CODENOMICON

Finding Critical Security Problems with Fuzzing

ITSF 2011, Nov 1-3, 2011

Heikki Kortti, Senior Security Specialist, Codenomicon

CODENOMICON

Fuzzing, Robustness Testing, Negative Testing

- Hackers are using fuzzing to find vulnerabilities
  - Found vulnerabilities are developed to exploits or used to launch DoS attacks
- As mitigation, companies have started to integrate the same security techniques
  - Fuzzing tools to automate security testing
  - Hardening devices and networks against attacks
- Not just against hacking
  - General quality improvement and preparing for unexpected
- Any software that processes inputs can be fuzzed: network interfaces, device drivers, user interface….

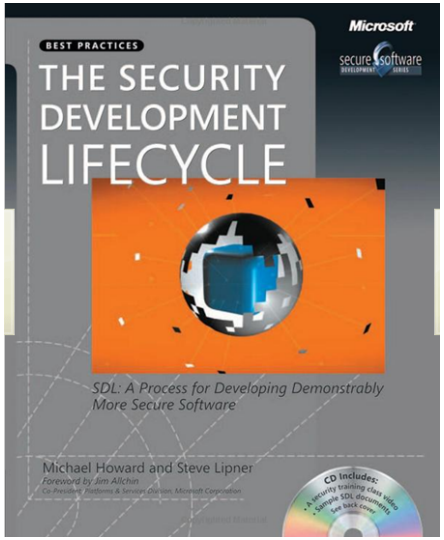
From a student course assignment…

- First documented application of fuzzing was Barton Miller's course assignment for students in 80's
- Students were asked to write programs to fuzz test Unix command line applications
- Random inputs were fed to applications, which exhibited previously hard-to-find bugs

CODENOMICON

...to a standard corporate secure development practice

Training	Requirements	Design	Implementation	Verification	Release	Response
• Core training	• Define quality gates/bug bar • Analyze security and privacy risk	• Attack surface analysis • Threat modeling	• Specify tools • Enforce banned functions • Static analysis	• Dynamic/Fuzz testing • Verify threat models/attack surface	• Response plan • Final security review • Release archive	• Response execution



Fuzzing happens in the verification phase of the SDL

CODENOMICON

The Original Definition

“Fuzz testing or fuzzing is a software testing technique that provides random data ("fuzz") to the inputs of a program. If the program fails (for example, by crashing, or by failing built-in code assertions), the defects can be noted.”

- http://en.wikipedia.org/wiki/Fuzz_testing

Original fuzzing was

Entirely random [input to DUT]

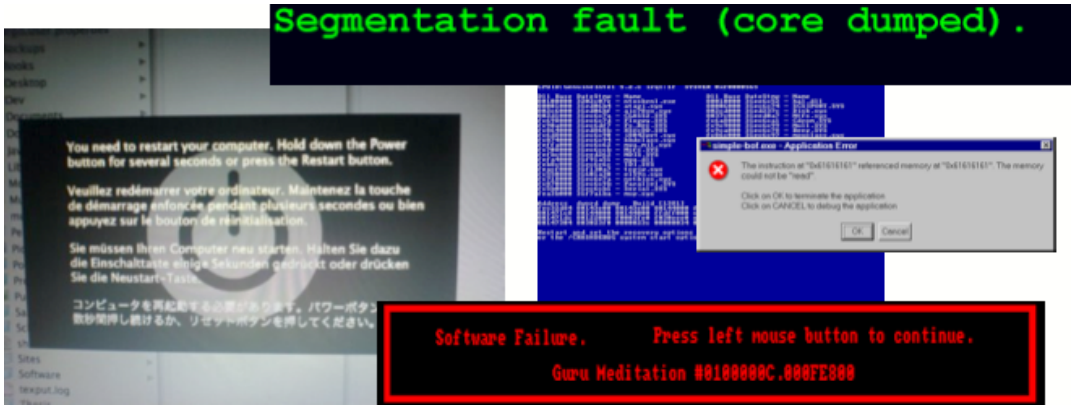
Simple pass/fail criteria: SW crash or no crash

Easy to automate due to simplicity

CODENOMICON

Next Generation Approaches

- From random to systematic and targeted
  - In fact, most fuzzing today is based on sending systematically broken (rarely random) inputs to a software, in order to crash it
- Two techniques for fuzzing:
  - Mutation (non-intelligent semi-random modifications)
  - **Generation** (intelligent and targeted model-based tests)

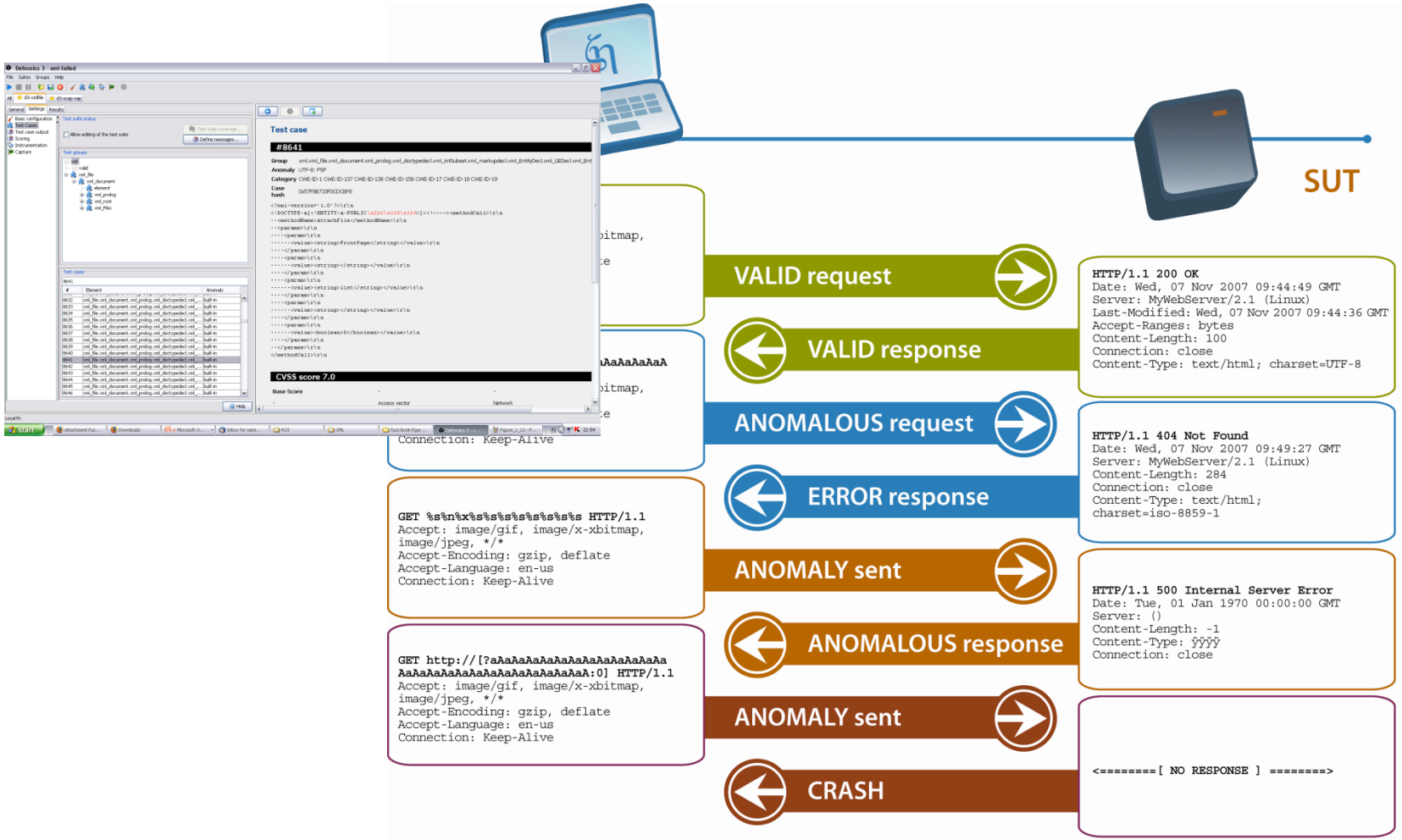


Expected Results

- The purpose of fuzzing is to find flaws
  - All found issues are true implementation errors (from a quality perspective)
  - Subset of found issues do have security implications
  - Typically, there are very few or no false positives with fuzzing
- All “stacks” or services can be vulnerable
  - Any protocol implementation can fail under negative testing
  - And based on experience, most do
- Complexity level predicts amount of flaws
  - The more complex the implementation, the more flaws there will be

CODENOMICON

Running Fuzz Tests



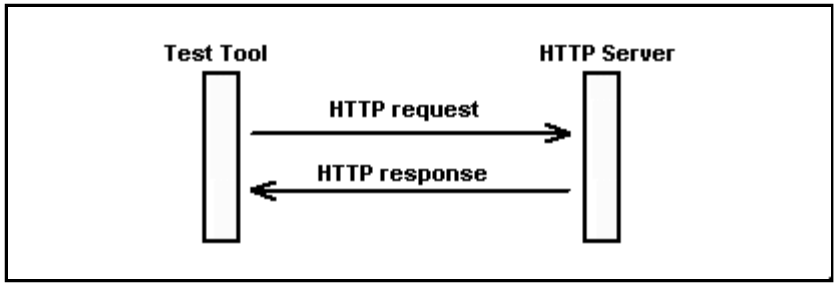
CODENOMICON

Example of a Fuzz Test Case

VALID PACKET

```
GET /index.html HTTP/1.0
User-Agent: Wget/1.11.4
Accept: */*
Host: localhost
Connection: Keep-Alive
```

SIGNALING

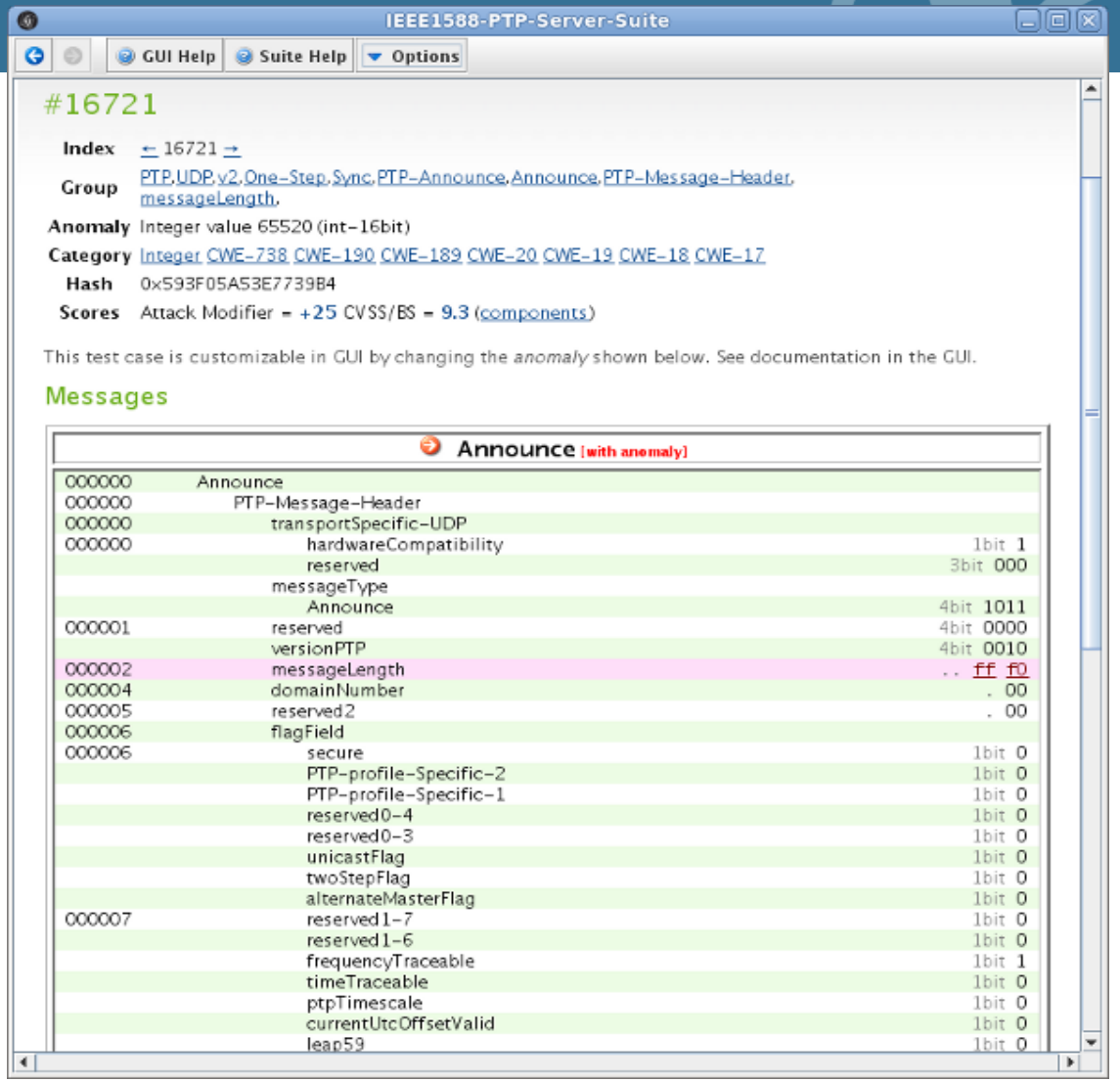


ATTACK PACKET

```
GET \x80\x81\x82\x83\x84\x80\x81\x82\x83\x84 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Connection: Keep-Alive
Host: www.example.com:80
User-Agent: Mozilla/4.0 (compatible; Codenomicon HTTP Server Test Tool)
```

CODENOMICON

Example of a Fuzz Test Case



CODENOMICON

Ethernet Testing

- Metro Ethernet is a technology operators have started deploying relatively recently
  - New technology with emerging new vendors is often prone to security vulnerabilities
- In 2008, UK's CPNI commissioned Codenomicon to assess security of several Metro Ethernet switches
  - http://www.cpni.gov.uk
  - Several new fuzzers were developed and run against the switches
  - CPNI communicated results to appropriate vendors
  - Project continued in 2009-2011

CODENOMICON

Attack Surface Analysis for L2 Devices

- Pure L2: Ethernet, Synchronous Ethernet, BFD, CFM, E-LMI (MEF-16), GARP/GMRP (802.1D), OAM/LFM (802.3ah), LLDP (802.1AB), PBT/PBB-TE (802.1ah), L2TP, LACP (802.3ad), STP (802.1D)
- Layer 3: IP, ICMP, IGMP, TCP, UDP, SCTP
- Management: SSH, Telnet, FTP, SNMP, TFTP, HTTP
- Auxiliary protocols: NTP, DHCP, DNS, PTP
- Other protocols: BGP, OSPF, RSVP, PIM, IS-IS
- Vendor-specific protocols (custom and/or proprietary)

CODENOMICON

Synchronous Ethernet & PTP Testing Scope

- Develop new fuzzing test suites for Synchronous Ethernet and PTP
- Repeat previous L2 tests against SynchE/PTP-capable devices and research robustness and security of these devices specifically
- Run higher-level L3+ tests as time and device features allow

CODENOMICON

Challenges

- Access to reference implementations during test suite development
- Logistics with test labs, vendors and network operators
- PTP telecoms profile requirements

CODENOMICON

Synchronous Ethernet Test Suite

- Can be used to test any Synchronous Ethernet device
- ESMC messages
- QL Synchronization functions
- QL TLVs
- Extension TLVs

CODENOMICON

IEEE1588 PTP Test Suites

- Can be used to test any PTP implementation
- Both “client” and “server” test suites
- PTPv1, PTPv2
- Tested messages: Announce, Sync, Follow-Up, Pdelay-Req, Delay-Req, Management, Signaling
- Unicast and Multicast modes
- Transport over UDP/UDP6 and Ethernet

CODENOMICON

Test Execution

- Ensure interoperability
- Start testing
- Monitor crashes and slowdowns
- Monitor other unexpected behaviour or protocol interactions in test network with a network analyzer
- Monitor changes in timing characteristics from system logs or external monitoring devices
- Repeat tests as required for remediation / debugging

CODENOMICON

Summary of Results

- All of the tested devices exhibited some failures
- Synchronous Ethernet caused mostly slowdowns, no crashes
- PTP implementations proved surprisingly robust
- Higher-layer protocol tests demonstrated more critical flaws
- Finding relevant R&D or security contacts within vendors was challenging
  - Except when testing was done in their premises
- “One faulty protocol is enough” - if a device can be killed with a single broken packet, it does not matter if all of the other protocol implementations in the device are robust

CODENOMICON

Generic Attack Scenarios

- Port scan crashes SUT
- SUT slows down when receiving large volume of malformed traffic
- Crash of SUT or subsystem when receiving one or more malformed packets

CODENOMICON

PTP-Specific Attack Scenarios

- Denial-of-Service attacks against GM, slave clocks, SSUs
- Most of the attacks require IP-level or other network-level access to target PTP interface
- Slave clock site could be compromised physically, attacker could plant a black box that send evil packets towards GM or other slave clocks
- Targeted malware (think Stuxnet) as an alternative
- GM DoS takes down whole network
- DoS of a remote slave clock is less critical but can be more costly to fix
- SSU attacks require direct connection from GM
- Solid network design and incident response are crucial

THANK YOU – QUESTIONS?

"Thrill to the excitement of the chase! Stalk bugs with care, methodology, and reason. Build traps for them.

....

Testers!

Break that software (as you must) and drive it to the ultimate

- but don't enjoy the programmer's pain."

[from Boris Beizer]

CODENOMICON

PROACTIVE SECURITY AND ROBUSTNESS SOLUTIONS