ENABLE SECURITY | Confidential Document

OpenSIPS security audit

Technical Report (minimized version)

Prepared by: Sandro Gauci, Senior Consultant and Director, Enable Security GmbH

Prepared for: Bogdan-Andrei Iancu, Founder and Developer, OpenSIPS Project

Date: 30 Mar 2022

Contents

1. Changelog

- 2021-11-03: status report produced
- 2022-02-24: status report updated
  - 3 more findings
  - additional methodology + coverage
- 2022-03-30: minimized report created

2. Introduction

The Penetration Test was performed by Sandro Gauci and Alfred Farrugia of Enable Security on the OpenSIPS server. In terms of methodology, a whitebox approach was taken especially since the source code was available due to the open source nature of the project. This gave the testers the advantage of being able to analyse and instrument the code for vulnerability discovery. Additionally, as a result, the testers had further insight into the vulnerabilities that were discovered and the solutions that were consequently applied. The tests were done in lab environment on various test environments to validate the findings.

The following is a report on the findings and fixes that can be applied to limit exposure to attack.

2.1 Purpose and Limitations

The purpose of this security audit was to find security vulnerabilities within the OpenSIPS server so that they can be addressed before adversaries abuse these loopholes.

2.2 Scope

- OpenSIPS parser [1] including:
  - **parser**
  - **parser/digest**
  - **parser/contact**
  - **parser/sdp**
- OpenSIPS auth module [2]
- OpenSIPS tm module [3]
- OpenSIPS dialog module [4]
- The following modules:
  - UDP [5]
  - TCP [6]
  - TLS [7]

---

[1] https://github.com/OpenSIPS/opensips/tree/master/parser

[2] https://github.com/OpenSIPS/opensips/tree/master/modules/auth

[3] https://github.com/OpenSIPS/opensips/tree/master/modules/tm

[4] https://github.com/OpenSIPS/opensips/tree/master/modules/dialog

[5] https://opensips.org/html/docs/modules/devel/proto_udp.html

[6] https://opensips.org/html/docs/modules/devel/proto_tcp.html

[7] https://opensips.org/html/docs/modules/devel/proto_tls.html

    - WS [1]
- The OpenSIPS Management Interface
- Focus on exposed internal IDs
- Topology hiding

[1] https://opensips.org/html/docs/modules/devel/proto_ws.html

3. Findings and recommendations

The following sections list each security-relevant finding identified during the security audit.

3.1 Segmentation fault due to invalid `Content-Length` header (CVSS: 8.6)

- Affects: core
- CVSS v3.1
  - CVSS Base Score: 7.5
  - Impact Subscore: 3.6
  - Exploitability Subscore: 3.9
  - CVSS Temporal Score: 7.0
  - CVSS Environmental Score: 8.6
  - Modified Impact Subscore: 5.4
  - Overall CVSS Score: 8.6
  - Vector: link [1]

3.1.1 Description

A malformed SIP message containing a large **`Content-Length`** value and a specially crafted Request-URI causes a segmentation fault in OpenSIPS. This issue occurs when a large amount of shared memory using the **`-m`** flag was allocated to OpenSIPS, such as 10Gb of RAM. On the tester's system this issue occurred when shared memory was set to 2362 or higher.

3.1.2 Impact

This issue causes OpenSIPS to shutdown due to a segmentation fault. No authentication is required to exploit this issue.

[1] https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/E:F/RL:O/RC:C/CR:X/IR:X/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:H&version=3.1

3.1.3 Solutions and recommendations

This issue was fixed in commit 7cab422 [1] which was tested and found to address the issue. The commit messages read as follows:

> core: Fix Content-Length parsing

3.2 Crash when specially crafted REGISTER message is challenged for authentication (CVSS: 8.6)

- Affects: core
- CVSS v3.1
  - CVSS Base Score: 7.5
  - Impact Subscore: 3.6
  - Exploitability Subscore: 3.9
  - CVSS Temporal Score: 7.0
  - CVSS Environmental Score: 8.6
  - Modified Impact Subscore: 5.4
  - Overall CVSS Score: 8.6
  - Vector: link [2]

3.2.1 Description

A specially crafted REGISTER message with a malformed authorization header may cause a crash.

3.2.2 Impact

This vulnerability will cause OpenSIPS to crash. It affects configurations that require authentication. To reproduce this issue, the credentials do not need to be valid.

It is highly unlikely that exploitation of this issue may lead to anything other than Denial of Service.

---

[1] https://github.com/OpenSIPS/opensips/commit/7cab422e2fc648f910abba34f3f0dbb3ae171ff5

[2] https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/E:F/RL:O/RC:C/CR:X/IR:X/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:H&version=3.1

3.2.3 Solutions and recommendations

This issue was fixed in commit 0fadc0a [1]. The commit message reads as follows:

> **Fix crash with REGISTER + incomplete Authorization header**
>
> Avoid re-using anonymous structures outside of the block scope they were declared in. The compiler allows such broken code, yet it is also quick to re-use/re-claim that memory quickly after exiting the block, leading to stack corruption later down the road, when the "now re-used struct" is read.

3.3 Buffer over-read in function `delete_sdp_line` leads to DoS or undefined behaviour (CVSS: 8.6)

- Affects: SIPMSGOPS module
- CVSS v3.1
  - CVSS Base Score: 7.5
  - Impact Subscore: 3.6
  - Exploitability Subscore: 3.9
  - CVSS Temporal Score: 7.0
  - CVSS Environmental Score: 8.6
  - Modified Impact Subscore: 5.4
  - Overall CVSS Score: 8.6
  - Vector: link [2]

3.3.1 Description

OpenSIPS crashes when a malformed SDP body is received and is processed by the **`delete_sdp_line`** function in the **`sipmsgops`** module.

[1] https://github.com/OpenSIPS/opensips/commit/0fadc0a6cb130d40fba6cf36bb1399d45d0496aa

[2] https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/E:F/RL:O/RC:C/CR:X/IR:X/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:H&version=3.1

3.3.2 Impact

By abusing this vulnerability, an attacker is able to crash the server. It affects configurations containing functions that rely on the affected code, such as the function `codec_delete_except_re`[1].

Due to the sanity check that is performed in the **`del_lump`** function, it is highly unlikely that exploitation of this issue may lead to anything other than Denial of Service.

3.3.3 Solutions and recommendations

This issue was fixed in commits 8f87c7c[2] and c6ab3bb[3] which were tested and found to address the issue. The commit messages read as follows:

> [sipmsgops] fix codec_delete_XX() parsing [sipmsgops] fix codec_delete_XX() parsing (2)

3.4 Buffer over-read in the function `parse_param_name` leads to DoS or undefined behaviour (CVSS: 8.6)

- Affects: SIP digest parser
- CVSS v3.1
  - CVSS Base Score: 7.5
  - Impact Subscore: 3.6
  - Exploitability Subscore: 3.9
  - CVSS Temporal Score: 7.0
  - CVSS Environmental Score: 8.6
  - Modified Impact Subscore: 5.4
  - Overall CVSS Score: 8.6
  - Vector: link[4]

[1] https://opensips.org/html/docs/modules/3.2.x/sipmsgops.html#func_codec_delete_except_re

[2] https://github.com/OpenSIPS/opensips/commit/8f87c7c03da55f9c79bd92e67fa2c94b2a7ce5cf

[3] https://github.com/OpenSIPS/opensips/commit/c6ab3bb406c447e30c7d33a1a8970048b4612100

[4] https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/E:F/RL:O/RC:C/CR:X/IR:X/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:H&version=3.1

3.4.1 Description

A specially crafted **`Authorization`** header causes OpenSIPS to crash or behave in an unexpected way due to a bug in the function **`parse_param_name`**.

3.4.2 Impact

This issue may cause erratic program behaviour or a server crash. It affects configurations containing functions that make use of the affected code, such as the function **`www_authorize`**[1].

3.4.3 Solutions and recommendations

This issue was fixed in commit dd9141b[2] which was tested and found to address the issue. The commit message reads as follows:

> **parse_param_name(): Improve param parsing macros**

3.5 Buffer over-read in the function `extract_field` leads to DoS or undefined behaviour (CVSS: 8.6)

- Affects: SDP parser
- CVSS v3.1
  - CVSS Base Score: 7.5
  - Impact Subscore: 3.6
  - Exploitability Subscore: 3.9
  - CVSS Temporal Score: 7.0
  - CVSS Environmental Score: 8.6
  - Modified Impact Subscore: 5.4
  - Overall CVSS Score: 8.6
  - Vector: link[3]

---

[1] https://opensips.org/docs/modules/3.2.x/auth_db#func_www_authorize

[2] https://github.com/OpenSIPS/opensips/commit/dd9141b6f67d7df4072f3430f628d4b73df5e102

[3] https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/E:F/RL:O/RC:C/CR:X/IR:X/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:H&version=3.1

3.5.1 Description

Parsing malformed SDP content using the **`parse_sdp`** function leads to a crash due to a buffer over-read bug triggered by the function **`extract_field`**.

3.5.2 Impact

This issue may cause erratic program behaviour or a server crash. It affects configurations containing functions that make use of the affected code, such as the function **`is_audio_on_hold`**[1].

3.5.3 Solutions and recommendations

This issue was fixed in commit 2617c97[2]. The commit message reads as follows:

> **Fix crash in parse_sdp when a= is empty**
>
> When a bogus SDP was provided, with an empty **a=** line, there was no check for the length to be compared, resulting in a bad memory access, hence a crash.

3.6 Buffer over-read in function `extract_rtpmap` leads to DoS or undefined behaviour (CVSS: 8.6)

- Affects: SDP parser
- CVSS v3.1
  - CVSS Base Score: 7.5
  - Impact Subscore: 3.6
  - Exploitability Subscore: 3.9
  - CVSS Temporal Score: 7.0
  - CVSS Environmental Score: 8.6
  - Modified Impact Subscore: 5.4
  - Overall CVSS Score: 8.6
  - Vector: link[3]

---

[1] https://opensips.org/html/docs/modules/3.2.x/sipmsgops.html#func_is_audio_on_hold

[2] https://github.com/OpenSIPS/opensips/commit/2617c97207b1fe2afead9f887f7a5df4da3b7d55

[3] https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/E:F/RL:O/RC:C/CR:X/IR:X/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:H&version=3.1

3.6.1 Description

When the patch for the **`extract_field`** issue is applied, parsing malformed SDP content using the **`parse_sdp`** function still leads to a crash. This occurs due to a buffer over-read bug triggered by the function **`extract_rtpmap`**.

3.6.2 Impact

This issue may cause erratic program behaviour or a server crash. It affects configurations containing functions that make use of the affected code, such as the function **`is_audio_on_hold`**[1].

3.6.3 Solutions and recommendations

This issue was fixed in commit aebac09[2]. The commit message reads as follows:

> **Fix crash in parse_sdp for fmtp, rtpmap and hold**
>
> When invalid strings would have been passed, the remaining value would have resulted in an invalid memory access.

3.7 Buffer over-read in the function `extract_fmtp` leads to DoS or undefined behaviour (CVSS: 8.6)

- Affects: SDP parser
- CVSS v3.1
  - CVSS Base Score: 7.5
  - Impact Subscore: 3.6
  - Exploitability Subscore: 3.9
  - CVSS Temporal Score: 7.0
  - CVSS Environmental Score: 8.6
  - Modified Impact Subscore: 5.4
  - Overall CVSS Score: 8.6
  - Vector: link[3]

---

[1] https://opensips.org/html/docs/modules/3.2.x/sipmsgops.html#func_is_audio_on_hold

[2] https://github.com/OpenSIPS/opensips/commit/aebac095b94607c86c6fe0278bae6e96bf53862e

[3] https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/E:F/RL:O/RC:C/CR:X/IR:X/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:H&version=3.1

3.7.1 Description

When the patch for the **`extract_field`** issue is applied, parsing malformed SDP content using the **`parse_sdp`** function still leads to a crash. This occurs due to a buffer over-read bug triggered by the function **`extract_fmtp`**.

3.7.2 Impact

This issue may cause erratic program behaviour or a server crash. It affects configurations containing functions that make use of the affected code, such as the function **`is_audio_on_hold`**[1].

3.7.3 Solutions and recommendations

This issue was fixed in commit aebac09[2]. The commit message reads as follows:

> **Fix crash in parse_sdp for fmtp, rtpmap and hold**
>
> When invalid strings would have been passed, the remaining value would have resulted in an invalid memory access.

3.8 Off-by-one error in the function `append_hf` leads to a crash (CVSS: 8.6)

- Affects: SIP message parser
- CVSS v3.1
  - CVSS Base Score: 7.5
  - Impact Subscore: 3.6
  - Exploitability Subscore: 3.9
  - CVSS Temporal Score: 7.0
  - CVSS Environmental Score: 8.6
  - Modified Impact Subscore: 5.4
  - Overall CVSS Score: 8.6
  - Vector: link[3]

---

[1] https://opensips.org/html/docs/modules/3.2.x/sipmsgops.html#func_is_audio_on_hold

[2] https://github.com/OpenSIPS/opensips/commit/aebac095b94607c86c6fe0278bae6e96bf53862e

[3] https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/E:F/RL:O/RC:C/CR:X/IR:X/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:H&version=3.1

3.8.1 Description

When the function **append_hf** handles a SIP message with a malformed **To** header, a call to the function **abort()** is performed resulting in a crash.

3.8.2 Impact

An attacker abusing this vulnerability will crash OpenSIPS leading to Denial of Service. It affects configurations containing functions that make use of the affected code, such as the function **append_hf** [1].

3.8.3 Solutions and recommendations

This issue was fixed in commit eab8ff6 [2]. The commit message reads as follows:

> [sipmsgops] fix parse_to_param() parsing

3.9 Segmentation fault in the function build_res_buf_from_sip_req might lead to DoS (CVSS: 6.2)

- Affects: core
- CVSS v3.1
  - CVSS Base Score: 5.9
  - Impact Subscore: 3.6
  - Exploitability Subscore: 2.2
  - CVSS Temporal Score: 4.7
  - CVSS Environmental Score: 6.2
  - Modified Impact Subscore: 5.4
  - Overall CVSS Score: 6.2
  - Vector: link [3]

[1] https://opensips.org/html/docs/modules/3.2.x/sipmsgops.html#func_append_hf

[2] https://github.com/OpenSIPS/opensips/commit/eab8ff654bebaa04dda7bed78266844b385f928b

[3] https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:H/E:U/RL:O/RC:U/CR:X/IR:X/AR:H/MAV:X/MAC:X/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:H&version=3.1

3.9.1 Description

A potential issue was found in **`msg_translator.c:2628`** which might lead to a server crash. This issue was found while fuzzing the function **`build_res_buf_from_sip_req`** but could not be reproduced against a running instance of OpenSIPS.

3.9.2 Impact

This issue could not be exploited against a running instance of OpenSIPS since no public function was found to make use of this vulnerable code. Even in the case of exploitation through unknown vectors, it is highly unlikely that this issue would lead to anything other than Denial of Service.

3.9.3 Solutions and recommendations

This issue was fixed in commit 9cf3dd3 [1]. The commit message reads as follows:

> [core] build_res_buf_from_sip_req(): fix hdr correlation

3.10 Segmentation fault when calling the function `calc_tag_suffix` leads to DoS (CVSS: 8.6)

- Affects: core
- CVSS v3.1
  - CVSS Base Score: 7.5
  - Impact Subscore: 3.6
  - Exploitability Subscore: 3.9
  - CVSS Temporal Score: 7.0
  - CVSS Environmental Score: 8.6
  - Modified Impact Subscore: 5.4
  - Overall CVSS Score: 8.6
  - Vector: link [2]

[1] https://github.com/OpenSIPS/opensips/commit/9cf3dd3398719dd91207495f76d7726701c5145c

[2] https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/E:F/RL:O/RC:C/CR:X/IR:X/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:H&version=3.1

3.10.1 Description

Sending a malformed **`Via`** header to OpenSIPS triggers a segmentation fault when the function **`calc_tag_suffix`** is called.

3.10.2 Impact

Abuse of this vulnerability leads to Denial of Service due to a crash. Since the uninitialized string points to memory location **`0x0`**, no further exploitation appears to be possible. No special network privileges are required to perform this attack, as long as the OpenSIPS configuration makes use of functions such as **`sl_send_reply`** or **`sl_gen_totag`** that trigger the vulnerable code.

3.10.3 Solutions and recommendations

This issue was fixed in commit ab611f7 [1]. The commit message reads as follows:

> [core] fix parse_via() parsing

3.11 Crash in the function `t_reply_matching` may lead to DoS (Info)

- Affects: TM module

3.11.1 Description

A crash was discovered while performing coverage guided fuzzing against the function **`t_reply_matching`**. It is highly unlikely that this issue is exploited since the payload that crashes the function lacks the **`CSeq`** header, which is a required field when matching replies.

3.11.2 Impact

This issue is informational since SIP messages lacking the **`CSeq`** header are not likely to be passed to this function.

[1] https://github.com/OpenSIPS/opensips/commit/ab611f74f69d9c42be5401c40d56ea06a58f5dd7

3.11.3 Solutions and recommendations

If the function **`t_reply_matching`** is called without proper checks for the **CSeq** header then the code would cause OpenSIPS to crash. It is recommended that a test for the existence of the **CSeq** header is performed before calling **`get_cseq()`**.

3.12 Heap-buffer-overflow in function `parse_hname2` leads to AddressSanitizer false positives (Info)

- Affects: SIP message parser

3.12.1 Description

The AddressSanitizer reported a heap-buffer-overflow while fuzzing **`parse_msg`**. However, this issue could not be abused against a running OpenSIPS instance.

Analysis showed that the AddressSanitizer report was due to a bug where the code reads beyond a few bytes, up to a maximum of 4 bytes, into a 32 bit dword. Since it does not read beyond the 32 bit dword, this issue does not appear to be exploitable and does not actually result in any memory violations. It does, however, trigger the AddressSanitizer's strict heap-buffer-overflow detection.

3.12.2 Impact

Although issue is not exploitable, it will generate AddressSanitizer false positives. Such results weaken the AddressSanitizer's ability to find further bugs.

3.12.3 Solutions and recommendations

This issue was fixed in commit 209218c [1]. The commit message reads as follows:

> Since fuzzers typically use the system allocator in order to run ASan checks (i.e. -DPKG_MALLOC is not enabled), they will often run into false-positive crashes in the parse_hname2() function due to various read overflows which are harmless in production, thanks to the pre-allocated nature of the PKG memory chunk.

> This patch adds the HAVE() parser macro (tied to -DFUZZ_BUILD), which will be optimized (removed) in the public build (0 changes), while protecting against any read overflow when

[1] https://github.com/OpenSIPS/opensips/commit/209218c91e64d9fb5e192ccaf5466c4140ba226d

building with -DFUZZ_BUILD.

3.13 Segmentation fault in the function `rewrite_ruri` leads to DoS (CVSS: 8.6)

- Affects: SIP message parser
- CVSS v3.1
  - CVSS Base Score: 7.5
  - Impact Subscore: 3.6
  - Exploitability Subscore: 3.9
  - CVSS Temporal Score: 7.0
  - CVSS Environmental Score: 8.6
  - Modified Impact Subscore: 5.4
  - Overall CVSS Score: 8.6
  - Vector: link [1]

3.13.1 Description

When a specially crafted SIP message is processed by the function **`rewrite_ruri`**, a crash occurs due to a segmentation fault.

3.13.2 Impact

This issue causes the server to crash. It affects configurations containing functions that make use of the affected code, such as the function **`setport`** [2].

3.13.3 Solutions and recommendations

This issue was fixed in commit b2dffe4 [3]. The commit message reads as follows:

> [core] fix parse_uri() parsing

[1] https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/E:F/RL:O/RC:C/CR:X/IR:X/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:H&version=3.1

[2] https://www.opensips.org/Documentation/Script-CoreFunctions-3-2#setport

[3] https://github.com/OpenSIPS/opensips/commit/b2dffe4b5cd81182c9c8eabb6c96aac96c7acfe3

3.14 Memory leak in `parse_mi_request` might lead to Denial of Service (CVSS: 7.1)

- Affects: MI module
- CVSS v3.1
  - CVSS Base Score: 7.5
  - Impact Subscore: 3.6
  - Exploitability Subscore: 3.9
  - CVSS Temporal Score: 7.0
  - CVSS Environmental Score: 7.1
  - Modified Impact Subscore: 5.4
  - Overall CVSS Score: 7.1
  - Vector: link [1]

3.14.1 Description

A memory leak was detected in the function **`parse_mi_request`** while performing coverage-guided fuzzing.

3.14.2 Impact

To abuse this memory leak, attackers need to reach the managment interface which typically, should be only exposed on trusted interfaces. In cases where the MI is exposed to the internet without authentication, abuse of this issue will lead to memory exhaustion which may affect the underlying system's availability. No authentication is typically required to reproduce this issue.

3.14.3 How to reproduce the issue

1. Start an OpenSIPS server with the following configuration:

```
debug_mode=no

log_level=3
xlog_level=3
log_stderror=no
log_facility=LOG_LOCAL0

udp_workers=1

socket=udp:PRIVATE_IP:5060
```

[1] https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/E:F/RL:O/RC:C/CR:X/IR:X/AR:H/MAV:N/MAC:H/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:H&version=3.1

```
mpath="/usr/local//lib64/opensips/modules/"

loadmodule "proto_udp.so"

loadmodule "mi_fifo.so"

modparam("mi_fifo", "fifo_name", "/var/run/opensips/opensips_fifo")
modparam("mi_fifo", "fifo_mode", 0666)


route {
    exit;
}
```

2. Save the below Python script to **`mi-memory-leak.py`**

```
while True:
    with open('/var/run/opensips/opensips_fifo', 'at') as fout:
        fout.write(':t: {"jsonrpc": "2.0","method": "log_le')
```

3. Run the above script

```
python mi-memory-leak.py
```

4. Run **`top`** and observe the memory usage of OpenSIPS slowly increase –>

3.14.4 Solutions and recommendations

This issue was fixed in commit 4175687[1]. The commit message reads as follows:

> cJSON: fix memory leak on object parsing error

3.15 Buffer over-read in function `stream_process` leads to DoS (CVSS: 8.6)

- Affects: SIPMSGOPS module
- CVSS v3.1
  - CVSS Base Score: 7.5
  - Impact Subscore: 3.6
  - Exploitability Subscore: 3.9
  - CVSS Temporal Score: 7.0
  - CVSS Environmental Score: 8.6
  - Modified Impact Subscore: 5.4
  - Overall CVSS Score: 8.6
  - Vector: link[2]

[1] https://github.com/OpenSIPS/opensips/commit/417568707520af25ec5c5dd91da18e6db3649dcb

[2] https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?vector=AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H/E:F/RL:O/RC:C/CR:X/IR:X/AR:H/MAV:N/MAC:L/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:

3.15.1 Description

OpenSIPS crashes when a malformed SDP body is sent multiple times to an OpenSIPS configuration that makes use of the **`stream_process`** function.

3.15.2 Impact

By abusing this vulnerability, an attacker is able to crash the server. It affects configurations containing functions that reply on the affected code, such as the function **`codec_delete_except_re`**[1].

3.15.3 Solutions and recommendations

This issue was fixed in commit 1d60e74[2]. The commit message reads as follows:

> [sipmsgops] fix codec_delete_XX() parsing

H&version=3.1

[1] https://opensips.org/html/docs/modules/3.2.x/sipmsgops.html#func_codec_delete_except_re

[2] https://github.com/OpenSIPS/opensips/commit/1d60e7422aff69ca083fac3f907d7183c2e7f58e

4. Conclusion

The OpenSIPS security audit led to 15 security relevant findings many of which cause denial of service due to memory related bugs. Although these issues affected critical areas, we did not identify ways to escalate to remote code execution, memory disclosure or similar exploitation. On the other hand, the severity of these vulnerabilities is high or critical due to the importance of availability in real-time communications systems.

As part of performing this exercise, we built a number of tools that allowed us to perform fuzzing exercises that were never, to our knowledge, done before on OpenSIPS. Some of this work involved slightly modifying the source code so that it can be used in fuzzing exercises. This was especially useful when it came to coverage-guided fuzzing of certain modules. Additionally, a number of tests were provided for fuzzing specific functions within the parts of OpenSIPS that were in scope. We hope that these tests will prove useful in finding future security issues that may be introduced in OpenSIPS and inspire future fuzzing exercises.

We hope that thanks to this security audit, OpenSIPS can be modified to become more *fuzzable*, which includes modifications that can be done to the project so that all modules can be easily included in automated fuzz tests such as those done on OSS-Fuzz. This will help make the project easier to test for security issues on the whole, thus leading to a more robust and secure project.

Enable Security would like to thank Bogdan-Andrei Iancu and the OpenSIPS Project technical team for their excellent project coordination, support and assistance, both before and during this assignment.

5. Appendix

5.1 Methodology

The version that was targeted was 3.2.2.

```
commit f380dc59eb79ec2153d5b0d6e8374c0877fca525 (tag: 3.2.2)
Author: Liviu Chircu <liviu@opensips.org>
Date:   Thu Aug 19 17:20:36 2021 +0300

    Update ChangeLog for 3.2.2
```

Our methodology included the following techniques:

- preparing docker images to run the full OpenSIPS server as a target and for debugging purposes
- usage of blackbox fuzzing techniques where malformed SIP messages are sent to the OpenSIPS server
- usage of coverage-guided fuzzing using libfuzzer where specific functions were fuzzed in memory
- same thing but done using AFL instead of libfuzzer
- tests with AFLNET which tries to combine best of both worlds (greybox approach)
- other vulnerability discovery methods were also taken such as manual code review

5.1.1 Setup of OpenSIPS as a target for fuzz testing, debugging and other general tests

Multiple Docker images were created to facilitate the execution and debugging of different server configurations. The following types of builds were created:

- vanilla: a vanilla build (**`make all`**)
- asan: compiled with AddressSanitizer (**`CFLAGS=-fsanitize=address LDFLAGS=-fsanitize=address MOD_CFLAGS=-fsanitize=address MOD_LDFLAGS=-fsanitize=address`**)
- no-optimization: built with compiler optimization turned off by adding **`DEFS+= -DCC_O0`** to the Makefile template **`Makefile.conf.template`**

These docker images were also used to analyse core dumps, verify bugs and measure their severity. Debugging was done using GDB.

Different configurations were made available to the OpenSIPS instance via mounted volumes, making it easier to switch between different configurations. Core dumps were extracted from the Docker containers by a monitoring script that moves the files from the container to the host machine.

The following is an example of how a test server was started using the vanilla build and a configuration that

made use of the **add_local_rport** [1] function.

```
server_type="vanilla" ./run.sh corefunctions/add_local_rport
```

To debug the OpenSIPS server, a script was used to attach **gdb** to a specific process (typically the UDP listener) inside the OpenSIPS docker container. This provided to be useful during crash analysis and source code debugging.

```
gdb -p `opensips-cli -x mi ps |
    jq '."Processes" | .[] | select(.Type | contains("SIP receiver udp")) | .PID'`
```

The deliverables for this setup are available at
https://github.com/EnableSecurity/opensips-security-audit-2021/tree/main/docker.

5.1.2 Blackbox fuzz testing setup

Blackbox fuzz testing was carried out by using SIPVicious PRO's SIP method fuzzing tool [2] against multiple OpenSIPS instances running different configurations targeting specific functionality. This was achieved by using the Docker Compose environment described in the section 5.1.1.

The first step when testing a specific section of the code was to define a configuration which reached the desired code. The following is an example configuration that tests the **add_body_part** function in the **sipmsgops** module.

```
debug_mode=yes

log_level=3
xlog_level=3
log_stderror=no
log_facility=LOG_LOCAL0

udp_workers=4

socket=udp:PRIVATE_IP:5060

mpath="/usr/local//lib64/opensips/modules/"

loadmodule "proto_udp.so"
loadmodule "sipmsgops.so"

loadmodule "mi_fifo.so"
modparam("mi_fifo", "fifo_name", "/var/run/opensips/opensips_fifo")
modparam("mi_fifo", "fifo_mode", 0666)
```

[1] https://www.opensips.org/Documentation/Script-CoreFunctions-3-2#add_local_rport

[2] https://www.sipvicious.pro/documentation/cui-reference/sip/fuzz/method/

```
route {
   add_body_part("Hello World!", "text/plain");
   exit;
}
```

The server would then be attacked via SIPVicious PRO's SIP fuzz method tool. In the above case, SIPVicious was executed in the following way:

```
sipvicious sip fuzz method udp://TARGET:5060 -m invite
```

TARGET would be replaced by the IP of the running docker container: **`docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' server-docker_opensips_1`**.

This whole process was automated so that multiple tests could be carried out with minimal manual intervention. This method was useful to find several bugs, such as a crash in the **`codec_delete_except_re`** function.

The following is a list of all the functions that were tested:

- **Core functions**:
  - **`add_local_rport`**
  - **`append_branch`**
  - **`force_rport`**
  - **`setdsturi`**
  - **`sethost`**
  - **`sethostport`**
  - **`setport`**
  - **`seturi`**
  - **`setuser`**
  - **`setuserpass`**
  - **`strip_tail`**
- **auth**:
  - **`www_authorize`**
  - **`www_challenge`**
- **sipmsgops**:
  - **`add_body_part`**
  - **`append_hf`**
  - **`append_time`**
  - **`append_urihf`**

- codec_delete_except_re
- codec_delete_re
- codec_delete
- codec_exists_re
- codec_exists
- codec_move_down_re
- codec_move_down
- codec_move_up_re
- codec_move_up
- has_body_part
- has_totag
- insert_hf
- is_audio_on_hold
- is_method
- is_privacy
- is_uri_user_e164
- list_hdr_add_option
- list_hdr_has_option
- list_hdr_remove_option
- remove_body_part
- remove_hf_glob
- remove_hf_re
- remove_hf
- ruri_add_param
- ruri_del_param
- ruri_has_param
- ruri_tel2sip
- sipmsg_validate
- stream_delete
- stream_exists

- sl:
  - reply_error
  - send_reply
- tm:
  - t_add_cancel_reason
  - t_add_hdrs
  - t_anycast_replicate
  - t_check_status
  - t_check_trans
  - t_flush_flags
  - t_inject_branches

        - **t_local_replied**
        - **t_new_request**
        - **t_newtran**
        - **t_on_branch**
        - **t_on_failure**
        - **t_relay**
        - **t_replicate**
        - **t_reply_with_body**
        - **t_reply**
        - **r_wait_for_newbranches**
        - **t_wait_no_more_branches**
        - **t_was_cancelled**
    - **topology hiding**:
        - **topology_hiding**
        - **topology_hiding_match**

Using this method, the following areas of OpenSIPS were tested:

- **parser/**
- **parser/digest/**
- **parser/contact/**
- **parser/sdp**
- AUTH module
- TM module
- DIALOG module
- SIPMSGOPS module
- TOPOLOGY_HIDING module
- MI module

5.1.3 Coverage-guided fuzz testing setup using libfuzzer

Coverage-guided fuzzing using libfuzzer was performed by using a Docker image based on https://github.com/EnableSecurity/fuzzing-images/pkgs/container/fuzzing-images%2Flibfuzzer.

The libfuzzer fuzzing project was structured in the following way:

```
- Dockerfile
- buildimage.sh
- run.sh
- grouprun.sh
- attach.sh
- global-linebyline-report.py
- scripts/
  - prebuild.sh
  - Makefile
  - patches/
```

```
    - ...
- tests/
  - parse_msg/
    - corpus/
    - dict/
    - build.sh
    - main.c
```

These files are part of the deliverables available at https://github.com/EnableSecurity/opensips-security-audit-2021/tree/main/fuzzing/libfuzzer. The following is a summary of what the scripts do:

- **buildimage.sh**: builds the Docker image **opensips-pentest/libfuzzer** that is used for fuzzing
- **run.sh**: runs a specific test
- **grouprun.sh**: runs a group of tests
- **attach.sh**: a helper script that runs an interactive bash in the container running the fuzzer
- **global-linebyline-report.py**: a Python script that aggregates the code coverage report of all the tests into one report

When running tests, the **run.sh** script would be used. The script usage is as follows:

```
./run.sh <test_name> compile
./run.sh <test_name> dryrun
./run.sh <test_name> sample
./run.sh <test_name> min
./run.sh <test_name> run <number_of_seconds> <number_of_rounds> <parallel_amount>
./run.sh <test_name> bash
```

The functions **compile**, **dryrun**, **sample**, **min** and **bash** are mostly used during the building of the tests or while analysing crashes.

- **compile**: compiles the fuzzer's code and exit
- **dryrun**: compiles and run the fuzzing binary against each of the sample corpus
- **sample**: runs the fuzzing binary for a few seconds and then exit
- **min**: minimizes the current corpus
- **bash** builds the fuzzing binary and runs an interactive bash, allowing the tester to manually test the fuzzing binary
- **run**: used to perform the actual fuzzing for a long period of time. The **number_of_seconds** argument determines how long each round takes. The number of rounds to perform is determined by the argument **number_of_rounds**. After each round executes, a minimization process is performed on the corpus. Finally, the **parallel_amount** determines how many parallel fuzzers are run at the same time.

Each test was created in a separate directory under the path **tests/**. Each test contains the following items:

```
- main.c
- build.sh
- corpus/
  - payload1
  - payload2
  - ...
- dict/
  - dict.dict
```

The **main.c** contains the fuzzer's code. **build.sh** contains a set of instructions that patches and builds the final binary. The **corpus** directory contains a set of files with the initial corpus of the test. Finally, a dictionary with potentially interesting payloads are defined in **dict/dict.dict**. For example, when fuzzing the function **parse_msg**, the dictionary contains quaternion values such as **max-**[1] and **forw**[2].

The **build.sh** script would typically perform the following operations:

- Call **/test/prebuild.sh** which will patch the code and perform additional modifications to the code which makes it more fuzzable. More information about this will be discussed later.
- Call **make -s -f /test/Makefile clean** to clean the project
- Call **make -s -f /test/Makefile all -j$(nproc)** to build the fuzzing binary. This Makefile is available in the Docker image and is part of the deliverables.

The following is the boilerplate code for a typical **main.c**:

```
int initialized;

static char buf[BUF_SIZE + 1];
int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size)
{
  if (Size > BUF_SIZE)
  {
    return 0;
  }

  memcpy(buf, Data, Size);
  buf[Size] = 0;

  if (initialized == 0)
  {
    initfuzzer();
    initialized = 1;
  }

  struct sip_msg *req;
  req = (struct sip_msg *)pkg_malloc(sizeof(struct sip_msg));
  if (req == NULL)
  {
    LM_ERR("No more memory\n");
    return 0;
```

[1] https://github.com/OpenSIPS/opensips/blob/master/parser/keys.h#L66

[2] https://github.com/OpenSIPS/opensips/blob/master/parser/keys.h#L67

```
  }
  memset(req, 0, sizeof(struct sip_msg));

  req->buf = buf;
  req->len = Size;
  req->rcv.src_ip.af = AF_INET;
  req->rcv.dst_ip.af = AF_INET;

  if (parse_msg(buf, Size, req) == 0)
  {
    if ((req->via1==0) || (req->via1->error!=PARSE_OK)){
        goto end;
    }

    // call function under test
  }

end:
  free_sip_msg(req);
  pkg_free(req);
  return 0;
}
```

The **initfuzzer()** function performs memory and module initialization, having most of its code from openSIPS' **main.c**. The code is available in https://github.com/EnableSecurity/opensips-security-audit-2021/blob/main/fuzzing/libfuzzer/scripts/patches/fuzzutils.c.

A fuzzer would normally perform the following operations:

1. Perform OpenSIPS initialization once
2. Create a null terminated buffer with the data buffer that is used by the fuzzer
3. Create a **struct sip_msg** and parse the data buffer
4. If the SIP message has been successfully parsed, and contains a Via header (basic check performed by the server code), pass the **sip_msg** structure to the function under test

A common **Makefile** was written to build fuzzer binaries, which converts the relevant source code to object files and then linking everything to build the binary called **fuzzer**.

In order to fuzz test modules such as **tm** and **sl**, and modules with dependencies such as the **dialog** module, slight modifications to the code were required. Although there seems to be an attempt to provide code that compiles the modules statically (such as the preprocessor directive **STATIC_TM**), not all modules were written in this way. The following steps were done to achieve a static build:

1. Changed the module exports name from **exports** to **tm_exports**, **dialog_exports**, etc. This was done by using **sed**

```
sed -i \
    's/struct module_exports exports= {/struct module_exports tm_exports= {/g' \
    modules/tm/tm.c
sed -i 's/exports\.stats/tm_exports.stats/g' modules/tm/tm.c
```

2. Added references to the module exports in **sr_module.c**

```
extern struct module_exports proto_udp_exports;
  extern struct module_exports tm_exports;
  extern struct module_exports dialog_exports;
  extern struct module_exports rr_exports;
```

3. Updated the **register_builtin_modules** function to register the static modules, such as:

```
ret=register_module(&tm_exports,"built-in", 0);
  if (ret<0) return ret;
```

It is suggested that preprocessor directives are added to all modules in order to make the source code more fuzzable.

The functions that are exported via the modules in scope (such as **www_challenge** in the **auth** module) were tested and after each test, the code coverage report was reviewed manually to guarantee that enough code is being covered. Given the size of the code base, a tool was built using Python and weggli [1] that analyzes coverage and inform the testers of potentially untested code which might be susceptible to vulnerabilities. This tool can be found in the deliverables under the **function-usage** directory. The script **weggli-coverage.py** uses the line by line reports generated by the fuzzers and output from weggli to perform these tests. A number of examples of how this script can be used can be found in **find-uncovered-code.sh**.

```
python3 weggli-coverage.py ../libfuzzer-docker/report '{_* $p; $p++;}' `pwd`/.work/opensips
```

This script will first aggregate all line by line coverage reports from the directory **../libfuzzer-docker/report**. Then it will perform a weggli lookup for **{_* $p; $p++;}**, which mean any code that references a pointer and increments its value using **++**. The script would then find lines in the code where this is the case and no fuzzing coverage was done.

Using this method, the following areas of OpenSIPS were tested:

- Core functions (such as **rewrite_ruri**)
- **parser**
- **parser/digest**
- **parser/contact**
- **parser/sdp**
- AUTH module

[1] https://github.com/googleprojectzero/weggli

- DIALOG module
- TM module
- SL module
- MI module
- SIPMSGOPS module
- TOPOLOGY_HIDING module

5.1.4 Coverage-guided fuzz testing setup using AFL

Coverage-guided fuzzing using AFL was performed by using a Docker image based on https://github.com/EnableSecurity/fuzzing-images/pkgs/container/fuzzing-images%2Fafl. The AFL fuzzing project was structured in the following way:

```
- Dockerfile
- buildimage.sh
- run.sh
- attach.sh
- tests/
   - parse_msg/
```

These files are part of the deliverables available at https://github.com/EnableSecurity/opensips-security-audit-2021/tree/main/fuzzing/afl. The following is a summary of what the scripts do:

- **`buildimage.sh`**: builds the Docker image **`opensips-pentest/afl`** that is used for fuzzing
- **`run.sh`**: runs a specific test
- **`attach.sh`**: a helper script that runs an interactive bash in the container running the AFL fuzzer

When running tests, the **`run.sh`** script is used. The script usage is as follows:

```
./run.sh <test_name> compile
./run.sh <test_name> run <number_of_seconds> <parallel_amount>
./run.sh <test_name> bash
```

The functions **`compile`** and **`bash`** are mostly used during the building of the tests or while analysing crashes.

- **`compile`**: compile the source code and exit
- **`bash`**: builds the fuzzer and runs an interactive bash, allowing the tester to manually test the fuzzing binary
- **`run`**: used to perform the actual fuzzing for a long period of time. The **`number_of_seconds`** argument determines how long the fuzzing takes. The **`parallel_amount`** determines how many parallel fuzzers are run at the same time.

Each test was created in a separate directory under the path **`tests/`**. Each test contains the following

items:

```
- main.c
- build.sh
- corpus/
   - payload1
   - payload2
   - ...
- dict/
- patches/
```

The **`main.c`** contains the fuzzer's code as expected by AFL. **`build.sh`** contains a set of instructions that patches and build the binary. The **`corpus`** directory contains a set of files with the initial corpus of the test.

This method only performed fuzzing against the SIP parser since we concentrated our efforts on libfuzzer. However, the tests can be easily ported from libfuzzer to AFL. It is recommended that a common interface is created so that the fuzzer's code can be shared between AFL and libfuzzer.

Using this method, the following areas of OpenSIPS were tested:

- **`parser/`**
- **`parser/digest/`**
- **`parser/contact/`**

5.1.5 Greybox fuzzing setup using AFLNet

Greybox fuzzing was perform using AFLNet [1] based on the Docker image **`ghcr.io/enablesecurity/fuzzing-images/clang12`**. AFLnet tries to bridge the gap between coverage guided fuzzing of source code and black box fuzzing of a running server. Although this technique is useful against OpenSIPS and might discover bugs which might have not been discovered by coverage guided fuzzing, the speed of the fuzzer was painstakingly slow. Given the time constraints of the project, it was decided that this method was not ideal. The setup is part of the deliverables at `https://github.com/EnableSecurity/opensips-security-audit-2021/tree/main/fuzzing/aflnet`.

[1] `https://github.com/aflnet/aflnet`