

---

# Beating the bugs out of software

---

Juho Myllylahti, OUSPG

<https://www.ee.oulu.fi/research/ouspg>

---

# Agenda

---

- What are bugs
  - How can we deal with them
  - What is this "fuzzing" thing
  - Radamsa, a framework of fuzzers
-

# What is left as an exercise to the reader?

---

- Talk is aimed as an intro to the fuzzing as a general testing method, so some of the security aspects are omitted, such as automating bug triaging...
  - ...since probably the attendees wish to fix all the bugs that cause the program to crash
-

9/9

0800 Anttan started  
1000 " stopped - anttan ✓  
1300 (032) MP-MC ~~1.582647000~~  
                  (033) PRO 2 2.130476415  
                              convd 2.130676415

{ 1.2700 9.037847025  
          9.037846995 convd

Relays 6-2 in 033 failed special speed test  
in relay .. 11.00 test.

Relays changed

1100 Started Cosine Tape (Sine check)  
1525 Started Multy Adder Test.

1545



Relay #70 Panel F  
(moth) in relay.

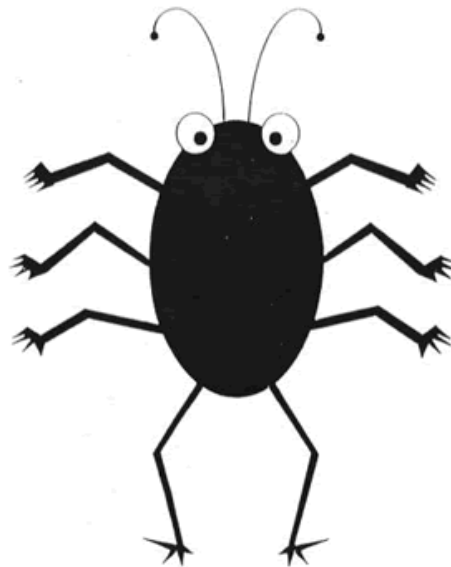
First actual case of bug being found.  
1630 Anttan started.  
1700 closed down.

Relay  
3145  
Relay 3370

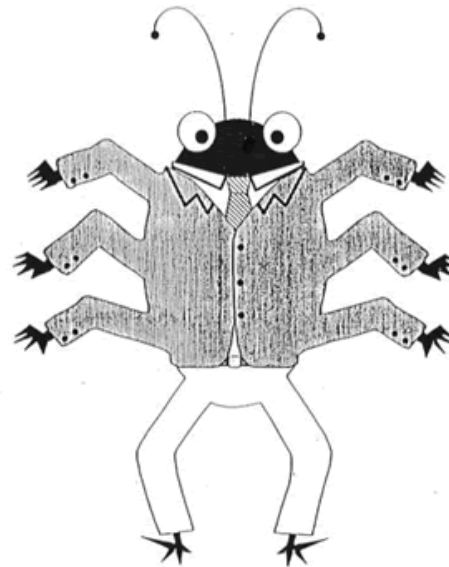
# Bugs, bugs, bugs!

---

- Bugs are errors or unintended "features" in programs



**BUG**



**FEATURE**

# Bugs, bugs, bugs!

- Bugs have a huge impact on software industry


**Bloomberg** Our Company | Professional | Anywhere

Search News, Quotes and Opinion

HOME QUICK NEWS OPINION MARKET DATA PERSONAL FINANCE **TECH** POLITICS SUSTAINABILITY LUXURY TV VIDEO RADIO

## The Big Cost of Software Bugs

By Jordan Robertson, Marcus Chan and Mark Milian - Aug 3, 2012 10:53 PM GMT+0300



9 of 11 [< PREV](#) | [NEXT >](#)

### Bad Brake for Toyota

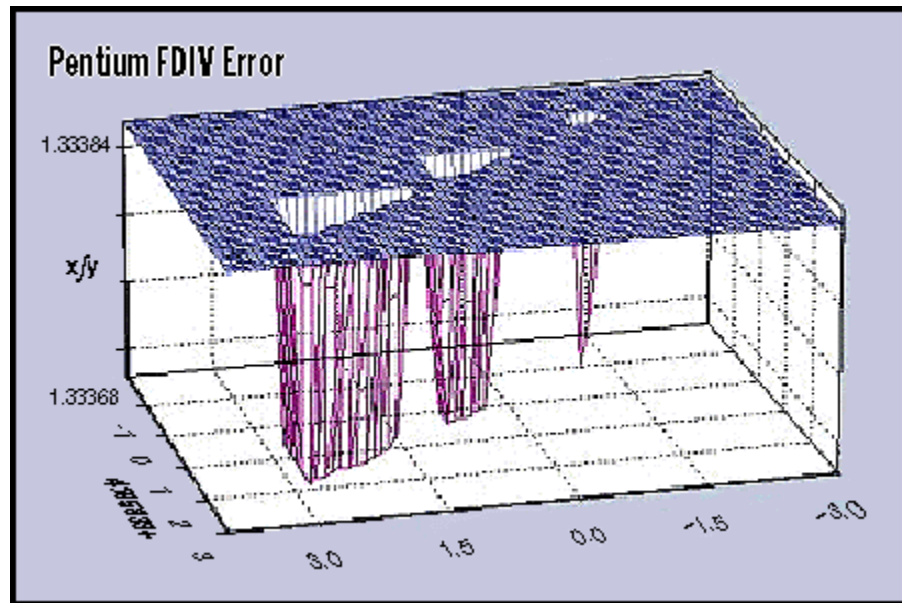
When Toyota recalled more than 400,000 of its hybrid vehicles in 2010, it wasn't because of a mechanical issue. The cars, including Toyota's Prius line, had a software glitch, which would cause a lag in the anti-lock-brake system. The news came at a time when Toyota had come under scrutiny for safety issues related to physical defects in millions of other cars it had made. Class-action lawsuits resulting from those recalls, including the software glitch, were estimated to cost Toyota as much as \$3 billion, according to an estimate by the Associated Press.

*Photograph by Kiyoshi Ota/Bloomberg*

# Bugs, bugs, bugs!

---

- Present themselves in all parts of the software and hardware stack

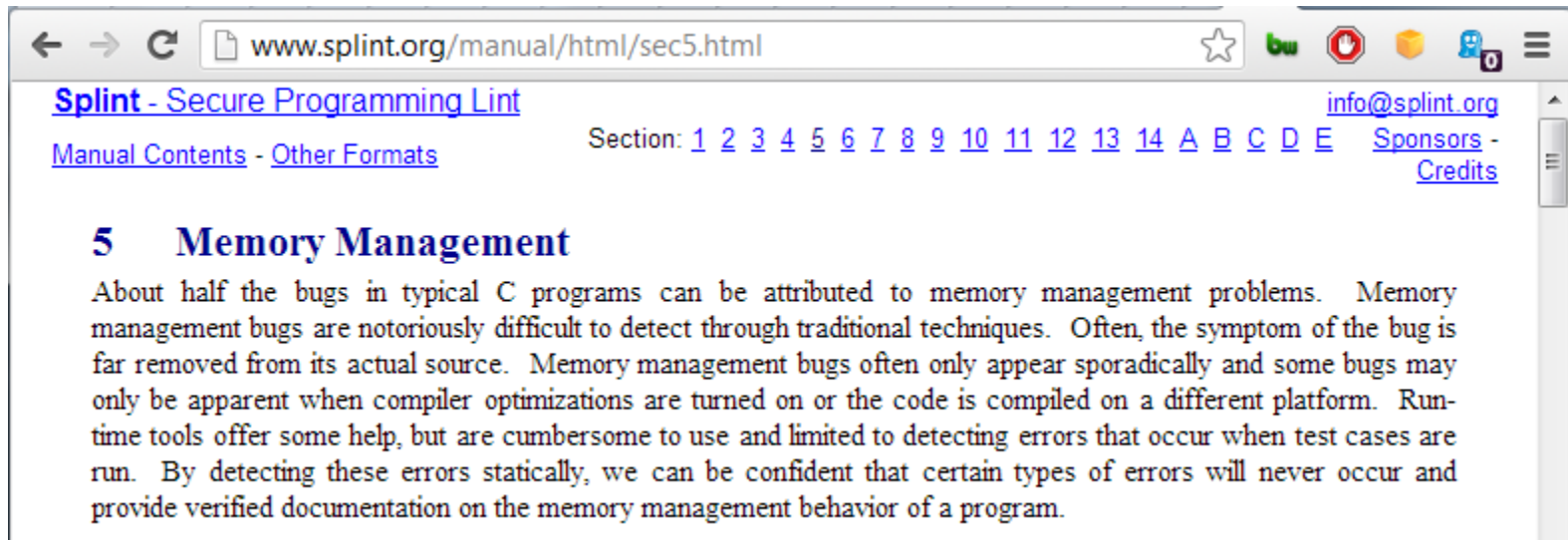




# Bugs, bugs, bugs!

---

- Choice of software development tools affects what kind of bugs you generally need to expect





# Bugs – why do they still exist?

---

- Programming is hard
  - Most tools we use today make it easy to program errors
    - Google "A quiz about integers in C"
    - Difficulties in (manual) memory management
    - Difficulties in managing side effects and state, especially in multi-threaded environment
    - Etc.
-

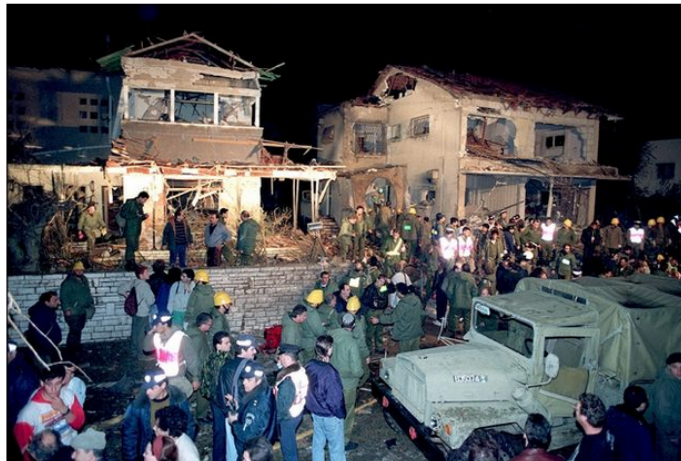
# So there are bugs. Can't we just turn it off and on again?

- Bugs are not always a mere nuisance
  - Bugs that cause dangerous behaviour
  - Bugs that have security repercussions



## The Big Cost of Software Bugs

By Jordan Robertson, Marcus Chan and Mark Millan - Aug 3, 2012 10:53 PM GMT+0300



Photograph by Patrick Baz/AFP/Getty Images

4 of 11

[< PREV](#) | [NEXT >](#)

### Patriot's Fatal Error

Sometimes, the cost of a software glitch can't be measured in dollars. In February of 1991, a U.S. Patriot missile defense system in Dhahran, Saudi Arabia, failed to detect an attack on an Army barracks. A government report found that a software problem led to an "inaccurate tracking calculation that became worse the longer the system operated." On the day of the incident, the system had been operating for more than 100 hours, and the inaccuracy "was serious enough to cause the system to look in the wrong place for the incoming Scud," said the General Accounting Office report. The attack killed 28 American soldiers.

Prior to the incident, Army officials had fixed the software to improve the Patriot system's accuracy. That modified software reached Dhahran the day after the attack.

Shown here is a Scud missile that got through the defenses near Tel Aviv, Israel on February 9, 1991.

# So we should take bugs seriously, now what?

---

- Consider the development tools
  - Unit testing
    - Combine with CI-servers and check the coverage
  - Code reviews
    - Tools like Gerrit
  - Fuzzing
  - Proper channels for users to report bugs
  - Bounty programs
    - "Outsource" some of the bug hunting
    - Increasing amount of companies see this beneficiary
    - If you're not willing to commit money on this...large amount of bug bounty hunters use fuzzing
-

# So we should take bugs seriously, now what?

---

- Consider the development tools
  - Unit testing
    - Combine with CI-servers and check the coverage
  - Code reviews
    - Tools like gerrit
  - **Fuzzing**
  - Proper channels for users to report bugs
  - Bounty programs
    - "Outsource" some of the bug hunting
    - Increasing amount of companies see this beneficiary
    - If you're not willing to commit money on this...large amount of bug bounty hunters use fuzzing
-



# Fuzzing

---

- Form of negative testing
    - Negative testing is often forgotten
    - Good negative testing methods are absent, apart from fuzzing
  - Randomly mess up the data, feed it to the program and see what happens
  - Power in numbers: test cases are generated fast
  - A really effective method, measured on (human) time spent / bug found
-

# Fuzzing

---

- Crudest example: `cat /dev/urandom | ./software`
  - ...and even this still works.
- Evolution: data that sort of looks right



# Example

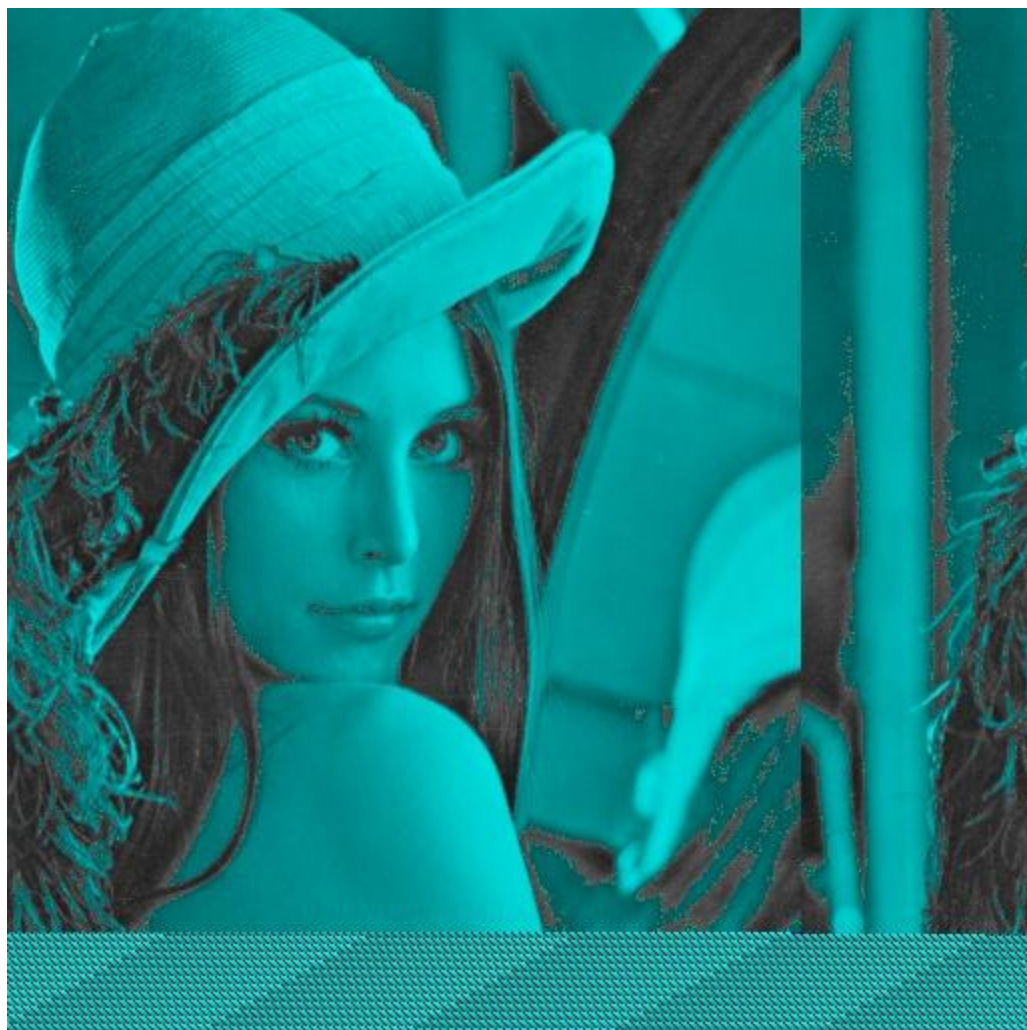
---

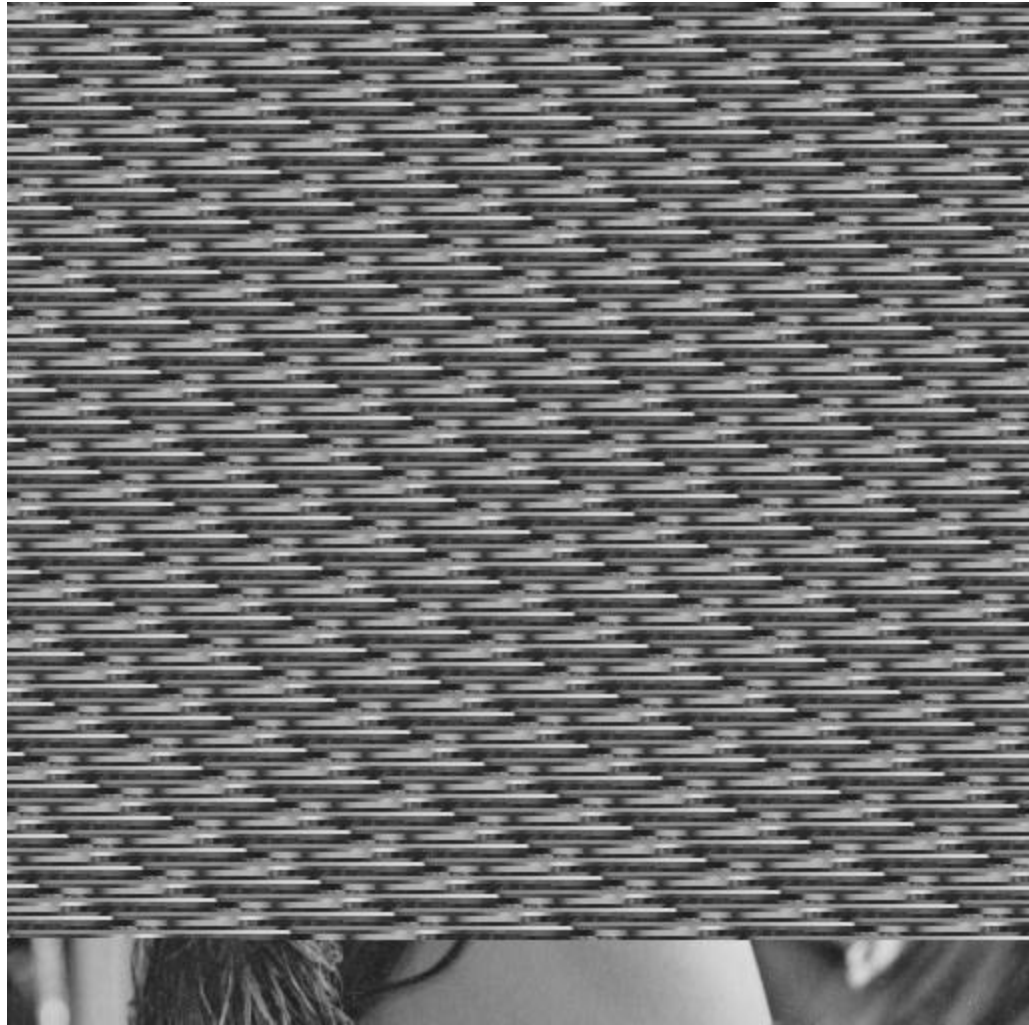
```
radamsa -n 100 -o lena-%n.bmp lena.bmp
```

---













# Advantages of fuzzing

---

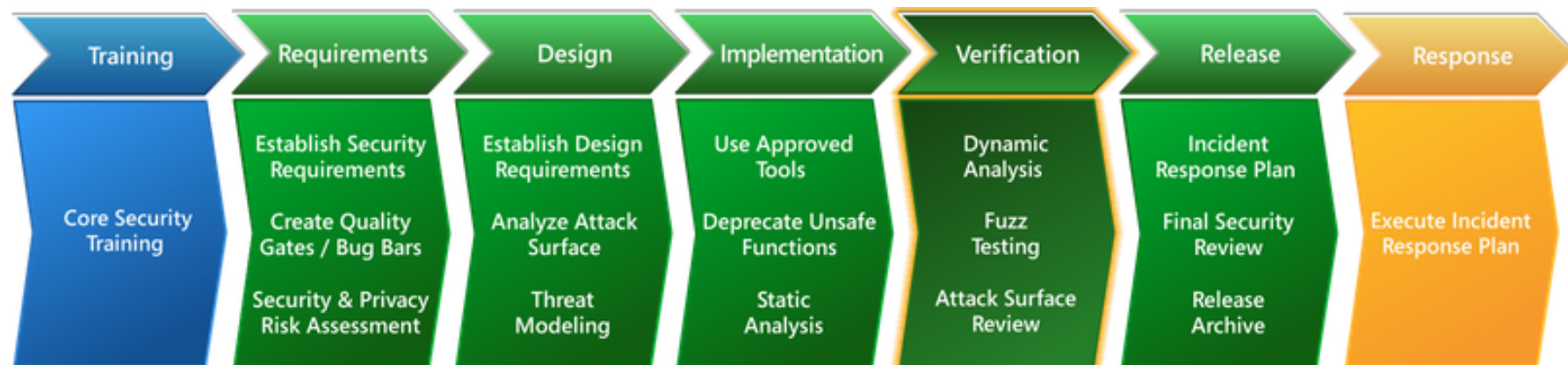
- Low barrier of entry
    - Download a fuzzer
    - Gather some input
    - Feed the input to the program
    - See what happens
  - Fun!
  - Cheap
  - Effective
    - CVEs
    - Recommended by Microsoft in their SDL
-



---

*Credit to Devdatta Akhawe, UC Berkeley.*

- [\$1000] [104056] **High** CVE-2011-3957: Use-after-free in PDF garbage collection. *Credit to Aki Helin of OUSPG.*
  - [\$2000] [105459] **High** CVE-2011-3958: Bad casts with column spans. *Credit to miaubiz.*
  - [\$1000] [106441] **High** CVE-2011-3959: Buffer overflow in locale handling. *Credit to Aki Helin of OUSPG.*
  - [\$500] [108416] **Medium** CVE-2011-3960: Out-of-bounds read in audio decoding. *Credit to Aki Helin of OUSPG.*
  - [\$1000] [108871] **Critical** CVE-2011-3961: Race condition after crash of utility process. *Credit to Shawn Goertzen.*
  - [\$500] [108901] **Medium** CVE-2011-3962: Out-of-bounds read in path clipping. *Credit to Aki Helin of OUSPG.*
  - [109094] **Medium** CVE-2011-3963: Out-of-bounds read in PDF fax image handling. *Credit to Atte Kettunen of OUSPG.*
  - [109245] **Low** CVE-2011-3964: URL bar confusion after drag + drop. *Credit to Code Audit Labs of VulnHunt.com.*
  - [109664] **Low** CVE-2011-3965: Crash in signature check. *Credit to Sławomir Błażek.*
  - [\$1000] [109716] **High** CVE-2011-3966: Use-after-free in stylesheet error handling. *Credit to Aki Helin of OUSPG.*
  - [109717] **Low** CVE-2011-3967: Crash with unusual certificate. *Credit to Ben Carrillo.*
-



[previous phase](#) | [next phase](#)

#### SDL Practice #11:

##### Perform Dynamic Analysis

Dynamic analysis is run-time verification of your software, leveraging tools which monitor application behavior for memory corruption, user privilege issues, and other critical security problems.

#### SDL Practice #12:

##### Fuzz Testing

Fuzz Testing is a specialized form of dynamic analysis that induces program failure by deliberately introducing malformed or random data to an application.

#### SDL Practice #13:

##### Attack Surface Review

Attack surface review is a security practice that ensures any design or implementation changes to the system have been taken into account, and that any new attack vectors created as a result of the changes have been reviewed and mitigated including threat models.

#### Why should I follow this practice?

Dynamic analysis ensures software functionality works as designed.

Fuzz testing is an effective way to find potential security issues prior to release while requiring modest resource investment.

Attack surface review is a security practice that ensures any design or implementation changes to the system have been taken into account, and that any new attack vectors created as a result of the changes have been reviewed and mitigated including threat models.

# What kinds of bugs fuzzing usually finds?

---

- Bugs in memory safety
    - Use-after-frees
    - Null pointers
    - Memory leaks
  - Parser bugs
  - Threading bugs
  - Invalid error handling
  - Correctness bugs e.g. "these two systems should function identically"
  - Cluster effects: how well the interoperability really works
-

# What kind of bugs fuzzing usually doesn't find?

---

- High-level bugs
    - For example design-level crypto bugs
    - Implementation bugs on the other hand...
  - Usability bugs
  - Bugs that lie behind input validation
    - If messages fuzzed are composed of input+checksum, valid mutations are not probable
    - But of course you can fuzz the data first and then calculate the checksum...if you are familiar with the file schema
-

# Things to note when fuzzing

---

- Effective fuzzing requires at least some form of an automated testing oracle
    - Testing oracle = something that tells us if the program behaves correctly
  - Feeding the input to the program should be possible to automate
  - A single test case should not take too long to execute
-

# How to improve the effectivity of fuzzing

---

- Structure of the input
    - If the program input is a ZIP archive, then fuzzing the ZIP might mostly test the extracting engine (Java .jar files are an example of this)
  - Does the program take in several different file formats?
    - It might or it might not be beneficial to mix samples of different formats
  - Try to determine all the input vectors
    - E.g. environment variables
  - Beware the smartness creep
    - "There surely can't be a fault in the favicon image handler!"
-

# Instrumentation and automation

---

- At minimum we should be able to automate the feeding of the test cases and the detection of crashes
  - Automatic aggregation and generation of test cases is nice but not obligatory
  - Crash detection = test oracle
    - Something that tells us if the program survived the test case or if it crashed or reacted improperly
    - Doesn't need to be perfect, just "good enough"
    - Usually not difficult to come up with a satisfactory result, especially if dealing with your own codebase
-



# Possible components for the test oracle

---

- Console output: STDOUT and STDERR
  - System logs
  - Time
  - System activity (network, disk, memory, CPU)
  - HTTP return codes
  - Tools such as address sanitizer
  - For servers: testing the state by ensuring a valid case is handled correctly
-

# Phases of a typical fuzzing script

---

1. Generate a batch of fuzzed input or input files
  2. Do preparations to clean the state, prepare the oracle etc.
  3. Start a loop which feeds the test cases to the application
    - 3.1. Consider if the program should be restarted for every run or if it is possible to reuse the same instance for numerous test cases
  4. If crash is found, stash the test case and log the error and the auxiliary information
-

# Ad hoc

---

- It's best to try to get something running as soon as possible
  - Next version can be worked on while the initial version runs
  - So, start with Something(TM) and then improve the testing with iterations
-

# When to stop or reiterate?

---

- Usually bugs should be frequent at first, less frequent after a while and finally they should become more and more infrequent
  - Some explanations:
    - Software has become more robust and the bugs are fixed
    - Sample files should be updated, since the current ones have become "depleted"
    - Fuzzer needs some adjustment to make it behave differently
    - The instrumentation might leak and miss bugs
-

# Free samples!

---



- Good starting point: the more the merrier
  - Next point to focus on is the quality: the samples should be as diverse as possible
  - When going pro the goal is usually a good collection of assorted diverse samples
  - Google filetype:pdf etc. is your friend...
  - ...but it's not the silver bullet
  - Also search for existing sample suites
  - Freebie: curated test sets are a nice resource in general when doing development stuff
-

# Determining magic values

---



- For example, how long to wait for the software to handle a single test case, how often to check logs if they are slow...
  - No right answers for this :-)
  - Proceeding with a gut feel might not be a bad idea, the decisions are easy to change if needed
  - Some random variance might not be a bad idea
  - or...
-

An idea to ponder on

---





# Instrumentation tools for Windows

---

- Powershell
    - Get-Eventlog: Get-Eventlog application
    - Get-WmiObject: Get-WmiObject -class win32\_reliabilityRecords -Filter "ProductName = ' '"
  - GFlags and pageheap
    - gflags /p /enable calculator.exe
  - Event Tracing for Windows
  - Console output, exit and error codes
  - WinDBG
    - !analyze
    - !exploitable (restrictions apply)
-

# Instrumentation Tools for UNIX

---

- Exit codes
  - dmesg
  - AddressSanitizer
  - strace and/or dtrace
  - systemtap
  - gdb and cgdb
  - valgrind
  - CERT Linux Triage Tools
  - CrashWrangler (OS X)
-

# Instrumentation tools for servers

---

- HTTP return codes
  - Checking valid case behaves correctly
  - Server logs
  - Time
  - Return value from the server
  - CPU load and other metrics
-

# Combining fuzzing with other testing activities

---

Some ideas:

- Automatically send crashed test cases to bug tracker accompanied with the crash info
  - It is possible to include a fuzzing step to CI-servers (at least for Buildbot), but it takes some planning to make it all work
  - If fuzzing is integrated to testing, add it as a separate test suite to prevent random (sic) build failures
-

# Additional info

---

- Google, obviously
  - <https://www.ee.oulu.fi/research/ouspg/>
  - <https://code.google.com/p/ouspg/>
  - <https://www.owasp.org/index.php/Fuzzing>
  - Books about fuzzing (there are a few)
  - IRC: #radamsa @ freenode
  - ouspg@ee.oulu.fi
-

# Thank you!

---

...and any questions? :-)

