Breakdown of My Research Process

By: Sam Quinn

What is zero-day research?

- 0-days = Vulnerabilities unknown to the vendor or public
- Often as close to “real” hacking as it gets
- Often not able to be detected through AV

Why do zero-day research?

- Fun and challenging
- Help secure products
- Legally get to hack things
- Hold vendors accountable

Step #1: Choosing a Research Target

Finding a Target

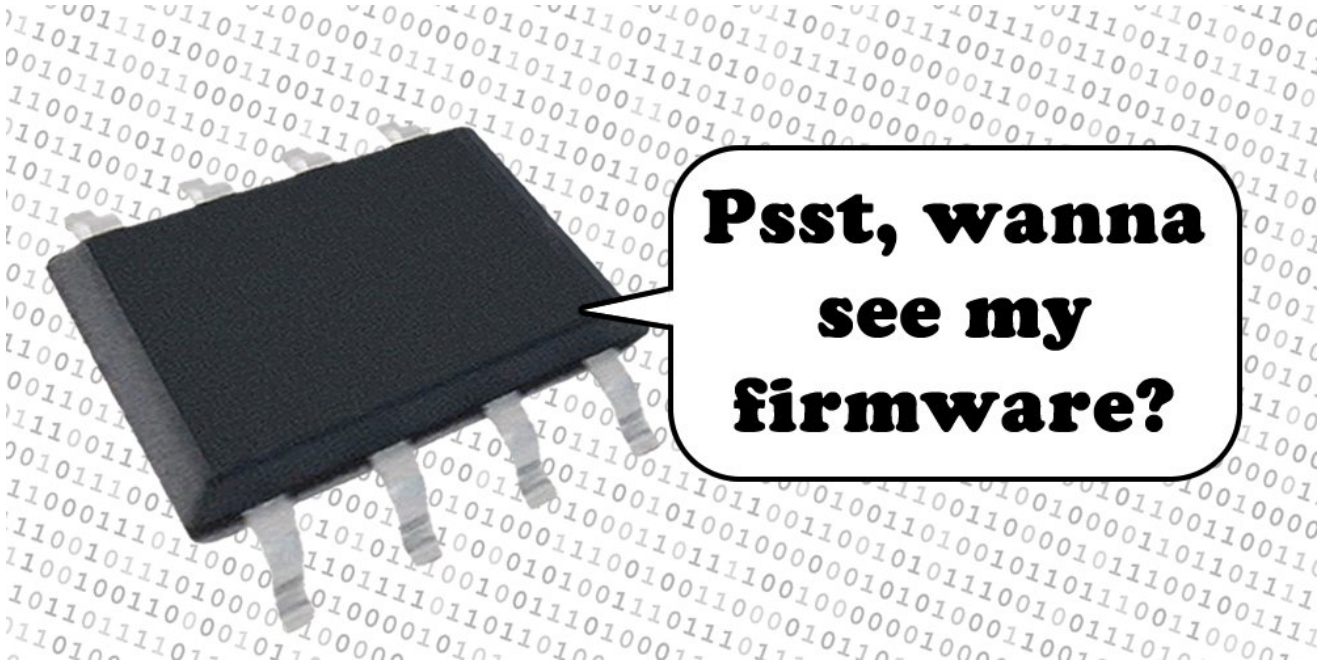
- Look at things you know
  - IOT Devices
  - Mobile Devices
  - Current OS

Finding a Target

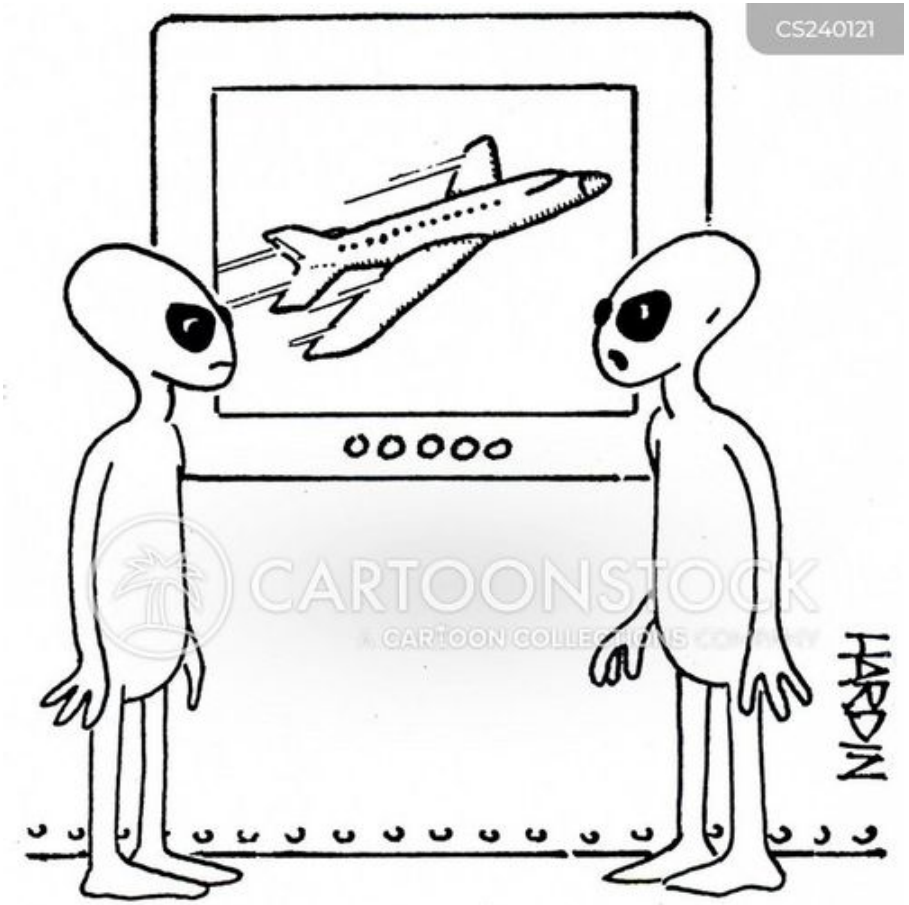
- Look at things you know
  - IOT Devices
  - Mobile Devices
  - Current OS
- Software
  - Free Trials
  - Software you own (video games)

Finding a Target

- Look at things you know
  - IOT Devices
  - Mobile Devices
  - Current OS
- Software
  - Free Trials
  - Software you own (video games)
- Hardware
  - Firmware Images
  - Emulation



Step #2: Setup and Configuration



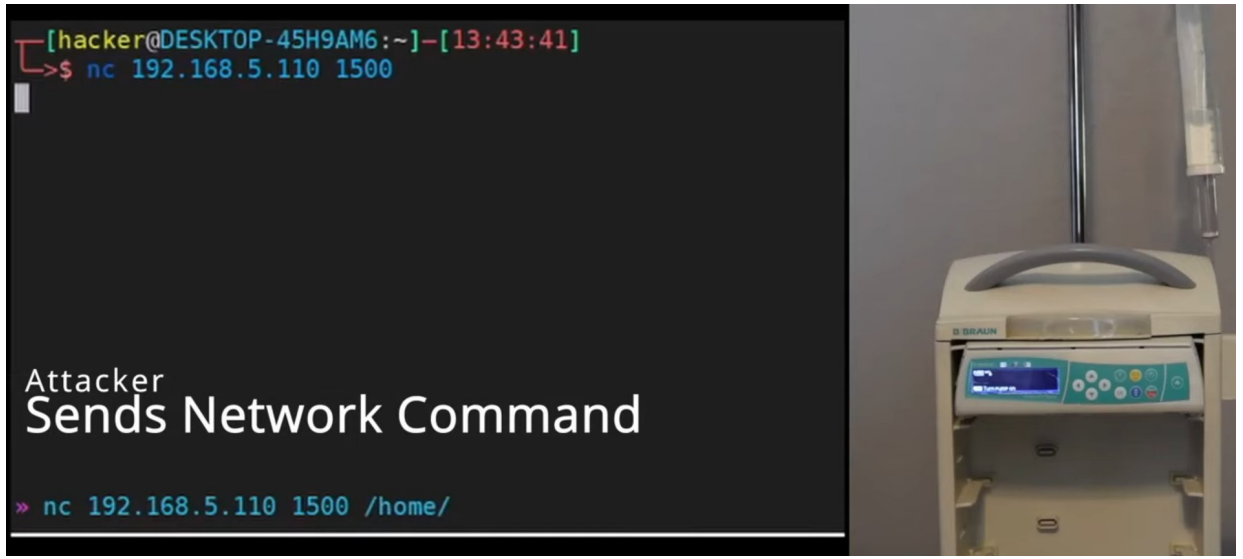
"It's obviously a fraud - Such a configuration could never get off the ground."

Normal Operation

- Ensure Normal setup
  - Make sure attack vectors are legit
  - Default configuration

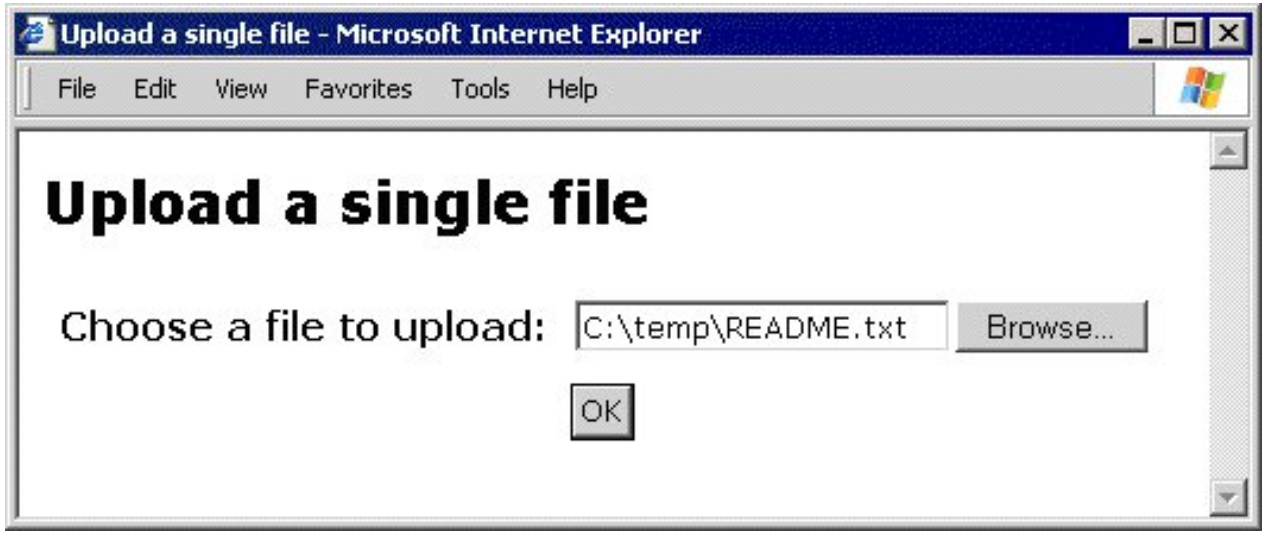
Normal Operation

- Ensure Normal setup
  - Make sure attack vectors are legit
  - Default configuration
- Learn the Ins and Outs
  - Know how the target works
  - Become an expert



Normal Operation

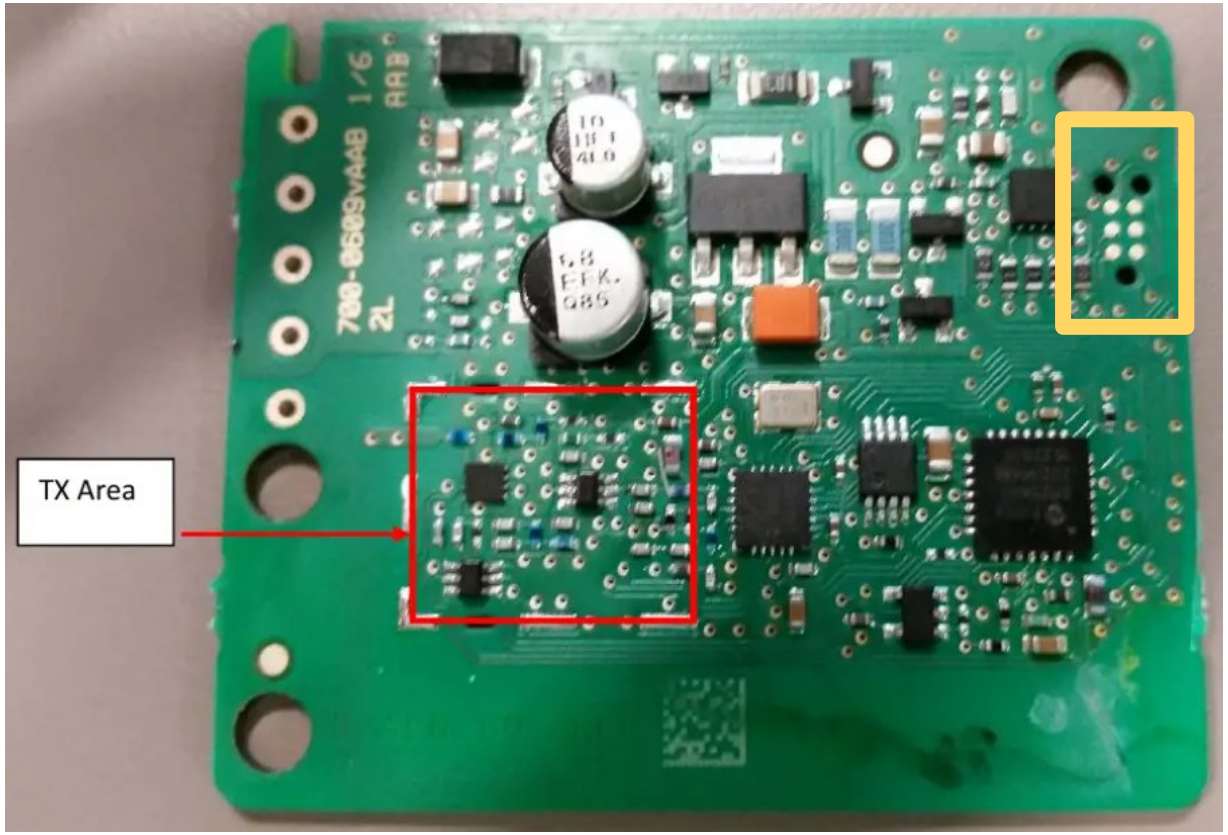
- Ensure Normal setup
  - Make sure attack vectors are legit
  - Default configuration
- Learn the Ins and Outs
  - Know how the target works
  - Become an expert
- Use normal ops to gain advantage
  - File uploads
  - Default passwords



Step #3: Become an Expert

Getting to know your Target

- Hardware? Does it transmit RF?
  - FCC Database
  - https://fccid.io



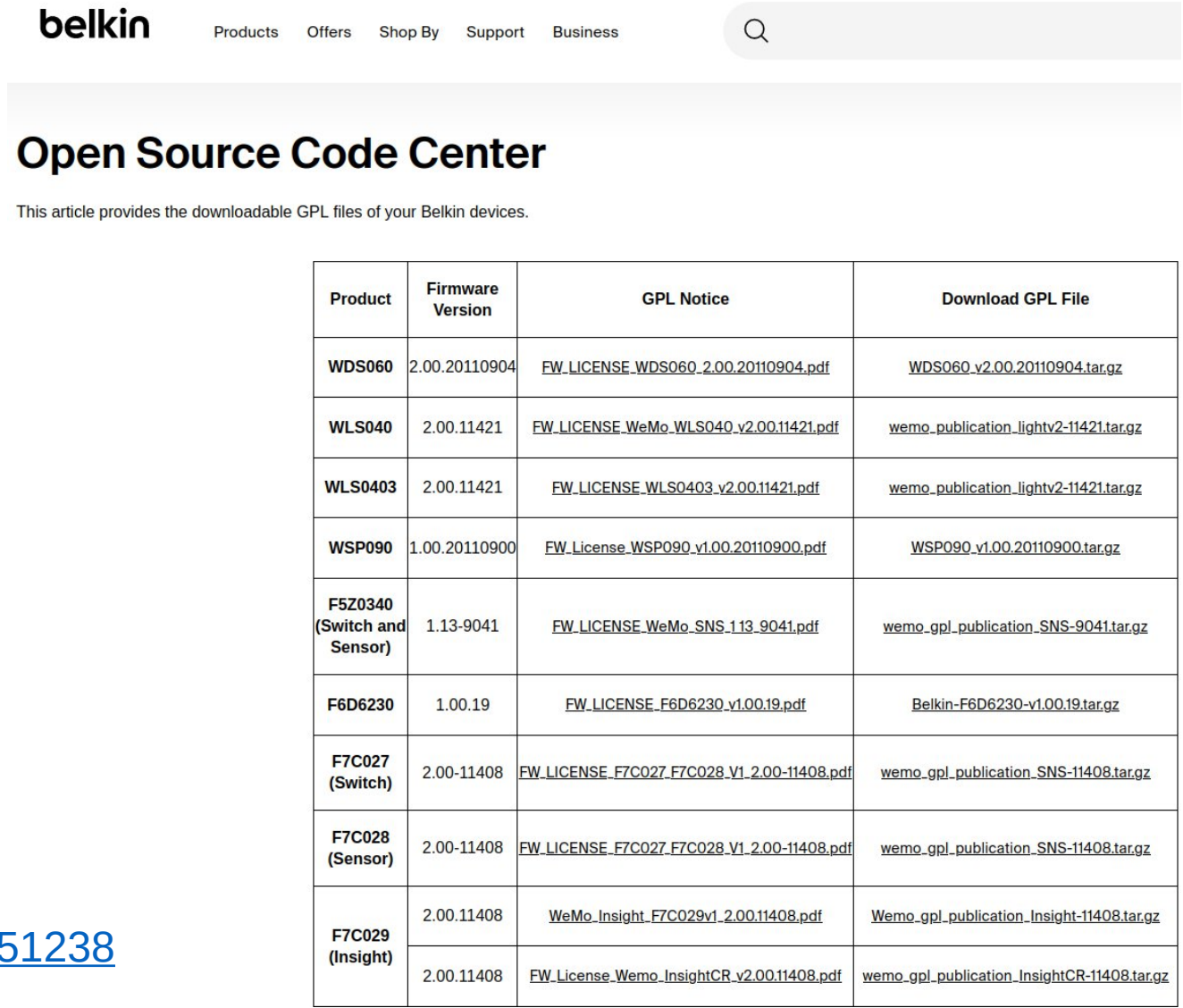
Getting to know your Target

- Hardware? Does it transmit RF?
  - FCC Database
  - https://fccid.io

https://fccid.io/H8N46116A/Internal-Photos/Internal-Photos-6834889.pdf

Getting to know your Target

- Hardware? Does it transmit RF?
  - FCC Database
  - https://fccid.io
- Is it built on FOSS?
  - GPL: Build root and toolchains



belkin Products Offers Shop By Support Business

Open Source Code Center

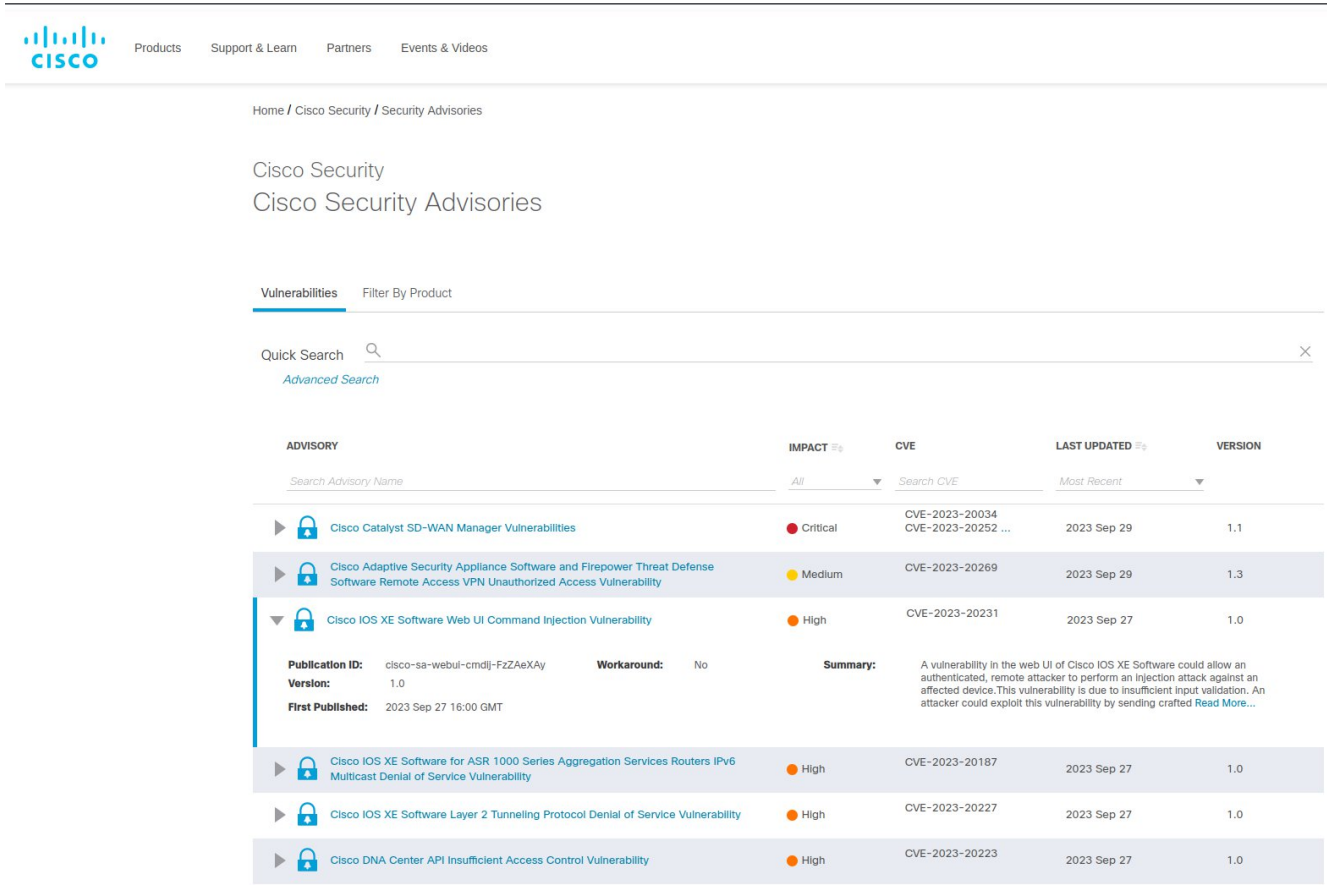
This article provides the downloadable GPL files of your Belkin devices.

Product	Firmware Version	GPL Notice	Download GPL File
WDS060	2.00.20110904	FW_LICENSE_WDS060_2.00.20110904.pdf	WDS060_v2.00.20110904.tar.gz
WLS040	2.00.11421	FW_LICENSE_WeMo_WLS040_v2.00.11421.pdf	wemo_publication_lightv2-11421.tar.gz
WLS0403	2.00.11421	FW_LICENSE_WLS0403_v2.00.11421.pdf	wemo_publication_lightv2-11421.tar.gz
WSP090	1.00.20110900	FW_License_WSP090_v1.00.20110900.pdf	WSP090_v1.00.20110900.tar.gz
F5Z0340 (Switch and Sensor)	1.13-9041	FW_LICENSE_WeMo_SNS_1.13_9041.pdf	wemo_gpl_publication_SNS-9041.tar.gz
F6D6230	1.00.19	FW_LICENSE_F6D6230_v1.00.19.pdf	Belkin-F6D6230-v1.00.19.tar.gz
F7C027 (Switch)	2.00-11408	FW_LICENSE_F7C027_F7C028_V1_2.00-11408.pdf	wemo_gpl_publication_SNS-11408.tar.gz
F7C028 (Sensor)	2.00-11408	FW_LICENSE_F7C027_F7C028_V1_2.00-11408.pdf	wemo_gpl_publication_SNS-11408.tar.gz
F7C029 (Insight)	2.00.11408	WeMo_Insight_F7C029v1_2.00.11408.pdf	Wemo_gpl_publication_Insight-11408.tar.gz
	2.00.11408	FW_License_Wemo_InsightCR_v2.00.11408.pdf	wemo_gpl_publication_InsightCR-11408.tar.gz

https://www.belkin.com/support-article/?articleNum=51238

Getting to know your Target

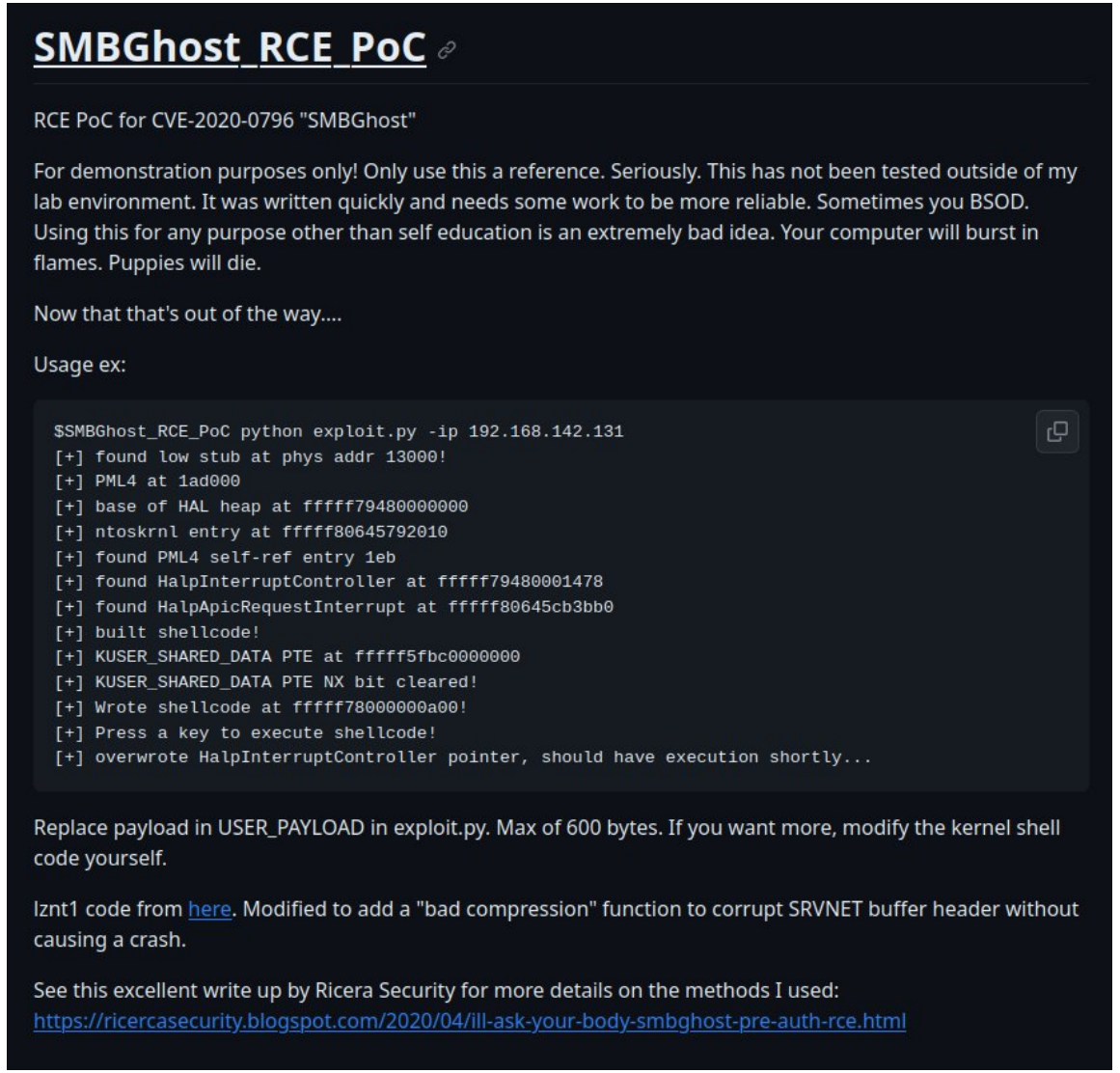
- Hardware? Does it transmit RF?
  - FCC Database
  - https://fccid.io
- Is it built on FOSS?
  - GPL: Build root and toolchains
- Has it been PWNed before?
  - Previous CVEs
  - POCs



https://sec.cloudapps.cisco.com/security/center/publicationListing.x

Getting to know your Target

- Hardware? Does it transmit RF?
  - FCC Database
  - https://fccid.io
- Is it built on FOSS?
  - GPL: Build root and toolchains
- Has it been PWNed before?
  - Previous CVEs
  - POCs



SMBGhost_RCE_PoC

RCE PoC for CVE-2020-0796 "SMBGhost"

For demonstration purposes only! Only use this a reference. Seriously. This has not been tested outside of my lab environment. It was written quickly and needs some work to be more reliable. Sometimes you BSOD. Using this for any purpose other than self education is an extremely bad idea. Your computer will burst in flames. Puppies will die.

Now that that's out of the way....

Usage ex:

```
$SMBGhost_RCE_PoC python exploit.py -ip 192.168.142.131
[+] found low stub at phys addr 13000!
[+] PML4 at 1ad000
[+] base of HAL heap at fffff79480000000
[+] ntoskrnl entry at fffff80645792010
[+] found PML4 self-ref entry 1eb
[+] found HalpInterruptController at fffff79480001478
[+] found HalpApicRequestInterrupt at fffff80645cb3bb0
[+] built shellcode!
[+] KUSER_SHARED_DATA PTE at fffff5fbc0000000
[+] KUSER_SHARED_DATA PTE NX bit cleared!
[+] Wrote shellcode at fffff78000000a00!
[+] Press a key to execute shellcode!
[+] overwrote HalpInterruptController pointer, should have execution shortly...
```

Replace payload in USER_PAYLOAD in exploit.py. Max of 600 bytes. If you want more, modify the kernel shell code yourself.

lznt1 code from here. Modified to add a "bad compression" function to corrupt SRVNET buffer header without causing a crash.

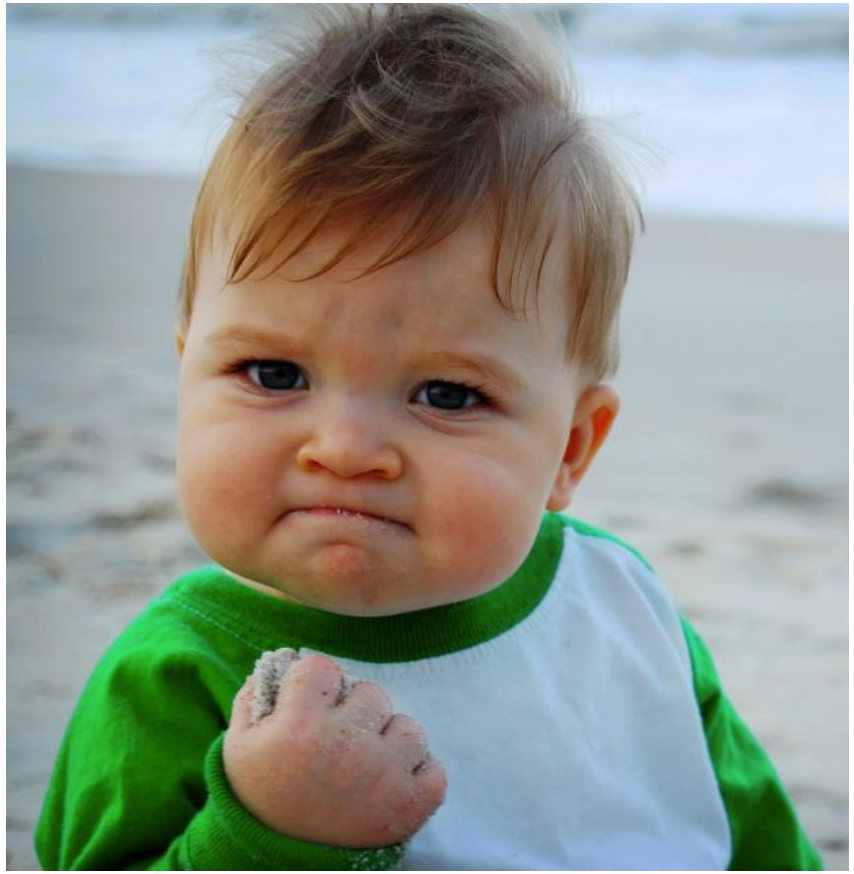
See this excellent write up by Ricera Security for more details on the methods I used:
https://ricercasecurity.blogspot.com/2020/04/ill-ask-your-body-smbghost-pre-auth-rce.html

https://github.com/chompie1337/SMBGhost_RCE_PoC

Step #4: Get Access to the Software

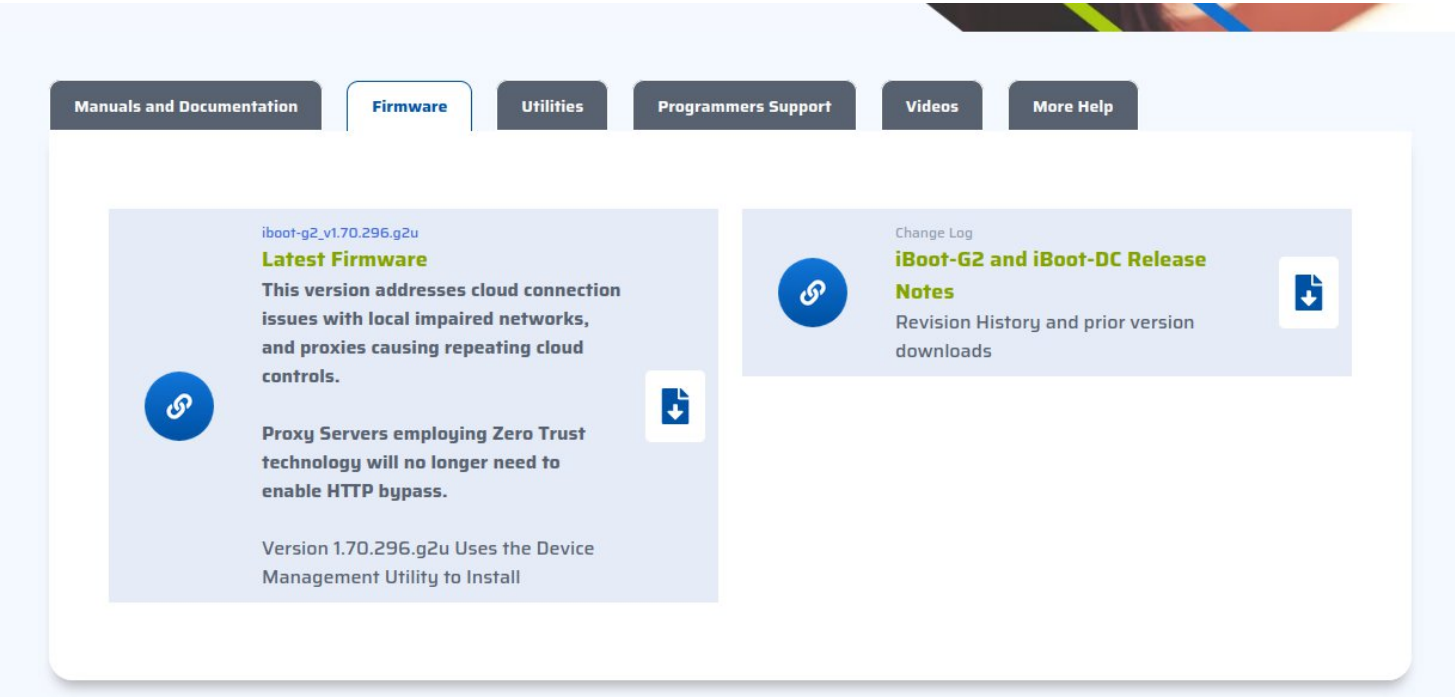
Gaining Access to The Software

- Software Target:
  - You’re done



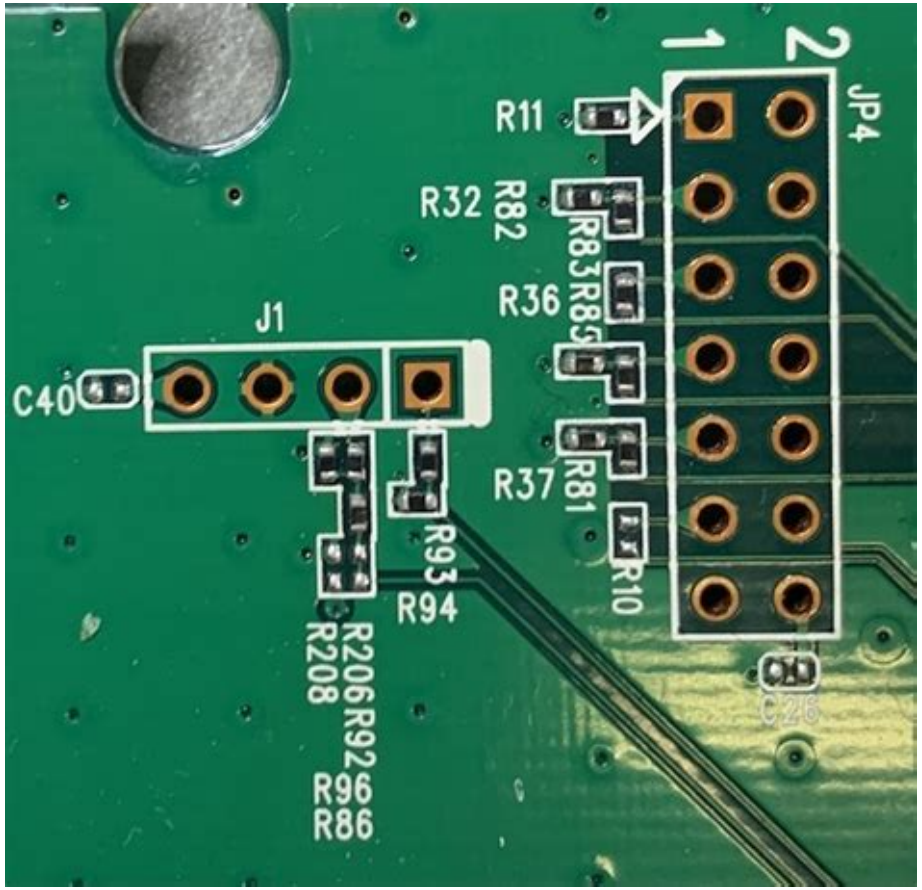
Gaining Access to The Software

- Software Target:
  - You’re done
- Hardware Target:
  - Firmware online



Gaining Access to The Software

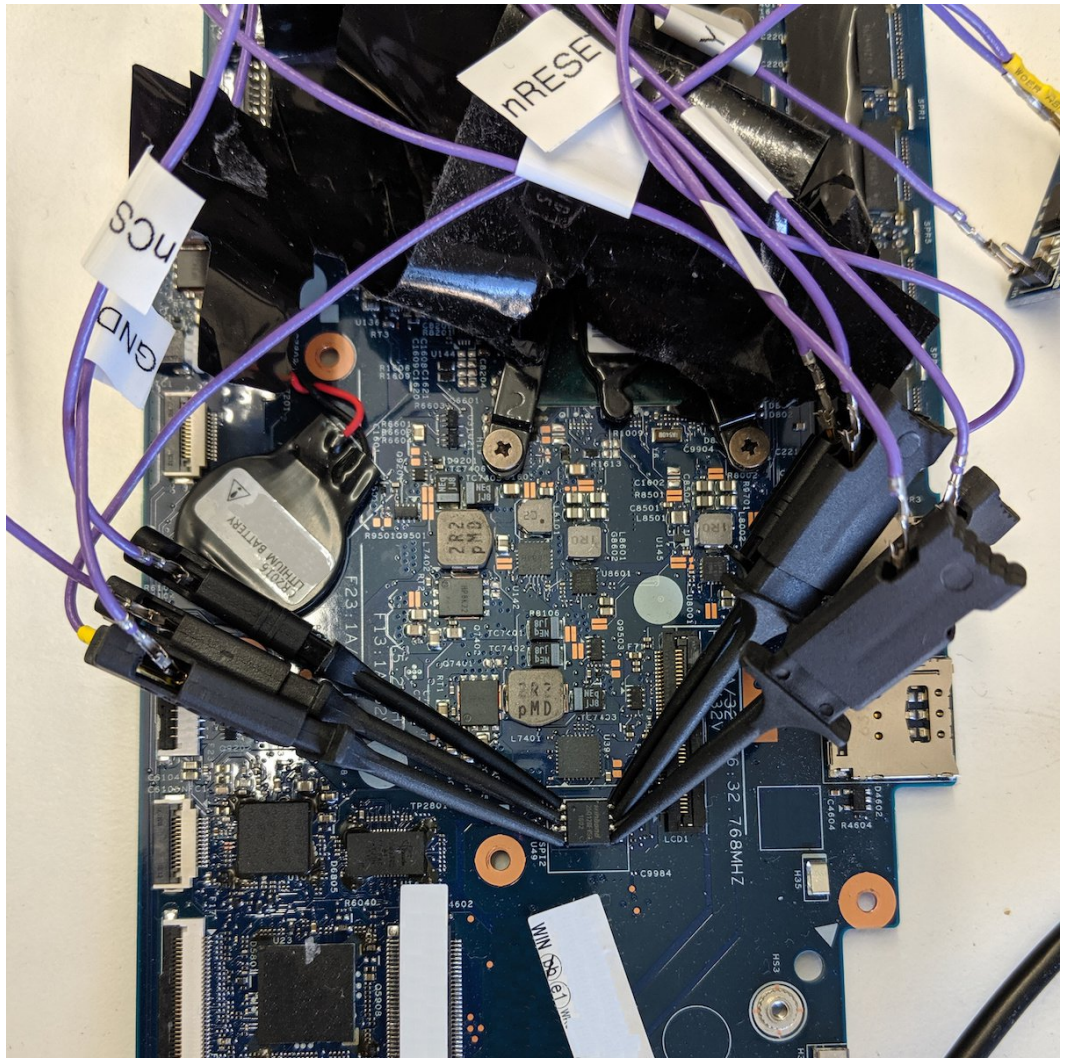
- Software Target:
  - You’re done
- Hardware Target:
  - Firmware online
  - Debug Ports



Quiz: Which one is JTAG and which one is UART?

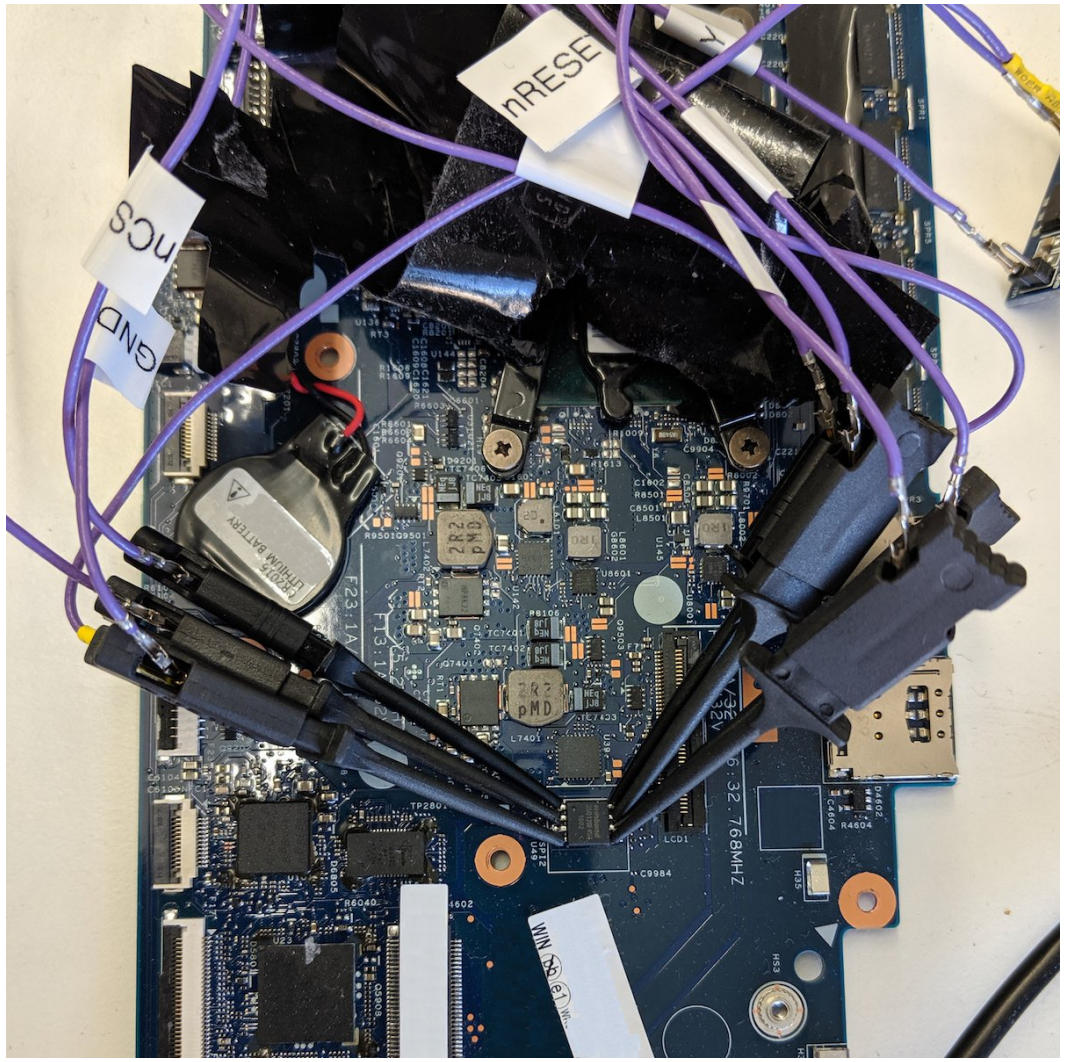
Gaining Access to The Software

- Software Target:
  - You’re done
- Hardware Target:
  - Firmware online
  - Debug Ports
  - Extract from flash



Gaining Access to The Software

- Software Target:
  - You’re done
- Hardware Target:
  - Firmware online
  - Debug Ports
  - Extract from flash



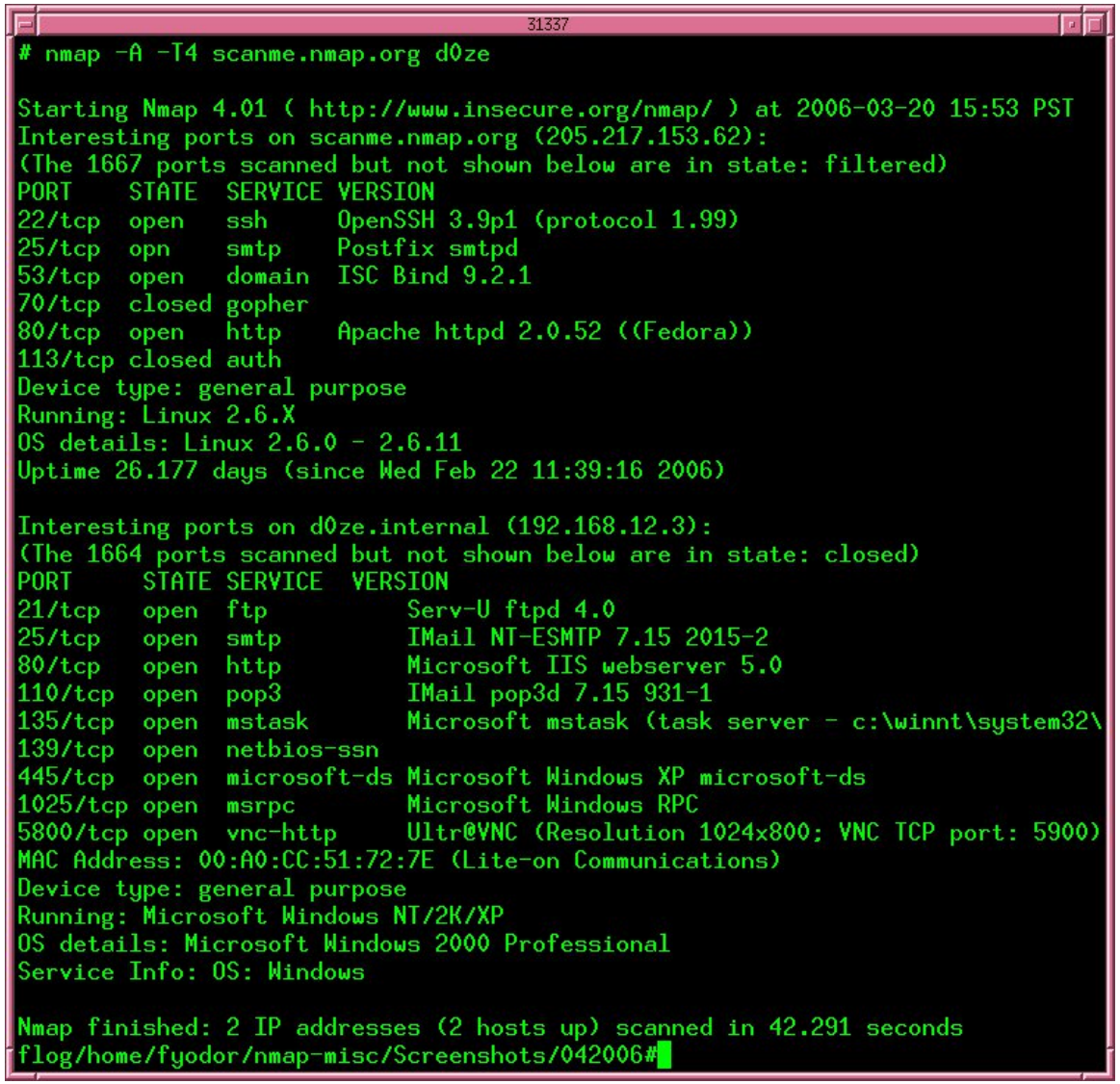
Step #5: Look for Attack Vectors

Identifying Attack Vectors

- Network: CVSS 3.1 = 8.1
  - Over the internet
- Network Adjacent: CVSS 3.1 = 7.6
  - Over a local network / intranet
- Local: CVSS 3.1 = 7.5
  - Access to the system or software
- Physical: CVSS 3.1 = 6.9
  - Can monkey with the hardware

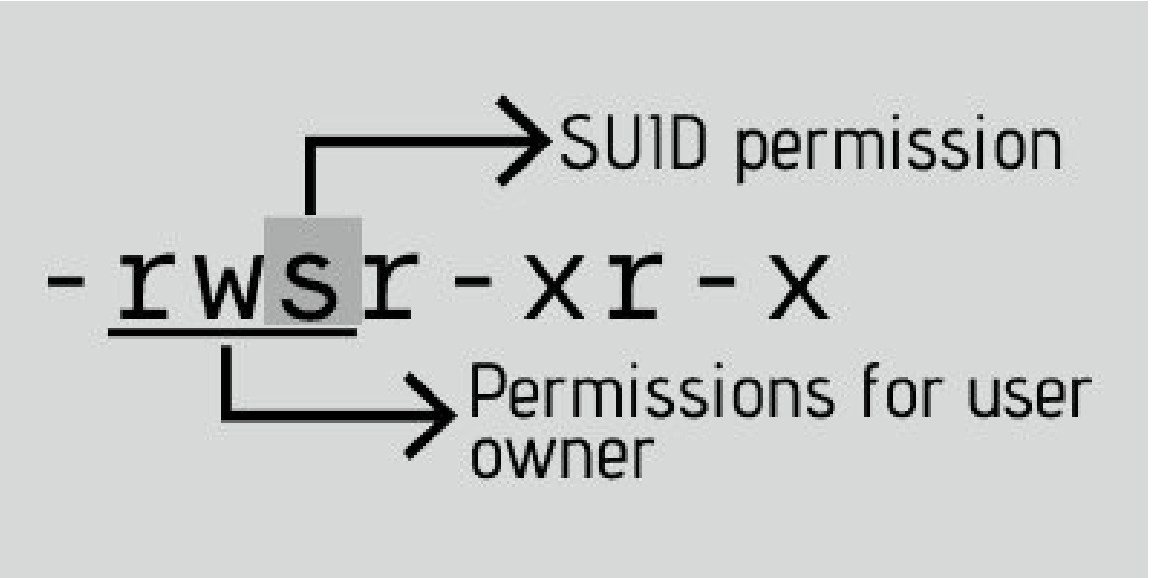
Identifying Attack Vectors: Network

- Network
  - Check network connectivity
  - Look for listening ports: NMAP
  - API access
- Network Adjacent
  - Same as above
  - Setup hotspots
  - LAN communication: IOT devices
- Other means of a Network:
  - Bluetooth
  - RF



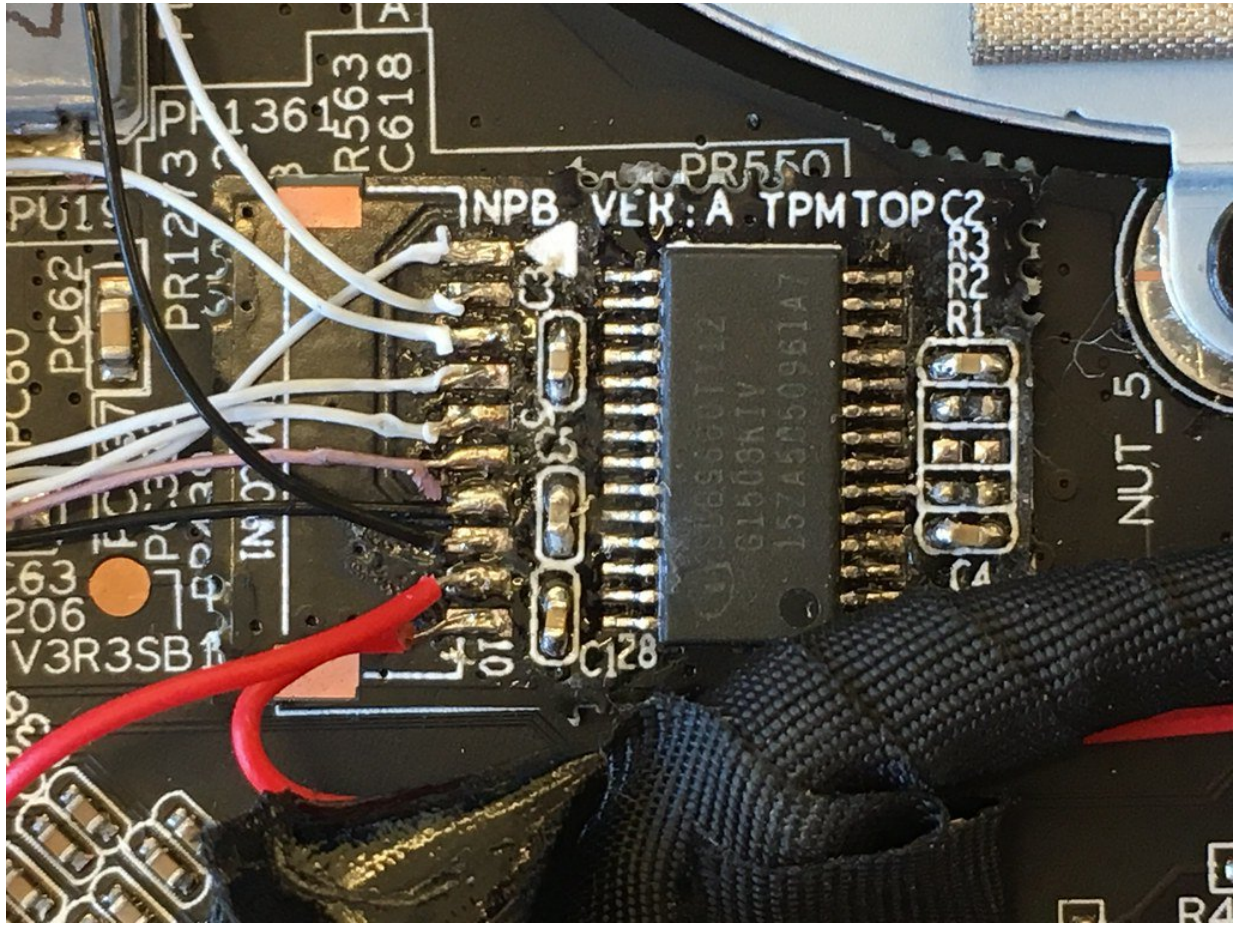
Identifying Attack Vectors: Local

- Check for privileged processes
- SUID / SGID permissions
- Config Permissions
- Output files



Identifying Attack Vectors: Physical

- Encryption
- Debug ports
- Inter-chip communications



Step #6: Looking for bugs

Work Smarter not Harder

- Try to do more than one thing at a time
  - Fuzzing
  - Reverse engineering

Work Smarter not Harder

- Try to do more than one thing at a time
  - Fuzzing
  - Reverse engineering

```
echo "Hello World"  | radamsa | xxd
00000000: 4865 6c6c 6fc2 a020 57f3 a081 806f 72e0  Hello.. W....or.
00000010: b980 6c6f 0a64                           ..lo.d
```

Work Smarter not Harder

- Try to do more than one thing at a time
  - Fuzzing
  - Reverse engineering

```
echo "Hello World"  | radamsa | xxd
00000000: 4865 6c6c 6fc2 a020 57f3 a081 806f 72e0  Hello.. W....or.
00000010: b980 6c6f 0a64                           ..lo.d
```

```
while : ; do A=$(echo "foobar" | radamsa); if ! curl https://172.16.0.195/backdoor.php -H "Accept-Encoding: $A" -H
"User-Agent: $A" -H "Referer: $A" -H "If-Modified-Since: $A" -H "Connection: $A" -H "Host: localhost" -H "Content-
Type: gzip" -H "Authorization: $A" -H "Accept-Language: $A" -k; then echo "Crash=$A" && break; fi; done
```

Sample Target: Dataprobe iBoot PDU

- Small to mid-sized data centers and on-prem
- All models use the same publicly available firmware

Start With Low-Hanging Fruit

How to quickly find Command Injections:

- Check custom binaries' strings for shell functions / scripts
- Format strings are a good sign
- Search for system(.*) calls
- Relate input to users supplied data
- Keep first tries simple

```
$ for i in $(\ls usr/bin/*d); do echo $i && strings $i | grep "\.sh "; done

usr/bin/consoled
    /root/scripts/update-ssh.sh %s %ld %s
usr/bin/ibootbar2d
  /root/scripts/update-snmp.sh %s %s %s %s
  /root/scripts/update-email.sh "%s" %ld "%s" "%s" '%s'
  /root/scripts/update-syslog.sh %s %s %s %ld %s
  /root/scripts/update-web-ssl.sh %s %ld %s %ld %s
  /root/scripts/update-fail2ban-config.sh %ld %ld %ld %s %s
  /root/scripts/update-fail2ban-ignoreip.sh %s %d %s
  /root/scripts/update-ntp.sh %s %s %d %s
  /root/scripts/update-timezone.sh "%s"
  /root/scripts/upgrade-download.sh "%s" "%s"
  /root/scripts/upgrade-install.sh "/tmp/%s"
usr/bin/usermanagerd
  /root/scripts/update-snmpv3-user.sh "%s" "%s" "%s" "%s" "%s"
```

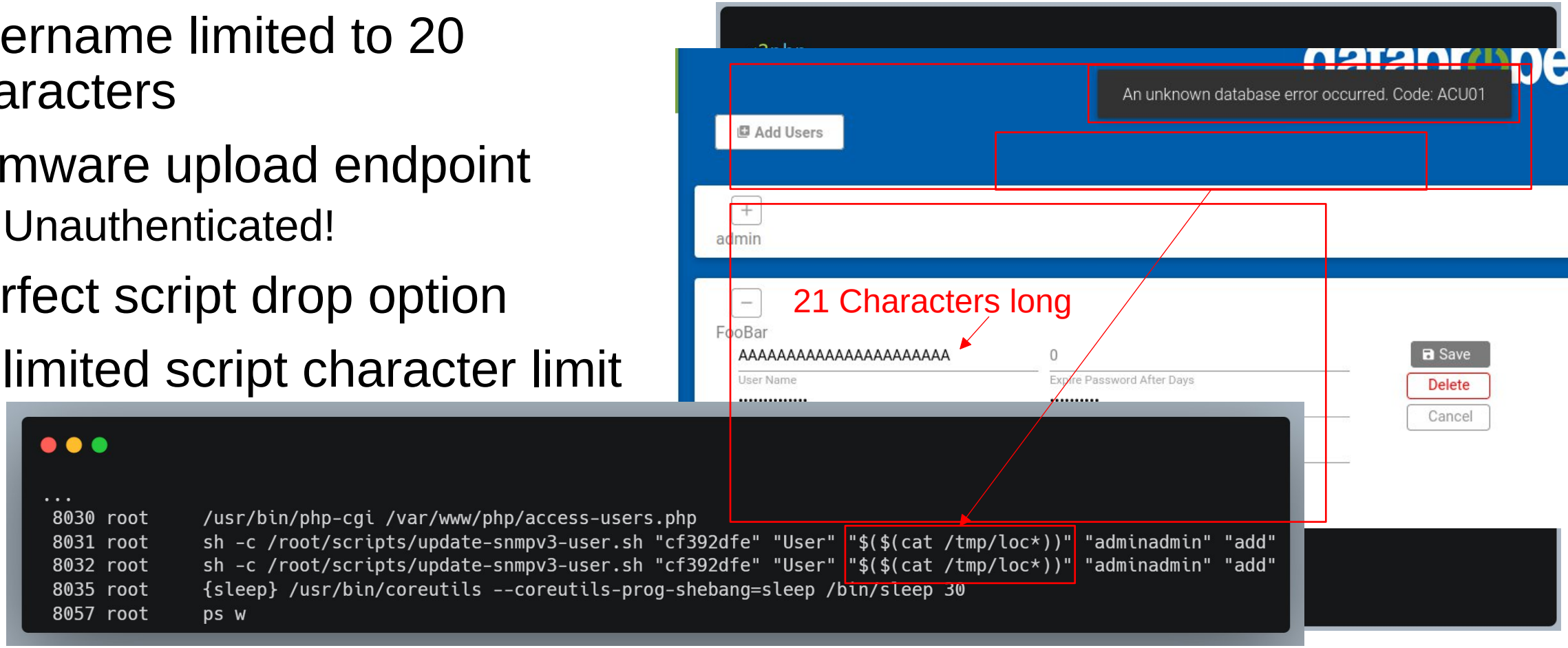
CVE-2023-3260: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

```
3648 root     sh -c /root/scripts/update-snmpv3-user.sh "a5327e05" "Admin" "$(sleep 20)" "adminadmin" "update"
3649 root     sh -c /root/scripts/update-snmpv3-user.sh "a5327e05" "Admin" "$(sleep 20)" "adminadmin" "update"
3650 root     {sleep} /usr/bin/coreutils --coreutils-prog-shebang=sleep /bin/sleep 20
```

controlled

Getting Creative

- Username limited to 20 characters
- Firmware upload endpoint
  - Unauthenticated!
- Perfect script drop option
- Unlimited script character limit



Step #7: Validation / Testing

Full System Emulation

All vulnerabilities were found via emulating the firmware!

1. Install QEMU and download a fresh Debian VM for 32-bit ARM (HF)

```
apt-get update
apt-get -y install qemu-system-arm wget
wget https://people.debian.org/~aurel32/qemu/armhf/debian_wheezy_armhf_standard.qcow2
wget https://people.debian.org/~aurel32/qemu/armhf/vmlinuz-3.2.0-4-vexpress
wget https://people.debian.org/~aurel32/qemu/armhf/initrd.img-3.2.0-4-vexpress
```

```
qemu-img resize debian_wheezy_armhf_standard.qcow2 32G
qemu-system-arm \
    -M vexpress-a9 \
    -kernel vmlinuz-3.2.0-4-vexpress \
    -initrd initrd.img-3.2.0-4-vexpress \
    -drive if=sd,file=debian_wheezy_armhf_standard.qcow2 \
    -append "root=/dev/mmcblk0p2" \
    -net user,hostfwd=tcp::8080-:80 \
    -net nic \
    -nographic
# Log in with root:root
```

2. Start the VM with internal port 80 mapped to an unused port on the host and log in with `root:root`

All commands tested on Ubuntu 23.04

Full System Emulation (cont.)

```
wget --no-check-certificate https://dataprobe.com/upgrade/iboot-pdu/iBoot-PDU-1.43.03312023.img
tar -xvf iBoot-PDU-1.43.03312023.img
md5sum iBoot-PDU-Root-1.43.03312023.img.gz | cut -d ' ' -f 1 | diff iBoot-PDU-Root-1.43.03312023.md5 -
gzip -cd iBoot-PDU-Root-1.43.03312023.img.gz > rootfs.ext2
```

3. Download the iBoot firmware and decompress the filesystem

```
mkdir rootfs
mount -o loop rootfs.ext2 rootfs
cd rootfs
mount -o bind /dev dev
mount -o bind /proc proc
mount -o bind /sys sys
chroot . /bin/bash
```

4. Prepare filesystem mountpoint and chroot into a Bash shell

5. Imitate the initialization sequence where possible.

```
mount -o remount,rw /
mkdir -p /dev/pts
mkdir -p /dev/shm
mount -a
hostname -F /etc/hostname
/etc/init.d/rcS
```

All core services are now up!

Login Page

localhost:8080/user-login.php

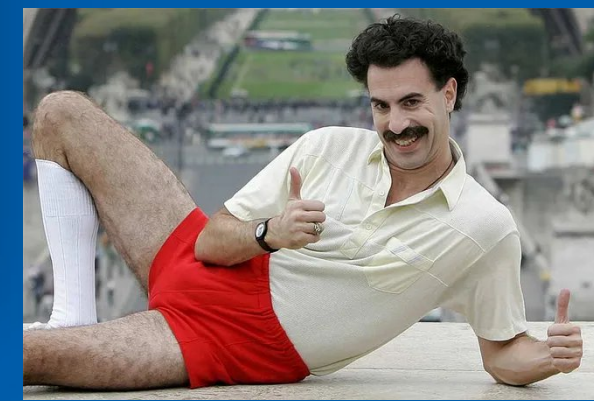
IBOOT-PDU

username

password

Remember me

LOGIN



Important Consideration

- When validating always attempt on fresh firmware
- Always validate on real unmodified hardware
- If you have access to multiple devices try on both
- Ensure you are testing against the correct SW / FW Version

Step #8: PWN’d... Now What?

- Bug bounties
  - Hacker one
  - Affected company
  - Get a CVE, sometimes.
- Responsible disclosure
  - Often allow for you to write a blog / publish details
  - Help the community learn
  - Harder to get a CVE

  https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/a-simple-guide-to-getting-cves-published/

- Who am I
- Target finding
  - Things you know
  - In your home
- OSINT
  - Find firmware
  - FCC
  - Previous CVES
- Normal Operation
- Gaining access to the software
  - Hardware Ports: JTAG, UART, Flash
  - Firmware Downloads
- Attack vector enumeration
  - Networking: Ports, APIs, Endpoints
  - RF connectivity
- Looking for bugs
  - Low hanging fruit
  - Command injections
  - File drops / Uploads
- Testing / Verification
  - Static tools: GDB
  - Emulation
- Responsible Disclosure