# Fuzzing – WhatTheFuzz?

# The story about the lazy programmer and the ugly hacker.

*The programmer's code:*

```
<?php
$age = $_REQUEST['age'];
...
?>
```

*The programmer's idea:*

*The ugly hack:*


*Repaired code:*

```
<?php
$age = $_REQUEST['age'];
if ($age > 120)
        fatal(“bad age”);
...
?>
```

☞ ☞

*Another ugly hack:*


*The bulletproof code:*

```
<?php
$age = $_REQUEST['age'];
if (($age > 120) || ($age < 0))
        fatal(“bad age”);
...
?>
```

# The first fuzzing tool

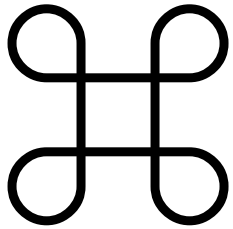
```
#!/bin/sh

i=0
while [ i -lt 1000000 ]
do
   wget  http://$1/$2?name=Really%20Me&age=$i
   wget  http://$1/$2?name=Really%20Me&age=%2d$i
    i=`expr $i + 1`
done
```

# Happy programmer

*Just (for)getting the new expierence*

- *Programmer has downloaded the "First fuzzing tool from the Internet".*
- *By this tool now he tests every page which he has developed.*
- *Does not matter if there is used the parameter $age.*
- *From now all his code is heavily tested by fuzzing.*

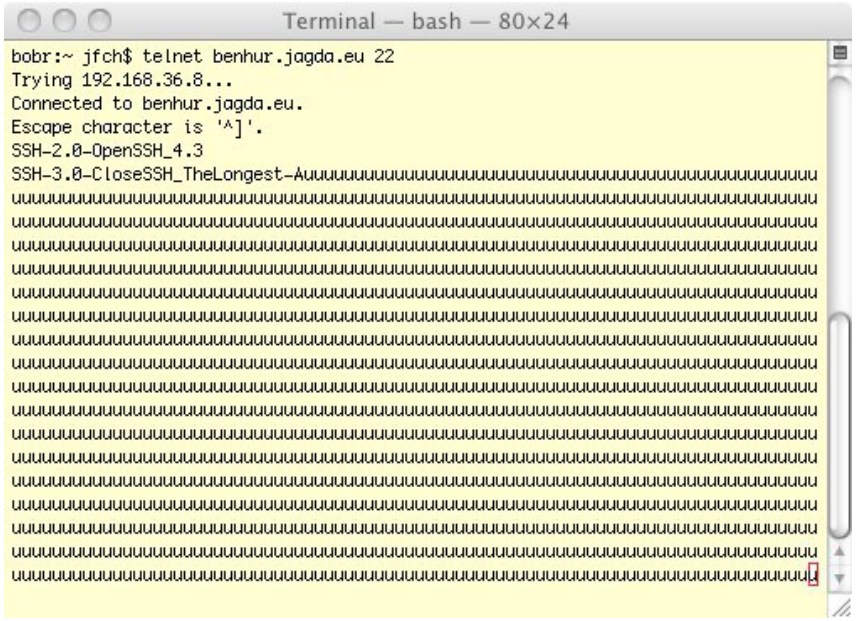


# What fuzzing is?

- Based on sending intentionally corrupted data to the application
  - Good enough to be accepted by the application
  - Corrupted enough to break it
- Evaluation of the application behavior
- Change the data corruption dependently on the previous results and the application output.
- Running on huge amount of retries

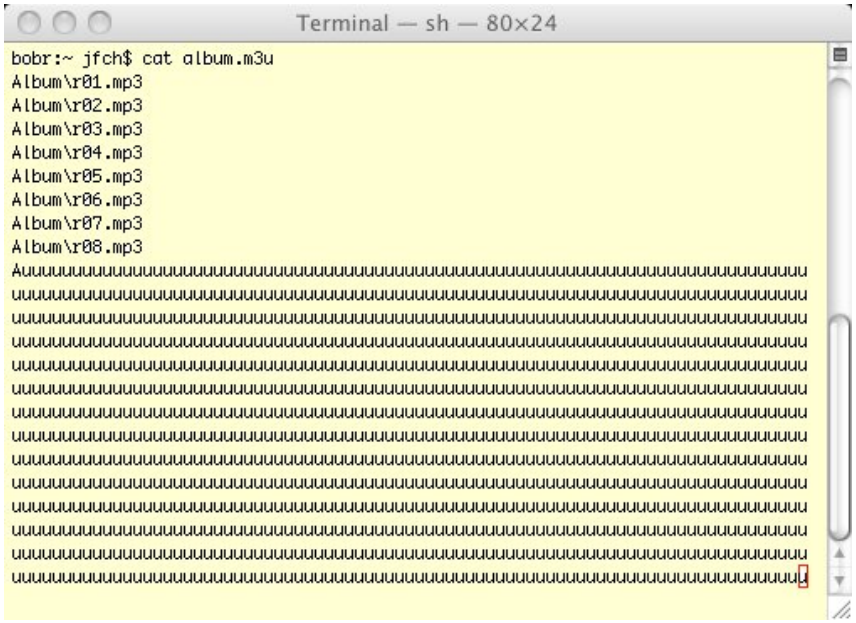
# Protocol fuzzing

- Playing with the network interface of the application.
- Level 0:
  - tcpdump (wireshark) is your best friend for obtaining patterns.
  - telnet (netcat) is the best replayer.
- Of course another levels exists.



# Data format fuzzing

- Modifying the content of the input file of the program.
- Level 0:
  - Hex or text editor is useful tool.
- Next Level:
  - The modified data may be encapsulated in the another correct format



# Other fuzzing

- Syscalls
- Library calls
- Environment variables
- File descriptors
- Signals

- TCP stack
- File system
- Http cookies
- RPC interface
- IPC

- Hardware
- Firmware

- And more

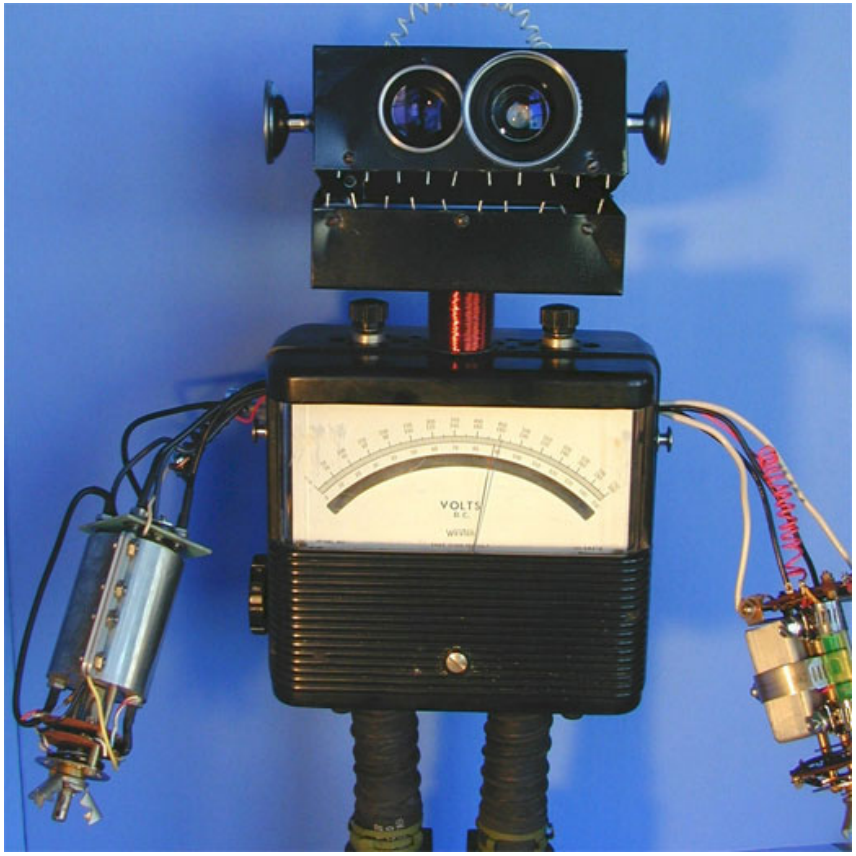
# Why fuzz?

- Just trying a cheap hack.
- Do not want to read source code or do not understand everything on it.
- Do not want disassemble or debug the binary code (if source code is unavailable)
- Investigate Black Box
  - Hardware
  - Network service

# Special case: white fuzzing

- Fuzzing based on detail knowledge of the source code.
- Used for verifying the code.
- The testing patterns are based on the known data structures.
- May supplement or replace the source code audit.
  - Cannot fully compensate the source code audit.
  - Often more effective than source code audit.

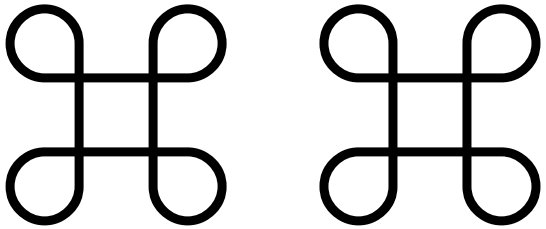
# Automated fuzzing, first generation




- Similar to brute forcing
- The tools for interception the successful fuzz are provided
  - Generating the core
  - Running debugger
  - Get tcpdump snapshot

# Automated fuzzing, next generation

- Test patterns are generated and modified automatically
  - From network snapshots
  - From known data structures
  - From complete source code



# Fuzzers

- One purpose fuzzers.
  - Only limited configuration.
- Frameworks.
  - Extensible.
  - Scriptable by some high level language.
  - Possibility to write new modules.
  - Sometimes only some templates are provided.

# Fuzz

- ftp://ftp.cs.wisc.edu/paradyn/fuzz/

- Authors: Miller, Fredriksen and So
- The first fuzzer
  - Created in 1990
  - Basically a stream generator of random characters. It produces a continuous string of characters on its standard output file.
  - Tested on ninety different utility programs on seven versions of UNIX.
  - It was able to crash more than 24% of them.

# Continued …

- Repeated in 1995
  - Tested on UNIX systems.
  - As bad as original.
- Repeated in 2001 with Windows (NT4 and 2000)
  - Even worse
  - 100% crashed programs
- Repeated in 2006 with MacOS
  - Unix personality programs crash rate 7%
  - Native MacOS applications 73%

# Protos, Genome, Radamsa

- http://www.ee.oulu.fi/research/ouspg/protos
- http://www.ee.oulu.fi/research/ouspg/genome
- http://code.google.com/p/ouspg/wiki/Radamsa

- Created at university Oulu
- Family of projects with the final project Radamsa
  - Protos written in Java, Radamsa in C
  - Generates random pieces of binary chunk.
  - Protos
    - Utilities with embeded randomness into the protocol
  - Radamsa
    - Programable output character set and distribution
    - Can be used as tcp server or client

# SPIKE

- http://www.immunitysec.com
- Autor: Dave Aitel
- Universal network fuzzing framework
  - Written in c.
  - Support for msrpc, sunrpc, ftp, smtp, http, ttp and more

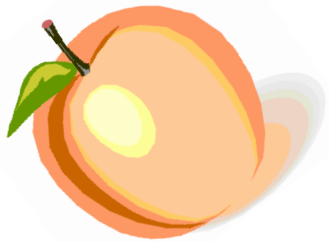
# The SPIKE scripting language

- Creating SPIKE fuzzers is actually done in C language.
- Can be mixed together with the scripting.
- The connection to the scripting is s_parse()
  - Loads the file line by line and does limited C parsing on it.
  - Not all the SPIKE api functions are supported.
    - Missing float type.
  - Uses dlopen() and dlsym() and some demarshalling to call any functions found within
- Typically a “generic” framework is built, then SPIKE script is used to quickly play with the protocol

# The SPIKE descendants

- SPIKE 2006 (part of the CANVAS framework)
  - Spike completely rewritten in python
  - Focused to DCERPC
- Smudge http://www.fuzzing.org/
- Sulley http://code.google.com/p/sulley/
  - Current general SPIKE replacement
- SPIKE proxy http://www.immunitysec.com
  - Web intercepting framework
- SPIKE file http://labs.idefense.org
  - Universal file format fuzzing framework

# Peach



- http://peachfuzzer.com
- Written by Micheal Eddington
- Smart fuzzer, network and file also.
  - Written in python.
  - Next version will be in C# + .NET
  - Configuration written in XML.
  - Include many tools for generating new fuzzers.

# Dumb fuzzers

- ProxyFuzz http://www.hacker-soft.net/tools/Defense/proxyfuzz.py.txt
  - Sits in the middle of traffic and randomly injects anomalies into live traffic
  - Completely protocol unaware
- Mangle https://ext4.wiki.kernel.org/index.php/Filesystem_Testing_Tools/mangle.c
  - Create random mutations of defined file content.
- Bugger http://www.fuzzing.org/
  - Randomly changes the data of the running program

# The single purpose fuzzers

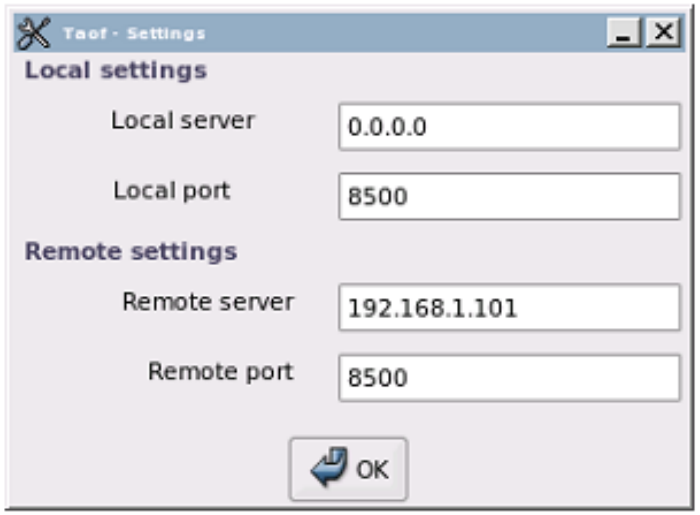
- Stress2 http://people.freebsd.org/~pho/stress/
  - Kernel fuzzer written for FreeBSD.
- Isic, isicng, ip6sic http://packetstormsecurity.org/search/?q=isic, http://isicng.sourceforge.net/, http://ip6sic.sourceforge.net/
  - Tools for fuzzing the network stacks.
- FTPFuzz http://www.infigo.hr/files/ftpfuzz.zip
  - Only fuzzes FTP servers
- Mangleme, htmler http://lcamtuf.coredump.cx/soft/mangleme.tgz, http://www.derkeiler.com/Mailing-Lists/Securiteam/2004-10/0088.html
  - Web browser fuzzers

# Taof

- http://sourceforge.net/projects/taof
- Author: Rodrigo Marcos
- Fully automatic network fuzzer.
  - Written in python.
  - Graphical interface.
    - Easy to use
  - Limited possibilities on many protocols.

# TAOF: Automatic data retrieval

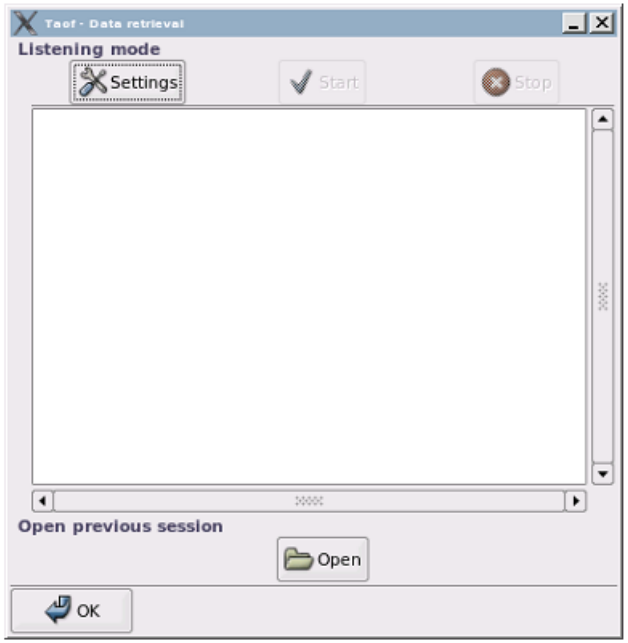
- Set the MITM environment.

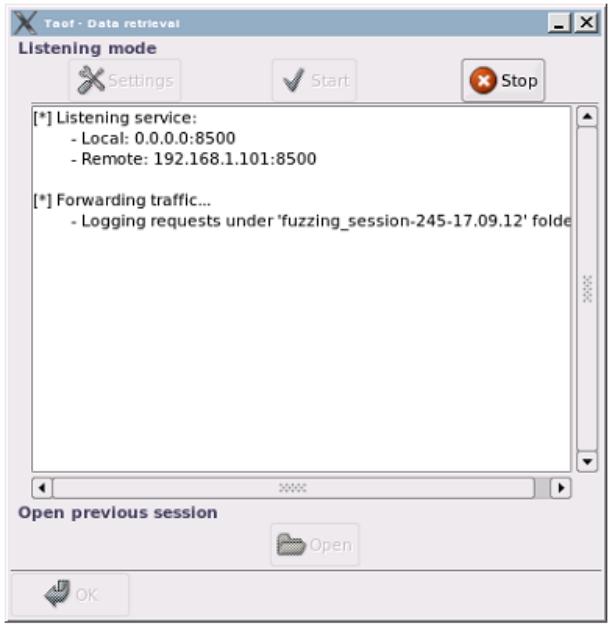



- The client will now connect to the TAOF computer to the port 8500 instead of to the server (192.168.1.101).

....

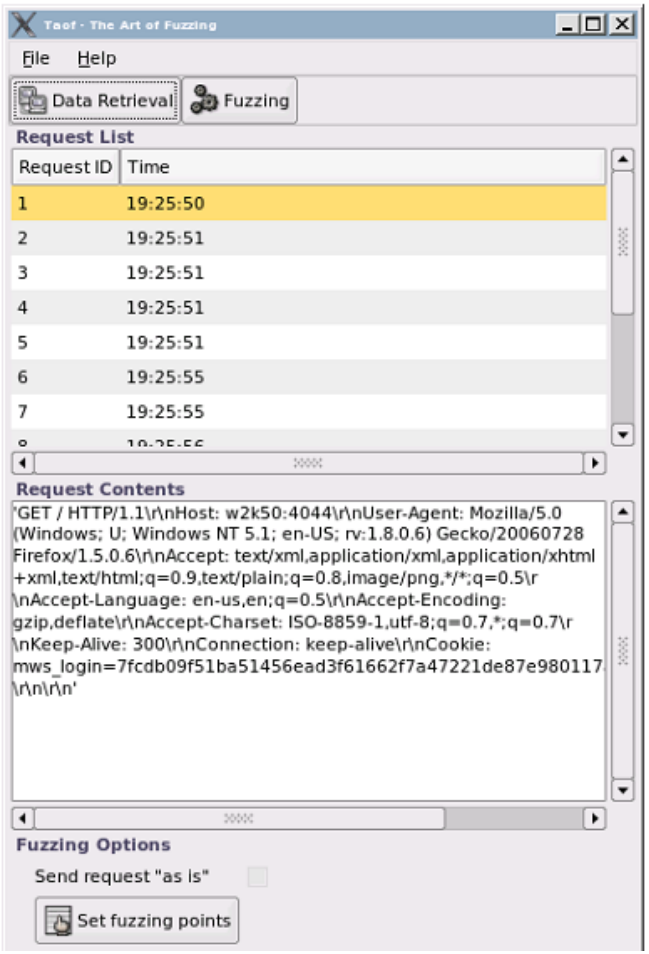
- Set forwarding mode.

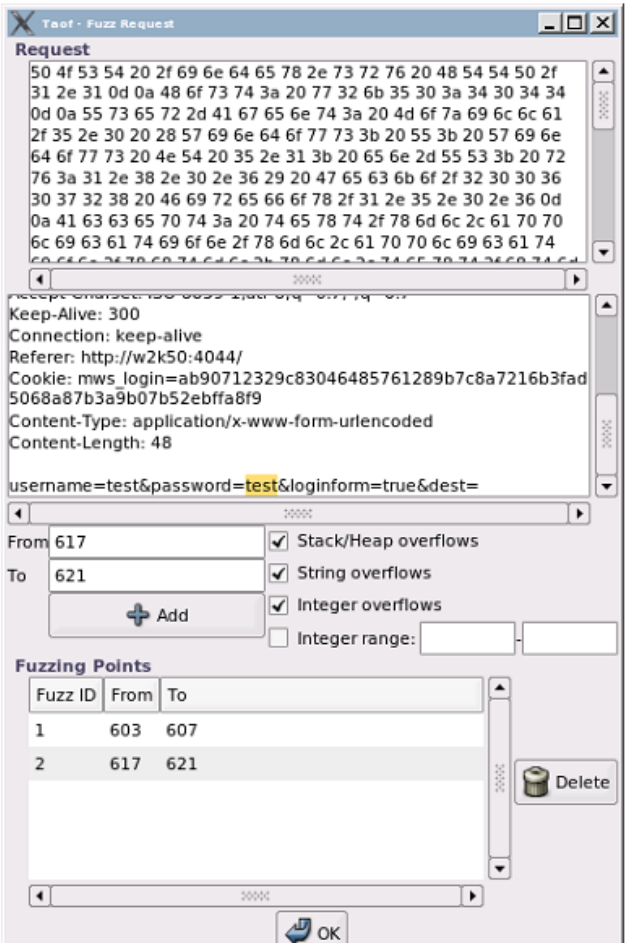



Capture the traffic.

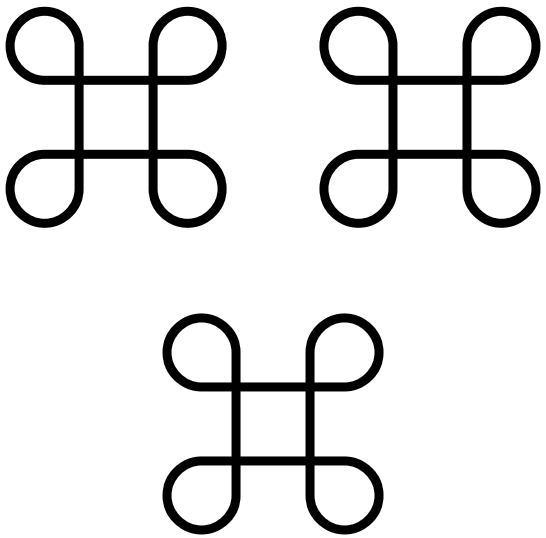
# TAOF: Data analysis



# TAOF: Fuzzing

- After setting all fuzzing requests.
  - Unlimited number of requests.
- Start fuzzing
  - The fuzzing will be run against the server.



# Create own fuzzer

- Reuse the code
  - Write from the scratch.
    - Use compiled language: c, c++ ...
    - Use scripting language: python, perl, ruby …
  - Use existing framework.
    - Rewrite the framework into another language.
    - Enlarge the framework.
    - Embed the framework into greater project.
- Dumb or smart fuzzer

# Dumb fuzzer

- Get some data (file, pcap snapshot, …).
- Modify randomly.
- Use (send to interface, use as argument).

- There are problems with protected data.
  - CRC, checksum.
  - Another data integrity checks.
  - Embedded data formats.

# Smart fuzzer: Wisdom excelleth folly

- To be protocol (data format) aware.
- Use smart loop, stop on target crash.

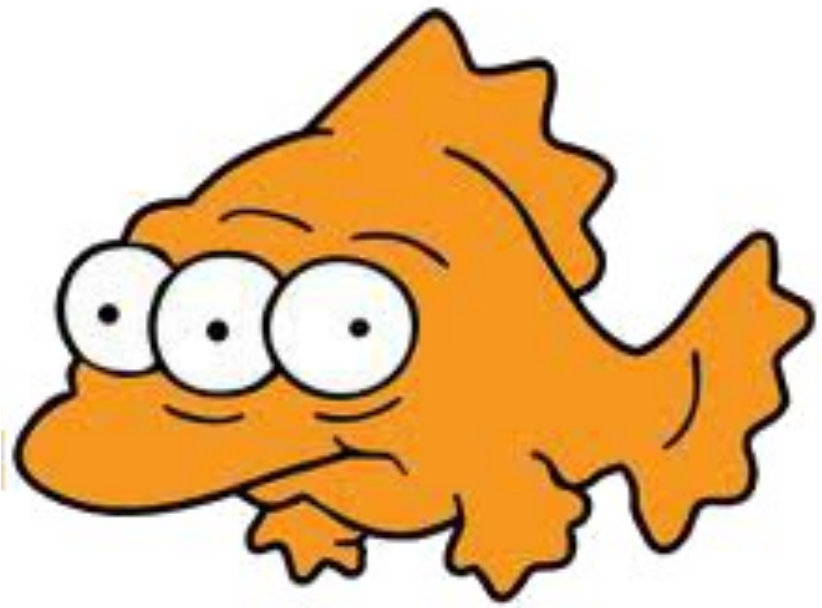
- The smart fuzzers gives much more results.
- It is necessary to understand the protocol for writing the smart fuzzer.

# The template for the generation

- Read and parse RFC's or other human-readable protocol descriptions
  - Generally will waste time by fuzzing non- implemented parts of the protocol
  - Will miss proprietary extensions
- Reverse engineering of protocol
  - Can be done semi-automatically
- Sniffing and statistical analysis
  - Even very dumb replay-and-bit-flipping can find many bugs
- If done poorly, target applications will ignore most of your traffic

# Data mutation

- Change some fields in existing formatted data.
  - Very fast to proceed.
  - Problem with embedded formats.
  - Often effective way how to fuzz.



# Data creation

- Generate data by the same way as they are usually did (with some fuzz).
  - Can generate encapsulated data.
  - Can play with all the possibilities.
  - More expensive than data mutation.
  - Runs slower than mutations.

# Data format for smart fuzzer.

- Strings
  - Very long strings [buffer overflow]
  - Containing %n [formatting bug]
  - Binary data
    - Zero byte inside the string
  - Empty strings [ x = strdup(str); c = x[2]];
  - Application specific dangerous format of the strings
    - SQL injection
    - Directory traversal
    - XSS
    - Command execution

# *Continuation*

- Special care on data delimiters.
  - Break by multiplying or reducing them.
- Data bracketing.
  - Illegail pairing. [ (<)> ]
  - Unpaired brackets.
  - Closing before opening. [ )( ]

# *Continuation II.*

- Size fields
  - Negative value (-1, 0x8000, 0x80000000, …)
  - If size correspond to string, size < strlen(string) + 1
  - Very huge positives (0xffff (uint32), 0x7fff (int), 0xffffffff (uint64), 0x7fffff (long long)…) [arithmetic or malloc overflow]
  - Small integers [ buf[len -2] = 0; ]

# Fuzzing sequence

- Sequential.
  - Finite in time, but sometimes in the long yonks.
  - Easy to create.
- Random.
  - Use /dev/urandom or random().
  - Infinite, must be terminated after some time.
  - Give better results.
- Sofisticated.
  - Based on knowledge the problem.
  - Use radamsa or some proprietal generator.

# Comments

- There may be more bugs chained.
  - Solving one opens another.
- There is the need turn off user responses.
  - Preload libraries.
  - Set up configuration.
  - Some scripting.
- It's necessary switch on the investigated sections.
  - And configure them correctly also.

# In case: crash

- Reboot in the case kernel testing.

- Create coredump.
- Attach debugger.

# In case: huge memory conpsumption.

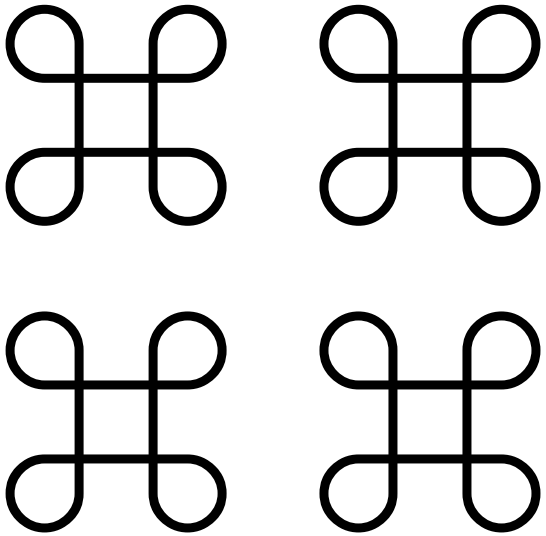
- Swap allocation.
- Slower response.
- Sometimes program aborts.

- Attach debuger.
- Use valgrind.

# In case: Program hangs.

- Run in infinite or very long loop.

- Attach debuger.
- Use strace or ltrace.



# Résumé

- Different fuzzers found different bugs.
  - There is huge difference between protocol driven versus random change driven fuzzers.
    - The protocol driven can found only that what is described but deeper in the code.
    - The random fuzzers found everything but only on the surface.
- The yesterday's fuzzers found mainly the yesterday's bugs
  - There is the need to update the testing tools everyday.
- Best result give the use of multiple various fuzzers.

# …

- The time to found a bug grow rapidly (exponentially) with the % of code coverage.
  - To run the fuzzer with rational coverage means to run it for long time.
  - The time can be estimated from known speed and known complexity of the fuzzed process and the fuzzer algorithm.
- Each fuzzer finds only a subset of potential bugs
  - There is impossible to assure the 100% coverage

# Literature

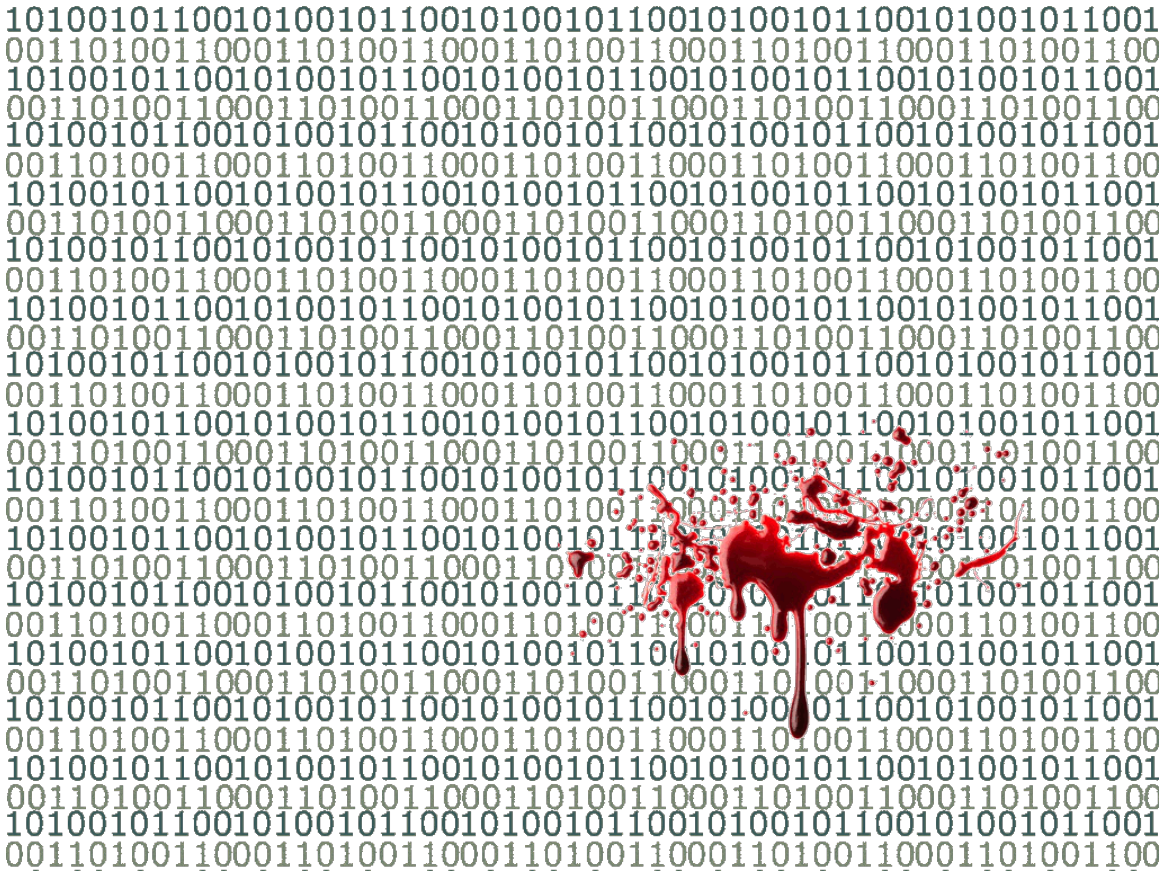
- Dave Aitel
  - An Introduction to SPIKE, the Fuzzer Creation Kit.
  - MSRPC Fuzzing with SPIKE 2006
  - MSRPC Heap Overflow – Part I, Part II
  - The Advantages of Block-Based Protocol Analysis for Security Testing
  - The Hacker strategy
- Pedram Amini
  - Fuzzing Frameworks.
- Pedram Amini, Aaron Portnoy
  - Introducing Sulley Fuzzing Framework
- Erik Pace Birkholz
  - Special Ops: Host and Network Security for Microsoft, UNIX and Oracle. (Syngres 2003) ISBN 1-931836-69-8
- Stephen Bradshaw
  - An introduction to fuzzing: Using fuzzers(SPIKE) to find vulnerabilities.
- Justin E. Forrester, Barton P. Miller
  - An Empirical Study of the Robustness of Windows NT Applications Using Random Testing

# Literature 2

- Laurent Gaffié
  - Fuzzing: The SMB case
- Barton P. Miller, Lars Fredriksen, Bryan So
  - An Empirical Study of the Reliability of UNIX Utilities
- Barton P. Miller, David Koski, Cjin Pheow Lee, Vivekananda Maganty, Ravi Murthy, Ajitkumar Natarajan, Jeff Steidl
  - Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services
- Barton P. Miller, Gregory Cooksey, Fredrick Moore
  - An Empirical Study of the Robustness of MacOS Applications Using Random Testing
- Charlie Miller
  - Fuzz by number
- Collin Mulliner, Nico Golde
  - Fuzzing the Phone in your Phone
  - Sms-o-Death
- Enno Rey, Daniel Mende
  - Advanced Protocol Fuzzing
- Ilja van Sprundel
  - Fuzzing.

# Literature 3

- Ilja van Sprundel
  - Fuzzing.
- Ari Takanen
  - Fuzzing : the Past, the Present and the Future
- Martin Vuagnoux
  - Autodafé: an Act of Software Torture.



# Thank you

Jan F. Chadima 2011

jchadima@redhat.com