Lecture 20: Fuzz Testing

17-355/17-665/17-819: Program Analysis

Rohan Padhye

April 5, 2022

* Course materials developed with Jonathan Aldrich and Claire Le Goues

Puzzle: Find `x` such `p1(x)` returns `True`

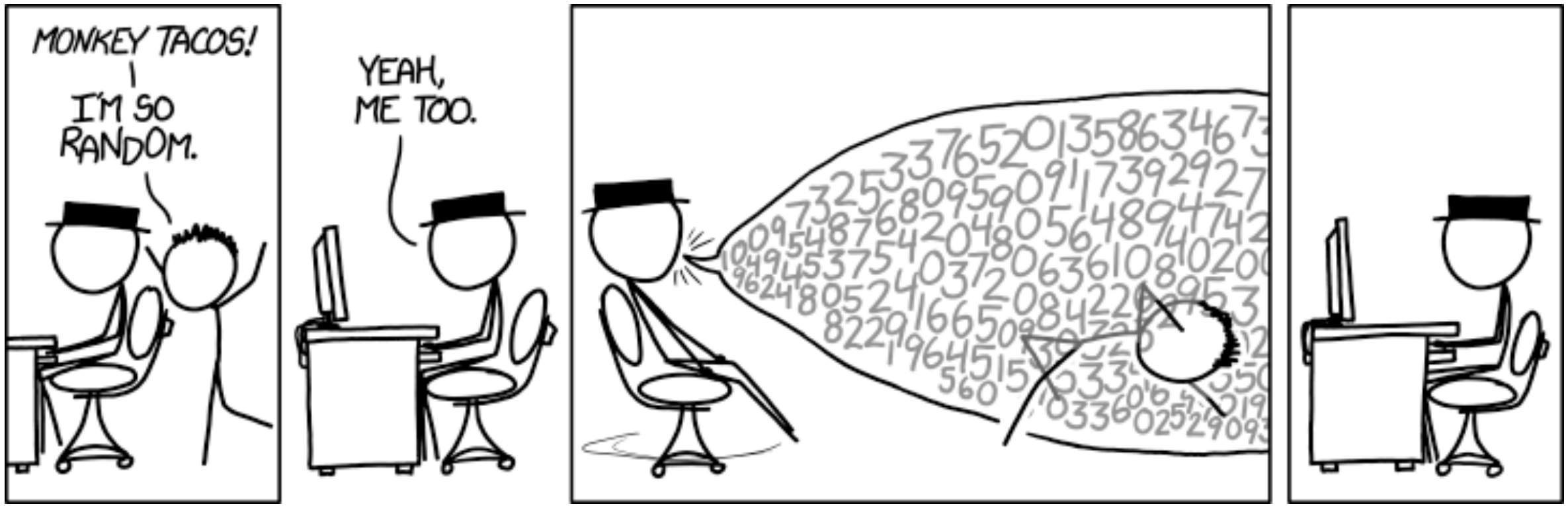
```
def p1(x):
    if x * x - 10 == 15:
        return True
    return False
```

Puzzle: Find `x` such `p2(x)` returns `True`

```
def p2(x):
    if x > 0 and x < 1000:
        if ((x - 32) * 5/9 == 100):
            return True
    return False
```

Puzzle: Find `x` such `p3(x)` returns `True`

```
def p3(x):
  if x > 3 and x < 100:
    z = x - 2
    c = 0
    while z >= 2:
      if z ** (x - 1) % x == 1:
        c = c + 1
      z = z - 1
    if c == x - 3:
      return True
  return False
```



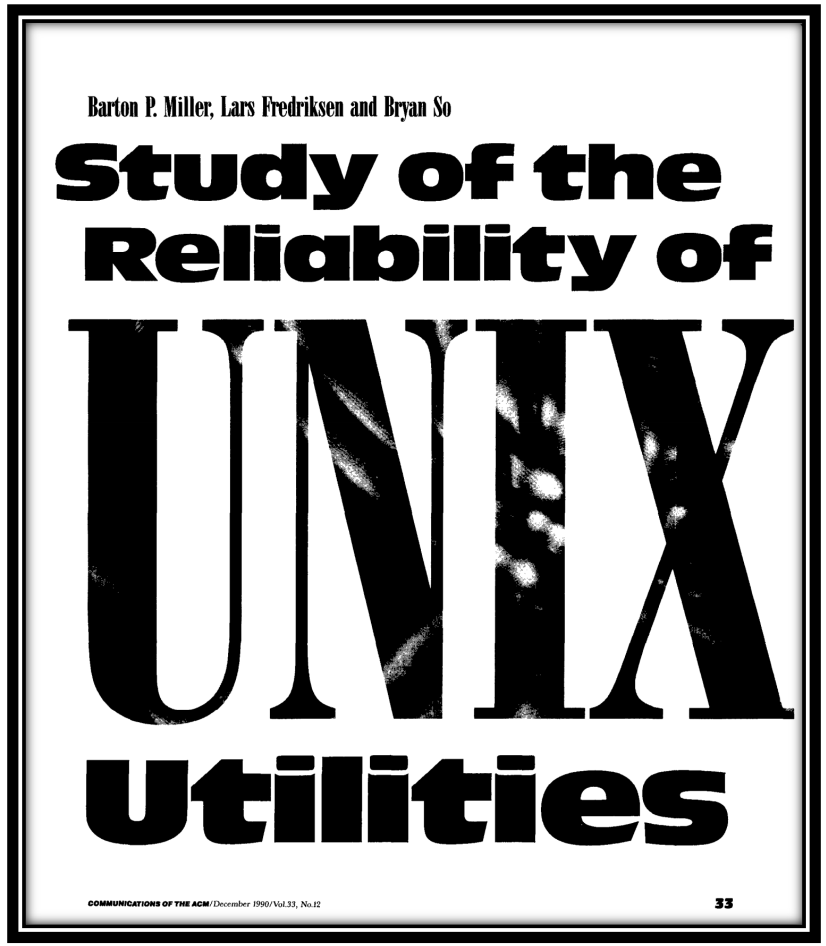
Original: https://xkcd.com/1210 CC-BY-NC 2.5

Fuzz Testing

Goal:
To find program inputs that reveal a bug

Approach:
Generate inputs randomly until program crashes

Fuzz Testing

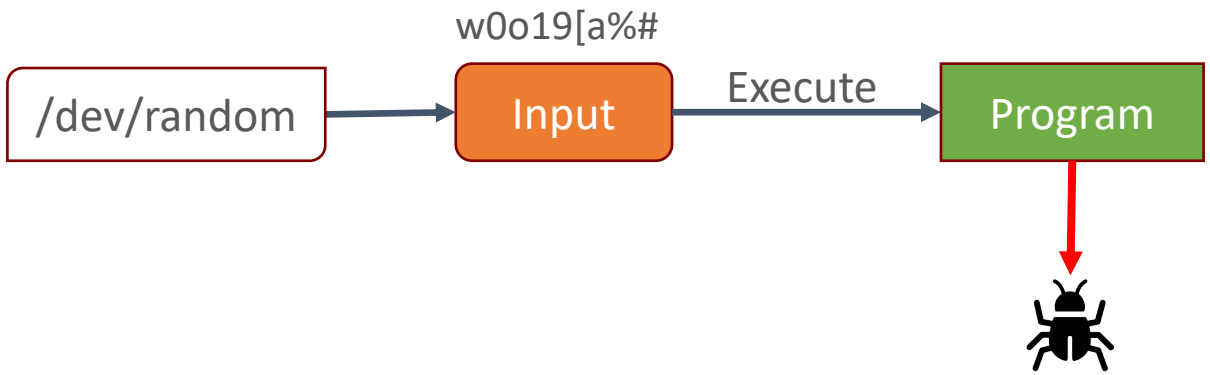


Communications of the ACM (1990)

> On a dark and stormy night one of the authors was logged on to his workstation on a dial-up line from home and the rain had affected the phone lines; there were frequent spurious characters on the line. The author had to race to see if he could type a sensible sequence of characters before the noise scrambled the command. This line noise was not surprising; but we were surprised that these spurious characters were causing programs to crash.

1990s

Fuzz Testing 101



1990 study found crashes in:
adb, as, bc, cb, col, diction, emacs, eqn, ftp, indent, lex, look, m4, make, nroff, plot, prolog, ptx, refer!, spell, style, tsort, uniq, vgrind, vi

Why do programs crash?

Common Fuzzer-Found Bugs

Causes: incorrect arg validation, incorrect type casting, executing untrusted code, etc.

Effects: buffer-overflows, memory leak, division-by-zero, use-after-free, assertion violation, etc. (“crash”)

Impact: security, reliability, performance, correctness

What are the **benefits**, **challenges**, & **limitations** of this approach?

Generate inputs randomly


```
$ ant –f build.xml
```

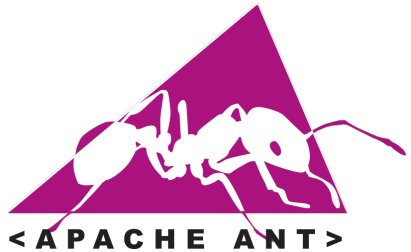
```
<project default=“dist”>
 <target name="init">
  <mkdir dir="${build}"/>
 </target>
 …
```

```
$ ant –f /dev/random
```

```
lrha3wn5p0w3uz;54 p0a23
rw3i 50a20 5a2y58a2p
y3wry3p285
q@P”uer9zparu9apur9qa3802
y5o2y 392r523a90wesu
```

Purely random data is not a very interesting input!!

Generate inputs randomly via mutation




```
$ ant –f build.xml
```

```
<project default=“dist”>
 <target name="init">
  <mkdir dir="${build}"/>
 </target>
 …
```

```
$ ant –f build.xml.mut
```

```
<project default=“dist”>
 <taWget name="init">
  <madir dir="2{build}"/@
 </tar?get>
 …
```

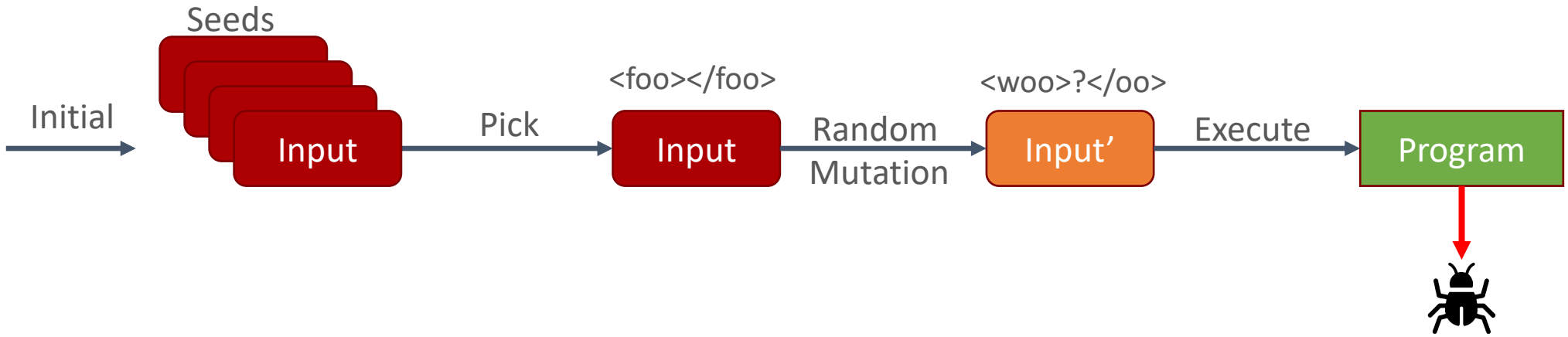
What are some good mutations?

Mutation Heuristics

- Binary input
  - Bit flips, byte flips
  - Change random bytes
  - Insert random byte chunks
  - Delete random byte chunks
  - Set randomly chosen byte chunks to *interesting* values e.g. INT_MAX, INT_MIN, 0, 1, -1, …
  - Other suggestions?
- Text input
  - Insert random symbols or keywords from a dictionary
  - Other suggestions?

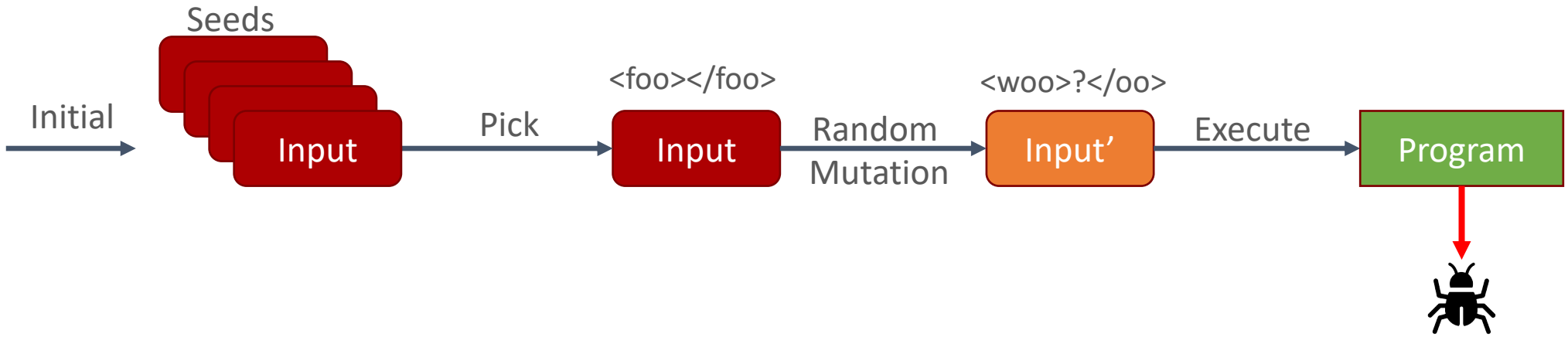
2000s

Mutation-Based Fuzzing (e.g. Radamsa, zzuf)



2000s

Mutation-Based Fuzzing (e.g. Radamsa, zzuf)



<APACHE ANT>

Valid Seed Input (build.xml)

```
<project default="dist">
 <target name="init">
  <mkdir dir="${build}"/>
 </target>
 …
```

New Input (Mutated from Seed)

```
<project default="dist">
 <taWget name="init">
  <madir dir="2{build}"/@
 </tar?get>
 …
```

What are the **benefits**, **challenges**, & **limitations** of this approach?

How do you know if you are making progress?
Can you think of some stopping criteria?

Code Coverage

LCOV - code coverage report

		Hit	Total	Coverage
Current view: top level - test	Lines:	6092	7293	83.5 %
Test: coverage.info	Functions:	481	518	92.9 %
Date: 2018-02-07 13:06:43				

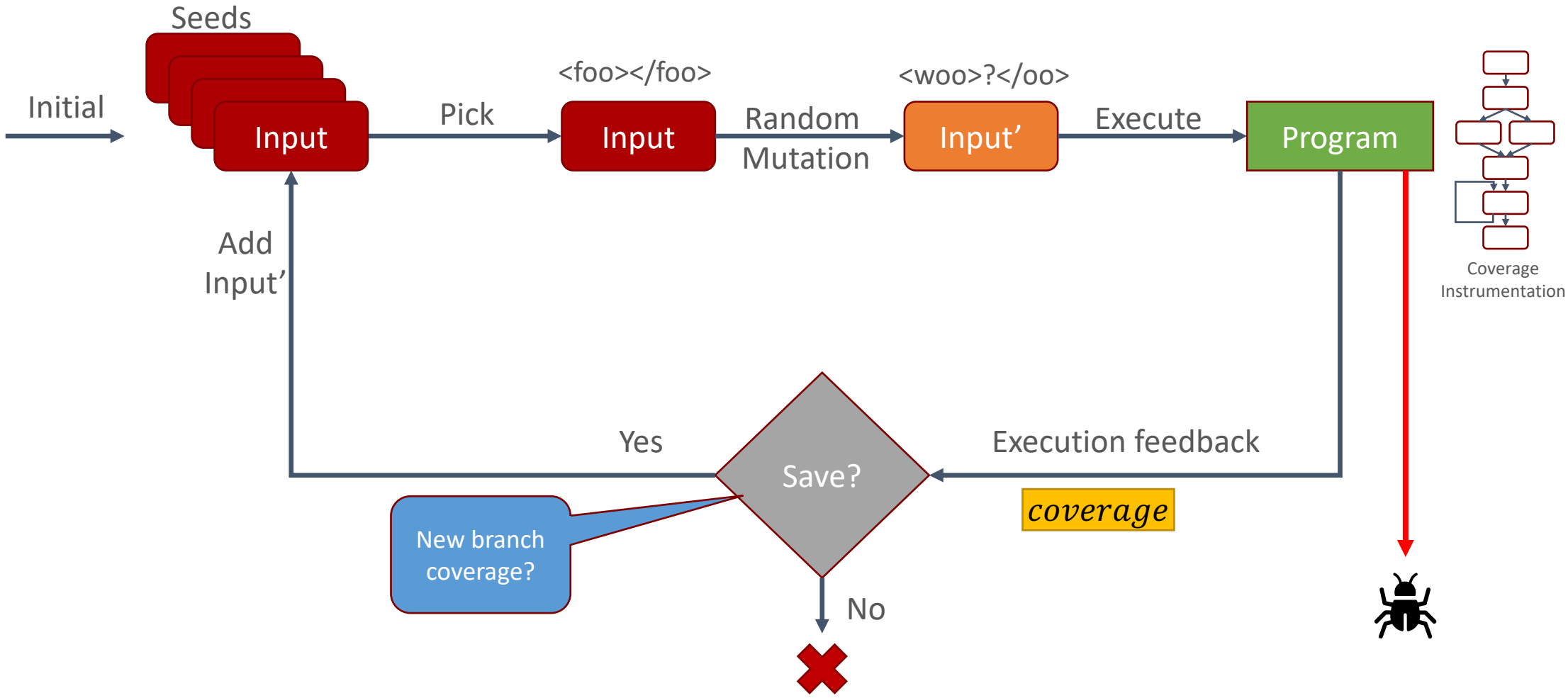
Filename	Line Coverage		Functions	
asn1_string_table_test.c	58.8 %	20 / 34	100.0 %	2 / 2
asn1_time_test.c	72.0 %	72 / 100	100.0 %	7 / 7
bad_dtls_test.c	97.6 %	163 / 167	100.0 %	9 / 9
bftest.c	65.3 %	64 / 98	87.5 %	7 / 8
bio_enc_test.c	78.7 %	74 / 94	100.0 %	9 / 9
bntest.c	97.7 %	1038 / 1062	100.0 %	45 / 45
chacha_internal_test.c	83.3 %	10 / 12	100.0 %	2 / 2
ciphername_test.c	60.4 %	32 / 53	100.0 %	2 / 2
crltest.c	100.0 %	90 / 90	100.0 %	12 / 12
ct_test.c	95.5 %	212 / 222	100.0 %	20 / 20
d2i_test.c	72.9 %	35 / 48	100.0 %	2 / 2
danetest.c	75.5 %	123 / 163	100.0 %	10 / 10
dhtest.c	84.6 %	88 / 104	100.0 %	4 / 4
drbgtest.c	69.8 %	157 / 225	92.9 %	13 / 14
dtls_mtu_test.c	86.8 %	59 / 68	100.0 %	5 / 5
dtlstest.c	97.1 %	34 / 35	100.0 %	4 / 4
dtlsv1listentest.c	94.9 %	37 / 39	100.0 %	4 / 4
ecdsatest.c	94.0 %	140 / 149	100.0 %	7 / 7
enginetest.c	92.8 %	141 / 152	100.0 %	7 / 7
evp_extra_test.c	100.0 %	112 / 112	100.0 %	10 / 10
fatalerrtest.c	89.3 %	25 / 28	100.0 %	2 / 2
handshake_helper.c	84.7 %	494 / 583	97.4 %	38 / 39
hmactest.c	100.0 %	71 / 71	100.0 %	7 / 7
ideatest.c	100.0 %	30 / 30	100.0 %	4 / 4
igetest.c	87.9 %	109 / 124	100.0 %	11 / 11
lhash_test.c	78.6 %	66 / 84	100.0 %	8 / 8
mdc2_internal_test.c	81.8 %	9 / 11	100.0 %	2 / 2
mdc2test.c	100.0 %	18 / 18	100.0 %	2 / 2
ocspapitest.c	95.5 %	64 / 67	100.0 %	4 / 4
packettest.c	100.0 %	248 / 248	100.0 %	24 / 24

```
      1  /    1:                 if ((err = SSLHashMD5.final(&hashCtx, &hashOut)) != 0)
      0  /    1:                     goto fail;
                :     }
                :     else {
                :             /* DSA, ECDSA - just use the SHA1 hash */
      0  /    1:             dataToSign = &hashes[SSL_MD5_DIGEST_LEN];
      0  /    1:             dataToSignLen = SSL_SHA1_DIGEST_LEN;
                :     }
                :
      1  /    1:     hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
      1  /    1:      hashOut.length = SSL_SHA1_DIGEST_LEN;
      1  /    1:      if ((err = SSLFreeBuffer(&hashCtx)) != 0)
      0  /    1:          goto fail;
                :
      1  /    1:      if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
      0  /    1:          goto fail;
      1  /    1:      if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
      0  /    1:          goto fail;
      1  /    1:      if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
      0  /    1:          goto fail;
      1  /    1:      if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
      0  /    1:          goto fail;
      1  /    1:          goto fail;
                :      if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                :          goto fail;
                :
                :     err = sslRawVerify(ctx,
                :                        ctx->peerPubKey,
                :                        dataToSign,                    /* plaintext *
                :                        dataToSignLen,                 /* plaintext l
                :                        signature,
                :                        signatureLen);
                :     if(err) {
                :             sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
                :                     "returned %d\n", (int)err);
                :             goto fail;
                :     }
                :
                : fail:
      1  /    1:      SSLFreeBuffer(&signedHashes);
      1  /    1:      SSLFreeBuffer(&hashCtx);
      1  /    1:      return err;
                :
      1  /    1: }
                :
```

Exercise: How to collect coverage?

```
if (x && y) {
   s1;
   s2;
} else {
   while(b) {
      s3;
    }
}
```

Coverage-Guided Fuzzing with AFL



2014+

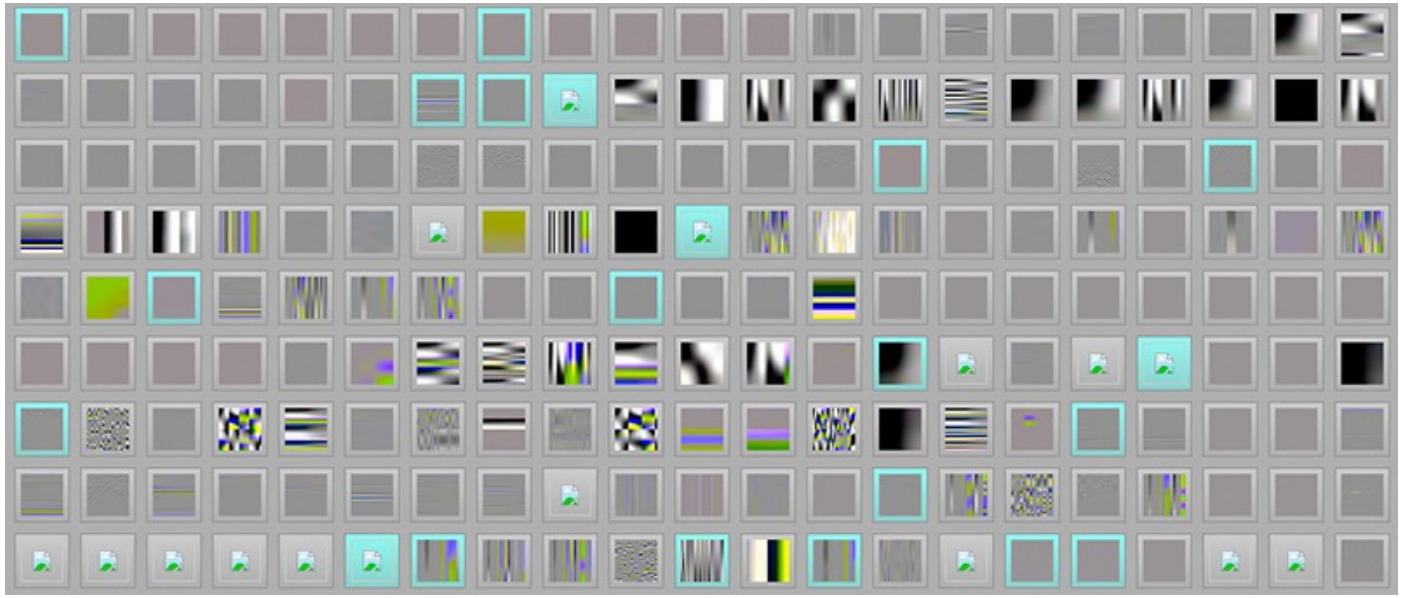
Coverage-Guided Fuzzing with AFL

November 07, 2014

Pulling JPEGs out of thin air

This is an interesting demonstration of the capabilities of afl; I was actually pretty surprised that it worked!

```
$ mkdir in_dir
$ echo 'hello' >in_dir/hello
$ ./afl-fuzz -i in_dir -o out_dir ./jpeg-9a/djpeg
```



2014+

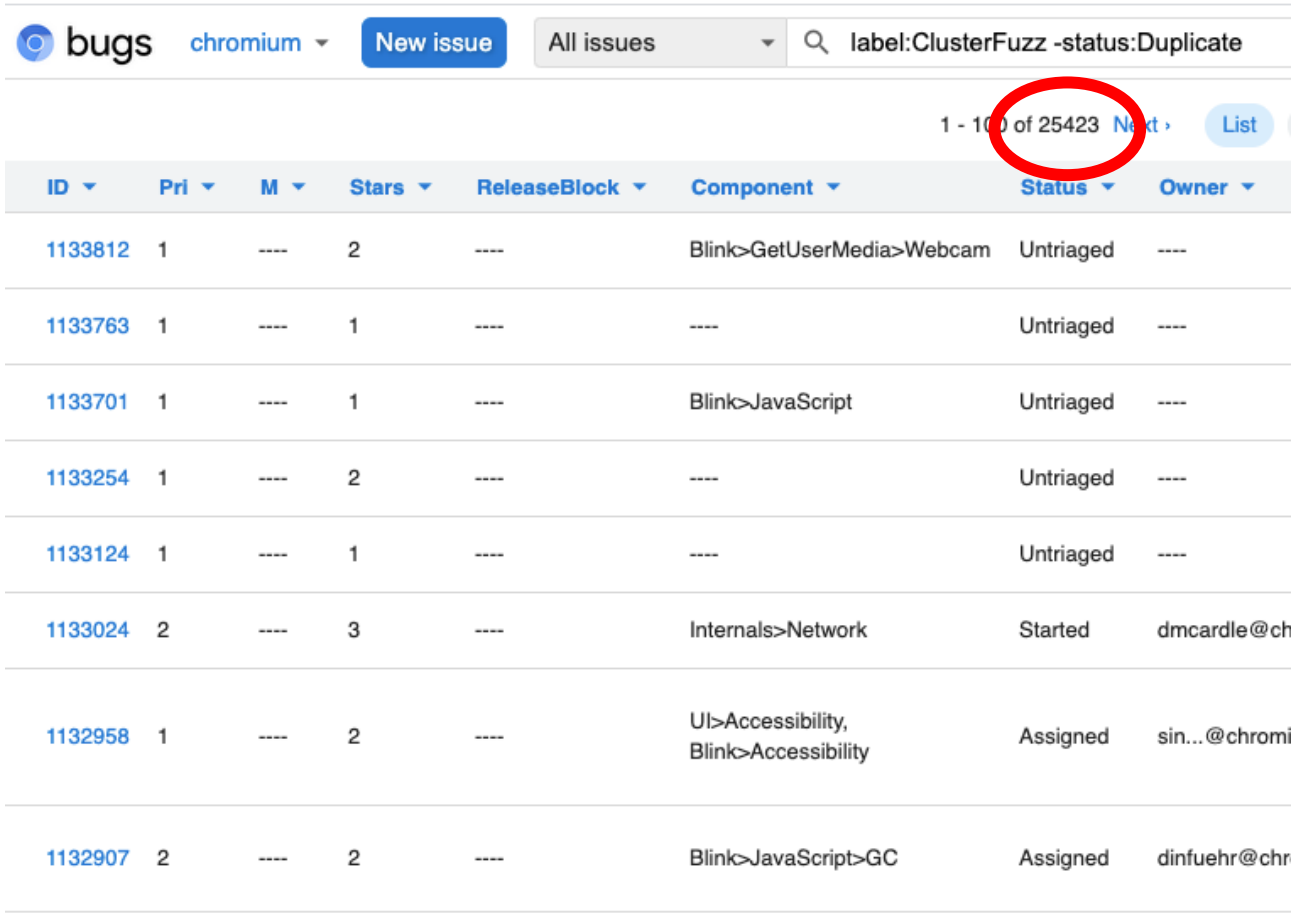
Coverage-Guided Fuzzing with AFL

The bug-o-rama trophy case

http://lcamtuf.coredump.cx/afl/

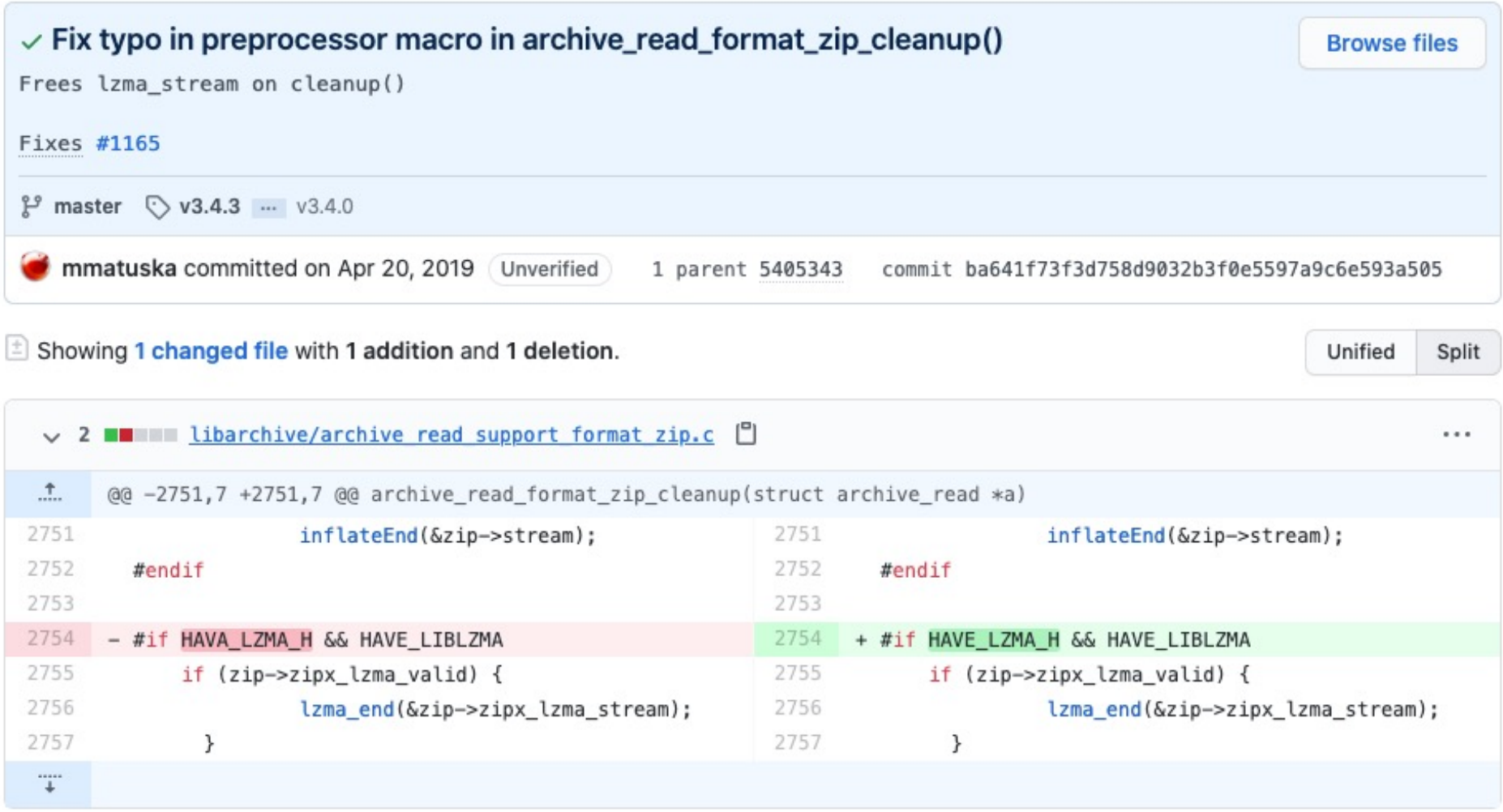
IJG jpeg [1]	libjpeg-turbo [1] [2]	libpng [1]
libtiff [1] [2] [3] [4] [5]	mozjpeg [1]	PHP [1] [2] [3] [4] [5] [6] [7] [8]
Mozilla Firefox [1] [2] [3] [4]	Internet Explorer [1] [2] [3] [4]	Apple Safari [1]
Adobe Flash / PCRE [1] [2] [3] [4] [5] [6] [7]	sqlite [1] [2] [3] [4]...	OpenSSL [1] [2] [3] [4] [5] [6] [7]
LibreOffice [1] [2] [3] [4]	poppler [1] [2]...	freetype [1] [2]
GnuTLS [1]	GnuPG [1] [2] [3] [4]	OpenSSH [1] [2] [3] [4] [5]
PuTTY [1] [2]	ntpd [1] [2]	nginx [1] [2] [3]
bash (post-Shellshock) [1] [2]	tcpdump [1] [2] [3] [4] [5] [6] [7] [8] [9]	JavaScriptCore [1] [2] [3] [4]
pdfium [1] [2]	ffmpeg [1] [2] [3] [4] [5]	libmatroska [1]
libarchive [1] [2] [3] [4] [5] [6] ...	wireshark [1] [2] [3]	ImageMagick [1] [2] [3] [4] [5] [6] [7] [8] [9] ...
BIND [1] [2] [3] ...	QEMU [1] [2]	lcms [1]

ClusterFuzz @ Chromium



ID	Pri	M	Stars	ReleaseBlock	Component	Status	Owner
1133812	1	----	2	----	Blink>GetUserMedia>Webcam	Untriaged	----
1133763	1	----	1	----	----	Untriaged	----
1133701	1	----	1	----	Blink>JavaScript	Untriaged	----
1133254	1	----	2	----	----	Untriaged	----
1133124	1	----	1	----	----	Untriaged	----
1133024	2	----	3	----	Internals>Network	Started	dmcardle@ch
1132958	1	----	2	----	UI>Accessibility, Blink>Accessibility	Assigned	sin...@chromi
1132907	2	----	2	----	Blink>JavaScript>GC	Assigned	dinfuehr@chr

Libarchive#1165 (CVE-2019-11463)



✓ **Fix typo in preprocessor macro in archive_read_format_zip_cleanup()**

Browse files

Frees lzma_stream on cleanup()

Fixes #1165

master v3.4.3 ··· v3.4.0

mmatuska committed on Apr 20, 2019 Unverified 1 parent 5405343 commit ba641f73f3d758d9032b3f0e5597a9c6e593a505

Showing 1 changed file with 1 addition and 1 deletion. Unified Split

2 libarchive/archive_read_support_format_zip.c

```
@@ -2751,7 +2751,7 @@ archive_read_format_zip_cleanup(struct archive_read *a)
2751                 inflateEnd(&zip->stream);                2751                 inflateEnd(&zip->stream);
2752   #endif                                                 2752   #endif
2753                                                          2753
2754 - #if HAVA_LZMA_H && HAVE_LIBLZMA                        2754 + #if HAVE_LZMA_H && HAVE_LIBLZMA
2755         if (zip->zipx_lzma_valid) {                      2755         if (zip->zipx_lzma_valid) {
2756                 lzma_end(&zip->zipx_lzma_stream);        2756                 lzma_end(&zip->zipx_lzma_stream);
2757           }                                              2757           }
```

Easy to fix,
hard to find!!

Challenging Problems

- Fuzzing heuristics
  - Mutation: Which input to mutate? How many times? Which mutations?
  - Feedback: What to instrument? How to keep overhead low?
- Oracles
  - What is a bug? Crash? Silent overflow? Infinite loop? Race condition? Undefined behavior? How do we know when we have found a bug?
- Debugging
  - Reproducibility
  - Crash triaging
  - Input minimization
- Fuzzing roadblocks
  - Magic bytes, checksums (see PNG, SSL)
  - Dependencies in binary inputs (e.g. length of chunks, indexes into tables – see PNG)
  - Inputs with complex syntax and semantics (e.g. XML, JSON, C++)
  - Stateful applications

Oracles: Sanitizers

- Address Sanitizer (ASAN) ***
- LeakSanitizer (comes with ASAN)
- Thread Sanitizer (TSAN)
- Undefined-behavior Sanitizer (UBSAN)

https://github.com/google/sanitizers

AddressSanitizer

```
int get_element(int* a, int i) {
  return a[i];
}
```

```
int get_element(int* a, int i) {
  if (a == NULL) abort();
  return a[i];
}
```

```
int get_element(int* a, int i) {
  if (a == NULL) abort();
  region = get_allocation(a);
  if (in_stack(region)) {
    if (popped(region)) abort();
    …
  }
  if (in_heap(region)) { ... }
  return a[i];
}
```

```
int get_element(int* a, int i) {
  if (a == NULL) abort();
  region = get_allocation(a);
  if (in_heap(region)) {
    low, high = get_bounds(region);
    if ((a + i) < low || (a +i) > high) {
      abort();
    }
  }
  return a[i];
}
```

AddressSanitizer

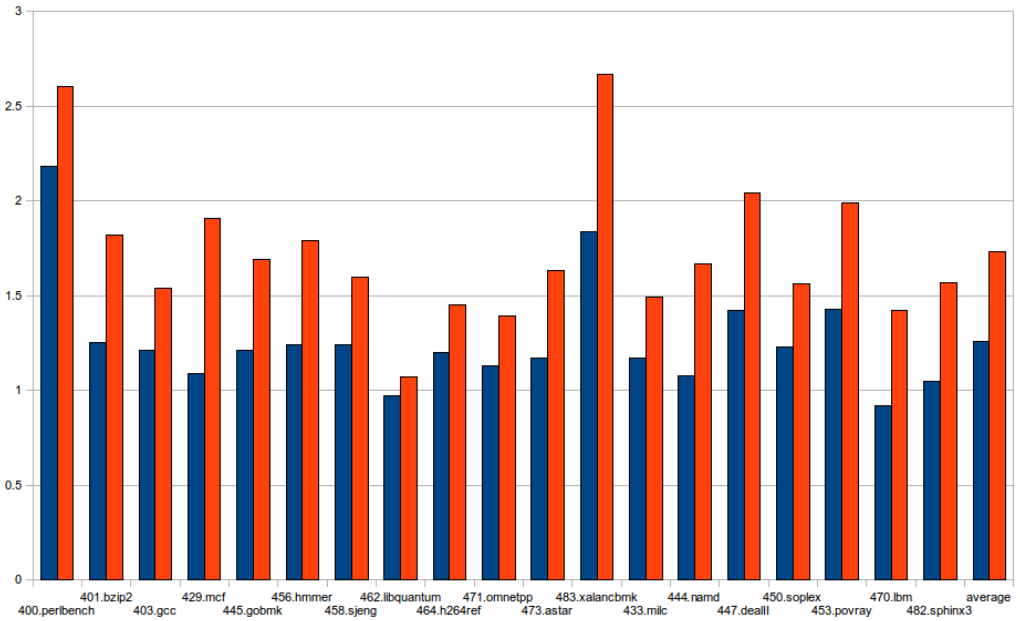
https://github.com/google/sanitizers/wiki/AddressSanitizer

Compile with `clang –fsanitize=address`

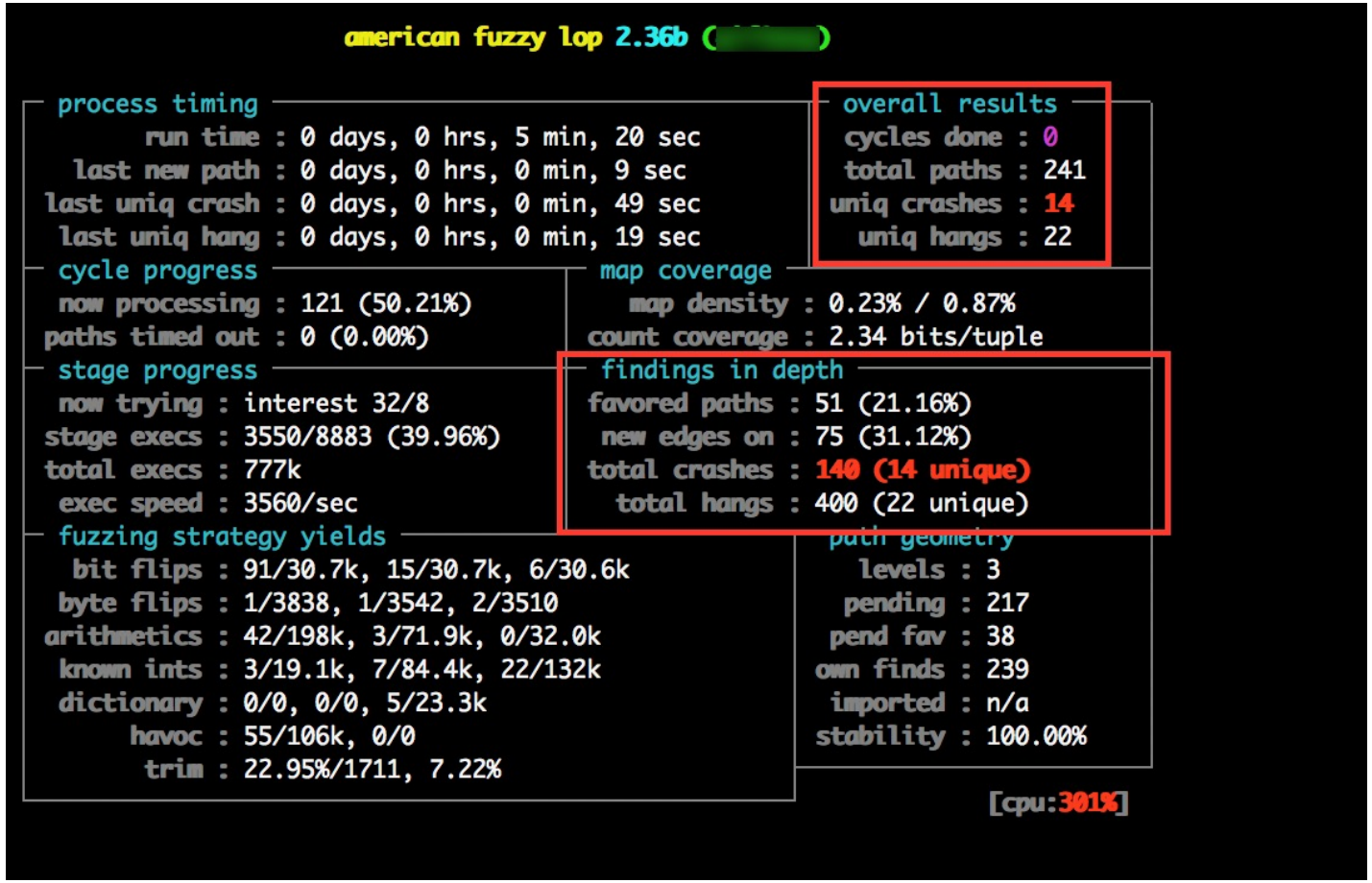
Asan is a memory error detector for C/C++. It finds:

- Use after free (dangling pointer dereference)
- Heap buffer overflow
- Stack buffer overflow
- Global buffer overflow
- Use after return
- Use after scope
- Initialization order bugs
- Memory leaks

Slowdown on SPEC CPU 2006



Crash Triaging



Crash Triaging

- Given two crashing inputs x1 and x2, do they trigger the same bug?
- *Very* difficult to answer in practice
- Herustics: bug(x1) = bug(x2) only if.... (consider pros/cons of each)
  - exitcode(x1) = exitcode(x2) // or exception or error msg
  - coverage(x1) = coverage(x2)
  - stacktrace(x1) = stacktrace(x2)
  - newcoverage(x1, old) = newcoverage(x2, old) // AFL
  - fix(x1) = fix(x2)

Open Problems – Research Opportunities!

- What if fuzzing doesn't find any bugs after X hours?
  - Is the program bug free?
    - **RQ: What is the probability that there are more bugs lurking around?**
  - Should we keep fuzzing?
    - **RQ: When should we stop to balance cost vs. results?**
  - Can we change the feedback function? Mutation?
    - **RQ: What changes can we make? How can we bring a human in the loop?**
- How to balance instrumentation overhead with feedback quality?
  - **RQ: What parts of the code should be instrumented?**
- How to generate *meaningful* test cases?
  - **RQ: What is "meaningful"?**
  - **RQ: How to generate good inputs by construction?**