

ModSecurity Migration Matrix

ModSecurity 2.X Changes and Migration Matrix

Version 1.0 / (March 22, 2007)

Copyright © 2004-2007 Breach Security, Inc. (<http://www.breach.com>)

Migrating from 1.X to 2.0

If you are already using an older version of ModSecurity and want to upgrade/migrate your existing custom rules, you will need to ensure that you properly translate all of your Directives to their corresponding 2.0 counterparts. Some directives have simply changed names, however some directives actually behave differently so it is important that you also review the entire 2.0 Reference Manual.

The migration matrix show below should help you to translate ModSecurity 1.X directives to the 2.0 values. There are also some notes that provide additional information is a directive significantly changed how it operates.

<u>Feature/ Capability</u>	<u>ModSecurity 1.x</u>	<u>ModSecurity 2.x</u>	<u>Notes/Comments</u>	<u>How to upgrade</u>
Apache Version Supported	Apache 1.x/2.x	Apache 2.X Only	ModSecurity 2.0 will only work with Apache 2.x and not the older 1.3 version.	If you are mainly an Apache 1.3 shop and/or you have other web servers that you want to protect (such as IIS) an alternative solution is to deploy an Apache 2.x reverse proxy server and implement ModSecurity 2.x on it.
Installation	Can be installed as either a DSO module or as a statically compiled module.	Can currently only be installed as a DSO module.	In 1.x, you could use apxs directly, while in 2.x you must use the provided Makefile.	If you can not use DSOs in your current Apache configs, you may look at implementing a front-end Apache reverse proxy server.
Configuration - IfModule	Apache 1.x - <IfModule mod_security.c> Apache 2.x - <IfModule security_module>	<IfModule security2_module>	The syntax of using IfModule has changed between Apache 1.x and 2.x.	Make sure that any existing <IfModule> directives uses the correct syntax.

Processing Phases Supported	2	5	<p>ModSecurity 1.x supports:</p> <ul style="list-style-type: none">• Inbound - which corresponds to current Mod 2.x Request Body phase and the Apache “fixups” phase.• Outbound - which corresponds to current Mod 2.x Response Body phase and just after the Apache “response” processing phase. <p>ModSecurity 2.x supports:</p> <ul style="list-style-type: none">• Request Headers – which corresponds with the Apache “post-read-request” phase.• Request Body – which corresponds with the Apache “fixups” phase.• Response Headers – which corresponds to the Apache “response” phase.• Response Body – which corresponds to just after the Apache “response” phase.• Logging	If you are translating existing 1.x rules (SecFilter/SecFilterSelective) then you should use phase:2 in the new rule syntax. Translate existing OUTPUT rules to run in phase:4.
------------------------------------	---	---	---	---

Directive to turn On/Off the Rule Engine	SecFilterEngine	SecRuleEngine ctl:ruleEngine=	<p>1.x – values were On, Off and DynamicOnly and was a Global directive.</p> <p>2.x - values are On, Off or DetectionOnly.</p> <p>2.x – the “ctl” action can control the RuleEngine dynamically for individual requests.</p>	Replace SecFilterEngine with SecRuleEngine. The DynamicOnly mode is not supported in ModSecurity 2.x because it was sometimes difficult for ModSecurity to determine if a particular request was dynamic in nature or not. Use of AddType vs. AddHandler would cause problems. Since this logic relies on the internal (and not entirely documented) workings of Apache and on the chosen configuration it also makes it somewhat unpredictable.
Directive to handle the Audit Engine	SecAuditEngine On, Off, RelevantOnly, DynamicOrRelevant	SecAuditEngine On, Off, RelevantOnly	In 2.x, the DynamicOrRelevant option was discontinued.	If you are using DynamicOrRelevant then switch it to RelevantOnly.
Default Rule Action	SecFilterDefaultAction	SecDefaultAction	<p>1.x – SecFilterDefaultAction could be used anywhere in the config and it would be picked up by all rules.</p> <p>2.x – SecDefaultAction must come before rules and be specified in each context. The default setting for this directive (if it is not specified otherwise) is –</p> <p><i>SecDefaultAction phase:2,log,deny,status:403,t:lowercase,t:replaceNulls,t:compressWhitespace</i></p>	<p>Replace SecFilterDefaultAction with SecDefaultAction. Optionally, you can group rules together where you would like to use the same action and then specify a SecDefaultAction line before each group.</p> <p>Also keep in mind that while most actions specified on individual rules will supersede those specified in SecDefaultAction, transformation functions are additive. So, if you specify a “t:base64Decode” transformation function to a rule, it will be added after the lowercase, replaceNulls and compressWhitespace transformation functions.</p>
Debug Logging	SecFilterDebugLog SecFilterDebugLogLevel	SecDebugLog SecDebugLogLevel	Name change only.	Change the names of these directives to their 2.x counterparts.

Rule Directive(s)	SecFilter SecFilterSelective	SecRule	In Mod 1.x, SecFilter and SecFilterSelective were case insensitive. In Mod 2.x, the case of data is not altered unless the lowercase transformation funce is used. SecRule has essentially the same rule syntax as SecFilterSelective,	Replace SecFilterSelective with SecRule and make sure to translate the variable tested according to this list. Replace any SecFilter with a new SecRule directive. You will need to specify a new Variable location and a phase. You can optionally specify a disruptive action, otherwise it will be inherited from a previous SecDefaultAction.
Rule Exceptions	Whitelist approach – use pass, allow actions False Positive Approach – use SecFilterRemove	Whitelist approach – use pass, allow and ctl actions. False Positive Approach – use SecRuleRemoveByld and Apache Scope context	In Mod 2.x, using the “allow” action may not be enough to truly let a request through as “allow” only applies to the current processing phase. This means that rules in subsequent phases may act on the request. This is why you need to also use the “ctl: ruleEngine=Off” action if you really want to let a request through.	See Blog post on handling false positives and creating custom rules - http://www.modsecurity.org/blog/archives/2007/02/handling_false.html
Directive to control rule inheritance to Apache Scope locations (Virtual Hosts, Location, Directory)	SecFilterInheritance	SecRuleInheritance	The best use of this directive is when you want to start with a “clean slate” so you can use SecRuleInheritance Off and then specify your new rule sets. Note – Rule Inheritance does not work across Apache Scope directives (such as Vhosts, Location and Directory directives). This means that you can not use SecRuleInheritance On to inherit a SecDefaultAction directive within these new contexts. This is an issue with the way that Apache inherits contexts. It is for this reason that we recommend that you specify new SecDefaultAction directives within each Apache scope location that you create.	Translate any existing “SecFilterInheritance Off” rules directly to “SecRuleInheritance Off”. Then replace any “SecFilterInheritance On” directives inside Apache Scope context locations with a new SecDefaultAction directive and then import the rules that you want with standard Apache Include directives.

Ability to manage rules in Apache Scope locations	SecFilterImport SecFilterRemove	SecRuleRemoveById SecRuleRemoveByMsg	SecFilterRemove is now SecRuleRemoveById or SecRuleRemoveByMsg. SecFilterImport is no longer supported.	Change all of your existing SecFilterRemove rules to SecRuleRemoveById. For any existing SecFilterImport rules, you will need to either copy the rule into the context or use an Apache Include Directive to include entire files (such as including the Core Rules files).
Ability to verify URL/UTF8 Encodings	SecFilterCheckURLEncoding SecFilterCheckUnicodeEncoding	@validateUrlEncoding @validateUtf8Encoding	In Mod 1.x, these were Global Directives and in Mod 2.x they are Operators that can be applied selectively to each rule.	Add the rules that will do exactly the same as the directives
Ability to enforce a Byte Range (allowed character set)	SecFilterForceByteRange	@validateByteRange	<p>In Mod 1.x, this was a Global Directive and in Mod 2.x it is an Operator that can be applied selectively to each rule.</p> <p>In Mod 1.x, this directive did not check POST payloads when <i>multipart/form-data</i> encoding was used.</p>	You can now add @validateByteRange operators to individual rules. This helps if you have differences in allowed character sets for different portions of the web application.
Ability to Normalize/ Transform Request Data	ModSecurity 1.x automatically applied the following transformations: <ul style="list-style-type: none"> On Windows only, convert \ to / Reduce // to / Reduce // to / Decode URL-encoded characters Converts Null Bytes to Space character 	<ul style="list-style-type: none"> base64Decode base64Encode compressWhitespace escapeSeqDecode hexDecode hexEncode htmlEntityDecode lowercase md5 	In Mod 1.x, the normalization functions were implicit and you could not control them. In Mod 2.x, not normalization is done by default. There are now “Transformation Functions” that allow you to selectively apply normalizations and other features.	You should add the appropriate transformation functions to either SecDefaultAction directive or each individual rule. See the Core Rules files for examples. <p>Keep in mind that transformation functions are inherited from parent SecDefaultAction directives. Care should be taken to ensure that RegEx patterns match the data after transformation functions are applied. In order to avoid possible unwanted inherited transformation functions, use “t:none” to either not apply any transformation functions or you can then specify specific transformation functions after “t:none”.</p>

		<ul style="list-style-type: none"> • none • normalisePath • normalisePathWin • removeNulls • removeWhitespace • replaceComments • replaceNulls • urlDecode • urlDecodeUni • urlEncode • sha1 		
Ability to specify an arbitrary Request Header in a rule	HEADER_ <i>headername</i> HTTP_ <i>headername</i>	REQUEST_HEADERS REQUEST_HEADERS: <i>headername</i> REQUEST_HEADERS:/ <i>RegEx</i> /	The HTTP_ <i>headername</i> syntax has been superseded by the new REQUEST_HEADERS: <i>headername</i> syntax and will not be supported in future releases. The advantage to using the new syntax is that you can also use RegEx in the <i>headername</i> portion.	Translate any existing HTTP_ <i>headername</i> directives to REQUEST_HEADERS: <i>headername</i> . Also consider consolidating header checks by using Regular Expressions in the header name portion of the Variable.
Variable/ Location for the entire URL Request Line	THE_REQUEST	REQUEST_LINE	Functions the same. The variable includes the Request Method, URI and HTTP version data.	Translate any existing THE_REQUEST directives to REQUEST_LINE directives.

Variable/ Location for Arguments	ARG_name	ARGS:name ARGS:/RegEx/	Similar to the HTTP_headername situation, the advantage of the new syntax is the ability to use RegEx in the argument name.	Translate any existing ARG_name directives to ARGS:name directives.
Accessing Request Bodies	SecFilterScanPOST POST_PAYLOAD	SecRequestBodyAccess Phase:2 REQUEST_BODY	In 2.x, the directive is now called SecRequestBodyAccess and it is more flexible than SecFilterScanPOST as it is able to inspect all request bodies (such as PUT and XML, etc...) and not just POST payloads.	Replace the existing SecFilterScanPOST directive with SecRequestBodyAccess. For individual rules where you want to inspect the request bodies, you must specify REQUEST_BODY as the variable and you also must ensure that it is running in phase:2 (by either an inherited SecDefaultAction setting or by explicitly specifying the phase within the rule action).
Ability to disable POST/ Request buffering dynamically	MODSEC_NOPOSTBUFFERING	ctl:requestBodyAccess=Off	In 2.x, you can use the ctl action to turn on/off request body access on a per rule basis.	Take any existing entries in the httpd.conf file that set the MODSEC_NOPOSTBUFFERING Env variable and translate them to Mod 2.x rules.
Accessing Cookies	COOKIES COOKIES_COUNT COOKIES_NAMES COOKIES_VALUES COOKIE_name	REQUEST_HEADERS:Cookie REQUEST_COOKIES_NAMES REQUEST_COOKIES_NAMES: name REQUEST_COOKIES_NAMES:/ RegEx/ REQUEST_COOKIES REQUEST_COOKIES:name REQUEST_COOKIES:/RegEx/	In 2.x, you can use the "&" character to "count" the number of variables. While there are different ways to access request cookies, the main difference between them are that REQUEST_HEADERS:Cookie will include all of the "raw" Cookie data while any of the REQUEST_COOKIES variable values are parsed.	Translate rules as follows – Mod 1.x -> Mod 2.x COOKIES -> REQUEST_COOKIES COOKIES_COUNT -> &REQUEST_COOKIES COOKIES_NAMES -> REQUEST_COOKIES_NAMES COOKIES_VALUES -> REQUEST_COOKIES COOKIE_name -> REQUEST_COOKIES:name
Counting Variables	ARGS_COUNT COOKIES_COUNT HEADERS_COUNT FILES_COUNT	&ARGS &REQUEST_COOKIES &REQUEST_HEADERS &FILES	In 2.x, prepending the "&" character will count the number of variables. Example – 1.x – HEADERS_COUNT 2.x - &REQUEST_HEADERS	Translate existing 1.x rules as listed.

Accessing HTTP Status Code	OUTPUT_STATUS	RESPONSE_STATUS Phase:3	In 2.x, you need to specify both the RESPONSE_STATUS variable and phase:3 with the rule.	Translate any existing 1.x OUTPUT STATUS rules to use RESPONSE_STATUS and phase:3.
Accessing Response Bodies/Post Payloads	SecFilterScanOutput SecFilterOutputMimeTypes OUTPUT	SecResponseBodyAccess SecResponseBodyMimeTypes RESPONSE_BODY Phase:4	In 1.x, neither skipnext nor chain could be used on the OUTPUT location. In 2.x, both actions can be used on RESPONSE_BODY	Translate directives/rules as follows – Mod 1.x -> Mod 2.x SecFilterScanOutput -> SecResponseBodyAccess SecFilterOutputMimeTypes -> SecResponseBodyMimeTypes OUTPUT -> RESPONSE_BODY/ Phase:4
Cookie Normalization	SecFilterCookieFormat SecFilterNormalizeCookies	SecCookieFormat	SecFilterNormalizeCookies is no longer supported as Mod 2.x transformation functions can now be used to normalize all Variables including Cookie data.	Change SecFilterCookieFormat to SecCookieFormat. When specifying Cookie variables, then apply the applicable transformation functions in the action field of the rule.
Ability to skip rules	skipnext	skip	In Mod 2.x – skip takes into account chained rulesets and treats them as 1 rule. In Mod 1.x – skipnext treated each rule directive as an individual rule regardless of whether or not they were tied together as a chained ruleset.	Translate all skipnext rules to skip, however make sure to factor in any chained rulesets that may follow and adjust the skip number accordingly.
Adding/ Removing Audit Log Data on a per rule basis	logparts	ctl:auditLogParts=	The rules function the same.	Translate any existing logparts actions to the ctl:auditLogParts equivalent.

Inspecting uploaded files	SecUploadApproveScript	@inspectFile FILES_TMPNAMES	<p>The main difference here is that now @inspectFile is an Operator vs. a global Directive. This means that you can apply @inspectFile to individual rules and use different scripts as appropriate.</p> <p>Also, the return codes are now reversed –</p> <p>In 1.x, a return code of “1” means that the file would be allowed.</p> <p>In 2.x, a return code of “1” means that the file would be denied.</p>	<p>In order to scan/inspect uploaded files in 2.x, you need to create specific rules that use the FILES_TMPNAMES variable (as these are the names of the files that are temporarily stored on disk) and then use the @inspectFile Operator on each rule.</p> <p>Also, make sure to swap your return codes in existing scripts as mentioned in the notes column.</p>
Memory limits for uploaded files	SecUploadInMemoryLimit	SecRequestBodyInMemoryLimit	These two directives function the same.	Change the SecUploadInMemoryLimit directive to SecRequestBodyInMemoryLimit.