


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

Lorenzo Miniero

 [@elminiero](https://twitter.com/elminiero)

Kamailio World
May 7th 2019, 



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!

Fuzzing the Janus WebRTC Server

And why you should fuzz too

Alessandro Toppi
Software Engineer @ Meetecho
<atoppi@meetecho.com>



FOSDEM



FOSDEM¹⁹





Hot topic in Kamailio talks already!



Sample output from afl

- process timing		overall results	
run time	52 days, 6 hrs, 6 min, 50 sec	cycles done	1
lost new path	0 days, 0 hrs, 0 min, 20 sec	total paths	10.5k
lost only crash	48 days, 16 hrs, 53 min, 55 sec	uniq crashes	7
lost only hang	11 days, 9 hrs, 1 min, 10 sec	uniq hangs	500+
- cycle progress			
new processing	10.5k (50.79%)	emp. coverage	12.99% / 38.35%
paths timed out	1 (0.01%)	count coverage	2.40 bits/tuple
- stage progress			
new trying	bitflip 4/1	findings in depth	
stage exec	498/2082 (71.18%)	favored paths	2688 (10.13%)
total execs	114N	new edges on	3217 (11.37%)
exec speed	47.86/sec (slow)	total crashes	30 (7 unique)
- fuzzing strategy yields			
bit flips	252/3.18M, 62/3.18M, 66/3.09M	path quantity	
byte flips	6/30M, 0/20M, 1/25M	levels	22
arithmetic	396/15.9M, 0/852M, 1/39.4M	pending	1880
known ints	391/1.04M, 21/0.43M, 112/13.0M	pending	0
dictionary	66/21.5M, 100/26.3M, 50/15.0M	new finds	1550
heaps	372/1.03M, 0/0	imported	n/a
urls	1.12M/30M, 25.7M	stability	98.40%
[cpu000: 57%]			

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



Kamailio lovers know fuzzing already



KAMAILIO WORLD
CONFERENCE & EXHIBITION

pascom

ev

asipto

How do you use AFL to fuzz RTC systems?

- AFL is great when fuzzing tools that take file input
 - e.g. ffmpeg or tcpdump
- AFL is not so great when it comes to fuzzing anything that doesn't take file input (e.g. servers)
- Major hurdle is wiring the target code so that it can be fuzzed with AFL
- Example 1: Asterisk: due to its modular system, we had problems testing specific modules; we ended up copying whole code to be able to load the modules
- Example 2: Kamailio: easier to wire it for fuzzing, except that building it with the compile-time instrumentation for AFL was painful

<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://webrtcchacks.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://webrtcchacks.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://webrtcchacks.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://webrtcchacks.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>



- 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.



- 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 (libsrtplib)
 - 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 (libsrtplib)
 - SCTP (usrsrcplib)
- 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 (libsrtplib)
 - 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)

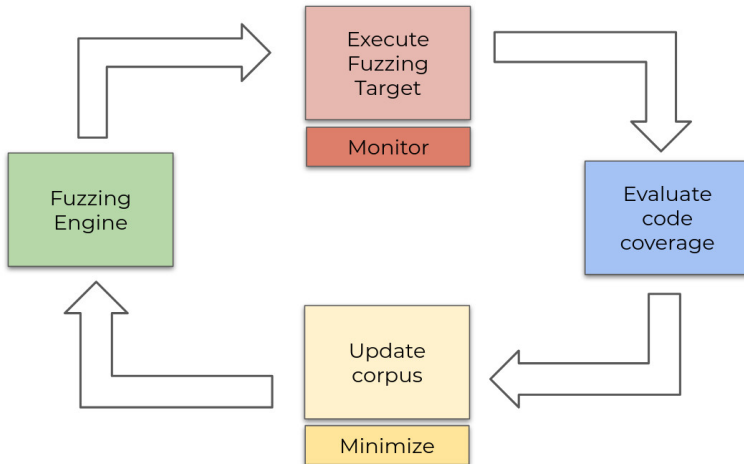


A quick intro to libFuzzer

- 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
```




- 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
```



- 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/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>



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

```
// 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 r
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!



The screenshot shows the GitHub interface for the repository `RTC-Cartel/webrtc-fuzzer-corpora`. At the top, there's a navigation bar with links for Pull requests, Issues, Marketplace, and Explore. Below this, the repository name is displayed with statistics: 3 watchers, 2 stars, and 1 fork. A tab bar shows 'Code' as the active tab, with other options like Issues (1), Pull requests (0), Projects (0), Wiki, Insights, and Settings. The main content area is titled 'libFuzzer corpus files for WebRTC' and includes a 'Manage topics' link and an 'Edit' button. A summary bar indicates 8 commits, 1 branch, 0 releases, and 2 contributors. Below this, there's a section for the 'master' branch with buttons for 'New pull request', 'Create new file', 'Upload files', 'Find File', and 'Clone or download'. A list of recent commits is shown, including one by 'atoppl and ibc' adding Janus RTP crash files. The file tree view at the bottom lists directories like 'corpora' and 'reports', and files like 'README.md' and 'add_sha1.sh'.

RTC-Cartel / [webrtc-fuzzer-corpora](#) Watch 3 Star 2 Fork 1

Code Issues 1 Pull requests 0 Projects 0 Wiki Insights Settings

libFuzzer corpus files for WebRTC Edit

Manage topics

8 commits 1 branch 0 releases 2 contributors

Branch: master New pull request Create new file Upload files Find File Clone or download

atoppl and ibc Add Janus RTP crash files. (#5) Latest commit 2e4c5a4 on 7 Mar

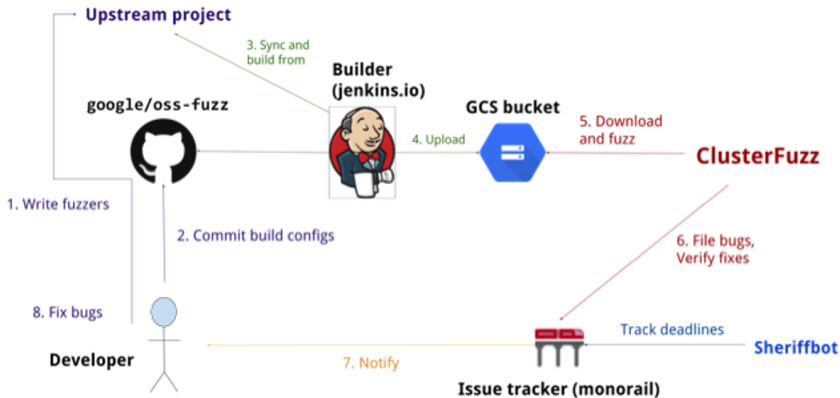
corpora	Add Janus corpus and crash files for RTCP	3 months ago
reports	Add Janus RTP crash files. (#5)	a month ago
README.md	Cosmetic	3 months ago
add_sha1.sh	Helper script to append sha1 hash to filename and detect duplicates (#4)	3 months ago

README.md

<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



 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.

[ALL CURRENT CRASHES](#) [BUILDS STATUS](#) [DOCUMENTATION](#) [REPORT A BUG](#)

janus-gateway

OPEN CRASHES

CRASH STATS

TOTAL COVERAGE

afl_asan_janus-gateway Fuzzing engine: AFL Sanitizer: address (ASAN) FUZZER STATS/COVERAGE	libfuzzer_asan_janus-gateway Fuzzing engine: libFuzzer Sanitizer: address (ASAN) FUZZER STATS/COVERAGE	libfuzzer_msan_janus-gateway Fuzzing engine: libFuzzer Sanitizer: memory (MSAN) FUZZER STATS/COVERAGE	libfuzzer_ubsan_janus-gateway Fuzzing engine: libFuzzer Sanitizer: undefined (UBSAN) FUZZER STATS/COVERAGE
--	--	---	--

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



A detailed tutorial on how to setup all this!



The screenshot shows a Google Chrome browser window with the title "How Janus Battled libFuzzer and Won (Alessandro Toppi) - webrtcHacks - Google Chrome". The address bar shows the URL "https://webrtcHacks.com/fuzzing-janus/". The page has a dark header with the text "webrtcH4cKS:~\$" and a green square. Below the header is a navigation bar with links: Home, About, Subscribe, Contact, and cogint.ai - AI in RTC. The main content area shows a post by Alessandro Toppi on March 6, 2019, titled "webrtcH4cKS: ~ How Janus Battled libFuzzer and Won (Alessandro Toppi)". The post includes a "Guide" link, a "Leave a Comment" link, and tags: fuzzing, janus, libfuzzer, OSS-Fuzz, wireshark. The text of the post discusses the work initiated by Google Project Zero and the Meetecho team. On the right side, there is a "SEARCH" section with a search bar, a "NEW POST" section, and a "NOTIFICATIONS" section with input fields for Email Address, First Name, and Last Name.

<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!

Januscon

NAPOLI
23-25/9/19

Organized by:

Sponsored by:

The First Edition
of a New Event
on all things Janus!!!

www.januscon.it
FOR MORE INFO, CFP AND
SPONSORSHIP OPPORTUNITIES:
Januscon@meetecho.com

Voip on the beach

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



See you soon in Napoli!



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