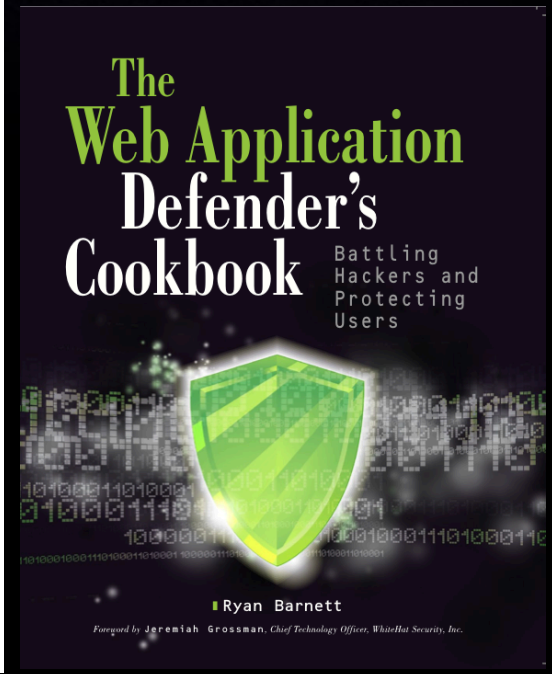
Blackhat Arsenal: ModSecurity Overview

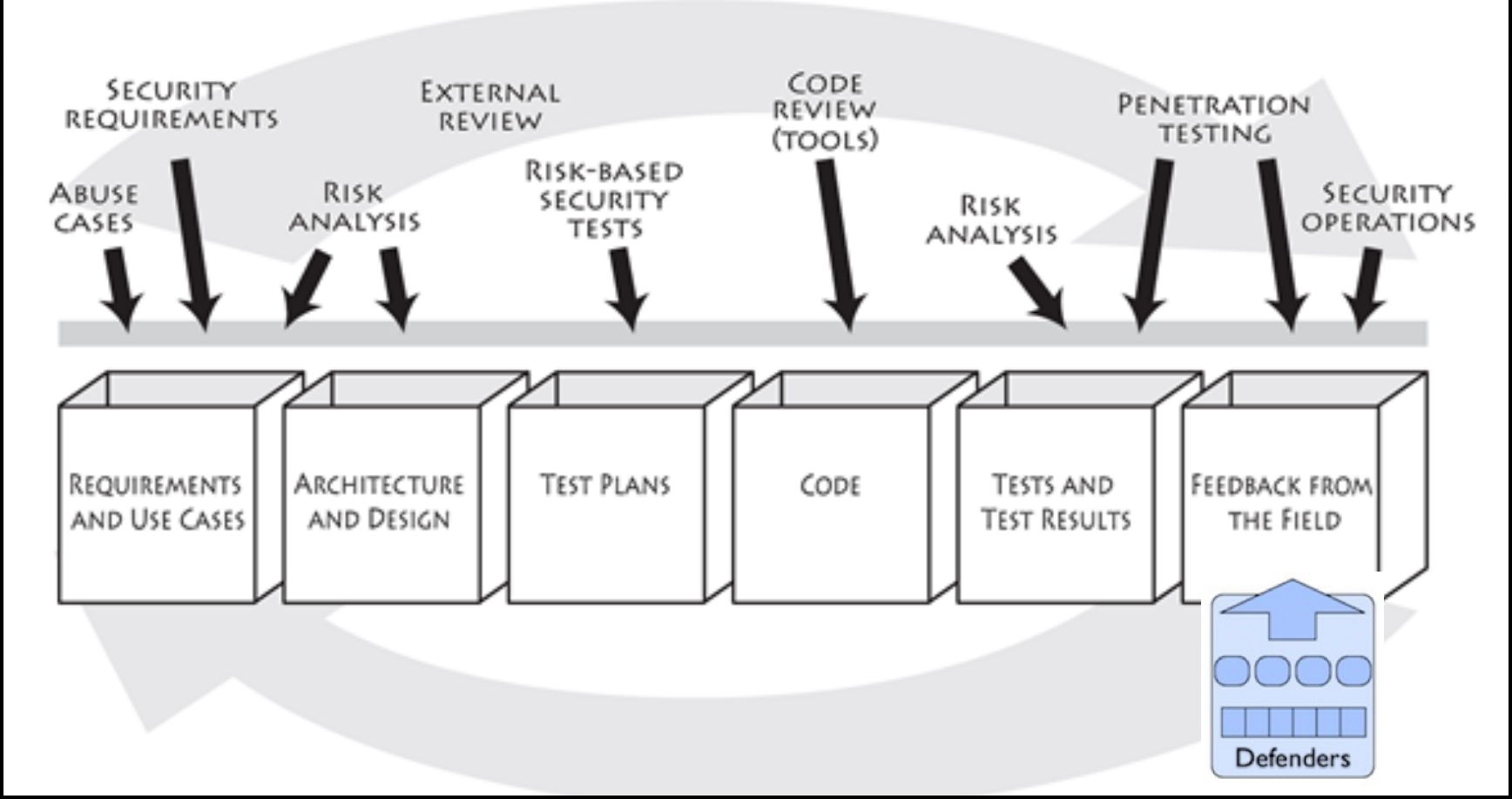
Lead Security Researcher
Trustwave SpiderLabs
rbarnett@trustwave.com
@ryancbarnett

Speaker Info: Ryan Barnett

- Trustwave SpiderLabs Research
  - Specialize in Web Application Defense/WAF Research
    - WebDefend (Commercial)
    - ModSecurity (Open Source)
- OWASP
  - Lead the ModSecurity Core Rule Set (CRS) Project
- Author
  - ***The Web Application Defender's Cookbook***



Target Audience: Web Defenders



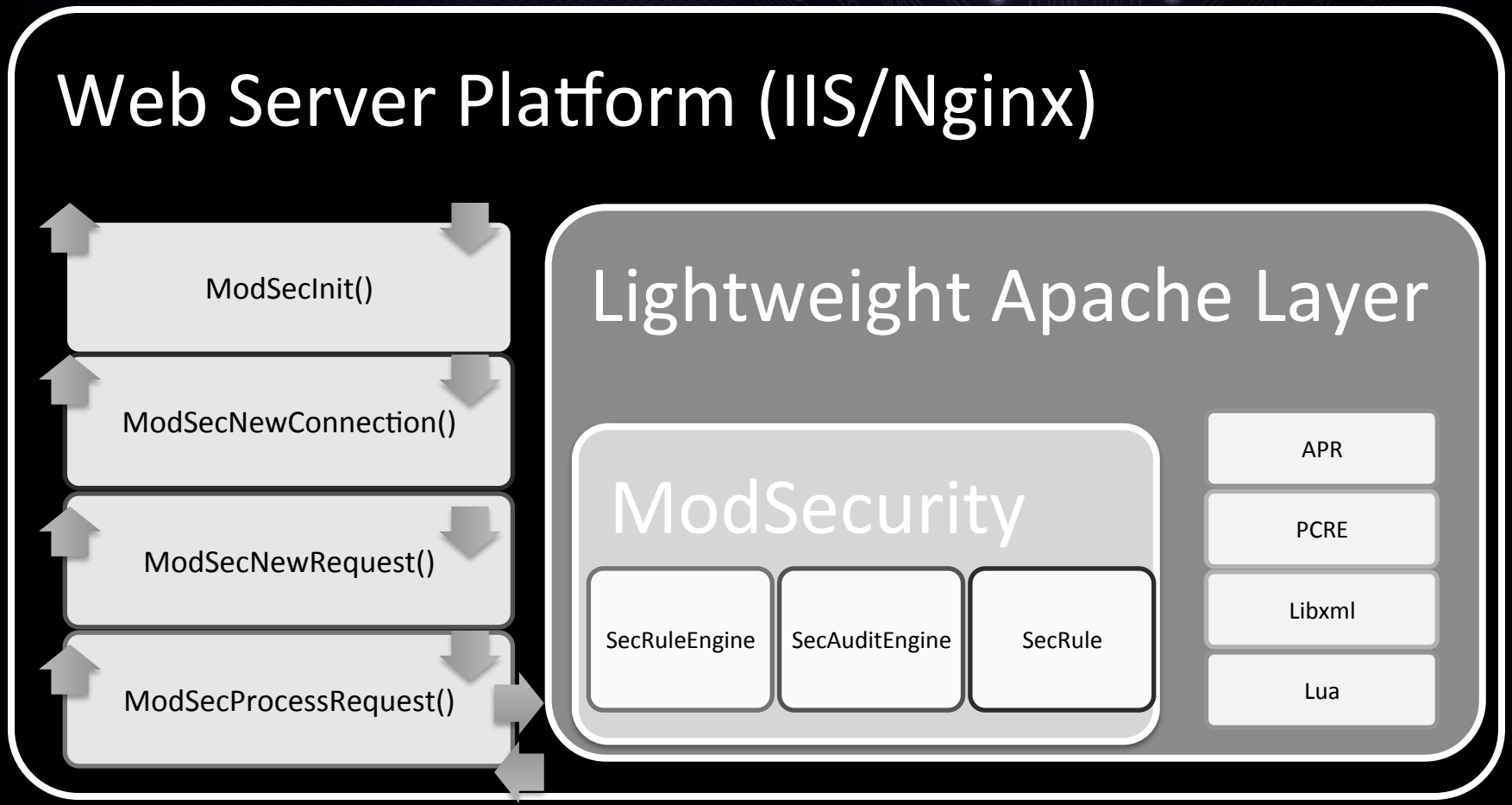
http://www.swsec.com/resources/touchpoints/

Agenda Topics

- ModSecurity Overview
  - Project Overview
  - Installation Options
  - Recommended Base Configuration
- OWASP Core Rule Set
  - GitHub Repo
  - Regression Tests
  - Convert DAST XML to Virtual Patches
- Advanced Usage Examples
  - DAST/WAF Integration
  - HMAC Token Validation
  - Bayesian Analysis
- Trustwave SpiderLabs Commercial Rules Feed
  - Virtual Patches
  - IP Reputation
  - Malware Detection


modsecurity
Open Source Web Application Firewall
Trustwave SpiderLabs
Home
Projects
Documentation
Download
Contact
Blog
Developers
ModSecurity 2.7
Now Available
IIS7
nginx
ModSecurity
ModSecurity Core Rules
ModSecurity Commercial Rules
News and Updates
Availability of ModSecurity 2.7.0 Stable Release
(October 16, 2012)
The ModSecurity Development Team is pleased to announce the availability of ModSecurity 2.7.0 Stable Release.The stability of this release is good and includes many new features and bug fixes. Highlights include:
Internationalization (I18N) Support
HMAC Token Injection to prevent data
ModSecurity Blog
SpiderLabs Anterior
Announcing the availability of ModSecurity extension for Nginx
2012-09-28 13:54:56-05:00
ModSecurity for Nginx ModSecurity for Nginx is a web server plug-in for the Nginx web server platf...
How Should WAFs Handle Authorized Vulnerability Scanning Traffic?
ModSecurity Status (v2.7.0)
Apache (Stable): download
IIS (Beta): download
Nginx (Beta): download
Search www.modsecurity.org
ModSecurity
ModSecurity
ModSecurity Announcing Availability of ModSecurity v2.7.0 Stable Release -

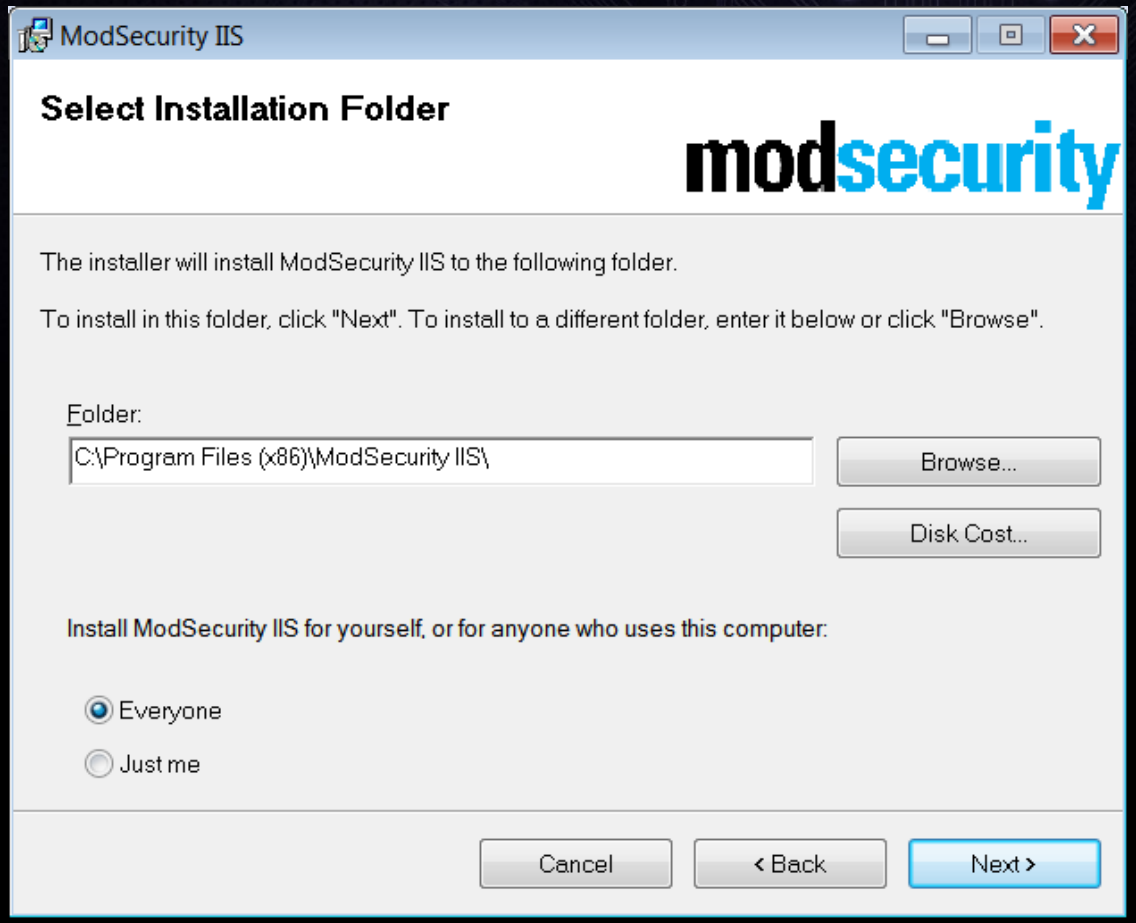
Cross-Platform Support:
Standalone ModSecurity API



Installation Options – OS Repos

- Install Binaries from OS Repositories
  - Fedora Core, CentOS and RedHat Enterprise Linux
    - **# yum install mod_security**
  - Debian or Ubuntu
    - **# apt-get install libapache-mod-security**
- Web Admin Panels
    - cPanel
- Pros
  - Easy Installation
- Cons
  - May not be the latest version

Installation Options - MSI Installer



Installation Options – Source Code

- Download Source Archive
  - **$ wget http://www.modsecurity.org/download/modsecurity-apache_2.7.0.tar.gz**
- Download from Repository
  - **svn export https://mod-security.svn.sourceforge.net/svnroot/mod-security/m2/trunk modsecurity-trunk**
- Dependencies
  - Required
    - Apache Portable Runtime (APR)
    - APR-Util
    - Mod_unique_id
    - PCRE
    - libxml2
  - Optional
    - Lua 5.1
    - libcurl

Configuration, Compilation and Installation

- Configure
  - **$ ./configure**
- New Performance Options for v2.7.0
  - --enable-pcre-jit
  - --enable-lua-cache
- Make
  - **$ make**
- Make Install
  - **$ sudo make install**
- Activate in httpd.conf
  - LoadFile /usr/lib/libxml2.so
  - LoadFile /usr/lib/liblua5.1.so
  - LoadModule security2_module modules/mod_security2.so

Example Config Activation

httpd.conf:

```
Include modsecurity.conf
```

web.config:

```
<Modsecurity enabled="true"
configFile="modsecurity.conf" />
```

nginx.conf:

```
ModSecurityConfig modsecurity.conf
ModSecurityEnable On
```

```
# Prevent path traversal
SecRule REQUEST_URI|ARGS

# Prevent XSS atacks (HTM
injection)
SecRule REQUEST_URI|ARGS

# Very crude filters to p
injection attacks
SecRule REQUEST_URI|ARGS
"delete[[:space:]]+from"
SecRule REQUEST_URI|ARGS
"insert[[:space:]]+into"
SecRule REQUEST_URI|ARGS
```

OWASP ModSecurity Core Rule Set (CRS)

Related Projects | Release History | Roadmap

Overview

ModSecurity™ is a web application firewall engine that provides very little protection on its own. In order to become useful, ModSecurity™ must be configured with rules. In order to enable users to take full advantage of ModSecurity™ out of the box, Trustwave's SpiderLabs is sponsoring and maintaining a free certified rule set for the community. Unlike intrusion detection and prevention systems, which rely on signatures specific to known vulnerabilities, the Core Rules provide generic protection from unknown vulnerabilities often found in web applications, which are in most cases custom coded. The Core Rules are heavily commented to allow it to be used as a step-by-step deployment guide for ModSecurity™.

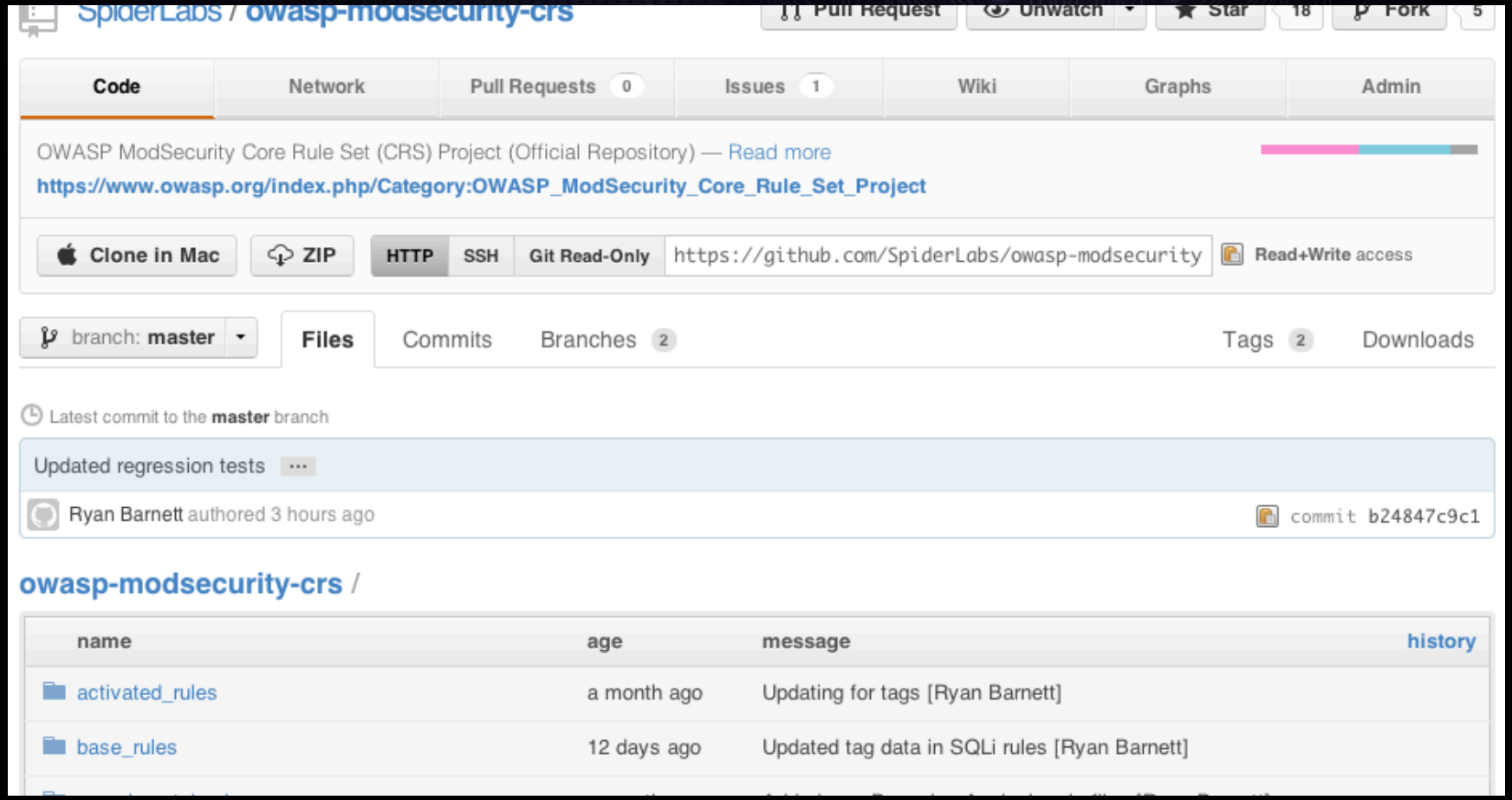
Donate funds to OWASP earmarked for ModSecurity Core Rule Set Project.

Core Rules Content

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

- **HTTP Protection** - detecting violations of the HTTP protocol and a locally defined usage policy.
- **Real-time Blacklist Lookups** - utilizes 3rd Party IP Reputation
- **Web-based Malware Detection** - identifies malicious web content by check against the Google Safe Browsing API.
- **HTTP Denial of Service Protections** - defense against HTTP Flooding and Slow HTTP DoS Attacks.
- **Common Web Attacks Protection** - detecting common web application security attack.
- **Automation Detection** - Detecting bots, crawlers, scanners and other surface malicious activity.
- **Integration with AV Scanning for File Uploads** - detects malicious files uploaded through the web application.
- **Tracking Sensitive Data** - Tracks Credit Card usage and blocks leakages.
- **Trojan Protection** - Detecting access to Trojans horses.

Download from GitHub Repo



Use Regression Test Suite

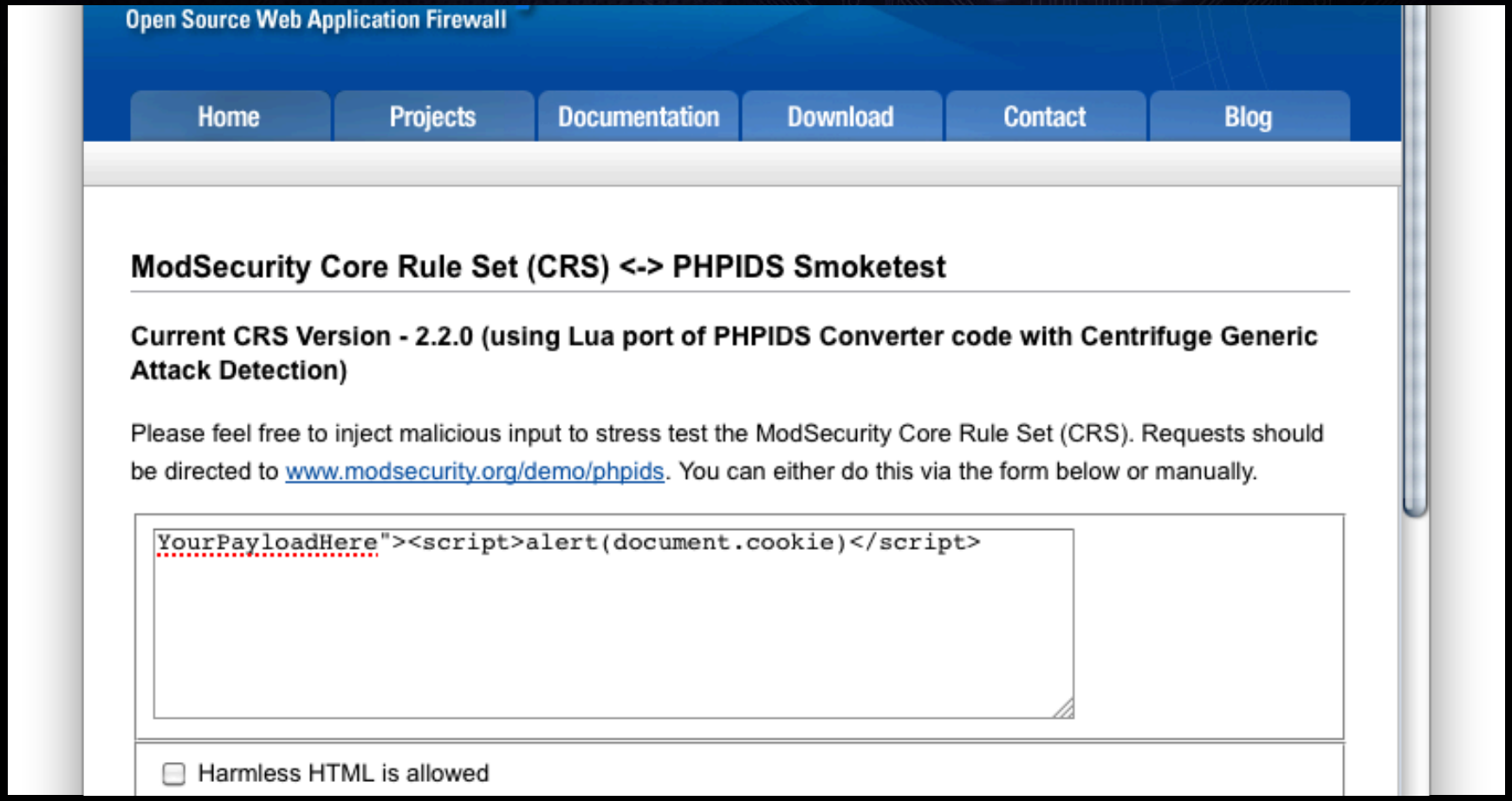
```
$ ./rulestest.pl -s 127.0.0.1:80 tests/*41*

ModSecurity rules test report generated to STDOUT on Wed Oct 17 10:08:28
2012
Produced by rulestest.pl, (c) Trustwave Holdings Inc, 2012

## reading tests file rulestest.conf

## reading tests file tests/modsecurity_crs_41_sql_injection_attacks.tests
OK: SQL Comment Sequence Detected (981231) (sig=%E2%80%98%20or%201%3D1--
%20-,hostname=mysite), status = 403, no events received
OK: SQL Comment Sequence Detected (981231) (sig=SELECT%2F*avoid-spaces*
%2Fpassword%2F**%2FFROM%2F**%2FMembers,hostname=mysite), status = 403, no
events received
OK: SQL Comment Sequence Detected (981231) (sig=%E2%80%98%20or
%201%3D1%23%0A,hostname=mysite), status = 403, no events received
…
```

CRS Demo


Open Source Web Application Firewall
Home
Projects
Documentation
Download
Contact
Blog
ModSecurity Core Rule Set (CRS) <-> PHPIDS Smoketest
Current CRS Version - 2.2.0 (using Lua port of PHPIDS Converter code with Centrifuge Generic Attack Detection)
Please feel free to inject malicious input to stress test the ModSecurity Core Rule Set (CRS). Requests should be directed to www.modsecurity.org/demo/phpids. You can either do this via the form below or manually.
YourPayloadHere"><script>alert(document.cookie)</script>
Harmless HTML is allowed

CRS Demo

CRS Anomaly Score Exceeded (score 50): IE XSS Filters - Attack Detected

All Matched Rules Shown Below

950901 SQL Injection Attack

Matched ***script>alert*** at ARGS:test

981173 Restricted SQL Character Anomaly Detection Alert - Total # of special characters exceeded

Matched ***"><script>alert(document.cookie)</script>*** at ARGS:test

---

981245 Detects basic SQL authentication bypass attempts 2/3

Matched ***"><script>a*** at ARGS:test

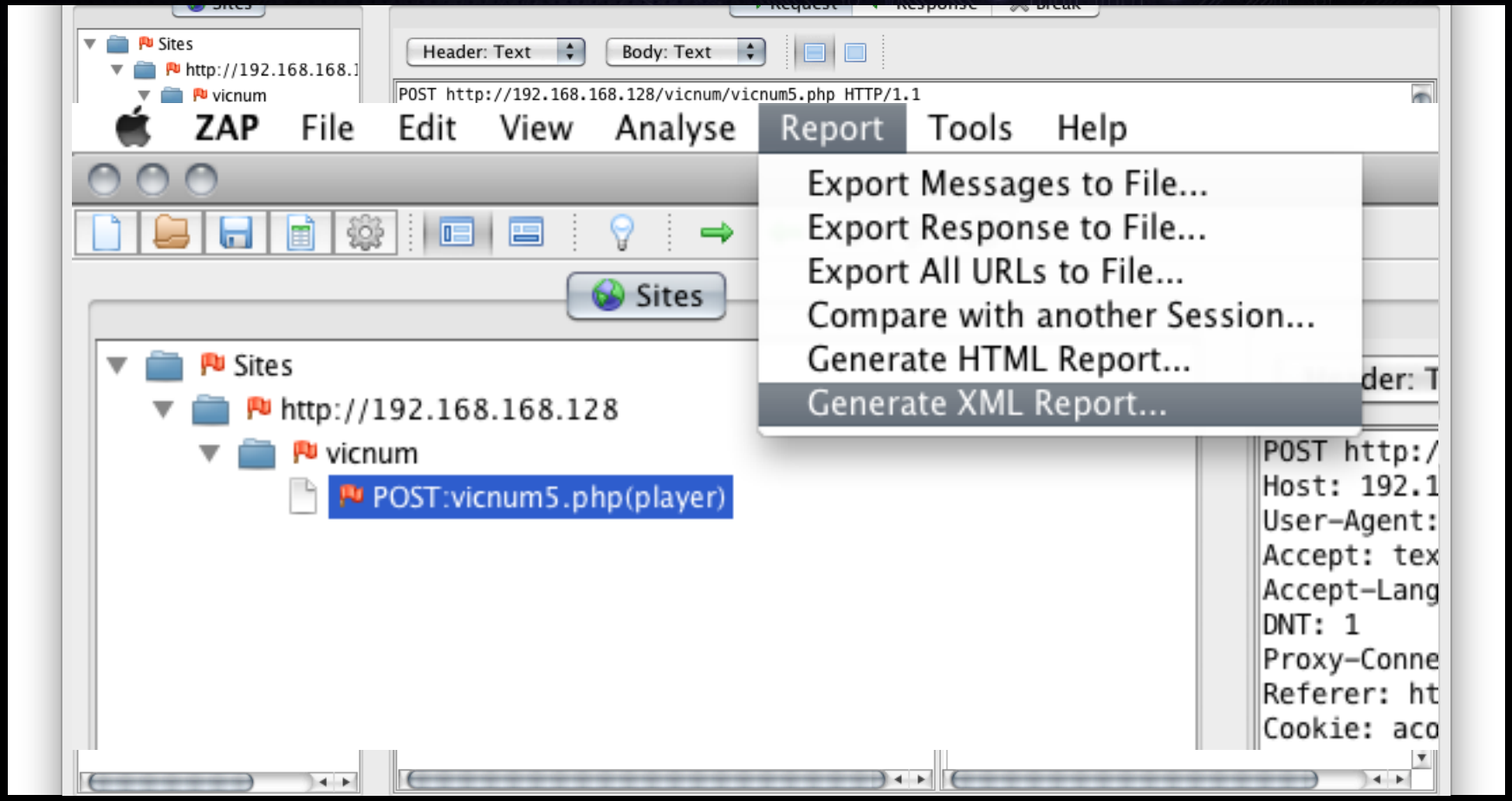
958001 Cross-site Scripting (XSS) Attack

Matched ***document.cookie*** at ARGS:test

958052 Cross-site Scripting (XSS) Attack

Matched ***alert(*** at ARGS:test

OWASP Zed Attack Proxy (ZAP)



https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

ZAP (v 1.4) XML Report Data

<alertitem>
<pluginid>40005</pluginid>
<alert>SQL Injection</alert>
<riskcode>3</riskcode>
<reliability>1</reliability>
<riskdesc>High (Suspicious)</riskdesc>
<desc>SQL injection is possible. User parameters submitted will be formulated into a SQL query for database processing. If the query is built by simple 'string concatenation', it is possible to modify the meaning of the query by carefully crafting the parameters. Depending on the access right and type of database used, tampered query can be used to retrieve sensitive information from the database or execute arbitrary code. MS SQL and PostGreSQL, which supports multiple statements, may be exploited if the database access right is more powerful.

This can occur in URL query strings, POST paramters or even cookies. Currently check on cookie is not supported by Paros. You should check SQL injection manually as well as some blind SQL injection areas cannot be discovered by this check.

</desc>
<uri>http://192.168.168.128/vicnum/vicnum5.php</uri>
<param>player</param>
<attack>test%27INJECTED_PARAM'INJECTED_PARAM</attack>
--CUT--
</alertitem>

Auto-Convert DAST XML to ModSecurity

sp-modsecurity-crs / util

name	age	message	his
.			
oneypot_sensor	19 days ago	Updated README [Ryan Barnett]	
egression_tests	20 hours ago	Updated regression tests [Ryan Barnett]	
unAV	a month ago	Updating for tags [Ryan Barnett]	
README	a month ago	Updating for tags [Ryan Barnett]	
rachni2modsec.pl	a month ago	Updating for tags [Ryan Barnett]	
ules-updater-example.conf	a month ago	Updating for tags [Ryan Barnett]	
ules-updater.pl	a month ago	Updating for tags [Ryan Barnett]	
ules-updater.pl.in	a month ago	Updating for tags [Ryan Barnett]	
unav.pl	a month ago	Updating for tags [Ryan Barnett]	
ap2modsec.pl	a month ago	Updating for tags [Ryan Barnett]	

https://github.com/SpiderLabs/owasp-modsecurity-crs/tree/master/util

Script Usage

```
$ ./zap2modsec.pl -f zap-vicnum.xml
============================================================
Vulnerability[3] -  Type: SQL Injection
Found a SQL Injection vulnerability.
Validating URL: http://192.168.168.128/vicnum/vicnum5.php
URL is well-formed
Continuing Rule Generation
Current vulnerable Param(s): player
SQL Injection (uricontent and param) rule successfully generated and
saved in ./modsecurity_crs_48_virtual_patches.conf.
============================================================
--CUT--

************ END OF SCRIPT RESULTS *****************
Number of Vulnerabilities Processed:    5
Number of ModSecurity rules generated:   2
Number of Unsupported vulns skipped:     2
Number of bad URLs (rules not gen):      0
************************************************
```

New Virtual Patches

```
#
# OWASP ZAP Virtual Patch Details:
# ID: 13
# Type: SQL Injection
# Vulnerable URL: vicnum/vicnum5.php
# Vulnerable Parameter: player
#
SecRule REQUEST_FILENAME "vicnum/vicnum5.php" "chain,phase:
2,t:none,block,msg:'Virtual Patch for SQL
Injection',id:'13',tag:'WEB_ATTACK/SQL_INJECTION',tag:'WASCTC/
WASC-19',tag:'OWASP_TOP_10/A1',tag:'OWASP_AppSensor/CIE1',tag:'PCI/
6.5.2',logdata:'%{matched_var_name}',severity:'2'"
        SecRule &TX:'/SQL_INJECTION.*ARGS:player/' "@gt 0"
"setvar:'tx.msg=%{rule.msg}',setvar:tx.sql_injection_score=+%
{tx.critical_anomaly_score},setvar:tx.anomaly_score=+%
{tx.critical_anomaly_score}"
```

DAST <-> WAF Integration

- In order to integrate DAST/WAF, the scanner needs to be run as a service
  - Not as a client desktop app
  - Need an API Service
- Using Arachni Scanner
  - Written by Tasos Laskos (@Zap0tek)
  - Developed in Ruby
  - XMLRPC service

http://arachni-scanner.com/

WAF <-> DAST Integration Workflow

Web Client

- Sends web request
- This initiates web assessment of resource (e.g. login page)

Web Server w/ ModSecurity

- Use Lua API to communicate w/Arachni RPC
- Initiate Scan
- Pull Report
- Update Protections

Arachni Scanning Host

- RPC Service
- Targeted Assessments
- Generate Reports

ModSecurity Rules

- Initiate an Arachni Scan

```
SecRule &RESOURCE:ARACHNI_SCAN_COMPLETED "@eq 0"
"chain,phase:5,t:none,log,pass"
      SecRule &ARGS "@gt 0" "exec:/etc/apache2/
modsecurity-crs/base_rules/arachni_integration.lua"
```

- Disable ModScurity for Arachni Scanning

```
SecRule REMOTE_ADDR "@ipMatch 192.168.168.128"
"chain,phase:1,t:none,nolog,pass"
      SecRule REQUEST_HEADERS:User-Agent
"@beginsWith Arachni/" "ctl:ruleEngine=Off"
```

Apache Access Log

1. **192.168.168.1** - - [05/Apr/2012:11:35:47 -0400] "POST /vicnum/vicnum5.php HTTP/1.1" 200 1022 "http://192.168.168.128/vicnum/" **"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:11.0) Gecko/20100101 Firefox/11.0"**

Scanning Script Initiated

```
Lua: Executing script: /etc/apache2/modsecurity-crs/base_rules/
arachni_integration.lua
Arachni: Host: 192.168.168.128
Arachni: Filename: /vicnum/vicnum5.php
Arachni: URL to scan is: http://192.168.168.128/vicnum/vicnum5.php
Arachni: Request Method is: POST
Arachni: Arg Name: player and Value: test.
Arachni: Updated ARGS table is: ---
player: test

Arachni: Updated Cookies table is: --- {}

Arachni: Yaml output of vectors is: ---
- inputs:
    player: test
  type: form
  method: POST
  action: http://192.168.168.128/vicnum/vicnum5.php
```

Arachni RPC Service

```
I, [2012-04-05T11:33:32.006918 #3771]  INFO -- System: RPC Server started.
I, [2012-04-05T11:33:32.007164 #3771]  INFO -- System: Listening on 192.168.168.128:44604
I, [2012-04-05T11:35:47.390623 #3746]  INFO -- Call: dispatcher.dispatch [192.168.168.128]
I, [2012-04-05T11:35:47.419363 #3748]  INFO -- Call: modules.load [192.168.168.128]
Arachni - Web Application Security Scanner Framework v0.4.1 [0.2.5]
       Author: Tasos "Zapotek" Laskos <tasos.laskos@gmail.com>
                                      <zapotek@segfault.gr>
               (With the support of the community and the Arachni Team.)

       Website:       http://github.com/Zapotek/arachni
       Documentation: http://github.com/Zapotek/arachni/wiki


I, [2012-04-05T11:35:47.451187 #3748]  INFO -- Call: plugins.load [192.168.168.128]
I, [2012-04-05T11:35:47.447358 #3837]  INFO -- System: RPC Server started.
I, [2012-04-05T11:35:47.453383 #3837]  INFO -- System: Listening on 192.168.168.128:61420
I, [2012-04-05T11:35:47.459832 #3748]  INFO -- Call: opts.set [192.168.168.128]
I, [2012-04-05T11:35:47.487119 #3748]  INFO -- Call: framework.run [192.168.168.128]
```

ModSecurity's RESOURCE Collection

```
Re-retrieving collection prior to store: resource
Wrote variable: name "__expire_KEY", value "1333644233".
Wrote variable: name "KEY", value "192.168.168.128_/vicnum/vicnum5.php".
Wrote variable: name "TIMEOUT", value "3600".
Wrote variable: name "__key", value "192.168.168.128_/vicnum/vicnum5.php".
Wrote variable: name "__name", value "resource".
Wrote variable: name "CREATE_TIME", value "1333640632".
Wrote variable: name "UPDATE_COUNTER", value "1".
Wrote variable: name "min_pattern_threshold", value "50".
Wrote variable: name "min_traffic_threshold", value "100".
Wrote variable: name "arachni_scan_initiated", value "1".
Wrote variable: name "arachni_instance_info_port", value "30118".
Wrote variable: name "arachni_instance_info_token", value "c5ab2feb9072ed8e7737f7d526e7b254".
Wrote variable: name "traffic_counter", value "1".
Wrote variable: name "request_method_counter_POST", value "1".
Wrote variable: name "NumOfArgs_counter_1", value "1".
Wrote variable: name "args_names_counter_player", value "1".
Wrote variable: name "ARGS:player_length_4_counter", value "1".
Wrote variable: name "ARGS:player_alpha_counter", value "1".
Wrote variable: name "LAST_UPDATE_TIME", value "1333640633".
Persisted collection (name "resource", key "192.168.168.128_/vicnum/vicnum5.php").
```

Apache Access Log

1. **192.168.168.1** - - [05/Apr/2012:11:35:47 -0400] "POST /vicnum/vicnum5.php HTTP/1.1" 200 1022 "http://192.168.168.128/vicnum/" **"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:11.0) Gecko/20100101 Firefox/11.0"**
2. **192.168.168.128** - - [05/Apr/2012:11:35:48 -0400] "POST /vicnum/vicnum5.php HTTP/1.1" 200 1107 "-" **"Arachni/0.4.1"**
3. 192.168.168.128 - - [05/Apr/2012:11:35:48 -0400] "POST /vicnum/vicnum5.php HTTP/1.1" 200 1022 "-" "Arachni/0.4.1”
4. 192.168.168.128 - - [05/Apr/2012:11:35:48 -0400] "POST /vicnum/vicnum5.php HTTP/1.1" 200 1022 "-" "Arachni/0.4.1"
5. 192.168.168.128 - - [05/Apr/2012:11:35:48 -0400] "POST /vicnum/vicnum5.php HTTP/1.1" 200 1116 "-" "Arachni/0.4.1”
6. 192.168.168.128 - - [05/Apr/2012:11:35:48 -0400] "POST /vicnum/vicnum5.php HTTP/1.1" 200 1100 "-" "Arachni/0.4.1"
7. 192.168.168.128 - - [05/Apr/2012:11:35:48 -0400] "POST /vicnum/vicnum5.php HTTP/1.1" 200 1081 "-" "Arachni/0.4.1”
8. 192.168.168.128 - - [05/Apr/2012:11:35:48 -0400] "POST /vicnum/vicnum5.php HTTP/1.1" 200 1082 "-" "Arachni/0.4.1”
9. …

Pulling Arachni Report

```
Arachni: Previous scan was initiated, checking scan status.
Arachni: Port info: 30118 and Token info: c5ab2feb9072ed8e7737f7d526e7b254
Arachni: Scan completed - calling for report.
Arachni: Yaml Results: ---
- cwe: '79'
  description: "Client-side code (like JavaScript) can\n                    be injected
    into the web application which is then returned to the user's browser.\n                    This
    can lead to a compromise of the client's system or serve as a pivoting point for
    other attacks."
  references:
    ha.ckers: http://ha.ckers.org/xss.html
    Secunia: http://secunia.com/advisories/9716/
  variations: []
  _hash: d241855ec9dd4694f6eaf28e28a0913f
  mod_name: XSS
  var: player
  elem: form
  url: http://192.168.168.128/vicnum/vicnum5.php
  cvssv2: '9.0'
  method: POST
```

Updated RESOURCE Data

```
Wrote variable: name "min_pattern_threshold", value "50".
Wrote variable: name "min_traffic_threshold", value "100".
Wrote variable: name "arachni_scan_initiated", value "1".
Wrote variable: name "arachni_instance_info_port", value "30118".
Wrote variable: name "arachni_instance_info_token", value
"c5ab2feb9072ed8e7737f7d526e7b254".
Wrote variable: name "traffic_counter", value "2".
Wrote variable: name "request_method_counter_POST", value "2".
Wrote variable: name "NumOfArgs_counter_1", value "2".
Wrote variable: name "args_names_counter_player", value "2".
Wrote variable: name "ARGS:player_length_4_counter", value "2".
Wrote variable: name "ARGS:player_alpha_counter", value "2".
Wrote variable: name "LAST_UPDATE_TIME", value "1333640642".
Wrote variable: name "xss_vulnerable_params", value "player".
Wrote variable: name "sqli_vulnerable_params", value "player".
Wrote variable: name "arachni_scan_completed", value "1".
Persisted collection (name "resource", key "192.168.168.128_/vicnum/
```

ModSecurity Correlation Rules

```
SecRule TX:/XSS-ARGS:/ ".*" "id:'999003',chain,phase:
2,t:none,msg:'XSS Attack Against Known Vulnerable
Parameter.',logdata:'%{matched_var}'"
        SecRule MATCHED_VARS_NAMES "-ARGS:(.*)$"
"chain,capture"
                SecRule TX:1 "@within %
{resource.xss_vulnerable_params}"


SecRule TX:/SQL_INJECTION-ARGS:/ ".*"
"id:'999004',chain,phase:2,t:none,msg:'SQLi Attack Against
Known Vulnerable Parameter.',logdata:'%{matched_var}'"
        SecRule MATCHED_VARS_NAMES "-ARGS:(.*)$"
"chain,capture"
                SecRule TX:1 "@within %
{resource.sqli_vulnerable_params}"
```

Time-to-Fix Metric

- On-Demand Arachni Scan Initiated

```
192.168.168.128 - - [05/Apr/2012:11:43:54 -0400]
"POST /vicnum/vicnum5.php HTTP/1.1" 200 1022 "-"
"Arachni/0.4.1"
```

- Report Pulled and Vulnerability Data Identified

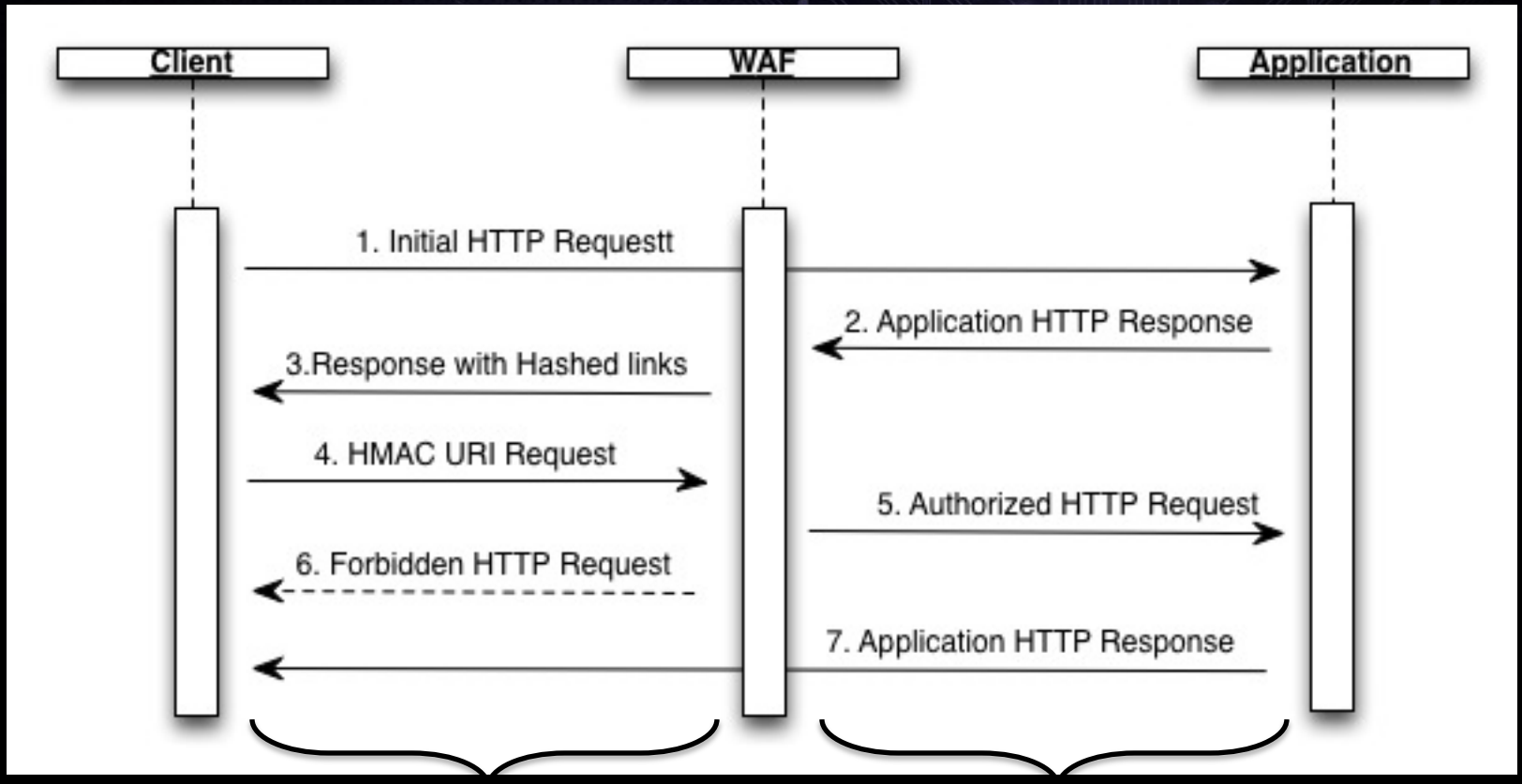
```
[05/Apr/2012:11:44:02 --0400] [192.168.168.128/
sid#b819f888][rid#b98cf7f8][/vicnum/vicnum5.php]
[9] Set variable
"RESOURCE.sqli_vulnerable_params" to "player".
```

- Time-to-Fix
  - **8 seconds ☺**

HMAC Token Validation

- Inspect Outbound HTML for the following elements:
  - Href, form action
  - frame, iframe
  - Location response header (http 3xx status)
  - PostBack
  - Hidden fields, etc
- Inject HMAC tokens to validate data.
  - Prevents data manipulation.
- Working with HMAC (RFC 2104) algorithm.
- Don't need to change the application.

Protocol



Data to be checked

Data to be authenticated

ModSecurity v2.7.0 Directives

```
SecEncryptionEngine On
SecEncryptionParam "hmac”
SecEncryptionKey "rand" "KeyOnly”
SecEncryptionMethodrx "HashHref”
“product_id”

SecRule REQUEST_URI
"@validateEncryption product_id"
"phase:2,id:1001,deny"
```

Example

Response body before MACing:

<html dir="LTR" lang="br"><head>
<title> ModSecurity Test Crypto </title>

NEW
</meta></head></html>

Data to be Authenticated

Response body After MACing:

<html dir="LTR" lang="br"><head>
<title> ModSecurity Test Crypto </title>

<a href=
http://192.168.0.101:80/catalog/index.php?
product_id=71&osCsid=12345&hmac=371fd57625df12abcd5646352ffd8432>NEW
</head></html>

Data Authenticated

Common SQL Injection Methodology

- Automation to identify injection points
  - NetSparker
  - Arachni
  - Sqlmap
  - Havij

- Manual testing to develop working SQLi payloads
  - ***An iterative process of trial and error***

1. Send initial payloads and observe DB responses
2. Use obfuscation tactics (comments, encodings, etc…)
3. Send payload and observe DB response
4. Repeat steps 2 - 3

Iterative Testing Example

div 1 union%23%0Aselect 1,2,current_user

div 1 union%23foo*/*bar%0Aselect 1,2,current_user

div 1 union%23foofoofoofoo*/*bar%0Aselect 1,2,current_user

div 1 union%23foofoofoofoofoofoofoofoofoofoo*/*bar %0Aselect 1,2,current_user

…

div 1 union **%23foofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoofoo*/*bar%0A**select 1,2,current_user

Time-to-Hack Metrics

Time-to-Hack Metric	Speed Hacking	Filter Evasion
Avg. # of Requests	170	433
Shortest # of Requests	36	118
Avg. Duration (Time)	5 hrs 23 mins	72 hrs
Shortest Duration (Time)	46 mins	10 hrs

Filter Evasion Conclusions

- Blacklist filtering will only slow down determined attackers
- Attackers need to try ***many permutations*** to identify a working filter evasion
- The OWASP ModSecurity Core Rules Set's blacklists SQLi signatures ***caught several hundred*** attempts before an evasion was found

Questions

- How can we use this methodology to our advantage?
- What detection technique can we use other than regular expressions?

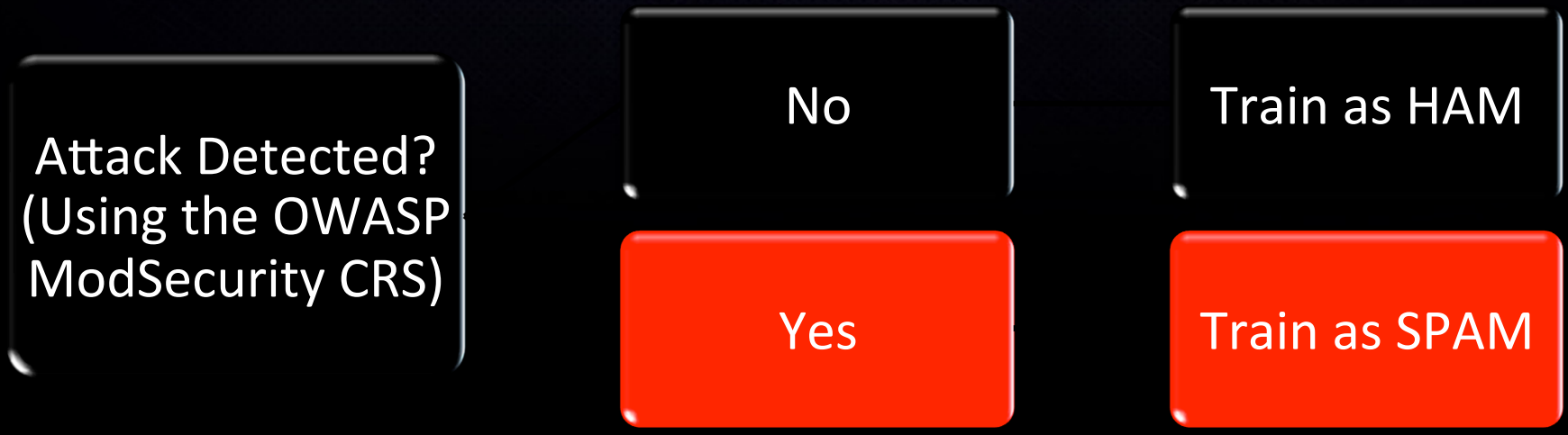
Bayesian Analysis for HTTP

- RegEx detection is binary
  - The operator either matched or it didn't
  - Need a method of detecting ***attack probability***
- Bayesian analysis has achieved great results in Anti-SPAM efforts for email
- Can't we use the same detection logic for HTTP data?
  - Data Source
    - Email – OS level text files
    - HTTP – text taken directly from HTTP transaction
  - Data Format
    - Email – Mime headers + Email body
    - HTTP – URI + Request Headers + Parameters
  - Data Classification
    - Non-malicious HTTP request = HAM
    - HTTP Attack payloads = SPAM

OSBF-Lua

- OSBF-Lua by Fidelis Assis
  - Orthogonal Sparse Bigrams with Confidence Factor (OSBF)
  - Uses space characters for tokenization (which means that it factors in meta-characters)
  - Very fast
  - Accurate classifiers
  - http://osbf-lua.luaforge.net/
- Moonfilter by Christian Siefkes
  - Wrapper script for OSBF
  - http://www.siefkes.net/software/moonfilter/
- Integrate with ModSecurity's Lua API

Training the OSBF Classifiers



Theory of Operation - HAM

1. Non-malicious user data does not trigger any blacklist rules
2. Lua script trains OSBF classifier that payloads are HAM

Lua: Executing script: /etc/httpd/modsecurity.d/bayes_train_ham.lua
Arg Name: ARGS:txtFirstName and Arg Value: Bob.
Arg Name: ARGS:txtLastName and Arg Value: Smith.
Arg Name: ARGS:txtSocialScurityNo and Arg Value: 123-12-9045.
Arg Name: ARGS:txtDOB and Arg Value: 1958-12-12.
Arg Name: ARGS:txtAddress and Arg Value: 123 Someplace Dr..
Arg Name: ARGS:txtCity and Arg Value: Fairfax.
Arg Name: ARGS:drpState and Arg Value: VA.
Arg Name: ARGS:txtTelephoneNo and Arg Value: 703-794-2222.
Arg Name: ARGS:txtEmail and Arg Value: bob.smith@mail.com.
Arg Name: ARGS:txtAnnualIncome and Arg Value: $90,000.
Arg Name: ARGS:drpLoanType and Arg Value: Car.
Arg Name: ARGS:sendbutton1 and Arg Value: Submit.
Low Bayesian Score: . Training payloads as non-malicious.

Theory of Operation - SPAM

1. Attacker sends malicious payloads during initial testing phase
2. Payloads are caught by our blacklist rules
3. Lua script trains OSBF classifier that payloads are SPAM

```
[Thu Nov 03 15:21:08 2011] [error] [client
72.192.214.223] ModSecurity: Warning. Pattern match
".*" at TX:981231-WEB_ATTACK/SQL_INJECTION-
ARGS:artist. [file "/etc/httpd/modsecurity.d/crs/
base_rules/modsecurity_crs_48_bayes_analysis.conf"]
[line "1"] [data "Completed Bayesian Training on
SQLi Payload: @@new
union#sqlmapsqlmapsqlmapsqlmapsqlmapsqlmapsqlmapsqlmapsqlm
apsqlmapsqlmapsqlmapsqlmapsqlmapsqlmapsqlmapsqlmapsqlmapsq
l\\x0aselect 1,2,database#sqlmap\\x0a()."] [hostname
"www.modsecurity.org"] [uri "/testphp.vulnweb.com/
artists.php"] [unique_id "VCqlxsCo8AoAADYJV3kAAAAH"]
```

Theory of Operation - Unknown

- Previous evasion payload is now caught

```
[Thu Nov 03 15:28:18 2011] [error] [client 72.192.214.223]
ModSecurity: Warning. Bayesian Analysis Alert for ARGS:artist with
payload: "@@new
union#sqlmapsqlmapsqlmapsqlmapsqlmapsqlmapsqlmapsqlmapsqlmapsqlmapsqlmapsqlm
apsqlmapsqlmapsqlmapsqlmapsqlmapsql\nselect 1,2,database#sqlmap
\n()" [file "/etc/httpd/modsecurity.d/crs/base_rules/
modsecurity_crs_48_bayes_analysis.conf"] [line "3"] [msg "Bayesian
Analysis Detects Probable SQLi Attack."] [data "Score:
{prob=0.99999999965698,probs={0.99999999965698,3.4301898614548e-10},cl
ass=\\x22/var/log/httpd/spam\\x22,pR=5.5841622861233,reinforce=true}"]
[severity "CRITICAL"] [tag "WEB_ATTACK/SQL_INJECTION"] [tag "WASCTC/
WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [tag
"PCI/6.5.2"] [hostname "www.modsecurity.org"] [uri "/
testphp.vulnweb.com/artists.php"] [unique_id
"bcjElMCo8AoAADYlSXMAAAAI"]
```

Where To Get Help?

- http://www.modsecurity.org/documentation/
  - Wiki Reference Guide
- ModSecurity Handbook (Free Getting Started Guide)
  - https://www.feistyduck.com/books/modsecurity-handbook/gettingStarted.html
- Community User Mail-list
  - http://lists.sourceforge.net/lists/listinfo/mod-security-users
- Twitter
  - http://twitter.com/modsecurity
- Bug Reporting (Jira)
  - https://www.modsecurity.org/tracker/

Thank You/Contact Info

- Ryan Barnett
  - rbarnett@trustwave.com
  - Twitter: @ryancbarnett
- ModSecurity
  - security@modsecurity.org
  - Twitter: @modsecurity