Manual and Automated Approaches to Auditing Web Applications

Mike Shema
<mikeshema@yahoo.com>
Agenda

- What is the problem we're trying to solve?
- How to reach the right balance of manual and automated testing
- Some examples – SQL injection, vulnerability impact, authentication
- Summary
Background

- Each layer of the OSI model affects a web application's security.
  - Vulnerabilities exhibit themselves differently.
    - SSL, SQL injection, user impersonation
  - Different approaches are required to identify vulnerabilities.
    - Network architecture, server configuration, application source code
  - Wide skill sets and knowledge are needed to identify vulnerabilities.
    - Cryptography, TCP/IP, SQL, PHP, Java, .NET
Background

- Many vulnerabilities share common traits or identification techniques.
  - Common traits improve the quality of automated analysis.
    - SQL Server error codes
  - Well-defined techniques lend themselves to automation.
    - SQL injection against URL parameters
Background

• Yet automation faces significant challenges.
  – Implementation of the web application affects the automated crawler
    • Use of JavaScript
  – Multi-stage processes may be easy to model, but hard to automate.
    • E-commerce shopping carts
    • Maintaining session through time-outs or re-authentication
  – Intentional countermeasures to automation
    • Human Interactive Proofs (HIP) and CAPTCHA
Background

- So the question becomes,

  "How can I find and address the most critical vulnerabilities using limited resources?"

  - How do I know where best to apply tools?
  - How do I know where best to rely on manual testing?
Example Test Areas

- Server software
  - May not be fully patched
  - May be misconfigured

- Application engine
  - May not be fully patched
  - May be misconfigured

- Application code
  - Syntax errors
  - Semantic errors
  - Logical errors

- Example servers
  - IIS
  - Apache

- Example engines
  - Java
  - ASP.NET

- Example vulnerabilities
  - SQL injection
  - User impersonation
  - Privilege escalation
Classifying Scanners

- Two possible approaches to classifying an automated scanner are based on...
  - Ability to interact with a web application
  - Types of tests, or number of tests, it can perform
- We'll consider scanners based on their ability to interact and understand a web application.
  - It's more important to have fewer, more accurate tests against the complete application.
  - Canned tests are misleading because they may not apply to your environment.
Manual & Automated Testing

- Manual + Scripts
- Manual + Paros
- 1st Gen. Scanners
- Web, HTTP Fuzzers
- 2nd Gen. Scanners
- (3rd Gen. Scanners)
- Logical
- Semantic
- Syntax
- Syntax (reported vulns)

Accuracy and Depth vs. Efficiency
Manual & Automated Testing

• 1\textsuperscript{st} Generation Scanners
  - Require manual configuration for forms, authentication
  - Limited understanding of customized errors
  - Cannot parse dynamic JavaScript

• 2\textsuperscript{nd} Generation Scanners
  - Automated form manipulation, authentication
  - Adapt to customized error pages
  - Parse some dynamic JavaScript
Manual & Automated Testing

• 3rd Generation Scanners
  - Deductive identification of parameter delimiters and URL construction
    • URLs that don't use normal “page?a=2&b=2” construction
  - Automatically identify and test transactional processes.
  - Textual understanding of site content.
  - Parse dynamic JavaScript, including AJAX-style usage.
Common Challenges

- JavaScript and Dynamic HTML
  - `<a href=void() onFoo=action()>`
- Multi-domain sites
- Multi-factor authentication
- Transactional processes
  - Shopping carts
  - Checkout process
  - Sequences of forms
Common Challenges

- Valid input prerequisites in step A in order to find vulnerability in step B
  - if (IsValid(nZipCode))
    {
      vulnerable_function(param)
    }
Web Applications Are Different

• Similar vulnerabilities may exhibit dissimilar properties due to error handling and underlying technologies.
  – SQL injection
  – XSS

• Vulnerabilities typically can't be mapped to vendor patches
  – Need to start with techniques, not exploits
  – Involves fewer signatures, more behavior-based analysis
Web Applications Are Different

• Audit checklists require effective vulnerability classifications.
  - Speak a common language between auditors.
  - Provide fundamental remediation recommendations.

• WASC Categories  
  (http://www.webappsec.org/)
  - Establish a common nomenclature
  - Classed by how attacks are executed
  - Provides guidance on secure web development
Threat Models

- Threat models look at the impact of a vulnerability.
  - Help to understand the application
  - Identify where to look for vulnerabilities
  - Identify architectural problems

- Good threat models can also help determine where to run automated tools and where to focus manual tests.
Threat Models

- A person defines the threat model
- A tool can only imitate attacks or a few specific threats.
  - Cross-site scripting
  - SQL injection
  - Generic parameter manipulation
  - Brute force password guessing
  - “Known” vulnerabilities based on default file names or default file locations
Threat Models

- Automated tools identify vulnerabilities, not threats.
  - SQL injection is a vulnerability that can be exploited to execute different threats.
  - SQL injection can be conducted against any parameter that the web application manipulates, not just GET or POST data.
- For example, a threat might be the ability of one person to impersonate a different account.
<table>
<thead>
<tr>
<th>Sites</th>
<th>Request</th>
<th>Response</th>
<th>Trap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sites</td>
<td>GET <a href="http://10.0.1.3/phpBB-1.0.0/bb_profile.php?mode=edit">http://10.0.1.3/phpBB-1.0.0/bb_profile.php?mode=edit</a> HTTP/1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, <em>/</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Referer: <a href="http://10.0.1.3/phpBB-1.0.0/prefs.php">http://10.0.1.3/phpBB-1.0.0/prefs.php</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Accept-Language: en-us</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322) Paros/3.2.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Host: 10.0.1.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Proxy-Connection: Keep-Alive</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cookie: LastVisit=2005-09-26+21%3A46; phpBBsession=1191402003'</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Content-length: 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>History</th>
<th>Spider</th>
<th>Alerts</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>413 POST <a href="http://10.0.1.3/phpBB-1.0.0/bb_profile.php">http://10.0.1.3/phpBB-1.0.0/bb_profile.php</a> HTTP/1.1 =&gt; HTTP/1.1 200 OK [9.043 s]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>414 GET <a href="http://10.0.1.3/phpBB-1.0.0/prefs.php">http://10.0.1.3/phpBB-1.0.0/prefs.php</a> HTTP/1.1 =&gt; HTTP/1.1 200 OK [9.053 s]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>415 POST <a href="http://10.0.1.3/phpBB-1.0.0/prefs.php">http://10.0.1.3/phpBB-1.0.0/prefs.php</a> HTTP/1.1 =&gt; HTTP/1.1 200 OK [9.053 s]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>416 GET <a href="http://10.0.1.3/phpBB-1.0.0/bb_profile.php?mode=edit">http://10.0.1.3/phpBB-1.0.0/bb_profile.php?mode=edit</a> HTTP/1.1 =&gt; HTTP/1.1 200 OK [9.043 s]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>417 GET <a href="http://10.0.1.3/phpBB-1.0.0/bb_profile.php?mode=edit">http://10.0.1.3/phpBB-1.0.0/bb_profile.php?mode=edit</a> HTTP/1.1 =&gt; HTTP/1.1 200 OK [9.043 s]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HTTP/1.1 200 OK
Date: Tue, 27 Sep 2005 04:47:29 GMT
Server: Apache/2.0.53 (Unix) PHP/4.3.11 mod_perl/1.999.23 Perl/M5.8.6
X-Powered-By: PHP/4.3.11
Content-Length: 261
Content-Type: text/html

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ')' AND (start_time > 1127792849) AND (remote_ip = '10.0.1.4') at line 1<br>
Error doing DB query in get_userid_from_session()

413 POST http://10.0.1.3/phpBB-1.0.0/bb_profile.php HTTP/1.1 => HTTP/1.1 200 OK [9.043 s]
414 GET http://10.0.1.3/phpBB-1.0.0/prefs.php HTTP/1.1 => HTTP/1.1 200 OK [9.053 s]
415 POST http://10.0.1.3/phpBB-1.0.0/prefs.php HTTP/1.1 => HTTP/1.1 200 OK [9.053 s]
416 GET http://10.0.1.3/phpBB-1.0.0/bb_profile.php?mode=edit HTTP/1.1 => HTTP/1.1 200 OK [9.043 s]
417 GET http://10.0.1.3/phpBB-1.0.0/bb_profile.php?mode=edit HTTP/1.1 => HTTP/1.1 200 OK [9.043 s]
SQL Injection Exploit

```sql
phpBBsession=1)%20OR%20user_id=1%20limit%201/*
SELECT user_id FROM session WHERE (sess_id = $sessid) AND (start_time..."

- Alternate $sessid payloads
  - 1) OR user_id=1 limit 1/*
  - 1) OR user_id=2 limit 1/*
  - 1) OR user_id=3 limit 1/*
Threat Models

- Threat models guide where to apply tools.
- For example, consider the threat of an attacker who obtains someone's password.
  - No tool will have a “steal password” test.
  - You must identify the threat in order to identify the relevant tests.
Example

● A few possible avenues of attack.
  - A cross-site scripting (XSS) attack from within the application steals the user's password.
    • A tool can test for common XSS payloads.
  - The attacker sniffs the password in transit
    • A tool can test if SSL is required during authentication and if the SSL encryption can be downgraded
  - The attacker uses a “phishing” attack via e-mail that does not involve the application.
    • A tool won't be able to test this scenario. Social engineering attacks are difficult to measure.
Example

POST /login.cgi
MemberName=mike&Password=intheclear

POST /login.cgi
MemberName=mike&Password=563b21c9be06f2141e162c1c0cc5e7d1

POST /login.cgi
MemberName=mike&Password=c8ad9bb304b288b58c0625586ccd5169&Challenge=abJruiLaP
Example

- Password at rest
  - Encrypted hash in the database

- Password in transit
  - Encrypted hash
    - SHA1('foo')
  - Encrypted hash + salt
    - SHA1('salt' + 'foo')
  - Encrypted hash + challenge
    - SHA1(SHA1(challenge) + 'foo')
Comprehensive Tests

• All tools generate a report, but someone must interpret each vulnerability.
  - Is it a false positive?
  - What is the immediate impact as tested?
  - Can the vulnerability be further exploited? Then what would be the impact?

• Does the tool provide any aid with resolving these questions?
Benefits of Automation

• Improves efficiency for repetitive tasks.
  – Oriented towards vulnerability checklists, e.g. “test each parameter for SQL injection”

• Produce more constant results
  – Less variance in quality than different auditors
  – Able to test the 100\textsuperscript{th} parameter as easily as the 1\textsuperscript{st}.

• Concise reports help focus on problem areas.
Benefits of Automation

• Tools are effective for aiding asset inventory.
  - Discover web applications on a network, especially for large institutions.
  - Provide simple description or summary of a web application.
Caveats of Automation

- False positives
  - Reduces trust in the tool
  - Creates additional, unnecessary work
- False negatives
  - How do you know what it doesn't find?
- Insufficient tests
  - Use of encryption
  - Session handling
  - Authorization and privilege levels
Benefits of Manual Audit

- Threat models are generated through a manual process.
  - Automated tools produce vulnerabilities known to the tool, not vulnerabilities related to the application's threats.
  - More manual interaction and understanding of the application leads to more complete security.

- Manual audits address more complex applications and more complex vulnerabilities.
Questions?
Thank you!
Resources

- WASC – http://www.webappsec.org/
- Web Hacking Tools – http://www.portswigger.net/
- Web Hacking Proxy – http://www.parosproxy.org/
- Python httpplib, urllib2 – http://www.python.org/