How to backup and pwn using Time Machine

Nguyen Hoang Thach (@hi_im_d4rkn3ss)

STAR Labs SG Pte. Ltd.

About me

- Nguyen Hoang Thach (@hi_im_d4rkn3ss)
- Security Researcher at STAR Labs SG Pte. Ltd.
- Focusing on Virtual Machine / Android / IOT bug hunting.
- Participated in Pwn2Own Tokyo 2020 and Pwn2wn Austin 2021 in Router, NAS and Mobile phone category, and Pwn2Own Vancouver 2022 in the Virtual Machine category

Agenda

1. File service in NAS devices
2. Bug in processing `appl` file
3. Bug in processing file ‘s `xattr`
4. Summary

File service in NAS devices

Nas devices

- Network-attached storage (NAS) device is a data storage device that connects to and is accessed through a network, instead of connecting directly to a computer.
- In recent year, ZDI added some NAS devices to list target (WD, Synology NAS) in their Pwn2Own contest
- Last year, I participated in Pwn2Own, I found 4 bugs and successful pwned 3 different NAS devices: WD Home Cloud NAS (release version), WD Home Cloud NAS (beta version) and WD Pro PR4100 NAS.
- Attack surface: File service

File service in NAS devices

Architecture

- WD Home Cloud (release version)

  Arm 32bit little endian
- WD Home Cloud (beta version)

  Arm 32bit little endian
- WD Pro PR4100

  Arch64 little endian

File service in NAS devices

File Service

- Usually, NAS devices implemented at least one File Service to support file sharing, file printing, file backup.
- I will focus on 2 popular file services: *netatalk afpd* and *samba smbd*
- WD Home Cloud (release version) and WD Pro Pr4100 implement both *afpd* and *smbd*
- WD Home Cloud (beta version) implement *smbd*
- Version:
  - Netatalk afpd : v3.1.12
  - Samba smbd : v4.9.5

File service in NAS devices

Configuration

- Usually, in NAS devices, at least, there is one public share folder.
- Some features also are implemented, for example: *Time Machine Backup*

-> extend the attack surface

File service in NAS devices

afpd configuration

```
[ Global ]
uam list = uams_guest.so,uams_dhx2_passwd.so
save password = no
unix charset = UTF8
use sendfile = yes
zeroconf = no
guest account = nobody
vol dbpath = /data/wd/diskVolume0/backups/.systemfile/netatalk/CNID
...
[ TimeMachineBackup ]
path = /data/wd/diskVolume0/backups/timemachine
ea = auto
...
```

- `uams_guest.so` is declared in `uam list`, it accepts guest authentication.
- *TimeMachineBackup* is a public share folder

File service in NAS devices

smbd configuration

- *TimeMachineBackup* is a public share folder
- `guest ok = yes` is declared, it allows guest authentication
- The `vfs objects` list contains 3 modules: *catia*, *fruit*, *streams_xattr*
- *vfs_fruit*: Enhanced OS X and Netatalk interoperability

```
[global]

...

[TimeMachineBackup]
path = /data/wd/diskVolume0/backups/timemachine
browseable = yes
public = yes
available = yes
oplocks = yes
follow symlinks = yes
map archive = no
guest ok = yes
writable = yes
vfs objects = catia fruit streams_xattr
durable handles = yes
kernel oplocks = no
kernel share modes = no
posix locking = no
inherit acls = yes
strict sync = yes
fruit:time machine = yes
fruit:time machine max size = 0M
```

File service in NAS devices

Mitigation

- afpd (WD Home)

```
$ checksec ./WDHome/afpd
[*] '/tmp/WDHome/afpd'
    Arch:     arm-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
    RUNPATH:  b'/system/lib:/space/jenkins/workspace/netatalk/db/../rootfs/
```

- afpd (WD Pro)

```
$ checksec ./WDPro/afpd
[*] '/tmp/WDPro/afpd'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

- smbd (WD Home Beta)

```
$ checksec ./WDHomeBeta/smbd
[*] '/tmp/WDHomeBeta/smbd'
    Arch:     aarch64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
    RUNPATH:  b'/usr/lib/aarch64-linux-gnu/samba'
    FORTIFY:  Enabled
```

Bug in processing `appl` file

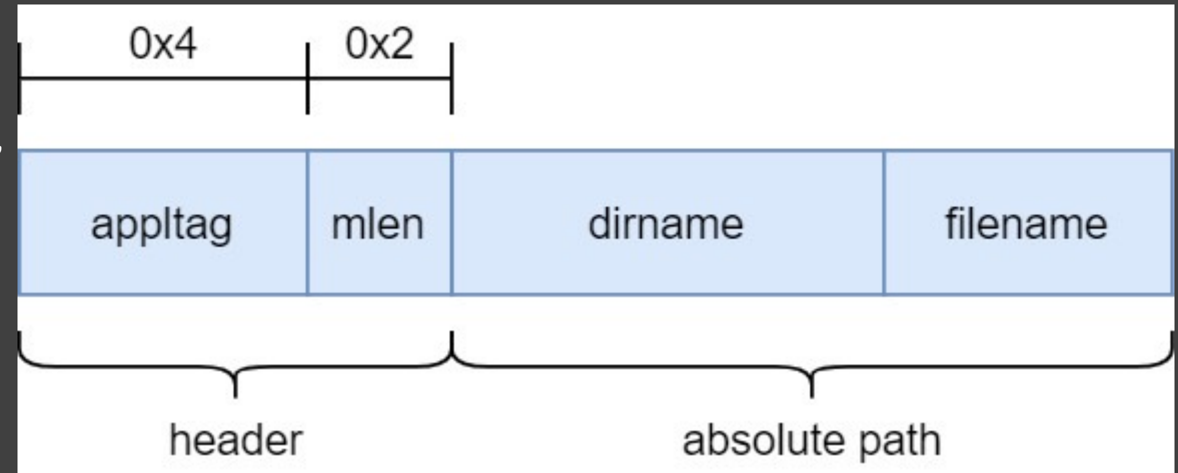
Target

- 2 bugs in *afpd*
- Bugs were used to exploit the WD Home Cloud (release version)

Bug in processing `appl` file

Background

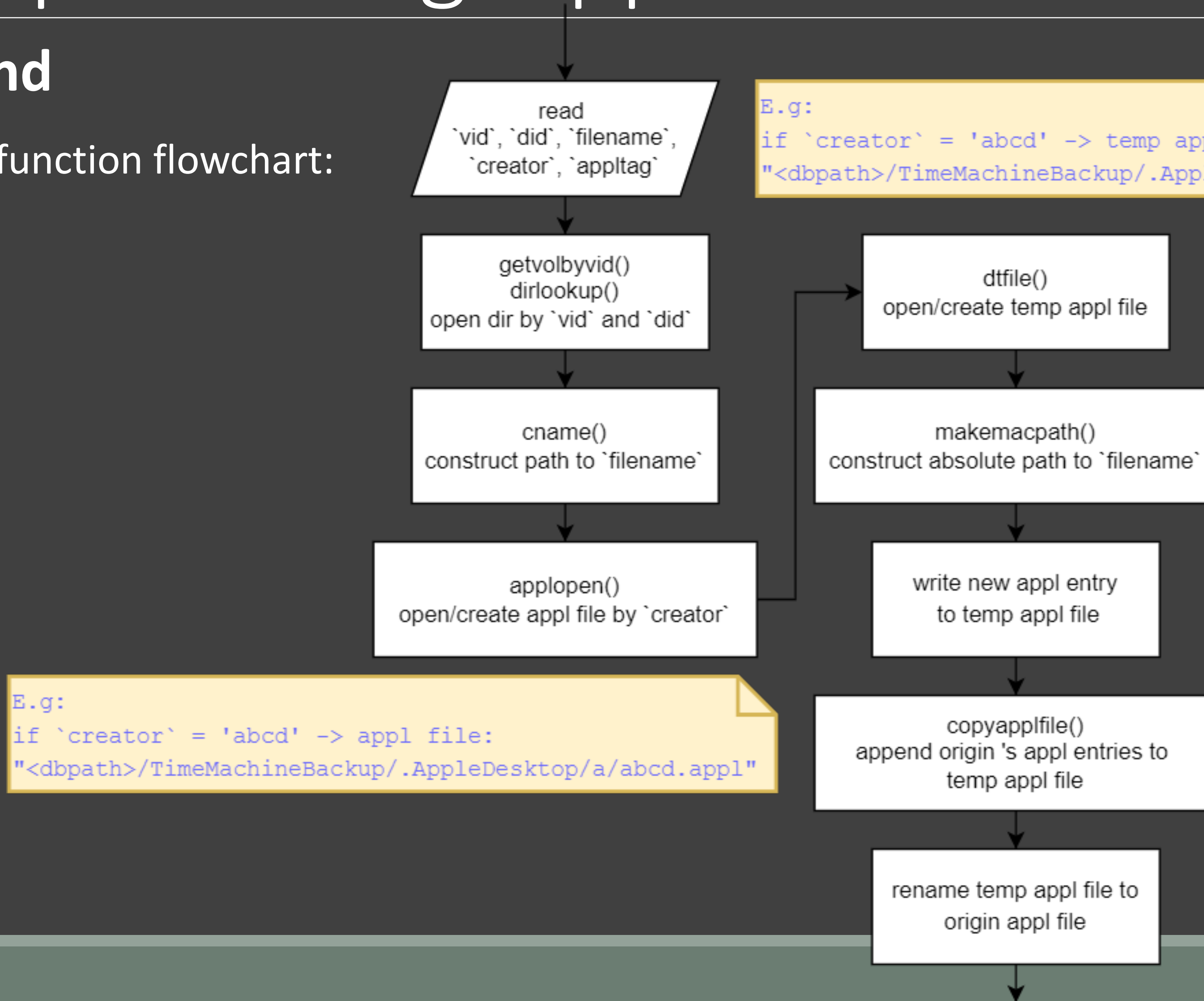
- *appl* file store database information when user read/write to files.
- In *afpd*, it has extension ".appl" and it is stored in `dbpath` which is declared in afpd.conf
- In *afpd*, there are 2 functions to create/delete *appl* file: `afp_addappl` and `afp_rmvappl`. Both functions require authentication to access. When call these functions, user will submit a `creator` value, then based on this value, a *appl* file is processed.
- *appl* file content: contains multiple chunks, chunk format:
  - `appltag`: 0x4 bytes, user supplied
  - `mlen`: 0x2 bytes, size of absolute path
  - `absolute path`: maximum 0x1000 bytes, it is absolute path of requested file by user.



Bug in processing `appl` file

Background

`afp_addappl` function flowchart:



Bug in processing `appl` file

Stack Out-Of-Bounds Write # root cause

```
86  static int copyapplfile(int sfd, int dfd, char *mpath, u_short mplen)
87  {
88    int    cc;
89    char   *p;
90    uint16_t  len;
91    unsigned char appltag[ 4 ];
92    char   buf[ MAXPATHLEN ];
93
94    while (( cc = read( sfd, buf, sizeof(appltag) + sizeof( u_short ))) > 0 ) {
95        p = buf + sizeof(appltag);
96        memcpy( &len, p, sizeof(len));
97        len = ntohs( len );
98        p += sizeof( len );
99        if (( cc = read( sa.sdt_fd, p, len )) < len ) {
100           break;
101       }
102       if ( pathcmp( mpath, mplen, p, len ) != 0 ) {
103           p += len;
104           if ( write( dfd, buf, p - buf ) != p - buf ) {
105               cc = -1;
106               break;
107           }
108       }
109   }
110   return( cc );
111 }
```

MAXPATHLEN = 0x1000

read 6 bytes
(mlen (2 bytes) + appltag (4 bytes))

read absolute path

append chunk to temporary file

Bug in processing `appl` file

Stack Out-Of-Bounds Write # root cause

- As mentioned above, maximum size of a chunk is 0x1000 + 6 = 0x1006
- `buf` is a stack-based buffer, size 0x1000
- Calling `read` function at line 99 cause Stack Out-Of-Bounds Write

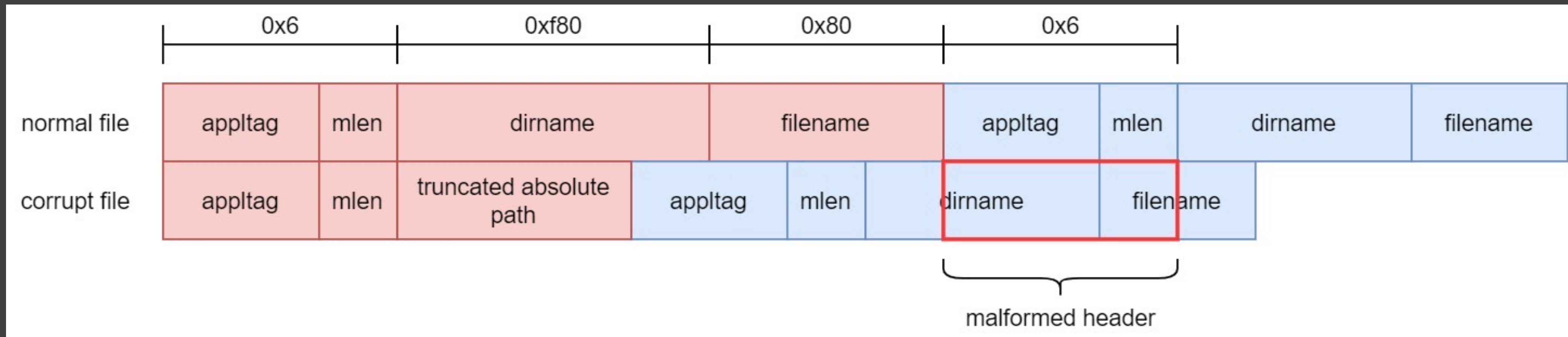
```
-00001026 buf             DCB 4096 dup(?)
-00000026                 DCB ? ; undefined
-00000025                 DCB ? ; undefined
-00000024                 DCB ? ; undefined
-00000023                 DCB ? ; undefined
-00000022 len             DCW ?
```

- With 6 bytes Out-Of-Bounds Write -> overwrite `len`
- Calling `write` function at line 104 cause writing a truncated chunk to temporary *appl* file

Bug in processing `appl` file

Stack Out-Of-Bounds Write # root cause

- The next time function `copyapplfile` parses corrupted *appl* file, the calling `read` at line 99 might cause Stack Out-Of-Bounds Write, and we could overwrite return address in stack -> RCE
- Here is a sample payload cause corrupting *appl* file:



Bug in processing `appl` file

Stack Out-Of-Bounds Write # exploitation

This bug is used to exploit WD Home Cloud (release version)

- Architecture: arm 32 bit
- Mitigation: ASLR + PIE

Notes

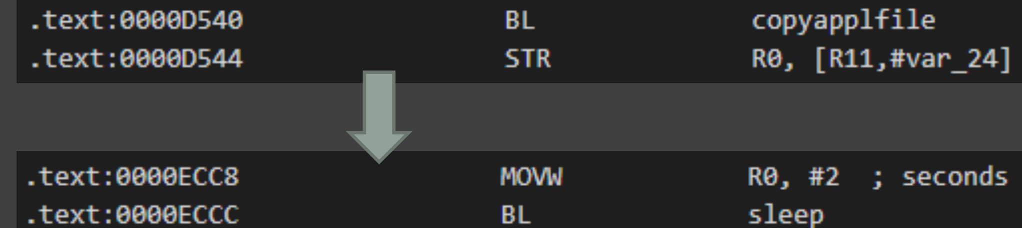
- The maximum size of filename in linux is 256, we need 0x1000 bytes -> we need to create multiple nested folder.
- Since absolute path cannot contains null char -> cannot store pointer address in it
- Red filename will overwrite the `len` value in stack, I set it to 0xf60, when translate to ascii, it is `\x60\0xf`, still valid to use in filename
- sizeof `appltag` == 4 bytes and controllable by user -> we will place malform `mlen` and malform return address in it.

Bug in processing `appl` file

Stack Out-Of-Bounds Write # exploitation

Step 1. Bypass ASLR

- *afpd* is multi-process server, using `fork` to create child process to handle a new connection -> we could partial overwrite ret address to bruteforce PIE base address.
- Partial overwrite origin ret address not work, because $r11 register is overwritten in stack and parent function use $r11 -> always crash
- I used timebase bruteforce method instead
- 2 address is different, but difference is not too large (< 0x2000) -> bruteforce still work



- 1st byte is always in range 0xa0 - 0xaf -> maximum 16 + 256 + 16 = 288 attempts to successful bruteforce PIE base address

Bug in processing `appl` file

Stack Out-Of-Bounds Write # exploitation

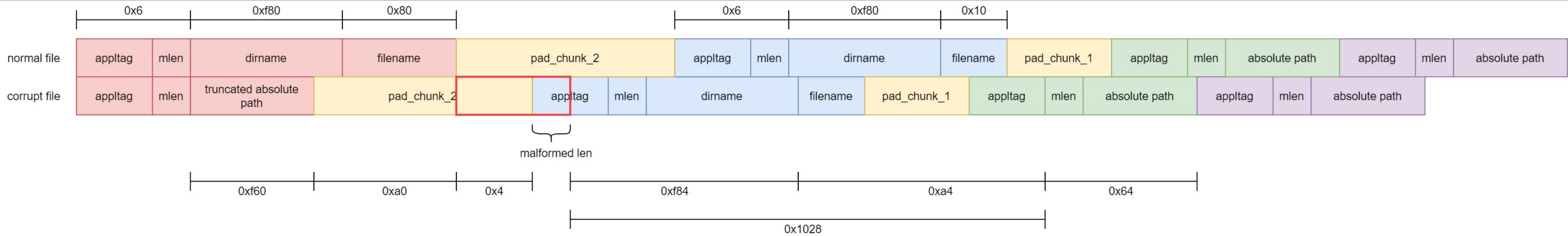
Step 2. Execute command as root

- Use the following gadget:

```
.text:000268A4        LDR        R1, [SP,#0x64]
.text:000268A8        BL         afprun_bg
```

`afprun_bg` function run command with root privilege

- Since the file content is copied to stack -> we could put address of command in file content
- The final payload:



  - Blue's `appltag` contains malformed length
  - Green's `appltag` contains address of above gadget
  - Purple's `appltag` contains address of command

Bug in processing `appl` file

Race condition # root cause

```
201 int afp_addappl(AFPObj *obj, char *ibuf, size_t ibuflen _U_, char *rbuf _U_, size_t *rbuflen)
202 {
        /* ... */
251     if (( tfd = open( tempfile, O_RDWR|O_CREAT, 0666 )) < 0 ) {
252         return( AFPERR_PARAM );
253     }
254     mpath = obj->newtmp;
255     mp = makemacpath( vol, mpath, AFPOBJ_TMPSIZ, curdir, path->m_name );
256     if (!mp) {
257         close(tfd);
258         return AFPERR_PARAM;
259     }
260     mplen =  mpath + AFPOBJ_TMPSIZ - mp;
261
262     /* write the new appl entry at start of temporary file */
263     p = mp - sizeof( u_short );
264     mplen = htons( mplen );
265     memcpy( p, &mplen, sizeof( mplen ));
266     mplen = ntohs( mplen );
267     p -= sizeof( appltag );
268     memcpy(p, appltag, sizeof( appltag ));
269     cc = mpath + AFPOBJ_TMPSIZ - p;
270     if ( write( tfd, p, cc ) != cc ) {
271         close(tfd);
272         unlink( tempfile );
273         return( AFPERR_PARAM );
274     }
        /* ... */
288 }
```

Open/Create the `tempfile` to edit

Contruct new chunk

Append new chunk to `tempfile`

Bug in processing `appl` file

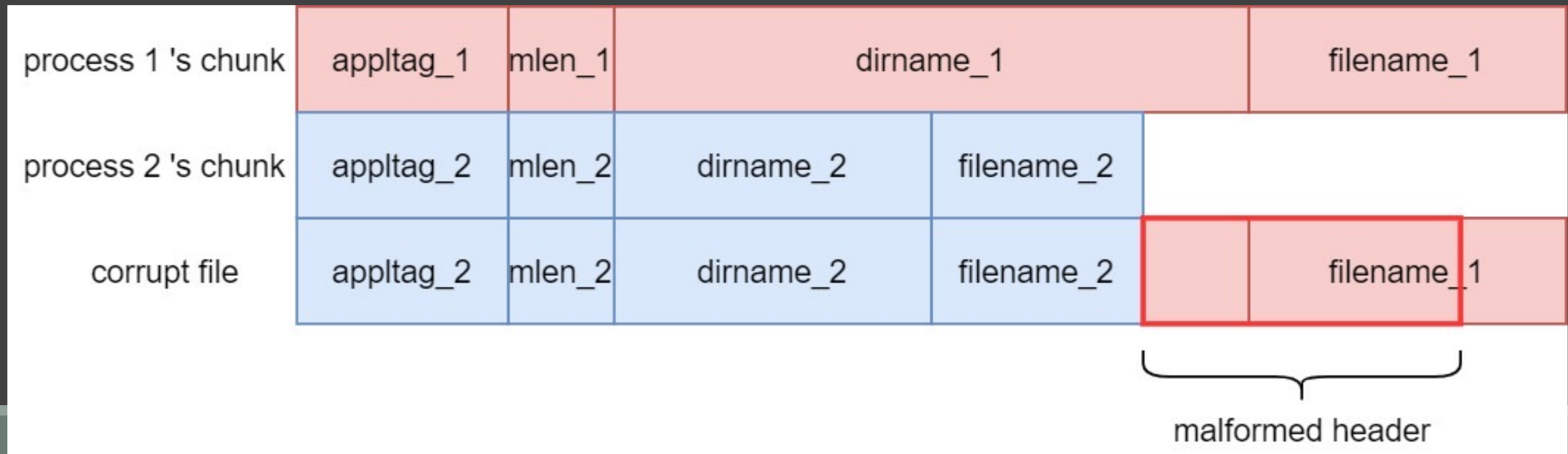
Race condition # root cause

- *afpd* is a multiple processes service – each command is processed in a separated process
- At line 251 and 270, perform file operator without lock.
- Sending multiple add appl file commands with same `creator` value -> multiples process processed a same file -> race condition
- Race condition -> chunks data might overlap each other -> corrupt the temporary appl file

Bug in processing `appl` file

Race condition # root cause

- As mentioned before, when function `copyapplfile` parses corrupted *appl* file, the calling `read` at line 99 might cause Stack Out-Of-Bounds Write, and we could overwrite return address in stack -> RCE
- Here is a sample payload cause corrupting *appl* file:



Bug in processing `appl` file

Race condition # exploitation

This bug is used to exploit WD Home Cloud (release version)

Architecture: arm 32 bit

Mitigation: ASLR + PIE

Bug in processing `appl` file

Race condition # exploitation

Step 1. Bypass ASLR

- Can reuse timebased bruteforce ?
- Race condition + bruteforce seems not reliable
- Need a information disclosure vulnerability

Bug in processing `appl` file

Race condition # exploitation

Step 2. Execute command as root

- Use the following gadget:

```
.text:000268A4    LDR    R1, [SP,#0x64]
.text:000268A8    BL     afprun_bg
```

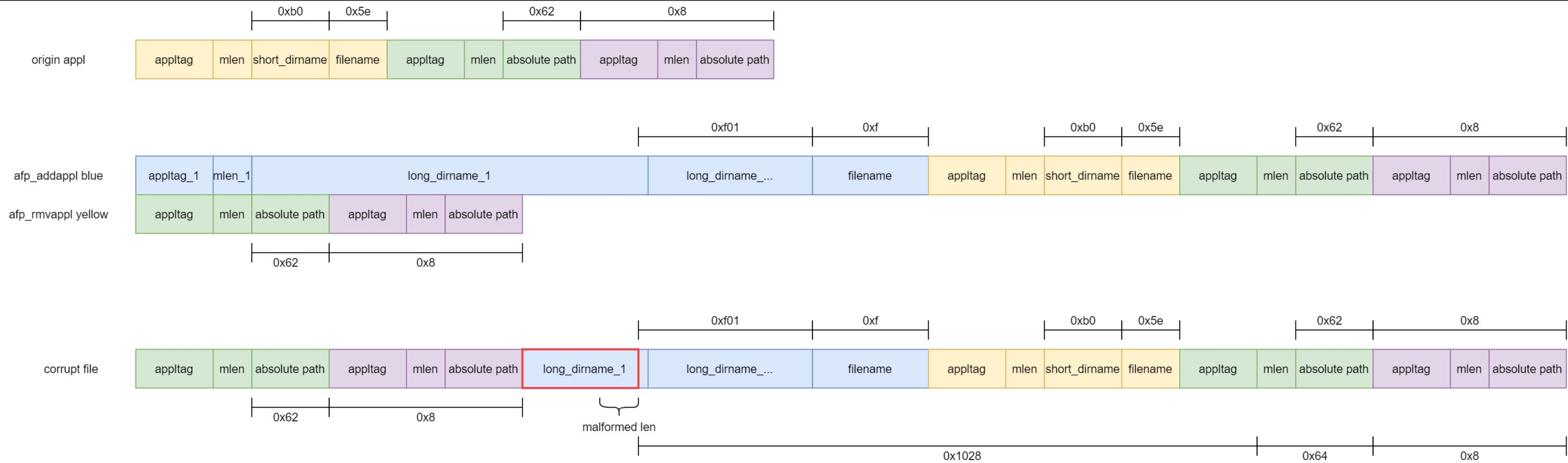
`afprun_bg` function run command with root privilege

- Since the file content is copied to stack -> we could put address of command in file content
- The race condition also occurred in `afp_rmappl` -> could race between `afp_addappl` and `afp_rmappl` processes

Bug in processing `appl` file

Race condition # exploitation

Step 2. Execute command as root



Bug in processing `appl` file

Race condition # exploitation

Step 2. Execute command as root

- Malformed length is lied on the end of Blue's directory name
- I choose `Malformed length` is 0x1108, translated to ascii name: `\x08\x11` - still valid to use as directory name
- Green's `appltag` contains address of gadget
- Purple's `appltag` contains address of command

Bug in processing `appl` file

Bonus

Netatalk weak hash function leads to information disclosure

- Bug is in “uams_dhx2_passwd.so”
- Using weak hash function to hash a pointer ??

```
152 static int dhx2_setup(void *obj, char *ibuf _U_, size_t ibuflen _U_,
153                       char *rbuf, size_t *rbuflen)
154 {
        /* ... */
209
210     /* Session ID first */
211     ID = dhxhash(obj);
212     uint16 = htons(ID);
213     memcpy(rbuf, &uint16, sizeof(uint16_t));
214     rbuf += 2;
215     *rbuflen += 2;
```

Session ID is generated by `dhxhash` function

```
47 #define dhxhash(a) ((((unsigned long) (a) >> 8) ^   \
48                     (unsigned long) (a)) & 0xffff)
```

Calculate hash by xor 🤔

- *Session ID* value here will be sent back to client later

Bug in processing `appl` file

Bonus

Netatalk weak hash function leads to information disclosure

- `obj` is a global pointer -> located in the .text section
- The NAS running 32bit OS -> 1st byte and 4th byte are known
- We could calculate the 2nd byte and 3rd byte from *Session ID*

  -> bypass ASLR

Bug in processing file ‘s `xattr`

Target

- 2 bugs: one in *afpd* and one in *smbd*
- Bugs were used to exploit the WD Pro PR4100 and WD Home Cloud (beta version)

Bug in processing file ‘s `xattr`

Background

- Extended attributes (xattr) are *name:value* pairs associated permanently with files and directories
- Both *afpd* and *smbd* have command to allow user to set xattr for a file/directory (require authentication).
- Some special xattr will be parsed when process files

Bug in processing file ‘s `xattr`

Background

In case of *afpd*:

- `afp_setextattr` command is responsible to set the *value* of the extended attribute identified by *name* and associated with the given path in the filesystem.
- It is done by invoking `setxattr`/`lsetxattr`/`fsetxattr` function.
- No checking in whole process -> user can set arbitrary *name*:*value* xattr

Bug in processing file ‘s `xattr`

Background

- `ad_open` function is responsible to open file.
- Some special xattrs are parsed here, one of them is AD_EA_META: "org.netatalk.Metadata"
- As mentioned before, no checking in the `afp_setextattr` function -> user can set the malform "org.netatalk.Metadata" xattr.

Bug in processing file ‘s `xattr`

Background

`parse_entries` function call stack

Bug in processing file ‘s `xattr`

Afpd Parsing xattr Out-Of-Bounds Access # root cause

```
401 static int parse_entries(struct adouble *ad, char *buf, uint16_t nentries)
402 {
403     uint32_t   eid, len, off;
404     int        ret = 0;
405 
406     /* now, read in the entry bits */
407     for (; nentries > 0; nentries-- ) {
408         memcpy(&eid, buf, sizeof( eid ));
409         eid = get_eid(ntohl(eid));
410         buf += sizeof( eid );
411         memcpy(&off, buf, sizeof( off ));
412         off = ntohl( off );
413         buf += sizeof( off );
414         memcpy(&len, buf, sizeof( len ));
415         len = ntohl( len );
416         buf += sizeof( len );
417 
418         ad->ad_eid[eid].ade_off = off;
419         ad->ad_eid[eid].ade_len = len;
420 
421         if (!eid
422             || eid > ADEID_MAX
423             || off >= sizeof(ad->ad_data)
424             || ((eid != ADEID_RFORK) && (off + len >  sizeof(ad->ad_data))))
425         {
426             ret = -1;
427             LOG(log_warning, logtype_ad, "parse_entries: bogus eid: %u, off: %u, len: %u",
428                 (uint)eid, (uint)off, (uint)len);
429         }
430     }
431 
432     return ret;
433 }
```

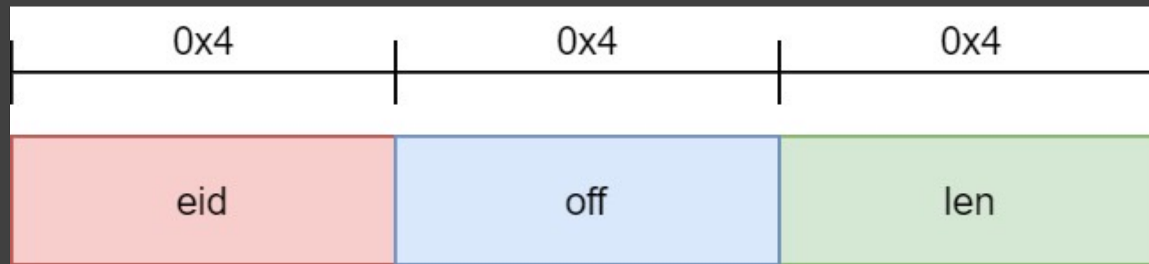
Get `eid`, `off` and `len` from ea metadata

Fill up `struct adouble *ad` object

Bug in processing file ‘s `xattr`

Afpd Parsing xattr Out-Of-Bounds Access # root cause

- Each entry in `struct adouble *ad` object has format:



- `eid`: ID of entry
- `off` is offset value from `ad->ad_data` buffer
- `len` is size of value.

- There are some checks to prevent accessing out-of-bounds of `ad->ad_eid` array and `ad->ad_data` buffer
- But no check if the `off` and `len` are valid for a specific `eid` -> leads to multiple Out-Of-Bounds access when use `struct adouble *ad` later.

Bug in processing file ‘s `xattr`

Afpd Parsing xattr Out-Of-Bounds Read # root cause

```
80  void *get_finderinfo(const struct vol *vol, const char *upath, struct adouble *adp, void *data, int islink)
81  {
82      struct extmap       *em;
83      void                *ad_finder = NULL;
84      int                 chk_ext = 0;
85
86      if (adp)
87          ad_finder = ad_entry(adp, ADEID_FINDERI);
88
89      if (ad_finder) {
90          memcpy(data, ad_finder, ADEDLEN_FINDERI);  // <-- ADEDLEN_FINDERI = 32
91          /* default type ? */
92          if (default_type(ad_finder))
93              chk_ext = 1;
94      }
95      else {
            /* ... */
122 }
```

(adp->ad_data + adp->ad_eid[ADEID_FINDERI].ade_off)

Bug in processing file ‘s `xattr`

Afpd Parsing xattr Out-Of-Bounds Read # root cause

- Line 87, get `ad_finder` pointer from `struct adouble *adp`. As mentioned above, `ad_finder` could point to the last byte of `adp->ad_data` buffer.
- Line 90, calling `memcpy` with the fixed size - 32, lack of the check if 32 > `adp->ad_eid[ADEID_FINDERI].ade_len` -> out-of-bounds read issue.
- The `data` will be sent back to user later -> information disclosure
- `adp->ad_data` is a stack-based buffer -> leak pie base address.

Bug in processing file ‘s `xattr`

Afpd Parsing xattr Out-Of-Bounds Write # root cause

```
833 static int ad_addcomment(const AFPObj *obj, struct vol *vol, struct path *path, char *ibuf)
834 {
835     struct ofork        *of;
836     char                *name, *upath;
837     int                 isadir;
838     int      clen;
839     struct adouble  ad, *adp;
840
841     clen = (u_char)*ibuf++;
842     clen = min( clen, 199 );
        /* ... */
862     if (ad_getentryoff(adp, ADEID_COMMENT)) {
863         if ( (ad_get_MD_flags( adp ) & O_CREAT) ) {
864             if ( *path->m_name == '\0' ) {
865                 name = (char *)curdir->d_m_name->data;
866             } else {
867                 name = path->m_name;
868             }
869             ad_setname(adp, name);
870         }
871         ad_setentrylen( adp, ADEID_COMMENT, clen );
872         memcpy( ad_entry( adp, ADEID_COMMENT ), ibuf, clen );
873         ad_flush( adp );
874     }
875     ad_close(adp, ADFLAGS_HF);
876     return( AFP_OK );
877 }
```

Get `clen` from user-supplied buffer

(adp->ad_data + adp->ad_eid[ADEID_COMMENT].ade_off)

Bug in processing file ‘s `xattr`

Afpd Parsing xattr Out-Of-Bounds Write # root cause

- Line 872, get pointer from `struct adouble *adp`. As mentioned above, this pointer could point to the last byte of `adp->ad_data` buffer.
- Line 872, calling `memcpy` with the controllable len `clen`, lack of the check if `clen` > `adp->ad_eid[ADEID_FINDERI].ade_len` -> out-of-bounds write issue.
- `adp->ad_data` is a stack-based buffer -> could overwrite the return address in stack.

Bug in processing file ‘s `xattr`

Afpd Parsing xattr Out-Of-Bounds Access # exploitation

This bug is used to exploit WD Pro Pr4100

Architecture: aarch64

Mitigation: ASLR + PIE

Bug in processing file ‘s `xattr`

Afpd Parsing xattr Out-Of-Bounds Access # exploitation

Step 1: Bypass ASLR

- Using Out-Of-Bounds Read to leak PIE base

Step 2: Execute command as root

- Using Out-Of-Bounds Write to overwrite the return address in stack with the following rop chain:

```
0x000000000003c429 : pop rsi ; pop r15 ; ret
```

Follow by address of a global buffer stored command + address of `afprun_bg` function

-> execute command as root

Bug in processing file ‘s `xattr`

Smbd Parsing xattr Out-Of-Bounds Access # root cause

- *smbd* also provide to user a command to set xattr of a file/directory.
- It is done by `set_ea` function.
- The *name* of our attribute must not be in the private *Samba* attribute name list (`user.SAMBA_PAI`, `user.DOSATTRIB`, `user.SAMBA_STREAMS`, `security.NTACL`)
- When *fruit* module process file/directory, it also parse some special xattr values, such as AFPINFO_EA_NETATALK: "org.netatalk.Metadata"
- `AFPINFO_EA_NETATALK` is not in the private attribute name list -> user can submit a malform xattr value

Bug in processing file ‘s `xattr`

Smbd Parsing xattr Out-Of-Bounds Access # root cause

```
870 static bool ad_unpack(struct adouble *ad, const size_t nentries,
871                       size_t filesize)
872 {
        /* ... */
907     for (i = 0; i < adentries; i++) {
908         eid = RIVAL(ad->ad_data, AD_HEADER_LEN + (i * AD_ENTRY_LEN));
909         eid = get_eid(eid);
910         off = RIVAL(ad->ad_data, AD_HEADER_LEN + (i * AD_ENTRY_LEN) + 4);
911         len = RIVAL(ad->ad_data, AD_HEADER_LEN + (i * AD_ENTRY_LEN) + 8);
            /* some checks to prevent accessing out-of-bounds of `ad->ad_eid` and `ad->ad_data` buffer */
988         ad->ad_eid[eid].ade_off = off;
989         ad->ad_eid[eid].ade_len = len;
990     }
991
992     ok = ad_unpack_xattrs(ad);
993     if (!ok) {
994         return false;
995     }
996
997     return true;
998 }
```

Get `eid`, `off` and `len` from xattr value

Fill up `struct adouble *ad` object

It is very similar to the *afpd* ‘s `parse_entries` function

Bug in processing file ‘s `xattr`

Smbd Parsing xattr Out-Of-Bounds Read # root cause

```
4264 static ssize_t fruit_pread_meta_adouble(vfs_handle_struct *handle,
4265                                         files_struct *fsp, void *data,
4266                                         size_t n, off_t offset)
4267 {
4268     AfpInfo *ai = NULL;
4269     struct adouble *ad = NULL;
4270     char afpinfo_buf[AFP_INFO_SIZE];
4271     char *p = NULL;
4272     ssize_t nread;
         /* ... */
4279     ad = ad_fget(talloc_tos(), handle, fsp, ADOUBLE_META);
4280     if (ad == NULL) {
4281         nread = -1;
4282         goto fail;
4283     }
4284
4285     p = ad_get_entry(ad, ADEID_FINDERI);
4286     if (p == NULL) {
4287         DBG_ERR("No ADEID_FINDERI for [%s]\n", fsp_str_dbg(fsp));
4288         nread = -1;
4289         goto fail;
4290     }
4291
4292     memcpy(&ai->afpi_FinderInfo[0], p, ADEDLEN_FINDERI);    // <-- ADEDLEN_FINDERI = 32
4293
4294     nread = afpinfo_pack(ai, afpinfo_buf);
4295     if (nread != AFP_INFO_SIZE) {
4296         nread = -1;
4297         goto fail;
4298     }
4299
4300     memcpy(data, afpinfo_buf, n);
4301     nread = n;
4302
4303 fail:
4304     TALLOC_FREE(ai);
4305     return nread;
4306 }
```

Create `struct adouble` object from metadata

p = ad->data + ad->ad_eid[ADEID_FINDERI].ade_off

Bug in processing file ‘s `xattr`

Smbd Parsing xattr Out-Of-Bounds Read # root cause

- Line 4285, get `p` pointer from `struct adouble *ad`. As mentioned above, `p` could point to the last byte of `ad->ad_data` buffer.
- Line 4293, calling `memcpy` with the fixed size - 32, lack of the check if 32 > `ad->ad_eid[ADEID_FINDERI].ade_len` -> out-of-bounds read issue.
- The `data` will be sent back to user later -> information disclosure
- `ad->ad_data` is a heap-based buffer.

Bug in processing file ‘s `xattr`

Smbd Parsing xattr Out-Of-Bounds Write # root cause

```
4642 static ssize_t fruit_pwrite_meta_netatalk(vfs_handle_struct *handle,
4643                               files_struct *fsp, const void *data,
4644                               size_t n, off_t offset)
4645 {
4646     struct adouble *ad = NULL;
4647     AfpInfo *ai = NULL;
4648     char *p = NULL;
4649     int ret;
4650     bool ok;
         /* ... */
4657     ad = ad_fget(talloc_tos(), handle, fsp, ADOUBLE_META);
4658     if (ad == NULL) {
4659         ad = ad_init(talloc_tos(), handle, ADOUBLE_META);
4660         if (ad == NULL) {
4661             return -1;
4662         }
4663     }
4664     p = ad_get_entry(ad, ADEID_FINDERI);
4665     if (p == NULL) {
4666         DBG_ERR("No ADEID_FINDERI for [%s]\n", fsp_str_dbg(fsp));
4667         TALLOC_FREE(ad);
4668         return -1;
4669     }
4670
4671     memcpy(p, &ai->afpi_FinderInfo[0], ADEDLEN_FINDERI);       // <-- ADEDLEN_FINDERI = 32
4672
          /*... */
4704 }
```

Create `struct adouble` object from metadata

p = ad->data + ad->ad_eid[ADEID_FINDERI].ade_off

Bug in processing file ‘s `xattr`

Smbd Parsing xattr Out-Of-Bounds Write # root cause

- Line 4664, get `p` pointer from `struct adouble *ad`. `p` pointer could point to the last byte of `ad->ad_data` buffer.
- Line 4671, calling `memcpy` with the fixed size - 32, lack of the check if 32 > `ad->ad_eid[ADEID_FINDERI].ade_len` -> out-of-bounds write issue.
- `ad->ad_data` is a heap-based buffer.

Bug in processing file ‘s `xattr`

Smbd Parsing xattr Out-Of-Bounds Access # exploitation

This bug is used to exploit WD Home Cloud (beta version)

- Architecture: arm 32 bit
- Mitigation: ASLR + PIE

Bug in processing file 's `xattr`

Smbd Parsing xattr Out-Of-Bounds Access # exploitation

Backgound

- *Smbd* implements both glibc 's `ptmalloc` and its own `talloc` memory allocation.
- Chunk format:

flags	next
prev	parent
child	refs
destructor	name
....	

Talloc chunk
- flags: chunk canary + some flags
- next, prev: point to the next/prev chunk
- destructor: function pointer, will be invoked when chunk is free

		size
fd	bk	
....		

Ptmalloc chunk:
- fd, bk: point to the next/prev chunk

Bug in processing file ‘s `xattr`

Smbd Parsing xattr Out-Of-Bounds Access # exploitation

Step 1. Bypass ASLR

- Require spray heap to create both `ptmalloc` chunk and `talloc` chunk
- Able to read up to 24 bytes past end of `ad->ad_data` chunk
  - Leak `talloc.flags` to bypass chunk canary check
  - Leak `talloc.next`, `talloc.prev` to know heap address
  - Leak `ptmalloc.fd`, `ptmalloc.bk` to know libc address (because it might point to main_arena)

Bug in processing file ‘s `xattr`

Smbd Parsing xattr Out-Of-Bounds Access # exploitation

Step 2. Control the $pc

- We know chunk canary -> can forge a valid `talloc` chunk in heap address
- We know heap address -> overwrite `next` pointer in a in used `talloc` chunk by our forge chunk -> when this in used `talloc` chunk is freed -> our forge chunk is also freed -> invoke `destructor` function
- We know libc address -> calculate other shared lib address -> set proper address in `destructor` function pointer -> execute command as root

Summary

Conclusion

- Check configuration of file service running on NAS/router devices, it might contains addition feature -> extend attack surface
- Same feature might contain same bug pattern

TODO

- Does the fuzzer work ?
- Samba: Check remain modules which declared in `vfs objects` list

Thanks for listening

Nguyen Hoang Thach (@hi_im_d4rkn3ss)

STAR Labs SG Pte. Ltd.