Tools and Techniques to automate the discovery of Zero Day Vulnerabilities
A.K.A – Fuzzing 101
Agenda

- GEEKZONE
- Overview of fuzzing techniques
- Tutorials on specific open-source fuzzers
- Demonstrations
- DIY fuzzing!
Who are we?

- Mark Rowe, Joe Moore
- IT Security Consultants and Researchers
- Pentest Limited
- Independent IT Security Consultancy
Software Security Assessment

• Information Gathering
• Decomposition of application
• Information Analysis and Planning
• Testing of application components
• Analysis and Reporting
Information Gathering

- Design documentation
- RFC’s
- Security requirement specifications
- Data flows
- Source code
- Reverse Engineering
- Informal interviews with key personnel (e.g. developers / product managers)
- Runtime analysis
- Goal is to obtain a detailed picture of the product’s composition, which technologies it uses, how it is typically deployed and how it integrates into its environment
Decomposition of an application

• Produce a list of interfaces and features
• Understand how end users and other systems interact with the application
• Identify the application’s attack surface
Information Analysis and Planning

- Develop security test scenarios (thinking like an attacker)
- Understanding of how vulnerabilities get into an application
- Threat/Risk modelling
  - Is the component security critical?
  - Ease of attack.
  - Impact.
  - Is the component or feature enabled by default?
  - Known vulnerabilities in similar products, technologies or components.
  - How potential attackers are likely to view the product.
- Prioritise based on risk
Testing of application components

- Use knowledge obtained from previous phases
- Uncover design and implementation flaws
- Regular progress meetings
- Discuss findings with developers
Analysis and Reporting

• Bug reports throughout the assessment
• Final written report
  – Details of discovered problems
  – Highlighting possible solutions
  – Prioritised issues
• Presentation
  – Senior management
  – Architects/Developers
• Or sell your 0days 😊
Which box is it in?

- White Box
- Black Box
- Grey Box
- Fuzzing complements more traditional testing
CERT statistics

- Vulnerabilities identified and cataloged
- 2000-2007

<table>
<thead>
<tr>
<th>Year</th>
<th>2000</th>
<th>2001</th>
<th>2002</th>
<th>2003</th>
<th>2004</th>
<th>2005</th>
<th>2006</th>
<th>1Q, 2007</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vulnerabilities</td>
<td>1,090</td>
<td>2,437</td>
<td>4,129</td>
<td>3,784</td>
<td>3,780</td>
<td>5,990</td>
<td>8,064</td>
<td>2,176</td>
</tr>
</tbody>
</table>
Fault Injection

- Understand how the application works
- Enumerate all inputs – “Attack Surface”
- Design tests with input that the application may struggle to handle
- Prioritise tests
What is Fuzzing?

• Sending invalid data to inputs of a program with the purpose of highlighting software defects
• Based on fault injection
• Often automated
• Barton Miller, University of Wisconsin-Madison first person credited with carrying out a rudimentary form of fuzzing (1990)
What can fuzzers discover?

- Buffer overflows
- Integer overflows
- Format string vulnerabilities
- Race condition vulnerabilities
- SQL injection
- Cross Site Scripting (XSS)
- Remote command execution
- Filesystem attacks (reverse traversal, etc)
- Information leaking vulnerabilities
- Memory/Resource exhaustion (DoS)
- Null pointer dereferences
Who uses Fuzzers?

- Security researchers (0days, exploit dev.)
- Software QA
- Developers
- Has gained in popularity over the last few years
- Vendors such as Microsoft have adopted fuzz testing as part of their SDL
What can you Fuzz?

- Network protocols
- Files
- IPC methods
- Command line arguments
- Environment variables
- APIs
- Network stacks
- Anything that uses a structured data format
Fuzzing process

• Choose your target
• Identify inputs (attack surface)
• Prioritise
• Develop fuzzer or fuzz test cases
• Supply to inputs
• Monitor for exceptions
• Determine exploitability (optional)
Deciding what to Fuzz

• You can’t test everything at once
• Need to be systematic
• Decide which areas to mutate with fuzz data
• Still relies on human expertise!
Runtime analysis

- Processes (ps, ProcessExplorer)
- Network ports (netstat, TCPView, portscanners)
- Network Sniffing (Wireshark, tcpdump)
- Proxies (Paros, WebScarab, ITR)
- Files (Filemon, lsof)
- IPC (OLEView, strace)
- Registry keys (Regmon)
- Debugging (gdb, ollydebug)
Approaches to fuzzing

• Manual
• Semi-automated
• Fully automated
Approaches to automated Fuzzing

• Generate valid inputs from scratch or work from captured inputs (e.g. RFC versus Sniffed traffic)
• Insert fuzz data to produce faulty inputs
• Random fuzzing
• Pre-generated test cases e.g. Protos
• Brute force – bit flipping, raw byte manipulation
• “Intelligent” Fuzzing
Fuzz data

- Bit flipping, random byte changes
- Varying length strings (larger than buffer)
- Large integers, zero, negative integers
- Format strings %n, %25n
- Metacharacters
- ../../../
- <script>alert('eek')</script>
- ‘ OR 1=1 etc.
- , ‘ “ ) ] } NULL
- 0x00
Block based fuzzing

- Originated from Dave Aitel, SPIKE
- Simple and flexible (not Dave! 😊 )
- Decompose protocol into length fields and data fields
- Avoids fuzz data being ignored
HTTP POST

POST /path/script.cgi HTTP/1.0
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

postcode=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA&county =cheshire
Spike

- `<block_size><data_block><block_size><data_block>…

- Data blocks and sizes (Word, Halfword, Little Endian etc.)
  - `s_block_start()`, `s_block_end()`,
  - `s_blocksize_halfword_bigendian();`
- String data
  - `s_string("Hello\r\n")`
- Binary data
  - `s_binary("41 41")`
- Fuzzing
  - `s_string_variable`
  - `s_string_repeat("A",30)`
Spike (2)

- Create .spk file
- Sending data
  - generic_send_tcp
  - generic_send_udp
- Lots of examples in audit directory
- Other useful tools like “dcedump”
- Spike Proxy for web apps
Bug Detection

- Segmentation faults
- Debuggers (can sometimes mask the presence of a bug)
- Search for core dumps
- Network port closure
- Processes restarting
- High CPU usage
- Memory usage
- Errors in error logs
- Other activity that you wouldn’t expect during normal operation
- Need to match with test case
Problems you may encounter

• Application becomes slow or unresponsive
• Encryption, checksums, compression, obfuscation
• One bug hides another bug
• Combinations of tests cause problems, single test doesn’t trigger the bug!
  – Memory depletion/leaking
  – Process exhaustion
  – Timing issues
Advantages

• Allows fast detection of exploitable security bugs, often serious
• Identify implementation errors not discovered during code reviews or other testing
• Useful when time is limited
• Reusable
• You don’t need source code
• Can make testing of complex environments easier
• Fire and Forget
Disadvantages

• Modelling complex protocols can be difficult and time consuming especially if they aren’t documented
• Maintaining state is often difficult
• Not guaranteed to expose all bugs
• Poor code coverage
• Low yield, simple faults
• Tedious to watch!
Fun Stuff!

- Putting it all together
- MS07-029 RPC DNS vulnerability
- Start with `dnscmd.exe`
Fun Stuff!
Fun Stuff!
Fun Stuff!

File | Edit | New | Sniff | Capture | Decode | Analyze | Statistics | Help

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>2.13</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>e/nmap &gt; 4030</td>
</tr>
<tr>
<td>15</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>1029 &gt; 4031</td>
</tr>
<tr>
<td>16</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>Seq=0 Ack=0</td>
</tr>
<tr>
<td>17</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>Seq=0 Ack=0</td>
</tr>
<tr>
<td>18</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>Seq=0 Ack=0</td>
</tr>
<tr>
<td>19</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>Seq=0 Ack=0</td>
</tr>
<tr>
<td>20</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>Seq=0 Ack=0</td>
</tr>
<tr>
<td>21</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>Seq=0 Ack=0</td>
</tr>
<tr>
<td>22</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>Seq=0 Ack=0</td>
</tr>
<tr>
<td>23</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>Seq=0 Ack=0</td>
</tr>
<tr>
<td>24</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>Seq=0 Ack=0</td>
</tr>
<tr>
<td>25</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>Seq=0 Ack=0</td>
</tr>
<tr>
<td>26</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>Seq=0 Ack=0</td>
</tr>
<tr>
<td>27</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>Seq=0 Ack=0</td>
</tr>
<tr>
<td>28</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>Seq=0 Ack=0</td>
</tr>
<tr>
<td>29</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>Seq=0 Ack=0</td>
</tr>
<tr>
<td>30</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>Seq=0 Ack=0</td>
</tr>
<tr>
<td>31</td>
<td>2.14</td>
<td>172.16.5.88</td>
<td>172.16.5.62</td>
<td>TCP</td>
<td>Seq=0 Ack=0</td>
</tr>
</tbody>
</table>

a frame 22 (24 bytes on wire, 24 bytes captured)
a internet protocol, src=172.16.5.1 (172.16.5.88), dst=172.16.5.1 (172.16.5.88)
a transmission control protocol, src port: 4001 (4001), dst port: 1234 (1234), seq=0, Ack=277, Len=160

Copyright © Pentest Limited 2007
EIP: 62626262!
Open Source Fuzzers

• SPIKE
• Autodafé
• PEACH
• And many, many more…
Commercial Fuzzers

- BeStorm protocol fuzzer
- OULU commercial fuzzer
- Codenomicon
- Mu-4000
- BreakingPoint
Homegrown Fuzzers

- Specific purpose
- Often quick, not very comprehensive
- Modify other (open source) fuzzers
- Not really something you could sell(!)
Autodafé

• An act of software “torture”
• Similar to SPIKE
  • Block based scripts
• File and Network fuzzing
• Monitoring tools built in
• Weighting attacks (very cool!)
Autodafé

block_begin("rmf_header");
hex(2e 52 4d 46);
block_size_b32("rmf_header"); /* chunk size */
hex(00 01); /* chunk version */
hex(00 00 00 00); /* file version (0) */
hex(00 00 00 06); /* number of headers */
block_end("rmf_header");
PEACH

• Written by Michael Eddington (IOActive)
• Python based framework
  • Cross-platform
• Can fuzz just about anything!
• Syntax and concepts needs to be learnt
• Easily re-usable code
PEACH (continued)

• Four components to a PEACH script
  • Generators
  • Transformers
  • Protocols
  • Publishers

• Sounds complicated, really isn’t!
File Fuzzing

• Targets
  – Common application formats
  – One format many targets

• Manual approach
  – Create a series of corrupted files (hex editor for binary protocols)
  – Open each file with the application
  – Very slow
  – Boring!
Automated File Fuzzing

- Binary file formats can be complicated
- In depth knowledge may be required
- Often not documented
- Makes intelligent fuzzing difficult
- Good news is dumb fuzzing often yields results 😊
- Randomly overwrite bytes or perform bit flipping
- File headers are often a good place to start
DIY fuzzing

- Modified BackTrack live-cd
- Real world example – RealPlayer 10 .smil file stack overflow
- Suggest you use SPIKEfile
- Sample .smil file in /usr/local/examples
BackTrack CD

• Should autoboot
  – Possible problems:
    • IRQ - bt irqpoll
    • PCMCIA – bt nopcmcia
    • ACPI – bt acpi=off
    • DHCP – bt nodhcp
Questions?
Useful resources

- http://www.threatmind.net/secwiki/FuzzingTools
- http://www.owasp.org/index.php/Fuzzing
- http://www.immunitysec.com/resources-freesoftware.shtml
- Fuzzing mailing list http://www.whitestar.linuxbox.org/mailman/listinfo/fuzzing