Fuzzing Suricata:

Finding Vulnerabilities in Large Projects

Sirko Höer



Special thanks





- Victor Julien, lead developer suricata, Open Infosec Foundation (OISF)
- Henning Perl, CTO, Code Intelligence
- Sergej Dechand, CEO, Code Intelligence





Disclaimer

- The methodologie mentioned here reflect my experiences about fuzz-testing and are not general approaches used by the BSI!
- All opinions about Fuzzing expressed in this presentation are my opinions only. They
 are not the general opinion of the BSI!

Table of content

- Introduction
- Methodology of Fuzzing Suricata
- Example: Ethernet Decoder Heap Buffer Overflow
- Conclusion

What is suricata

- Open Source IDS / IPS / NSM
- Developed by Open Information Security Foundation
- written in **c** and **rust**
- Buildsystem: automake
- Version 5.0.3

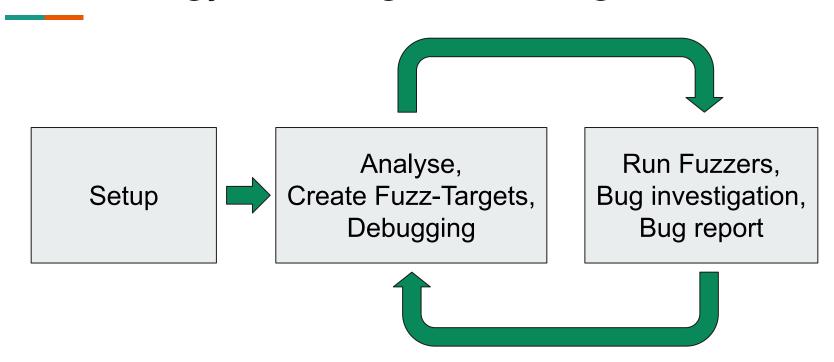


Facts:

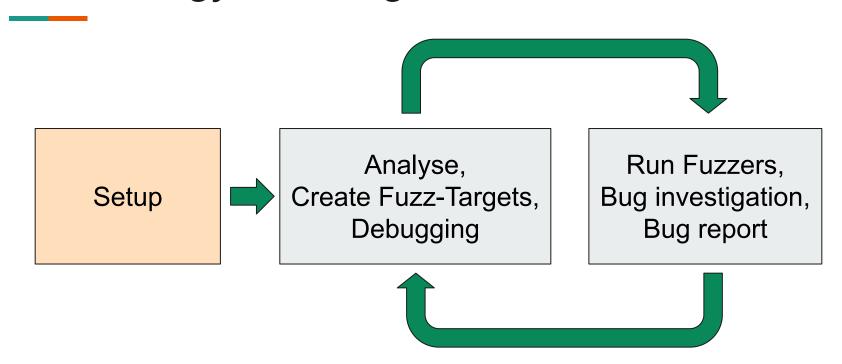
- about 600k lines of code
- more than 600 source files
- Uses Unit Tests and AFL Fuzzing (2019)
- since March 2020 integration to google oss-fuzz



Methodology of Fuzzing Projects in general



Methodology of Fuzzing Suricata



Build process

How to build the project?

\$ user@host ./autogen \$ user@host ./configure

\$ user@host make all -j\$(nproc)

Build the project

Build process

- How to build the project?
- Create build script with sanitzer
 - o patch object file
 - o pack all together

Build parent

Create build script

```
#!/usr/bin/sh
export CC=clang
export CXX=clang++
export ASAN_OPTIONS=detect leaks=0
./configure --disable-rust CFLAGS="-01 -v -g -fPIC \
-fsanitize-coverage=indirect-calls, trace-cmp, trace-div, trace-gep \
-fsanitize=address,fuzzer-no-link,undefined,signed-integer-overflow,bool,pointer-o
LDFLAG="-fsanitize=address, fuzzer-no-link, undefined, signed-integer-overflow, bool, p
ointer-overflow \
-fsanitize-coverage=indirect-calls, trace-cmp, trace-div, trace-gep" \
make -j$(nproc)
echo "patch suricata.o ..."
sed -i -e 's/main/mmmm/g' suricata.o
echo "patched ..."
echo "generate archiv suricata fuzz.a ..."
ar rv suricata_fuzz.a *.o
echo "generated ..."
```

build_suricata.sh

Build process

- How to build the project?
- Create build script with sanitzer
- Build fuzzing infrastructure (Docker)

```
#install main packages
RUN echo "installing basic packages ..."

RUN pacman -Syu --noconfirm
RUN pacman -S --noconfirm \
screen \
git \
...

# install dep for suricata
RUN pacman -Syu --noconfirm
RUN pacman -Sy--noconfirm
RUN pacman -Su--noconfirm \
jansson \
libnet \
libyaml \
nss
...

Dockerfile
```

Build parent

Create build script

Create infra.

Build process

- How to build the project?
- Create build script with sanitzer
- Build fuzzing infrastructure (Docker)
- Build Makefile for fuzz-targets

```
FUZZERS =
            fuzz_app \
            fuzz decoder udp
fuzz_%: $(src_target)/fuzz_%.c
    $(shell mkdir -p $(build target)/$*)
    clang -01 -g \
        $< \
        ${CFLAGS} ${LDFLAGS} \
        -DCLS=64 \
        -D HAVE MAGIC \
        -I../src -I../libhtp \
        -fsanitize=fuzzer,address,undefined, \
        signed-integer-overflow, \
        bool, pointer-overflow \
        -fsanitize-coverage=trace-pc-guard \
        -fsanitize-thread-memory-access \
        -lstdc++ \
        -lmagic -lcap-ng -lpcap -lpthread -lnet -lyaml -lpcre -lz -llzma \
        ../src/suricata fuzz.a \
        ../libhtp/htp/.libs/libhtp.a \
        /usr/lib/liblz4.so \
        -o $(build_target)/$*/$@
                                                                  Makefile
```

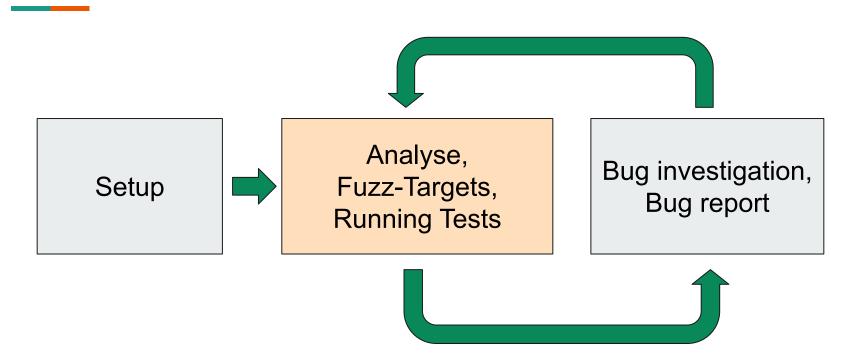
Build parent

Create build script

Create infra.

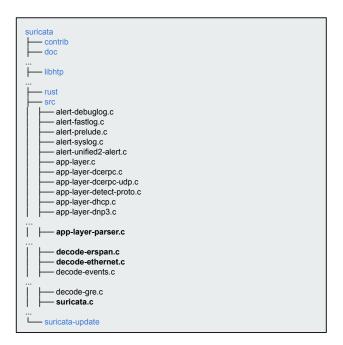
Makefile

Methodology of Fuzzing Suricata



Biggest challenge: Finding entry points

- Look at Unit-Test
- Look at Fuzz-Test (if fuzz-tests are available)
- Look at the Bug-Tracker

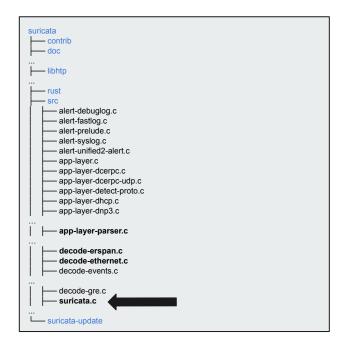


Biggest challenge: Finding entry points

- Look at Unit-Test
- Look at Fuzz-Test (if fuzz-tests are available)
- Look at the Bug-Tracker

In this case, I look at:

- Entrypoint : src/suricata.c
- Unit-Tests: src/tests/*.c



Find interesting things

```
1183
       static void ParseCommandLineAFL(const char *opt_name, char *opt_arg){
1184
       #ifdef AFLFUZZ_RULES
          if(strcmp(opt_name, "afl-rules") == 0) {
1185
              MpmTableSetup();
              SpmTableSetup();
              exit(RuleParseDataFromFile(opt_arg));
          } else
       #endif
       #ifdef AFLFUZZ APPLAYER
          if(strcmp(opt_name, "afl-http-request") == 0) {
              //printf("arg: //%s\n", opt_arg);
              MpmTableSetup();
              SpmTableSetup();
              AppLayerProtoDetectSetup();
              AppLayerParserSetup();
              RegisterHTPParsers();
              exit(AppLayerParserRequestFromFile(
                                                                PROTO_HTTP, opt_arg));
1268
                                                                                   suricata.c
```

Fuzz-targets for AFL:

- About 25 written Targets (2019)
- App-Layer:
 - http-request / response
 - smb-request / response
 - smtp
- Low-Level decoder
 - IPv4 / IPv6 decoder
 - PPP-decoder
 - Ethernet-decoder

Finding good Fuzz-Targets

- RuleParseDataFromFile(...)
- AppLayerParserRequestFromFile(...)
- AppLayerParserFromFile(...)
- MimeParserDataFromFile(...)
- DecoderParseDataFromFile(...)
- DerParseDataFromFile(...)
- ConfYamlLoadString(...)

Example:

```
int RuleParseDataFromFile(char *filename)
 SigTableSetup();
  SCReferenceConfInit():
  SCClassConfInit():
 DetectEngineCtx *de_ctx = DetectEngineCtxInit();
 while (__AFL_LOOP(10000)) {
   size_t result = fread(&buffer, 1, sizeof(buffer), fp);
   if (result < sizeof(buffer)) {
     buffer[result] = '\0';
      Signature *s = SigInit(de_ctx, buffer);
      if (s!= NULL) {
        SigFree(s);
 DetectEngineCtxFree(de_ctx);
  SCClassConfDeinit();
  SCReferenceConfDeinit():
                                                                      decoder-afl.c
```

The Fuzz-Target (Example)

Creating Fuzz-Targets

```
if(strcmp(opt_name, "afl-http-request") == 0) {
       //printf("arg: //%s\n", opt_arg);
       MpmTableSetup();
       SpmTableSetup();
       AppLayerProtoDetectSetup()
       AppLayerParserSetup();
       RegisterHTPParsers();
       exit(AppLayerParserRequestFromFile(IPPROTO_TCP, ALPROTO_HTTP, opt_arg));
   } else if(strcmp(opt_name, "afl-tls") == 0) {
       MpmTableSetup();
       SpmTableSetup();
       AppLayerProtoDetectSetup();
       AppLayerParserSetup();
       RegisterSSLParsers();
       exit(AppLayerParserFromFile(IPPROTO_TCP, ALPROTO_TLS, opt_arg));
                                                                     suricata.c
```

```
/* Include files */
int LLVMFuzzerInitialize(nt *argc, char ***argv) {
 MpmTableSetup();
 SpmTableSetup();
 AppLayerProtoDetectSetup();
 AppLayerParserSetup();
 AppLayerParserRegisterProtocolParsers();
 /* Initialize random number generator */
 srand(0);
 return 0;
AppProto AppProtoFromData() {
 return rand() % ALPROTO MAX;
                                                 fuzz app.c
```

The Fuzz-Target (Example)

Creating Fuzz-Targets

```
int AppLayerParserRequestFromFile(uint8_t ipproto, AppProto alproto, char *filename) {
  Flow *f = NULL;
  TcpSession ssn;
  AppLayerParserThreadCtx *alp tctx = AppLayerParserThreadCtxAlloc();
  f->flags |= FLOW IPV4;
  f - > src.addr data32[0] = 0x01020304
  f->dst.addr data32[0] = 0x0506070
  f->sp = 10000;
  f->dp = 80;
  f->alproto = alproto;
  uint8 t buffer[ 65536];
  uint32 t cnt = 0;
  uint8 t flags = STREAM TOSERVER;
  if (start--) {
    flags |= STREAM START;
  if (done) {
    flags |= STREAM EOF;
 (void)AppLayerParserParse( NULL, alp tctx, f, alproto, flags, buffer, size);
                                                              app-layer-parser.c
```

```
int LLVMFuzzerTestOneInput(const uint8 t *data, size t size) {
if (size < 1) return 0;
// Data setup
AppProto alproto = AppProtoFromData(*data);
Flow *f = NULL;
TcpSession ssn;
AppLayerParserThreadCtx *alp_tctx = AppLayerParserThreadCtxAlloc();
 f->flags |= FLOW IPV4;
 f - > src.addr data32[0] = 0x01020304;
 f->dst.addr data32[0] = 0x05060708;
 f->sp = 10000;
 f->dp = 80;
 f->alproto = alproto;
int start = 1;
int flip = 0;
uint8_t flags = STREAM_TOSERVER | STREAM_START;
AppLayerParserParse(NULL, alp tctx, f, alproto, flags, data, size);
                                                                      fuzz_app.c
```

Methodology of Fuzzing Suricata

Creating Fuzz-Targets: The conclusion

- Reuse of written code
- Customize and generalize the Fuzz Targets
- Think about Seeds and dictionaries
 - Configuration Parser use YAML
 - Decoder needs network packages
 - Rule parser needs rule-files
- Run the fuzzer

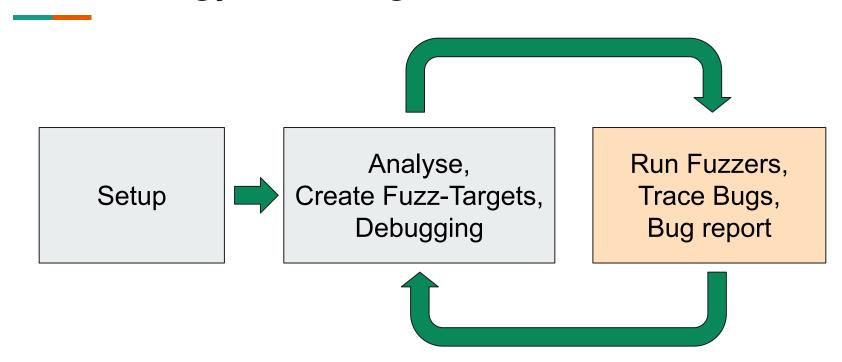
```
NN PC: 0.03bcc104-076 in LiMPsuzertestoncingut /homo/sirso/Projects/CI/Vazizna/paricata/paricata/sizzing/itazzing/itazzing/itaz-app.:39-7
NN PC: 100 Tr. 128 Gorge 2,5806 libit 17 Gorce/16 Pros 1506 to 17/17 BS: 2 Fresheltes-Changelit-Crossofter-Appelit-Crossofter-Appelit-Crossofter-Appelit-Crossofter-Appelit-Crossofter-Appelit-Crossofter-Appelit-Crossofter-Appelit-Crossofter-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appelit-Appe
     ==12314==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x6020003bdbd4 at pc 0x55bc668778ed bp 0x7ffe0005b050 sp 0x7ffe0005b048
    READ of size 1 at 0x6020003bdbd4 thread TO
               #0 0x55bc668778ec in BasicSearch /home/sirko/Projects/CI/fuzzing/suricata/suricata-fuzzing-gitlab/src/util-spm-bs.c:65:13
               #1 0x55bc665lacb8 in SSHParseBanner /home/sirko/Projects/CI/fuzzing/suricata/suricata-fuzzing-gitlab/src/app-layer-ssh.c:99:26
#2 0x55bc665lacb8 in SSHParseBecord /home/sirko/Projects/CI/fuzzing/suricata/suricata-fuzzing-gitlab/src/app-layer-ssh.c:220:15
            #10 05566509407 in Tuzzer: https://doi.org/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.1009/10.
               #14 0x55bc66378ecd in start (/home/sirko/Projects/CI/fuzzing/suricata/suricata-fuzzing-gitlab/fuzzing/playground/app/fuzz app+0x6d0ecd)
                               3bdbd4 is located 1 bytes to the right of 3-byte region [0x6020003bdbd0,0x6020003bdbd3]
           #B 0x7f0734f99ce2 in libc start main (/usr/lib/libc.so.6+0x23ce2)
     SUMMARY: AddressSanitizer: heap-buffer-overflow /home/sirko/Projects/CI/fuzzing/suricata/suricata-fuzzing-gitlab/src/util-spm-bs.c:65:13 in BasicSearch
  Surpour: Address saint let begin der Frei over four / mome/sixo/rroject.

Stadow lyres o sound the begin did ress.

Stadow lyres o sound the begin did ress.

And the first fi
     waysca48006fb70. to tolo31to to to so
           Shadow byte legend (one shadow byte represents 8 application bytes)
Addressable: 80
        Partially addressable: 01 02 03 04 05 06 07
Heap left redzone: fa
         Freed heap region:
Stack left redzone:
           Stack mid redzone:
           Stack right redzone:
           Stack use after scope
           Global init order:
            Poisoned by user:
         Container overflow:
           Intra object redzone
            ASan internal:
         Left alloca redzone:
Right alloca redzone:
           Shadow gap:
  MS: 1 ChangeBinInt-; base unit: 51c70aaaf741b05bd92e7b623f1afc49f76fd316
    \x55\x65\x99\x89
artifact prefix='./'; Test unit written to ./crash-418ddc91da8e43261cfeca4b8a244d8d52e2187
```

Methodology of Fuzzing Suricata



- Targeting the decoder of an ethernet packet
- Found bug in: decoder-ethernet.c
- Heap Buffer Overflow (read)
- can lead to a crash of the program or at least a undefined behaviour

```
#1545 NEW cov: 5 ft: 21 corp: 11/568b lim: 14 exec/s: 0 rss: 69Mb L: 64/122 MS: 3 ChangeASCIIInt-ChangeByte-EraseBytes-
decode-mpls.c:61:16: runtime error: load of misaligned address 0x60700001adce for type 'uint32_t' (aka 'unsigned int'), which requires 4 byte alignment
0x6070000ladce: note: pointer points here
55 66 88 48 88 90 62 01 00 68 55 90 01 00 10 94 55 00 01 00 10 94 56 00 01 88 64 11 00 00 01 00
SUMMARY: UndefinedBehaviorSanitizer: undefined-behavior decode-mpls.c:61:16 in
#19407 NEW cov: 5 ft: 22 corp: 12/759b lim: 191 exec/s: 0 rss: 85Mb L: 191/191 MS: 5 CopyPart-ChangeByte-InsertByte-ShuffleBytes-CrossOver-
#32768 pulse cov: 5 ft: 22 corp: 12/759b lim: 317 exec/s: 16384 rss: 103Mb
#58589 NEW cov: 5 ft: 23 corp: 13/1206b lim: 580 exec/s: 9764 rss: 140Mb L: 447/447 MS: 2 CrossOver-EraseBytes
#65536 pulse cov: 5 ft: 23 corp: 13/1206b lim: 643 exec/s: 9362 rss: 152Mb
#101445 NEW cov: 5 ft: 24 corp: 14/2165b lim: 1003 exec/s: 7246 rss: 216Mb L: 959/959 MS: 4 CrossOver-CrossOver-CMP-EraseBytes- DE: "\x01\x00\x00\x00\x00\x00"
#131072 pulse cov: 5 ft: 24 corp: 14/2165b lim: 1290 exec/s: 5698 rss: 281Mb
INFO: libFuzzer disabled leak detection after every mutation.
       Most likely the target function accumulates allocated
       memory in a global state w/o actually leaking it.
        You may try running this binary with -trace malloc=[12]
                                                                                       to get a trace of mallocs and frees.
       If LeakSanitizer is enabled in this process it will still
        run on the process shutdown
#221189 NEW cov: 5 ft: 25 corp: 15/4148b lim: 2193 exec/s: 4915 rss: 470Mb L: 1983/1983 MS: 2 CrossOver-EraseBytes-
#262144 pulse cov: 5 ft: 25 corp: 15/4148b lim: 2600 exec/s: 5825 rss: 532Mb
#467196 NEW cov: 5 ft; 26 corp; 16/8179b lim; 4636 exec/s; 10156 rss; 825Mb L; 4031/4031 MS; 2 CrossOver-EraseBytes-
#524288 pulse cov: 5 ft: 26 corp: 16/8179b lim: 5200 exec/s: 11155 rss: 935Mb
                           cov: 5 ft: 27 corp: 17/16306b lim: 10215 exec/s: 20984 rss: 1548Mb L: 8127/8127 MS: 5 ShuffleBytes-ShuffleBytes-CrossOver-CMP-EraseBy
                    pulse cov: 5 ft: 27 corp: 17/16306b lim: 10410 exec/s: 21399 rss: 1614Mb
                    NEW cov: 5 ft: 28 corp: 18/31Kb lim: 16418 exec/s: 31774 rss: 1699Mb L: 16319/16319 MS: 5 CrossOver-ChangeASCIIInt-EraseBytes-ShuffleBytes
                    pulse cov: 5 ft: 28 corp: 18/31Kb lim: 20842 exec/s: 38130 rss: 1699Mb
                    pulse cov: 5 ft: 28 corp: 18/31Kb lim: 24000 exec/s: 61680 rss: 1699Mb
==2677==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60800018ca7a at pc 0x5604b5f5ce0e bp 0x7fff3fd61e00 sp 0x7fff3fd61df8
READ of size 1 at 0x60800018ca7a thread T0
     #0 0x5604b5f5ce0d in DecodeMPLS /home/sirko/Projects/CI/fuzzing/suricata/suricata-fuzzing-gitlab/src/decode-mpls.c:96:13
     #1 0x5604b5f3ef15 in DecodeEthernet /home/sirko/Projects/CI/fuzzing/suricata/suricata-fuzzing-gitlab/src/decode-ethernet.c:89:13
    #2 0x5604b5eab0cf in LLVMFuzzerTestOneInput /home/sirko/Projects/CI/fuzzing/suricata/suricata-fuzzing-gitlab/fuzzing/fuzz decoder ethernet.c:106:5
    #3 0x560405d72e05 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) (/home/sirko/Projects/CI/fuzzing/suricata/suricata-fuzzing-gitl
#4 0x560405d75600 in fuzzer::Fuzzer::RunOne(unsigned char const*, unsigned long, bool, fuzzer::InputInfo*, bool*) (/home/sirko/Projects/CI/fuzzing/suricata-fuzzing-gitl
     #5 0x560405d771b9 in fuzzer::Fuzzer::MutateAndTestOne() (/home/sirko/Projects/CI/fuzzing/suricata/suricata-fuzzing-gitlab/fuzzing/playground/decoder eth
     #6 0x5604b5d79e07 in fuzzer::Fuzzer::Loop(std::vector<std:: cxxll::basic string<char, std::char traits<char>, std::allocator<char>, fuzzer::fuzzer all
     #7 0x5694b506a955 in fuzzer::fuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)) //home/sirko/Projects/CI/fuzzing/suricata/suricata
     #8 0x5004b505de73 in main (/home/sirko/Projects/CI/fuzzing/suricata/suricata-fuzzing-gitlab/fuzzing/playground/decoder ethernet/fuzz decoder ethernet+0x6
     #9 0x7ff63c7dbce2 in libc start main (/usr/lib/libc.so.6+0x23ce2)
     #10 0x5604b5d5decd in start (/home/sirko/Projects/CI/fuzzing/suricata/suricata-fuzzing-gitlab/fuzzing/playground/decoder ethernet/fuzz decoder ethernet+
0x60800018ca7a is located 0 bytes to the right of 90-byte region [0x60800018ca20,0x60800018ca7a]
allocated by thread TO here:
    #8 0x5094b5e6c459 in interceptor malloc (/home/sirko/Projects/CI/fuzzing/suricata/suricata-fuzzing-gitlab/fuzzing/playground/decoder ethernet/fuzz deco
     #1 0x5604b5eab02a in LLWFuzzerTestOneInput/home/sirko/Projects/Cl/fuzzing/suricata/suricata-fuzzing-gitlab/fuzzing/fuzz decoder ethernet.c:100:23
    #2 0x560405d72c05 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) (/home/sirko/Projects/CI/fuzzing/suricata/suricata-fuzzing-gitl
#3 0x560405d75000 in fuzzer::Fuzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer::Muzzer:
    #5 0x5604b5d79e07 in fuzzer::Fuzzer::Loop(std::vector<std:: cxx11::basic string<char, std::char traits<char>, std::allocator<char> >, fuzzer::fuzzer all
     #6 0x5604b5d6a955 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)) (/home/sirko/Projects/Clffuzzing/suricata/suricata
    #7 0x5604b5d5de73 in main (/home/sirko/Projects/CI/fuzzing/suricata/suricata-fuzzing-gitlab/fuzzing/playground/decoder ethernet/fuzz decoder ethernet+0xe
     #8 0x7ff63c7dbce2 in libc start main (/usr/lib/libc.so.6+0x23ce2)
SUMMARY: AddressSanitizer: heap-buffer-overflow /home/sirko/Projects/CI/fuzzing/suricata/suricata-fuzzing-gitlab/src/decode-mpls.c:96:13 in DecodeMPLS
```

Recap: Ethernet frame



- DST: Destination MAC Address
- SRC : Source MAC Address
- TYPE: Type of the ethernet frame
- PAYLOAD : contains data
- CRC : Checksum storage

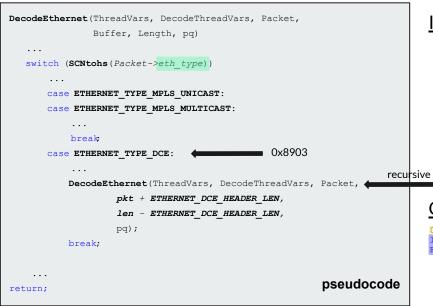
```
DecodeEthernet (ThreadVars, DecodeThreadVars, Packet,
               Buffer, Length, pg)
   switch (SCNtohs (Packet->eth type))
      case ETHERNET TYPE MPLS UNICAST:
       case ETHERNET TYPE MPLS MULTICAST:
           break
      case ETHERNET TYPE DCE:
          DecodeEthernet(ThreadVars, DecodeThreadVars, Packet,
                   pkt + ETHERNET DCE HEADER LEN,
                   len - ETHERNET DCE HEADER LEN,
                   pq);
          break:
                                                        pseudocode
return;
```

Input Data / Ethernet Packet

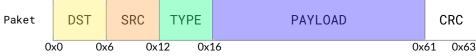


Original input data of the crash





Input Data / Ethernet Packet



Original input data of the crash

```
DecodeEthernet (ThreadVars, DecodeThreadVars, Packet,
               Buffer, Length, pg)
      case ETHERNET TYPE MPLS UNICAST:
       case ETHERNET TYPE MPLS MULTICAST:
           break
       case ETHERNET TYPE DCE:
          DecodeEthernet(ThreadVars, DecodeThreadVars, Packet,
                   pkt + ETHERNET DCE HEADER LEN,
                   len - ETHERNET DCE HEADER LEN,
                   pq);
          break:
                                                        pseudocode
return;
```


2. iteration

3. <u>iteration</u>

5. iteration

? E8 •

What have we found?

- About 30 Fuzz targets
 - o around 150 lines of code per target
- 14 Bugs found
 - o read / write heap buffer overflows, memory leaks, ...
 - SSL Parser, Ethernet Decoder, IPv4/IPv6 Decoder, AppLayerParser for SSH
- 12 CVE's
 - o CVE-2018-10242,
 - o CVE-2018-10244,
 - o CVE-2019-10050-10056
 - o CVE-2019-16411,CVE-2019-16410
 - o CVE-2019-15699
- All bugs are patched in version 5.0.3 / 4.1.8

Conclusion

The big plus of coverage guided fuzzing ...

- Fuzz-Engines like libFuzzer, honggfuzz or AFL performs very well against parser
 - o XML, YAML, JSON, etc...
 - Decoder
 - Image parser (ffmpeg, libpng)
- Coverage driven
- Scalable and really fast

Conclusion

What can be improved ...

- Sometimes it is difficult to set up the environment, especially for "grown" projects with unconventional build environments (i.e. self written build scripts)
- high barrier to entry
 - deep c/c++ knowledge
 - also deep security knowledge
 - time for experimentation 0
- Struction aware fuzzing / socket fuzzing / state dependent fuzzing has to be implemented by "hand"
- Smart device fuzzing is a big challenge

Questions?