How Smart is Intelligent Fuzzing - *or* - How Stupid is Dumb Fuzzing?

Charles Miller

Independent Security Evaluators

August 3, 2007

cmiller@securityevaluators.com

Agenda

- Introduction
- Portable Network Graphics
- libpng
- Mutation vs Generation Based Fuzzing
- Conclusions

Introduction

- “Intelligent fuzzing usually gives more results” - Ilja van Sprundel
- Can we quantify this statement?
- How important is the choice of inputs for mutation-based fuzzing?

Fuzzing

- Generate test cases - files, network traffic, command line arguments, environment variables, etc.
- Test cases should be “close” to real program inputs but should contain anomalies
- Test cases fed into the target application which is monitored for faults
- These anomalies are meant to defy programmer assumptions and find bugs

How to Get the Test Cases

- Mutate existing inputs (*dumb fuzzing*)
  - § Take a valid input, say a file, and make changes to it
  - § These changes can include modifying bytes, adding strings, %n's, etc.
  - § Easy and fast to do
  - § Doesn't require knowledge of the program or protocol
  - § Dependent on the existing inputs

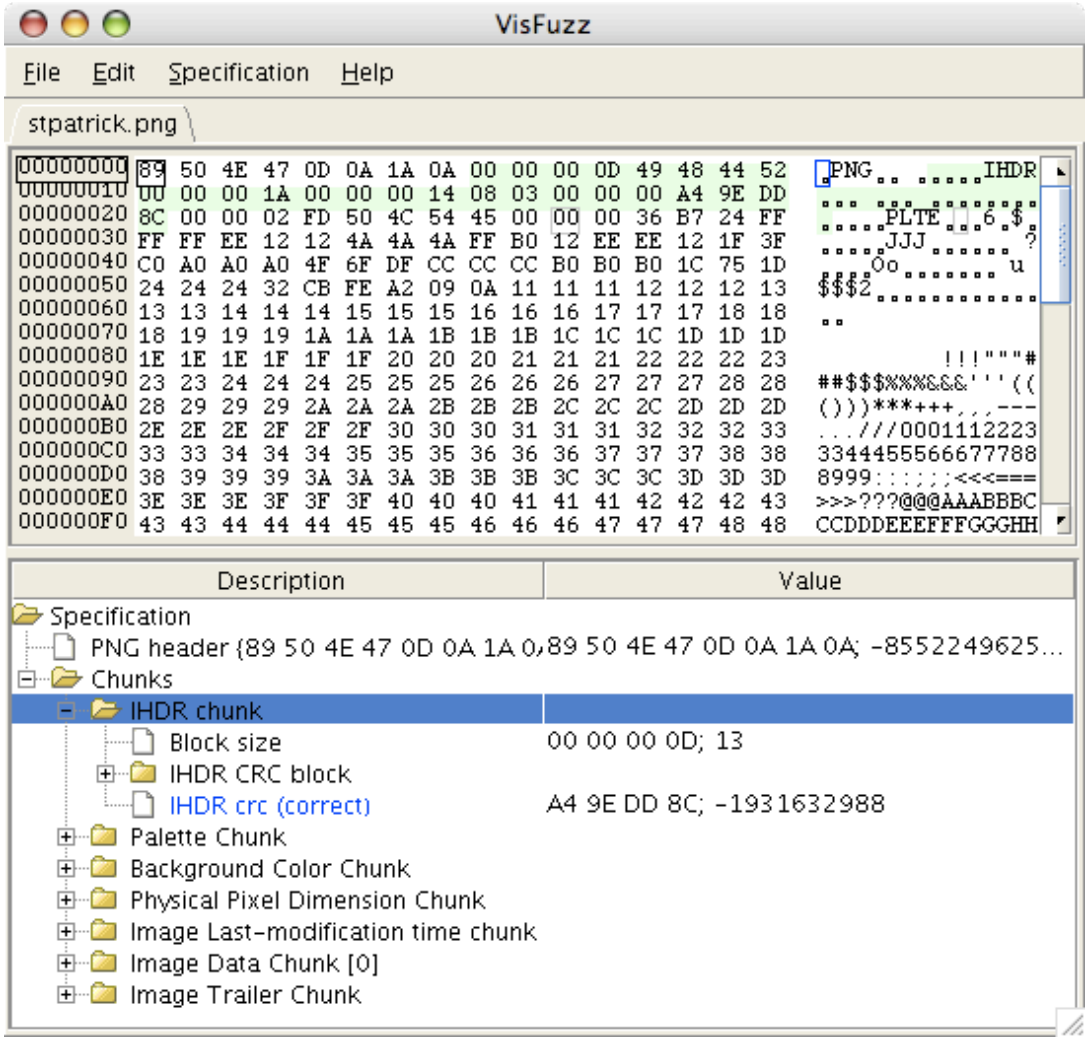
How to Get the Test Cases (Cont)

- Generate inputs from protocol description(*Intelligent fuzzing*)
  - § Start from RFC or documentation
  - § Generate inputs based on documentation
  - § For each field in the description, add an anomaly, such as a long strings, negative numbers, %n's etc
  - § Takes a long time to create the inputs
  - § Tedious work
  - § Requires ***complete*** knowledge or program or protocol
  - § Since all possible fields are fuzzed, *should* be more thorough

PNG Specification

- 8 byte signature followed by “chunks”
- Each chunk has
  - 4 byte length field
  - 4 byte type field
  - optional data
  - 4 byte CRC checksum
- 18 chunk types, 3 of which are mandatory
- Additional types are defined in extensions to the specification (I look at 21 types; the number known by libpng)

Sample PNG File

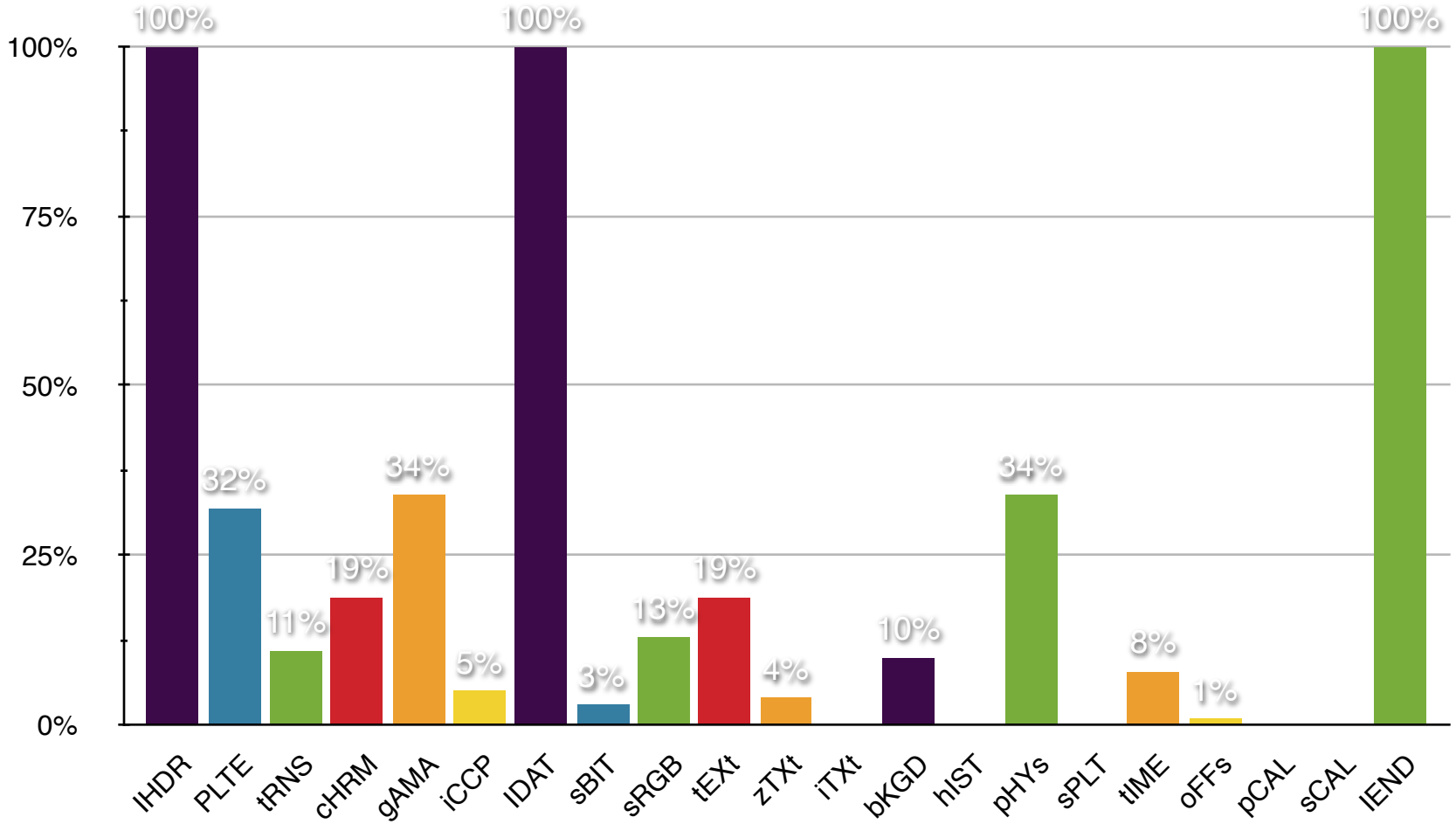


PNG's From the Wild

- Collected 1631 unique PNG files from the Internet
- Each file was processed and the chunk types present in each was recorded
- Typically, very few chunk types were present

Number of files	Mean number of chunk types	Standard deviation	Maximum	Minimum
1631	4.9	1.3	9	3

Distribution of Chunks Found



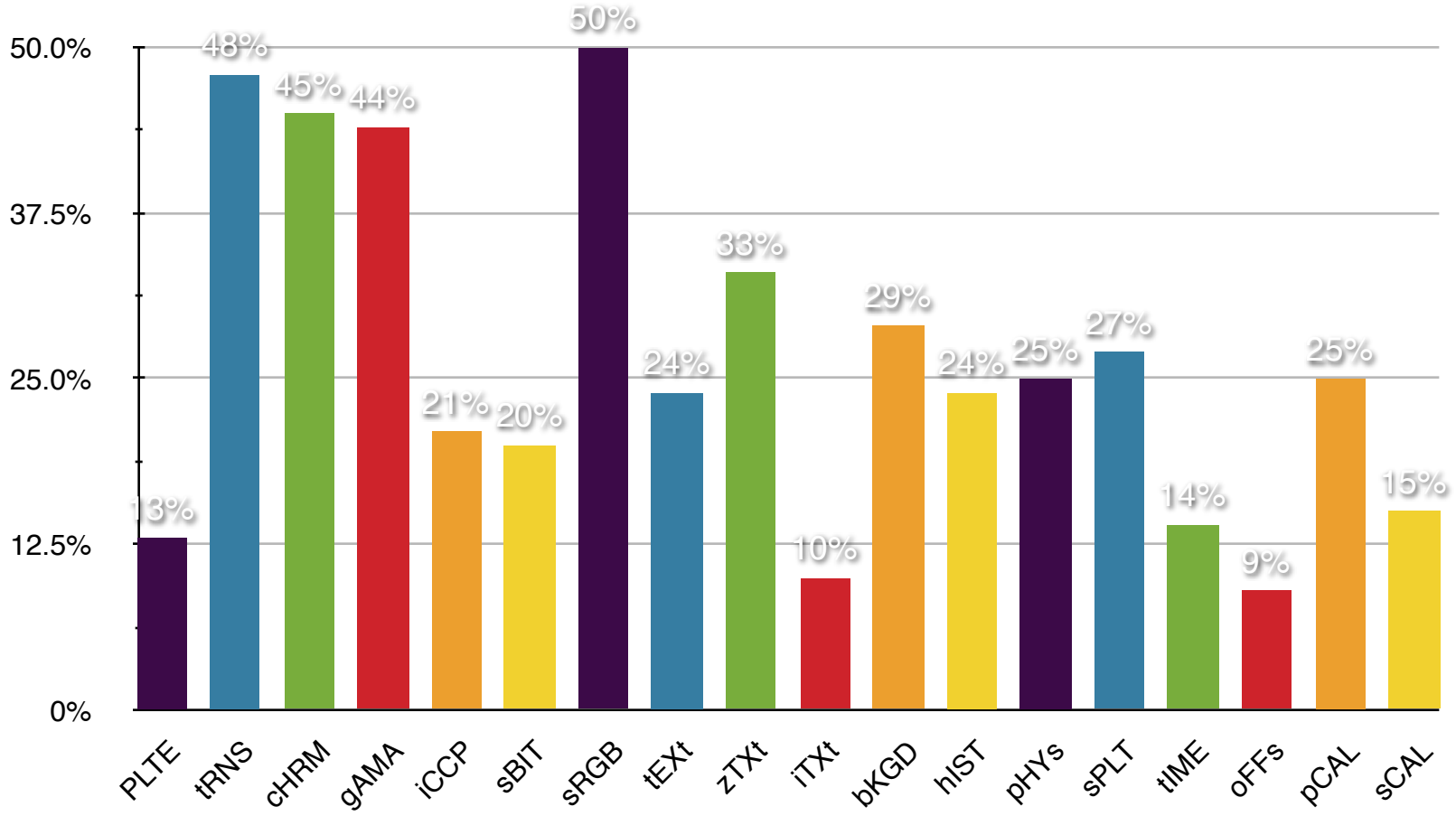
Observations

- On average, only five of the chunk types are present in a random file!
- 9 of the 21 types occurred in less than 5% of files
- 4 of the chunk types never occurred
- Mutation based fuzzers will typically only test the code from these five chunks
- They will *never* fuzz the code in chunks which are not present in the original input

libpng

- libpng is an open source PNG decoder
- Used in Firefox, Opera, and Safari
- We want to check that each chunk type really has unique processing code
- We generate PNG's containing the 3 mandatory and then one more chunk type
- We use gcov to record code coverage while it processes fuzzed versions of this type (approximately 1000 files per type)

Code Coverage for Each Chunk Type



Number of lines of code required to process each type as a percentage of the total number of lines required to process a minimal PNG file

So...

- Some chunk types require more code than others for processing
- The 4 chunk types which were not found in the wild represent 76% more code than a minimal PNG.
- This code will not be fuzzed using a mutation based method

Mutation vs Generation Based Fuzzing

- Generation based fuzzing is *better*... but how much better?
- How much does mutation based fuzzing depend on the input being mutated?
- We examine the case for PNG and libpng

Experiment 1

- We ran a mutation based fuzzer (similar to FILEfuzz) starting from 3 PNG's.
  - § 5 chunk types (most likely to be used by chance)
  - § 7 chunk types (unlikely to be used by chance)
  - § 9 chunk types (extremely unlikely)
- For each file, we tested the application with 100,000 test cases.

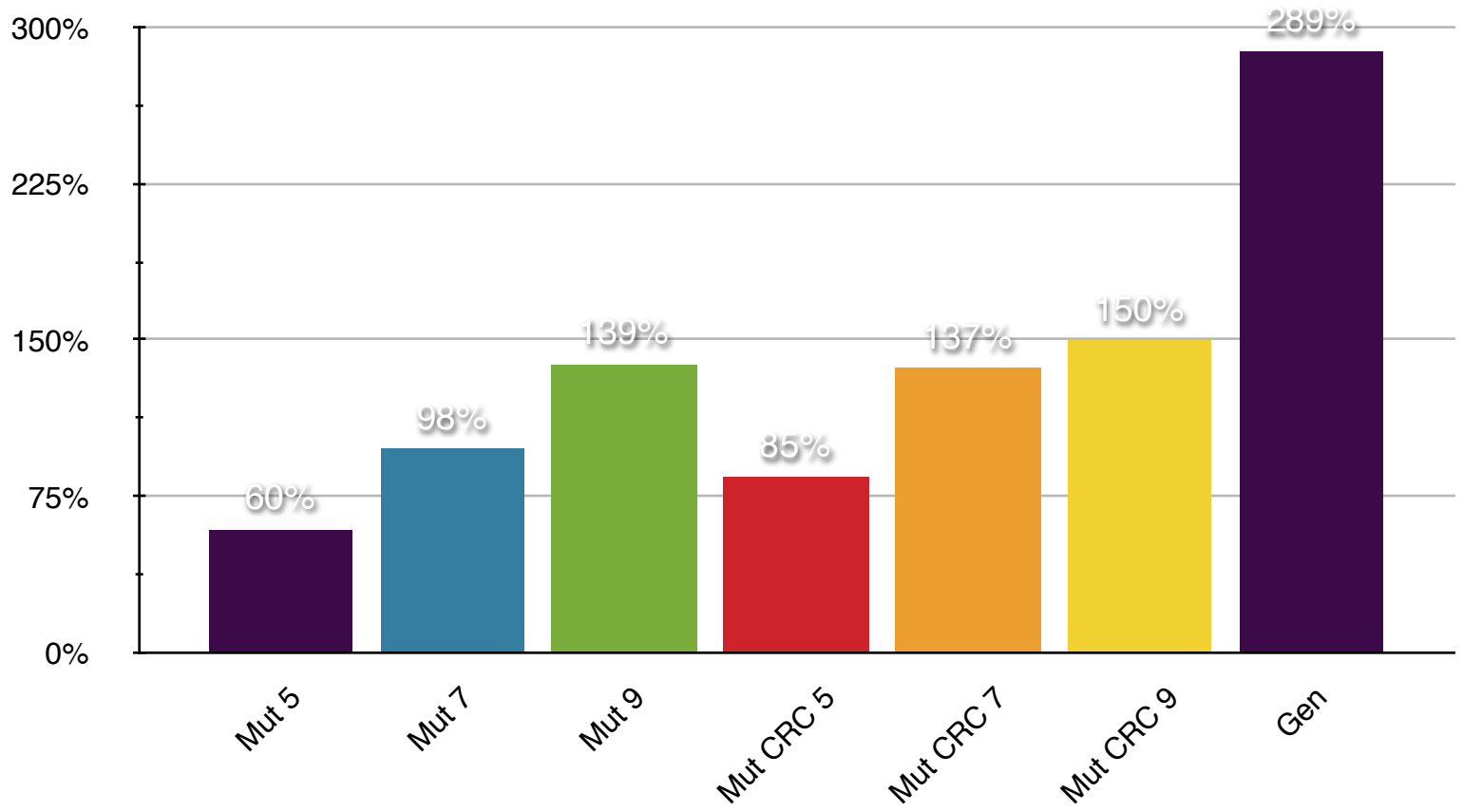
Experiment 2

- The existence of the CRC's may completely hinder the mutation-based fuzzer.
- We used the same starting file and same fuzzer as experiment 1.
- We ensured that the CRC's were all corrected before testing the application.
- Again used 100,000 test cases.

Experiment 3

- Used SPIKEfile and the PNG specification to generate fuzzed PNG's.
- Fuzzed all 21 chunk types as well as the length, CRC, and chunk name fields.
- Generated 29,511 test files.

Results



Number of lines executed as a percentage of code required to fuzz a minimal PNG file

Conclusions

- Mutation based fuzzing is very dependent on the inputs being mutated.
- Choosing the right inputs can double the amount of code executed with mutation based fuzzing.
- Generation based fuzzing is substantially better in this case
- In this case, 2-5 times more code may be executed using generation based fuzzing over mutation based.
- All this is specific to the fuzzers used and this specific filetype.

Does This Generalize?

- Who knows?
- Related information
  - § In “Fuzzing: Brute Force Vulnerability Discovery“, they examined 10,000 SWF files

SWF Version	% of Total
Flash 8	< 1%
Flash 7	2%
Flash 6	11%
Flash 5	55%
Flash 4	28%
Flash 1-3	3%

Questions?

- Please contact me at: cmiller@securityevaluators.com