Dynamic DAST/WAF Integration

Presented by:

Ryan Barnett
Senior Security Researcher
OWASP ModSecurity CRS Leader
Ryan Barnett - Background

• Trustwave
  – Senior Security Researcher
  – Member of SpiderLabs Research
  – Surveillance Team Lead
    • IDS/IPS
    • MailMax
    • WAF
  – Web Application Defense
  – ModSecurity Project Leader

• Author
  – “Preventing Web Attacks with Apache”
    • Pearson Publishing - 2006
  – “The Web Application Defenders’ Cookbook”
    • Wiley Publishing – (Due end of 2012)
Agenda

• Virtual Patching
  – Time-to-Fix
  – Attack Surface Reduction

• DAST and WAF Comparison
  – Challenges
  – Valuable Data

• Level I Integration – DAST -> WAF
  – WAF Imports/Translates DAST Data for Virtual Patches

• Level II Integration – DAST <-> WAF
  – Full Integration between WAF/DAST
  – Reducing Time-to-Fix Metrics

• Conclusion
  – Development Plans
  – Call for participation
Target Audience: Defender/Breaker Communities

A Vision for OWASP

Outreach
Projects
StakeHolders
Focus
Support
Platform

Global Committees
Board

OWASP

https://www.owasp.org/index.php/Defenders
Defending Live Web Applications

http://www.swsec.com/resources/touchpoints/
Virtual Patching: Theory
What is Virtual Patching?

• Definition
  – A security policy enforcement layer which prevents the exploitation of a known vulnerability.

• Method
  – A reactive, remediation-oriented, tactical response that relies upon other processes to identify vulnerabilities.

• Process
  – The virtual patch logic analyzes HTTP transactions and intercepts attacks in transit so that malicious traffic never reaches the web application.

• Result
  – While application flaws still exist, attackers are unable to exploit them.
How is this different from WAF?

• There has to be a known vulnerability that you are protecting.
• With a known vulnerability, a WAF can then become more aggressive with blocking options when attacks are identified in the vulnerable location.
Strategic vs. Tactical Remediation

• Organizations need to utilize both **Strategic** and **Tactical** remediation efforts to address vulnerabilities.

• **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
  – Examples include adding in OWASP Enterprise Security API (ESAPI) components
  – Keep in mind that this takes *TIME*

• **Tactical Responses (Virtual Patching)**
  – Ownership is operations security staff
  – Focus on web applications that are *already in production* and exposed to attacks
  – *Attack Surface Reduction*
  – *Minimize the Time-to-Fix exposures*
Virtual Patching: 
Attack Surface Reduction

- Improper Input Handling (e.g. SQL Injection)
- Improper Output Handling (e.g. Cross-site Scripting)
- Information Leakage
- Logic Flaws
- Insufficient Process Validation (e.g. Cross-site Request Forgery)
- Insufficient Anti-Automation (e.g. Brute Force and Scraping Attacks)
- Session Handling Flaws (e.g. Cookie Tampering and Session Hijacking)
- Malicious Code Identification and Removal

OWASP Virtual Patching Survey 2012
DAST and WAF Comparison: 

Challenges
DAST and WAF Comparison

• Different Purposes
  – DAST - Identify vulnerabilities on live web applications
  – WAF – Prevent the exploitation of vulnerabilities within live web applications

• Different Perspectives
  – DAST – Acts as an HTTP client, sends simulated malicious requests and inspects responses
  – WAF – Acts as a middle-man and inspects requests and responses looking for signs of malicious behavior

• Different Teams
  – DAST – Information Security
  – WAF – Operational Security
DAST Challenges
DAST Challenges: Vulnerability Existence

- Black-box Scanning or dynamic testing of web applications works well to confirm the existence of vulnerabilities but not the total absence of them.
DAST Challenges:
Rules of Engagement Restrictions

• Active scanning can be “harmful” to some applications
• Rules of Engagement
  – Restrictive controls around who, what, where, when and how web applications may be actively scanned
  – Normally exclude mission-critical, sensitive systems
  – Often exclude testing subcategories such as Denial of Service or Brute Force attacks

• Result is a decreased scope of testing

DAST Challenges: Time Restrictions

- Testing is often *time restricted*
  - *Test for N days*

- Scanners perform a breadth-first traversal of a web site for links to map a site and identify areas of user input
  - These crawls are usually only a few levels deep and miss large portions of the application
  - Credentialed vs. Anonymous access
  - Unless properly configured, scanners can miss possible navigation options (pull-down, user fields) or multi-step business flows
  - Handling client-side code such as AJAX
DAST Challenges: Scan Accuracy Decay

- **Scan occurs**
- **Accuracy decay**

$t$ - Coherence time due to application changes
Two Biggest Questions: DAST

• When should I scan?
  – Have I scanned the entire site?
  – When are there new code pushes?

• What should I scan?
  – What are the URLs?
  – What are the Parameters?
  – What are the Cookies?
WAF Challenges
WAF Challenges: Lack of Vulnerability Data

• Most WAFs are run as “Attack Detection Systems”
  – Lack knowledge of vulnerability information.

• Many vulnerabilities can not be identified passively
  – You must act as a client and send stimulus and review responses to confirm vulnerabilities
WAF Challenges: Alert Prioritization and Blocking

• Security analysts have a difficult time with security event prioritization.
  – App without any SQL Injection Vulnerabilities
    • SQL Injection Alert for Site A -> URL1 -> Param:foo = Notice
  – App with confirmed SQL Injection Vulnerabilities
    • SQL Injection Alert for Site B -> URL2 -> Param:bar = Critical

• Users are hesitant to utilize disruptive actions without confirmation of a vulnerability
  – Fear of false positive blocking causing business disruption
Biggest Questions: WAF

• What are the vulnerabilities?
  – What vulnerability type?

• What are the injection points?
  – What are the URLs?
  – What are the Parameters?
  – What are the Cookies?
DAST/WAF: Valuable Data
Valuable Data

- **DAST**
  - Vulnerability Intelligence
  - Injection Points
    - URL
    - Parameter/Cookie Name
    - Vulnerability Type (SQLi, XSS, etc...)

- **WAF**
  - Site Tree Data (URLs and Parameters)
  - Application Credentials (Cookies)
  - *Gathered from Live Application Users*

- Wouldn’t it be great if we could share data?
AppSec Wisdom from Reese’s

• Hey, you got your DAST in my WAF!
• No, you got your WAF in my DAST!
• Mmmm, Delicious!
• DAST <-> WAF Integration, two great tastes, that taste great together 😊
Level I Integration: 
\textit{DAST -> WAF}
Welcome to the Vicnum Game

The computer will think of a three digit number with unique digits. After you attempt to guess the number, the computer will tell you how many of your digits match and how many are in the right position. Keeping on submitting three digit numbers until you have guessed the computer's number.

Enter your name and the click on the PLAY button to begin playing Vicnum!

Click here to see those who may have played a perfect game.
Or here to see those who have clearly hacked the game.
Or here to see those who have hacked the game and the database.

You can search for your favorite Vicnum player by entering the player's name below and then clicking on the SEARCH button.

Vicnum Player: test

The Vicnum project was developed for educational purposes to demonstrate common web vulnerabilities. For comments
OWASP Zed Attack Proxy (ZAP)
OWASP Zed Attack Proxy (ZAP)
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 parameters 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.

  <param>player</param>
  <attack>test%27INJECTED_PARAM'INJECTED_PARAM</attack>
  --CUT--
</alertitem>
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 providing a free certified rule set for ModSecurity™ 2.x. 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™.

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.
- **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 attacks.
- **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.
- **Identification of Application Defects** - alerts on application misconfigurations.
- **Error Detection and Hiding** - Disguising error messages sent by the server.

http://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project
## Auto-Convert DAST XML to ModSecurity

<table>
<thead>
<tr>
<th>File</th>
<th>Rev.</th>
<th>Age</th>
<th>Author</th>
<th>Last log entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Directory</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>regression_tests/</td>
<td>1787</td>
<td>10 months</td>
<td>rcbarnett</td>
<td>- Created new INSTALL file outlining quick config setup - Added a new rule regre...</td>
</tr>
<tr>
<td>runAV/</td>
<td>1571</td>
<td>15 months</td>
<td>rcbarnett</td>
<td>Improvements: - Added Experimental Lua Converter script to normalize payloads. B...</td>
</tr>
<tr>
<td>README</td>
<td>1518</td>
<td>22 months</td>
<td>rcbarnett</td>
<td>Improvements: - Added CSRF Protection Ruleset which will use Content Injection t...</td>
</tr>
<tr>
<td>arachni2modsec.pl</td>
<td>1828</td>
<td>7 months</td>
<td>rcbarnett</td>
<td>- Added example script to the /util directory to convert Arachni DAST scanner ...</td>
</tr>
<tr>
<td>httpc-guardian.pl</td>
<td>1337</td>
<td>2 years</td>
<td>b1v1r</td>
<td>Move CRS to its own structure.</td>
</tr>
<tr>
<td>rules-updater-example.conf</td>
<td>1764</td>
<td>11 months</td>
<td>rcbarnett</td>
<td>- Changed Licensing from GPLv2 to Apache Software License v2 (ASLv2)</td>
</tr>
<tr>
<td>rules-updater.pl</td>
<td>1527</td>
<td>19 months</td>
<td>rcbarnett</td>
<td>Improvements: - Updated the PHPIDS filters - Updated the SQL Injection filters t...</td>
</tr>
<tr>
<td>rules-updater.pl.in</td>
<td>1518</td>
<td>22 months</td>
<td>rcbarnett</td>
<td>Improvements: - Added CSRF Protection Ruleset which will use Content Injection t...</td>
</tr>
<tr>
<td>runav.pl</td>
<td>1571</td>
<td>15 months</td>
<td>rcbarnett</td>
<td>Improvements: - Added Experimental Lua Converter script to normalize payloads. B...</td>
</tr>
<tr>
<td>zap2modsec.pl</td>
<td>1911</td>
<td>8 days</td>
<td>rcbarnett</td>
<td>- Added the zap2modsec.pl script to the /util directory which converts OWASP Z...</td>
</tr>
</tbody>
</table>

$ ./zap2modsec.pl

Flag:

-f: path to ZAP xml report file

Usage:

./zap2modsec.pl -f ./zap_report.xml
Script Usage

$ ./zap2modsec.pl -f zap-vicnum.xml
=====================================================================
Vulnerability[3] - Type: SQL Injection
Found a SQL Injection vulnerability.
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:'WASC/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}"
Denim Group’s ThreadFix App

https://code.google.com/p/threadfix/
Level II Integration:

\[ DAST \leftrightarrow WAF \]
Current Limitations

• Manual Process is slow
  – Spidering the entire site
  – Scanning the entire site
  – Exporting XML Reports
  – Converting XML Data into Virtual Patches
  – Implementing Virtual Patches
Announcing the ModSecurity SQL Injection Challenge

The ModSecurity Project Team is happy to announce our first community hacking challenge!

This is a SQL Injection and Filter Evasion Challenge. We have setup ModSecurity to proxy to the following 4 commercial vuln scanner demo sites:

- IBM (AppScan) - demo.testfire.net site
- Cenzic (HailStorm) - CrackMe Bank site
- HP (Weblnspect) - Free Bank site
- Acunetix (Acunetix) - Acuart site

Challenge Details

To successfully complete the challenge, participants must do the following:

1. Identify a SQL Injection vector within one of the demo websites listed above.
2. Successfully enumerate the following information about the database:
   - DB User(s) - provide request data.
   - DB Name(s) - provide request data.
   - Table Name(s) - provide request data.
   - Column Name(s) - provide request data.

Challenge Submission

Please send challenge submissions to security@modsecurity.org with the details from above.
## Time-to-Hack Metrics

<table>
<thead>
<tr>
<th>Time-to-Hack Metric</th>
<th>Speed Hacking</th>
<th>Filter Evasion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. # of Requests</td>
<td>170</td>
<td>433</td>
</tr>
<tr>
<td>Avg. Duration (Time)</td>
<td>5 hrs 23 mins</td>
<td>72 hrs</td>
</tr>
<tr>
<td>Shortest # of Requests</td>
<td>36</td>
<td>118</td>
</tr>
<tr>
<td>Shortest Duration (Time)</td>
<td>46 mins</td>
<td>10 hrs</td>
</tr>
</tbody>
</table>

Level II Goal: Integration/Automation

• To decrease the Time-to-Fix metrics for web application vulnerabilities.
  – We must beat the Time-to-Hack metric for attackers

• Use Automation for Integration
  – Attackers use automation – so should we!
  – Use WAF to initiate DAST scans of individual resources
  – DAST Scans Resource and generates report
  – WAF pulls report and extracts vulnerability data
  – WAF correlates vulnerability data for protection
Challenge #1: DAST Service API

• 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
  – RPC service

http://arachni-scanner.com/
# Starting the Arachni RPC Service

```bash
# arachni_rpcd --address=192.168.168.128

Arachni - Web Application Security Scanner Framework
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
```

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

```
```
Arachni RPC Lua Client

Arachni-RPC Lua Client
Simple Arachni-RPC client written in Lua, not a big deal and still under dev.

Example
See the examples/directory.

Requirements
- LuaSec -- SSL binding for Lua
- yaml -- YAML binding for Lua.

https://github.com/Arachni/arachni-rpc-lua
Example Process Flow

Vicnum

Search Results

You have requested results for Vicnum player test; but no entries were found.

Play Again

The Vicnum project was developed for educational purposes to demonstrate common web vulnerabilities. For comments please visit the OWASP project page.
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 ModSecurity 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"
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: 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
Arachni RPC Service


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


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

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. …
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\n  into the web application which is then returned to the user's browser.\nThis\n  can lead to a compromise of the client's system or serve as a pivoting point for\n  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
  cvssv2: '9.0'
  method: POST
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/vicnum5.php").
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.sqlbi_vulnerable_params}"
Malicious Client Attempts SQLi Attack

Welcome to the Vicnum Game

The computer will think of a three digit number with unique digits. After you attempt to guess the number, the computer will tell you how many of your digits match and how many are in the right position. Keeping on submitting three digit numbers until you have guessed the computer's number.

Enter your name and click on the PLAY button to begin playing Vicnum!

Click here to see those who have played a perfect game. Or here to see those who hacked the game. Or here to see those who have hacked the game and the database.

You can search for your favorite Vicnum player by entering the player's name below and then clicking on the SEARCH button.

Vicnum Player: test' or "1" < "5";--|

The Vicnum project was developed for educational purposes to demonstrate common web vulnerabilities. For comments...
ModSecurity Alerts

"(?i:((?:(|\|\|\|\xc2\xb4||\xe2\x80\x99|\xe2\x80\x98)\||\xe2\x80\x9d)\||\xe2\x80\x93)\||\xe2\x80\x9f)\|(?:::\^('|'|`|\xe2\x80\x90|\xe2\x80\x98)\|(?:::\w\|\s("")|\xe2\xb4||\xe2\x80\x99|\xe2\x80\x98))\|(?:::\w\|\s("")|\xe2\xb4||\xe2\x80\x99|\xe2\x80\x98)-]+(" or") at ARGS:player. [file "/etc/apache2/modsecurity-crs/base_rules/modsecurity_crs_41_sql_injection_attacks.conf"] [line "573"] [Id "981243"] [msg "Detects classic SQL Injection probings 2/2"] [data "" or"] [severity "CRITICAL"] [tag "WEB_ATTACK/SQILI"] [tag "WEB_ATTACK/ID"] [tag "WEB_ATTACK/LFI"] [hostname "192.168.168.128"] [uri "/vicnum/vicnum5.php"] [unique_id "T329538AQEAAA-3DtwAAAAAJ"]

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 😊
Conclusion
Development Plans/Call for Assistance

• This proof of concept will eventually be put into the OWASP ModSecurity CRS
• Need to account for Changed Resources
• Need to incorporate more vulnerability classes
  – Currently handle
    • SQL Injection
    • Cross-site Scripting
    • Directory Traversals
• Integration with other DAST tools
  – Zed Attack Proxy API
Call for Assistance

• Need more testing
  – Performance Testing against live users
• If you would like to help with testing, please contact me and I will provide you access to the Lua scripts.
ModSecurity T-Shirt Giveaway

• What was the average “Time-to-Evasion” from Level II?
• 72 hrs.
Contact/Resources

• Email
  – OWASP: ryan.barnett@owasp.org
  – Trustwave: rbarnett@trustwave.com

• Twitter
  – @ryancbarnett
  – @ModSecurity
  – @SpiderLabs

• Blog
  – http://tacticalwebappsec.blogspot.com
  – http://blog.spiderlabs.com