Fuzzing Suricata:

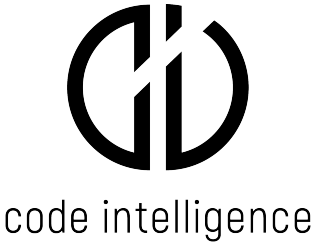
Finding Vulnerabilities in Large Projects

Sirko Höer

@golle0x90

Special thanks

- Robert Haist, DCSO
- 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 !

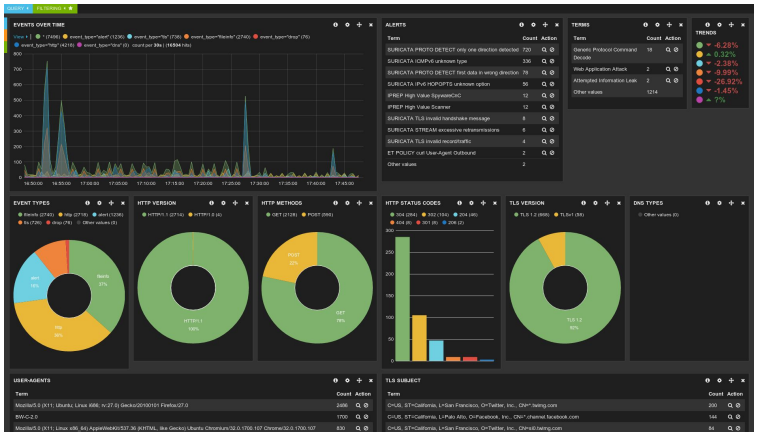
Table of content

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

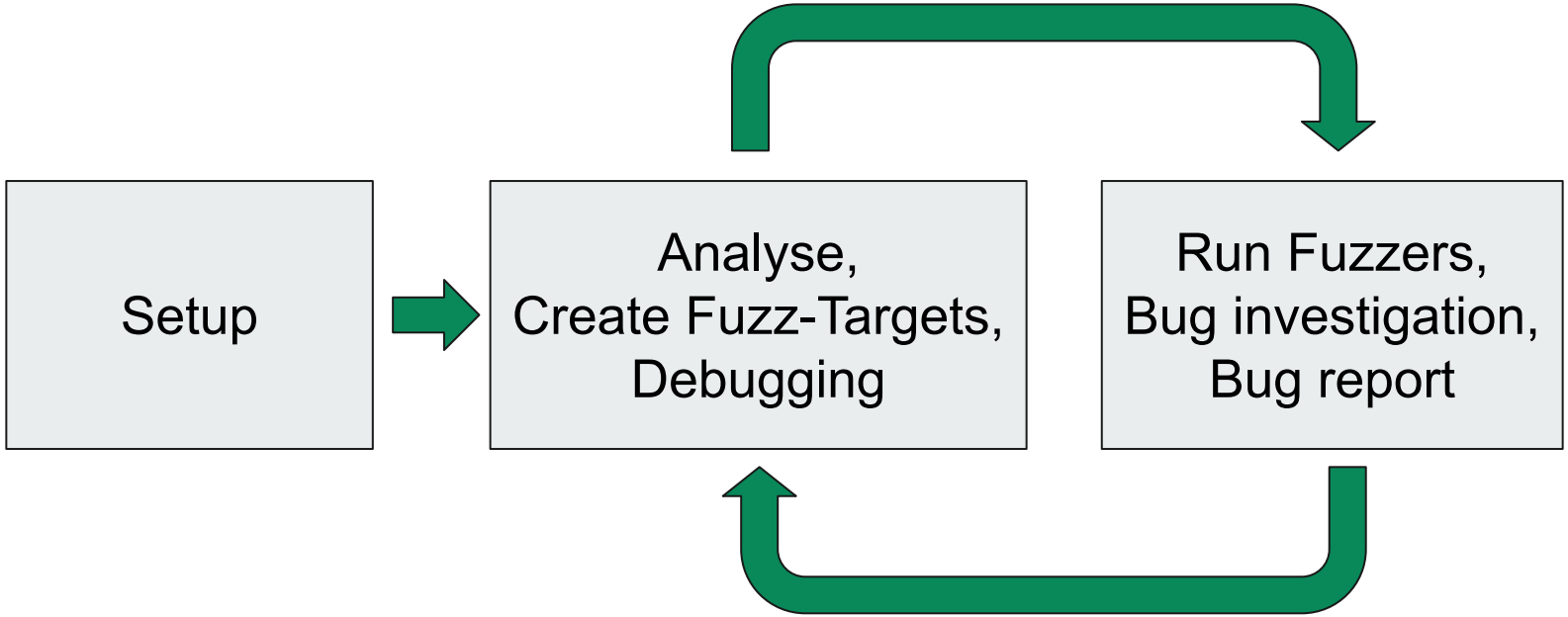
What is suricata

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

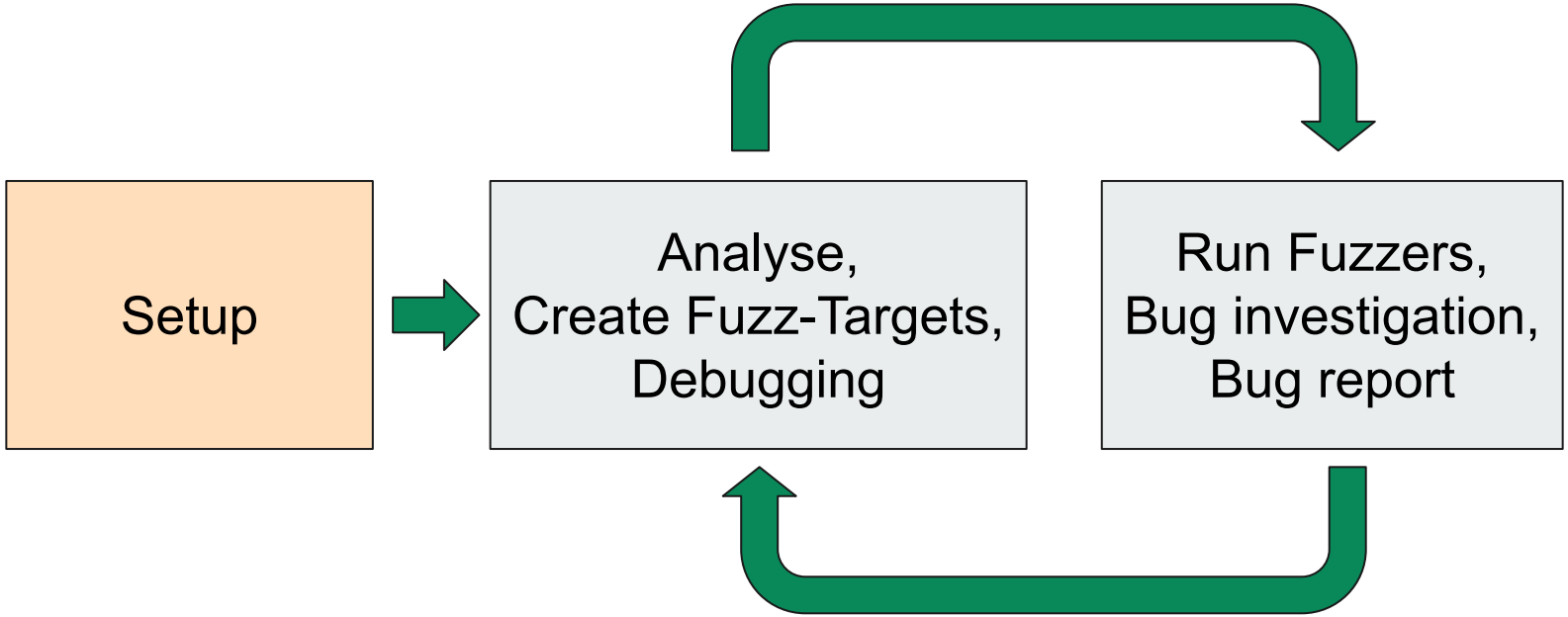
Facts :
- about 600k lines of code
- more than 600 source files
- Uses Unit Tests and AFL Fuzzing (2019)
- since March 2020 integration to google oss-fuzz



Methodology of Fuzzing Projects in general



Methodology of Fuzzing Suricata

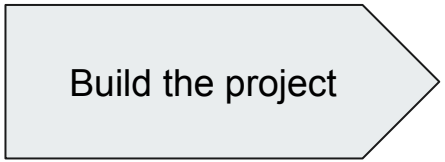


My Setup

Build process

- How to build the project ?

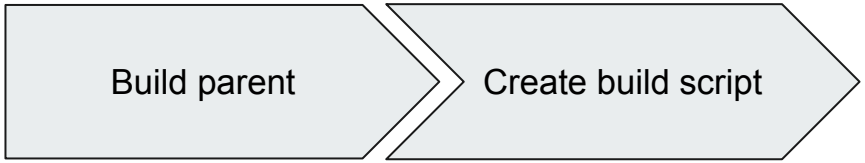
```
$ user@host ./autogen
$ user@host ./configure
$ user@host make all -j$(nproc)
```



My Setup

Build process

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




```
#!/usr/bin/sh
set -e

export CC=clang
export CXX=clang++
export ASAN_OPTIONS=detect_leaks=0

./configure --disable-rust CFLAGS="-O1 -v -g -fPIC \
-fsanitize-coverage=indirect-calls,trace-cmp,trace-div,trace-gep \
-fsanitize=address,fuzzer-no-link,undefined,signed-integer-overflow,bool,pointer-overflow" \
LDFLAG="-fsanitize=address,fuzzer-no-link,undefined,signed-integer-overflow,bool,pointer-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

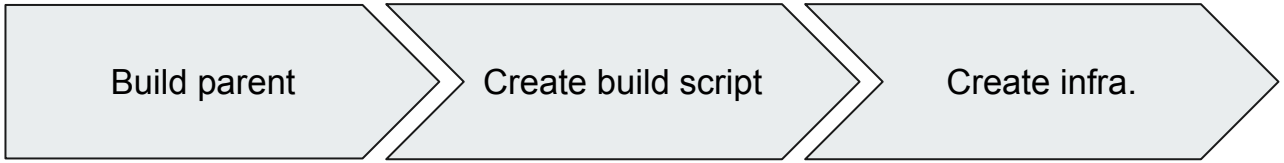
My Setup

Build process

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

```
FROM base/archlinux

#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 -S --noconfirm  \
   jansson \
   libnet \
   libyaml \
   nss
...
```

Dockerfile



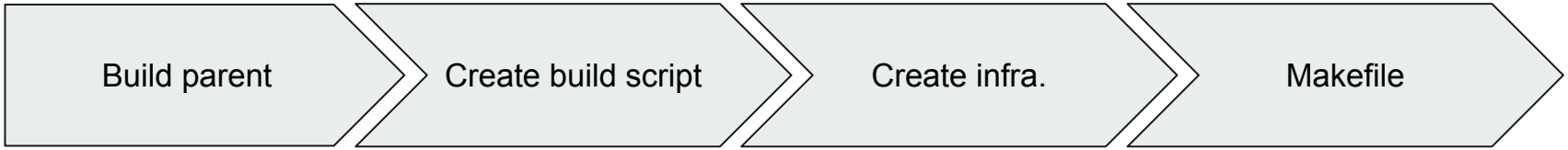
My Setup

Build process

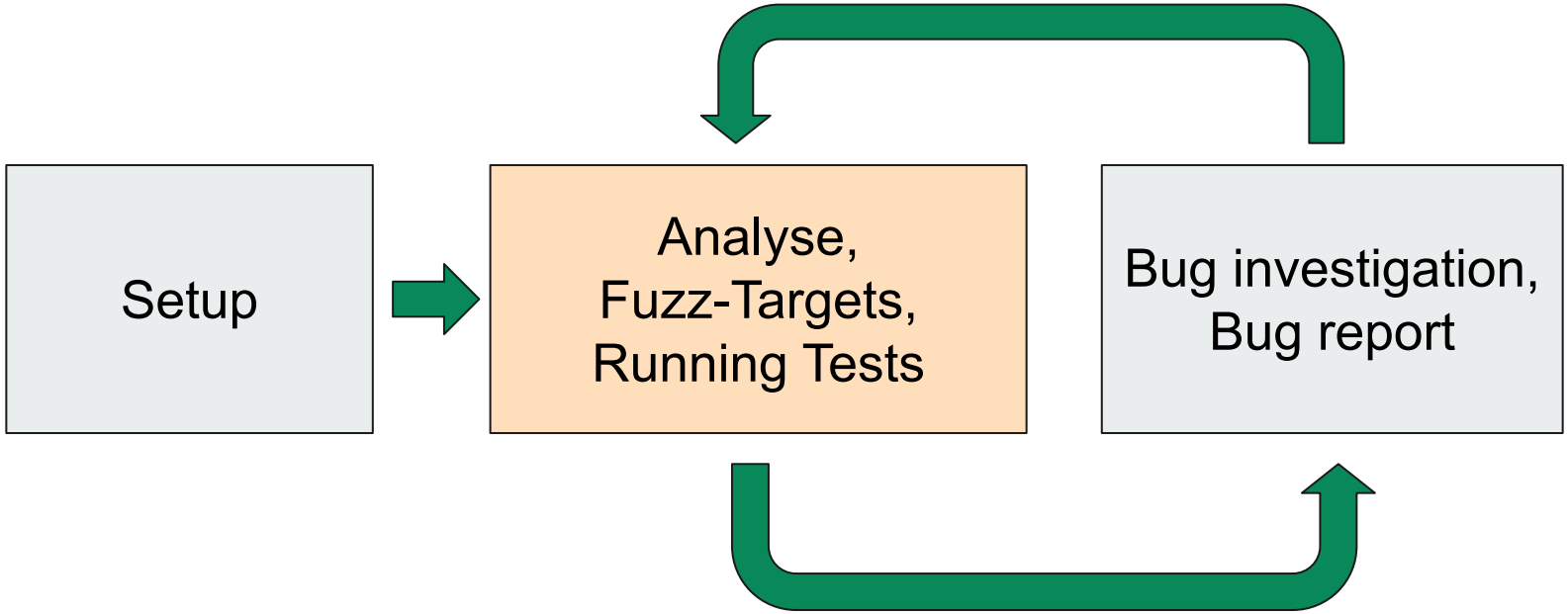
- How to build the project ?
- Create build script with sanitzer
- Build fuzzing infrastructure (Docker)
- Build Makefile for fuzz-targets

```
FUZZERS =   fuzz_app \
            ...
            fuzz_decoder_udp
...
fuzz_%: $(src_target)/fuzz_%.c
    $(shell mkdir -p $(build_target)/$*)
    clang -O1 -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



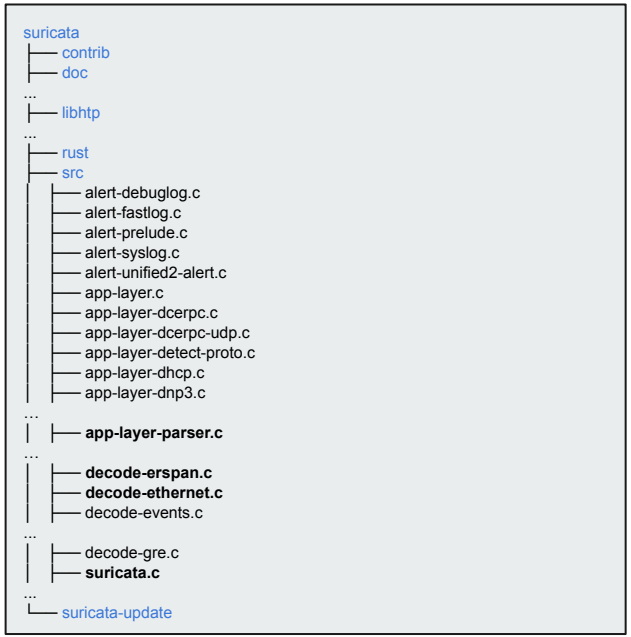
Methodology of Fuzzing Suricata



Analyse

Biggest challenge: Finding entry points

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



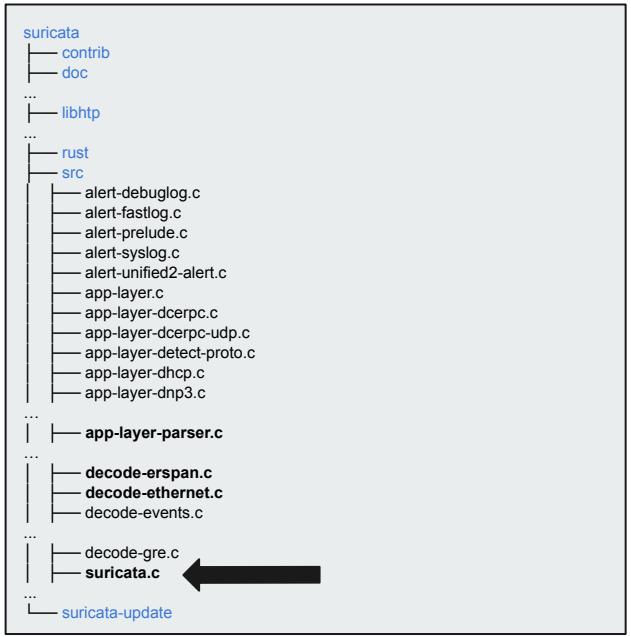
Analyse

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



Analyse

Find interesting things

```
...
1183 static void ParseCommandLineAFL(const char *opt_name, char *opt_arg){
1184 #ifdef AFLFUZZ_RULES
1185    if(strcmp(opt_name, "afl-rules") == 0) {
           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([redacted]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

Analyse

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) {
       //printf("arg: //%s\n", opt_arg);
       MpmTableSetup();
       SpmTableSetup();
       AppLayerProtoDetectSetup();
       AppLayerParserSetup();
       RegisterSSLParsers();
       exit(AppLayerParserFromFile(IPPROTO_TCP, ALPROTO_TLS, opt_arg));
...
```

suricata.c

```
/* Include files */
…
int LLVMFuzzerInitialize(int *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] = 0x05060708;
   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

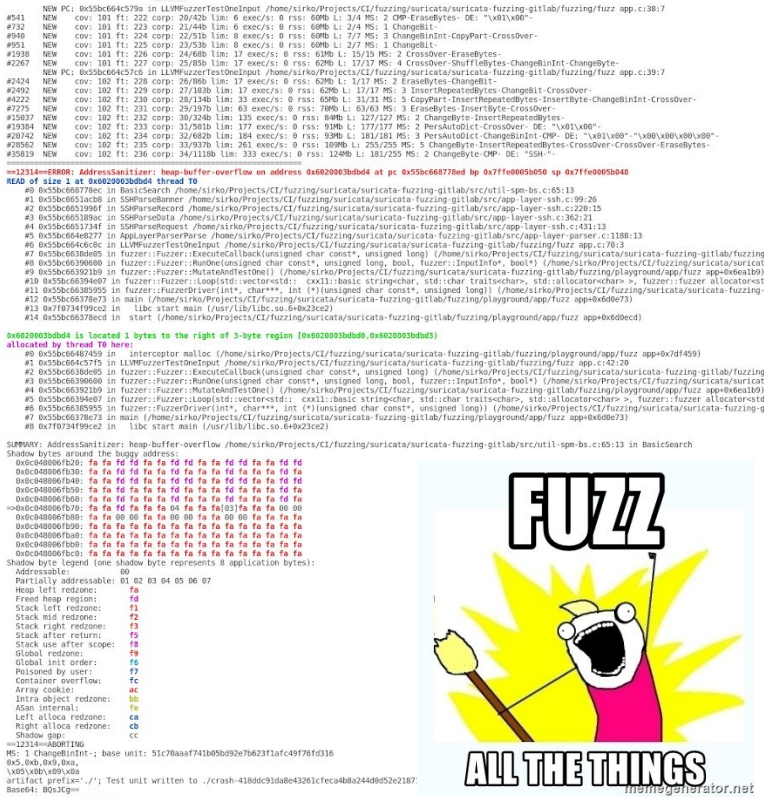
Methodology of Fuzzing Suricata

Creating Fuzz-Targets : The conclusion

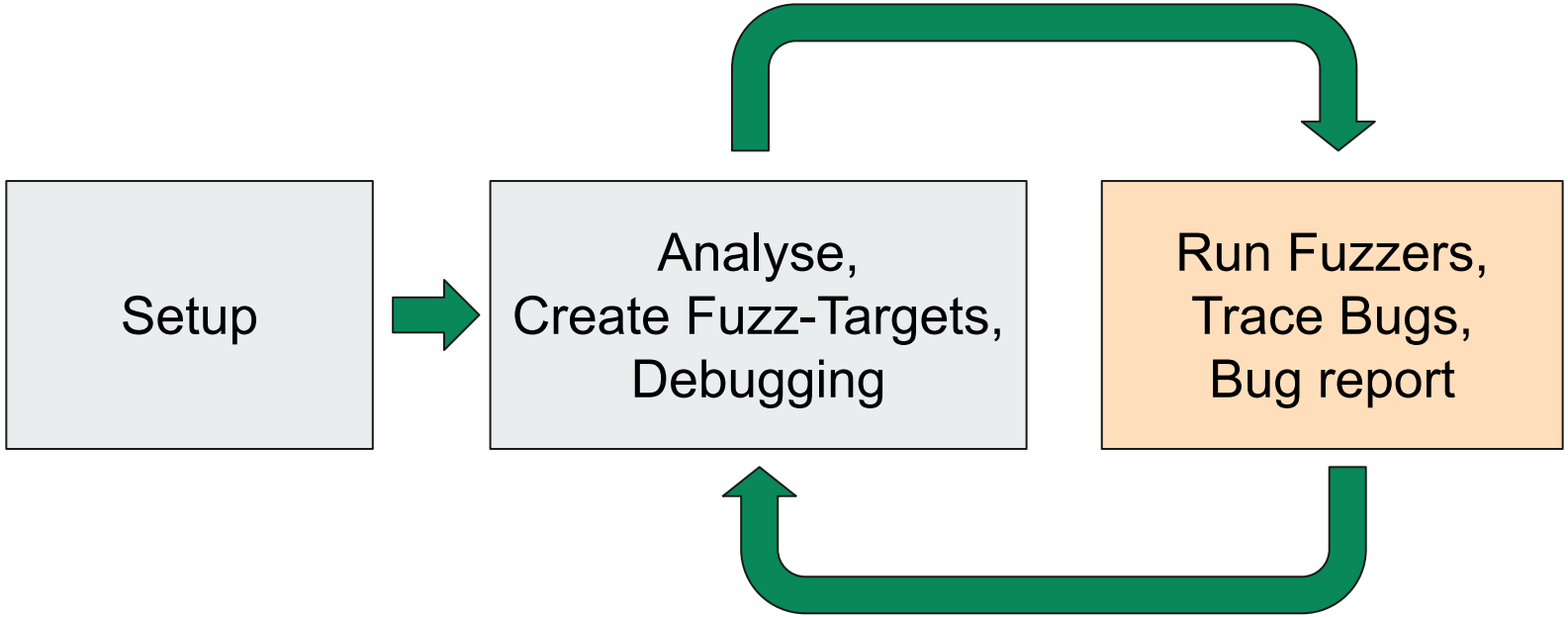
- Reuse of written code
- Customize and generalize the Fuzz Targets

- Think about Seeds and dictionaries
  - Configuration Parser use YAML
  - Decoder needs network packages
  - Rule parser needs rule-files

- Run the fuzzer

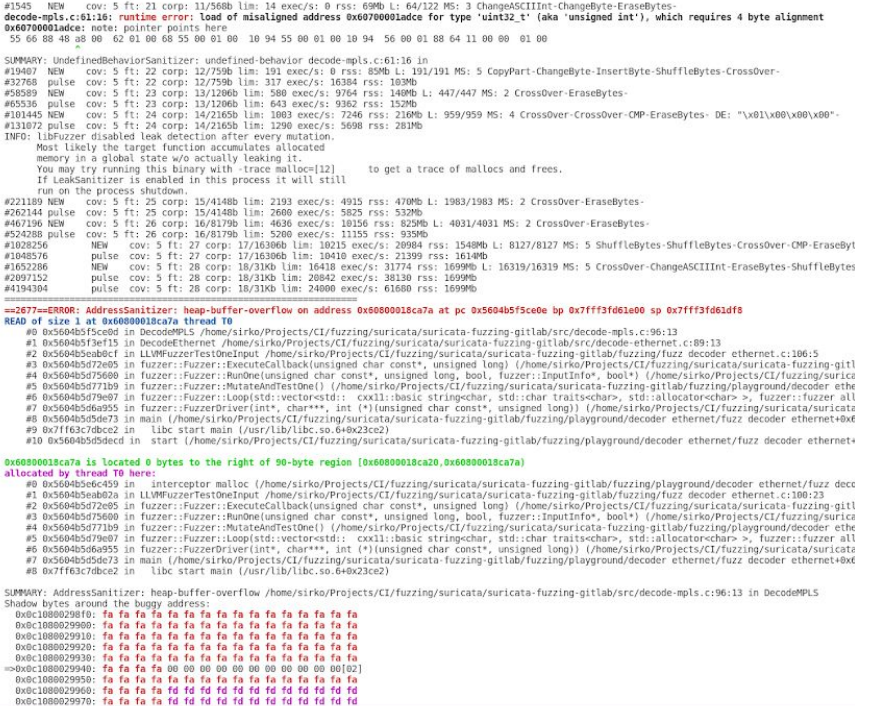


Methodology of Fuzzing Suricata



Ethernet Decoder Heap Buffer Overflow

- 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**



Recap: Ethernet frame

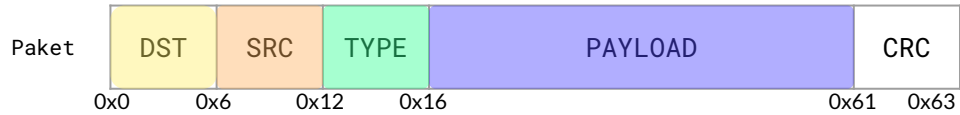
- DST : Destination MAC Address
- SRC : Source MAC Address
- TYPE : Type of the ethernet frame
- PAYLOAD : contains data
- CRC : Checksum storage

Ethernet Decoder Heap Buffer Overflow

```
DecodeEthernet(ThreadVars, DecodeThreadVars, Packet,
               Buffer, Length, pq)
   ...
   switch (SCNtohs(Packet->eth_type))
       ...
       case ETHERNET_TYPE_MPLS_UNICAST:
       case ETHERNET_TYPE_MPLS_MULTICAST:
           ...
           break;
       case ETHERNET_TYPE_DCE:        <-- 0x8903
           ...
           DecodeEthernet(ThreadVars, DecodeThreadVars, Packet,
                   pkt + ETHERNET_DCE_HEADER_LEN,
                   len - ETHERNET_DCE_HEADER_LEN,
                   pq);
           break;

    ...
return;
```

pseudocode

Input Data / Ethernet Packet



Original input data of the crash

```
0A E8 E8 E8 E8 E8 E8 E8 E8 E8 E8 89 03 11 22 33 44 55 66 11 22 33 44 55 66 89 03
11 22 33 44 55 66 11 22 33 44 55 66 89 03 11 22 33 44 55 66 11 22 33 44 55 66 89 03
E8 E8 E8 E8 E8 E8 E8 E8 E8 E8 E8 E8
```

Ethernet Decoder Heap Buffer Overflow

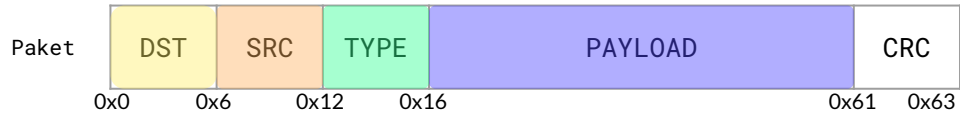
```
DecodeEthernet(ThreadVars, DecodeThreadVars, Packet,
               Buffer, Length, pq)
   ...
   switch (SCNtohs(Packet->eth_type))
       ...
       case ETHERNET_TYPE_MPLS_UNICAST:
       case ETHERNET_TYPE_MPLS_MULTICAST:
           ...
           break;
       case ETHERNET_TYPE_DCE:
           ...
           DecodeEthernet(ThreadVars, DecodeThreadVars, Packet,
                   pkt + ETHERNET_DCE_HEADER_LEN,
                   len - ETHERNET_DCE_HEADER_LEN,
                   pq);
           break;

    ...
return;
```

0x8903

recursive

pseudocode

Input Data / Ethernet Packet



Original input data of the crash

```
0A E8 E8 E8 E8 E8 E8 E8 E8 E8 E8 89 03 11 22 33 44 55 66 11 22 33 44 55 66 89 03
11 22 33 44 55 66 11 22 33 44 55 66 89 03 11 22 33 44 55 66 11 22 33 44 55 66 89 03
E8 E8 E8 E8 E8 E8 E8 E8 E8 E8 E8
```

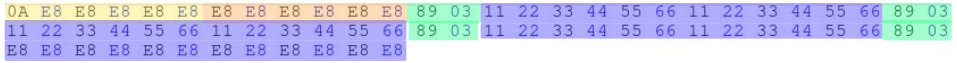
Ethernet Decoder Heap Buffer Overflow

```
DecodeEthernet(ThreadVars, DecodeThreadVars, Packet,
               Buffer, Length, pq)
   ...
   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;

    ...
return;
```

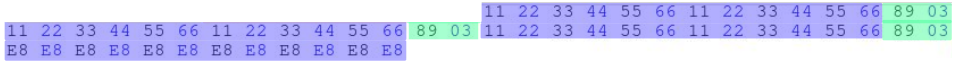
0x8903

pseudocode

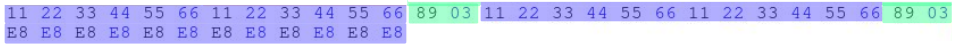
1. iteration



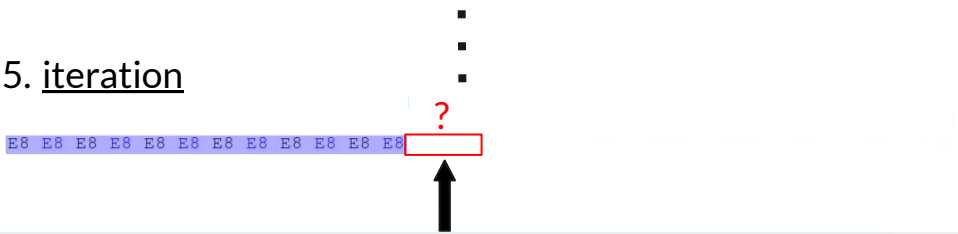
2. iteration



3. iteration



5. iteration



What have we found ?

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

- Struction aware fuzzing / socket fuzzing / state dependent fuzzing has to be implemented by “hand”

- Smart device fuzzing is a big challenge

Questions ?