State of The Art:
Automated Black Box
Web Application Vulnerability Testing

Jason Bau  Elie Bursztein
Divij Gupta  John Mitchell
Background

- Web Application Vulnerability Protection
  - High incidence vulnerabilities (XSS, SQLI, …)
  - Required for standards compliance (e.g. PCI)

Data from VUPEN
Security Tools for Web Apps

- Vulnerability Detection Techniques:
  - Manual vs. Automated
  - White-Box vs. Black-Box
  - Code review, Static analysis, Pen tester
- Automated Black Box Testing
  - Cheaper? Less intrusive to workflow?
  - Accepted method of PCI compliance
Scanner 1

Simulation of the pop-up that will appear when this page is opened in a browser.
## Security Dashboard

### Device Compliance
- McAfee Secure: 100%
- PCI: 100%

### Network IP Addresses
- 0%

### Status
- Unread Alerts: 0
- Network Scans In Progress: 0
- Device Audits In Progress: 0
- Networks Pending Approval: 1

### Vulnerabilities By Severity

<table>
<thead>
<tr>
<th>Severity</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>35</td>
</tr>
<tr>
<td>Medium</td>
<td>2</td>
</tr>
<tr>
<td>Critical</td>
<td>1</td>
</tr>
</tbody>
</table>

### Recent Vulnerabilities

<table>
<thead>
<tr>
<th>Timeframe</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 Hours</td>
<td>25</td>
</tr>
<tr>
<td>1 Week</td>
<td>10</td>
</tr>
<tr>
<td>72 Hours</td>
<td>10</td>
</tr>
<tr>
<td>1 Month</td>
<td>5</td>
</tr>
</tbody>
</table>

### Device Open Ports

<table>
<thead>
<tr>
<th>Range</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>6 - 10</td>
<td>3</td>
</tr>
<tr>
<td>&gt; 20</td>
<td>1</td>
</tr>
<tr>
<td>1 - 5</td>
<td>1</td>
</tr>
<tr>
<td>11 - 20</td>
<td>2</td>
</tr>
</tbody>
</table>
Goals of Study

• What vulnerabilities are tested by scanners?

• How representative are scanner tests of in-the-wild vulnerabilities?

• What can the user expect from scanner?

• What is difficult for the scanner to detect?
Non-Goals

• Not a product ranking

• Not a benchmark of particular tools
Outline

• Vulnerability categories tested by scanners
• How prevalent are these in the wild?
• Common application results
• Custom testbed design
• Custom testbed results
  • Coverage
  • Detection
  • False Positives
Survey of Leading Products

Local Installation

- Acunetix
- IBM
- Rapid7
- N-Stalker
- Cenzic

Service

- McAfee Secure
- Qualys

>$100K total retail price
## Vulnerability Categories From Scanners

<table>
<thead>
<tr>
<th>Category</th>
<th>Example Vulnerabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross Site Scripting</td>
<td>XSS</td>
</tr>
<tr>
<td>SQL Injection</td>
<td>SQLI</td>
</tr>
</tbody>
</table>
| Cross Channel Scripting (Other forms of injection) | Arbitrary File Upload  
Remote File Inclusion  
OS command Injection |
| Session Management                          | Session Fixation and Prediction  
Authentication Bypass                                                                     |
| Cross-Site Request Forgery                  | CSRF                                                                                    |
| SSL/Server Config                           | Self-Signed Cert, HTTP Trace                                                            |
| Info Leakage                                | Temp file access, path traversal  
Error message disclosure                                                                  |
Test Vectors By Category

Test Vector Percentage Distribution

- Info leaks
- Configuration
- CSRF
- Session
- XCS
- SQLI
- XSS

Test vectors

Test Vector Percentage Distribution
Reported Vulnerabilities "In the Wild"

Data from VUPEN
Scanners vs. In-the-Wild

- Top 4 for both:
  - XSS
  - SQLI
  - XCS
  - Info Leak

- Scanners have many more info leak vectors
- Easier to write?
### Detecting Known Vulnerabilities

Vulnerabilities for previous versions of Drupal, phpBB2, and WordPress

<table>
<thead>
<tr>
<th>Category</th>
<th>Drupal 4.7.0 NVD</th>
<th>Drupal 4.7.0 Scanner</th>
<th>phpBB2 2.0.19 NVD</th>
<th>phpBB2 2.0.19 Scanner</th>
<th>Wordpress 1.5strayhorn NVD</th>
<th>Wordpress 1.5strayhorn Scanner</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSS</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>SQLI</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>XCS</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Session</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>CSRF</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Info Leak</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

**Good:** Info leak, Session (Anecdote from re-test)

**Decent:** XSS/SQLI

**Poor:** XCS, CSRF (low vector count?)
Custom Testbed for Scanners

- Vulnerabilities covering
  - OWASP Top 10
  - WASC Web Security Threat Classifications

- NIST and WASC scanner selection criteria
  - Test all of NIST recommendations
  - Test 37 of 41 capabilities listed by WASC
Our Custom Testbed

- Linux + Apache + MySQL + PHP (LAMP)

- Measure Performance
  - Test Duration / Network Traffic

- Measure Coverage
  - Links coded in various technologies (Flash, SilverLight, ...)
  - Can scanner follow link?

- Measure Vulnerability Detection Rate
  - XSS (Type 1, Type 2, Advanced)
  - SQLI (Type 1, Type 2)
  - Cross Channel Scripting
  - CSRF

- Session Management
  - Server/Crypto Config
  - Information Leak
  - Malware
Performance did not correlate well with vulnerability detection.
% Successful Link Traversals By Technology, Averaged over all Scanners
Vulnerability Detection Rate

- Malware: 0%
- Info leak: 31.2%
- Config: 32.5%
- Session: 26.5%
- SQL 2nd order: 0%
- SQL 1st order: 21.4%
- CSRF: 15%
- XCS: 20.4%
- XSS advance: 11.25%
- XSS type 2: 15%
- XSS type 1: 62.5%
XSS Testbed

• Type 1: Textbook “Reflected” Vulnerability
  • User input → page w/o sanitization

• Type 2: Textbook Stored Vulnerability
  • User input → DB → Served Page
  • Some viewable only by different user

• Advanced (all reflected)
  • Novel Tags: e.g. <object>, <prompt>
  • Novel Channels:
    • URL → $_SERVER['HTTP_SELF']
    • Filename → error msg
XSS Results

Scanner Detection Rate for XSS

- XSS adv: 11.25%
- XSS type 2: 15%
- XSS type 1: 62.5%

Anecdote about Type 2 “alert”
**SQLI Testbed**

- **Type 1:** User input → SQLI on page generation
  - Basic: `' ; --`
  - Advanced: `', LIKE, UNION`

- **Type 2:** Input → DB → SQL Query
  - Only basic cases
  - Unsanitized form input (username) → DB
    - Later used in SQL query
SQLI Results

Scanner Detection Rate for SQL injections

<table>
<thead>
<tr>
<th>Rank</th>
<th>SQL 1st</th>
<th>SQL 2nd</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>14.2</td>
<td>21.4</td>
</tr>
<tr>
<td>2nd</td>
<td>28.5</td>
<td>28.5</td>
</tr>
<tr>
<td>3rd</td>
<td>28.5</td>
<td>28.5</td>
</tr>
<tr>
<td>4th</td>
<td>14.2</td>
<td>14.2</td>
</tr>
<tr>
<td>5th</td>
<td>28.5</td>
<td>28.5</td>
</tr>
<tr>
<td>6th</td>
<td>14.2</td>
<td>14.2</td>
</tr>
<tr>
<td>7th</td>
<td>14.2</td>
<td>14.2</td>
</tr>
<tr>
<td>8th</td>
<td>14.2</td>
<td>14.2</td>
</tr>
<tr>
<td>Average</td>
<td>14.2</td>
<td>21.4</td>
</tr>
</tbody>
</table>
“Other forms of Injection” by attacker
Manipulates server or client browser
Tests:
- XPATH injection
- Malicious File Upload
- Cross-Frame Scripting
- File Includes
- Open Redirects
- Header Injection
- Flash Parameter
- SMTP Injection
**CSRF Results**

- **Post-login forms**
  - w/o hidden random token
  - with weak [0,9] token
  - with same token each time

- **JSON Hijacking**
  - Sensitive AJAX request
  - No session id sent

- **Anecdote:**
  Not checked on purpose
Session Management

• Login / form errors
  o Login form not https
  o Reg. credentials in clear
  o Autocomplete pwd field
  o Weak pwds and pwd recovery question
  o Weak reg. page CAPTCHA

• Cookie errors
  o Not HttpOnly
  o Auth tokens not https
  o Persistent Auth token value MD5 (pwd)
  o Logout fails to clear cookie
  o Path restriction to '/'
Server/Crypto Mis-Config

- **Server Mis-Config:**
  - HTTP Trace enabled
  - PHP settings allowing code includes
  - PHP img parsed as code

- **Crypto Mis-Config**
  - Self Signed Cert
  - Weak SSL Cipher
Info Leak

- SQL error message
- Username existence
- Backup files
- Comment/Path Disclosure
- Path Traversal
  - Inclusion of /etc/secret.txt
Malware Presence

- JavaScript key-logger on login page
- Malicious graphic uploaded by user
  - Directly reference-able
- No Scanner Detected
  - Because not part of PCI compliance?
False Positives

- Testbed Traps
  - alert()s as site behavior (not part of injection)
    - Scanners avoided
  - Benign (comment) region within <script> tags
    - Tripped 2 scanners (reported 1 and 13 times)

- On a testbed of ~90 confirmed vulnerabilities

- Low FP rates due to high vulnerability density in testbed?
False Positive Observations

- Scanners exist in all these categories:
  - High Detection Rate, Low False Positive Rate
  - Low Detection Rate, High False Positive Rate
  - Low Detection Rate, Low False Positive Rate

- False positive rate not indicative of detection rate
Conclusions

• No scanner was top 3 performer across all categories

• Scanners relatively good at detecting
  o Historical vulnerabilities
  o Textbook XSS and SQLI
  o Info Leak, Session, and Server/Crypto Mis-config
    ▪ Easier test vectors to write/interpret

• Can improve
  o Understanding of active content such as Flash, SL
  o CSRF, Malware, XCS
    ▪ Low test vector count → Not vendor focus?
  o Advanced (novel) forms of XSS, SQLI
    ▪ Faster reactive process
  o Stored forms of XSS, SQLI (acknowledged by a CTO)
    ▪ Better DB modeling
Thank You

Jason Bau
jbau@stanford.edu