ModSecurity 2 Rule Language

Processing Phases

- ModSecurity splits processing into 5 processing phases:
  1. Request Headers
  2. Request Body
  3. Response Headers
  4. Response Body
  5. Logging
- This many phases allow you to decide what you want to happen at key points of transaction processing.

Rule Syntax

- The most used directive is SecRule:

SecRule VARIABLES OPERATOR [ACTIONS]

- This directive will:

1. Expand collection variables from the VARIABLES section.
2. Apply the operator as specified in the OPERATOR section to the expanded variables.
3. One rule will trigger once for a match in every variable.
4. A match will either execute the per-rule actions, or perform the default actions.

Simple Rule

- In the simplest case:

```
SecRule REQUEST_URI aaa
```

- The above will look for the pattern aaa in the variable REQUEST_URI.
- The pattern is a regular expression.
- A similar pattern can be written as:

```
SecRule REQUEST_URI b{3}
```

- ModSecurity uses PCRE (http://www.pcre.org)

Multiple Variables As Targets

- There can be any number of variables in the VARIABLES section (separated by pipes):

```
SecRule "REQUEST_URI|QUERY_STRING" \
ccc
```

- Configuration directives can be split over several lines (that’s an Apache feature) by terminating the line with a backslash.
- The whitespace at the beginning of next line will become part of the directive.
- If you need to have a whitespace use double quotes to delimit parameter.

Variable Collections

- Some variables expand at runtime:

SecRule ARGS ddd

- The above will expand into variables representing individual request parameters, but only if there are parameters present.
- Only the content is examined.
- Another variable is used for the names:

SecRule ARGS_NAMES eee

- There is a variable for every bit of transaction.

Targeting Individual Parameters

- You can target individual parameters with the help of the selection operator:

SecRule ARGS:p fff

- Or you can target all parameters except the ones you specify:

SecRule ARGS|!ARGS:q ggg

- You can even use a regular expression to select the parameters (* does the opposite in beta-3):

SecRule ARGS:/^z/ hhh

Counting Variables In a Collection

- You can count how many variables there are in a collection (e.g. parameters, request headers, response headers, etc):

SecRule &ARGS !^0$

- The above triggers if there are any parameters supplied in the request.
- You might have noticed the exclamation mark; it negates the regular expression.

Variable Names (1)

- ARGS, ARGS_COMBINED_SIZE, ARGS_NAMES
- REQBODY_PROCESSOR, REQBODY_PROCESSOR_ERROR, REQBODY_PROCESSOR_ERROR_MSG
- XML
- WEBSERVER_ERROR_LOG
- FILES, FILES_TMPNAMES, FILES_NAMES, FILE_SIZES, FILES_COMBINED_SIZE
- TX
- ENV

Variable Names (2)

- REMOTE_HOST, REMOTE_ADDR, REMOTE_PORT, REMOTE_USER
- PATH_INFO, QUERY_STRING
- AUTH_TYPE
- SERVER_NAME, SERVER_PORT, SERVER_ADDR
- REQUEST_LINE, REQUEST_URI, REQUEST_METHOD, REQUEST_PROTOCOL
- REQUEST_FILENAME, REQUEST_BASENAME
- SCRIPT_FILENAME, SCRIPT_BASENAME

Variable Names (3)

- TIME, TIME_EPOCH
- TIME_YEAR, TIME_MON, TIME_DAY, TIME_HOUR, TIME_MIN, TIME_SEC, TIME_WDAY
- SCRIPT_UID, SCRIPT_GID
- SCRIPT_USERNAME, SCRIPT_GROUPNAME
- SCRIPT_MODE
- REQUEST_HEADERS, REQUEST_HEADERS_NAMES

Variable Names (4)

- REQUEST_COOKIES, REQUEST_COOKIES_NAMES
- REQUEST_BODY
- RESPONSE_LINE, RESPONSE_STATUS
- RESPONSE_PROTOCOL
- RESPONSE_HEADERS, RESPONSE_HEADERS_NAMES
- RESPONSE_BODY
- WEBAPPID, SESSIONID

Explicit Operators In Rules

- Regular expression matcher is the default operator.
- In a general case you can choose exactly which operator you want to use:

```
SecRule REQUEST_URI "@rx iii"
```

- You can still use the exclamation mark in front of the @ character (and the meaning is the same).

Supported Operators

- The following operators are supported in 2.0.0-beta-3:

eq

ge

gt

inspectFile

le

lt

rbl

rx

validateByteRange

validateDTD

validateSchema

validateUrlEncoding

validateUtf8Encoding

Operator Usage Examples

- Validate files that are uploaded:

```
SecRule FILES_TMPNAMES "@inspectFile \
/opt/apache/bin/inspect_script.pl"
```

- Check only certain bytes are used in parameters:

```
SecRule ARGS "@validateByteRange \
10,13,32-126"
```

- Validate UTF-8 encoding:

```
SecRule ARGS "@validateUtf8Encoding"
```

- Real-time Block List lookup:

```
SecRule REMOTE_ADDR "@rbl sc.surbl.org"
```

Actions

- There are five types of action:
  1. **Disruptive actions** – interrupt current transaction.
  2. **Non-disruptive actions** – change state.
  3. **Flow actions** – change rule flow.
  4. **Meta-data actions** – contain rule metadata.
  5. **Data actions** – mere placeholders for other actions.
- Usage example:

```
SecRule ARGS ddd log,deny,status:500
SecAction nolog,pass,exec:/bin/this/that.pl
```

Disruptive Actions

- Interrupt or disrupt transaction:
  - **deny** – stops transaction.
  - **drop** – drops connection
  - **redirect** – respond with a redirection.
  - **proxy** – forward request to another server.
  - **pause** – slow down execution.

Meta-data Actions

- Meta-data actions describe the rule:
  - **id** – unique rule ID.
  - **rev** – rule revision.
  - **msg** – custom message.
  - **severity** – as syslog (0-7).
  - **phase** – the phase where the rule is supposed to run.
  - **log**, **nolog** – whether or not to log the match.
  - **auditlog**, **noauditlog** – whether or not to count the match toward audit logging.

Flow Actions

- Flow actions affect how rules are processed:
  - **allow** – stop processing rules.
  - **chain** – combine the rule with the next one.
  - **pass** – ignore match in the current rule.
  - **skip** – skip over one or more rules.

Data Actions

- Data actions are helpers for other parts of the rule:
  - **capture** – used in combination with @rx to capture subexpressions.
  - **status** – which status code to use for deny, redirect.
  - **t** – defines which transformation functions need to be run against the variables.
  - **xmlns** – defines namespace for XPath expressions.

Audit Log Sanitisation Actions

- There are four actions:
  - **sanitiseArg**
  - **sanitiseMatched**
  - **sanitiseRequestHeader**
  - **sanitiseResponseHeader**
- Examples:

```
SecAction nolog,pass,sanitiseArg:p
SecAction \
nolog,pass,sanitiseRequestHeader:Authorization
SecRule ARGS secret \
nolog,pass,sanitiseMatched
```

Variable Actions

- Working with environment variables:

```
setenv:name=value
setenv:!name
```

- Working with variables:

```
setvar:tx.score=10
setvar:tx.score=+5
setvar:!tx.score
deprecatevar:session.score=60/3600
expirevar:session.blocked=3600
```

Collection Actions

- **initcol** – create a persistent collection:

```
initcol:ip=%{REMOTE_ADDR}
```

- **setsid** – initialise session storage:

```
SecRule REQUEST_COOKIES:PHPSESSID !^$ chain,nolog,pass
SecAction setsid:%{REQUEST_COOKIES.PHPSESSID}
```

- This action will initialise variable **SESSIONID**.
- Use **SecWebAppId** directive to create session storage namespace for each application.

Built-in Collection Variables

- Some variables are automatically generated:
  - CREATE_TIME
  - KEY
  - LAST_UPDATE_TIME
  - TIMEOUT
  - UPDATE_COUNTER
  - UPDATE_RATE
- Some variable names have pre-defined purpose:
  - BLOCKED
  - SCORE

Other Actions

- Execute external script:

  exec:/bin/script.pl

- Update transaction settings dynamically:
  - **ctl**
    - auditEngine
    - auditLogParts
    - debugLogLevel
    - requestBodyAccess
    - requestBodyLimit
    - requestBodyProcessor
    - responseBodyAccess
    - responseBodyLimit
  - For example:
    - **ctl:auditEngine=off**

Transformation Functions (1)

- Transformation functions will automatically convert data before matching:

  lowercase

  replaceNulls

  compressWhitespace

  replaceComments

  urlDecode

  urlDecodeUni

  base64Encode

  base64Decode

  hexDecode

  hexEncode

  htmlEntityDecode

  escapeSeqDecode

  normalisePath

  normalisePathWin

  md5

  sha1

Transformation Functions (2)

- The following is performed by default (and in this order):
  - **lowercase**
  - **replaceNulls**
  - **compressWhitespace**
- But you can change the default setting for all subsequent rules:

```
SecDefaultAction log,deny,status:500,\
t:replaceNulls,t:compressWhitespace
```

- Or, just for one rule:

```
SecRule ARG:base64 ABC t:base64decode
```

Complete XML Example (1)

- Detect XML and instruct ModSecurity to parse it:

```
# Phase 1
SecDefaultAction phase:1

# Detect XML requests and process them as XML
SecRule REQUEST_HEADERS:Content-Type ^text/xml$ \
nolog,pass,ctl:requestBodyProcessor=XML
```

Complete XML Example (2)

```
# Phase 2
SecDefaultAction phase:2

# Stop on request body processing errors
# (e.g. XML is not well formed)
SecRule REQBODY_PROCESSOR_ERROR "@eq 1"

# Validate XML against a DTD
SecRule REQBODY_PROCESSOR "^XML$ chain
SecRule XML "@validateDTD /opt/apache-frontend/conf/xml.dtd"

# Look into only one part of the XML
SecRule XML:/person/name/firstname/text() Ivan
```

THE END!

Questions?