MODSECURITY HANDBOOK

The Complete Guide to the Popular Open Source Web Application Firewall

Ivan Ristić

Feisty Duck

Last update: Mon Jun 03 17:36:08 BST 2013 (build 595)

ModSecurity Handbook

Ivan Ristić

Feisty Duck
LONDON

ModSecurity Handbook

by Ivan Ristić

Copyright © 2010-2013 Feisty Duck Limited. All rights reserved.

ISBN: 978-1-907117-02-2

Development version (revision 595).

First published in March 2010. Fully revised in April 2012.

Feisty Duck Limited
www.feistyduck.com
contact@feistyduck.com

Address:
6 Acantha Court
Montpelier Road
London W5 2QP
United Kingdom

Production editor: Jelena Girić-Ristić

Copyeditor: Nancy Kotary

Cover design: Peter Jovanović

Cover illustration: Maja Veselinović

Technical reviewer: Brian Rectanus

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, without the prior permission in writing of the publisher.

The author and publisher have taken care in preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

ModSecurity is a registered trademark of Trustwave Holdings, Inc. All other trademarks and copyrights are the property of their respective owners.

Table of Contents

Preface xix
  Scope and Audience xix
  Contents xx
  Updates xxiii
  Feedback xxiii
  About the Author xxiii
  About the Technical Reviewer xxiv
  Acknowledgments xxiv
I. User Guide 1
  1. Introduction 3
    Brief History of ModSecurity 3
    What Can ModSecurity Do? 4
      Guiding Principles 6
      Deployment Options 7
      Is Anything Missing? 8
    Getting Started 9
      Hybrid Nature of ModSecurity 9
      Main Areas of Functionality 10
      What Rules Look Like 11
      Transaction Lifecycle 11
      Impact on Web Server 16
      What Next? 17
    Resources 18
      General Resources 19
      Developer Resources 20
      AuditConsole 20
    Summary 21
  2. Installation 23
    Installation from Source 24

Downloading Releases 24
Downloading from Repository 25
Installation on Unix 27
Installation from Binaries 30
Fedora Core, CentOS, and Red Hat Enterprise Linux 30
Debian and Ubuntu 31
Installation on Windows 31
Summary 32
3. Configuration **33**
Folder Locations 34
Configuration Layout 36
Adding ModSecurity to Apache 37
Powering Up 37
Request Body Handling 38
Response Body Handling 40
Filesystem Locations 42
File Uploads 42
Debug Log 43
Audit Log 44
Miscellaneous Options 44
Default Rule Match Policy 45
Handling Processing Errors 45
Verifying Installation 47
Summary 48
4. Logging **51**
Debug Log 51
Debugging in Production 52
Audit Log 54
Audit Log Entry Example 55
Concurrent Audit Log 57
Remote Logging 58
Configuring Remote Logging 59
Activating Remote Logging 61
Troubleshooting Remote Logging 63
File Upload Interception 64
Storing Files 64
Inspecting Files 65
Integrating with ClamAV 66
Advanced Logging Configuration 68

        - Increasing Logging from a Rule 69
        - Dynamically Altering Logging Configuration 69
        - Removing Sensitive Data from Audit Logs 69
        - Selective Audit Logging 71
    - Summary 71

5. Rule Language Overview 73

    - Anatomy of a Rule 73
    - Variables 74
        - Request Variables 75
        - Server Variables 76
        - Response Variables 77
        - Miscellaneous Variables 77
        - Parsing Flags 78
        - Collections 79
        - Time Variables 79
    - Operators 80
        - String Matching Operators 80
        - Numerical Operators 80
        - Validation Operators 81
        - Miscellaneous Operators 81
    - Actions 82
        - Disruptive Actions 82
        - Flow Actions 82
        - Metadata Actions 83
        - Variable Actions 83
        - Logging Actions 84
        - Special Actions 84
        - Miscellaneous Actions 84
    - Summary 85

6. Rule Language Tutorial 87

    - Introducing Rules 87
    - Working with Variables 88
    - Combining Rules into Chains 89
    - Operator Negation 89
    - Variable Counting 89
    - Using Actions 90
        - Understanding Action Defaults 90
        - Actions in Chained Rules 92
        - Unconditional Rules 93

  Using Transformation Functions 93
  Blocking 95
  Changing Rule Flow 95
    Smarter Skipping 97
    If-Then-Else 97
  Controlling Logging 98
  Capturing Data 98
  Variable Manipulation 100
  Variable Expansion 100
  Recording Data in Alerts 101
  Adding Metadata 103
  Embedded vs. Reverse Proxy Mode 104
  Summary 106
7. Rule Configuration 107
  Apache Configuration Syntax 107
    Breaking Lines 108
    Directives and Parameters 108
    Spreading Configuration Across Files 109
    Container Directives 110
    Configuration Contexts 111
    Configuration Merging 112
  Configuration and Rule Inheritance 113
    Configuration Inheritance 113
    Rule Inheritance 114
    Location-Specific Configuration Restrictions 115
    SecDefaultAction Inheritance Anomaly 115
  Rule Manipulation 116
    Removing Rules at Configure Time 116
    Updating Rule Actions at Configure Time 117
    Updating Rule Targets at Configure Time 118
    Removing Rules at Runtime 118
    Updating Rule Targets at Runtime 118
  Configuration Tips 119
  Summary 119
8. Persistent Storage 121
  Manipulating Collection Records 122
    Creating Records 122
    Application Namespaces 123
    Initializing Records 124

    - Controlling Record Longevity 124
    - Deleting Records 125
    - Detecting Very Old Records 126
- Collection Variables 127
    - Built-in Variables 127
    - Variable Expiry 128
    - Variable Value Depreciation 128
- Implementation Details 129
    - Retrieving Records 130
    - Storing a Collection 130
    - Record Limits 132
- Applied Persistence 133
    - Periodic Alerting 133
    - Denial of Service Attack Detection 136
    - Brute Force Attack Detection 138
- Session Management 140
    - Initializing Sessions 140
    - Blocking Sessions 142
    - Forcing Session Regeneration 143
    - Restricting Session Lifetime 143
    - Detecting Session Hijacking 146
- User Management 148
    - Detecting User Sign-In 148
    - Detecting User Sign-Out 149
- Summary 149

9. Practical Rule Writing 151
- Whitelisting 151
    - Whitelisting Theory 151
    - Whitelisting Mechanics 152
    - Granular Whitelisting 153
    - Complete Whitelisting Example 154
- Virtual Patching 155
    - Vulnerability versus Exploit Patching 156
    - Failings of Exploit Detection 157
    - Impedance Mismatch 157
    - Preferred Virtual Patching Approach 159
- IP Address Reputation and Blacklisting 159
    - IP Address Blocking 160
    - Geolocation 161

        Real-Time Block Lists 162
        Local Reputation Management 163
    Integration with Other Apache Modules 163
        Conditional Logging 165
        Header Manipulation 165
        Securing Session Cookies 166
    Advanced Blocking 167
        Immediate Blocking 167
        Keeping Detection and Blocking Separate 168
        User-Friendly Blocking 169
        External Blocking 170
        Honeypot Diversion 171
        Delayed Blocking 171
        Score-Based Blocking 172
    Making the Most of Regular Expressions 173
        How ModSecurity Compiles Patterns 174
        Changing How Patterns Are Compiled 175
        Common Pattern Problems 176
        Regular Expression Denial of Service 176
        Resources 177
    Working with Rule Sets 177
        Deploying Rule Sets 178
        Writing Rules for Distribution 179
        Resources for Rule Writers 181
    Summary 182
10. Performance 183
    Understanding Performance 183
        Top 10 Performance Rules 184
    Performance Tracking 186
        Performance Metrics 186
        Performance Logging 187
        Real-Time Performance Monitoring 187
    Load Testing 187
    Rule Benchmarking 191
        Preparation 191
        Test Data Selection 192
        Performance Baseline 194
    Optimizing Pattern Matching 196
        Rule per Keyword Approach 196

- Combined Regular Expression Pattern 197
- Optimized Regular Expression Pattern 197
- Parallel Pattern Matching 198
- Test Results 199

Summary 199

11. Content Injection **201**

Writing Content Injection Rules 201

- Communicating Back to the Server 203
- Interrupting Page Rendering 204
- Using External JavaScript Code 204

Communicating with Users 205

Summary 206

12. Writing Rules in Lua **207**

Rule Language Integration 207

Lua Rules Skeleton 208

Accessing Variables 208

Setting Variables 209

Logging 210

Lua Actions 210

Summary 211

13. Handling XML **213**

XML Parsing 213

DTD Validation 217

XML Schema Validation 218

XML Namespaces 220

XPath Expressions 222

XPath and Namespaces 224

XML Inspection Framework 224

Summary 226

14. Extending Rule Language **227**

Extension Template 228

Adding a Transformation Function 230

Adding an Operator 233

Adding a Variable 237

Adding a Request Body Processor 240

Summary 243

II. Reference Manual 245

15. Directives **247**

SecAction 247

SecArgumentSeparator 247
SecAuditEngine 248
SecAuditLog 248
SecAuditLog2 249
SecAuditLogDirMode 249
SecAuditLogFileMode 250
SecAuditLogParts 250
SecAuditLogRelevantStatus 252
SecAuditLogStorageDir 252
SecAuditLogType 252
SecCacheTransformations 253
SecChrootDir 254
SecCollectionTimeout 254
SecComponentSignature 255
SecContentInjection 255
SecCookieFormat 255
SecDataDir 256
SecDebugLog 256
SecDebugLogLevel 256
SecDefaultAction 257
SecDisableBackendCompression 257
SecGeoLookupDb 258
SecGsbLookupDb 258
SecGuardianLog 258
SecInterceptOnError 259
SecMarker 259
SecPcreMatchLimit 260
SecPcreMatchLimitRecursion 260
SecPdfProtect 261
SecPdfProtectMethod 261
SecPdfProtectSecret 261
SecPdfProtectTimeout 262
SecPdfProtectTokenName 262
SecReadStateLimit 262
SecRequestBodyAccess 263
SecRequestBodyLimit 263
SecRequestBodyLimitAction 264
SecRequestBodyNoFilesLimit 264
SecRequestBodyInMemoryLimit 265

SecResponseBodyLimit 265
SecResponseBodyLimitAction 265
SecResponseBodyMimeType 266
SecResponseBodyMimeTypesClear 266
SecResponseBodyAccess 267
SecRule 267
SecRuleInheritance 267
SecRuleEngine 268
SecRulePerfTime 269
SecRuleRemoveById 269
SecRuleRemoveByMsg 269
SecRuleRemoveByTag 270
SecRuleScript 270
SecRuleUpdateActionById 271
SecRuleUpdateTargetById 272
SecSensorId 273
SecServerSignature 274
SecStreamInBodyInspection 274
SecStreamOutBodyInspection 275
SecTmpDir 275
SecUploadDir 275
SecUploadFileLimit 276
SecUploadFileMode 276
SecUploadKeepFiles 277
SecWebAppId 277
SecUnicodeCodePage 278
SecUnicodeMapFile 278
SecWriteStateLimit 278
16. Variables 281
ARGS 281
ARGS_COMBINED_SIZE 281
ARGS_GET 281
ARGS_GET_NAMES 281
ARGS_NAMES 281
ARGS_POST 282
ARGS_POST_NAMES 282
AUTH_TYPE 282
DURATION 282
ENV 282

FILES 283
FILES_COMBINED_SIZE 283
FILES_NAMES 283
FILES_SIZES 283
FILES_TMPNAMES 283
GEO 283
HIGHEST_SEVERITY 284
INBOUND_DATA_ERROR 284
MATCHED_VAR 284
MATCHED_VAR_NAME 285
MATCHED_VARS 285
MATCHED_VARS_NAMES 285
MODSEC_BUILD 285
MULTIPART_CRLF_LF_LINES 286
MULTIPART_INVALID_PART 286
MULTIPART_STRICT_ERROR 286
MULTIPART_UNMATCHED_BOUNDARY 287
OUTBOUND_DATA_ERROR 288
PATH_INFO 288
PERF_ALL 288
PERF_COMBINED 288
PERF_GC 288
PERF_LOGGING 288
PERF_PHASE1 289
PERF_PHASE2 289
PERF_PHASE3 289
PERF_PHASE4 289
PERF_PHASE5 289
PERF_RULES 289
PERF_SREAD 289
PERF_SWRITE 289
QUERY_STRING 290
REMOTE_ADDR 290
REMOTE_HOST 290
REMOTE_PORT 290
REMOTE_USER 291
REQBODY_ERROR 291
REQBODY_ERROR_MSG 291
REQBODY_PROCESSOR 291

REQBODY_PROCESSOR_ERROR 291
REQBODY_PROCESSOR_ERROR_MSG 292
REQUEST_BASENAME 292
REQUEST_BODY 292
REQUEST_BODY_LENGTH 292
REQUEST_COOKIES 292
REQUEST_COOKIES_NAMES 293
REQUEST_FILENAME 293
REQUEST_HEADERS 293
REQUEST_HEADERS_NAMES 293
REQUEST_LINE 293
REQUEST_METHOD 294
REQUEST_PROTOCOL 294
REQUEST_URI 294
REQUEST_URI_RAW 294
RESPONSE_BODY 294
RESPONSE_CONTENT_LENGTH 295
RESPONSE_CONTENT_TYPE 295
RESPONSE_HEADERS 295
RESPONSE_HEADERS_NAMES 295
RESPONSE_PROTOCOL 295
RESPONSE_STATUS 296
RULE 296
SCRIPT_BASENAME 296
SCRIPT_FILENAME 296
SCRIPT_GID 296
SCRIPT_GROUPNAME 296
SCRIPT_MODE 297
SCRIPT_UID 297
SCRIPT_USERNAME 297
SERVER_ADDR 297
SERVER_NAME 297
SERVER_PORT 297
SESSION 298
SESSIONID 298
STREAM_INPUT_BODY 298
STREAM_OUTPUT_BODY 298
TIME 299
TIME_DAY 299

TIME_EPOCH 299
TIME_HOUR 299
TIME_MIN 299
TIME_MON 299
TIME_SEC 300
TIME_WDAY 300
TIME_YEAR 300
TX 300
UNIQUE_ID 301
URLENCODED_ERROR 301
USERAGENT_IP 301
USERID 301
WEBAPPID 301
WEBSERVER_ERROR_LOG 301
XML 302
17. Transformation Functions 303
base64Decode 304
base64DecodeExt 304
base64Encode 304
cmdLine 304
compressWhitespace 304
cssDecode 305
decodeBase64Ext 305
escapeSeqDecode 305
hexDecode 305
hexEncode 305
htmlEntityDecode 305
jsDecode 306
length 306
lowercase 306
md5 306
none 306
normalisePath 306
normalisePathWin 307
normalizePath 307
normalizePathWin 307
parityEven7bit 307
parityOdd7bit 307
parityZero7bit 307

removeComments 307
removeCommentsChar 307
removeNulls 307
removeWhitespace 308
replaceComments 308
replaceNulls 308
urlDecode 308
urlDecodeUni 308
urlEncode 308
utf8toUnicode 308
sha1 309
sqlHexDecode 309
trimLeft 309
trimRight 309
trim 309
18. Actions 311
accuracy 311
allow 311
append 312
auditlog 312
block 312
capture 313
chain 314
ctl 314
deny 315
deprecatevar 316
drop 316
exec 316
expirevar 317
id 317
initcol 318
log 318
logdata 318
maturity 318
msg 319
multiMatch 319
noauditlog 319
nolog 319
pass 320

pause 320
phase 320
prepend 321
proxy 321
redirect 322
rev 322
sanitiseArg 322
sanitiseMatched 322
sanitiseMatchedBytes 323
sanitiseRequestHeader 323
sanitiseResponseHeader 323
sanitizeArg 323
sanitizeMatched 323
sanitizeMatchedBytes 323
sanitizeRequestHeader 324
sanitizeResponseHeader 324
severity 324
setuid 324
setsid 325
setenv 325
setvar 325
skip 326
skipAfter 326
status 326
t 327
tag 327
ver 327
xmlns 327
19. Operators 329
beginsWith 329
contains 329
endsWith 329
eq 329
ge 330
geoLookup 330
gsbLookup 330
gt 331
inspectFile 332
ipMatch 332

ipMatchF 333
ipMatchFromFile 333
le 333
lt 333
pm 333
pmf 334
pmFromFile 334
rbl 335
rsub 335
rx 336
streq 336
validateByteRange 337
validateDTD 337
validateSchema 337
validateUrlEncoding 338
validateUtf8Encoding 338
verifyCC 339
verifyCPF 339
verifySSN 339
within 340

20. Data Formats 341

Alerts 341
- Alert Action Description 341
- Alert Justification Description 342
- Metadata 343
- Escaping 344
- Alerts in the Apache Error Log 344
- Alerts in Audit Logs 345

Audit Log 345
- Parts 346
- Storage Formats 354
- Remote Logging Protocol 356

Index 359

Preface

I didn't mean to write this book, I really didn't. Several months ago I started to work on the second edition of *Apache Security*, deciding to rewrite the ModSecurity chapter first. A funny thing happened: the ModSecurity chapter kept growing and growing. It hit 40 pages. It hit 80 pages. And then I realized that I was nowhere near the end. That was all the excuse I needed to put *Apache Security* aside—for the time being—and focus on a ModSecurity book instead.

I admit that I couldn't be happier, although it was an entirely emotional decision. After spending years working on ModSecurity, I knew it had so much more to offer, yet the documentation wasn't there to show the way. But it is now, I am thrilled to say. The package is complete: you have an open source tool that is able to compete with the best commercial products out there, *and* you have the documentation to match.

With this book I am also trying something completely new—*continuous writing and publishing*. You see, I had published my first book with a major publisher, but I never quite liked the process. It was too slow. You write a book pretty much in isolation, you publish it, and then you never get to update it. I was never happy with that, and that's why I decided to do things differently this time.

Simply said, *ModSecurity Handbook* is a living book. Every time I make a change, a new digital version is made available to you. If I improve the book based on your feedback, you get the improvements as soon as I make them. If you prefer a paper book, you can still get it of course, through the usual channels. Although I can't do anything about updating the paper version of the book, we can narrow the gap slightly by pushing out book updates even between editions. That means that, even when you get the paper version (as most people seem to prefer to), it is never going to be too much behind the digital version.

Scope and Audience

This book exists to document every single aspect of ModSecurity and to teach you how to use it. It is as simple as that. ModSecurity is a fantastic tool, but it is let down by the poor quality of the documentation. As a result, the adoption is not as good as it could be; application security is difficult on its own and you don't really want to struggle with poorly documented tools too. I felt a responsibility to write this book and show how ModSecurity can compete with commercial web application firewalls, in spite of being the underdog. Now that the book is finished, I feel I've done a proper job with ModSecurity.

If you are interested in application security, you are my target audience. Even if you're not interested in application security as such, and only want to deal with your particular problems (it's difficult to find a web application these days that's without security problems), you are still my target audience.

You don't need to know anything about ModSecurity to get started. If you just follow the book from the beginning, you will find that every new chapter advances a notch. Even if you are a long-time ModSecurity user, I believe you will benefit from a fresh start. I will let you in on a secret—I have. There's nothing better for completing one's knowledge than having to write about a particular topic. I suspect that long-time ModSecurity users will especially like the second half of the book, which discusses many advanced topics and often covers substantial new ground.

But, there is only so much a book can cover. *ModSecurity Handbook* assumes you already know how to operate the Apache web server. You don't have to be an expert, but you do need to know how to install, configure, and run it. If you don't know how to do that already, you should get my first book, *Apache Security*. I wrote it five years ago, but it's still remarkably fresh. (Ironically, it is only the ModSecurity chapter in *Apache Security* that is completely obsolete. But that's why you have this book.)

On the other end, *ModSecurity Handbook* will teach you how to use ModSecurity and write good rules, but it won't teach you application security. In my earlier book, *Apache Security*, I included a chapter that served as an introduction to application security, but, even then, I was barely able to mention all that I wanted, and the chapter was still the longest chapter in the book. Since then, the application security field has exploded and now you have to read several books and dozens of research papers just to begin to understand it.

Contents

Once you go past the first chapter, which is the introduction to the world of ModSecurity, the rest of the book consists of roughly three parts. In the first part, you learn how to install and configure ModSecurity. In the second part, you learn how to write rules. As for the third part, you could say that it contains the advanced stuff—a series of chapters each dedicated to one important aspect of ModSecurity.

At the end of the book is the official reference documentation, reproduced with the permission from Breach Security.

Chapter 1, *Introduction*, is the foundation of the book. It contains a gentle introduction to ModSecurity, and then explains what it can and cannot do. The main usage scenarios are listed to help you identify where you can use ModSecurity in your environment. The middle of the chapter goes under the hood of ModSecurity to give you an insight into how it works, and finishes with an overview of the key areas you will need to learn in order to deploy it. The end of the chapter lists a series of resources (sites, mailing lists, tools, etc.) that you will find useful in your day-to-day work.

Chapter 2, *Installation*, teaches you how to install ModSecurity, either compiling from source (using one of the released versions or downloading straight from the development repository), or by using one of the available binary packages, on Unix and Windows alike.

Chapter 3, *Configuration*, explains how each of the available configuration directives should be used. By the end of the chapter, you get a complete overview of the configuration options and will have a solid default configuration for all your ModSecurity installations.

Chapter 4, *Logging*, deals with the logging features of ModSecurity. The two main logging facilities explained are the debug log, which is useful in rule writing, and the audit log, which is used to log complete transaction data. Special attention is given to remote logging, which you'll need to manage multiple sensors, or to use any of the user-friendly tools for alert management. File interception and validation is covered in detail. The chapter ends with an advanced section of logging, which explains how to selectively log traffic, and how to use the sanitation feature to prevent sensitive data from being stored in the logs.

Chapter 5, *Rule Language Overview*, is the first of the three chapters that deal with rule writing. This chapter contains an overview of the entire rule language, which will get you started as well as give you a feature map to which you can return whenever you need to deal with a new problem.

Chapter 6, *Rule Language Tutorial*, teaches how to write rules, and how to write them well. It's a very fun chapter that adopts a gradual approach, introducing the features one by one. By the end of the chapter, you will know everything about writing individual rules.

Chapter 7, *Rule Configuration*, completes the topic of rule writing. It takes a step back to view the rules as the basic block for policy building. You first learn how to put a few rules together and add them to the configuration, as well as how the rules interact with Apache's ability to use different configuration contexts for different sites and different locations within sites. The chapter spends a great deal of time making sure you take advantage of the inheritance feature, which helps make ModSecurity configuration much easier to maintain.

Chapter 8, *Persistent Storage*, is quite possibly the most exciting chapter in the book. It describes the persistent storage mechanism, which enables you to track data and events over time and thus opens up an entire new dimension of ModSecurity. This chapter is also the most practical one in the entire book. It gives you the rules for periodic alerting, brute force attack detection, denial of service attack detection, session and user management, fixing session management weaknesses, and more.

Chapter 9, *Practical Rule Writing*, is, as the name suggests, a tour through many of the practical activities you will perform in your day-to-day work. The chapter starts by covering whitelisting, virtual patching, IP address reputation and blacklisting. You then learn how to integrate with other Apache modules, with practical examples that show how to perform conditional logging and fix insecure session cookies. Special attention is given to the topic of blocking; several approaches, starting from the simple to the very sophisticated, are presented. A section on regular expressions gets you up to speed with the most important ModSecurity operator. The chapter ends with a discussion of rule sets, discussing how to use the rule sets others have written, as well as how to write your own.

Chapter 10, *Performance*, covers several performance-related topics. It opens with an overview of where ModSecurity usually spends its time, a list of common configuration mistakes that should be avoided, and a list of approaches that result in better performance. The second part of the chapter describes how to monitor ModSecurity performance in production. The third part tests the publicly available rule sets in order to give you a taste of what they are like, as well as document a methodology you can use to test your own rules. The chapter then moves to rule set benchmarking, which is an essential part of the process of rule writing. The last part of this chapter gives very practical advice on how to use regular expressions and parallel matching, comparing several approaches and explaining when to use them.

Chapter 11, *Content Injection*, explains how to reach from ModSecurity, which is a server-side tool, right into a user's browser and continue with the inspection there. This feature makes it possible to detect the attacks that were previously thought to be undetectable by a server-side tool, for example DOM-based cross-site scripting attacks. Content injection also comes in handy if you need to communicate with your users—for example, to tell them that they have been attacked.

Chapter 12, *Writing Rules in Lua*, discusses a gem of a feature: writing rules using the Lua programming language. The rule language of ModSecurity is easy to use and can get a lot done, but for the really difficult problems you may need the power of a proper programming language. In addition, you can use Lua to react to events, and it is especially useful when integrating with external systems.

Chapter 13, *Handling XML*, covers the XML capabilities of ModSecurity in detail. You get to learn how to validate XML using either DTDs or XML Schemas, and how to combine XPath expressions with the other features ModSecurity offers to perform both whitelist- and blacklist-based validation. The XML features of ModSecurity have traditionally been poorly documented; here you will find details never covered before. The chapter ends with an XML validation framework you can easily adapt for your needs.

Chapter 14, *Extending Rule Language*, discusses how you can extend ModSecurity to implement new functionality. It gives several step-by-step examples, explaining how to implement a transformation function, an operator, and a variable. Of course, with ModSecurity being open source, you can extend it directly at any point, but when you use the official APIs, you avoid making a custom version of ModSecurity (which is generally time consuming because it prevents upgrades).

Updates

If you purchased this book directly from Feisty Duck, your purchase includes access to newer digital versions of the book. Updates are made automatically after I update the manuscript, which I keep in DocBook format in a Subversion repository. At the moment, there is a script that runs every hour, and rebuilds the book when necessary. Whenever you visit your personal digital download link, you get the most recent version of the book.

I use a dedicated Twitter account (`@modsecuritybook`) to announce relevant changes I make to the book. By following that account you'll find out about the improvements pretty much as they happen. You can also follow my personal Twitter account (`@ivanristic`) or subscribe to my blog, if you are about computer security in general.

In the first two years of its life, I kept *ModSecurity Handbook* up-to-date with every ModSecurity release. There was a full revision in February 2012, which made the book essentially as good and as current as it was on day of the first release back in 2010. Don't take my past performance as a guarantee of what is going to happen in the future, however. At the launch in 2010 I offered a guarantee that the book will be kept up-to-date for at least a year from your purchase. I dropped that promise at the end of 2011, because I could see the possibility that I would stop with the updates at some point. I will keep my promise until the end of 2012, but I don't know what will happen after that.

Feedback

To get in touch with me please write to *ivanr@webkreator.com*. I would like to hear from you very much, because I believe that a book can fulfill its potential only through the interaction among its author(s) and the readers. Your feedback is particularly important when a book is continuously updated, like this one is. When I change the book as a result of your feedback, all the changes are immediately delivered back to you. There is no more waiting for years to see the improvements!

About the Author

Ivan Ristić is a respected security expert and author, known especially for his contribution to the web application firewall field and the development of ModSecurity, the open source web application firewall. He is also the author of *Apache Security*, a comprehensive security guide for the Apache web server. A frequent speaker at computer security conferences, Ivan is an active participant in the application security community, a member of the Open Web Application Security Project (OWASP), and an officer of the Web Application Security Consortium (WASC).

About the Technical Reviewer

Brian Rectanus is a developer turned manager in the web application security field. He has worked in the past on various security software related projects such as the IronBee open source WAF framework, the ModSecurity open source WAF and the Suricata open source IDS/IPS. Brian is an open source advocate and proud `NIX loving, Mac using, non-Windows user who has been writing code on various `NIX platforms with `vi` since 1993. Today he still does all his development work in the more modern `vim` editor—like there is any other—and loves every bit of it. Brian has spent the majority of his career working with web technology from various perspectives, be it manager, developer, administrator or security assessor. Brian has held many certifications in the past, including GCIA and GCIH certification from the SANS Institute and a BS in computer science from Kansas State University.

Acknowledgments

To begin with, I would like to thank the entire ModSecurity community for their support, and especially all of you who used ModSecurity and sent me your feedback. ModSecurity wouldn't be what it is without you. Developing and supporting ModSecurity was a remarkable experience; I hope you enjoy using it as much as I enjoyed developing it.

I would also like to thank my former colleagues from Breach Security, who gave me a warm welcome, even though I joined them pretty late in the game. I regret that, due to my geographic location, I didn't spend more time working with you. I would especially like to thank —in no particular order—Brian Rectanus, Ryan Barnett, Ofer Shezaf, and Avi Aminov, who worked with me on the ModSecurity team. Brian was also kind to work with me on the book as a technical reviewer, and I owe special thanks to him for ensuring I didn't make too many mistakes.

I mustn't forget my copyeditor, Nancy Kotary, who was a pleasure to work with, despite having to deal with DocBook and Subversion, none of which is in the standard copyediting repertoire.

For some reason unknown to me, my dear wife Jelena continues to tolerate my long working hours. Probably because I keep promising to work less, even though that never seems to happen. To her I can only offer my undying love and gratitude for accepting me for who I am. My daughter Iva, who's four, is too young to understand what she means to me, but that's all right —I have the patience to wait for another 20 years or so. She is the other sunshine in my life.

I User Guide

This part, with its 14 chapters, constitutes the main body of the book. The first chapter is the introduction to ModSecurity and your map to the rest of the book. The remaining chapters fall into roughly four groups: installation and configuration, rule writing, practical work, and advanced topics.

1 Introduction

ModSecurity is a tool that will help you secure your web applications. No, scratch that. Actually, ModSecurity is a tool that will help you sleep better at night, and I will explain how. I usually call ModSecurity a *web application firewall* (WAF), because that's the generally accepted term to refer to the class of products that are specifically designed to secure web applications. Other times I will call it an *HTTP intrusion detection tool*, because I think that name better describes what ModSecurity does. Neither name is entirely adequate, yet we don't have a better one. Besides, it doesn't really matter what we call it. The point is that web applications—yours, mine, everyone's—are terribly insecure on average. We struggle to keep up with the security issues and need any help we can get to secure them.

The idea to write ModSecurity came to me during one of my sleepless nights—I couldn't sleep because I was responsible for the security of several web-based products. I could see how most web applications were slapped together with little time spent on design and little time spent on understanding the security issues. Furthermore, not only were web applications insecure, but we had no idea how insecure they were or if they were being attacked. Our only eyes were the web server access and error logs, and they didn't say much.

ModSecurity will help you sleep better at night because, above all, it solves the visibility problem: it lets you see your web traffic. That visibility is key to security: once you are able to see HTTP traffic, you are able to analyze it in real time, record it as necessary, and react to the events. The best part of this concept is that you get to do all of that without actually touching web applications. Even better, the concept can be applied to any application—even if you can't access the source code.

Brief History of ModSecurity

Like many other open source projects, ModSecurity started out as a hobby. Software development had been my primary concern back in 2002, when I realized that producing secure web applications is virtually impossible. As a result, I started to fantasize about a tool that would sit in front of web applications and control the flow of data in and out. The first version was released in November 2002, but a few more months were needed before the tool became useful. Other people started to learn about it, and the popularity of ModSecurity started to rise.

Initially, most of my effort was spent wrestling with Apache to make request body inspection possible. Apache 1.3.x did not have any interception or filtering APIs, but I was able to trick it into submission. Apache 2.x improved things by providing APIs that do allow content interception, but there was no documentation to speak of. Nick Kew released the excellent *The Apache Modules Book* (Prentice Hall) in 2007, which unfortunately was too late to help me with the development of ModSecurity.

By 2004, I was a changed man. Once primarily a software developer, I became obsessed with web application security and wanted to spend more time working on it. I quit my job and started treating ModSecurity as a business. My big reward came in the summer of 2006, when ModSecurity went head to head with other web application firewalls, in an evaluation conducted by Forrester Research, and came out very favorably. Later that year, my company was acquired by Breach Security. A team of one eventually became a team of many: Brian Rectanus came to work on ModSecurity, Ofer Shezaf took on the rules, and Ryan C. Barnett the community management and education. ModSecurity 2.0, a complete rewrite, was released in late 2006. At the same time we released ModSecurity Community Console, which combined the functionality of a remote logging sensor and a monitoring and reporting GUI.

I stopped being in charge of ModSecurity in January 2009, when I left Breach Security. Brian Rectanus subsequently took the lead. In the meantime, Ryan C. Barnett took charge of the ModSecurity rules and produced a significant improvement with CRS v2. In 2010, Trustwave acquired Breach Security and promised to revitalize ModSecurity. The project is currently run by Ryan C. Barnett and Breno Silva, and there are indeed some signs that the project is getting healthier. I remain involved primarily through my work on this book.

Something spectacular happened in March 2011: Trustwave announced that they would be changing the license of ModSecurity from GPLv2 to Apache Software License (ASLv2). This is a great step toward a wider use of ModSecurity because ASL falls into the category of permissive licenses. Later, the same change was announced for the Core Rule Set project (which is hosted with OWASP).

What Can ModSecurity Do?

ModSecurity is a toolkit for real-time web application monitoring, logging, and access control. I like to think about it as an enabler: there are no hard rules telling you what to do; instead, it is up to you to choose your own path through the available features. That's why the title of this section asks what ModSecurity can do, not what it does.

The freedom to choose what to do is an essential part of ModSecurity's identity and goes very well with its open source nature. With full access to the source code, your freedom to choose extends to the ability to customize and extend the tool itself to make it fit your needs. It's not a matter of ideology, but of practicality. I simply don't want my tools to restrict what I can do.

Back on the topic of what ModSecurity can do, the following is a list of the most important usage scenarios:

Real-time application security monitoring and access control
At its core, ModSecurity gives you access to the HTTP traffic stream, in real-time, along with the ability to inspect it. This is enough for real-time security monitoring. There's an added dimension of what's possible through ModSecurity's persistent storage mechanism, which enables you to track system elements over time and perform event correlation. You are able to reliably block, if you so wish, because ModSecurity uses full request and response buffering.

Virtual patching
Virtual patching is a concept of vulnerability mitigation in a separate layer, where you get to fix problems in applications without having to touch the applications themselves. Virtual patching is applicable to applications that use any communication protocol, but it is particularly useful with HTTP, because the traffic can generally be well understood by an intermediary device. ModSecurity excels at virtual patching because of its reliable blocking capabilities and the flexible rule language that can be adapted to any need. It is, by far, the activity that requires the least investment, is the easiest activity to perform, and the one that most organizations can benefit from straight away.

Full HTTP traffic logging
Web servers traditionally do very little when it comes to logging for security purposes. They log very little by default, and even with a lot of tweaking you are not able to get everything that you need. I have yet to encounter a web server that is able to log full transaction data. ModSecurity gives you that ability to log anything you need, including raw transaction data, which is essential for forensics. In addition, you get to choose which transactions are logged, which parts of a transaction are logged, and which parts are sanitized.

Continuous passive security assessment
Security assessment is largely seen as an active scheduled event, in which an independent team is sourced to try to perform a simulated attack. Continuous passive security assessment is a variation of real-time monitoring, where, instead of focusing on the behavior of the external parties, you focus on the behavior of the system itself. It's an early warning system of sorts that can detect traces of many abnormalities and security weaknesses before they are exploited.

Web application hardening
One of my favorite uses for ModSecurity is attack surface reduction, in which you selectively narrow down the HTTP features you are willing to accept (e.g., request methods, request headers, content types, etc.). ModSecurity can assist you in enforcing many

similar restrictions, either directly, or through collaboration with other Apache modules. They all fall under web application hardening. For example, it is possible to fix many session management issues, as well as cross-site request forgery vulnerabilities.

Something small, yet very important to you
Real life often throws unusual demands to us, and that is when the flexibility of ModSecurity comes in handy where you need it the most. It may be a security need, but it may also be something completely different. For example, some people use ModSecurity as an XML web service router, combining its ability to parse XML and apply XPath expressions with its ability to proxy requests. Who knew?

> **Note**
>
> I often get asked if ModSecurity can be used to protect Apache itself. The answer is that it can, in some limited circumstances, but that it isn't what it is designed for. You may sometimes be able to catch an attack with ModSecurity before it hits a vulnerable spot in Apache or in a third-party module, but there's a large quantity of code that runs before ModSecurity. If there's a vulnerability in that area, ModSecurity won't be able to do anything about it.

What Are Web Application Firewalls, Anyway?

I said that ModSecurity is a web application firewall, but it's a little known fact that no one really knows what web application firewalls are. It is generally understood that a web application firewall is an intermediary element (implemented either as a software add-on or process, or as a network device) that enhances the security of web applications, but opinions differ once you dig deeper. There are many theories that try to explain the different views, but the best one I could come up with is that, unlike anything we had before, the web application space is so complex that there is no easy way to classify what we do security-wise. Rather than focus on the name, you should focus on what a particular tool does and how it can help.

If you want to learn more about the topic, there are two efforts that focus on understanding web application firewalls:

- *Web application firewall evaluation criteria* (WAFEC) is a project of the Web Application Security Consortium (WASC). It's an older effort (which has been inactive for a couple of years now) that focuses on the technical features of web application firewalls.
- *Best practices: Web Application Firewalls* is a project of Open Web Application Security Project (OWASP) that focuses largely on the practicalities of WAF deployment, which is an important aspect that is often overlooked.

Guiding Principles

There are four guiding principles on which ModSecurity is based, as follows:

Flexibility

I think that it's fair to say that I built ModSecurity for myself: a security expert who needs to intercept, analyze, and store HTTP traffic. I didn't see much value in hardcoded functionality, because real life is so complex that everyone needs to do things just slightly differently. ModSecurity achieves flexibility by giving you a powerful rule language, which allows you to do exactly what you need to, in combination with the ability to apply rules only where you need to.

Passiveness

ModSecurity will take great care to never interact with a transaction unless you tell it to. That is simply because I don't trust tools, even the one I built, to make decisions for me. That's why ModSecurity will give you plenty of information, but ultimately leave the decisions to you.

Predictability

There's no such thing as a perfect tool, but a predictable one is the next best thing. Armed with all the facts, you can understand ModSecurity's weak points and work around them.

Quality over quantity

Over the course of six years spent working on ModSecurity, we came up with many ideas for what ModSecurity could do. We didn't act on most of them. We kept them for later. Why? Because we understood that we have limited resources available at our disposal and that our minds (ideas) are far faster than our implementation abilities. We chose to limit the available functionality, but do really well at what we decided to keep in.

There are bits in ModSecurity that fall outside the scope of these four principles. For example, ModSecurity can change the way Apache identifies itself to the outside world, confine the Apache process within a jail, and even implement an elaborate scheme to deal with a once-infamous universal XSS vulnerability in Adobe Reader. Although it was I who added those features, I now think that they detract from the main purpose of ModSecurity, which is a reliable and predictable tool that allows for HTTP traffic inspection.

Deployment Options

ModSecurity supports two deployment options: embedded and reverse proxy deployment. There is no one correct way to use them; choose an option based on what best suits your circumstances. There are advantages and disadvantages to both options:

Embedded

Because ModSecurity is an Apache module, you can add it to any compatible version of Apache. At the moment that means a reasonably recent Apache version from the 2.0.x branch, although a newer 2.2.x version is recommended. The embedded option is a great choice for those who already have their architecture laid out and don't want to change it. Embedded deployment is also the only option if you need to protect hundreds of web servers. In such situations, it is impractical to build a separate proxy-based security layer. Embedded ModSecurity not only does not introduce new points of failure, but it scales seamlessly as the underlying web infrastructure scales. The main challenge with embedded deployment is that server resources are shared between the web server and ModSecurity.

Reverse proxy

Reverse proxies are effectively HTTP routers, designed to stand between web servers and their clients. When you install a dedicated Apache reverse proxy and add ModSecurity to it, you get a "proper" network web application firewall, which you can use to protect any number of web servers on the same network. Many security practitioners prefer having a separate security layer. With it you get complete isolation from the systems you are protecting. On the performance front, a standalone ModSecurity will have resources dedicated to it, which means that you will be able to do more (i.e., have more complex rules). The main disadvantage of this approach is the new point of failure, which will need to be addressed with a high-availability setup of two or more reverse proxies.

Is Anything Missing?

ModSecurity is a very good tool, but there are a number of features, big and small, that could be added. The small features are those that would make your life with ModSecurity easier, perhaps automating some of the boring work (e.g., persistent blocking, which you now have to do manually). But there are really only two features that I would call missing:

Learning

Defending web applications is difficult, because there are so many of them, and they are all different. (I often say that every web application effectively creates its own communication protocol.) It would be very handy to have ModSecurity observe application traffic and create a model that could later be used to generate policy or assist with false positives. While I was at Breach Security, I started a project called ModProfiler as a step toward learning, but that project is still as I left it, as version 0.2.

Passive mode of deployment

ModSecurity can be embedded only in Apache 2.x, but when you deploy it as a reverse proxy, it can be used to protect any web server. Reverse proxies are not everyone's cup of tea, however, and sometimes it would be very handy to deploy ModSecurity passively, without having to change anything on the network.

Although a GUI is not within the scope of the project, there are currently two options when it comes to remote logging and alert management. You will find them in the Resources section later in this chapter.

Getting Started

In this first practical section in the book, I will give you a whirlwind tour of the ModSecurity internals, which should help you get started.

Hybrid Nature of ModSecurity

ModSecurity is a hybrid web application firewall engine that relies on the host web server for some of the work. The only supported web server at the moment is Apache 2.x, but it is possible, in principle, to integrate ModSecurity with any other web server that provides sufficient integration APIs.

Apache does for ModSecurity what it does for all other modules—it handles the infrastructure tasks:

1. Decrypts SSL
2. Breaks up the inbound connection stream into HTTP requests
3. Partially parses HTTP requests
4. Invokes ModSecurity, choosing the correct configuration context (`<VirtualHost>`, `<Location>`, etc.)
5. De-chunks request bodies as necessary

There a few additional tasks Apache performs in a reverse proxy scenario:

1. Forwards requests to backend servers (with or without SSL)
2. Partially parses HTTP responses
3. De-chunks response bodies as necessary

The advantage of a hybrid implementation is that it is very efficient—the duplication of work is minimal when it comes to HTTP parsing. A couple of disadvantages of this approach are that you don't always get access to the raw data stream and that web servers sometimes don't process data in the way a security-conscious tool would. In the case of Apache, the hybrid approach works reasonably well, with a few minor issues:

Request line and headers are NUL-terminated

This is normally not a problem, because what Apache doesn't see cannot harm any module or application. In some very rare cases, however, the purpose of the NUL-byte evasion is to hide things, and this Apache behavior only helps with the hiding.

Request header transformation

Apache will canonicalize request headers, combining multiple headers that use the same name and collapsing those that span two or more lines. The transformation may make it difficult to detect subtle signs of evasion, but in practice this hasn't been a problem yet.

Quick request handling

Apache will handle some requests quickly, leaving ModSecurity unable to do anything but notice them in the logging phase. Invalid HTTP requests, in particular, will be rejected by Apache without ModSecurity having a say.

No access to some response headers

Because of the way Apache works, the `Server` and `Date` response headers are invisible to ModSecurity; they cannot be inspected or logged.

Main Areas of Functionality

The functionality offered by ModSecurity falls roughly into four areas:

Parsing

ModSecurity tries to make sense of as much data as available. The supported data formats are backed by security-conscious parsers that extract bits of data and store them for use in the rules.

Buffering

In a typical installation, both request and response bodies will be buffered. This means that ModSecurity usually sees complete requests before they are passed to the application for processing, and complete responses before they are sent to clients. Buffering is an important feature, because it is the only way to provide reliable blocking. The downside of buffering is that it requires additional RAM to store the request and response body data.

Logging

Full transaction logging (also referred to as *audit logging*) is a big part of what ModSecurity does. This feature allows you to record complete HTTP traffic, instead of just rudimentary access log information. Request headers, request body, response header, response body—all those bits will be available to you. It is only with the ability to see what is happening that you will be able to stay in control.

Rule engine

The rule engine builds on the work performed by all other components. By the time the rule engine starts operating, the various bits and pieces of data it requires will all be prepared and ready for inspection. At that point, the rules will take over to assess the transaction and take actions as necessary.

> ### Note
>
> There's one thing ModSecurity purposefully avoids to do: as a matter of design, ModSecurity does not support data sanitization. I don't believe in sanitization, purely because I believe that it is too difficult to get right. If you know for sure that you are being attacked (as you have to before you can decide to sanitize), then you should refuse

> to process the offending requests altogether. Attempting to sanitize merely opens a new battlefield where your attackers don't have anything to lose, but everything to win. You, on the other hand, don't have anything to win, but everything to lose.

What Rules Look Like

Everything in ModSecurity revolves around two things: configuration and rules. The configuration tells ModSecurity how to process the data it sees; the rules decide what to do with the processed data. Although it is too early to go into how the rules work, I will show you a quick example here just to give you an idea what they look like.

For example:

```
SecRule ARGS "<script>" log,deny,status:404
```

Even without further assistance, you can probably recognize the part in the rule that specifies what we wish to look for in input data (`<script>`). Similarly, you will easily figure out what will happen if we do find the desired pattern (`log,deny,status:404`). Things will become more clear if I tell you about the general rule syntax, which is the following:

```
SecRule VARIABLES OPERATOR ACTIONS
```

The three parts have the following meanings:

1. The `VARIABLES` part tells ModSecurity where to look. The `ARGS` variable, used in the example, means all request parameters.
2. The `OPERATOR` part tells ModSecurity how to look. In the example, we have a regular expression pattern, which will be matched against `ARGS`.
3. The `ACTIONS` part tells ModSecurity what to do on a match. The rule in the example gives three instructions: `log` problem, `deny` transaction and use the status 404 for the denial (`status:404`).

I hope you are not disappointed with the simplicity of this first rule. I promise you that by combining the various facilities offered by ModSecurity, you will be able to write very useful rules that implement complex logic where necessary.

Transaction Lifecycle

In ModSecurity, every transaction goes through five steps, or phases. In each of the phases, ModSecurity will do some work at the beginning (e.g., parse data that has become available), invoke the rules specified to work in that phase, and perhaps do a thing or two after the phase rules have finished. At first glance, it may seem that five phases are too many, but there's a reason why each of the phases exist. There is always one thing, sometimes several, that can only be done at a particular moment in the transaction lifecycle.

Request headers (1)
The request headers phase is the first entry point for ModSecurity. The principal purpose of this phase is to allow rule writers to assess a request before the costly request body processing is undertaken. Similarly, there is often a need to influence how ModSecurity will process a request body, and this phase is the place to do it. For example, ModSecurity will not parse an XML request body by default, but you can instruct it do so by placing the appropriate rules into phase 1. (If you care about XML processing, it is described in detail in Chapter 13, *Handling XML*).

Request body (2)
The request body phase is the main request analysis phase and takes place immediately after a complete request body has been received and processed. The rules in this phase have all the available request data at their disposal.

Response headers (3)
The response headers phase takes place after response headers become available, but before a response body is read. The rules that need to decide whether to inspect a response body should run in this phase.

Response body (4)
The response body phase is the main response analysis phase. By the time this phase begins, the response body will have been read, with all its data available for the rules to make their decisions.

Logging (5)
The logging phase is special in more ways than one. First, it's the only phase from which you cannot block. By the time this phase runs, the transaction will have finished, so there's little you can do but record the fact that it happened. Rules in this phase are run to control how logging is done.

Lifecycle Example

To give you a better idea what happens on every transaction, we'll examine a detailed debug log of one POST transaction. I've deliberately chosen a transaction type that uses the request body as its principal method to transmit data, because following such a transaction will exercise most parts of ModSecurity. To keep things relatively simple, I used a configuration without any rules, removed some of the debug log lines for clarity, and removed the timestamps and some additional metadata from each line.

> ### Note
>
> Please do not try to understand everything about the logs at this point. The idea is just to get a general feel about how ModSecurity works, and to introduce you to debug logs. Very quickly after starting to use ModSecurity, you will discover that the debug logs will be an indispensable rule writing and troubleshooting tool.

The transaction I am using as an example in this section is very straightforward. I made a point of placing request data in two different places, parameter *a* in the query string and parameter *b* in the request body, but there is little else of interest in the request:

```
POST /?a=test HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 6

b=test
```

The response is entirely unremarkable:

```
HTTP/1.1 200 OK
Date: Sun, 17 Jan 2010 00:13:44 GMT
Server: Apache
Content-Length: 12
Connection: close
Content-Type: text/html

Hello World!
```

ModSecurity is first invoked by Apache after request headers become available, but before a request body (if any) is read. First comes the initialization message, which contains the unique transaction ID generated by `mod_unique_id`. Using this information, you should be able to pair the information in the debug log with the information in your access and audit logs. At this point, ModSecurity will parse the information on the request line and in the request headers. In this example, the query string part contains a single parameter (`a`), so you will see a message documenting its discovery. ModSecurity will then create a transaction context and invoke the `REQUEST_HEADERS` phase:

```
[4] Initialising transaction (txid SopXW38EAAE9YbLQ).
[5] Adding request argument (QUERY_STRING): name "a", value "test"
[4] Transaction context created (dcfg 8121800).
[4] Starting phase REQUEST_HEADERS.
```

Assuming that a rule didn't block the transaction, ModSecurity will now return control to Apache, allowing other modules to process the request before control is given back to it.

In the second phase, ModSecurity will first read and process the request body, if it is present. In the following example, you can see three messages from the input filter, which tell you what was read. The fourth message tells you that one parameter was extracted from the request body. The content type used in this request (`application/x-www-form-urlencoded`) is one of the types ModSecurity recognizes and parses automatically. Once the request body is processed, the `REQUEST_BODY` rules are processed.

```
[4] Second phase starting (dcfg 8121800).
```

```
[4] Input filter: Reading request body.
[9] Input filter: Bucket type HEAP contains 6 bytes.
[9] Input filter: Bucket type EOS contains 0 bytes.
[5] Adding request argument (BODY): name "b", value "test"
[4] Input filter: Completed receiving request body (length 6).
[4] Starting phase REQUEST_BODY.
```

The filters that keep being mentioned in the logs are parts of ModSecurity that handle request and response bodies:

```
[4] Hook insert_filter: Adding input forwarding filter (r 81d0588).
[4] Hook insert_filter: Adding output filter (r 81d0588).
```

There will be a message in the debug log every time ModSecurity sends a chunk of data to the request handler, and one final message to say that there isn't any more data in the buffers.

```
[4] Input filter: Forwarding input: mode=0, block=0, nbytes=8192 ↩
(f 81d2228, r 81d0588).
[4] Input filter: Forwarded 6 bytes.
[4] Input filter: Sent EOS.
[4] Input filter: Input forwarding complete.
```

Shortly thereafter, the output filter will start receiving data, at which point the `RESPONSE_HEADERS` rules will be invoked:

```
[9] Output filter: Receiving output (f 81d2258, r 81d0588).
[4] Starting phase RESPONSE_HEADERS.
```

Once all the rules have run, ModSecurity will continue to store the response body in its buffers, after which it will run the `RESPONSE_BODY` rules:

```
[9] Output filter: Bucket type MMAP contains 12 bytes.
[9] Output filter: Bucket type EOS contains 0 bytes.
[4] Output filter: Completed receiving response body (buffered full - 12 bytes).
[4] Starting phase RESPONSE_BODY.
```

Again, assuming that none of the rules blocked, the accumulated response body will be forwarded to the client:

```
[4] Output filter: Output forwarding complete.
```

Finally, the logging phase will commence. The `LOGGING` rules will be run first to allow them to influence logging, after which the audit logging subsystem will be invoked to log the transaction if necessary. A message from the audit logging subsystem will be the last transaction message in the logs. In this example, ModSecurity tells us that it didn't find anything of interest in the transaction and that it sees no reason to log it:

```
[4] Initialising logging.
```

```
[4] Starting phase LOGGING.
[4] Audit log: Ignoring a non-relevant request.
```

File Upload Example

Requests that contain files are processed slightly differently. The changes can be best understood by again following the activity in the debug log:

```
[4] Input filter: Reading request body.
[9] Multipart: Boundary: ---------------------------2411583925858
[9] Input filter: Bucket type HEAP contains 256 bytes.
[9] Multipart: Added part header "Content-Disposition" "form-data; name=\"f\"; ↩
filename=\"eicar.com.txt\""
[9] Multipart: Added part header "Content-Type" "text/plain"
[9] Multipart: Content-Disposition name: f
[9] Multipart: Content-Disposition filename: eicar.com.txt
[4] Multipart: Created temporary file: ↩
/opt/modsecurity/var/tmp/20090819-175503-SowuZ38AAQEAACV-Agk-file-gmWmrF
[9] Multipart: Changing file mode to 0600: ↩
/opt/modsecurity/var/tmp/20090819-175503-SowuZ38AAQEAACV-Agk-file-gmWmrF
[9] Multipart: Added file part 9c870b8 to the list: name "f" file name ↩
"eicar.com.txt" (offset 140, length 68)
[9] Input filter: Bucket type EOS contains 0 bytes.
[4] Reqest body no files length: 96
[4] Input filter: Completed receiving request body (length 256).
```

In addition to seeing the multipart parser in action, you see ModSecurity creating a temporary file (into which it will extract the upload) and adjusting its privileges to match the desired configuration.

Then, at the end of the transaction, you will see the cleanup and the temporary file deleted:

```
[4] Multipart: Cleanup started (remove files 1).
[4] Multipart: Deleted file (part) ↩
"/opt/modsecurity/var/tmp/20090819-175503-SowuZ38AAQEAACV-Agk-file-gmWmrF"
```

The temporary file will not be deleted if ModSecurity decides to keep an uploaded file. Instead, it will be moved to the storage area:

```
[4] Multipart: Cleanup started (remove files 0).
[4] Input filter: Moved file from ↩
"/opt/modsecurity/var/tmp/20090819-175503-SowuZ38AAQEAACV-Agk-file-gmWmrF" to ↩
"/opt/modsecurity/var/upload/20090819-175503-SowuZ38AAQEAACV-Agk-file-gmWmrF".
```

In the example traces, you've observed an upload of a small file that was stored in RAM. When large uploads take place, ModSecurity will attempt to use RAM at first, switching to on-disk storage once it becomes obvious that the file is larger:

```
[9] Input filter: Bucket type HEAP contains 8000 bytes.
[9] Input filter: Bucket type HEAP contains 8000 bytes.
[9] Input filter: Bucket type HEAP contains 8000 bytes.
[9] Input filter: Bucket type HEAP contains 8000 bytes.
[9] Input filter: Bucket type HEAP contains 8000 bytes.
[9] Input filter: Bucket type HEAP contains 8000 bytes.
[9] Input filter: Bucket type HEAP contains 8000 bytes.
[9] Input filter: Bucket type HEAP contains 1536 bytes.
[9] Input filter: Bucket type HEAP contains 8000 bytes.
[9] Input filter: Bucket type HEAP contains 8000 bytes.
[9] Input filter: Bucket type HEAP contains 8000 bytes.
[9] Input filter: Bucket type HEAP contains 576 bytes.
[9] Input filter: Bucket type HEAP contains 8000 bytes.
[9] Input filter: Bucket type HEAP contains 8000 bytes.
[9] Input filter: Bucket type HEAP contains 8000 bytes.
[9] Input filter: Bucket type HEAP contains 8000 bytes.
[9] Input filter: Bucket type HEAP contains 8000 bytes.
[9] Input filter: Bucket type HEAP contains 8000 bytes.
[4] Input filter: Request too large to store in memory, switching to disk.
```

A new file will be created to store the entire raw request body:

```
[4] Input filter: Created temporary file to store request body: ↩
/opt/modsecurity/var/tmp//20090819-180105-Sowv0X8AAQEAACWAArs-request_body-4nZjqf
[4] Input filter: Wrote 129559 bytes from memory to disk.
```

This file is always deleted in the cleanup phase:

```
[4] Input filter: Removed temporary file: ↩
/opt/modsecurity/var/tmp//20090819-180105-Sowv0X8AAQEAACWAArs-request_body-4nZjqf
```

Impact on Web Server

The addition of ModSecurity will change how your web server operates. As with all Apache modules, you pay for the additional flexibility and security ModSecurity gives you with increased CPU and RAM consumption on your server. The exact amount will depend on your configuration of ModSecurity and the usage of your server. Following is a detailed list of the various activities that increase resource consumption:

- ModSecurity will add to the parsing already done by Apache, and that results in a slight increase of CPU consumption.
- Complex parsers (e.g., XML) are more expensive.
- The handling of file uploads may require I/O operations. In some cases, inbound data will be duplicated on disk.
- The parsing will add to the RAM consumption, because every extracted element (e.g., a request parameter) will need to be copied into its own space.

- Request bodies and response bodies are usually buffered in order to support reliable blocking.
- Every rule in your configuration will use some of the CPU time (for the operator) and RAM (to transform input data before it can be analyzed).
- Some of the operators used in the rules (e.g., the regular expression operator) are CPU-intensive.
- Full transaction logging is an expensive I/O operation.

In practice, this list is important because it keeps you informed; what matters is that you have enough resources to support your ModSecurity needs. If you do, then it doesn't matter how expensive ModSecurity is. Also, what's expensive to someone may not be to someone else. If you don't have enough resources to do everything you want with ModSecurity, you will need to monitor the operation of your system and remove some of the functionality to reduce the resource consumption. Virtually everything that ModSecurity does is configurable, so you should have no problems doing that.

It is generally easier to run ModSecurity in reverse proxy mode, because then you usually have an entire server (with its own CPU and RAM) to play with. In embedded mode, ModSecurity will add to the processing already done by the web server, so this method is more challenging on a busy server.

For what it's worth, ModSecurity generally uses the minimal necessary resources to perform the desired functions, so this is really a case of exchanging functionality for speed: if you want to do more, you have to pay more.

What Next?

The purpose of this section is to map your future ModSecurity activities and help you determine where to go from here. Where you will go depends on what you want to achieve and how much time you have to spend. A complete ModSecurity experience, so to speak, consists of the following elements:

Installation and configuration

This is the basic step that all users must learn how to perform. The next three chapters will teach you how to make ModSecurity operational, performing installation, general configuration, and logging configuration. Once you are done with that, you need to decide what you want to do with it. That's what the remainder of the book is for.

Rule writing

Rule writing is an essential skill. You may currently view rules as a tool to use to detect application security attacks. They are that, but they are also much more. In ModSecurity, you write rules to find out more about HTTP clients (e.g., geolocation and IP address reputation), perform long-term activity tracking (of IP addresses, sessions and users, for example), implement policy decisions (use the available information to make the decisions to warn or block), write virtual patches, and even to check on the status of ModSecurity itself.

It is true that the attack detection rules are in a class of its own, but that's mostly because, in order to write them successfully, you need to know so much about application security. For that reason, many ModSecurity users generally focus on using third-party rule sets for the attack detection. It's a legitimate choice. Not everyone has the time and inclination to become an application security expert. Even if you end up not using any inspection rules whatsoever, the ability to write virtual patches is reason enough to use ModSecurity.

Rule sets

The use of existing rule sets is the easiest way to get to the proverbial low hanging fruit: invest small effort and reap big benefits. Traditionally, the main source of ModSecurity rules has been the Core Rule Set project, now hosted with OWASP. On the other hand, if you are keen to get your hands dirty, I can tell you that I draw great pleasure from writing my own rules. It's a great way to learn about application security. The only drawback is that it requires a large time investment.

Remote logging and alert management GUI

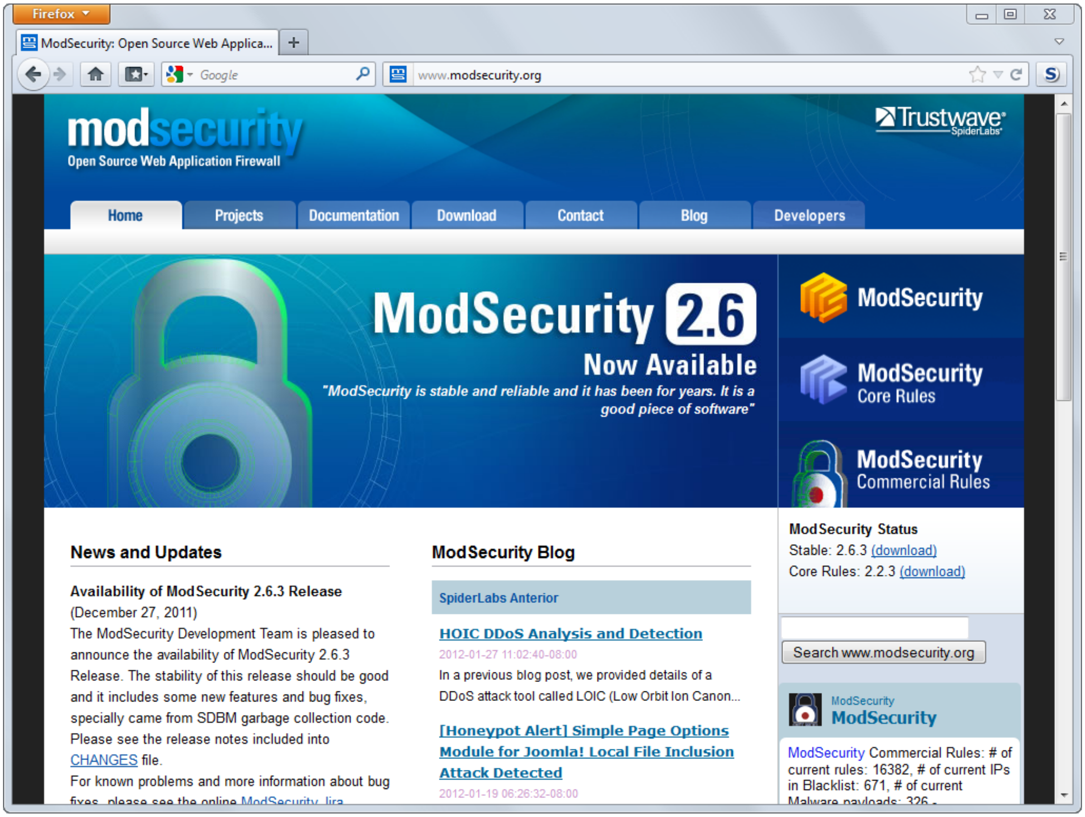
ModSecurity is perfectly usable without a remote logging solution and without a GUI (the two usually go together). Significant error messages are copied to Apache's error log. Complete transactions are usually logged to the audit log. With a notification system in place, you will know when something happens, and you can visit the audit logs to investigate. For example, many installations will divert Apache's error log to a central logging system (via syslog).

The process does become more difficult with more than one sensor to manage. Furthermore, GUIs make the whole experience of monitoring much more pleasant. For that reason you will probably seek to install one of the available remote centralization tools and use their GUIs. The available options are listed in the Resources section, which follows.

Resources

This section contains a list of assorted ModSecurity resources that can assist you in your work.

Figure 2-1. The homepage of www.modsecurity.org


Firefox
ModSecurity: Open Source Web Applica...
Google
www.modsecurity.org
modsecurity
Open Source Web Application Firewall
Trustwave SpiderLabs
Home
Projects
Documentation
Download
Contact
Blog
Developers
ModSecurity 2.6
Now Available
ModSecurity
ModSecurity Core Rules
ModSecurity Commercial Rules
Search www.modsecurity.org
ModSecurity
ModSecurity

"ModSecurity is stable and reliable and it has been for years. It is a good piece of software"

News and Updates

Availability of ModSecurity 2.6.3 Release
(December 27, 2011)
The ModSecurity Development Team is pleased to announce the availability of ModSecurity 2.6.3 Release. The stability of this release should be good and it includes some new features and bug fixes, specially came from SDBM garbage collection code. Please see the release notes included into CHANGES file.
For known problems and more information about bug fixes, please see the online ModSecurity Jira

ModSecurity Blog

SpiderLabs Anterior

HOIC DDoS Analysis and Detection
2012-01-27 11:02:40-08:00
In a previous blog post, we provided details of a DDoS attack tool called LOIC (Low Orbit Ion Canon...

[Honeypot Alert] Simple Page Options Module for Joomla! Local File Inclusion Attack Detected
2012-01-19 06:26:32-08:00

ModSecurity Status
Stable: 2.6.3 (download)
Core Rules: 2.2.3 (download)

ModSecurity Commercial Rules: # of current rules: 16382, # of current IPs in Blacklist: 671, # of current Malware payloads: 326 -

General Resources

The following resources are the bare essentials:

ModSecurity web site

ModSecurity's web site is probably going to be your main source of information. You should visit the web site from time to time, as well as subscribe to receive the updates from the blog.

Official documentation

The official ModSecurity documentation is maintained in a wiki, but copies of it are made for inclusion with every release.

Issue tracker

The ModSecurity issue tracker is the place you will want to visit for one of two reasons: to report a problem with ModSecurity itself (e.g., when you find a bug) or to check out the progress on the next (major or minor) version. Before reporting any problems, go through the Support Checklist, which will help you assemble the information required to help resolve your problem. Providing as much information as you can will help the developers understand and replicate the problem, and provide a fix (or a workaround) quickly.

Users' mailing list

The users' mailing list (mod-security-users@lists.sourceforge.net) is a general-purpose mailing list where you can discuss ModSecurity. Feel free to ask questions, propose improvements, and discuss ideas. That is the place where you'll hear first about new ModSecurity versions.

ModSecurity@Freshmeat

If you subscribe to the users' mailing list, you will generally find out about new versions of ModSecurity as soon as they are released. If you care only about version releases, however, you may consider subscribing to the new version notifications at the ModSecurity page at Freshmeat.

Core Rules mailing list

Starting with version 2, the Core Rules project is part of OWASP, and has a separate mailing list (owasp-modsecurity-core-rule-set@lists.owasp.org).

Developer Resources

If you are interested in development work, you will need these:

Developers' mailing list

The developers' mailing list is generally a lonely place, but if you do decide to start playing with the ModSecurity source code, this list is the place to go to discuss your work.

Source code access

The source code of ModSecurity is hosted at a Subversion repository at SourceForge, which allows you to access it directly or through a web-based user interface.

FishEye interface

If you are not looking to start developing immediately but still want to have a look at the source code of ModSecurity, I recommend that you use the ModSecurity FishEye interface, which is much better than the stock interface available at SourceForge.

AuditConsole

Using ModSecurity entirely from the command line is possible but not much fun. The configuration part is not a problem, but reviewing logs is difficult without higher-level tools. Your best choice for a log centralization and GUI tool is AuditConsole, which is built by Christian Bockermann and hosted on *www.jwall.org*.

AuditConsole is free and provides the following features:

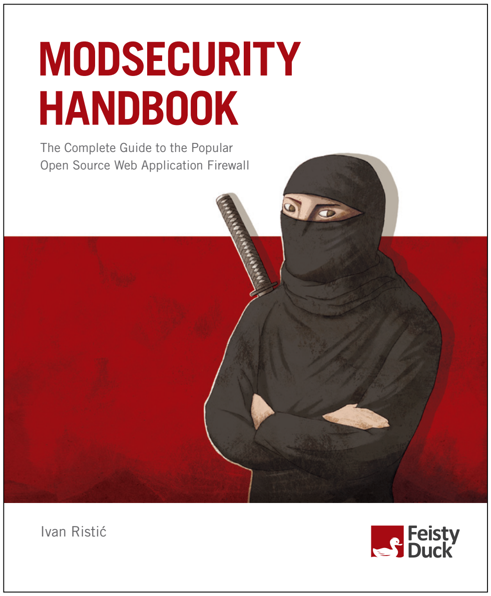
- Event centralization from multiple remote ModSecurity installations
- Event storage and retrieval
- Support for multiple user accounts and support for different views
- Event tagging
- Event rules, which are executed in the console

Summary

This chapter was your ModSecurity orientation. I introduced ModSecurity at a high level, discussed what it is and what it isn't, and what it can do and what it cannot. I also gave you a taste of what ModSecurity is like and described common usage scenarios, as well as covered some of the interesting parts of its operation.

The foundation you now have should be enough to help you set off on a journey of ModSecurity exploration. The next chapter discusses installation.

Everything you need to know about ModSecurity, in one place



MODSECURITY HANDBOOK

The Complete Guide to the Popular Open Source Web Application Firewall

Ivan Ristić

Feisty Duck

To purchase the full book, go to:
https://www.feistyduck.com

Feisty Duck
FINE TECHNOLOGY BOOKS