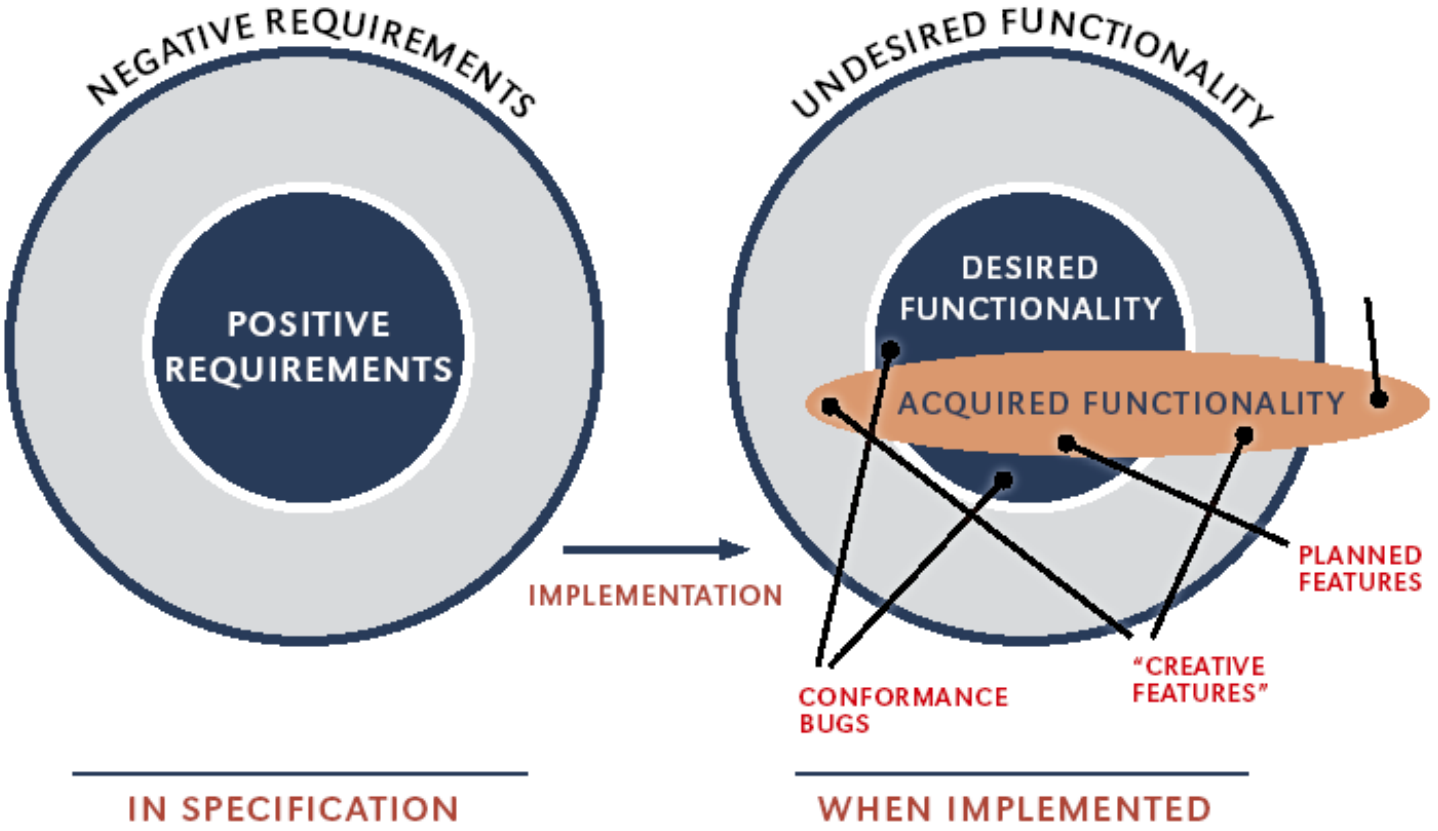
Fuzzing with Radamsa and some thoughts about coverage

What we do – finding “creative features”



(A) grouping of fuzzing tools

- Random fuzzing
- Model-based
  - Models of input -> test cases
  - Works well for well-formed network protocols with spec, handles checksums etc.
  - Much effort required for generating test suite
- Model-inferred-based
  - Samples of input -> test case
  - Cannot capture all nuances of spec
  - Low effort required for generating test suite

Radamsa

- Collection of model-inferred-based fuzzers with some elements of generation based ones
- Main goal: Easy to get started and get results
  - Select test subject
  - Gather input files
  - Run
  - Wait for crashes
  - Profit

Samples?

- Should be "representative" of format features
  - If conformance tests are available, they make great sources for robustness tests.
  - Regression tests are also awesome
  - For network protocols, captures of usage
  - If all else fails, random files from google
    - filetype:pdf etc.
- “Works” even without any samples, but better results if you have several of the same operation

Lots of small fuzzers (0.1)

grafu: Mutate a grammar inferred from the data.
fubwt: Mutate the with a disturbed Burrows-Wheeler transformation.
permu: Permute content around the edges of common substrings.
rambo: A traditional file fuzzer making simple changes to raw data.
enumr: Enumerate all byte sequences after a few shared prefixes.
stutr: Add repetitions to the data.
tehfu: Build a tree using simple rules and mutate it.
cutup: Splice and permute the contents of the files.
flipr: A bit flipper.
walkr: Make systematically mutations to all positions of all files.
range: generate chunks of random data
noise: mix sample data and noise signals together
forml: generate data from random formal languages
proby: Fill holes from files by using statistics.
surfy: Jump between locations with common data.
small: Test case minimizer. Hopefully not effective if used as a fuzzer.

Basic usage

```
[pp@laptop ~]$ echo "" | ./radamsa

[pp@laptop ~]$ echo "" | ./radamsa


[pp@laptop ~]$ echo "Testing 101" | ./radamsa
Testing 65535
[pp@laptop ~]$ echo "Testing 101" | ./radamsa
Terting 101
[pp@laptop ~]$ echo "Testing 101" | ./radamsa
Testing 10111
```

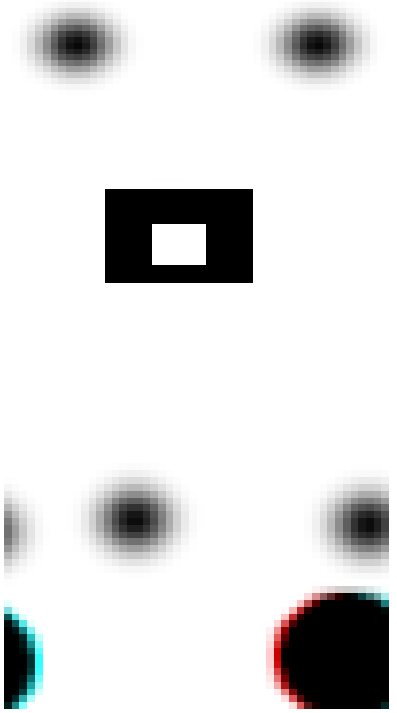
Picture example

50x50px raw PPM images

radamsa -e all -o tmp/%f-%n.ppm

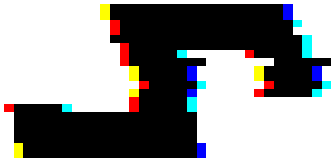
permu

- Permute content around shared substrings



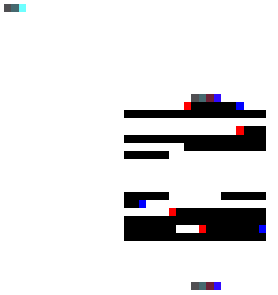
grafu

- Mutate grammar built



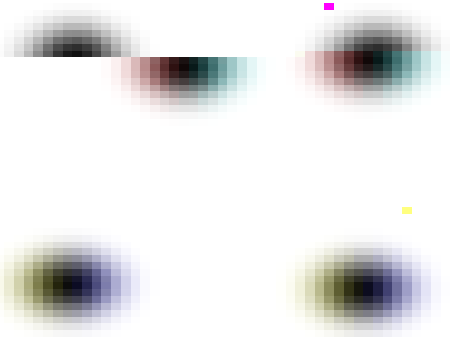
fubwt

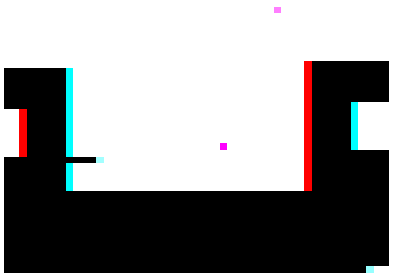
Compute transformed samples, mutate one slightly, perform the inverse transformation, and flip between original and mutated data at random positions.



rambo

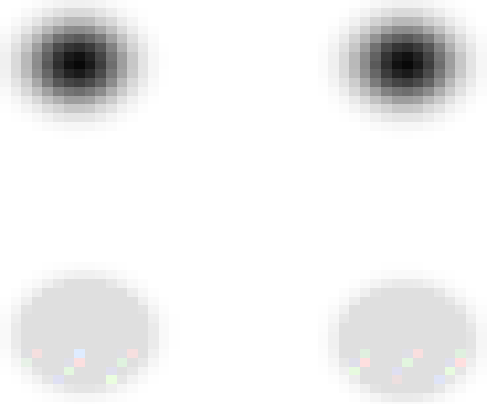
Brute obvious byte-level mutations at random positions (drop, repeat, increment mod 256, increment with overflow to right, toggle high bit)





noise

- Mix sample data together with noise signals
- Multiple signals may overlap
- Use rising (+n mod 256), flat, random data etc



Complementary tools

- Fuzzers
  - Bunny the Fuzzer (http://code.google.com/p/bunny-the-fuzzer/)
  - Peach (http://peachfuzzer.com/)
  - Zzuf (http://caca.zoy.org/wiki/zzuf)
  - Sulley (http://code.google.com/p/sulley/)
  - Skipfish (http://code.google.com/p/skipfish/)
- Diagnostics
  - GDB (http://www.gnu.org/software/gdb/)
  - Valgrind (http://valgrind.org/)
  - strace (http://sourceforge.net/projects/strace/)
  - tcpflow (http://www.circlemud.org/~jelson/software/tcpflow/)

Aki Helin's / aki.helin@ee.oulu.fi quest against bugs

CVE-2011-1434 Chrome, CVE-2010-0001 gzip,

CVE-2010-0192 Adobe, CVE-2011-0155 WebKit,

CVE-2011-0074 Firefox, CVE-2011-0075 Firefox,

CVE-2010-1205 LibPNG,CVE-2010-1793 WebKit,

CVE-2010-1404 WebKit, CVE-2010-1410 WebKit,

CVE-2010-1415 WebKit, CVE-2010-1415 WebKit,

CVE-2011-1186 Chrome,CVE-2011-2348 Chrome

Fuzzing Coverage

When are we done with robustness testing?

Code coverage does not help

Simple sample:

```
void foo(char *str) {
 char buffer[16];
 strcpy(buffer,str);
}
```

Useful merely to show which parts were not touched by a test ….

A fault model for malformed messages

Beizer, 1990:

1. A valid input is not recognized.
2. An invalid input is accepted.
3. An input, whether valid or invalid, causes a failure.

- Basis for anomaly (aka exceptional element creation)

Towards measurable test purpose

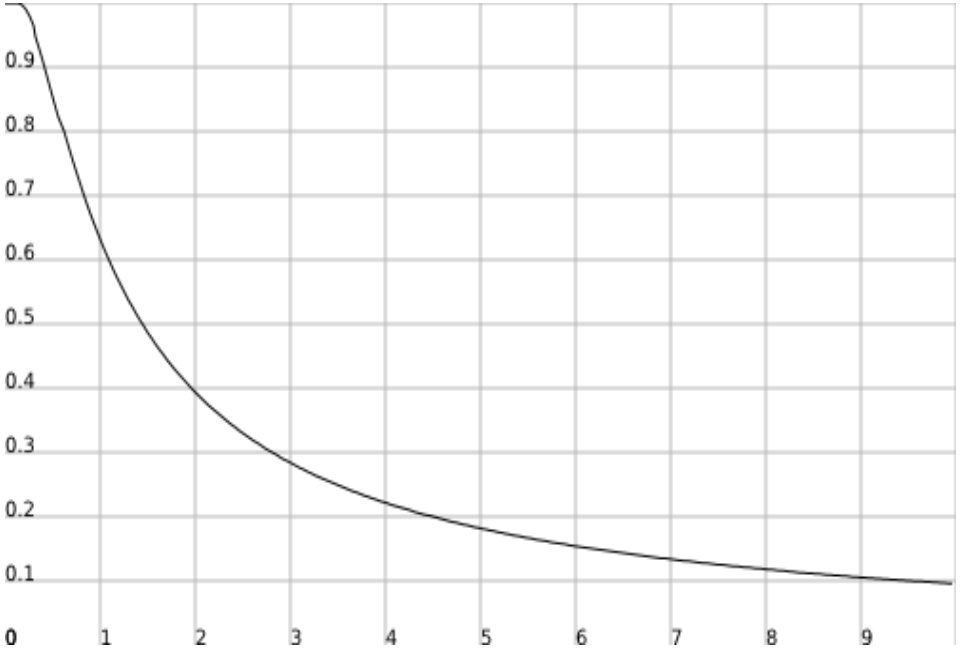
- Issue: input space of a typical software interface is practically infinite (esp. in negative testing)

Anomaly-based test purpose

- Exceptional elements informally collected (and guessed)
- Part of the Common Weakness Enumeration
  - Publicly available http://cwe.mitre.org/
  - Systematically counted
  - ID’d
- Combining CWE-ID’s with anomalies
  - Via tagging

Depletion fuzzing

- Assumption
  - Many bugs are found in the beginning of fuzzing
  - The more tests – the better.
  - Good enough after n fuzzed test cases passed
- Microsoft SDL: 100k passed iterations



Bounties

Security fixes and rewards:

Firstly, we have some special rewards for some special bugs!

- [$10,000] [116661] **Rockstar** CVE-1337-d00d1: Excessive WebKit fuzzing. *Credit to miaubiz.*
- [$10,000] [116662] **Legend** CVE-1337-d00d2: Awesome variety of fuzz targets. *Credit to Aki Helin of OUSPG.*
- [$10,000] [116663] **Superhero** CVE-1337-d00d3: Significant pain inflicted upon SVG. *Credit to Arthur Gerkis.*

- Technology independent
- Utilizing grey/black hats moving their capabilities away(?) from OrgCrime
- Simplifies vulnerability management

Final thoughts

- Fuzzing is effective
  - Internally: training/awareness of developers
  - Is hardening the software
- Fuzzing coverage - no silver bullet in sight.
  - Software and their dependencies are (too?) complex
  - “Wer mißt - mißt Mist”
- Practical approach
  - Depletion fuzzing as part of the SDLC
  - Bounties for externals

Thank you!

ouspg@ee.oulu.fi
http://www.ee.oulu.fi/research/ouspg/