[1day to 0day] Netatalk from Pwn2own 2021 to 0x00 cent in 2022 (https://rushbnt.github.io/bug%20analysis/netatalk-0day/)

5 minute read

TL;DR

Hi all, This is my first blog in this page. Actually this is my second blog, but first one is missed on the previous website. I will push it again in this page. Please feed free comment or create issue in github if you detect my misstakes.

Update: reg as CVE-2022-45188

Overview

Oftenly, I found a product from pwn2own to audit. I found an amazing blog (https://research.nccgroup.com/2022/03/24/remote-code-execution-on-western-digital-pr4100-nas-cve-2022-23121) which discription about CVE-2022-23121 in Pwn2own. I tried to audit the source code of Netatalk 3.1.13 to understand the vulnerability clearly. Of couse with a hope to find a new vulnerability for next Pwn2own.

Vulnerability

When finding new bug, I often focus to memory corruption, so I find some function like `memcpy`, `strcpy`, ... Because the Netatalalk works with specific protocol (AFP) and file formats. Moreover, CVE-2022-23121 occurs when parsing `.AppleDouble` file. Therefore I found some other file format and found `.appl`. The function to read `.appl` file is `afp_getappl`:

```
/* fake up a cname */
cbuf = obj->newtmp;
q = cbuf;
*q++ = 2;    /* long path type */
*q++ = (unsigned char)len;
memcpy( q, p, len );
```

`afp_getappl` also has:

```
buf = obj->oldtmp;
while (( cc = read( sa.sdt_fd, buf, sizeof( appltag )
                    + sizeof( u_short ))) > 0 ) {
    p = buf + sizeof( appltag );
    memcpy( &len, p, sizeof( len ));
    len = ntohs( len );
    p += sizeof( u_short );
    if (( cc = read( sa.sdt_fd, p, len )) < len ) {
        break;
    }
    if ( sa.sdt_index == aindex ) {
        break;
    }
    sa.sdt_index++;
}
if ( cc <= 0 || sa.sdt_index != aindex ) {
    *rbuflen = 0;
    return( AFPERR_NOITEM );
}
sa.sdt_index++;
```

We can see `len` variable parse from 2 bytes (`u_short` types) and read `len` bytes to some buffer afterward. we also need understand `obj` (is object of `AFPObj`):

```
typedef struct AFPObj {
    const char *cmdlineconfigfile;
    int cmdlineflags;
    const void *signature;
    struct DSI *dsi;
    struct afp_options options;
    dictionary *iniconfig;
    char username[MAXUSERLEN];
    /* to prevent confusion, only use these in afp_* calls */
    char oldtmp[AFPOBJ_TMPSIZ + 1], newtmp[AFPOBJ_TMPSIZ + 1];
    void *uam_cookie; /* cookie for uams */
    struct session_info  sinfo;
    uid_t uid;  /* client login user id */
    uid_t euid; /* client effective process user id */
    int ipc_fd; /* anonymous PF_UNIX socket for IPC with afpd parent */
    gid_t *groups;
    int ngroups;
    int afp_version;
    int cnx_cnt, cnx_max;
    /* Functions */
    void (*logout)(void);
    void (*exit)(int);
    int (*reply)(void *, int);
    int (*attention)(void *, AFPUserBytes);
    int fce_version;
    char *fce_ign_names;
    char *fce_notify_script;
    struct sl_ctx *sl_ctx;
} AFPObj;
```

with `AFPOBJ_TMPSIZ` is 4096 as default, `len` is 0xffff as max, so we have a heap-based buffer overflow at here.

Exploit

Between parsing `len` and call `memcpy` like above, `afp_getappl` have one more piece:

```
    {
#define hextoint( c )   ( isdigit( c ) ? c - '0' : c + 10 - 'a' )
#define islxdigit(x)    (!isupper(x)&&isxdigit(x))

        char    utomname[ MAXPATHLEN + 1];
        char            *u, *m;
        int             i, h;

        u = p;
        m = utomname;
        i = len;
        while ( i ) {
            if ( *u == ':' && *(u+1) != '\0' && islxdigit( *(u+1)) &&
                    *(u+2) != '\0' && islxdigit( *(u+2))) {
                ++u, --i;
                h = hextoint( *u ) << 4;
                ++u, --i;
                h |= hextoint( *u );
                *m++ = h;
            } else {
                *m++ = *u;
            }
            ++u, --i;
        }

        len = m - utomname;
        p = utomname;

        if ( p[ len - 1 ] == '\0' ) {
            len--;
        }
    }
```

Basically, the above code will copy and decode byte by byte from `p` to a buffer in stack. If I exploit stack buffer overflow in `m` to override return address, overlap memory will be occured. Therefore so hard to control addresses. Therefore, I tried to find to bypass it.

I decided using

```
    while (( cc = read( sa.sdt_fd, buf, sizeof( appltag )
        ...
        if (( cc = read( sa.sdt_fd, p, len )) < len )
```

to overflow buffer `p` (point to `obj->oldtmp + 6`) in the first piece code which I showed above and using

```
if ( cc <= 0 || sa.sdt_index != aindex ) {
    *rbuflen = 0;
    return( AFPERR_NOITEM );
}
```

to break loop and exit function without reaching decoding byte to byte. Becasue `aindex` is parsed from AFP packet, I try to set it to some values (different 0 and 1) and it worked :))).

Finally, we need trigger shell and run command, I do not use any `House of ...` because I am so noob :((, so I tried to override address at end of `AFPObj`. In first try, I tried override `reply` or `exit` function pointer in `obj`. I found `send_reply` function call:

```
obj->reply(obj->dsi, err);
obj->exit(0);
```

But noway to control argument. Luckily I found that `send_fce_event` function will check `obj->fce_notify_script` and run command:

```
bstring cmd = bformat("%s -v %d -e %s -i %" PRIu32 "",
                      obj->fce_notify_script,
                      FCE_PACKET_VERSION,
                      fce_event_names[event],
                      event_id);
(void)afprun_bg(1, bdata(cmd));
```

and `afprun_bg`:

```
int afprun_bg(int root, char *cmd){
    ...
    execl("/bin/sh","sh","-c", cmd, NULL);
    ...
```

I override the `*fce_notify_script` with pointer which point to command. It is easy to indentify because `AFPObj` is saved on `.bss` segment.

Exploit strategy

I found that `.appl` file will be stored in current sharing dictionary if having config (`vol dbnest = yes`) and it is default setting in FreeBSD. It means, this is RCE vulnerability in FreeBSD and LPE in other OS. I create a demo in TrueNAS with `guest` allow and SMB enable.

Checksec of `afpd` :

```
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    Canary found
NX:       NX enabled
PIE:      No PIE (0x200000)
RUNPATH:  b'/usr/local/lib'
```

1. Create pre `.appl` file with:
    - `len` is over 4096 and contain length of command and padding in `AFPObj ( = (fce_off+8*3).to_bytes(2, 'big') )`
    - command to run
    - padding with `\x00` in `fce_off-len(cmd)` times.
    - `fce_version` is 1
    - address point command.
2. using SMB to update and modify the `.appl` file in local in `./AppleDouble/<x>/<xyzt>.appl`
3. Send AFP packet `afp_getappl` with `aindex` is a big number (like 10, 15)
4. Trigger excute command send AFP packet `afp_logout`

Conclusion

With hoping get a bounty because Netatalk appeared in Pwn2own 2021, I tried report this vulnerability to zdi, TrueNAS, Synology. But noone resolve this. After a half of year, I decided public this blog and no bounty :((.

Tags: 0day analysis CVE-2022-45188 netatalk

Categories: Bug analysis

Updated: November 9, 2022