Smarter fuzzing using sound and precise static analyzers

Pascal Cuoq, Chief Scientist, TrustInSoft

January 31, 2017

Basis of this presentation:

Result graphs for an abstract interpretation-based static analyzer
JFLA 2017

American Fuzzy Lop (afl-fuzz)

Work of Michal Zalewski (lcamtuf)

Now a constellation of tools, some contributed by others

Easy to use

- compile target with afl-gcc (instrumenting compiler)
- let afl-fuzz discover input files that explore program behaviors

American Fuzzy Lop (afl-fuzz)

Work of Michal Zalewski (lcamtuf)

Now a constellation of tools, some contributed by others

Easy to use

- compile target with afl-gcc (instrumenting compiler)
- let afl-fuzz discover input files that explore program behaviors

Genetic: new inputs are generated by random mutations of existing inputs

Example 1: the ASN.1 parser libksba

afl-fuzz generates a 1KB file.

Parsed as a X.509 certificate on a 128GiB workstation, parser

- allocates 60GiB
- several seconds to initialize this memory

Denial of Service
(details: http://www.openwall.com/lists/oss-security/2016/08/20/3)

Example 2: mbedTLS ASN.1 parser

```
int main(int c, char*v[]) {
...
int ret = mbedtls_x509_crt_parse_file( &crt, v[1] );
if( ret != 0 ) {
    printf( "failed\n" );
  }
else {
  printf( "success\n" );
  uint32_t flags;
  ret = mbedtls_x509_crt_verify( &crt, &crt, NULL, NULL, &flags
}
```

Function `crt_parse_file` is very well-tested by AFL-generated inputs

Example 2: mbedTLS ASN.1 parser

```
int main(int c, char*v[]) {
...
int ret = mbedtls_x509_crt_parse_file( &crt, v[1] );
if( ret != 0 ) {
    printf( "failed\n" );
  }
else {
  printf( "success\n" );
  uint32_t flags;
  ret = mbedtls_x509_crt_verify( &crt, &crt, NULL, NULL, &flags
}
```

Function `crt_parse_file` is very well-tested by AFL-generated inputs
Problem: only a few tests find their way through to `crt_verify`

First example: difficult to reproduce: requires 60GiB or RAM

First example: difficult to reproduce: requires 60GiB or RAM

Solution: whole-program dependency analysis

Identifies bytes in input file that are decoded into a size passed to `calloc`

First example: difficult to reproduce: requires 60GiB or RAM

Solution: whole-program dependency analysis

Identifies bytes in input file that are decoded into a size passed to `calloc`

Second example: adequate coverage requires too much patience

First example: difficult to reproduce: requires 60GiB or RAM

Solution: whole-program dependency analysis

Identifies bytes in input file that are decoded into a size passed to `calloc`

Second example: adequate coverage requires too much patience

Solution: whole-program dependency analysis

Identifies input bytes that can be changed while still passing validation
focus on exploring second phase of program, with each testcase

Whole-program dependencies analysis

- data dependencies: `y` is computed from `z`
- path dependencies: `y` is computed differently depending on `c`
- address dependencies: `y` is different depending where `p` points

```
if (c)
  y = 0;
else
  y = z + *p;

int res = f();

if (res)
  error();
else
  g(y);
```

ABS as a metaphor for sound static analysis

The Anti-lock Breaking System initially was mostly reserved to aircrafts
Concorde sported the first fully electronic ABS system

ABS as a metaphor for sound static analysis

The Anti-lock Breaking System initially was mostly reserved to aircrafts
Concorde sported the first fully electronic ABS system

Then race cars

ABS as a metaphor for sound static analysis

The Anti-lock Breaking System initially was mostly reserved to aircrafts
Concorde sported the first fully electronic ABS system

Then race cars

Then everybody's car

ABS as a metaphor for sound static analysis

The Anti-lock Breaking System initially was mostly reserved to aircrafts
Concorde sported the first fully electronic ABS system

Then race cars

Then everybody's car

Fuzzing does not guarantee to find all bugs
Wouldn't do a precise, sound dependency analysis just to help with fuzzing
But if you have such an analysis, it really helps