No way to enable SSH access to your new router?

Stanislav Dashevskyi
Francesco La Spina

hack.lu@2025

About us

Stanislav Dashevskyi

FORESCOUT RESEARCH | VEDERE LABS

Francesco La Spina

Vulnerability Research

- Focus on vulnerabilities against managed and unmanaged devices (IT/IoT/IoMT/OT)
- 200+ vulnerabilities discovered in last 5 years

Threat Reports

- Manual and automatic analysis of malware samples collected via customer telemetry and other sources

Agenda

- Motivation: why rooting a router and where to begin?
- Introducing today’s target: TP-Link ER605 v2
- Reopening closed (back)doors and reversing LuCI scripts
- Demo time!

Why rooting a router?

- Most consumer and business-grade network devices do now allow root access these days

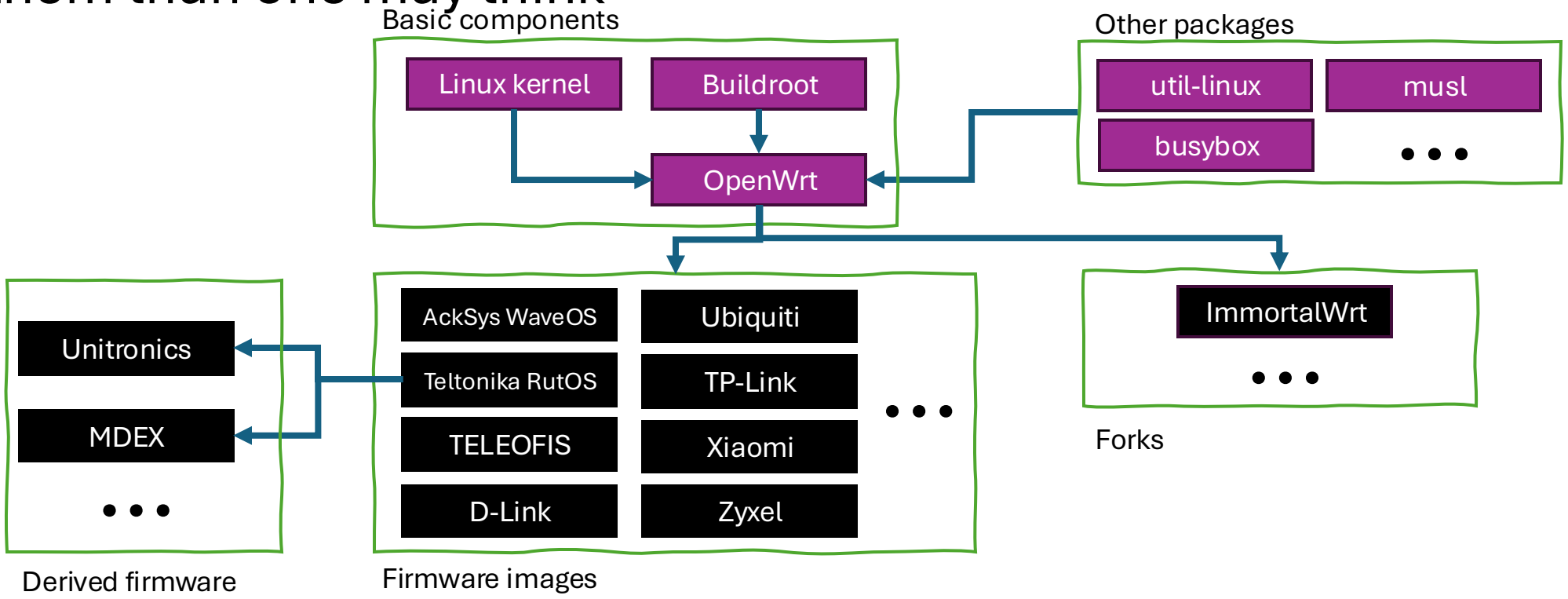
- Facilitate VR
  - Dynamic reverse engineering
  - Decrypting other firmware versions
  - Exploit development

- Fun and profit
  - Better value for your money, get more features
  - Bounty programs
  - Doing crimes

- Yet, network devices run some kind of Linux (mostly)

OpenWRT and its derivatives

- Loosely speaking, there are as many router OSes, as there are vendors…
- …however, there are way more similarities among them than one may think



OpenWRT and its derivatives (continued)

- Many used to support OpenWrt as an alternative, but not anymore. E.g., WiFi 6 routers are not supported (Broadcomm SoC)
  - https://openwrt.org/meta/infobox/broadcom_wifi

The new FCC rules are in effect in the United States from June 2nd 2015 for WiFi devices such as Access Points. They require to have the firmware locked down so End-Users can't operate with non-compliant parameters (channels/frequencies, transmit power, DFS, …). In response, WiFi access point vendors start to lock down firmwares to prevent custom firmwares (such as OpenWRT) to be installed, using code signing, etc. Since the same type of devices are often sold world wide, this change does not only affect routers in the US, but also Europe, and this will also effect wireless communities.

- **Simon Wunderlich "OpenWRT vs FCC – forced firmware lockdown?", Wireless Battle of the Mesh, 2015**

The firmware lockdown may have backfired

- OpenWRT did not go away, instead we have dozens of forks with various degrees of ”freshness”
- The proprietary derivatives of OpenWRT are unlikely to provide root access
- Rather than disabling “unwanted” functionality to common users, it hides **unforgivable*** bugs and ”**interesting engineering decisions**”

*https://www.ncsc.gov.uk/report/a-method-to-assess-forgivable-vs-unforgivable-vulnerabilities

Where to begin?

- There are three ways to achieve this (from **easy** to **hard**):
  - Read the **documentation**, **user guides**, and **forums**
  - Study the **recurring/historical vulnerabilities**, maybe they didn't fix something in the model you are looking at?
  - Find a **new vulnerability** and pop up a reverse shell

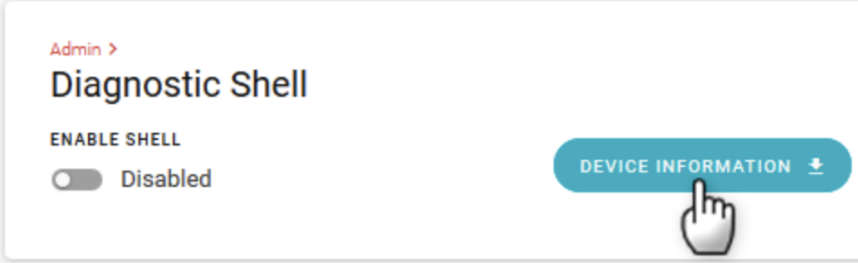
- Things to look at:
  - **Backdoors**, e.g., "technical support features"
  - **Web-based UI** – we have never seen one without bugs
  - Any **functionality that translates settings from a WebUI** into the system

Read the docs: Sierra Wireless

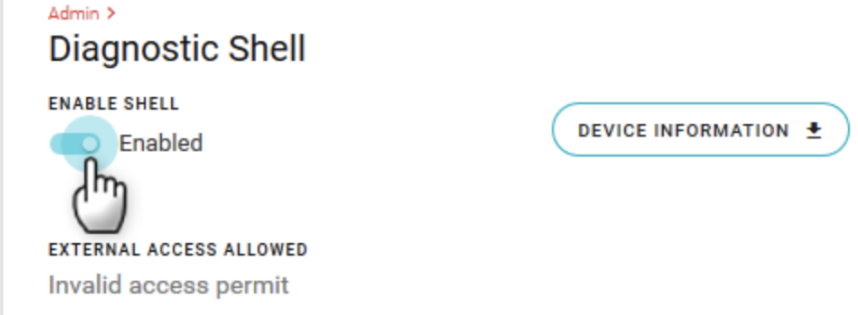
When remote help is required, enabling the Diagnostic Shell allows Sierra Wireless support to access your device via SSH.

> Do not enable this feature unless directed by Sierra Wireless support.

1. After being directed by Sierra Wireless support, click **DEVICE INFORMATION**.



2. Save the request.txt file.
3. Send the request file to Sierra Wireless support, who will generate an access permit file for you to upload to the device.
4. Enable the Diagnostic Shell.



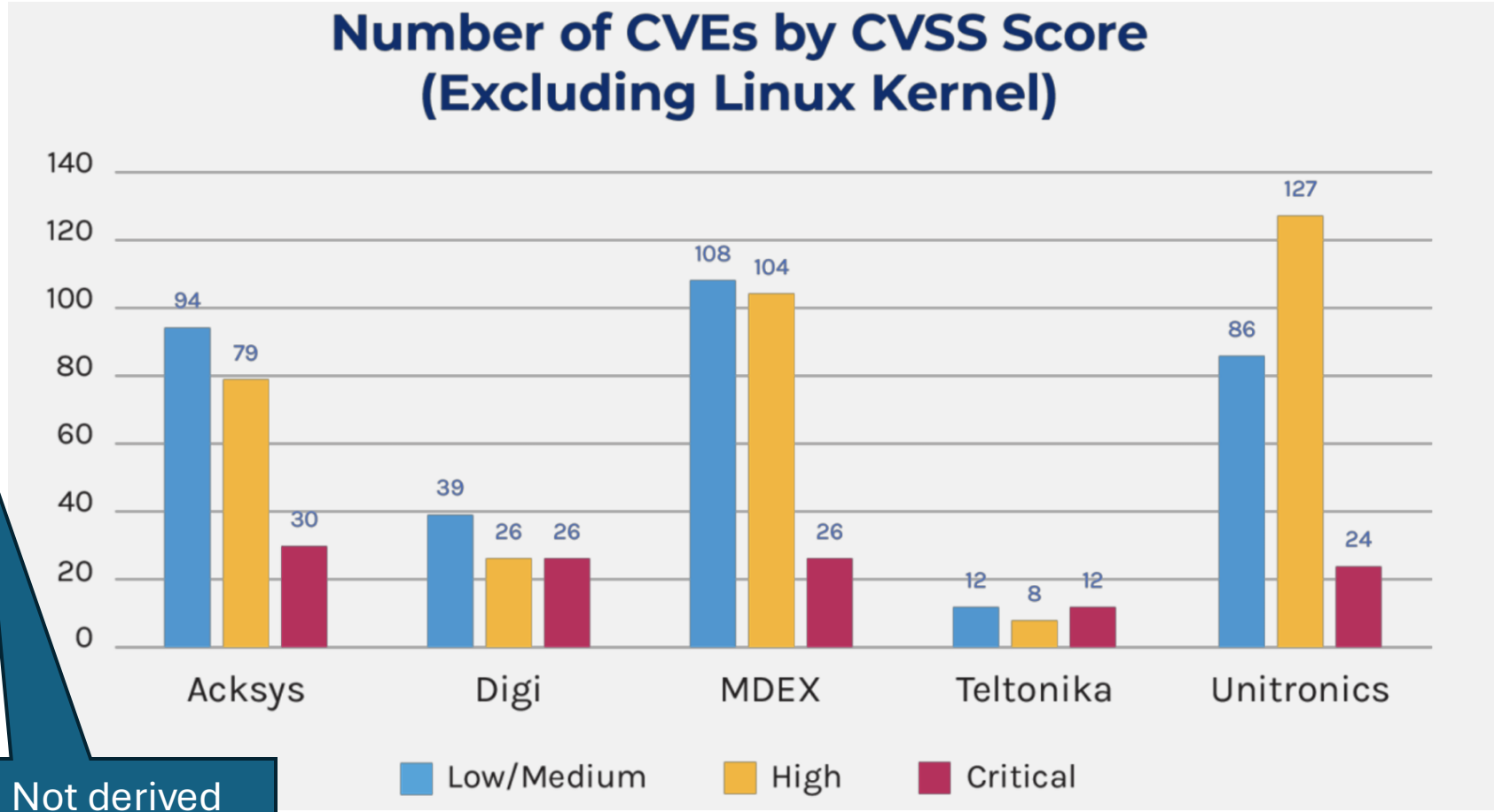

```
root:$1$SKo0RWlm$/0nRV0MbO1ZfXXXXXXXXX.:0:0::/root:/bin/sh
...
nobody:*:99:99:nobody:/:/bin/false
uasuser:*:1000:16:Linux User,,,:/tmp/RA:/bin/sh
rauser:*:1001:16:Linux User,,,:/tmp/RA:/bin/false
sconsole:*:0:0::/root:/usr/local/sbin/runpico
acemanager:x:1002:1001:Acemanager UI:/tmp/acemanager:/bin/false
logs:x:1004:1004:logging:/mnt/hda1/junxion/log:/bin/false
```

*https://source.sierrawireless.com/airlinkos/XR90-5.1/reference/system/howto/diagnosticshell/

Historical vulnerabilities: proprietary OpenWRT

Vendor	Model(s)	Version (Release Date)
Acksys	AirBox LTE, AirWan-M12, AirLink, WaveNet-Ex	4.22.3.1 (Jan/2024)
Digi	IX10	23.9.20.67 (Dec/2023)
MDEX	MX880	02.517 (Oct/2022)
Teltonika	RUT950	00.07.06.1 (May/2023)
Unitronics	UCR	51.06.06.185 (Sept/2021)

Not derived from OpenWRT



Recurring vulnerabilities: DrayTek

CVE ID	Description	CVSS v3.1
CVE-2023-47254	OS command injection in the CLI interface.	9.8
CVE-2023-31447	Stack-based buffer overflow in Web UI via “/cgi-bin/user_login.cgi”. An unspecified payload triggers the issue.	9.8
CVE-2023-24229	OS command injection in Web UI via “/cgi-bin/malfunction.cgi”. The root-cause of the issue is unchecked user input that comes into one of the form / query string parameters.	7.8
CVE-2023-1162	OS command injection in Web UI via “/cgi-bin/malfunction.cgi”. The root-cause of the issue is unchecked user input that comes into one of the form / query string parameters.	8.8
CVE-2022-32548	Stack-based buffer overflow in Web UI via “/cgi-bin/wlogin.cgi”. The issue can be triggered with malformed input going into “username” or “password” form / query string parameters.	9.8
CVE-2021-43118	OS command injection in Web UI via “/cgi-bin/malfunction.cgi”. The root-cause of the issue is unchecked user input that comes into one of the form / query string parameters.	9.8
CVE-2021-42911	Stack-based buffer overflow in Web UI via “/cgi-bin/malfunction.cgi”. A format string issue with one of the form / query string parameters.	9.8
CVE-2020-19664	OS command injection in Web UI via “/cgi-bin/malfunction.cgi”. The root-cause of the issue is unchecked user input that comes into one of the form / query string parameters.	8.8
CVE-2020-15415	OS command injection in Web UI via “/cgi-bin/malfunction.cgi”. The root-cause of the issue is unchecked user input that comes into one of the form / query string parameters.	9.8
CVE-2020-14472	OS command injection in Web UI via “/cgi-bin/malfunction.cgi”. The root-cause of the issue is unchecked user input that comes into one of the form / query string parameters.	9.8
CVE-2020-14993	Stack-based buffer overflow in Web UI via “/cgi-bin/malfunction.cgi”. User input from one of the form / query string parameters is not checked.	9.8

CVE-2020-10823	Stack-based buffer overflow in Web UI via “/cgi-bin/activate.cgi”. User input from one of the form / query string parameters is not checked.	9.8
CVE-2020-10824	Stack-based buffer overflow in Web UI via “/cgi-bin/activate.cgi”. User input from one of the form / query string parameters is not checked.	9.8
CVE-2020-10825	Stack-based buffer overflow in Web UI via “/cgi-bin/activate.cgi”. User input from one of the form / query string parameters is not checked.	9.8
CVE-2020-10826	OS command injection in Web UI via “/cgi-bin/activate.cgi”. User input from one of the form / query string parameters is not checked.	9.8
CVE-2020-10827	Stack-based buffer overflow in the “apcmd” service.	9.8
CVE-2020-10828	Stack-based buffer overflow in the “cvmd” service.	9.8
CVE-2020-8515	Stack-based buffer overflow in Web UI via “/cgi-bin/malfunction.cgi”. User input from one of the form / query string parameters is not checked	9.8

10 Stack buffer overflows

8 OS command injections

Today’s target: TP-Link ER605 v2

- Based on the MediaTek MT7621 SoC with 5 10/100/1000 Mbps Ethernet ports
- Used to “support” OpenWRT

  OpenWrt can be installed by first getting a root shell on the device. The root password is derived from the device MAC and the admin user ID. The whole process is described in this GitHub repository

  - https://openwrt.org/toh/tp-link/er605_v2
  - https://github.com/chill1Penguin/er605v2_openwrt_install/

```bash
$> echo -ne "A8:6E:84:70:9E:30adminA8:6E:84:70:9E:30admin" | md5sum | awk '{print $1}' | cut -c1-16
$> 191564caff3aaf0c
```

TP-Link: no more OpenWRT

[redacted] opened on Aug 19, 2024

Hi!

I can't generate root password (CLI debug password) on version 2.2.5 of router. CLI says "Password is wrong". :(

P.S.
MAC address and login is 100% correct

[redacted] Aug 22, 2024 · [redacted] Edits ···

Having same issue. Triple checked MAC address and tried it with both AA:BB:CC:DD:EE:FF and AA-BB-CC-DD-EE-FF formatting. "Password is wrong" is the only response that comes back on the 605. This is hardware version 2.6. It appears that it gives you some type of random code when you type debug that you use as a seed for something.

Firmware: 2.2.6 Build 20240718 Rel.82712

[redacted] on Jan 31

I can't even downgrade the firmware...
Everything looks like downgrade was sucessful, but it still boots up with V2.2.5.

I don't know if it is related, but I have HW 2.2. On the DL page of HW2.2, nothing older than 2.2.5 is even listed:
https://support.omadanetworks.com/en/product/er605/v2.20/?resourceType=download

I am woried, that router detects "wrong FW", and just rejects the downgrade. :(

Any help would be appreciated. I bought this router specifically for openWRT, so it would be a shame, if I would not be able to use it in this was.

TP-Link: CVE-2024-21827

TALOS-2024-1947

Tp-Link ER7206 Omada Gigabit VPN Router cli_server debug leftover debug code vulnerability

JUNE 25, 2024

CVE NUMBER

CVE-2024-21827

VENDOR RESPONSE

The vendor provided a fix at: https://www.tp-link.com/en/support/download/er7206/v1/#Firmware

TIMELINE

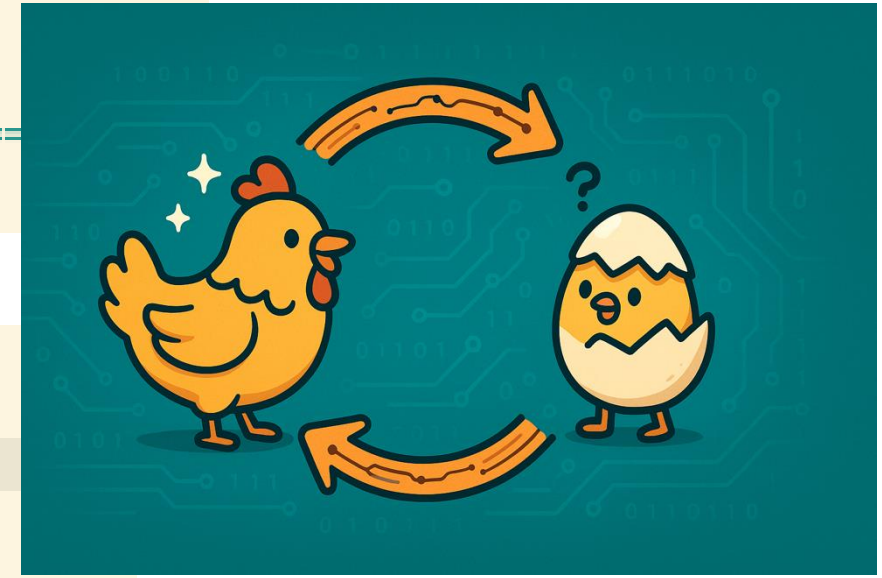
2024-02-21 - Vendor Disclosure
2024-06-20 - Vendor Patch Release
2024-06-25 - Public Release

TP-Link: CVE-2025-7851

```
>debug
================================================================
Sign   String:    Y7YP4HZ7ROEOH85
Sign   Platform:  TP-LINK_SOHOI_Platform
================================================================
Please Input Sign Result:
```

```
//...

if ( access("/usr/sbin/image_type_debug", 0) ) {
    // Send a challenge and verify the signature (requires the private key).
    // If the signature is correct, grant the root shell.
}
else {
    // Ask for a password derived from the LAN mac address.
    // If the password is correct, grant the root shell.
}
//...
```



Getting root this way requires having root or obtaining that private key.

But then it gets more interesting…

```
143   if ( access("/usr/sbin/image_type_debug", 0) )
144   {
145     ((void (*)(__int64, const char *, ...))cli_print)(
146       term_related,
147       "================================================================================");
148     ((void (*)(__int64, const char *, ...))cli_print)(term_related, "Sign    String:    %s", (const char *)&v30);
149     ((void (*)(__int64, const char *, ...))cli_print)(term_related, "Sign    Platform:  TP-LINK_SOHOI_Platform");
150     ((void (*)(__int64, const char *, ...))cli_print)(
151       term_related,
152       "================================================================================");
153     cli_print_no_enter(term_related, "Please Input Sign Result: ");
154     input = maybe_read_input(term_related, v45, 172);
155     v45[172] = 0;
156     if ( input != -1 )
157     {
158       if ( input )
159         return 0LL;
160       sub_40DD80((unsigned __int8 *)v45, 172, s);
161       v25 = strlen(
162               "BgIAAACkAABSU0ExAAQAAAEAAQD91xDCQ5DFNSYJBriTmTmZlEMYVgGcZTO+AIwmdVjhaeJI6wWtN7DqCaHQlOqJ2xvKNrLB+wA1NxUh"
163               "7VDViymotq/+9QDf7qEtJHmesjirvPN6Hfrf+FO4/hmjbVXgytHORxGta5KW4QHVIwyMSVPOvMC4A51FIh+D1kJW5GXWtA==");
164       v23 = sub_40DE5C(
165               "BgIAAACkAABSU0ExAAQAAAEAAQD91xDCQ5DFNSYJBriTmTmZlEMYVgGcZTO+AIwmdVjhaeJI6wWtN7DqCaHQlOqJ2xvKNrLB+wA1NxUh"
166               "7VDViymotq/+9QDf7qEtJHmesjirvPN6Hfrf+FO4/hmjbVXgytHORxGta5KW4QHVIwyMSVPOvMC4A51FIh+D1kJW5GXWtA==",
167               v25,
168               (__int64)&v30,
169               0xFu,
170               (__int64)s,
171               0x80uLL);
172       if ( (_DWORD)v23 == 1 )
173         goto LABEL_27;
174     }
175   }
```

Same private key is now used to sign firmware
(remember that “can’t downgrade fw” post?)

Finding another way

- We could try to **downgrade the firmware**, but our goal was to study the latest version

- We needed some kind of **OS command execution**
  - At the very least, we could create that "***image_type_debug***" file, and proceed with the good old "backdoor"

- We had to take a step back and find a "quick and easy" target

LuCI: the short(est) intro

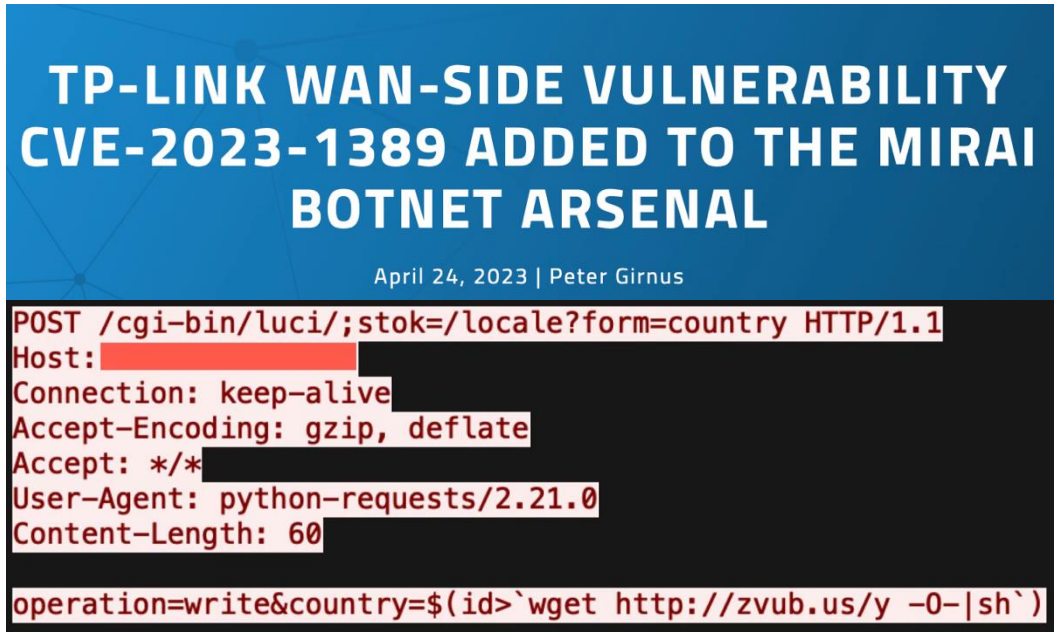
- After some back and forth, we decided to look at LuCI
  - https://github.com/openwrt/luci/wiki/

- Clean, extensible, and easily maintainable web user interface for embedded devices ©

- Built around the Unified Configuration Interface (UCI) principles that serves as a system for centralizing the configuration of OpenWRT services

- Uses the Lua programming language and splits up the interface into logical parts, e.g., models and views

LuCI and TP-Link: past research



https://www.zerodayinitiative.com/blog/2023/4/21/tp-link-wan-side-vulnerability-cve-2023-1389-added-to-the-mirai-botnet-arsenal

REPORT ID	TITLE	REPORT DATE	CVE NUMBER
TALOS-2023-1855	TP-Link ER7206 Omada Gigabit VPN Router uhttpd GRE command injection vulnerability	2024-02-06	CVE-2023-47167
TALOS-2023-1859	TP-Link ER7206 Omada Gigabit VPN Router uhttpd web filtering Command injection Vulnerability	2024-02-06	CVE-2023-47618
TALOS-2023-1856	TP-Link ER7206 Omada Gigabit VPN Router uhttpd PPTP global config Command injection Vulnerability	2024-02-06	CVE-2023-42664
TALOS-2023-1858	TP-Link ER7206 Omada Gigabit VPN Router uhttpd web group command injection vulnerability	2024-02-06	CVE-2023-47617
TALOS-2023-1854	TP-Link ER7206 Omada Gigabit VPN Router uhttpd ipsec command injection vulnerability	2024-02-06	CVE-2023-47209
TALOS-2023-1857	TP-Link ER7206 Omada Gigabit VPN Router uhttpd Wireguard VPN command injection vulnerability	2024-02-06	CVE-2023-46683
TALOS-2023-1853	TP-Link ER7206 Omada Gigabit VPN Router uhttpd PPTP client Command injection Vulnerability	2024-02-06	CVE-2023-36498
TALOS-2023-1850	TP-Link ER7206 Omada Gigabit VPN Router uhttpd freeStrategy Command injection Vulnerability	2024-02-06	CVE-2023-43482

https://www.talosintelligence.com/vulnerability_reports

Reversing TP-Link's LuCI scripts

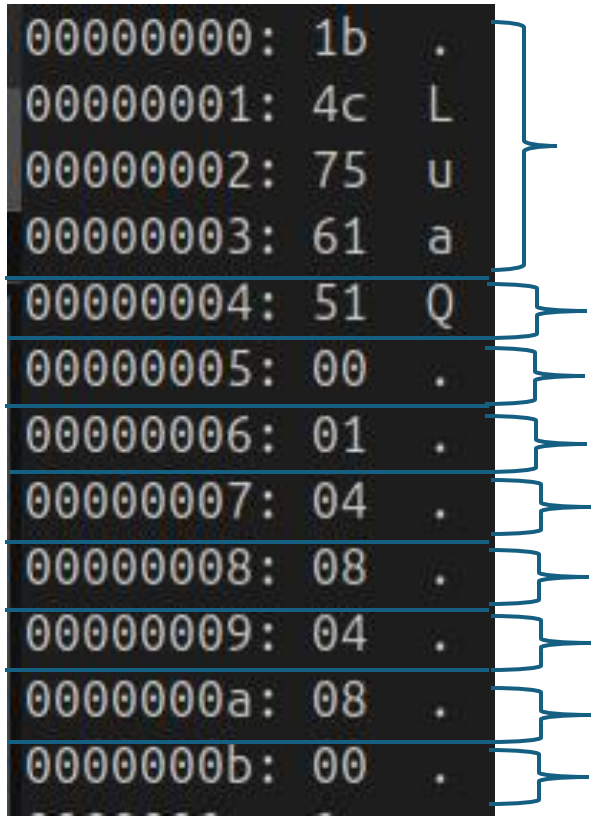
- The LuCI files are compiled into Lua Bytecode, and are executed by the Lua Virtual Machine (LVM)
- Typically, it's trivial to convert Lua Bytecode in human-readable scripts, but we couldn't

```
standash@thelab42-1:~/stuff/vr/ot_firewalls/tplink/lua_deobfuscate/luadec/luadec$ ./luadec ~/Downloads/temp/wizard.lua
./luadec: /home/standash/Downloads/temp/wizard.lua: bad header in precompiled chunk
```

Did they modify LVM?

https://luadec.metaworm.site

Lua Bytecode: the header block*



Header signature (“\x1bLua” or 0x1b4c7561)

Lua version (high hex digit – major; low hex digit – minor)

Format version (0 == official version)

Endianness (0 == big; 1 == little)

Size of int (4 bytes by default)

Size of size_t (4 bytes by default)

Size of an instruction (4 bytes by default)

Size of lua_Number (8 bytes by default)

Integral flag (0 == floating point; 1 == integral number type)

*Kein-Hong Man, esq. “A No-Frills Introduction to Lua 5.1 VM Instructions”

Lua Bytecode: constant lists

```
00000000: 1b  .
00000001: 4c  L
00000002: 75  u
00000003: 61  a
00000004: 51  Q
00000005: 00  .
00000006: 01  .
00000007: 04  .
00000008: 08  .
00000009: 04  .
0000000a: 08  .
0000000b: 00  .
```

Constant list
Holds list of constants referenced in the function (it's a constant pool.)

Integer size of constant list (sizek)
[
  1 byte type of constant (value in parentheses):
  • 0=LUA_TNIL, 1=LUA_TBOOLEAN,
  • 3=LUA_TNUMBER, 4=LUA_TSTRING
  Const the constant itself: this field does not exist if the constant type is 0; it is 0 or 1 for type 1; it is a **Number** for type 3, or a **String** for type 4.
]

```
00000270: 04  .
00000271: 07  .
00000272: 00  .
00000273: 00  .
00000274: 00  .
00000275: 6d  m
00000276: 6f  o
00000277: 64  d
00000278: 75  u
00000279: 6c  l
0000027a: 65  e
0000027b: 00  .
```

08 00

```
00001f91: 03  .
00001f92: 00  .
00001f93: 00  .
00001f94: 00  .
00001f95: 00  .
00001f96: 00  .
00001f97: 00  .
00001f98: 30  0
00001f99: 40  @
```

04 01

```
00001f5d: 03  .
00001f5e: 10  .
00001f5f: 00  .
00001f60: 00  .
00001f61: 00  .
```

Lua Bytecode: constant lists

```
00000000: 1b  .
00000001: 4c  L
00000002: 75  u
00000003: 61  a
00000004: 51  Q
00000005: 00  .
00000006: 01  .
00000007: 04  .
00000008: 08  .
00000009: 04  .
0000000a: 08  .
0000000b: 00  .
```

Constant list
Holds list of constants referenced in the function (it's a constant pool.)

Integer size of constant list (sizek)
[
1 byte type of constant (value in parentheses):
• 0=LUA_TNIL, 1=LUA_TBOOLEAN,
• 3=LUA_TNUMBER, 4=LUA_TSTRING
Const the constant itself: this field does not exist if the constant type is 0; it is 0 or 1 for type 1; it is a **Number** for type 3, or a **String** for type 4.
]

```
00000270: 04  .
00000271: 07  .
00000272: 00  .
00000273: 00  .
00000274: 00  .
00000275: 6d  m
00000276: 6f  o
00000277: 64  d
00000278: 75  u
00000279: 6c  l
0000027a: 65  e
0000027b: 00  .
```

“module\x00”

08 00

```
00001f91: 03  .
00001f92: 00  .
00001f93: 00  .
00001f94: 00  .
00001f95: 00  .
00001f96: 00  .
00001f97: 00  .
00001f98: 30  0
00001f99: 40  @
```

04 01

```
00001f5d: 03  .
00001f5e: 10  .
00001f5f: 00  .
00001f60: 00  .
00001f61: 00  .
```

Lua Bytecode: constant lists

```
00000000: 1b  .
00000001: 4c  L
00000002: 75  u
00000003: 61  a
00000004: 51  Q
00000005: 00  .
00000006: 01  .
00000007: 04  .
00000008: 08  .
00000009: 04  .
0000000a: 08  .
0000000b: 00  .
```

Constant list
Holds list of constants referenced in the function (it's a constant pool.)

Integer size of constant list (sizek)
[
1 byte type of constant (value in parentheses):
• 0=LUA_TNIL, 1=LUA_TBOOLEAN,
• 3=LUA_TNUMBER 4=LUA_TSTRING
Const the constant itself: this field does not exist if the constant type is 0; it is 0 or 1 for type 1; it is a **Number** for type 3, or a **String** for type 4.
]

```
00000270: 04  .
00000271: 07  .
00000272: 00  .
00000273: 00  .
00000274: 00  .
00000275: 6d  m
00000276: 6f  o
00000277: 64  d
00000278: 75  u
00000279: 6c  l
0000027a: 65  e
0000027b: 00  .
```

08 00

```
00001f91: 03  .
00001f92: 00  .
00001f93: 00  .
00001f94: 00  .
00001f95: 00  .
00001f96: 00  .
00001f97: 00  .
00001f98: 30  0
00001f99: 40  @
```

04 01

```
00001f5d: 03  .
00001f5e: 10  .
00001f5f: 00  .
00001f60: 00  .
00001f61: 00  .
```

16 (0x10)

16.0
(0x4030000000000000)

TP-Link’s Lua Bytecode

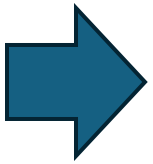
```
00000000: 1b  .
00000001: 4c  L
00000002: 75  u
00000003: 61  a
00000004: 51  Q
00000005: 00  .
00000006: 01  .
00000007: 04  .
00000008: 04  .
00000009: 04  .
0000000a: 08  .
0000000b: 04  .
```

08 04

```
00001f5d: 09  .
00001f5e: 10  .
00001f5f: 00  .
00001f60: 00  .
00001f61: 00  .
```

TP-Link's Lua Bytecode (fixed)

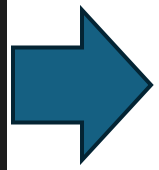
```
00000000: 1b  .
00000001: 4c  L
00000002: 75  u
00000003: 61  a
00000004: 51  Q
00000005: 00  .
00000006: 01  .
00000007: 04  .
00000008: 04  .
00000009: 04  .
0000000a: 08  .
0000000b: 04  .
```



```
00000000: 1b  .
00000001: 4c  L
00000002: 75  u
00000003: 61  a
00000004: 51  Q
00000005: 00  .
00000006: 01  .
00000007: 04  .
00000008: 04  .
00000009: 04  .
0000000a: 04  .
0000000b: 01  .
```

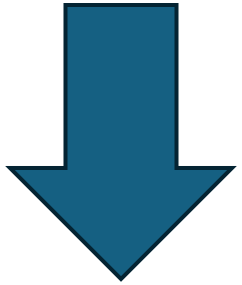
08 04

```
00001f5d: 09  .
00001f5e: 10  .
00001f5f: 00  .
00001f60: 00  .
00001f61: 00  .
```



04 01

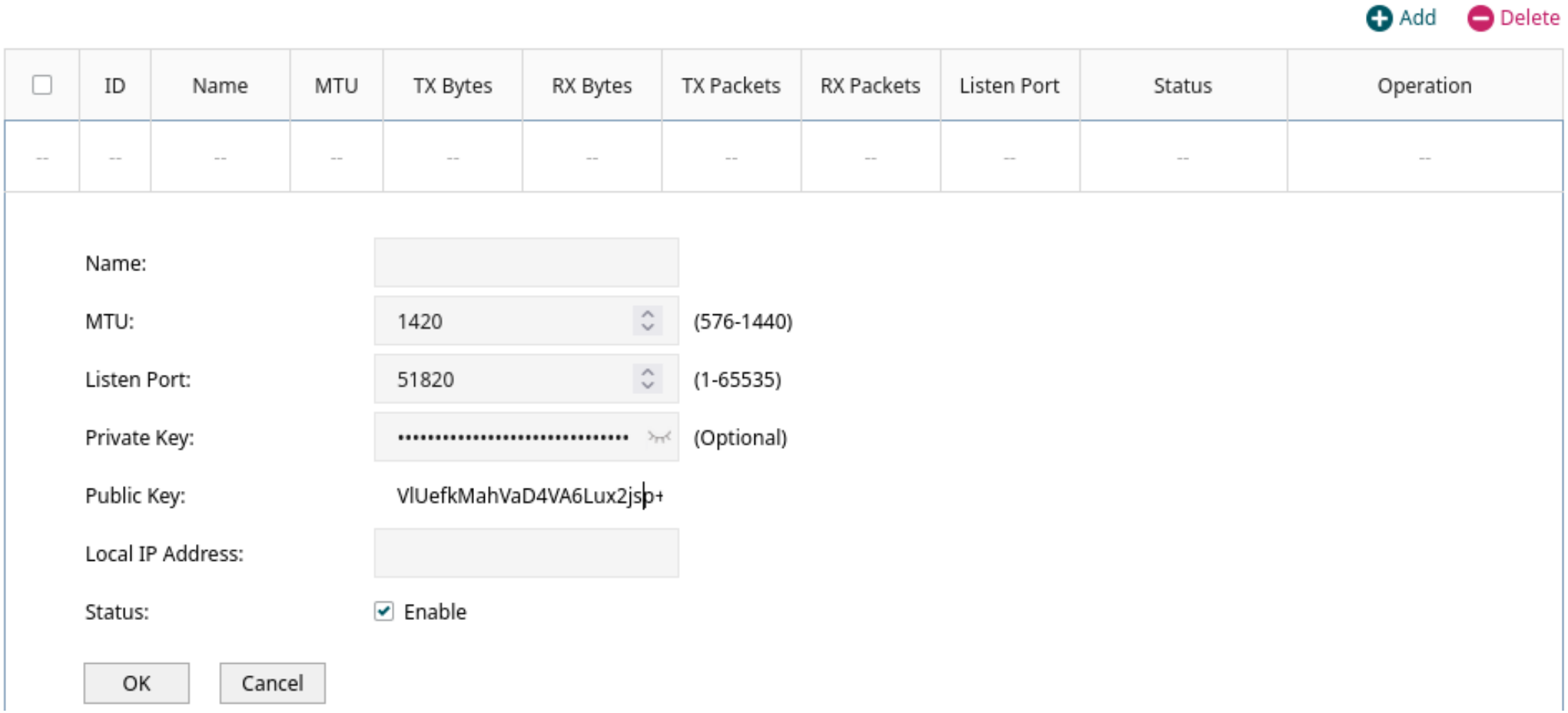
```
00001f5d: 03  .
00001f5e: 10  .
00001f5f: 00  .
00001f60: 00  .
00001f61: 00  .
```



```lua
local r1_14 = string.sub(r0_14, 1, 6)
local r3_14 = tonumber(string.sub(r0_14, -6, -1), 16)
local r4_14 = {}
local r5_14 = {}
```

TP-Link’s LuCI: they’ve missed one thing

- Wireguard VPN private key setting



TP-Link's LuCI: CVE-2025-7850

- Wireguard VPN private key setting

Newline characters are not sanitized

```
209  function check_key(r0_14, r1_14)
210    -- line: [226, 241] id: 14
211    local r2_14 = "echo \"" .. r9_0.safeSpecialString(r0_14) .. "\" |wg pubkey"
212    if r4_0.fork_call(r2_14) == 1 then
213      return true
214    end
215    if r1_14 then
216      r1_14.public_key = io.popen(r2_14):read("*a")
217    end
218    return false
219  end
```

```
function safeSpecialString(r0_51)
  -- line: [517, 532] id: 51
  r0_51 = r5_0(r0_51)
  local r1_51 = ""
  for r5_51 = 1, #r0_51, 1 do
    local r6_51 = r0_51:sub(r5_51, r5_51)
    if r6_51 == "$" or r6_51 == "`" or r6_51 == "\\" or r6_51 == "\"" then
      r6_51 = "\\" .. r6_51
    end
    r1_51 = r1_51 .. r6_51
  end
  return "\"" .. r1_51 .. "\""
```

```
echo ""KPzPGI6WQ7z6bfwKYrQtpwRQwzgaU8/6wVEUtRIlslI="" | wg pubkey

echo ""hello\n{ARBITRARY_OS_COMMAND}"" | wg pubkey
```

Demo time!

Takeaways

- **There is always a way to root a "locked down" router**, especially if its operating system is not built from the ground up*
- Often, **one does need** to know how to emulate firmware images or have any **hardware hacking skills** to root a network device
- **Stand on the shoulders of giants –** study documentation, forums and prior work
- **"Locking down"** forks of open-source firmware **can only introduce more security issues**

* That's (almost) never the case!