Fuzzing ntopng

ntopConf ‘23, Pisa, Italy
Riccardo Mori <rmori@quarkslab.com>
Riccardo Mori

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

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

Three approaches

- Blackbox fuzzing
- Greybox fuzzing
- Whitebox fuzzing

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

Run the software

Generate new input

Provide feedback data
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

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
Identify **interesting** functions (*parsers, cryptographic functions, ...*)
Isolate the target function *(focus on speed and reliability)*
Fuzzing ntopng: The workflow

1. TARGET IDENTIFICATION
2. ISOLATE THE FUNCTION
3. INITIAL CORPUS CREATION

Build an initial corpus of inputs *(good corpus is saving time!)*
Fuzzing ntopng: The workflow

1. TARGET IDENTIFICATION
2. ISOLATE THE FUNCTION
3. INITIAL CORPUS CREATION
4. HARNESSING

Glue code that combines the fuzzer, the target function and the configuration
Fuzzing ntopng: The workflow

1. TARGET IDENTIFICATION
2. ISOLATE THE FUNCTION
3. INITIAL CORPUS CREATION
4. HARNESSING
5. FUZZING

Eat, sleep, fuzz, repeat
Fuzzing ntopng: The workflow

1. TARGET IDENTIFICATION
2. ISOLATE THE FUNCTION
3. INITIAL CORPUS CREATION
4. HARNESSING
5. FUZZING
6. CI INTEGRATION

Integrate everything in the CI/CD pipeline
Identify *interesting* functions (*parsers, cryptographic functions, ...*)
ntopng: an overview
ntopng: an overview

C/C++ Core

Lua HTTP engine

Python Bindings

The target

ntopng

nDPI

nEdge

...
ntopng: choosing the target function
ntopng: choosing the target function
ntopng: choosing the target function
ntopng: choosing the target function

Inspecting the network interface

Ntop
Ntop.cpp

start()
Ntop.cpp

NetworkInterface::startPacketPolling()
NetworkInterface.cpp

NetworkInterface::dissectPacket()
NetworkInterface.cpp

Thread
Packet polling
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
);
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
Isolate the target function

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

```c
#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

1. TARGET IDENTIFICATION
2. ISOLATE THE FUNCTION
3. INITIAL CORPUS CREATION

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
Glue code that combines the fuzzers, 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
Harnessing LibFuzzer interface

```c
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
```
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;
}
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

1. TARGET IDENTIFICATION
2. ISOLATE THE FUNCTION
3. INITIAL CORPUS CREATION
4. HARNESSING
5. FUZZING

Eat, sleep, fuzz, repeat
# Eat sleep fuzz repeat

## LibFuzzer

<table>
<thead>
<tr>
<th>#</th>
<th>Status</th>
<th>Line</th>
<th>Function</th>
<th>Coverage</th>
<th>File Size</th>
<th>Limit</th>
<th>Exec/s</th>
<th>RSS</th>
<th>MB</th>
<th>Memory</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1526</td>
<td>NEW</td>
<td>2893</td>
<td>ft: 13850</td>
<td>corp: 362/3315Kb</td>
<td>lim: 521916</td>
<td>exec/s: 127</td>
<td>rss: 84Mb</td>
<td>L: 26490/521916</td>
<td>MS: 1</td>
<td>CrossOver-</td>
<td></td>
</tr>
<tr>
<td>#1550</td>
<td>NEW</td>
<td>2893</td>
<td>ft: 13866</td>
<td>corp: 364/3331Kb</td>
<td>lim: 521916</td>
<td>exec/s: 129</td>
<td>rss: 84Mb</td>
<td>L: 6250/521916</td>
<td>MS: 2</td>
<td>ChangeBit-CopyPart-</td>
<td></td>
</tr>
<tr>
<td>#1551</td>
<td>NEW</td>
<td>2894</td>
<td>ft: 13867</td>
<td>corp: 365/3332Kb</td>
<td>lim: 521916</td>
<td>exec/s: 129</td>
<td>rss: 84Mb</td>
<td>L: 172/521916</td>
<td>MS: 1</td>
<td>CopyPart-</td>
<td></td>
</tr>
<tr>
<td>#1572</td>
<td>NEW</td>
<td>2894</td>
<td>ft: 13876</td>
<td>corp: 366/3332Kb</td>
<td>lim: 521916</td>
<td>exec/s: 151</td>
<td>rss: 84Mb</td>
<td>L: 839/521916</td>
<td>MS: 1</td>
<td>InsertRepeatedBytes-</td>
<td></td>
</tr>
<tr>
<td>#1601</td>
<td>NEW</td>
<td>2894</td>
<td>ft: 13880</td>
<td>corp: 367/3334Kb</td>
<td>lim: 521916</td>
<td>exec/s: 133</td>
<td>rss: 84Mb</td>
<td>L: 1135/521916</td>
<td>MS: 4</td>
<td>ShuffleBytes-ShuffleByte-</td>
<td></td>
</tr>
<tr>
<td>#1602</td>
<td>NEW</td>
<td>2895</td>
<td>ft: 13882</td>
<td>corp: 368/3334Kb</td>
<td>lim: 521916</td>
<td>exec/s: 133</td>
<td>rss: 84Mb</td>
<td>L: 115/521916</td>
<td>MS: 1</td>
<td>ShuffleBytes-</td>
<td></td>
</tr>
<tr>
<td>#1603</td>
<td>NEW</td>
<td>2895</td>
<td>ft: 13886</td>
<td>corp: 369/3340Kb</td>
<td>lim: 521916</td>
<td>exec/s: 134</td>
<td>rss: 84Mb</td>
<td>L: 6055/521916</td>
<td>MS: 1</td>
<td>CopyPart-</td>
<td></td>
</tr>
<tr>
<td>#1610</td>
<td>NEW</td>
<td>2895</td>
<td>ft: 13890</td>
<td>corp: 370/3347Kb</td>
<td>lim: 521916</td>
<td>exec/s: 134</td>
<td>rss: 84Mb</td>
<td>L: 7365/521916</td>
<td>MS: 2</td>
<td>CrossOver-EraseBytes-</td>
<td></td>
</tr>
<tr>
<td>#1632</td>
<td>NEW</td>
<td>2895</td>
<td>ft: 13892</td>
<td>corp: 371/3351Kb</td>
<td>lim: 521916</td>
<td>exec/s: 136</td>
<td>rss: 84Mb</td>
<td>L: 3933/521916</td>
<td>MS: 2</td>
<td>InsertRepeatedBytes-</td>
<td></td>
</tr>
</tbody>
</table>
Eat sleep fuzz repeat

AFL++
Fuzzing ntopng: The workflow

1. TARGET IDENTIFICATION
2. ISOLATE THE FUNCTION
3. INITIAL CORPUS CREATION
4. HARNESSING
5. FUZZING
6. CI INTEGRATION

Integrate everything in the CI/CD pipeline
Continuous fuzzing

➤ **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