Smashing Web Apps
Applying Fuzzing to Web Applications and Web Services

Michael Sutton, Security Evangelist
Overview

• Background
  – Vulnerability discovery methodologies
  – What is fuzzing?

• Web application fuzzing
  – Challenges
  – Inputs
  – Detection

• Web 2.0 fuzzing

• Fuzzing with Google

• Conclusions
Whitebox vs. Blackbox

Whitebox Testing
- Internal perspective
- Static analysis
- Manual or automated testing
  - Insecure programming practices
  - Improper input validation

Blackbox Testing
- External perspective
- Run-time analysis
- Manual or automated testing
  - Known vulnerabilities
  - Unknown vulnerabilities
# Vulnerability Discovery Methodologies

<table>
<thead>
<tr>
<th></th>
<th>Source Code Analysis</th>
<th>Binary Auditing</th>
<th>Security Audit</th>
<th>Fuzzing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Coverage</td>
<td>● ●</td>
<td>● ●</td>
<td>● ●</td>
<td>● ●</td>
</tr>
<tr>
<td>Speed</td>
<td>● ●</td>
<td>● ●</td>
<td>● ●</td>
<td>● ●</td>
</tr>
<tr>
<td>False Positives</td>
<td>● ●</td>
<td>● ●</td>
<td>● ●</td>
<td>● ●</td>
</tr>
<tr>
<td>False Negatives</td>
<td>● ●</td>
<td>● ●</td>
<td>● ●</td>
<td>● ●</td>
</tr>
<tr>
<td>Complex Vulns.</td>
<td>● ●</td>
<td>● ●</td>
<td>● ●</td>
<td>● ●</td>
</tr>
</tbody>
</table>

Verdict - There is no silver bullet.
A Brief History of Fuzzing

1989
Professor Barton Miller Uses Fuzzing to Test Robustness of UNIX Apps at UW-Madison

1999
Oulu University Begins Work on PROTOS Test Suites

2000
Oulu University Releases PROTOS SNMP Test Suite 2002

2002
SPIKE Demonstrated at BH USA by Dave Aitel

2005
Commercial Fuzzers (i.e., Codenomicon, Mu Security, etc.) Introduced

2004
Mangleme Released by Michal Zalewski (aka lcmtuf)
Fuzzing Approaches

1. Test cases
   - Hard coded data packets or files
     ✓ Broad coverage of studied protocols
     ✗ Time consuming to develop
     ✗ Impractical for custom applications

2. Brute force fuzzing
   - All possible values attempted
     ✓ Minimal preparation
     ✓ Broad coverage of targeted inputs
     ✗ Many wasted CPU cycles

3. Intelligent fuzzing
   - Dynamically generated input adhering to predefined constraints
     ✓ Decreased false negatives
     ✗ Time consuming to develop rules

Examples

- PROTOS Test Suites
- FileFuzz
- SPIKE
Fuzzing Phases

1. Identify Target
2. Identify Inputs
3. Generate Fuzzed Data
4. Execute Fuzzed Data
5. Monitor for Exceptions
6. Determine Exploitability
## Network vs. Web App Fuzzing

<table>
<thead>
<tr>
<th></th>
<th>Network</th>
<th>Web Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability of tools</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Protocol structure</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Identifying inputs</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Detecting exceptions</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Code coverage</td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>
Web App Fuzzing - Challenges

• Multi-layered technology
  – Web server, application server, database server, etc.
    • Where does the vulnerability lie?

• Network latency
  – Network creates a bottle neck
    • How can we speed up the process?

• Exception detection
  – Numerous signals must be monitored/reviewed
    • Did we miss anything?

• Code coverage
  – Tracking business logic reached
    • How do we know when to stop?
Web App Fuzzing - Inputs

• Request-URI
  – /[path]/[page].[extension]?[name]=[value] & [name]=[value]

• Protocol
  – HTTP/[major].[minor]

• Headers
  – [Header name]: [Header value]

• Post Data
  – [Name1]=[Value1] & [Name2]=[Value2]

• Cookies
  – Cookie: [Name1]=[Value1]; [Name2]=[Value2] ...

Think Outside the Box
Input – Request-URI

/[path]/[page].[extension]?[name]=[value] & [name]=[value]

• Path
  – Path traversal

• Page
  – Predictable resource location
  – Directory indexing
  – Information leakage

• Extension
  – Web filter bypass
  – DoS

• Name
  – Abuse of functionality (hidden functionality)

• Value
  – SQL injection, XSS, file inclusion, command injection, etc.

• Separator
  – Content spoofing (URI obfuscation)
Input – Protocol

HTTP/[major]. [minor]

- Fuzz variables
  - Unsupported protocol version
    - HTTP 1.1 (RFC 2616)
    - HTTP 1.0 (RFC 1945)
    - HTTP 0.9 (Deprecated)
  - Non-RFC compliant values
    - HTTP X.Y
    - HTTP 2.2
    - AAAAA

- Proxy issues
  - Request may altered/blocked by ‘non-transparent’ proxies
    - RFC 2145 - Use and Interpretation of HTTP Version Numbers
Input – Headers

[Header name]: [Header value]

- Buffer Overflow
  - Content-Length
  - User-Agent
  - Accept Language
  - Referer

- DoS
  - Host

- Script/Code Injection
  - User-Agent
  - Referer

- SQL Injection
  - User-Agent
Input – Post Data

[Name1]=[Value1] & [Name2]=[Value2]

• Name
  – Abuse of functionality (hidden functionality)

• Value
  – SQL injection
  – XSS
  – File inclusion
  – Command injection
  – Buffer Overflows
Case Study – Buffer Overflow

Linksys WRT54G Router Remote Admin apply.cgi Buffer Overflow

• CVE-2005-2799

• Exploit

  POST /apply.cgi HTTP/1.1
  Host: 192.168.1.1

  ...

  A x 10000+

• Notes
  – Buffer overflows rare for web applications
  – Fuzzing web applications also tests underlying technologies
Input – Cookies

Cookie: [Name1]=[Value1]; [Name2]=[Value2] ...

• Name
• Value
  – Cross Site Request Forgery (CSRF)
  – Credential/session prediction
  – Insufficient authentication
  – Insufficient session expiration
  – SQL Injection
  – XSS
Case Study – Buffer Overflow

MyBB Index.PHP Referrer Cookie SQL Injection Vulnerability

• BID 16443

• Exploit
  
  GET /index.php HTTP/1.1
  Host: example.com

  ...

  **Cookie:** referer=99999999999'UNION'20SELECT'20password,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5

• Notes
  
  – Name/value pairs in cookies are often used to transfer values in the same way that they are used in GET/POST requests
Web App Fuzzing - Detection

- HTTP Status codes
  - 200 OK – predictable resource location
  - 403 Forbidden – Restricted page
  - 500 Internal server error – Unhandled exception
- Web server error messages
  - Verbose SQL error messages
  - Information leakage
- Dropped connections
- Log files
- Event Logs
- Debuggers
Web App Fuzzing - Tools

• Open Source
  – WebFuzz
    • michaelssutton.net/download/WebFuzz.zip
  – SPIKE Proxy
    • www.immunitysec.com/resources-freesoftware.shtml
  – OWASP WebScarab
    • www.owasp.org/index.php/Category:OWASP_WebScarab_Project

• Commercial
  – SPI Fuzzer
    • Included with SPIDynamics WebInspect
What is Web 2.0?
- "Web 2.0 is the business revolution in the computer industry caused by the move to the internet as platform, and an attempt to understand the rules for success on that new platform. Chief among those rules is this: Build applications that harness network effects to get better the more that people use them."
  - Tom O’Reilly

Web 2.0 vs. Web 1.0
- Same vulnerabilities
  - Additional input vectors
- More complexity
Web Services Fuzzing

Service Provider
- SOAP
- WSDL

Publish

Discovery Agencies
- UDDI
- DISCO
- Etc.

Find

Service Requestor

Interact
Web Services Fuzzing - Challenges

- Inputs
  - XML parsing and generation
  - Documented vs. undocumented
    - WSDL (Web Services Description Language)
- Targets
  - UDDI (Universal Description, Discovery and Integration)
    - OASIS
  - DISCO (Discovery of Web Services)
    - Microsoft
- Protocol
  - SOAP
    - exchanging XML-based messages over HTTP
Web Services Fuzzing - Inputs

- Identify Targets
  - UDDI
  - DISCO
  - Etc.

- Identify Inputs - WSDL
  - Blueprint for expected inputs
    - Data types (i.e. integer)
    - Data ranges (i.e. 1-1000)
  - Facilitates intelligent fuzzing
    - Generate fuzz variables outside of expected inputs
Web Services Fuzzing – Inputs - WSDL

http://api.google.com/GoogleSearch.wsdl

<message name="doGoogleSearch">
  <part name="key" type="xsd:string"/>
  <part name="q" type="xsd:string"/>
  <part name="start" type="xsd:int"/>
  <part name="maxResults" type="xsd:int"/>
  <part name="filter" type="xsd:boolean"/>
  <part name="restrict" type="xsd:string"/>
  <part name="safeSearch" type="xsd:boolean"/>
  <part name="lr" type="xsd:string"/>
  <part name="ie" type="xsd:string"/>
  <part name="oe" type="xsd:string"/>
</message>

...
Web Services Fuzzing - Tools

• Open Source
  – OWASP WSFuzzer
    • http://www.neurofuzz.com/modules/software/wsfsfuzer.php

• Commercial
  – SPI Dynamics WebInspect
AJAX Fuzzing

classic web application model (synchronous)

client
user activity

data transmission

user activity

data transmission

system processing

server

data transmission

system processing

server

time

user activity

data transmission

user activity

data transmission

user activity

data transmission

user activity
AJAX Fuzzing - Challenges

- AJAX frameworks may employ alternate data interchange formats
  - JSON - Atlas
  - Serialized Java - Google Web Toolkit
  - HTML
  - XML
- Business logic dispersed between client and server side code
- Business logic dispersed among many client side pages and script files
- Increased attack surface
AJAX Fuzzing - Implementations

- Multiple frameworks
  - Prototype (http://www.prototypejs.org/)
  - Script.aculo.us
  - Dojo (http://dojotoolkit.org/)
  - ASP.Net AJAX (http://ajax.asp.net/)
  - Etc.

- Multiple browser objects
  - Internet Explorer
    - IE6 - XMLHTTP ActiveX control
    - IE7 – XMLHTTP native script object
  - Firefox
    - XMLHttpRequest object
AJAX Fuzzing - Inputs

• Dynamic analysis (e.g. FireBug)
  – Allows for targeted fuzzing
  – No setup required

• Static analysis (e.g. spider/grep)
  – Spider website and grep for XHR calls
  – Challenging as logic for XHR is often spread among >1 web page or JavaScript file
    • Web page
      – <script src="ajax" type="text/javascript"></script>
      – Ajax.Request()

• Script page
How Not to Implement AJAX - BlinkList
select usertag name from usertag where usertag userid = order by usertag name
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 'order by usertag.name' at line 1

Warning: implode() [function.implode]: Bad arguments in /home/blinklis/public_html/Userpage/Startpage/getmytag.ax.php on line 13
BlinkList XMLHttpRequests

- Verbose SQL errors
  - Multiple
- XSS
- Exposed functionality
  - Web based email
- Directory browsing
Fuzzing Using Google Gets Low hanging fruit Easily
A Russian hackers broke into a Rhode Island government Web site and allegedly stole credit card data from individuals who have done business online with state agencies.

The story was first reported by The Providence Journal this morning and comes two days after state and local government officials released national surveys indicating they need more cybersecurity guidance and help in strengthening their systems.
Fugle Fuzzing Phases

- Identify Target
- Identify Inputs
- Generate Fuzzed Data
- Execute Fuzzed Data
- Monitor for Exceptions
- Determine Exploitability
## Fuggle vs. Google Hacking

<table>
<thead>
<tr>
<th>Fuggle</th>
<th>Google Hacking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus on input</td>
<td>Focus on output</td>
</tr>
<tr>
<td><em>e.g. URI parameters</em></td>
<td><em>e.g. page content</em></td>
</tr>
<tr>
<td>Identifying targets for further testing</td>
<td>Identifying pages using vulnerable 3rd party apps or leaking confidential information</td>
</tr>
<tr>
<td>Flexible search terms</td>
<td>Fixed signature based searches</td>
</tr>
<tr>
<td><em>e.g. inurl:&quot;id=10&quot;</em></td>
<td><em>e.g. intitle:index.of &quot;parent directory&quot;</em></td>
</tr>
<tr>
<td>Custom vulnerabilities</td>
<td>Known vulnerabilities</td>
</tr>
</tbody>
</table>
Fuggle Prerequisites

• Vulnerabilities
  – Input vectors must be indexed by Google and accessible via search operators
    ✓ Title
    ✓ Displayed page content
    ✓ URI
    ✗ Request/response headers
    ✗ Page source code
  – Effectively limits using Fuggle to pages using GET method
    • Input vectors indexed in URL
Fuggle Threat

• How can Fuggle be abused?
  • Indiscriminate web application hacking
  • Vulnerability scanning for self propagating worms / web application worms
**Fuggle SQL Injection – Identify Input**

- **Input**
  - User supplied values concatenated into SQL queries

```
www.example.com?id=10

SELECT product from products WHERE id=10;
```

- **Goal**
  - Identify pages with verbose SQL errors
**Fuggle SQL Injection – Identify Targets**

- **Search Term**
  - `inurl:"id=10"`

- **Targets**
  - Retail stores
    - E.g. Product catalog
  - Informational sites
    - E.g. News archive

- **Search results**
  - Results 1 - 10 of about **2,010,000** for `inurl:"id=10"`. (0.05 seconds)

- **Cleanse results**
  - Remove URLs w/out “id=10”
  - Remove duplicate results form single domain
Fuggle SQL Injection – Generate Data

- Goal
  - Identify pages with verbose SQL errors
- Fuzz data
  - id='10"
  - Blind SQL injection
    - id=10 OR 1=1
  - Comment remainder of query
    - id='10--
  - Encode query
    - id=%2710
Fuggle SQL Injection – Execute Data

- Submit queries
- Capture responses
  - Raw response
    - Headers
    - HTML source code
  - HTML Status codes
- Associate requests with responses
- Archive for automated and manual review
Fuggle SQL Injection – Monitor Exceptions
Fuggle SQL Injection - Exploitability

- Execute additional queries
  - Confidentiality
    - SELECT
  - Integrity
    - DROP
    - INSERT
    - DELETE
  - System compromise
    - Stored procedures
    - Extended stored procedures
### Fuggle SQL Injection - Results

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial population of URLs</td>
<td>1,000</td>
</tr>
<tr>
<td>Population after removal of duplicate servers</td>
<td>732</td>
</tr>
<tr>
<td>Population after removal of failed requests</td>
<td>708</td>
</tr>
<tr>
<td>Total number of verbose SQL errors</td>
<td>80</td>
</tr>
<tr>
<td>Percentage of sample web sites potentially vulnerable to SQL injection attacks</td>
<td><strong>11.3%</strong></td>
</tr>
</tbody>
</table>
Fuggle XSS – Identify Input

• Input
  – User supplied values echoed back in displayed web page

  www.example.com?user=joe

  Welcome back <?php echo $_GET["user"]; ?>

• Goal
  – Identify pages which display unfiltered user input
Fuggle XSS – Identify Targets

• Search Terms
  – inurl:"search=xxx" intext:"search results for xxx"
  – inurl:"query=xxx" intext:"search results for xxx"
  – inurl:"q=xxx" intext:"search results for xxx"

• Targets
  – Search pages
    • Blogs
    • Video sharing
    • News

• Search results
  – Typically < 1000
  – Numerous duplicate sites

• Cleanse results
  – Remove URLs w/out "search|query|q=xxx"
  – Remove duplicate results form single domain
Fuggle XSS – Generate Data

- **Goal**
  - Identify pages echoing unfiltered user input in responses

- **Fuzz data**
  - Client side script
    - JavaScript, VBScript, EMCA Script, HTML, etc.
  - Encoded data
    - URL encoding
    - Hexadecimal encoding
    - Unicode encoding
    - US-ASCII
    - Etc.
Fuggle XSS – Execute Data

• Fuzz Variable
  – IMG tag
    • Non existent page on local web server

• Detection
  – Allows implicit ‘phone home’ capability
  – Log entry = vulnerable web page
  – HTML likely to evade ineffective input filters
IIS Web Server Log File

#Software: Microsoft Internet Information Services 5.1
#Version: 1.0
#Date: 2007-01-31 00:57:34
#Fields: time c-ip cs-method cs-uri-stem sc-status

00:57:34 127.0.0.1 GET /xss-vulnerable.com 404

- Vulnerable site dynamically concatenated into request
- Requested resource does not need to exist on local web server
  - 404 status code is just as good as 200
Fuggle XSS – Exploitability

• Reflected XSS
  – DOM based content spoofing in phishing attacks
  – Stealing session credentials and confidential data

• Persistent XSS
  – Web based worm propagation
    • October 4, 2005 – MySpace Samy worm
# Fuggle XSS - Results

<table>
<thead>
<tr>
<th>Description</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique sites identified by Google</td>
<td>288</td>
</tr>
<tr>
<td>Unique sites accessible at time of testing</td>
<td>272</td>
</tr>
<tr>
<td>Sites with confirmed XSS vulnerabilities</td>
<td>47</td>
</tr>
<tr>
<td>Percentage vulnerable</td>
<td>17.3%</td>
</tr>
</tbody>
</table>
Lessons Learned

• Vulnerable websites are everywhere
• Previously unknown vulnerabilities can easily be identified through a combination of search engine queries and basic web page requests
• Viable tactic for phishers and worms that do not discriminate when selecting victims
• Google knows that you’re vulnerable. Do you?
The future of Fuzzing

- Tools
  - Frameworks
  - Integrated test environments
  - Commercial tools
- People
  - Wider audience
  - Proactive fuzzing – the shift from offense to defense
Any Questions?

Michael Sutton
Security Evangelist
SPI Dynamics
http://portal.spidynamics.com/blogs/msutton