Fuzzing Janus for Fun and Profit

A. Amirante, T. Castaldi, L. Miniero, S.P. Romano, P. Saviano, A. Toppi

IPTComm 2019
October 15\textsuperscript{th} 2019, Chicago, IL, USA
A few words about me

Lorenzo Miniero

- Ph.D @ UniNA
- Chairman @ Meetecho
- Main author of Janus®

Contacts and info

- lorenzo@meetecho.com
- https://twitter.com/elminiero
- https://www.slideshare.net/LorenzoMiniero
Vulnerabilities in RTC communications

- Project Zero is a team of security analysts employed by Google
  - [https://googleprojectzero.blogspot.com/](https://googleprojectzero.blogspot.com/)

- Recently focused on videoconferencing applications
  - Focus on end-to-end, and RTP testing
  - Malicious endpoint generating randomized input
  - Built new tools required for the task

- Targeted many applications, and found dangerous bugs
  - Apple FaceTime
  - WhatsApp
  - WebRTC

Philipp Hancke’s wakeup call
Vulnerabilities in RTC communications

- Project Zero is a team of security analysts employed by Google
  - https://googleprojectzero.blogspot.com/

- Recently focused on videoconferencing applications
  - Focus on end-to-end, and RTP testing
  - Malicious endpoint generating randomized input
  - Built new tools required for the task

- Targeted many applications, and found dangerous bugs
  - Apple FaceTime
  - WhatsApp
  - WebRTC

Philipp Hancke’s wakeup call

Vulnerabilities in RTC communications

• Project Zero is a team of security analysts employed by Google
  • [https://googleprojectzero.blogspot.com/](https://googleprojectzero.blogspot.com/)

• Recently focused on videoconferencing applications
  • Focus on end-to-end, and RTP testing
  • Malicious endpoint generating randomized input
  • Built new tools required for the task

• Targeted many applications, and found dangerous bugs
  • Apple FaceTime
  • WhatsApp
  • **WebRTC**

Philipp Hancke’s wakeup call
Vulnerabilities in RTC communications

- Project Zero is a team of security analysts employed by Google
  - https://googleprojectzero.blogspot.com/

- Recently focused on videoconferencing applications
  - Focus on end-to-end, and RTP testing
  - Malicious endpoint generating randomized input
  - Built new tools required for the task

- Targeted many applications, and found dangerous bugs
  - Apple FaceTime
  - WhatsApp
  - WebRTC

Philipp Hancke’s wakeup call
What can cause trouble in WebRTC?

- WebRTC is signalling agnostic, so typically not that
  - You can use SIP, XMPP, some JSON flavour, etc.
- A lot of media-related protocols to worry about, though
  - STUN/TURN (NAT traversal)
  - DTLS/DTLS-SRTP (secure exchange of keys and data)
  - RTP/RTCP (or actually, SRTP/SRTCP), including RTP extensions
  - SCTP (data channels)
- ... and codec specific payloads
  - Identifying keyframes (VP8, VP9, H.264)
  - Simulcast & SVC (inspecting payloads)
What can cause trouble in WebRTC?

- WebRTC is signalling agnostic, so typically not that
  - You can use SIP, XMPP, some JSON flavour, etc.
- A lot of media-related protocols to worry about, though
  - STUN/TURN (NAT traversal)
  - DTLS/DTLS-SRTP (secure exchange of keys and data)
  - RTP/RTCP (or actually, SRTP/SRTCP), including RTP extensions
  - SCTP (data channels)
- ... and codec specific payloads
  - Identifying keyframes (VP8, VP9, H.264)
  - Simulcast & SVC (inspecting payloads)
What can cause trouble in WebRTC?

- WebRTC is signalling agnostic, so typically not that
  - You can use SIP, XMPP, some JSON flavour, etc.

- A lot of media-related protocols to worry about, though
  - STUN/TURN (NAT traversal)
  - DTLS/DTLS-SRTP (secure exchange of keys and data)
  - RTP/RTCP (or actually, SRTP/SRTCP), including RTP extensions
  - SCTP (data channels)

- ... and codec specific payloads
  - Identifying keyframes (VP8, VP9, H.264)
  - Simulcast & SVC (inspecting payloads)
Why can fuzz testing help?

- Automated software testing technique
  - Unexpected or invalid data submitted to a program
  - Input pattern modified according to a defined strategy (e.g., for coverage)

- Typical workflow
  1. Engine generates input based on existing dataset (“Corpus”)
  2. Input mutated slightly over time
  3. Input data passed to target function and monitored (e.g., via sanitizers)
  4. Coverage of new lines updates stats and Corpus (new pattern)
  5. Repeat until it crashes

- Repeatability can be ensured using the same seeds or previous dumps
Why can fuzz testing help?

- Automated software testing technique
  - Unexpected or invalid data submitted to a program
  - Input pattern modified according to a defined strategy (e.g., for coverage)

- Typical workflow
  1. Engine generates input based on existing dataset (“Corpus”)
  2. Input mutated slightly over time
  3. Input data passed to target function and monitored (e.g., via sanitizers)
  4. Coverage of new lines updates stats and Corpus (new pattern)
  5. Repeat until it crashes

- Repeatability can be ensured using the same seeds or previous dumps
Why can fuzz testing help?

• Automated software testing technique
  • Unexpected or invalid data submitted to a program
  • Input pattern modified according to a defined strategy (e.g., for coverage)

• Typical workflow
  1. Engine generates input based on existing dataset (“Corpus”)
  2. Input mutated slightly over time
  3. Input data passed to target function and monitored (e.g., via sanitizers)
  4. Coverage of new lines updates stats and Corpus (new pattern)
  5. Repeat until it crashes

• Repeatability can be ensured using the same seeds or previous dumps
Introducing the Janus WebRTC server

Janus

General purpose, open source WebRTC server

- [https://github.com/meetecho/janus-gateway](https://github.com/meetecho/janus-gateway)
- Demos and documentation: [https://janus.conf.meetecho.com](https://janus.conf.meetecho.com)
- Community: [https://groups.google.com/forum/#!forum/meetecho-janus](https://groups.google.com/forum/#!forum/meetecho-janus)
Modular architecture

- The core only implements the WebRTC stack
  - JSEP/SDP, ICE, DTLS-SRTP, Data Channels, Simulcast, VP9-SVC, ...
- Plugins expose Janus API over different “transports”
  - Currently HTTP / WebSockets / RabbitMQ / Unix Sockets / MQTT / Nanomsg
- “Application” logic implemented in plugins too
  - Users attach to plugins via the Janus core
  - The core handles the WebRTC stuff
  - Plugins route/manipulate the media/data
- Plugins can be combined on client side as “bricks”
  - Video SFU, Audio MCU, SIP gatewaying, broadcasting, etc.
Modular architecture

- The core only implements the WebRTC stack
  - JSEP/SDP, ICE, DTLS-SRTP, Data Channels, Simulcast, VP9-SVC, ...
- Plugins expose Janus API over different “transports”
  - Currently HTTP / WebSockets / RabbitMQ / Unix Sockets / MQTT / Nanomsg
- “Application” logic implemented in plugins too
  - Users attach to plugins via the Janus core
  - The core handles the WebRTC stuff
  - Plugins route/manipulate the media/data
- Plugins can be combined on client side as “bricks”
  - Video SFU, Audio MCU, SIP gatewaying, broadcasting, etc.
Modular architecture

• The core only implements the WebRTC stack
  • JSEP/SDP, ICE, DTLS-SRTP, Data Channels, Simulcast, VP9-SVC, ...

• Plugins expose Janus API over different “transports”
  • Currently HTTP / WebSockets / RabbitMQ / Unix Sockets / MQTT / Nanomsg

• “Application” logic implemented in plugins too
  • Users attach to plugins via the Janus core
  • The core handles the WebRTC stuff
  • Plugins route/manipulate the media/data

• Plugins can be combined on client side as “bricks”
  • Video SFU, Audio MCU, SIP gatewaying, broadcasting, etc.
Modular architecture

• The core only implements the WebRTC stack
  • JSEP/SDP, ICE, DTLS-SRTP, Data Channels, Simulcast, VP9-SVC, ...

• Plugins expose Janus API over different “transports”
  • Currently HTTP / WebSockets / RabbitMQ / Unix Sockets / MQTT / Nanomsg

• “Application” logic implemented in plugins too
  • Users attach to plugins via the Janus core
  • The core handles the WebRTC stuff
  • Plugins route/manipulate the media/data

• Plugins can be combined on client side as “bricks”
  • Video SFU, Audio MCU, SIP gatewaying, broadcasting, etc.
Choosing the fuzzing targets

• Many protocols via dependencies are fuzzed already
  • ICE/STUN/TURN (libnice)
  • DTLS/DTLS-SRTP (OpenSSL/LibreSSL/BoringSSL)
  • SRTP/SRTCP (libsrtpt)
  • SCTP (usrscptplib)

• Some other dependencies MAY need fuzzing (but not in Janus?)
  • Transports (HTTP, WebSockets, RabbitMQ, etc.)
  • JSON support (Jansson)

• Custom code DEFINITELY needs fuzzing
  • RTCP parsing (e.g., compound packets)
  • RTP processing (e.g., RTP extensions, codec specific payloads)
  • SDP parsing and processing
Choosing the fuzzing targets

- Many protocols via dependencies are fuzzed already
  - ICE/STUN/TURN (libnice)
  - DTLS/DTLS-SRTP (OpenSSL/LibreSSL/BoringSSL)
  - SRTP/SRTCP (libsrt)  
  - SCTP (usrscplib)

- Some other dependencies MAY need fuzzing (but not in Janus?)
  - Transports (HTTP, WebSockets, RabbitMQ, etc.)
  - JSON support (Jansson)

- Custom code DEFINITELY needs fuzzing
  - RTCP parsing (e.g., compound packets)
  - RTP processing (e.g., RTP extensions, codec specific payloads)
  - SDP parsing and processing
Choosing the fuzzing targets

- Many protocols via dependencies are fuzzed already
  - ICE/STUN/TURN (libnice)
  - DTLS/DTLS-SRTP (OpenSSL/LibreSSL/BoringSSL)
  - SRTP/SRTCP (libsrtp)
  - SCTP (usrsctplib)

- Some other dependencies MAY need fuzzing (but not in Janus?)
  - Transports (HTTP, WebSockets, RabbitMQ, etc.)
  - JSON support (Jansson)

- Custom code DEFINITELY needs fuzzing
  - RTCP parsing (e.g., compound packets)
  - RTP processing (e.g., RTP extensions, codec specific payloads)
  - SDP parsing and processing
A quick intro to libFuzzer

- Popular coverage-guided fuzzing engine, part of the LLVM project
  - [https://llvm.org/docs/LibFuzzer.html](https://llvm.org/docs/LibFuzzer.html)

- Used by several well known applications
  - glibc, OpenSSL/LibreSSL/BoringSSL, SQLite, FFmpeg and many more

- A few key characteristics
  - Needs sources to be compiled with Clang
  - Works in-process (linked with the library/application under test)
  - Feeds inputs to the target via a fuzzing entrypoint (target function)
  - Execution of the target function is monitored with sanitizers tools (e.g., libasan)
A quick intro to libFuzzer

- Popular coverage-guided fuzzing engine, part of the LLVM project
  - [https://llvm.org/docs/LibFuzzer.html](https://llvm.org/docs/LibFuzzer.html)
- Used by several well known applications
  - glibc, OpenSSL/LibreSSL/BoringSSL, SQLite, FFmpeg and many more

- A few key characteristics
  - Needs sources to be compiled with Clang
  - Works in-process (linked with the library/application under test)
  - Feeds inputs to the target via a fuzzing entrypoint (target function)
  - Execution of the target function is monitored with sanitizers tools (e.g., libasan)
A quick intro to libFuzzer

- Popular coverage-guided fuzzing engine, part of the LLVM project
  - [https://llvm.org/docs/LibFuzzer.html](https://llvm.org/docs/LibFuzzer.html)
- Used by several well known applications
  - glibc, OpenSSL/ LibreSSL/ BoringSSL, SQLite, FFmpeg and many more
- A few key characteristics
  - Needs sources to be compiled with Clang
  - Works in-process (linked with the library/application under test)
  - Feeds inputs to the target via a fuzzing entrypoint (target function)
  - Execution of the target function is monitored with sanitizers tools (e.g., libasan)
Coverage-guided fuzzing
libFuzzer in (simplified) practice

1. Implement the method to receive and process the input data

```c
// my_fuzzer.c
int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
    ProcessData(Data, Size);
    return 0;
}
```

2. Compile with Clang and the right flags

```bash
> clang -g -O1 -fsanitize=fuzzer,address,undefined my_fuzzer.c
```

3. Launch passing the Corpus folder as the argument

```bash
> ./my_fuzzer CORPUS_DIR
```

4. In case of crashes, pass the dumped input (e.g., via gdb, or to test regressions)

```bash
> gdb --args ./my_fuzzer crash-file-dump
```
libFuzzer in (simplified) practice

1. Implement the method to receive and process the input data

```c
// my_fuzzer.c
int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
    ProcessData(Data, Size);
    return 0;
}
```

2. Compile with Clang and the right flags

```bash
> clang -g -O1 -fsanitize=fuzzer,address,undefined my_fuzzer.c
```

3. Launch passing the Corpus folder as the argument

```bash
> ./my_fuzzer CORPUS_DIR
```

4. In case of crashes, pass the dumped input (e.g., via gdb, or to test regressions)

```bash
> gdb --args ./my_fuzzer crash-file-dump
```
libFuzzer in (simplified) practice

1. Implement the method to receive and process the input data

```c
// my_fuzzer.c
int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
    ProcessData(Data, Size);
    return 0;
}
```

2. Compile with Clang and the right flags

```
> clang -g -O1 -fsanitize=fuzzer,address,undefined my_fuzzer.c
```

3. Launch passing the Corpus folder as the argument

```
> ./my_fuzzer CORPUS_DIR
```

4. In case of crashes, pass the dumped input (e.g., via gdb, or to test regressions)

```
> gdb --args ./my_fuzzer crash-file-dump
```
libFuzzer in (simplified) practice

1. Implement the method to receive and process the input data

   // my_fuzzer.c
   int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
       ProcessData(Data, Size);
       return 0;
   }

2. Compile with Clang and the right flags

   > clang -g -O1 -fsanitize=fuzzer,address,undefined my_fuzzer.c

3. Launch passing the Corpus folder as the argument

   > ./my_fuzzer CORPUS_DIR

4. In case of crashes, pass the dumped input (e.g., via gdb, or to test regressions)

   > gdb --args ./my_fuzzer crash-file-dump
Integrating libFuzzer in Janus

- First step was Clang support (Janus normally built with gcc)
  - Streamlined compilation flags in the process
  - Got useful warnings that led to some fixes too

- Next step was choosing what to fuzz
  - Decided to start with RTCP
  - Compound packets + length values + overflows = “fun”...

- Then worked on the libFuzzer workflow
  1. Fuzzing target with critical RTCP-related functions
  2. Helper script to build the fuzzer
  3. Helper script to run the fuzzer

Original pull request (now merged, with RTP and SDP fuzzing as well)
https://github.com/meetecho/janus-gateway/pull/1492
Integrating libFuzzer in Janus

- First step was Clang support (Janus normally built with gcc)
  - Streamlined compilation flags in the process
  - Got useful warnings that led to some fixes too

- Next step was choosing what to fuzz
  - Decided to start with RTCP
  - Compound packets + length values + overflows = “fun”...

- Then worked on the libFuzzer workflow
  1. Fuzzing target with critical RTCP-related functions
  2. Helper script to build the fuzzer
  3. Helper script to run the fuzzer

Original pull request (now merged, with RTP and SDP fuzzing as well)
https://github.com/meeteecho/janus-gateway/pull/1492
Integrating libFuzzer in Janus

- First step was Clang support (Janus normally built with gcc)
  - Streamlined compilation flags in the process
  - Got useful warnings that led to some fixes too

- Next step was choosing what to fuzz
  - Decided to start with RTCP
  - Compound packets + length values + overflows = “fun”...

- Then worked on the libFuzzer workflow
  1. Fuzzing target with critical RTCP-related functions
  2. Helper script to build the fuzzer
  3. Helper script to run the fuzzer

Original pull request (now merged, with RTP and SDP fuzzing as well)
https://github.com/meetecho/janus-gateway/pull/1492
Integrating libFuzzer in Janus

- First step was Clang support (Janus normally built with gcc)
  - Streamlined compilation flags in the process
  - Got useful warnings that led to some fixes too

- Next step was choosing what to fuzz
  - Decided to start with RTCP
  - Compound packets + length values + overflows = “fun”...

- Then worked on the libFuzzer workflow
  1. Fuzzing target with critical RTCP-related functions
  2. Helper script to build the fuzzer
  3. Helper script to run the fuzzer

Original pull request (now merged, with RTP and SDP fuzzing as well)
https://github.com/meetecho/janus-gateway/pull/1492
// fuzz-rtcp.c
#include "janus/rtcp.h"

int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
    if (size < 8 || size > 1472)
        return 0;
    if (!janus_is_rtcp(data, size))
        return 0;
    /* Initialize an empty RTCP context */
    janus_rtcp_context ctx;
    janus_rtcp_parse(ctx, (char *)data, size);
    GSList *list = janus_rtcp_get_nacks((char *)data, size);
    ...
    if (list)
        g_slist_free(list);
    return 0;
}
Presenting the code coverage

```c
1.00k  //
1.00k  }
228   13  if (len < (int)sizeof(janus_rtcp_header) + (int)sizeof(uint32_t)) {
229   13    JANUS_LOG(LOG_VERB, "Packet size is too small (%d bytes) to contain RTCP\n", len);
230   13    return FALSE;
231   13  }
232  995  int header_def_len = 4*sizeof(rtcp->length) + 4;
233  995  if (len < header_def_len) {
234    78    JANUS_LOG(LOG_VERB, "Invalid RTCP packet defined length, expected %d bytes > actual %d bytes\n", header_def_len, len);
235    78    return FALSE;
236    78  }
237    917  return TRUE;
238  917 }
239  917 }
240  12  }
241  12  if (len < (int)sizeof(janus_rtcp_header) + (int)sizeof(uint32_t) + (int)sizeof(sender_info)) {
242    8    JANUS_LOG(LOG_VERB, "RTCP Packet is too small (%d bytes) to contain SR\n", len);
243    8    return FALSE;
244    8  }
245    8  }
246    8  }
247    8    if (actual_rb_len < header_rb_len) {
248    8    JANUS_LOG(LOG_VERB, "SR got %d RB count, expected %d bytes > actual %d bytes\n", rtcp->rc, header_rb_len, actual_rb_len);
249    8    return FALSE;
250    8  }
251    12  return TRUE;
252    12  }
253    12  }
254  24  }
```
Corpora files: a shared effort

https://github.com/RTC-Cartel/webrtc-fuzzer-corpora
Scalable distributed fuzzing via OSS-Fuzz

https://github.com/google/oss-fuzz/pull/2241 (Janus addition)
Scalable distributed fuzzing via OSS-Fuzz

Welcome

Welcome to ClusterFuzz, the fuzzing infrastructure behind OSS-Fuzz. Here you can look at crashes, statistics, and coverage information for your fuzzers. Below is an overview of your projects and their fuzzing configurations.

Janus Gateway

afl_asan_janus-gateway
Fuzzing engine: AFL
Sanitizer: address (ASAN)

libfuzzer_asan_janus-gateway
Fuzzing engine: libFuzzer
Sanitizer: address (ASAN)

libfuzzer_msan_janus-gateway
Fuzzing engine: libFuzzer
Sanitizer: memory (MSAN)

libfuzzer_ubsan_janus-gateway
Fuzzing engine: libFuzzer
Sanitizer: undefined (UBSAN)

https://github.com/google/oss-fuzz/pull/2241 (Janus addition)
A detailed tutorial on how to setup all this

https://webrtchacks.com/fuzzing-janus/
What’s next?

- So far, we only fuzzed RTP, RTCP and in part SDP in the core
  - SDP fuzzing should be improved (maybe with structure-aware fuzzing?)
  - What about plugins and their custom interactions?
- Definitely expand the corpora
  - The shared RTC-Cartel repo should help with that
  - Exchanging crash causes with other projects will make both more robust
- libFuzzer is not the only option here
  - Some popular alternatives are AFL, Radamsa, Gasoline, etc.
  - KITE and its “weaponised” browsers can be very helpful as an orthogonal testing tool
What’s next?

• So far, we only fuzzed RTP, RTCP and in part SDP in the core
  • SDP fuzzing should be improved (maybe with structure-aware fuzzing?)
  • What about plugins and their custom interactions?

• Definitely expand the corpora
  • The shared RTC-Cartel repo should help with that
  • Exchanging crash causes with other projects will make both more robust

• libFuzzer is not the only option here
  • Some popular alternatives are AFL, Radamsa, Gasoline, etc.
  • KITE and its “weaponised” browsers can be very helpful as an orthogonal testing tool
What’s next?

- So far, we only fuzzed RTP, RTCP and in part SDP in the core
  - SDP fuzzing should be improved (maybe with structure-aware fuzzing?)
  - What about plugins and their custom interactions?

- Definitely expand the corpora
  - The shared RTC-Cartel repo should help with that
  - Exchanging crash causes with other projects will make both more robust

- libFuzzer is not the only option here
  - Some popular alternatives are AFL, Radamsa, Gasoline, etc.
  - KITE and its “weaponised” browsers can be very helpful as an orthogonal testing tool
Thanks! Questions? Comments?

Get in touch!
- https://twitter.com/elminierbo
- https://twitter.com/meetecho
- https://www.meetecho.com