Fuzzing ntopng

ntopConf ‘23, Pisa, Italy

Riccardo Mori <rmori@quarkslab.com>

Quarkslab

WHOAMI

Riccardo Mori

- Security researcher @ **Quarkslab**
- Automated analysis team
- Expert in Reverse Engineering & Fuzzing

What is the objective?

Identify **bugs** in the software

A bug is a misbehavior of the program unintended by the developer. It might lead or not to a crash of the program

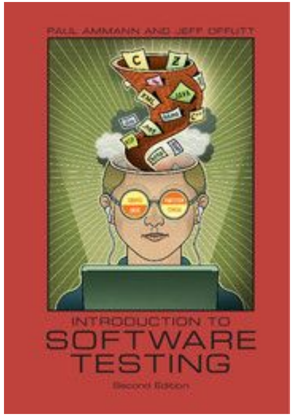
- Automatize the process
- Integrate it in the CI/CD pipeline

What is fuzzing

Introducing fuzzing

Software testing technique that uses automatically generated inputs

- **1988**: First mention of "fuzzing" by *Barton Miller*
- **2012**: Google announced ClusterFuzz infrastructure to fuzz Chrome
- **2013**: First AFL version released
- **2016**: Google OSS-fuzz
- story goes on ...



Introduction to Software Testing, *Paul Ammann*, *Jeff Offutt*, 2008

What is fuzzing

Generate input test → Run the software → Crash? (Y/N)

Three approaches

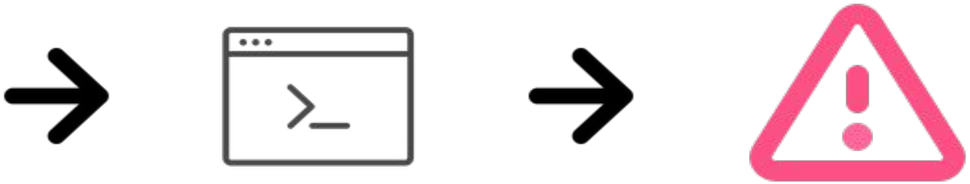
- **Blackbox fuzzing**
- **Greybox fuzzing**
- **Whitebox fuzzing**

Blackbox fuzzing

Blackbox fuzzer

- ➢ Randomly generated inputs
- ➢ No knowledge of the target required
- ➢ Very fast & easy to setup
- ➢ Shallow exploration of the software

/dev/urandom



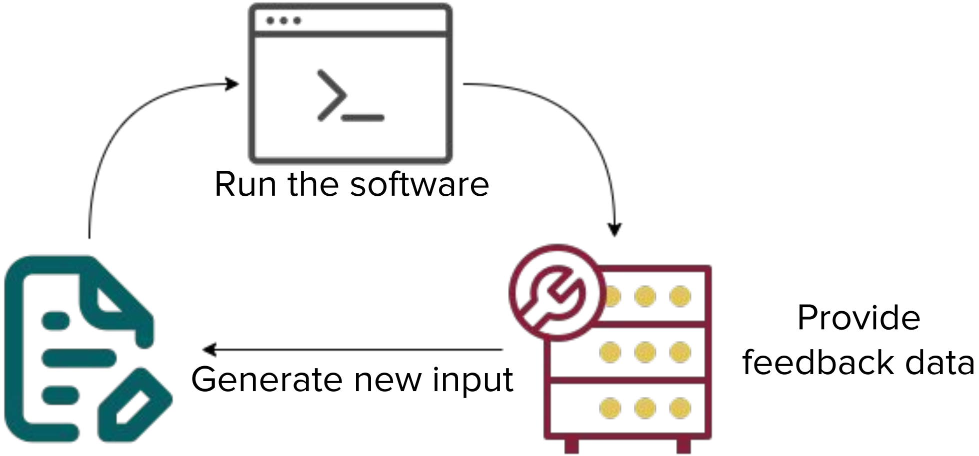
Run the software

Crash?
(Y/N)

Greybox fuzzing

Greybox fuzzer

- Instrument the binary to have some knowledge of the target
- **Good balance** between blackbox and whitebox
- A lot of tools (libfuzzer, AFL++, honggfuzz)



Whitebox fuzzing

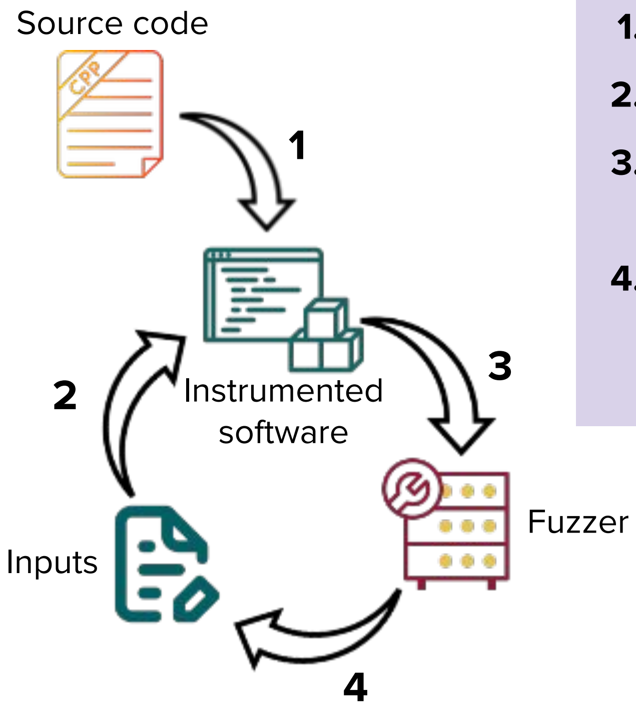
Whitebox fuzzer

- Consider a full program semantic knowledge
- Doesn’t rely on bruteforce to reach deep states
- Can check more advanced properties
- Slower and harder to put in action
- Fewer tools available (pastis + TritonDSE)

Fuzzing ntopng: The setup

The approach

- **Greybox**
- **Coverage based**
- **Evolutionary mutational inputs**



1. Instrument the software
2. Run the input
3. Collect the code coverage data
4. New generation of inputs using the evolutionary algorithm

Source code instrumentation

Source code instrumentation

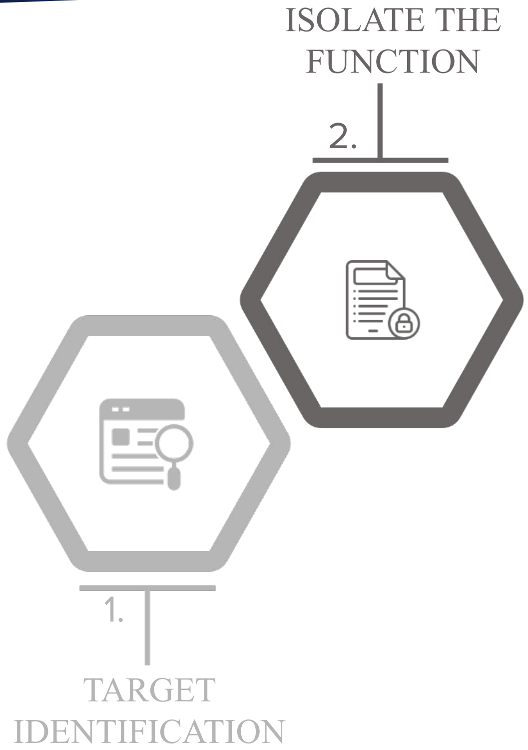
Addition of run-time code that will **trace** the execution of the program in order to get the **code coverage** for the input

- Plenty of tools to do it: libfuzzer *(LLVM)*, AFL++, honggfuzz, ...
- Wrapper over clang/gcc

Fuzzing ntopng: The workflow

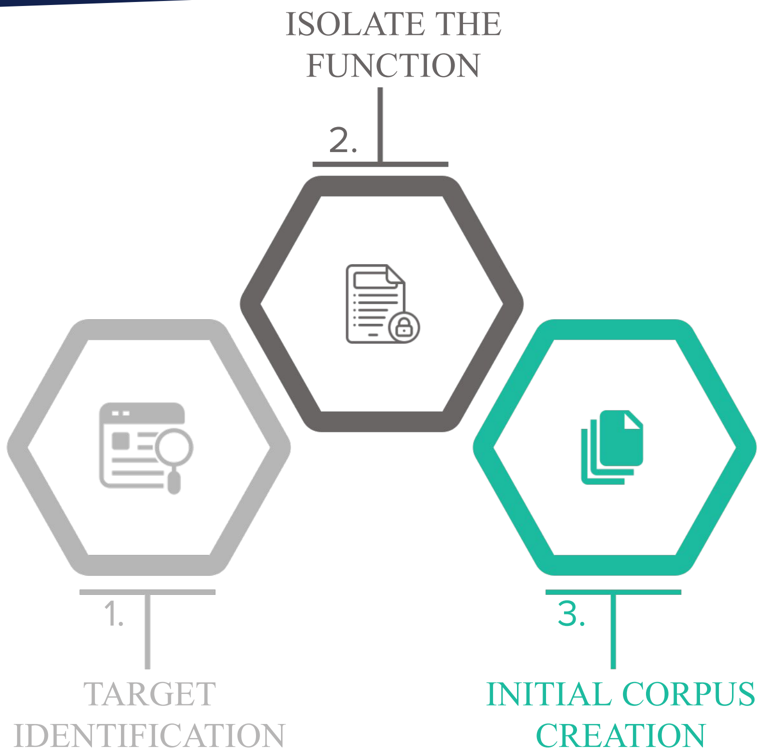
Identify **interesting** functions *(parsers, cryptographic functions, …)*

Fuzzing ntopng: The workflow



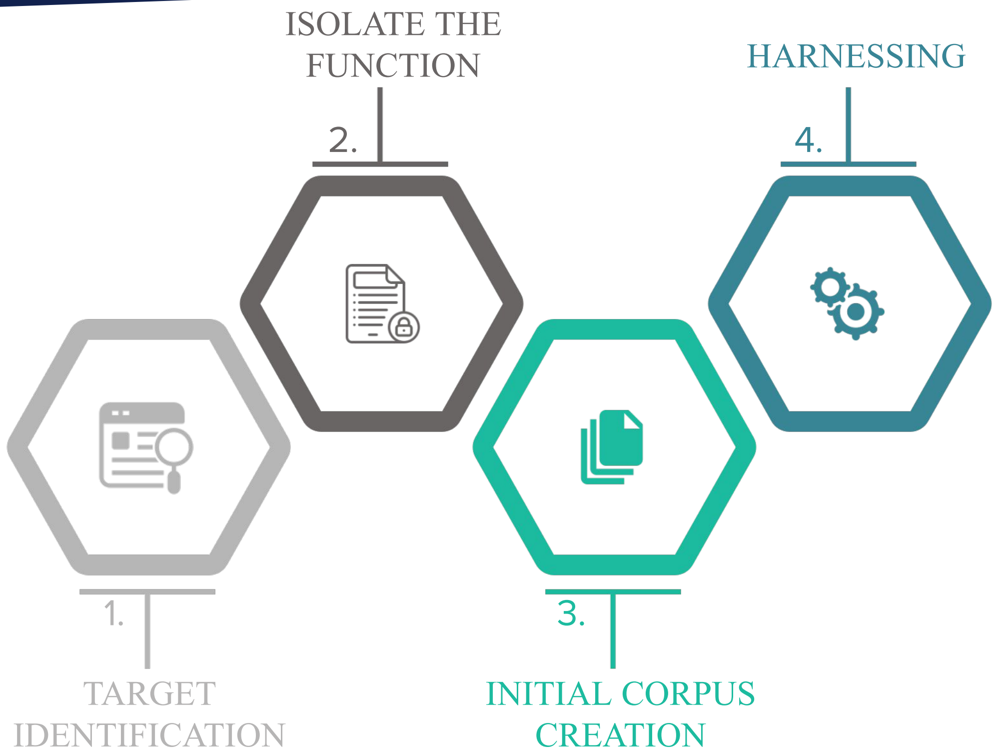
Isolate the target function *(focus on speed and reliability)*

Fuzzing ntopng: The workflow



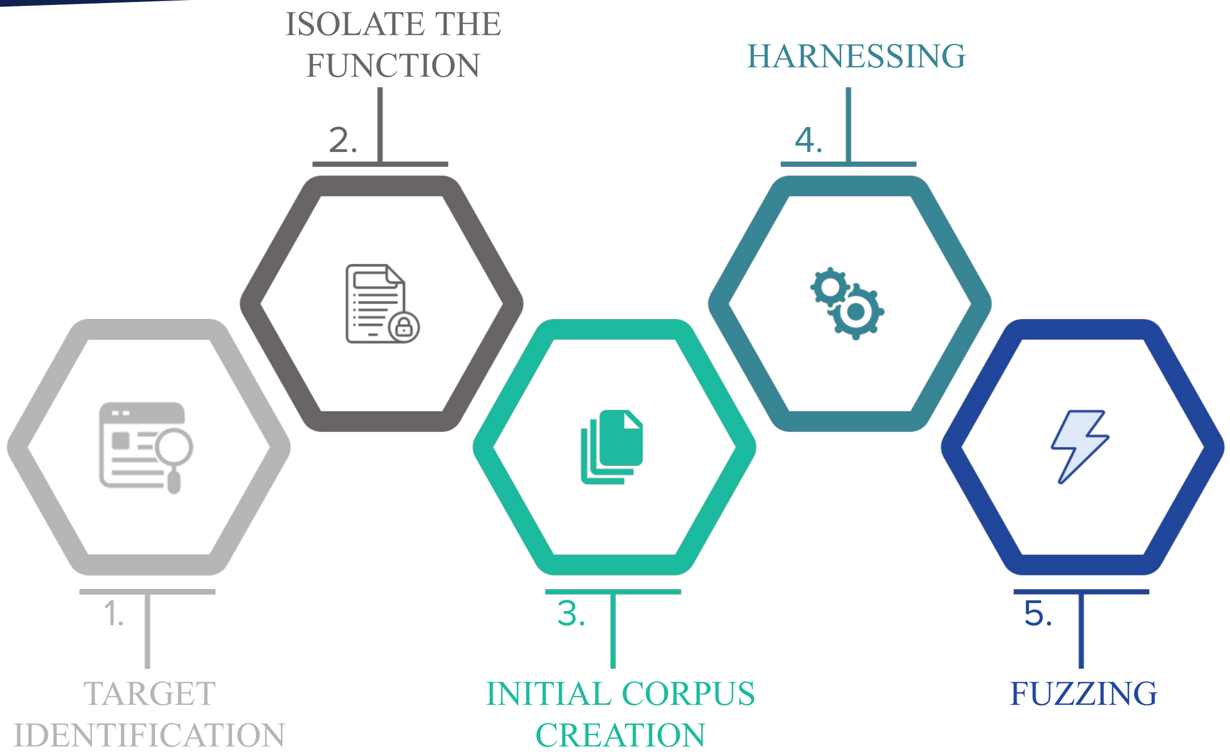
Build an initial corpus of inputs *(good corpus is saving time!)*

Fuzzing ntopng: The workflow



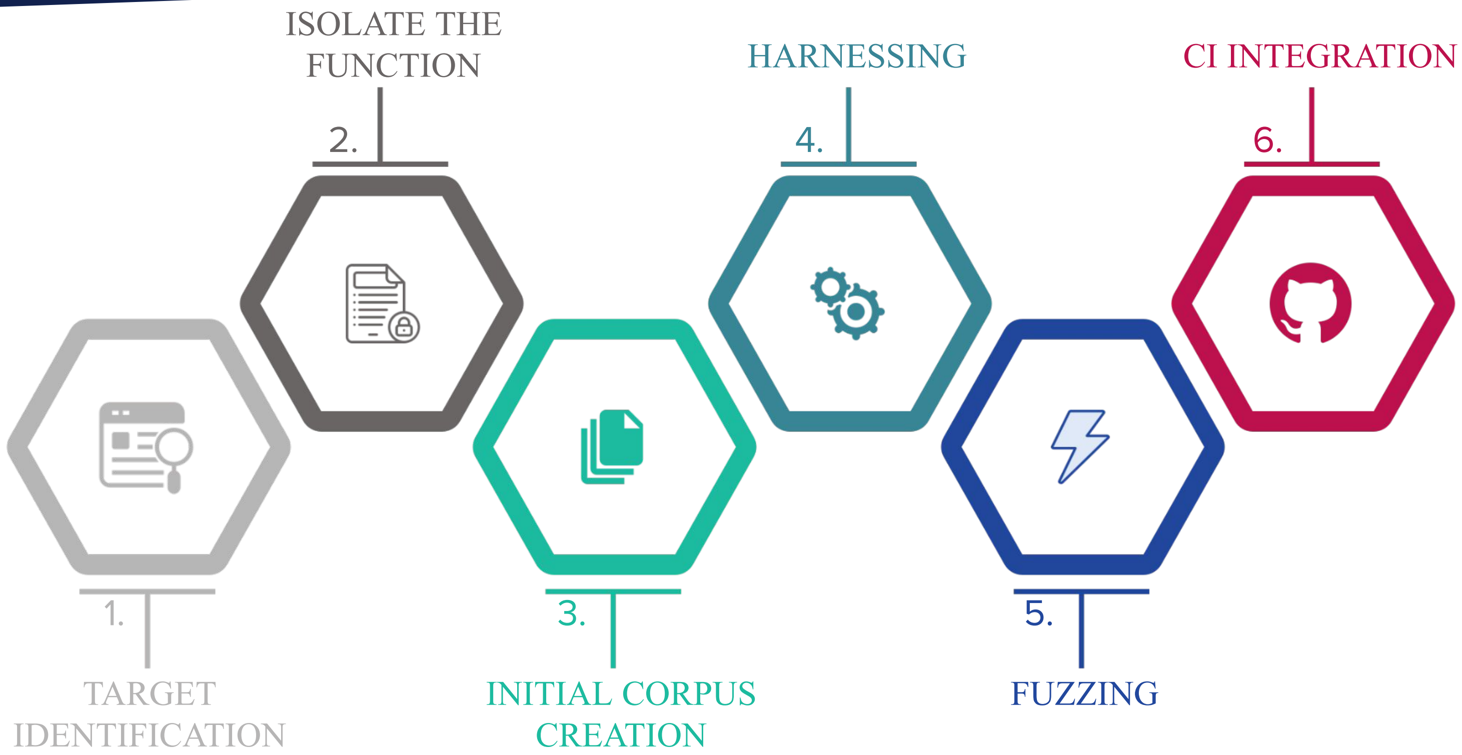
Glue code that combines the **fuzzer**, the **target** function and the **configuration**

Fuzzing ntopng: The workflow



Eat, sleep, fuzz, repeat

Fuzzing ntopng: The workflow

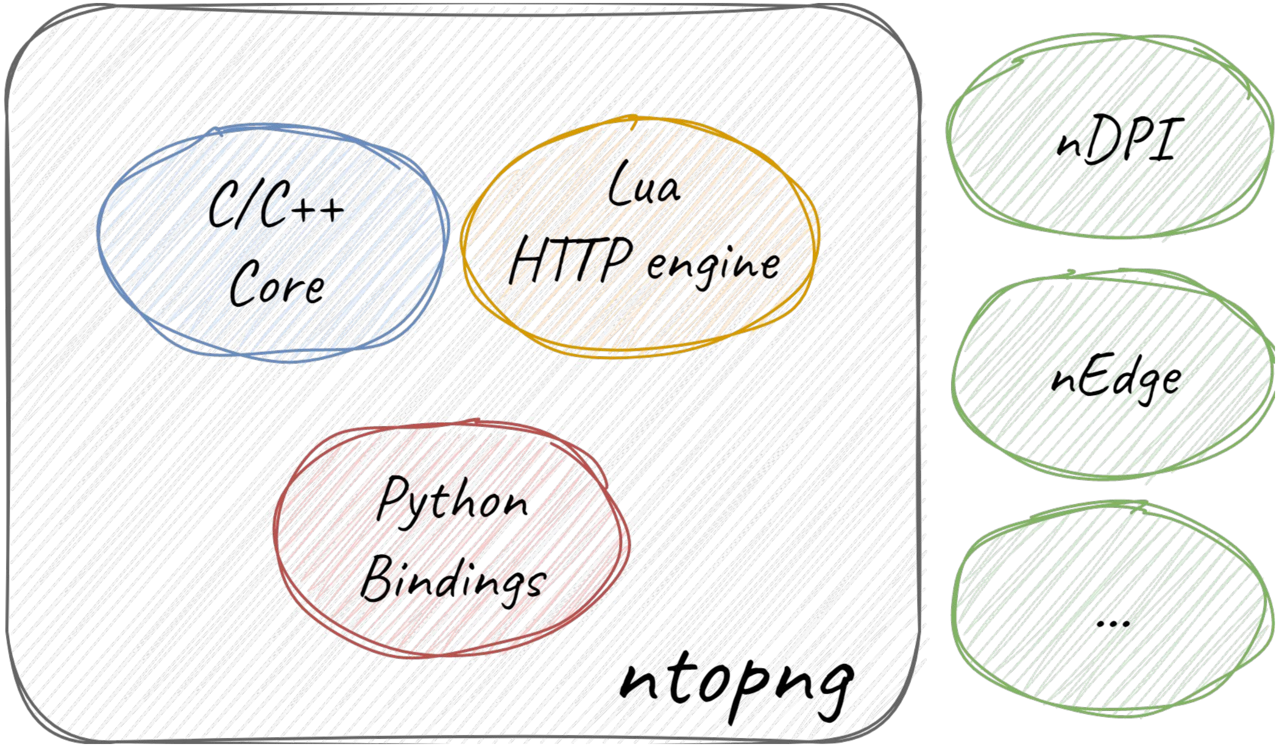


Integrate everything in the CI/CD pipeline

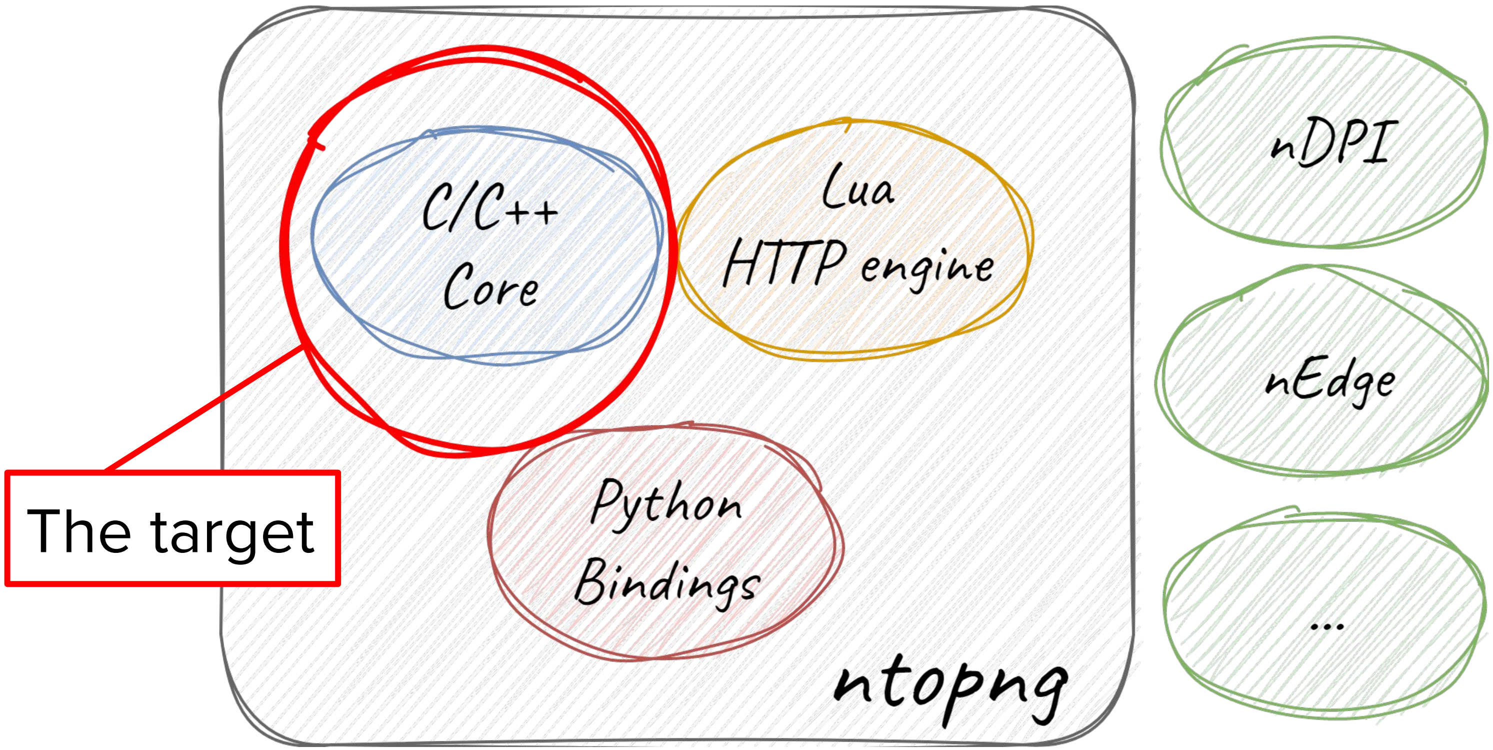
Fuzzing ntopng: The workflow

Identify **interesting** functions *(parsers, cryptographic functions, …)*

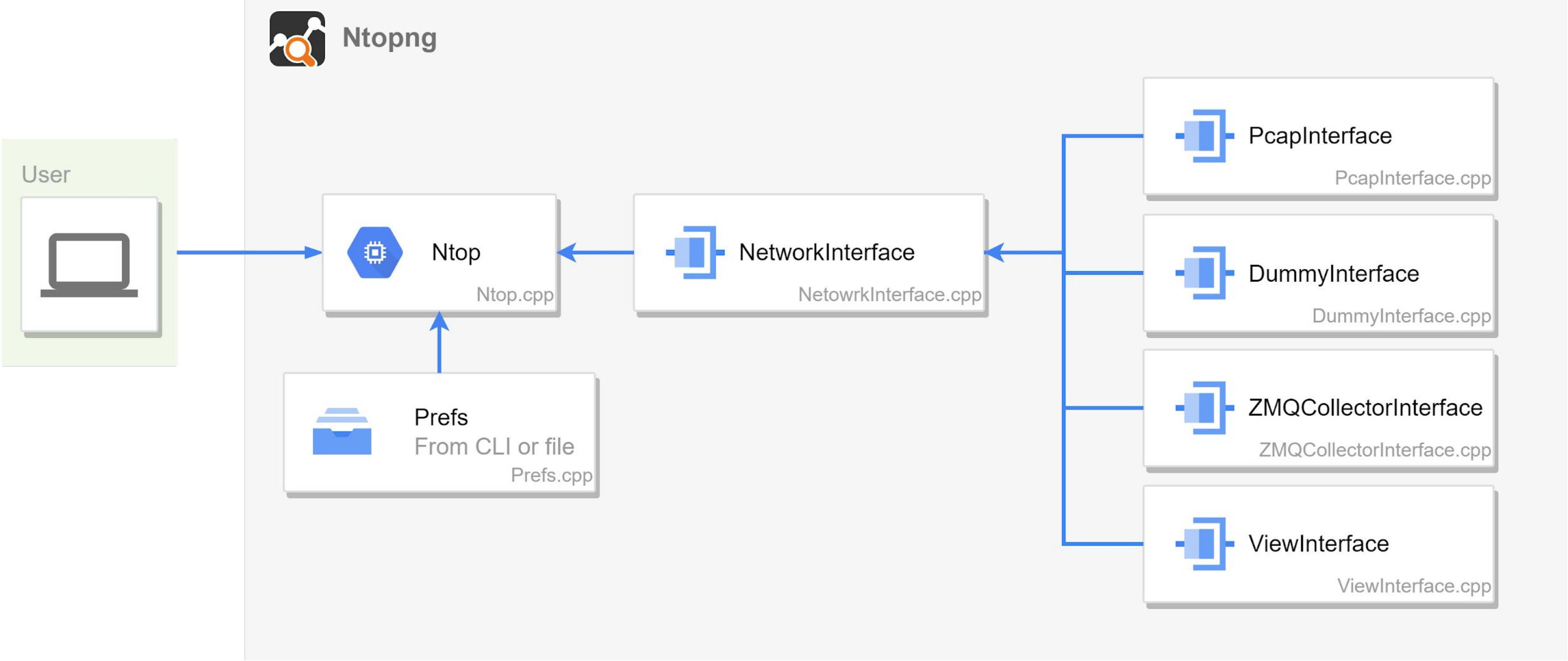
ntopng: an overview



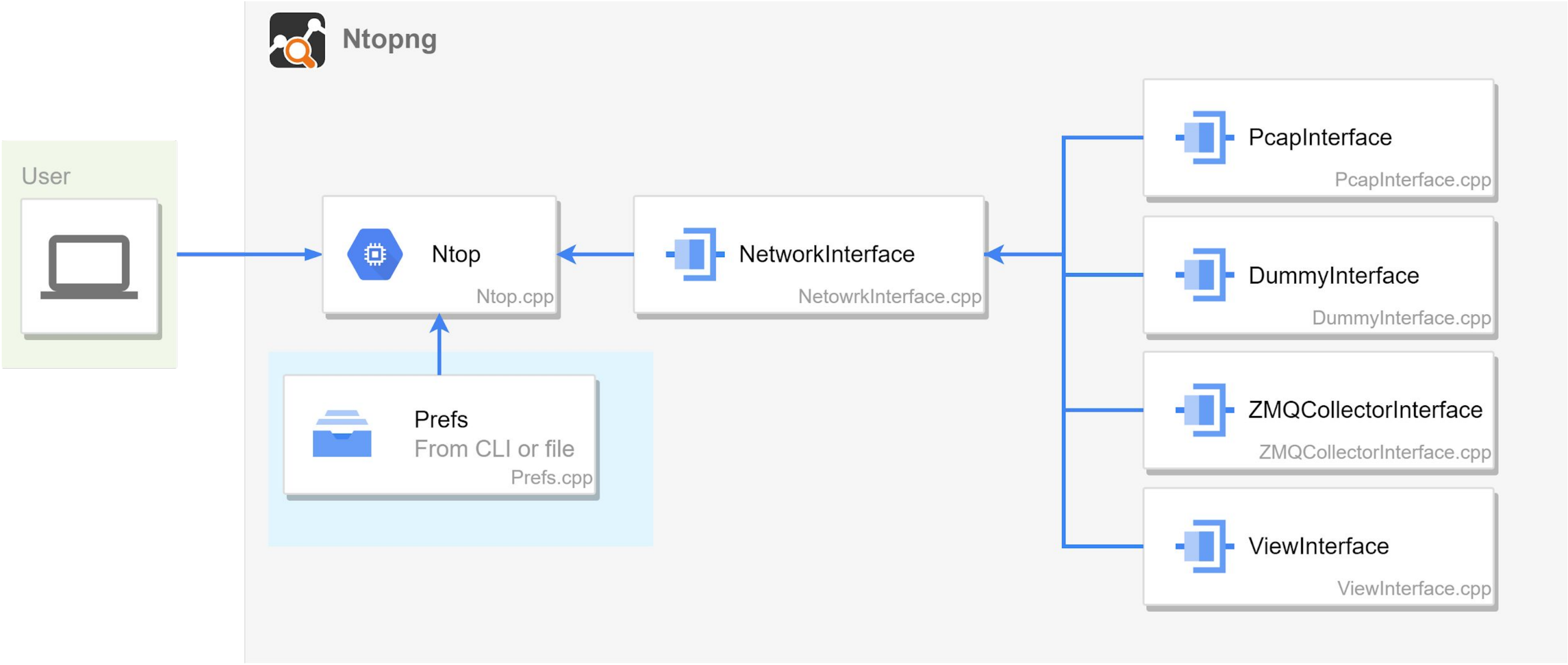
ntopng: an overview



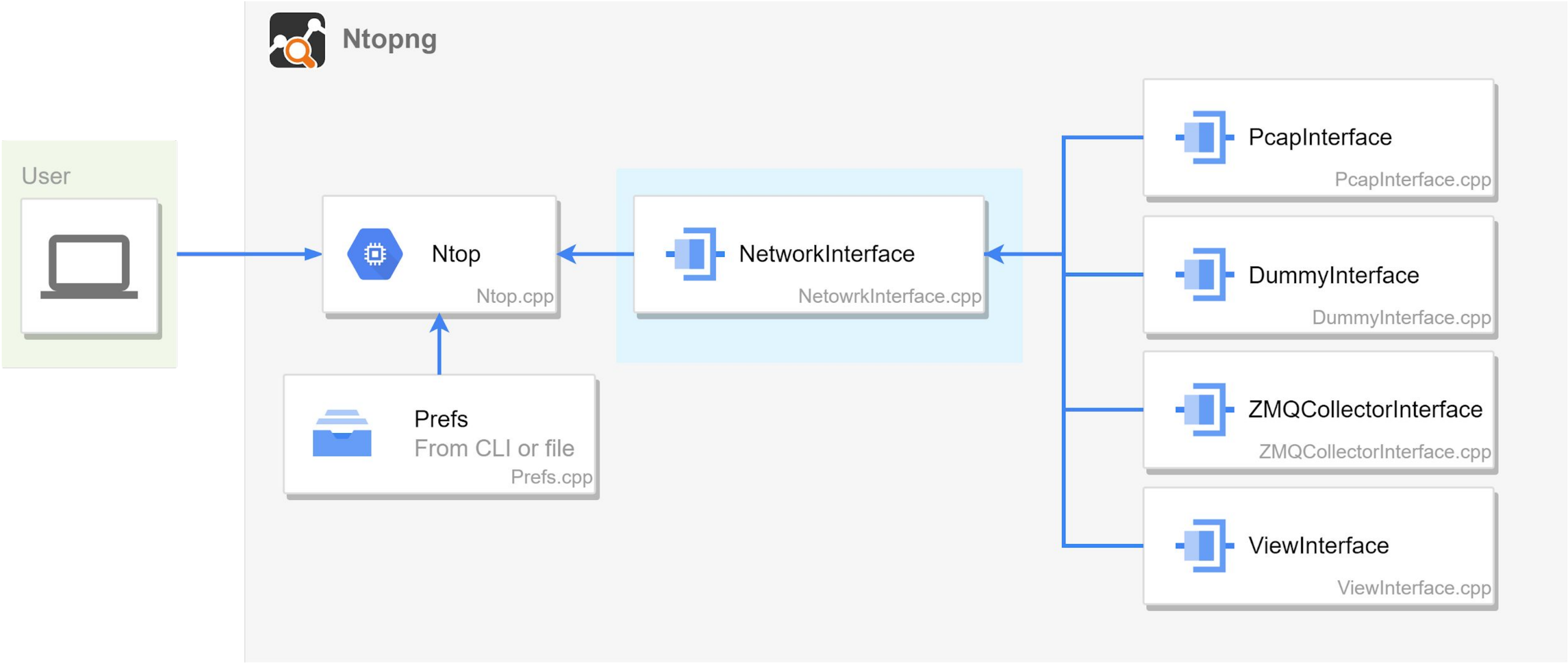
ntopng: choosing the target function



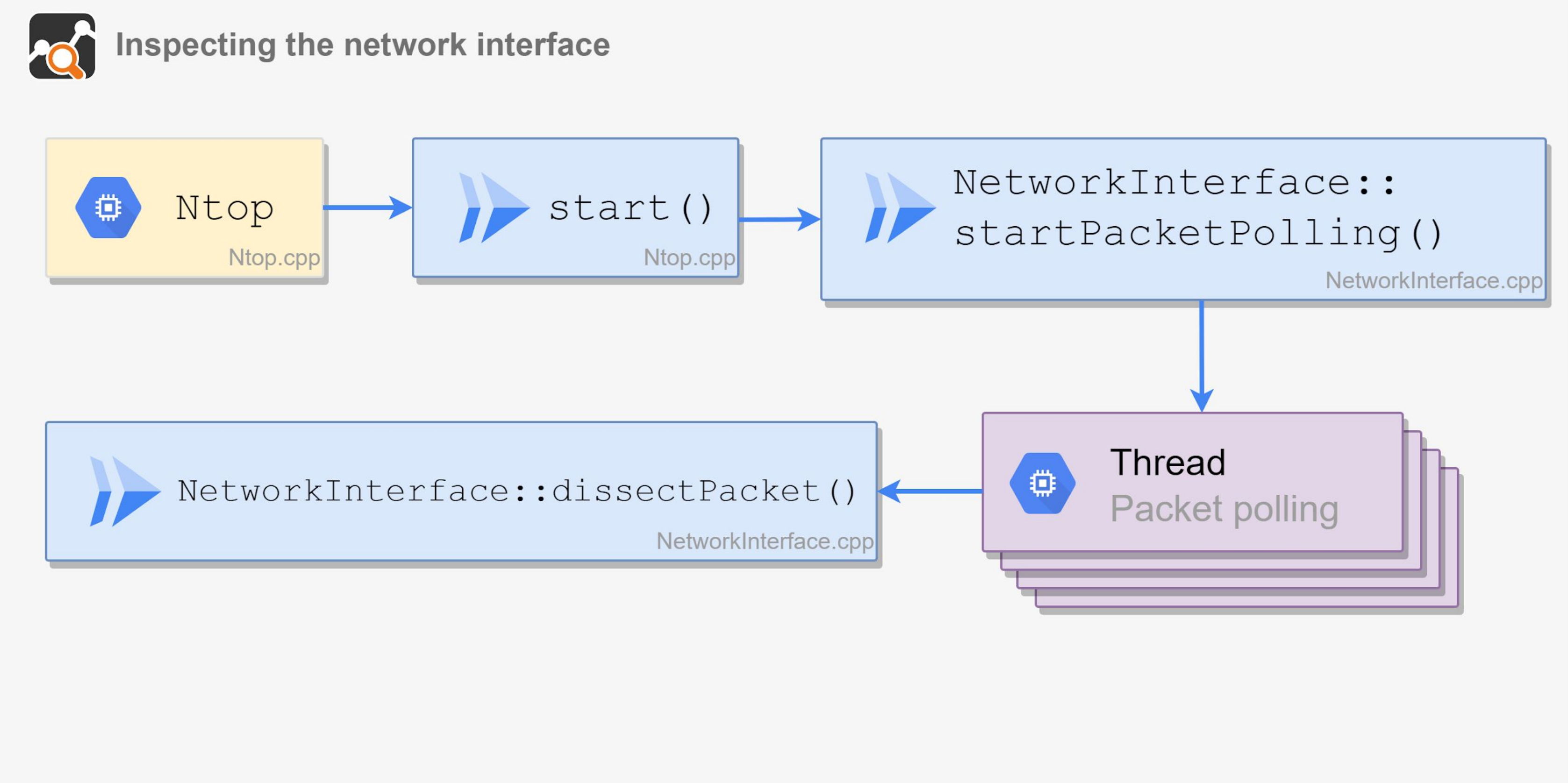
ntopng: choosing the target function



ntopng: choosing the target function

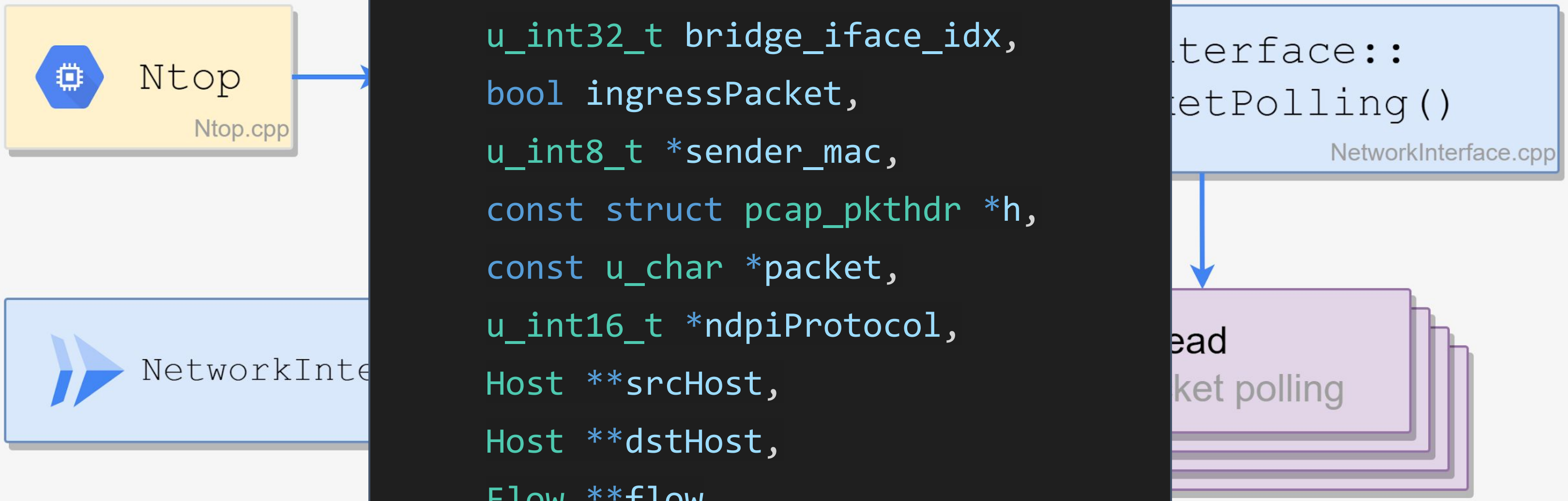


ntopng: choosing the target function



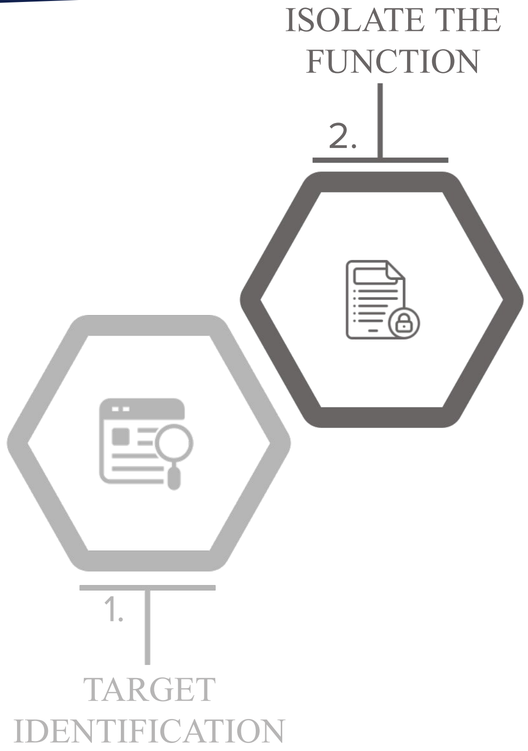
ntopng: choosing the target function

Inspecting the network interface




```
bool NetworkInterface::dissectPacket(
    u_int32_t bridge_iface_idx,
    bool ingressPacket,
    u_int8_t *sender_mac,
    const struct pcap_pkthdr *h,
    const u_char *packet,
    u_int16_t *ndpiProtocol,
    Host **srcHost,
    Host **dstHost,
    Flow **flow
);
```

Fuzzing ntopng: The workflow



Isolate the target function *(focus on speed and reliability)*

Isolate the target function

Many problems

- ntopng was not meant to be run as a library
- Startup takes a lot of time (~1 sec)
- Thread pool to process traffic asynchronously
- **REDIS** caching system present

Isolate the target function

Many problems

- ntopng was not meant to be run as a library
- Startup takes a lot of time (~1 sec)
- Thread pool to process traffic asynchronously
- **REDIS** caching system present

The solution

- Use `FUZZING_BUILD_MODE_UNSAFE_FOR_PRODUCTION` to **patch** the code that we don't want to execute
- Don't start the thread pool
- Initialize only the **absolute minimum** set of components

How to decouple from REDIS?

- **Patch** every call to REDIS *(possible side effects)*
- Create a **stub** class that implements the same functionalities of REDIS and **swap** it to the official client

Isolate the target function

```
#ifdef FUZZING_BUILD_M...
#include "RedisStub.h"
#else
#include "Redis.h"
#endif
```

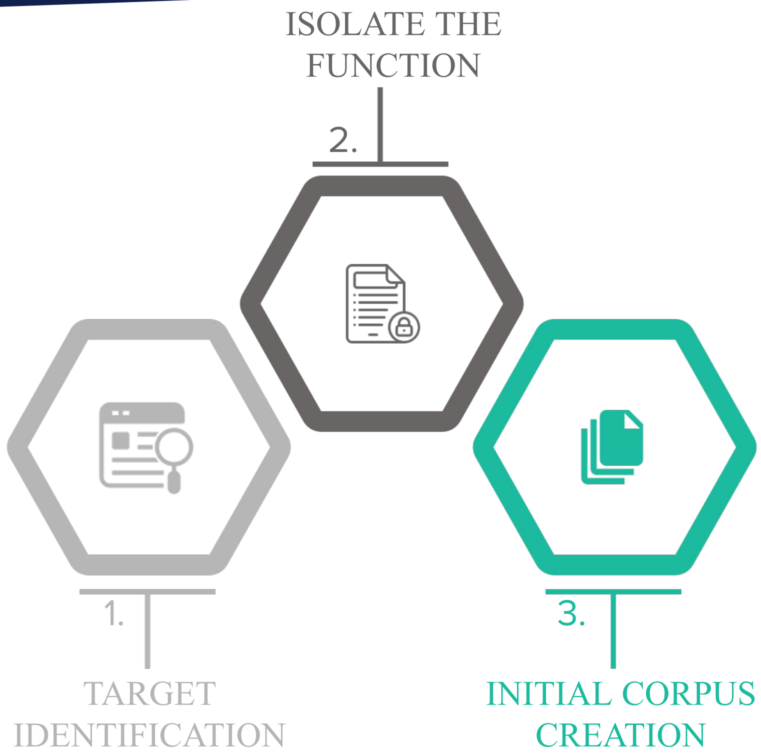
```
# Redis.h
class Redis {
private:
 /* ... */
public:
 Redis(...);

 int expire(char *, u_int);
 int get(...);
 int hashGet(...);
 int hashDel(...);
 char *dump(char *key);
 /* ... continue ... */
}
```

```
# RedisStub.h
class Redis {
/* Keep only the public
   interface */
public:
 Redis(...);

 int expire(char *, u_int);
 int get(...);
 int hashGet(...);
 int hashDel(...);
 char *dump(char *key);
 /* ... continue ... */
}
```

Fuzzing ntopng: The workflow



Build an initial corpus of inputs *(good corpus is saving time!)*

Building corpus

A good corpus

- Achieve a good **code coverage**
- Be as small as possible *(avoid duplicate samples)*
- Try to trigger edge cases *(use invalid packets)*

Building corpus

A good corpus

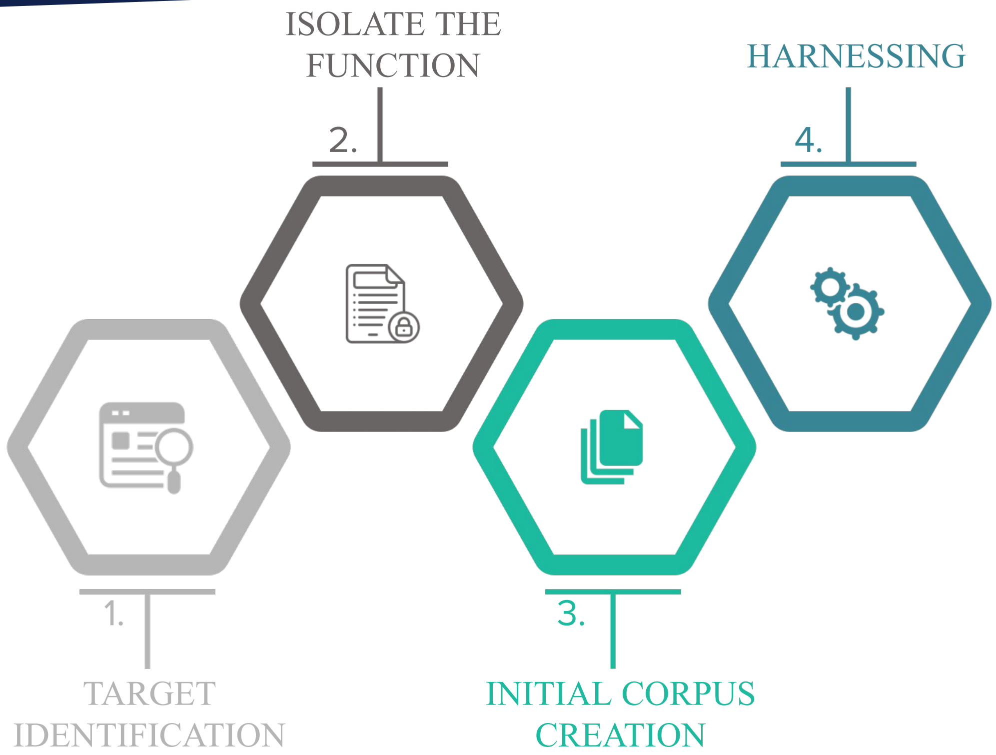
- ➢ Achieve a good **code coverage**
- ➢ Be as small as possible *(avoid duplicate samples)*
- ➢ Try to trigger edge cases *(use invalid packets)*

- **Never reinvent the wheel**

- Take it from tcpdump **test set**.

  674 pcap files ranging from 24 bytes to 510 KB

Fuzzing ntopng: The workflow



Glue code that combines the **fuzzer**, the **target** function and the **configuration**

Harnessing

The goals

- **Initialize** the software state *(Ntop, Prefs, NetworkInterface objects)*
- Test the code with the provided input *(bypass checksum, ...)*
- Clean the state for a new iteration *(free the memory, ...)*

- The interface to the fuzzing engine is not common

LibFuzzer interface

```
extern "C" int LLVMFuzzerInitialize(int *argc, char ***argv);
```

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *buf, size_t len);
```

```
$ clang++ -fsanitize=fuzzer -o fuzzer source.cpp
$ ./fuzzer corpus-dir
```

Initialization

```
extern "C" int LLVMFuzzerInitialize(int *argc, char ***argv) {
    if ((ntop = new (std::nothrow) Ntop(PROG_NAME)) == NULL) _exit(1);
    if ((prefs = new (std::nothrow) Prefs(ntop)) == NULL) _exit(1);

    setCLIArgs(prefs, 11, PROG_NAME, "-1", "_PATH_docs", "-2", "_PATH_scripts", "-3",
               "_PATH_scripts/callbacks", "-d", "_PATH_data-dir", "-t", "_PATH_install");
    ntop->registerPrefs(prefs, false);
    ntop->loadGeolocation();  // Ntop initialization
    iface = new NetworkInterface("custom");  // Interface initialization
    iface->allocateStructures();
    return 0;
}
```

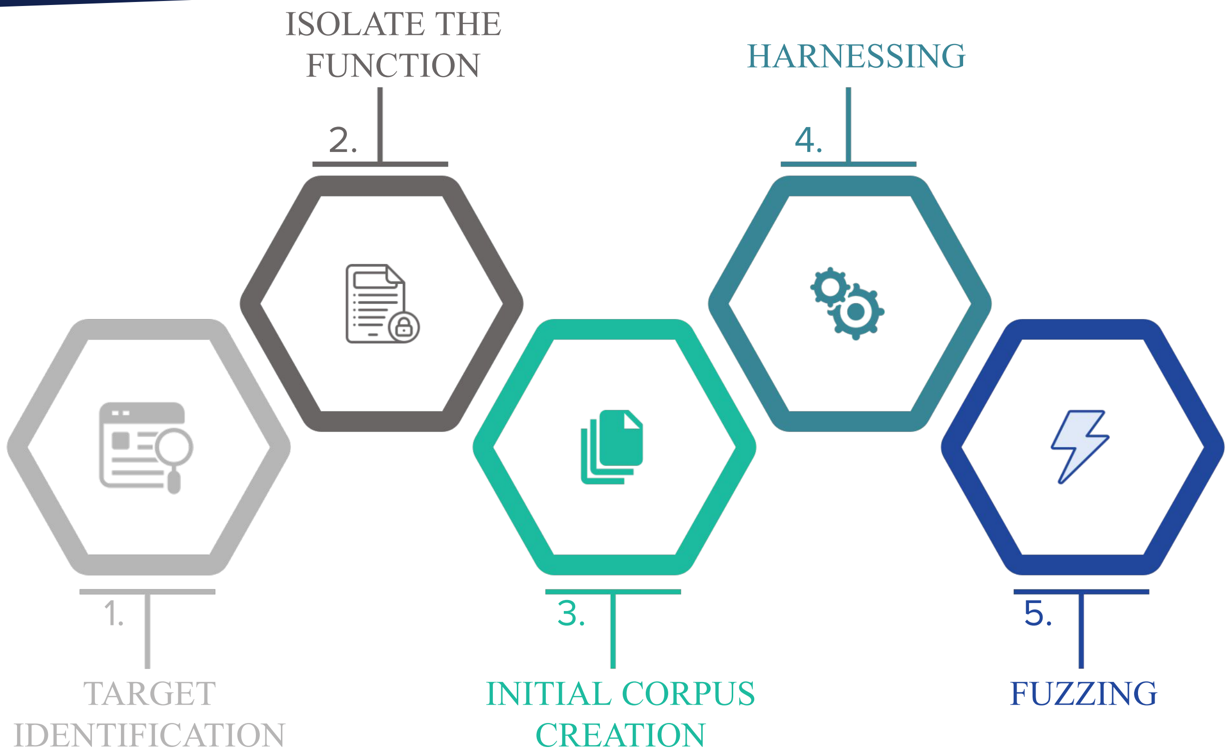
Harness

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *buf, size_t len) {
    char pcap_error_buffer[PCAP_ERRBUF_SIZE];
    const u_char *pkt; struct pcap_pkthdr *hdr;
    u_int16_t p; Host *srcHost = NULL, *dstHost = NULL; Flow *flow = NULL;

    FILE *fd = fmemopen((void *)buf, len, "r");  // Parse the bytes as pcap file
    pcap_t *pcap_handle = pcap_fopen_offline(fd, pcap_error_buffer);
    pcap_setnonblock(pcap_handle, 1, pcap_error_buffer);
    iface->set_datalink(pcap_datalink(pcap_handle));  // Set the pcap handle

    while (pcap_next_ex(pcap_handle, &hdr, &pkt) > 0)
        iface->dissectPacket(IFACE_ID, true, NULL, hdr, pkt, &p, &srcHost, &dstHost, &flow);
    return 0;
}
```

Fuzzing ntopng: The workflow



Eat, sleep, fuzz, repeat

Eat sleep fuzz repeat

LibFuzzer

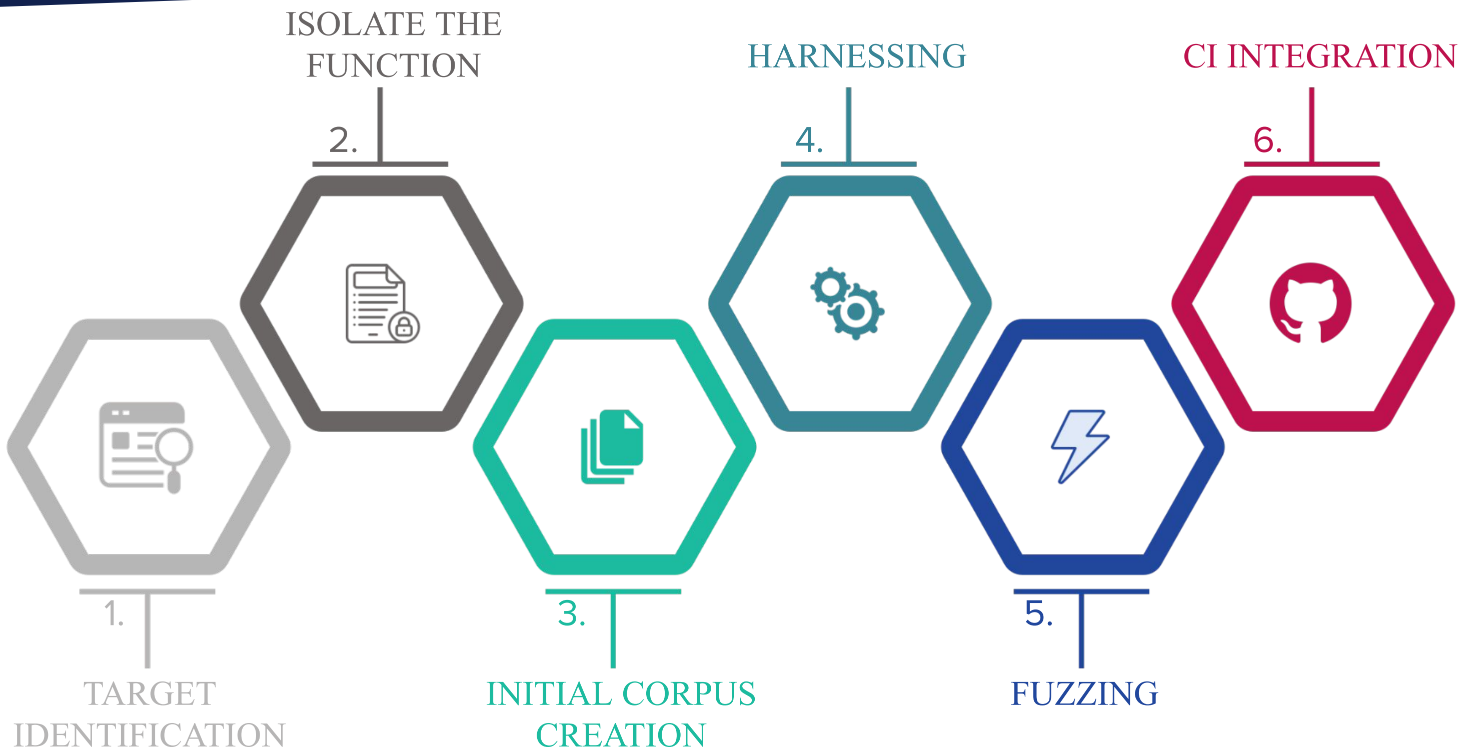
```
#1526   NEW    cov: 2893 ft: 13850 corp: 362/3315Kb lim: 521916 exec/s: 127 rss: 84Mb L: 26490/521916 MS: 1 CrossOver-
#1528   NEW    cov: 2893 ft: 13858 corp: 363/3326Kb lim: 521916 exec/s: 127 rss: 84Mb L: 10483/521916 MS: 2 ChangeBit-PersAutoDi
#1548   REDUCE cov: 2893 ft: 13858 corp: 363/3325Kb lim: 521916 exec/s: 129 rss: 84Mb L: 474/521916 MS: 5 ChangeASCIIInt-CMP-Cop
#1550   NEW    cov: 2893 ft: 13866 corp: 364/3331Kb lim: 521916 exec/s: 129 rss: 84Mb L: 6250/521916 MS: 2 ChangeBit-CopyPart-
#1551   NEW    cov: 2894 ft: 13867 corp: 365/3332Kb lim: 521916 exec/s: 129 rss: 84Mb L: 172/521916 MS: 1 CopyPart-
#1572   NEW    cov: 2894 ft: 13876 corp: 366/3332Kb lim: 521916 exec/s: 131 rss: 84Mb L: 839/521916 MS: 1 InsertRepeatedBytes-
#1601   NEW    cov: 2894 ft: 13880 corp: 367/3334Kb lim: 521916 exec/s: 133 rss: 84Mb L: 1135/521916 MS: 4 ShuffleBytes-ShuffleB
#1602   NEW    cov: 2895 ft: 13882 corp: 368/3334Kb lim: 521916 exec/s: 133 rss: 84Mb L: 115/521916 MS: 1 ShuffleBytes-
#1608   NEW    cov: 2895 ft: 13886 corp: 369/3340Kb lim: 521916 exec/s: 134 rss: 84Mb L: 6055/521916 MS: 1 CopyPart-
#1610   NEW    cov: 2895 ft: 13890 corp: 370/3347Kb lim: 521916 exec/s: 134 rss: 84Mb L: 7365/521916 MS: 2 CrossOver-EraseBytes-
#1632   NEW    cov: 2895 ft: 13892 corp: 371/3351Kb lim: 521916 exec/s: 136 rss: 84Mb L: 3933/521916 MS: 2 InsertRepeatedBytes-C
```

Eat sleep fuzz repeat

AFL++

```
      american fuzzy lop ++4.04c {default} (./fuzz_dissect_packet_afl) [fast]
┌─ process timing ────────────────────────────────────┬─ overall results ────┐
│        run time : 0 days, 0 hrs, 2 min, 51 sec      │  cycles done : 0     │
│   last new find : 0 days, 0 hrs, 0 min, 0 sec       │ corpus count : 1289  │
│last saved crash : none seen yet                     │saved crashes : 0     │
│ last saved hang : 0 days, 0 hrs, 1 min, 42 sec      │  saved hangs : 1     │
├─ cycle progress ─────────────────┬─ map coverage────┴──────────────────────┤
│  now processing : 283.0 (22.0%)  │    map density : 2.01% / 7.74%          │
│  runs timed out : 0 (0.00%)      │ count coverage : 6.81 bits/tuple        │
├─ stage progress ─────────────────┼─ findings in depth ─────────────────────┤
│  now trying : splice 3           │ favored items : 167 (12.96%)            │
│ stage execs : 124/144 (86.11%)   │  new edges on : 263 (20.40%)            │
│ total execs : 430k               │ total crashes : 0 (0 saved)             │
│  exec speed : 3166/sec           │  total tmouts : 1 (0 saved)             │
├─ fuzzing strategy yields ────────┴───────────────┬─ item geometry ─────────┤
│   bit flips : disabled (default, enable with -D) │    levels : 4           │
│  byte flips : disabled (default, enable with -D) │   pending : 1020        │
│ arithmetics : disabled (default, enable with -D) │  pend fav : 142         │
│  known ints : disabled (default, enable with -D) │ own finds : 379         │
│  dictionary : n/a                                │  imported : 0           │
│havoc/splice : 347/306k, 32/84.7k                 │ stability : 31.81%      │
│py/custom/rq : unused, unused, unused, unused     ├─────────────────────────┘
│    trim/eff : 0.02%/26.0k, disabled              │          [cpu000: 18%]
└──────────────────────────────────────────────────┘
```

Fuzzing ntopng: The workflow



Integrate everything in the CI/CD pipeline

Continuous fuzzing

OSS-Fuzz

- **Continuous fuzzing** for open source software for free! *(Sponsored by Google)*
- Support AFL++, Honggfuzz, libfuzzer, Centipede
- Support C/C++, Rust, Go, Python and Java/JVM code
- Easy to integrate within the github CI
- Based on docker containers
- **Reward** program for relevant contribution

Continuous fuzzing

OSS-Fuzz

- **Continuous fuzzing** for open source software for free! *(Sponsored by Google)*
- Support AFL++, Honggfuzz, libfuzzer, Centipede
- Support C/C++, Rust, Go, Python and Java/JVM code
- Easy to integrate within the github CI
- Based on docker containers
- **Reward** program for relevant contribution

The cons

- Everything must be **automatized** (compiling, building the corpus, fuzzing)
- Harder to set up

Conclusion

- Several **bugs** were found
- Coverage based fuzzing proved again to be very powerful
- Integrating the project with oss-fuzz requires some effort but it's definitely worth it

Future work

- Use **structured fuzzing** for more efficient fuzzing
- Choose other targets *(configuration parser, lua scripts)*

Thank you

Contact information:

Email: rmori@quarkslab.com

Website: quarkslab.com

@quarkslab

Quarkslab