative to anomaly scoring mode the first rule that matches with the block action will execute the default action, which is normally set to “deny”.

```
SecDefaultAction "phase:1,deny,log"
SecDefaultAction "phase:2,deny,log"
```

Collaborative Detection Severity Levels - **Anomaly mode**

These are the default Severity ratings (with anomaly scores) of the individual rules -

- Critical - Anomaly Score of 5 Is the highest severity level possible without correlation. It is normally generated by the web attack rules (40 level files).
- Error - Anomaly Score of 4 Is generated mostly from outbound leakage rules (50 level files).
- Warning - Anomaly Score of 3 Is generated by malicious client rules (35 level files).
- Notice - Anomaly Score of 2 Is generated by the Protocol policy and anomaly files.

```
# These are the default Severity ratings

SecAction \
  "id:'900001', \
  phase:1, \
  t:none, \
  setvar:tx.critical_anomaly_score=5, \
  setvar:tx.error_anomaly_score=4, \
  setvar:tx.warning_anomaly_score=3, \
  setvar:tx.notice_anomaly_score=2, \
  nolog, \

  pass"
```

Detection Scoring Initialization and Threshold Levels

If set to "5" it will work similarly to previous Mod CRS rules and will create an event in the error_log file if there are any rules that match.

If you would like to lessen the number of events generated in the error_log file, you should increase the anomaly score threshold to something like "20".

This would only generate an event in the error_log file if there are multiple lower severity rule matches or if any 1 higher severity item matches.

```
SecAction \
 "id:'900003', \
 phase:1, \
 t:none, \
 setvar:tx.inbound_anomaly_score_level=5, \
 setvar:tx.outbound_anomaly_score_level=4, \
 nolog, \
 pass"
```

Anomaly_score_blocking=on

This is a collaborative detection mode where each rule will increment an overall anomaly score for the transaction. (make sure to uncomment)

- The scores are then evaluated in the following files:
  Inbound anomaly scores - checked in the modsecurity_crs_49_inbound_blocking.conf file
- Outbound anomaly score s- checked in the modsecurity_crs_59_outbound_blocking.conf file

```
SecAction \
  "id:'900004', \
  phase:1, \
  t:none, \
  setvar:tx.anomaly_score_blocking=on, \
  nolog, \
  pass"
```

Other CRS Rules to Consider

900006 Maximum number of arguments in
max_num_args=255

900007 Limit argument name length
arg_name_length=100

900008 Limit value name length
arg_length=400

900009 Limit arguments total length
total_arg_length=64000

900012 restricted_extensions

restricted_extensions=.asa/ .asax/ .ascx/ .axd/ .backup/ .bak/ .bat/ .cdx/ .cer/ .cfg/ .cmd/ .com/ .config/ .conf/ .cs/ .csproj/ .csr/ .dat/ .db/ .dbf/ .dll/ .dos/ .htr/ .htw/ .ida/ .idc/ .idq/ .inc/ .ini/ .key/ .licx/ .lnk/ .log/ .mdb/ .old/ .pass/ .pdb/ .pol/ .printer/ .pwd/ .resources/ .resx/ .sql/ .sys/ .vb/ .vbs/ .vbproj/ .vsdisco/ .webinfo/ .xsd/ .xsx/', \

Writing Rules

Rule Format

DIRECTIVE VARIABLES OPERATOR ACTIONS

- DIRECTIVE - Action to be taken by mod_security.
- VARIABLES
  - Parsed information from http request and http response.
    - https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual#Variables
- OPERATOR
  - Regular expression or some way of interpreting the variables.
    - https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual#Operators
- ACTIONS
  - What to do if the rule matches.
    - https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual#Actions

Directives

SecAction: Unconditionally processes the action list it receives as the first and only parameter.

SecDefaultAction: Defines the default list of actions, which will be inherited by the rules in the same configuration context.

SecMarker: Adds a fixed rule marker that can be used as a target in a skipAfter action. A SecMarker directive essentially creates a rule that does nothing and whose only purpose is to carry the given ID.

SecRule Creates a rule.

SecRuleInheritance: Configures whether the current context will inherit the rules from the parent context.

SecRuleRemoveById: Removes the matching rules from the current configuration context.

SecRuleRemoveByMsg: Removes the matching rules from the current configuration context.

SecRuleScript: Description: This directive creates a special rule that executes a Lua script to decide whether to match or not.

SecRuleUpdateActionById: This directive will overwrite the action list of the specified rule with the actions provided in the second parameter.

Phases

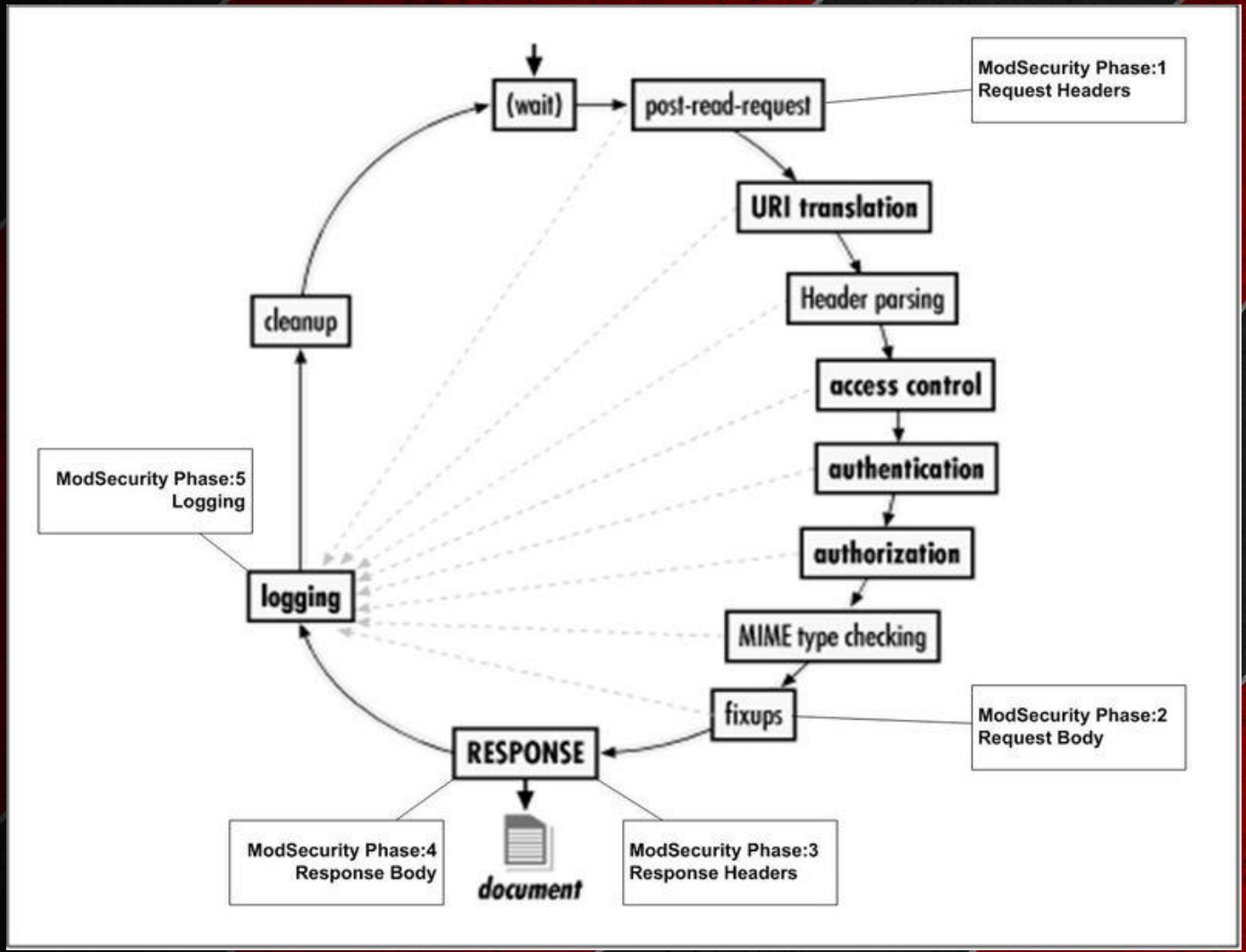
Request headers (REQUEST_HEADERS) - Phase 1

Request body (REQUEST_BODY) - Phase 2

Response headers (RESPONSE_HEADERS) - Phase 3

Response body (RESPONSE_BODY) - Phase 4

Logging (LOGGING) - - Phase 5


(wait)
post-read-request
ModSecurity Phase:1
Request Headers
URI translation
Header parsing
access control
authentication
authorization
MIME type checking
fixups
ModSecurity Phase:2
Request Body
RESPONSE
ModSecurity Phase:3
Response Headers
document
ModSecurity Phase:4
Response Body
logging
ModSecurity Phase:5
Logging
cleanup

Example Rules

SecRule REQUEST_URI "sausage" "id:'80000',rev:1,log,deny,msg:'sausage is not allowed'"

SecRule ARGS:user "@rx ^(test|land)$" "id:'80002',rev:1,msg:'test or land is not allowed',log,deny"

Example Match of Custom Rule

curl -i http://127.0.0.1/sausage

[Wed Oct 12 03:31:43.428512 2016] [:error] [pid 20462] [client 127.0.0.1] ModSecurity: Access denied with code 403 (phase 2). Pattern match "sausage" at REQUEST_URI. [file "/etc/httpd/modsecurity.d/activated_rules/nemus_rules.conf"] [line "1"] [id "80000"] [rev "1"] [msg "sausage is not allowed"] [hostname "127.0.0.1"] [uri "/sausage"] [unique_id "V-3m3-CDibgrMZ5scNrNTwAAAAQ"]

Rules in Lua

You can do most things with the ModSecurity ruleset, but it's not a full language.

Writing rules in Lua gives you the power to do whatever you want with a request that can be done in a full feature programming language.

In ModSecurity Lua is implemented as a rule language add-on.

To use LUA use the SecRuleScript directive.

```
SecRuleScript /opt/modsecurity/myscript.lua phase:1,log,deny
```

Lua Script

Lua rule needs an entry point that ModSecurity can find which is the main function.

```
function main()
    return nil;
end
```

The above script only returns nil, which means that there is no match.

For a Lua rule to match, it needs to return a message:

```
function main()
    return "Matched";
end
```

Getvar

A call to the m.getvar() function will retrieve the variable named in the function call

```
function main()

        local remote_ip   = m.getvar("REMOTE_ADDR");
        local first_name  = m.getvar("ARGS.first_name");
        local last_name  = m.getvar("ARGS.last_name");

        if(first_name == ‘’Homer”) then
                return “No Homers are allowed” .. remote_ip;
        end
        return nil;
end
```

Lua as an ACTION

```
SecRule ARGS test phase:2,log,pass,exec:/opt/modsecurity/myscript.lua
```

Handling False Positives

(Exception Handling aka Whitelisting)

https://samhobbs.co.uk/2015/09/example-whitelisting-rules-apache-modsecurity-and-owasp-core-rule-set

Basic Whitelisting

#You can put them inside any Apache config file or a VirtualHost file.

Include /etc/modsecurity/whitelists/application1_list.conf

SecRuleRemoveById 981204

#more specific to a resource
<LocationMatch "^/orders/[0-9]+/approve$">
SecRuleRemoveById 981204
</LocationMatch>
The methods of whitelisting works in the beginning, but overtime you'll end up with CRS turned off or broken.

Specific Whitelisting

When ModSecurity is in traditional mode when it hits the first rule that matches with a block action it will execute the SecDefaultAction, which blocks the request

To prevent a rule from causing a request to be denied, but still make sure it's still logged, you can enter the SecRuleUpdateActionById and set it to “pass”

```
SecRuleUpdateActionById 981204 "pass"
```

Specific Whitelisting part 2

SecRuleUpdateTargetById, SecRuleUpdateTargetByMsg and SecRuleUpdateTargetByTag

“Each ModSecurity rule specifies a list of variables to be tested against the operator (e.g. @rx, @beginsWith, @streq etc.). If you want to add additional variables to the list to be inspected, or remove a particular one that is causing a problem, you can use one of these parameters.” - samhobbs

“The following rule would remove the comment_body argument from the list inspected by rule 981143:” - samhobbs

SecRuleUpdateTargetById 981143 !ARGS:comment_body

https://samhobbs.co.uk/2015/09/example-whitelisting-rules-apache-modsecurity-and-owasp-core-rule-set

Setup Security Page

It is inevitable that users will be blocked when trying to use a legitimate resource. To make things easier on your clients it's recommended to setup a custom error document when they are blocked. This makes it less frustrating for them than a standard "500: Internal Server Error" that has no explanation of what caused the issue.

```
SecRuleUpdateActionById 981204 "chain,deny,status:501"

# also update the outbound blocking rules
SecRuleUpdateActionById 981204 "chain,deny,status:501"
```

Add the following to your httpd.conf or apache config and create a page for it

```
ErrorDocument 501 /security-error.php
```

DetectionOnly mode for new content

When getting started with ModSecurity your entire site will be in DetectionOnly mode while you remove false positives.

At some point, you may want to add a new content to your server after your initial whitelisting is finished and now have ModSecurity in blocking mode. In this scenario, you only want to put the engine in DetectionOnly mode for just the new content.

```
SecRule REQUEST_URI
"@beginsWith /new/content" \
 "id:'0000100', \
 phase:1, \
 t:none, \
 ctl:ruleEngine=DetectionOnly, \
 nolog, \
 pass"
```

Missing HTTP only flag in cookies

If you see a lot of misconfiguration messages that fill up audit log you might have forgotten to configure your site to set the HTTP only flag in your cookies.

To resolve this issue by changing the following parameter in /etc/php.ini for Centos.

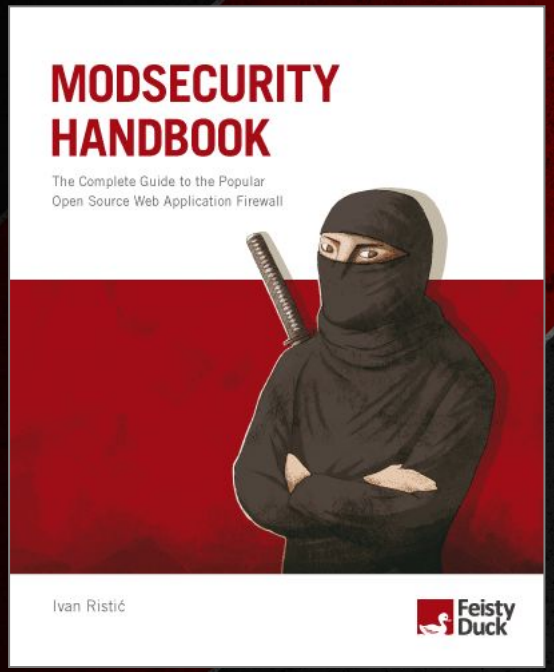
```
session.cookie_httponly = 1
```

Project Honeypot

ModSecurity supports Project Honeypot http://www.projecthoneypot.org/index.php blacklists.

This is a great project and all you need to do to leverage it is sign up for an API key http://www.projecthoneypot.org/httpbl_api.php

Recommend Book



The definitive guide to the popular open source web application firewall, by Ivan Ristić, the principal author of ModSecurity

https://www.feistyduck.com/books/modsecurity-handbook/

Sources

http://blog.modsecurity.org/2007/02/handling-false.html

https://www.feistyduck.com/books/modsecurity-handbook/modsecurity-rule-writing-workshop.pdf

https://samhobbs.co.uk/2015/09/example-whitelisting-rules-apache-modsecurity-and-owasp-core-rule-set

http://resources.infosecinstitute.com/configuring-modsecurity-firewall-owasp-rules/

http://stackoverflow.com/questions/33989273/modsecurity-excessive-false-positives/34027786#34027786

https://www.trustwave.com/Resources/SpiderLabs-Blog/Advanced-Topic-of-the-Week--Traditional-vs--Anomaly-Scoring-Detection-Modes/

Sources Part 2

https://www.trustwave.com/Resources/SpiderLabs-Blog/ModSecurity-Advanced-Topic-of-the-Week--(Updated)-Exception-Handling/

http://linoxide.com/security/securing-centos-7-modsecurity/

http://resources.infosecinstitute.com/analyzing-mod-security-logs/