Reaching Beyond Boundaries

Out-of-Bounds Exploration in
Out-of-Band Management

kevingwn

Kevin Wang (@kevingwn_)

- Synology Security Incident Response Team
- Balsn CTF Team
- Top 3 at DEFCON CTF, HITCON CTF
- Speaker of Cybersec 2023
- Reported vulnerability to TP-Link, Netgear, Realtek, etc.

Outline

- Introduction
- Analysis
- Vulnerabilities
- Mitigation
- Conclusion

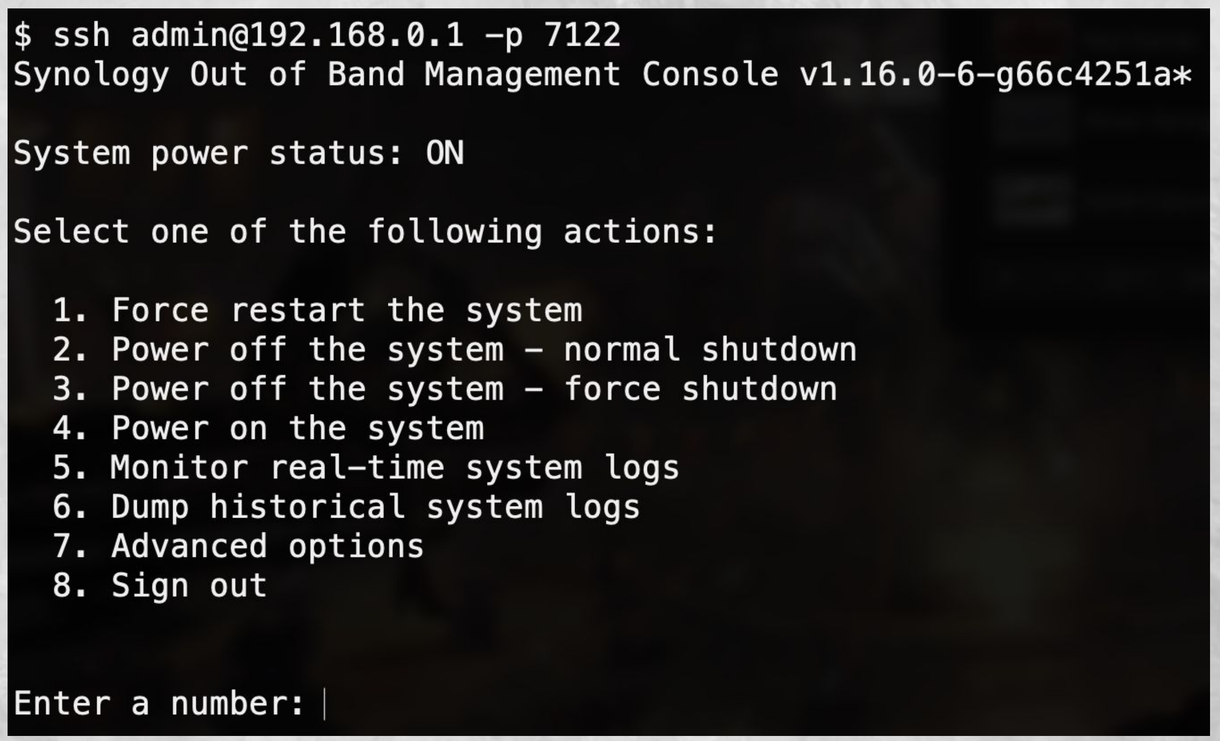
Introduction

Synology DS3622xs+

- Has an individual out-of-band port
- Isolated environment
- Our Target!

Out-of-Band Management

- SSH over separate channel
  - Its own processor, memory, …
- Custom shell after login
- Administration functionalities
  - Power on/off, monitor logs, …



Analysis

Initial Discoveries

- MIPS 32 bit
- Realtek self-defined firmware structure
- Can serve multiple SSH connection at a time
- Custom shell… must have a lot of command parsing

- Bingo! Using command `power AAAAAAAAAAAAAAAAAAAA...`
  - Stack Overflow ⇒ `system("ls")` ⇒ Post-auth RCE
- If we find an authentication vulnerability, it becomes Pre-auth RCE!

```
if (supported) {
    sprintf(buffer, EVENT_FMT_POWER_CTRL, argv[2]);
    syno_event_log(buffer);
    ret = CMD_COMPLETE;
} else if (discharge_protect) {
    sprintf(buffer, EVENT_FMT_IGNORE_POWER_CTRL, argv[2]);
    syno_event_log(buffer);
    ret = CMD_COMPLETE;
}
```

Easy Peasy!

- Bingo! Using command `power AAAAAAAAAAAAAAAAAAAA...`
  - Stack Overflow ⇒ `system("ls")` ⇒ Post-auth RCE
- If we find an authentication vulnerability, it becomes Pre-auth RCE!

```
if (supported) {
    sprintf(buffer, EVENT_FMT_POWER_CTRL, argv[2]);
    syno_event_log(buffer);
    ret = CMD_COMPLETE;
} else if (discharge_protect) {
    sprintf(buffer, EVENT_FMT_IGNORE_POWER_CTRL, argv[2]);
    syno_event_log(buffer);
    ret = CMD_COMPLETE;
}
```

Too Naive...

Command length > 2 is dropped, only quick commands (0-9)

```
/* limit commands by len, we support single digit quick commands ONLY */
if (len <= 0 || len > 2) {
    goto out;
}
```

Too Naive...

Com (0-9)

```
/* li                                    LY */
if (l
    g
}
```

「逃避可恥，但有用。」

Further Analysis

- Firmware is modified from µC/OS, there's no shell
- Objcopy binary, hard to reverse
- No debugger, memory layout is unknown
  - ASLR? Canary?

Remember our dream exploit chain?

- Authentication vulnerability ⇒ ~~post-auth stack overflow~~ ⇒ RCE
- A pre-auth vulnerability is needed anyway

SSH Protocol

- The only attack surface is SSH
- SSH is modified from Dropbear SSH
- What can be done prior to SSH authentication
  - Enter username and password? ~~Public key authentication?~~

```
$ ssh user@127.0.0.1
user@127.0.0.1's password:
Permission denied, please try again.
user@127.0.0.1's password:
```

SSH messages with number ≤ 60 can be performed before authentication

```
#define MAX_UNAUTH_PACKET_TYPE SSH_MSG_USERAUTH_PK_OK
#define SSH_MSG_USERAUTH_PK_OK                      60
```

```
/* Kindly the protocol authors gave all the preauth packets type values
 * less-than-or-equal-to 60 ( == MAX_UNAUTH_PACKET_TYPE ).
 * NOTE: if the protocol changes and new types are added, revisit this
 * assumption */
if ( !ses.authstate.authdone && type > MAX_UNAUTH_PACKET_TYPE ) {
        dropbear_exit("Received message %d before userauth", type);
}
```

```
/* message numbers */
#define SSH_MSG_DISCONNECT             1
#define SSH_MSG_IGNORE                 2
#define SSH_MSG_UNIMPLEMENTED          3
#define SSH_MSG_DEBUG                  4
#define SSH_MSG_SERVICE_REQUEST        5
#define SSH_MSG_SERVICE_ACCEPT         6
#define SSH_MSG_EXT_INFO               7
#define SSH_MSG_KEXINIT               20
#define SSH_MSG_NEWKEYS               21
#define SSH_MSG_KEXDH_INIT            30
#define SSH_MSG_KEXDH_REPLY           31

/* userauth message numbers */
#define SSH_MSG_USERAUTH_REQUEST          50
#define SSH_MSG_USERAUTH_FAILURE          51
#define SSH_MSG_USERAUTH_SUCCESS          52
#define SSH_MSG_USERAUTH_BANNER           53

/* packets 60-79 are method-specific, aren't one-one mapping */
#define SSH_MSG_USERAUTH_SPECIFIC_60   60

#define SSH_MSG_USERAUTH_PASSWD_CHANGEREQ   60

#define SSH_MSG_USERAUTH_PK_OK                  60
```

```
/* keyboard interactive auth */
#define SSH_MSG_USERAUTH_INFO_REQUEST          60
#define SSH_MSG_USERAUTH_INFO_RESPONSE         61


/* If adding numbers here, check MAX_UNAUTH_PACKET_TYPE in process-packet.c
 * is still valid */

/* connect message numbers */
#define SSH_MSG_GLOBAL_REQUEST                 80
#define SSH_MSG_REQUEST_SUCCESS                81
#define SSH_MSG_REQUEST_FAILURE                82
#define SSH_MSG_CHANNEL_OPEN                   90
#define SSH_MSG_CHANNEL_OPEN_CONFIRMATION      91
#define SSH_MSG_CHANNEL_OPEN_FAILURE           92
#define SSH_MSG_CHANNEL_WINDOW_ADJUST          93
#define SSH_MSG_CHANNEL_DATA                   94
#define SSH_MSG_CHANNEL_EXTENDED_DATA          95
#define SSH_MSG_CHANNEL_EOF                    96
#define SSH_MSG_CHANNEL_CLOSE                  97
#define SSH_MSG_CHANNEL_REQUEST                98
#define SSH_MSG_CHANNEL_SUCCESS                99
#define SSH_MSG_CHANNEL_FAILURE                100
```

Vulnerabilities

ssh_process

- After key exchange, `cipher_mode` will be set and all packets will be encrypted

```
int ssh_process(struct sshsession *ses, INT8U *rxbuf, INT16U buf_len)
{
    int ret = 0;
    ses->exitflag = 0;
    INT32U len = 0, msg_len;
    INT8U *msg_ptr;
    INT16U macsize;

    LWIP_DEBUGF(SSH_DEBUG, ("SSH Process %d bytes\n", buf_len));

    /* read the version string */
    if (!ses->remoteident)
        ret = read_session_identification(ses, rxbuf, buf_len);
    else {
        while (len < buf_len && 0 == ret) {
            if (ses->keys->recv.algo_crypt->cipher_mode !=
                MBEDTLS_CIPHER_NONE) {
                msg_ptr = decrypt_packet(ses, rxbuf + len, &msg_len, &macsize);

                /* in case of error, just skip this packet */
                if (!msg_ptr)
                    break;
            } else {
                msg_ptr = rxbuf + len;
                msg_len = get_msg_len(msg_ptr);
                macsize = 0;
            }

            ret = process_packet(ses, msg_ptr, msg_len);

            len += msg_len + macsize;
        }
    }

    return ret;
}
```

decrypt_packet

- The format of ssh msg:

  `|len1|msg1|len2|msg2|...|`

- `buf` is a malloced buffer of size 1500

```
/* Decrypt the first block to get packet length first */
mbedtls_cipher_update(&ses->recv_ctx, buf, blocksize, ses->cipherbuf,
                      &declen);

/* 1st block, getting the whole packet length */
packet_len = get_msg_len(ses->cipherbuf);

/* Decrypt remaining data in the same packet */
if (packet_len > blocksize) {
    left = packet_len - blocksize;
    LWIP_DEBUGF(SSH_DEBUG, ("Decrypt remaining %d bytes\n", left));
    mbedtls_cipher_update(&ses->recv_ctx, buf + blocksize, left,
                          ses->cipherbuf + blocksize, &declen);
    mbedtls_cipher_finish(&ses->recv_ctx,
                          ses->cipherbuf + blocksize + declen, &olen);
    declen += olen;
}
```

Heap Overflow

- `packet_len` is controlled by user

  `|len1|msg1|len2|msg2|...|`

- If given > 1500, we can overflow `buf`

```
/* Decrypt the first block to get packet length first */
mbedtls_cipher_update(&ses->recv_ctx, buf, blocksize, ses->cipherbuf,
                      &declen);

/* 1st block, getting the whole packet length */
packet_len = get_msg_len(ses->cipherbuf);

/* Decrypt remaining data in the same packet */
if (packet_len > blocksize) {
    left = packet_len - blocksize;
    LWIP_DEBUGF(SSH_DEBUG, ("Decrypt remaining %d bytes\n", left));
    mbedtls_cipher_update(&ses->recv_ctx, buf + blocksize, left,
                          ses->cipherbuf + blocksize, &declen);
    mbedtls_cipher_finish(&ses->recv_ctx,
                          ses->cipherbuf + blocksize + declen, &olen);
    declen += olen;
}
```

Wait a minute…

1500 is a relatively small value, wouldn’t the vulnerability be easily triggered even on normal situations?

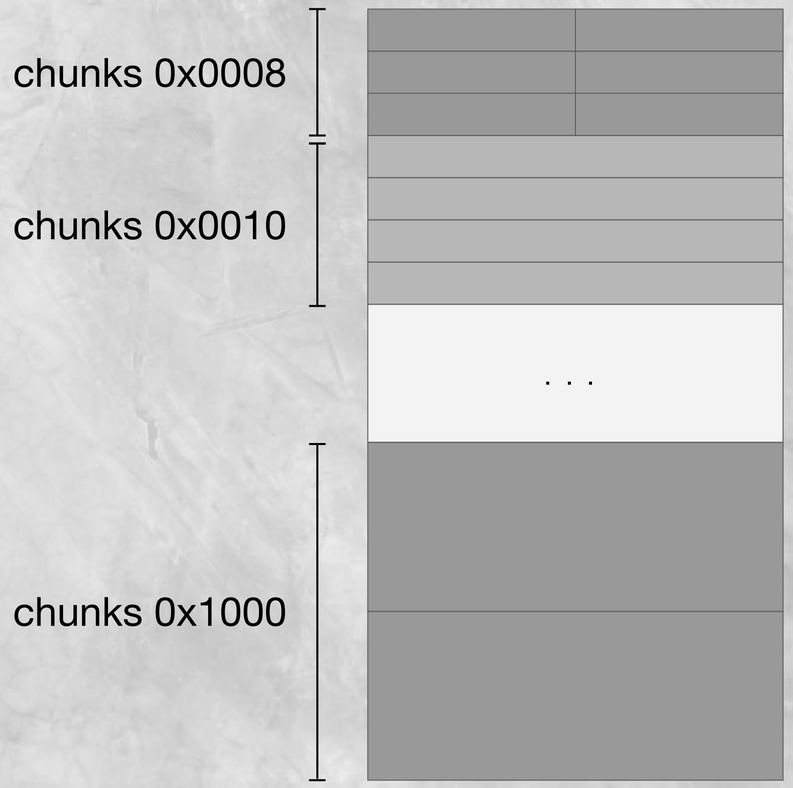
Actually, if we enter 2000 "A", the heap overflow will not be triggered

- During `SSH_MSG_CHANNEL_OPEN`, server and client will agree on `remote_maxpacket`, and the following messages will not violate it

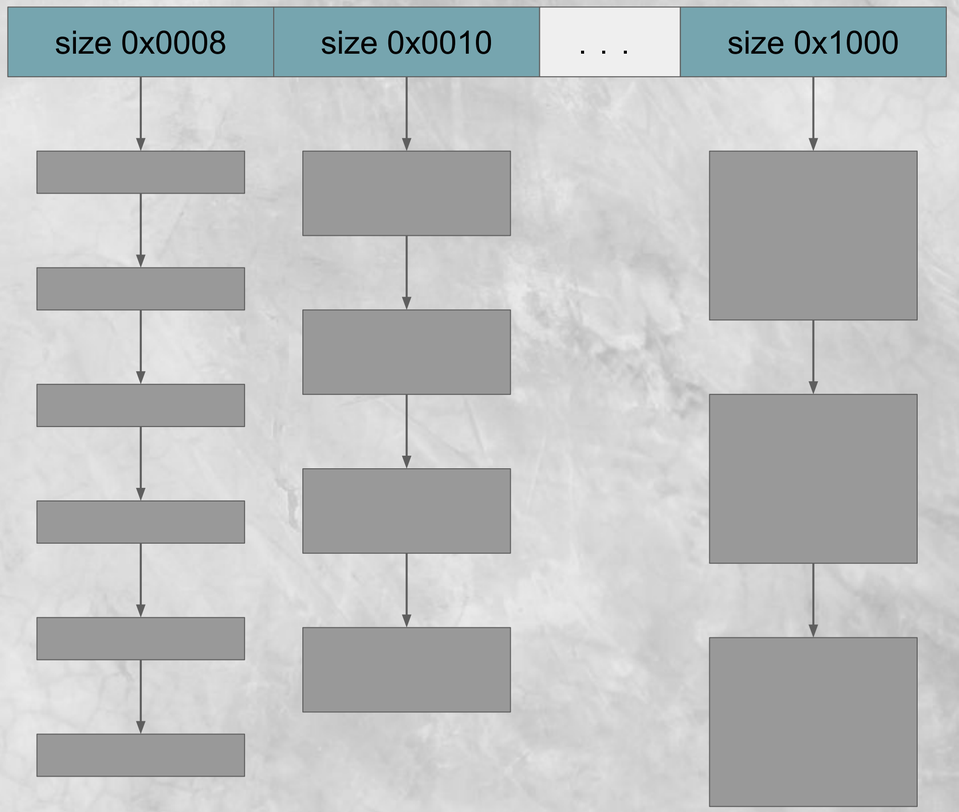
```
/* Enqueue packet for buffered data. */
if (len > c->remote_window)
        len = c->remote_window;
if (len > c->remote_maxpacket)
        len = c->remote_maxpacket;
if (len == 0)
        return;
if ((r = sshpkt_start(ssh, SSH2_MSG_CHANNEL_DATA)) != 0 ||
    (r = sshpkt_put_u32(ssh, c->remote_id)) != 0 ||
    (r = sshpkt_put_string(ssh, sshbuf_ptr(c->input), len)) != 0 ||
    (r = sshpkt_send(ssh)) != 0)
        fatal_fr(r, "channel %i: send data", c->self);
if ((r = sshbuf_consume(c->input, len)) != 0)
        fatal_fr(r, "channel %i: consume", c->self);
c->remote_window -= len;
```

µC/OS Memory Management

- There’s no ASLR ٩(๑•̀ω•́๑)۶
- A fixed large segment of memory is divided into chunks of 10 different sizes:
  - 0x0008
  - 0x0010
  - 0x0020
  - 0x0040
  - 0x0080
  - 0x0100
  - 0x0200
  - 0x0600
  - 0x0800
  - 0x1000

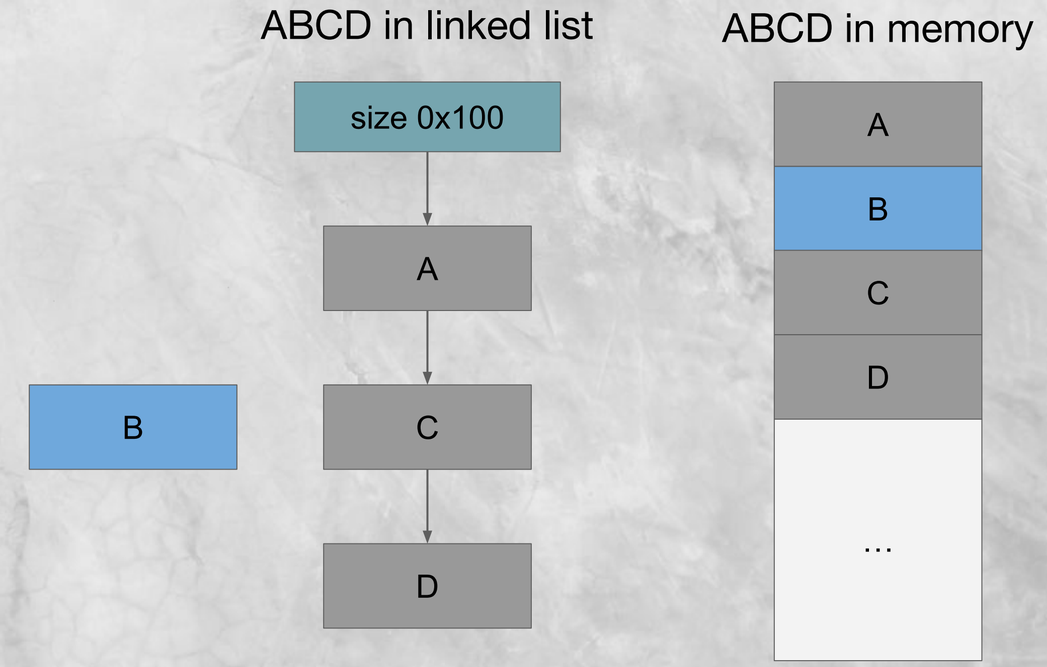


- Chunks of the same are stored in a linked list
- `malloc(size)`:
  - The first chunk > `size` is served
  - If no chunks left ⇒ error
- `free(ptr)`:
  - Return to corresponding list according to its address
- It’s like tcache without checks!



Tcache Poisoning

- Chunks A, B, C, D are continuous
- Chunk B is in use



Tcache Poisoning

- Heap overflow on B will overwrite C’s fd
  - Let C point to anywhere we want
- If we malloc 0x100 3 times
  - "anywhere" will be allocated
  - Arbitrary memory write

decrypt_packet Exploitation

- `buf` size is 1500 (0x5dc) ⇒ chunk 0x600
- If we can malloc chunk 0x600 3 times
  - heap overflow ⇒ tcache poisoning ⇒ arbitrary write

```
/* Decrypt the first block to get packet length first */
mbedtls_cipher_update(&ses->recv_ctx, buf, blocksize, ses->cipherbuf,
                      &declen);

/* 1st block, getting the whole packet length */
packet_len = get_msg_len(ses->cipherbuf);

/* Decrypt remaining data in the same packet */
if (packet_len > blocksize) {
    left = packet_len - blocksize;
    LWIP_DEBUGF(SSH_DEBUG, ("Decrypt remaining %d bytes\n", left));
    mbedtls_cipher_update(&ses->recv_ctx, buf + blocksize, left,
                          ses->cipherbuf + blocksize, &declen);
    mbedtls_cipher_finish(&ses->recv_ctx,
                          ses->cipherbuf + blocksize + declen, &olen);
    declen += olen;
}
```

- Couldn’t find anywhere mallocing chunk 0x600 3 times… ﴾д﴿
- When logging in, username/password uses 511+1 (0x200) 3 times

```
INT8U *get_string(struct sshsession *ses, INT8U *ptr, INT32U *len)
{
    INT32U strlen;
    INT8U *str = NULL;

    memcpy(&strlen, ptr, sizeof(int));

    strlen = ntohl(strlen);                                  ( ´•ω•` )

    *len = strlen;     512, 1 byte away from chunk 0x600…

    if (strlen < MAX_STRING_LEN)
        str = malloc(strlen + 1);
    else
        ses->exitflag = -1;

    if (str) {
        memcpy(str, ptr + sizeof(int), strlen);
        str[strlen] = '\0';
    }

    return str;
}
```

「排 heap 就是浪費時間，不如再找一個洞」

- Angelboy 10.17.2022

ADVERSARY AND HARMONY, THE EVOLUTION OF AI SECURITY

「排 heap 就是浪費時間，不如再找一個洞」

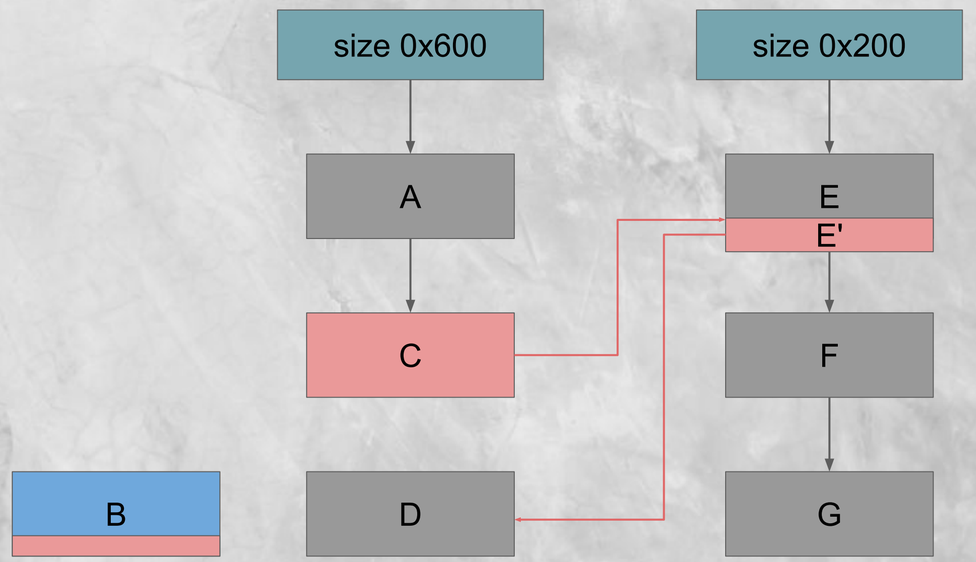
- Angelboy 10.17.2022

Can we "transfer" the poisoned chunk?

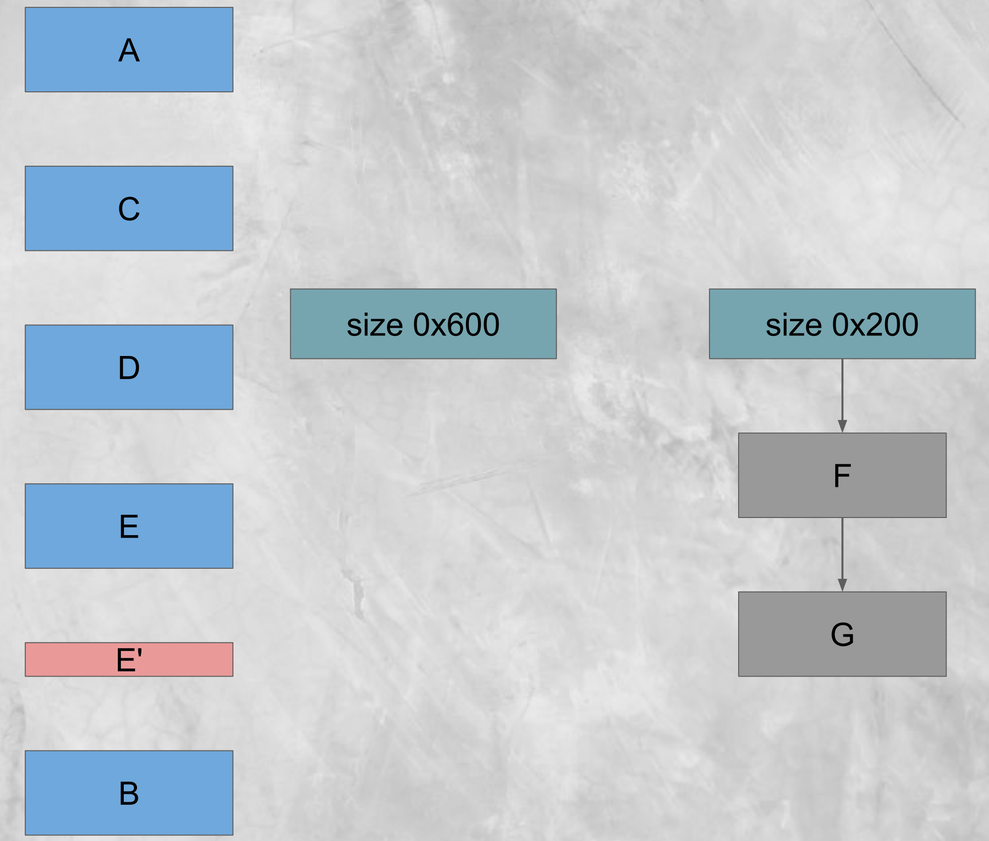
❌ Chunk 0x600 overflow ⇒ malloc 0x600 3 times to trigger

✔️ Chunk 0x600 overflow ⇒ malloc 0x200 3 times to trigger

- Overwrite C’s fd so that it points to the middle of E
- Use E remanent value to point E' back to D
  - Username/password uses chunk 0x200



- If we establish a second connection now
- Socket, mbedtls will allocate some buffers
  - A, C, E' (the middle of E), D will be allocated
  - E will also be allocated

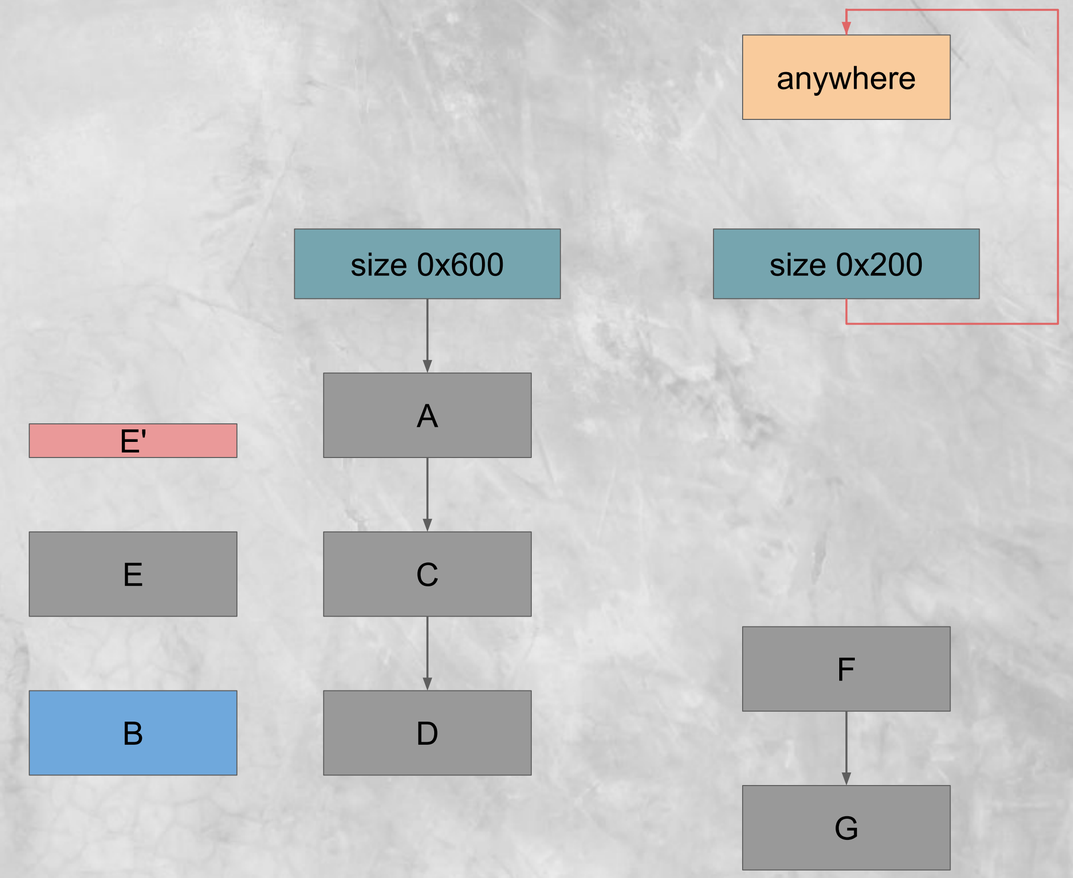


- Disconnect the second connection
  - A, C, E' (the middle of E), D will be freed
  - E' will be freed before E
  - E' will be put on linked list 0x200

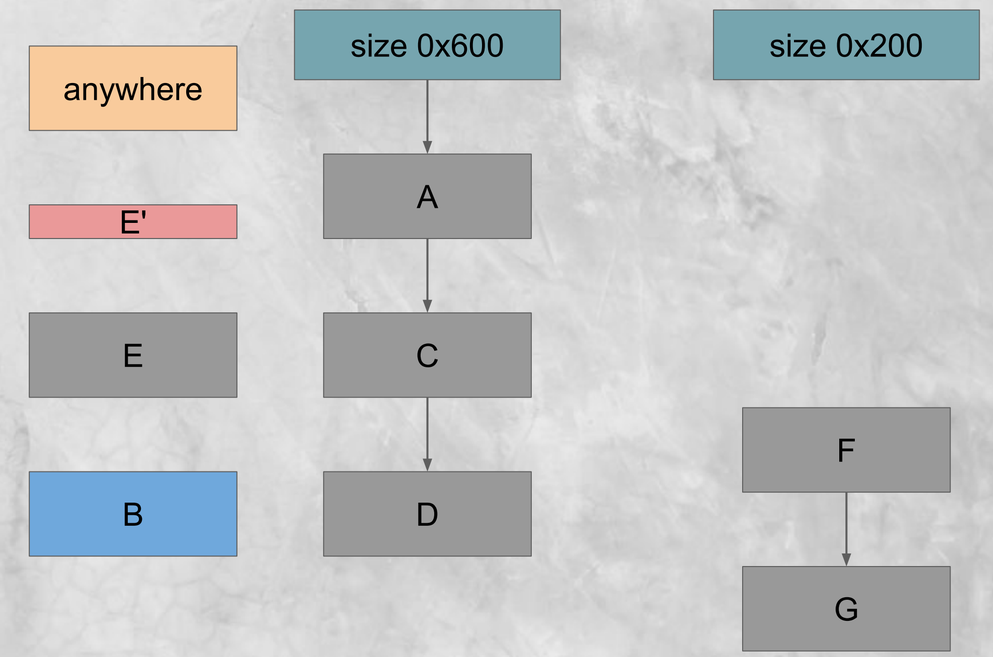
- Establish a third connection
- Log in allocates chunks 0x200 3 times

- Establish a third connection
- Log in allocates chunks 0x200 3 times
  - #1: allocates E, overwrite the fd of E'

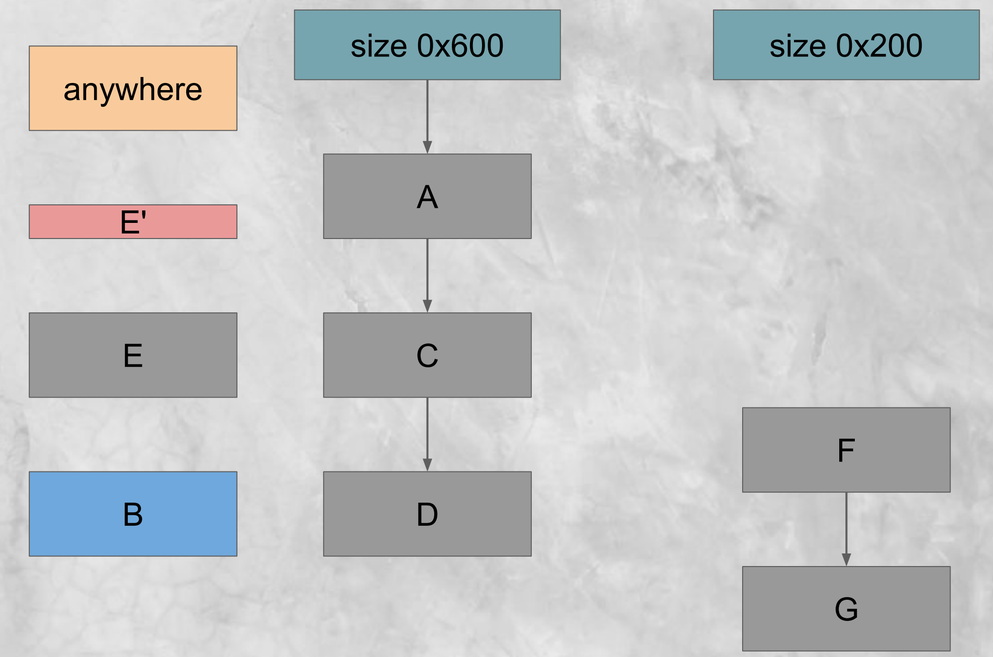
- Establish a third connection
- Log in allocates chunks 0x200 3 times
  - #1: allocates E, overwrite the fd of E'
  - #2: allocates E'



- Establish a third connection
- Log in allocates chunks 0x200 3 times
  - #1: allocates E, overwrite the fd of E'
  - #2: allocates E'
  - #3: allocates "anywhere" ⇒ arbitrary write



- Establish a third connection
- Log in allocates chunks 0x200 3 times
  - #1: allocates E, overwrite the fd of E'
  - #2: allocates E'
  - #3: allocates "anywhere" ⇒ arbitrary write
- No ASLR, no NX ⇒ return to shellcode



Arbitrary Code Execution, but…

- Firmware is modified from µC/OS, there’s no shell
- In an isolated environment

Changing Scope

- OOB is capable of resetting NAS
  - Mode 2 Reset: preserve data + reinstall ⇒ Get full admin privilege on NAS
  - Code execution in OOB (network socket) ⇒ We're in intranet when resetting
- CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H 10.0 (Critical)

Advanced functions	
7. **Advanced options**	
1) Dump historical out-of-band management console logs	Out-of-band (OOB) management console dumps all historical **OOB** Management system logs.
2) Reset the system (does not impact the data)	Resetting mode 2 on Synology NAS. For detailed reset information, please refer to this article.
3) Back to main menu	

alloc_session

- `session_pool` is a global variable
- Store structures of SSH session

```
static void *alloc_session()
{
    int i;
    for (i = 0; i < ARRAY_SIZE(session_pool); i++) {
        if (session_pool[i].sock == 0) {
            return &session_pool[i];
        }
    }
    LWIP_DEBUGF(SSH_DEBUG, ("Failed to alloc session [NULL]\n"));
    return NULL;
}
```

Race Condition

- OOB has 2 threads processing SSH
- No lock when allocating session

```
static void *alloc_session()
{
    int i;
    for (i = 0; i < ARRAY_SIZE(session_pool); i++) {
        if (session_pool[i].sock == 0) {
            return &session_pool[i];
        }
    }
    LWIP_DEBUGF(SSH_DEBUG, ("Failed to alloc session [NULL]\n"));
    return NULL;
}
```

What happens if we raced the same session?

- Raced an admin session ⇒ log in OOB
- Need user interaction + precise timing
  - CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:N/I:H/A:H 6.8 (Medium)
  - Too mediocre and uninteresting

What if we race our own session?

- Control the timing of both connections

  ⇒ success rate ↑ ↑ ↑

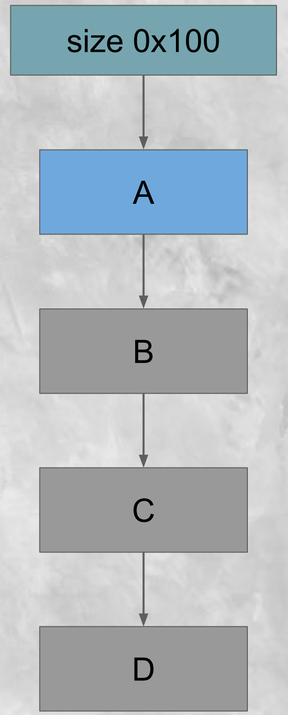
- The buffers of 2 sessions will be the same
  - When disconnecting, they will free the same buffer
  - Double free!

```
void common_session_close(struct sshsession *ses)
{
    LWIP_DEBUGF(SSH_DEBUG, ("session close\n"));

    if (ses->keys)
        free(ses->keys);

    if (ses->newkeys)
        free(ses->newkeys);

    if (ses->remoteident)
        free(ses->remoteident);

    if (ses->writebuf)
        free(ses->writebuf);
```

Tcache Dup Attack

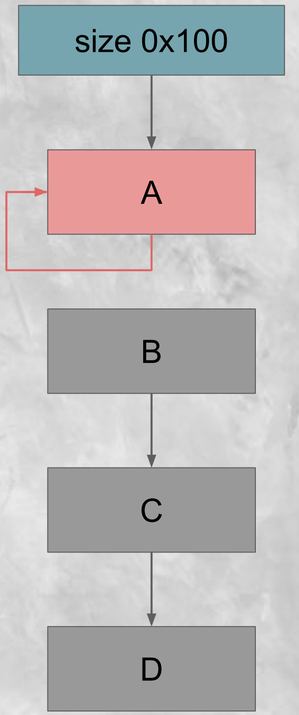
Tcache Dup Attack

- `free(A)`



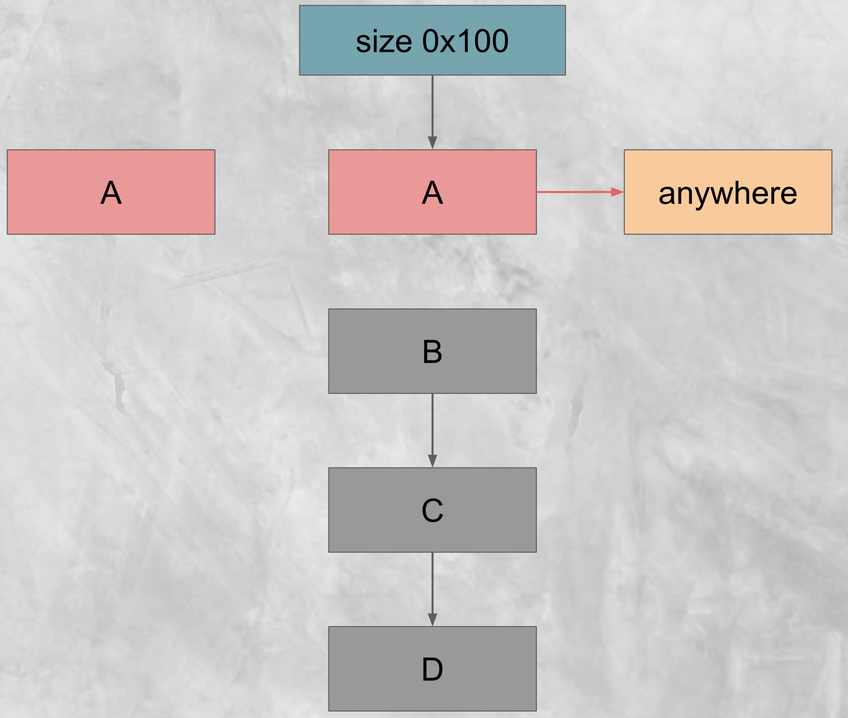
Tcache Dup Attack

- `free(A)`
- `free(A)`



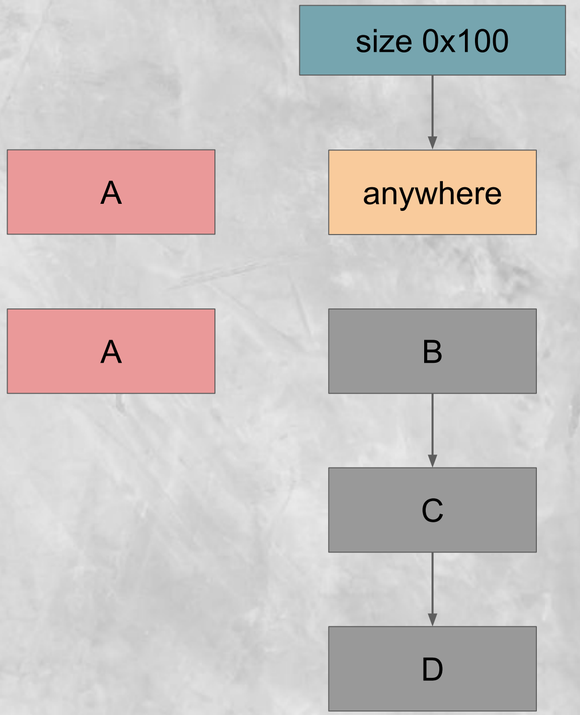
Tcache Dup Attack

- `free(A)`
- `free(A)`
- `malloc(0x100)` gets A, overwrite A's fd



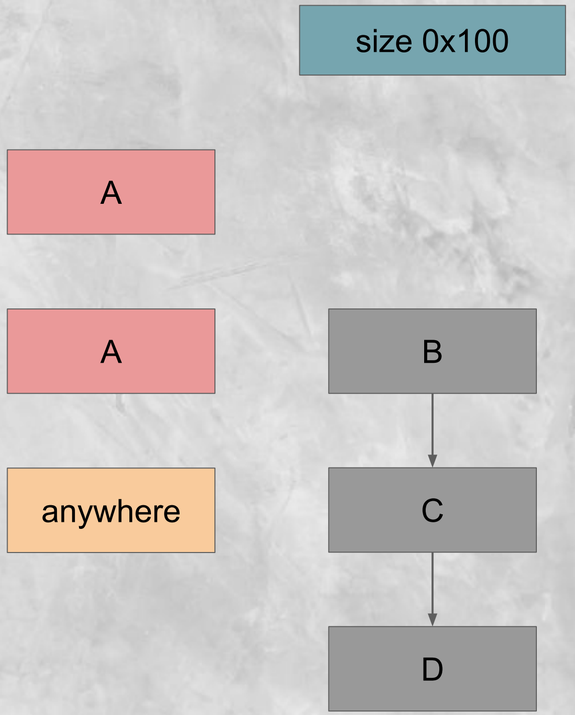
Tcache Dup Attack

- `free(A)`
- `free(A)`
- `malloc(0x100)` gets A, overwrite A's fd
- `malloc(0x100)` gets A again



Tcache Dup Attack

- `free(A)`
- `free(A)`
- `malloc(0x100)` gets A, overwrite A's fd
- `malloc(0x100)` gets A again
- `malloc(0x100)` gets "anywhere"
  - Arbitrary memory write



Changing Scope

- Write stack ⇒ return to shellcode ⇒ reset NAS ⇒ get admin RCE
  - ~~CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:N/I:H/A:H 6.8 (Medium)~~
  - CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H 10.0 (Critical)

Changing Scope

- Write stack ⇒ return to shellcode ⇒ reset NAS ⇒ get admin RCE
  - ~~CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:N/I:H/A:H 6.8 (Medium)~~
  - CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H 10.0 (Critical)

Everything going too smoothly…?

Session is a complex structure, no segfault during exploitation?

Everything going too smoothly…?

Session is a complex structure, no segfault during exploitation?

- Dereferencing an invalid address results in 0xdeadbeef
  - Dereferencing 0xdeadbeef results in 0 `[invalid address] → 0xdeadbeef → 0 → 0xdeadbeef → …`
  - Moreover, every address is writable `[invalid address] → 0xdeadbeef → 0x1234 → 0xdeadbeef → …`

Everything going too smoothly…?

Session is a complex structure, no segfault during exploitation?

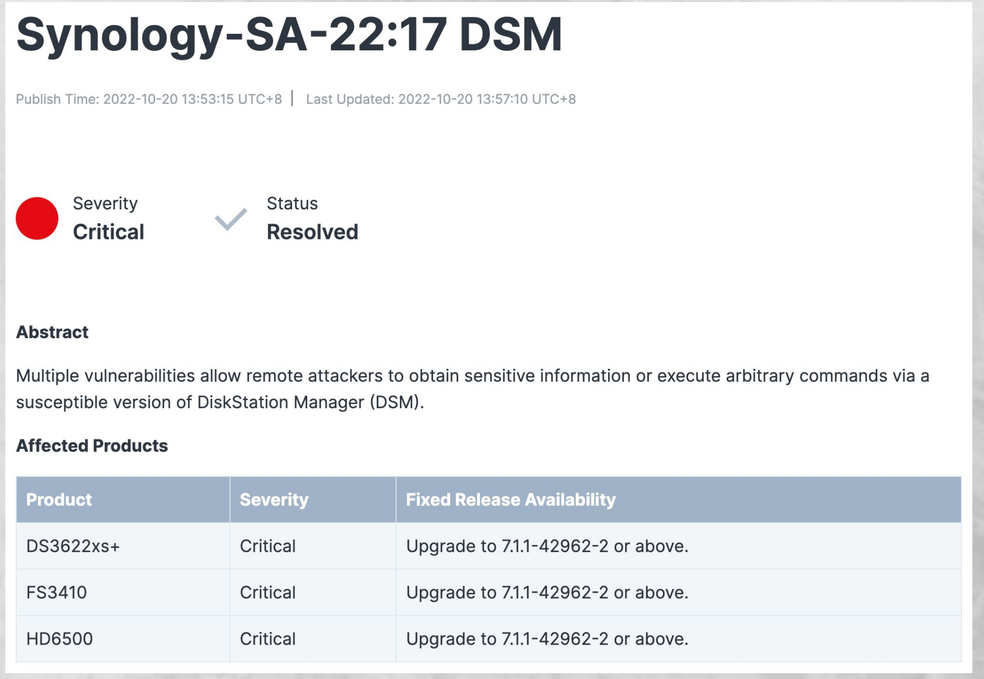
- Dereferencing an invalid address results in 0xdeadbeef
  - Dereferencing 0xdeadbeef results in 0 `[invalid addr`
  - Moreover, every address is writable `[invalid addr`
  - Windows, Linux, macOS should all implement this :)

Mitigation

Mitigation

- For DS3622xs+, FS3410 and HD6500:

  Update to DSM 7.1.1-42962-2 or above



Synology-SA-22:17 DSM

Publish Time: 2022-10-20 13:53:15 UTC+8 | Last Updated: 2022-10-20 13:57:10 UTC+8

Severity
Critical

Status
Resolved

Abstract

Multiple vulnerabilities allow remote attackers to obtain sensitive information or execute arbitrary commands via a susceptible version of DiskStation Manager (DSM).

Affected Products

Product	Severity	Fixed Release Availability
DS3622xs+	Critical	Upgrade to 7.1.1-42962-2 or above.
FS3410	Critical	Upgrade to 7.1.1-42962-2 or above.
HD6500	Critical	Upgrade to 7.1.1-42962-2 or above.

Conclusion

Conclusion

- New attack surface for specific NAS
- Firmwares are troublesome to analyze, but also rewarding
  - Due to the lack of security measurements (ASLR, NX, canary, …)
- µC/OS is the best OS

ADVERSARY AND HARMONY,
THE EVOLUTION OF
AI SECURITY

Thank You!

𝕏 @kevingwn_

kevingwnn@gmail.com