KAMAILIO WORLD
CONFERENCE & EXHIBITION

What The Fuzz! Or Why You Should Really Fuzz Your RTC Code

Lorenzo Miniero
@elminiero

Kamailio World
May 7th 2019, Berlin

A fuzzy intro

Lorenzo Miniero

- Ph.D @ UniNA
- Chairman @ Meetecho
- Barber shop avoider

Contacts and info

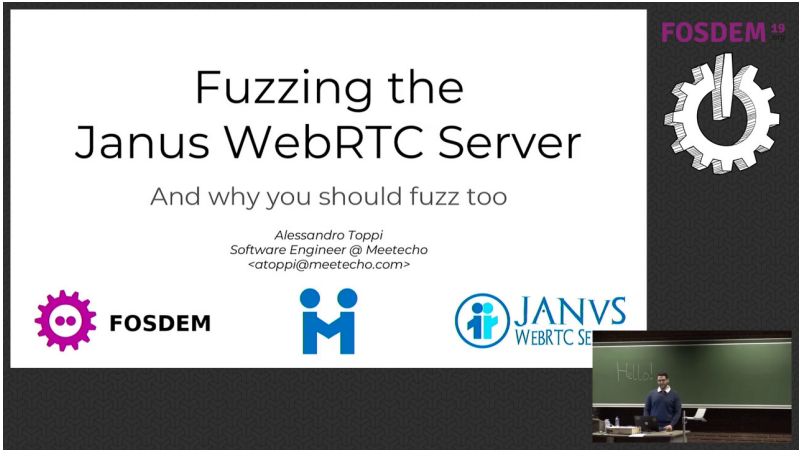
- lorenzo@meetecho.com
- https://twitter.com/elminiero
- https://www.slideshare.net/LorenzoMiniero

Just a few words on Meetecho

- Co-founded in 2009 as an academic spin-off
  - University research efforts brought to the market
  - Completely independent from the University
- Focus on real-time multimedia applications
  - Strong perspective on standardization and open source
- Several activities
  - Consulting services
  - Commercial support and Janus licenses
  - Streaming of live events (IETF, ACM, etc.)
- Proudly brewed in sunny Napoli(*), Italy

(*You may have heard of it)

Kudos to Alessandro Toppi for this content!



Hot topic in Kamailio talks already!

https://www.youtube.com/watch?v=bhy7-uxZGqk

Kamailio lovers know fuzzing already

https://www.youtube.com/watch?v=CuxKD5zljVI

Why another talk on fuzzing, then?

- 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 **(yikes!)**

Philipp Hancke's wakeup call (crashing Janus of all things!)

https://webrtchacks.com/lets-get-better-at-fuzzing-in-2019-heres-how/

Why another talk on fuzzing, then?

- 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 **(yikes!)**

Philipp Hancke's wakeup call (crashing Janus of all things!)

https://webrtchacks.com/lets-get-better-at-fuzzing-in-2019-heres-how/

Why another talk on fuzzing, then?

- 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 **(yikes!)**

Philipp Hancke's wakeup call (crashing Janus of all things!)

https://webrtchacks.com/lets-get-better-at-fuzzing-in-2019-heres-how/

Why another talk on fuzzing, then?

- 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 **(yikes!)**

Philipp Hancke's wakeup call (crashing Janus of all things!)

https://webrtchacks.com/lets-get-better-at-fuzzing-in-2019-heres-how/

Project Zero scaring the fuzz out of us

- In Kamailio, focus is on SIP/SDP signalling, of course
  - Media often taken care of in other components
- WebRTC is signalling agnostic, though
  - You can use SIP, or XMPP, or some JSON flavour, or [INSERT_PROTOCOL]
- A lot of media-related protocols to worry about instead!
  - 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)
  - VP8 simulcast (VP8 payload descriptor)
  - VP9 SVC (VP9 payload descriptor)

Project Zero scaring the fuzz out of us

- In Kamailio, focus is on SIP/SDP signalling, of course
  - Media often taken care of in other components
- WebRTC is signalling agnostic, though
  - You can use SIP, or XMPP, or some JSON flavour, or [INSERT_PROTOCOL]
- A lot of media-related protocols to worry about instead!
  - 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)
  - VP8 simulcast (VP8 payload descriptor)
  - VP9 SVC (VP9 payload descriptor)

Project Zero scaring the fuzz out of us

- In Kamailio, focus is on SIP/SDP signalling, of course
  - Media often taken care of in other components
- WebRTC is signalling agnostic, though
  - You can use SIP, or XMPP, or some JSON flavour, or [INSERT_PROTOCOL]
- A lot of media-related protocols to worry about instead!
  - 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)
  - VP8 simulcast (VP8 payload descriptor)
  - VP9 SVC (VP9 payload descriptor)

Project Zero scaring the fuzz out of us

- In Kamailio, focus is on SIP/SDP signalling, of course
  - Media often taken care of in other components
- WebRTC is signalling agnostic, though
  - You can use SIP, or XMPP, or some JSON flavour, or [INSERT_PROTOCOL]
- A lot of media-related protocols to worry about instead!
  - 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)
  - VP8 simulcast (VP8 payload descriptor)
  - VP9 SVC (VP9 payload descriptor)

Ok, we're scared now... what is fuzz testing?

- 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
  2. Pattern mutated depending on existing dataset ("Corpus")
  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

Ok, we're scared now... what is fuzz testing?

- 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
  2. Pattern mutated depending on existing dataset ("Corpus")
  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

Ok, we're scared now... what is fuzz testing?

- 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
  2. Pattern mutated depending on existing dataset ("Corpus")
  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

Enter Janus!

Janus

General purpose, open source WebRTC server

- https://github.com/meetecho/janus-gateway
- Demos and documentation: https://janus.conf.meetecho.com
- Community: 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.

What should we fuzz, here?

- 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)
- Others were done by us, so DEFINITELY need fuzzing ☺
  - RTCP parsing (e.g., compound packets)
  - RTP processing (e.g., RTP extensions, codec specific payloads)
  - SDP parsing and processing

What should we fuzz, here?

- 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)
- Others were done by us, so DEFINITELY need fuzzing ☺
  - RTCP parsing (e.g., compound packets)
  - RTP processing (e.g., RTP extensions, codec specific payloads)
  - SDP parsing and processing

What should we fuzz, here?

- 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)
- Others were done by us, so DEFINITELY need 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
- Used by several well known applications
  - glibc, OpenSSL/LibreSSL/BoringSSL, SQLite, FFmpeg and many more
  - Made sense for us to have a look at it too!
- 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)

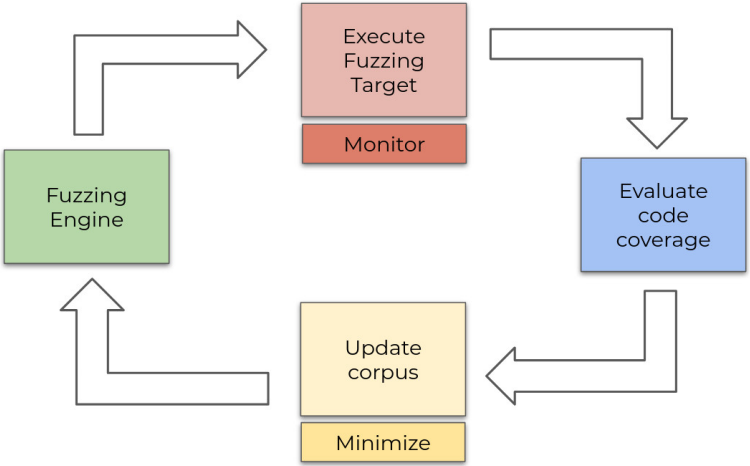
A quick intro to libFuzzer

- Popular coverage-guided fuzzing engine, part of the LLVM project
  - https://llvm.org/docs/LibFuzzer.html
- Used by several well known applications
  - glibc, OpenSSL/LibreSSL/BoringSSL, SQLite, FFmpeg and many more
  - Made sense for us to have a look at it too!
- 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
- Used by several well known applications
  - glibc, OpenSSL/LibreSSL/BoringSSL, SQLite, FFmpeg and many more
  - Made sense for us to have a look at it too!
- 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)

Wait, “coverage-guided fuzzing”?



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
```

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
```

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
```

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 <u>what</u> 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 <u>what</u> 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 <u>what</u> 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

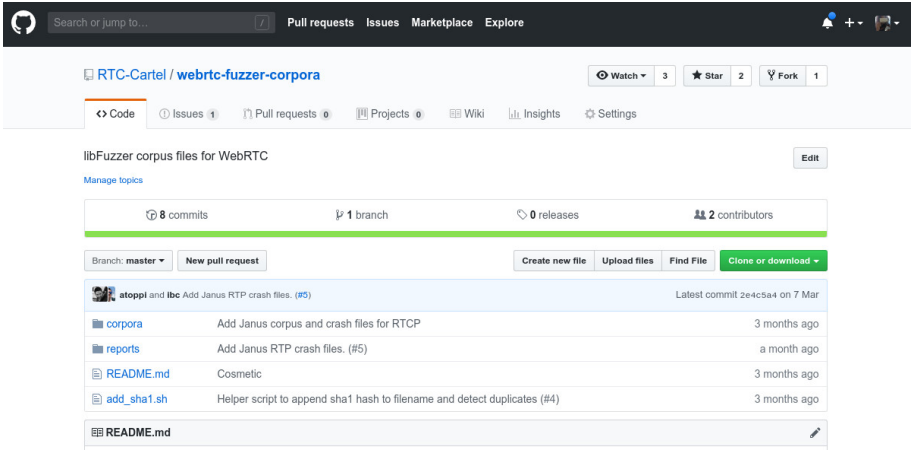
Integrating libFuzzer in Janus

```
// 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

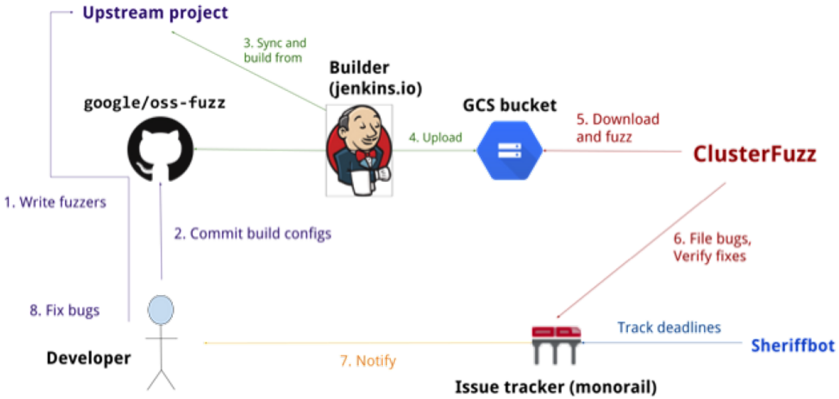
```
224      0   }
225     12 }
226
227  1.00k gboolean janus_rtcp_check_len(janus_rtcp_header *rtcp, int len) {
228  1.00k   if (len < (int)sizeof(janus_rtcp_header) + (int)sizeof(uint32_t)) {
229     13     JANUS_LOG(LOG_VERB, "Packet size is too small (%d bytes) to contain RTCP\n", len);
230     13     return FALSE;
231     13   }
232    995   int header_def_len = 4*(int)ntohs(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
240     12 gboolean janus_rtcp_check_sr(janus_rtcp_header *rtcp, int len) {
241     12   if (len < (int)sizeof(janus_rtcp_header) + (int)sizeof(uint32_t) + (int)sizeof(sender_info)) {
242      0     JANUS_LOG(LOG_VERB, "RTCP Packet is too small (%d bytes) to contain SR\n", len);
243      0     return FALSE;
244      0   }
245     12   int header_rb_len = (int)(rtcp->rc)*(int)sizeof(report_block);
246     12   int actual_rb_len = len - (int)sizeof(janus_rtcp_header) - (int)sizeof(uint32_t) - (int)sizeof(sender_info);
247     12   if (actual_rb_len < header_rb_len) {
248      0     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      0     return FALSE;
250      0   }
251     12   return TRUE;
252     12 }
253
254     24 gboolean janus_rtcp_check_rr(janus_rtcp_header *rtcp, int len) {
```

Corpora files: a shared effort!



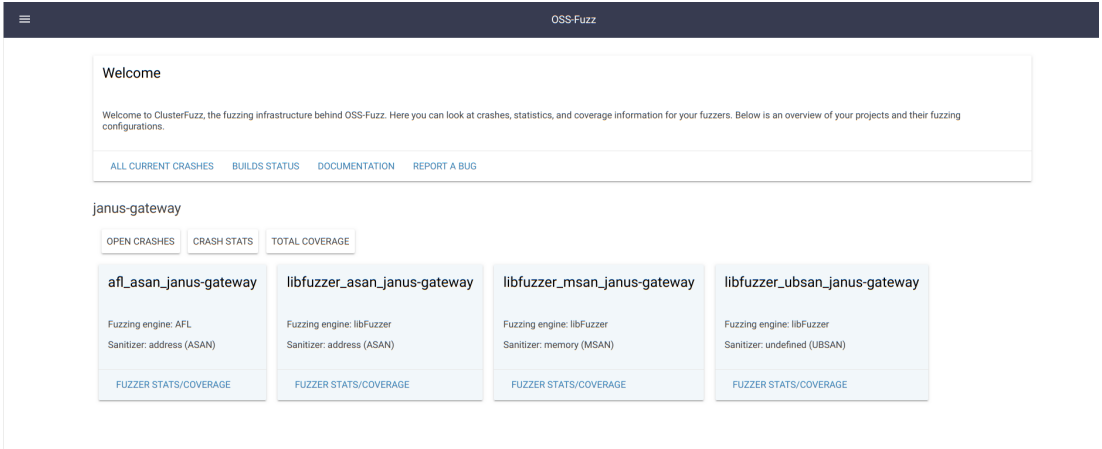
https://github.com/RTC-Cartel/webrtc-fuzzer-corpora (thx, Iñaki!)

Scalable distributed fuzzing via OSS-Fuzz



https://github.com/google/oss-fuzz/pull/2241 (Janus is in, yay!)

Scalable distributed fuzzing via OSS-Fuzz



https://github.com/google/oss-fuzz/pull/2241 (Janus is in, yay!)

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?)
  - Fuzzing signalling might be nice, but so many transports!
  - What about plugins and their custom interactions?
- Definitely expand the corpora
  - The shared RTC-Cartel repo should help with that
  - Let's see if what crashed you crashed us too, and viceversa!
- libFuzzer is not the only option here
  - Henning and Sandro introduced AFL, Radamsa, Gasoline and others last year
  - 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?)
  - Fuzzing signalling might be nice, but so many transports!
  - What about plugins and their custom interactions?
- Definitely expand the corpora
  - The shared RTC-Cartel repo should help with that
  - Let's see if what crashed you crashed us too, and viceversa!
- libFuzzer is not the only option here
  - Henning and Sandro introduced AFL, Radamsa, Gasoline and others last year
  - 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?)
  - Fuzzing signalling might be nice, but so many transports!
  - What about plugins and their custom interactions?
- Definitely expand the corpora
  - The shared RTC-Cartel repo should help with that
  - Let's see if what crashed you crashed us too, and viceversa!
- libFuzzer is not the only option here
  - Henning and Sandro introduced AFL, Radamsa, Gasoline and others last year
  - KITE and its "weaponised" browsers can be very helpful as an orthogonal testing tool

Thanks! Questions? Comments?

Get in touch!

- https://twitter.com/elminiero
- https://twitter.com/meetecho
- http://www.meetecho.com

See you soon in Napoli!

September 23-25, 2019, Napoli — https://januscon.it

See you soon in Napoli!

KAMAILIO WORLD
CONFERENCE & EXHIBITION

September 23-25, 2019, Napoli — https://januscon.it