About the Speakers

Ari Takanen

- Researcher/Teacher 1998-2002 @University of Oulu OUSPG research group on Software Quality
- CEO/CTO 2001-today @Codenomicon, author of two books on VoIP Security and Fuzzing
About the Speakers

• Ian Bryant
  • **Technical Director**, *Trustworthy Software Initiative* (TSI)
    • UK’s public-private partnership for Making Software Better
    • Formerly SSDRI
  – **Visiting Lecturer**, *Cyber Security Centre, De Montfort University*
  – **Deputy Chair**, *BSI IT Security Techniques Committee - Security Controls and Services Panel*
    • IST/033/-/4 – UK shadow of ISO/IEC JTC1 SC27 WG4
How is Security Compromised?

- **Availability:**
  - A zero day attack is used to compromise a specific host

- **Integrity:**
  - Spawned processes can change anything in the system

- **Confidentiality:**
  - All data and communications can be monitored

**Target of Assurance:**
- Estimate how many vulnerabilities are in the code?
- Estimate how easy it is for a hacker to find zero-day vulnerability?
Product Security Terminology

- **Vulnerability** – a weakness in software, a bug.
- **Threat/Attack** – exploit against a specific vulnerability
- **Protocol Modeling** – functional behavior, interface message sequences and message structures
- **Anomaly** – abnormal or unexpected input
- **Failure** – crash, busy-loop, memory corruption, or other indication of a bug in software
Approaches to security testing and analysis

- Dynamic (DAST):
  - Feature/conformance testing
  - Performance/load testing
  - Security scanning
  - Robustness testing
  - Fuzzing

- Static (SAST)
  - Code Analysis
Functional security

- Security requirements and related features need “testing”
  - ETSI TVRA = Risk and Threat Analysis

- Tests against a specific conformance criteria, a set of security requirements implemented as security features
• DDoS has been a trigger for the load test industry to adapt the “security story”
• For some telecommunication players, this is the biggest threat:
  — No protection as there is always a load that will succeed in denying valid subscribers
  — Direct and measurable cost of downtime
What is Fuzzing?

• A robustness testing technique where purposefully unexpected and/or invalid input data is fed to tested system in hope to find robustness and security problems.
• Used in development (R&D), verification (test-labs), and in security audits (operations).
### Example Fuzz Test Results: TVs

<table>
<thead>
<tr>
<th>Protocol/TV</th>
<th>TV 1</th>
<th>TV 2</th>
<th>TV 3</th>
<th>TV 4</th>
<th>TV 5</th>
<th>TV 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4</td>
<td>pass</td>
<td>FAIL</td>
<td>FAIL</td>
<td>pass</td>
<td>pass</td>
<td>FAIL</td>
</tr>
<tr>
<td>DVB</td>
<td>FAIL</td>
<td>FAIL</td>
<td>FAIL</td>
<td>FAIL</td>
<td>FAIL</td>
<td>FAIL</td>
</tr>
<tr>
<td>UPnP</td>
<td>n/a</td>
<td>FAIL</td>
<td>pass</td>
<td>n/a</td>
<td>n/a</td>
<td>FAIL</td>
</tr>
<tr>
<td>Images</td>
<td>pass</td>
<td>FAIL</td>
<td>FAIL</td>
<td>n/a</td>
<td>n/a</td>
<td>FAIL</td>
</tr>
<tr>
<td>Audio</td>
<td>pass</td>
<td>pass</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>pass</td>
</tr>
<tr>
<td>Video</td>
<td>FAIL</td>
<td>FAIL</td>
<td>n/a</td>
<td>FAIL</td>
<td>FAIL</td>
<td>FAIL</td>
</tr>
</tbody>
</table>

“**FAIL**” means multiple repeatable crashes were found
“**pass**” means the system did not crash (more fuzzing needed?)
“**n/a**” means the interface did not exist, or was not tested

We did not analyze the failures for exploitability.
How Model-Based Protocol Fuzzing Works

- Protocol model defines interface functionality
- Anomaly library knows inputs that break software
- Attack simulation generates test cases
- SUT analysis monitors what happens during test execution
- Reporting creates reproduction details
“Models” and “Rules” (PROTOS 1998-2001)

- Syntax/Grammar + Semantics + Behavior

- ABNF or ASN.1
- scripting/programming
Vulnerability Details

CODENOMICON

• CVSS scoring
  • All test cases have calculated CVSS scores
  • Scoring is configurable for any environment

• CWE enumeration
  • Common Weakness Enumeration
  • Industry standard for describing software weaknesses
  • Enumeration available for all test cases

From http://cwe.mitre.org

CWE-190: Integer Overflow or Wraparound

Example 1:
The following code excerpt from OpenSSH 3.3 demonstrates an integer overflow:

C Example:

```c
mresp = packet_get_int();
if (mresp < 0) {
    response = xmalloc(mresp*sizeof(char*));
    for (i = 0; i < mresp; i++) response[i] = packet_get_string();
}
```

From http://cwe.mitre.org
## Fuzzing Effectiveness against WiFi

<table>
<thead>
<tr>
<th></th>
<th>AP1</th>
<th>AP2</th>
<th>AP3</th>
<th>AP4</th>
<th>AP5</th>
<th>AP6</th>
<th>AP7</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLAN(*)</td>
<td>INC</td>
<td>FAIL</td>
<td>INC</td>
<td>FAIL</td>
<td>N/A</td>
<td>INC</td>
<td>INC</td>
</tr>
<tr>
<td>IPv4</td>
<td>FAIL</td>
<td>PASS</td>
<td>FAIL</td>
<td>PASS</td>
<td>N/A</td>
<td>FAIL</td>
<td>INC</td>
</tr>
<tr>
<td>ARP</td>
<td>PASS</td>
<td>PASS</td>
<td>PASS</td>
<td>N/A</td>
<td>FAIL</td>
<td>PASS</td>
<td>PASS</td>
</tr>
<tr>
<td>TCP</td>
<td>N/A</td>
<td>N/A</td>
<td>FAIL</td>
<td>N/A</td>
<td>FAIL</td>
<td>PASS</td>
<td>N/A</td>
</tr>
<tr>
<td>HTTP</td>
<td>N/A</td>
<td>PASS</td>
<td>FAIL</td>
<td>PASS</td>
<td>INC</td>
<td>FAIL</td>
<td>FAIL</td>
</tr>
<tr>
<td>DHCP</td>
<td>FAIL</td>
<td>FAIL</td>
<td>INC</td>
<td>N/A</td>
<td>FAIL</td>
<td>FAIL</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Failure %:
- WLAN: 33%
- IPv4: 50%
- ARP: 16%
- TCP: 66%
- HTTP: 50%
- DHCP: 80%

http://www.codenomicon.com/labs/results
Fuzzer Smartness vs Coverage

• “Dumbest” fuzzer is doing random mutations to a template sequence (file, PCAP)
• “Smart” fuzzers use interface models and carefully selected test generation algorithms
Fuzzer Efficiency Case Study

CODENOMICON

• Most important efficiency metric for fuzzers:
  – How many bugs does it find
  – How much time does it take to find them

“Smart” model-based generational fuzzer found 10 unique bugs

Both found 2 same bugs

Mutation fuzzer found 4 unique bugs

Generation fuzzer executed for 17 hours

Mutation fuzzer took 118 hours (5 days) to execute, after which no more new bugs were found
Fuzzing in the Software Development Process

- Fuzzing is done in other parts of the SDL as well.
Mapping the LifeCycle

• This work has been done by the MTS (Methods for Testing and Specification) Security Special Interest Group (SIG)
• The SIG has been working on a context for Security Specification and Testing
  – Using a LifeCycle based on ISO/IEC 15288
    “Systems and software engineering -- System life cycle processes”
  – Mapping Specification to Enterprise Architecture Concepts
  – Mapping Testing to Assurance Concepts
• Will be used in ETSI #201581 “Security Design Guide”
Levels of Decomposition

Enterprise Architecture Level model (after Zachman)

- Conceptual
- Contextual
- Logical
- Physical
- Detailed
<table>
<thead>
<tr>
<th>Activity</th>
<th>Stakeholder Requirements Definition (SRD)</th>
<th>Requirements Analysis (REQ)</th>
<th>Architectural Design (DES)</th>
<th>Implementation (IMP)</th>
<th>Integration (INT)</th>
<th>Verification (TST)</th>
<th>Transition (TRA)</th>
<th>Validation (VAL)</th>
<th>Operation (OPE)</th>
<th>Maintenance (MAI)</th>
<th>Disposal (DIS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture Framework Level 2 (Contextual)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture Framework Level 3 (Logical)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture Framework Level 4 (Physical)</td>
<td>Design / Effect Class Selection</td>
<td>Design / Effect Pattern / Package Selection</td>
<td>Design / Effect Implementation</td>
<td>Design / Effect Integration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture Framework Level 5 (Detailed)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assurance Framework Level 1 (Conceptual)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assurance Framework Level 2 (Contextual)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assurance Framework Level 3 (Logical)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assurance Framework Level 4 (Physical)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assurance Framework Level 5 (Detailed)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ISO/IEC 15288**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Stakeholder Requirements Definition (SRD)</th>
<th>Requirements Analysis (REQ)</th>
<th>Architectural Design (DES)</th>
<th>Implementation (IMP)</th>
<th>Integration (INT)</th>
<th>Verification (TST)</th>
<th>Transition (TRA)</th>
<th>Validation (VAL)</th>
<th>Operation (OPE)</th>
<th>Maintenance (MAI)</th>
<th>Disposal (DIS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture Framework Level 2 (Contextual)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture Framework Level 3 (Logical)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture Framework Level 4 (Physical)</td>
<td>Design / Effect Class Selection</td>
<td>Design / Effect Pattern / Package Selection</td>
<td>Design / Effect Implementation</td>
<td>Design / Effect Integration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Architecture Framework Level 5 (Detailed)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Key**

- Common application for Systems and Products
- Differing application for Products and Systems
- Applies only to one of Product or Systems
Linked Activities
Speaker Contact

- Ari Takanen
  - Phone: +358 40 5067678
  - Email: art@codenomicon.com
  - Website: www.codenomicon.com

- Ian Bryant
  - Phone: +44 79 7312 1924
  - Email: ian.bryant@uk-tsi.org
  - Website: www.uk-tsi.org