Bug Classes

Bug Classes

- Off-by-One
- Arithmetic & Integer Boundaries
- Type conversions
- Type confusions
- Use after free
- Format string
- Heap overflows

Off-by-One

- https://cwe.mitre.org/data/definitions/193.html
- Memory corruption caused by miscalculating length of an array
- Usually due to failure to account for terminator element (\n, NULL) or misunderstanding in array indexing (0-based or 1-based)
- What is the problem here?

```
void process_string(char *src)
{
    char dest[32];
    char c = 'a'; //change to b when debugging!
    for (i = 0; src[i] && i <= sizeof(dest)); i++)
        dest[i] = src[i];
    if (c == 'b') run_sudo();
}
```

sizeof(dest) is 32
The <= allows the for loop to continue when i == 32

However,
valid dest indexes go from 0 – 31
c will be overwritten with src[32], thus making it user controllable

Off-by-One

- What is the problem here?

```
int get_user(char* user)
{
    char buf[1024];

    if (strlen(user) > sizeof(buf))
        quit();

    strcpy(buf, user);
}
```

strlen(user) is length of the string excluding null terminator
sizeof(buf) is sizeof(char)*1024 => 1024 bytes

if user is 1024 chars, it will actually take up 1025 chars (+1 for null terminator)
strcpy will write 1 extra byte to buf

Case Study: Exim

- Differences between calculated and expected size of buffer causes an off-by-one write
- In Base64, 1 byte of data is converted into 24 bits, as Base64 has a char set of 64 (2^6), 24 bits is represented by 4 bytes/chars
- Therefore, valid Base64 strings are in multiples of 4
- Encoding formula: if decoded string length == n, Base64 encoded string will be 4[n/3]
- An invalid Base64 length will cause a mismatch between the required decoded length vs the calculated value
- Valid example:
  - Expected: sizeof(code) == 32, sizeof(result) should be 24
  - Calculated value == 3 * 32 / 4 + 1 = 24 + 1
- Invalid example:
  - Expected: sizeof(code) == 43, sizeof(result) should be 32
  - Calculated value == 3 * 43 / 4 + 1 = 3 * 10 + 1 = 30 + 1
  - “result” will be allocated with size 31 but the decode algorithm expects 32

```
152  int
153  b64decode(uschar *code, uschar **ptr)
154  {
155  int x, y;
156  uschar *result = store_get(3*(Ustrlen(code)/4) + 1);
157
158  *ptr = result;
```

Case Study: Exim

- Exim Off-By-One writeup + exploit(!): https://devco.re/blog/2018/03/06/exim-off-by-one-RCE-exploiting-CVE-2018-6789-en/
- Vulnerable code: https://github.com/Exim/exim/blob/exim-4_89+fixes/src/src/base64.c#L153

Case Study: sudo (CVE-2021-3156)

- sudo –s or sudo –i (MODE_SHELL a

cont

For loop reads NewArgv to unescape characters

Eg sudo –i aaa bbb ccc
NewArgv will be [aaa, bbb, ccc]

What happens if input has a backslash?
Eg sudo –i \ bbb?

from[0] == '\\' and from[1] == "<null>"
from will be incremented to point at "<null>"

to++ = from++
to is incremented and points to "<null>". from is incremented to point to "bbb", the while loop will keep looping until bbb ends. Note that at this juncture, the enclosing for loop hasn't actually moved on to process "bbb" at all

```
/* set user_args */
if (NewArgc > 1) {
    char *to, *from, **av;
    size_t size, n;

    /* Alloc and build up user_args. */
    for (size = 0, av = NewArgv + 1; *av; av++)
        size += strlen(*av) + 1;
    user_args = emalloc(size);
    if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
        /*
        * When running a command via a shell, the sudo front-end
        * escapes potential meta chars.  We unescape non-spaces
        * for sudoers matching and logging purposes.
        */
        for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
            while (*from) {
                if (from[0] == '\\' && !isspace((unsigned char)from[1]))
                    from++;
                *to++ = *from++;
            }
            *to++ = ' ';
        }
        *--to = '\0';
    }
    ...
}
```

sudo (CVE-2021-3156)

sudo –i \ bb, NewArgv = [“\”, “bb”]

user_args = malloc(3 + 1)

1st iteration in for loop:

to = user_args[0]; av = “\”; from = “\”
while (from != null)
(from[0] == ‘\\’ and from[1] == “<null>”) == true
therefore, from = “<null>”
to[1] = “<null>”; from = “bb”

While loop executes again as from is NOT null
from = “bb”
while (from != null)
if stmt skipped
to[2] = “b”; from = “b”

While loop executes again as from is NOT null
from = “b”
while (from != null)
if stmt skipped
to[3] = “b”; from = “<null>”
While loop exits, from is incremented to the next argument “bb”

2nd iteration in for loop:

from = “bb”
while (from != null)
if stmt skipped
to[4] = “b”; from = “b”

While loop executes again as from is NOT null
from = “b”
while (from != null)
if stmt skipped
to[5] = “b”; from = “<null>” ← **OOB**

```
/* set user_args */
if (NewArgc > 1) {
    char *to, *from, **av;
    size_t size, n;

    /* Alloc and build up user_args. */
    for (size = 0, av = NewArgv + 1; *av; av++)
        size += strlen(*av) + 1;
    user_args = emalloc(size);
    if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
        /*
        * When running a command via a shell, the sudo front-end
        * escapes potential meta chars.  We unescape non-spaces
        * for sudoers matching and logging purposes.
        */
        for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
            while (*from) {
                if (from[0] == '\\' && !isspace((unsigned char)from[1]))
                    from++;
                *to++ = *from++;
            }
            *to++ = ' ';
        }
        *--to = '\0';
    }
    ...
}
```

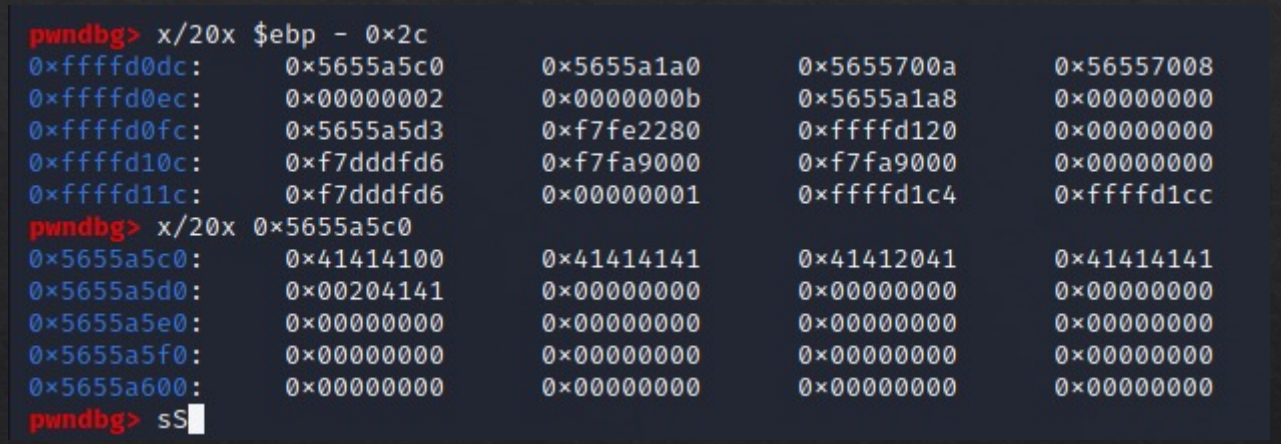
Case Study (CVE-2021-3156)

```c
int main()
{
    char **NewArgv;
    int NewArgc = 2;
    char *to, *from, *user_args, **av;
    size_t size, n;

    char *arg1 = "\\";
    char *arg2 = "AAAAAAAA";

    NewArgv = malloc((NewArgc) * sizeof(char*));

    NewArgv[0] = arg1;
    NewArgv[1] = arg2;

    for (size = 0, av = NewArgv; *av; av++)
    {
        size += strlen(*av) + 1;
    }
    user_args = malloc(size); // size == 1 + 8 + 2
    for (to = user_args, av = NewArgv; (from = *av); av++)
    {
        printf("\nfrom %s %c\n", from, from[0]);
        while (*from)
        {
            printf("zzzz %c, ", from[0]);
            if (from[0] == '\\') // dropped isspace check
                from++;
            *to++ = *from++;
        }
        *to++ = ' ';
    }
    printf("\ntest: %s %d\n", user_args, sizeof(user_args));
}
```

Case Study: sudo (CVE-2021-3156)

$ebp – 0x2c → ptr of user_args

user_args contains 16 A's, 2 spaces and 2 nulls
But user_args was malloced with 11



Case Study: sudo (CVE-2021-3156)

- Successful exploitation of this CVE causes LPE
- https://blog.qualys.com/vulnerabilities-threat-research/2021/01/26/cve-2021-3156-heap-based-buffer-overflow-in-sudo-baron-samedit
- https://github.com/worawit/CVE-2021-3156
- https://github.com/sudo-project/sudo/blob/8255ed69b9c426d90a10c6d68e8d2241d7f3260e/plugins/sudoers/sudoers.c
- Line numbers don't match up with Qualys' blog but still looks kosher

Case Study: OSSEC

- OS_CleanMSG tries to decode syslog messages
- Msg = “[ID xxx] this is a syslog”, the char ptr will be moved to remove “[ID xxx] “

```
int OS_CleanMSG(char *msg, Eventinfo *lf)
{
    size_t loglen;
    char *pieces;
    struct tm *p;
    ...
    /* Remove [ID xx facility.severity] */
    if (pieces) {
        /* Set log after program name */
        lf->log = pieces;

        if ((pieces[0] == '[') &&
                (pieces[1] == 'I') &&
                (pieces[2] == 'D') &&
                (pieces[3] == ' ')) {
            pieces += 4;

            /* Going after the ] */
            pieces = strchr(pieces, ']');
            if (pieces) {
                pieces += 2;
                lf->log = pieces;
            }
        }
    }
```

If pieces starts with “[ID “, advance pieces by 4

Find index of “]”
If index is found, advance pieces by 2

What is the bug here?

The code makes an assumption that the “]” character is followed by a character

What will happen if pieces is “[ID]”?

Case Study: OSSEC

- Full writeup: https://github.com/ossec/ossec-hids/issues/1816
- Code: https://github.com/ossec/ossec-hids/blob/abb36d4460b9f44ee90790d1138192bb01a2b21c/src/analysisd/cleanevent.c#L25
- Extra material: https://x41-dsec.de/lab/advisories/x41-2021-002-nginx-resolver-copy/

fgetwln reads a stream character by character and puts it into a filewbuf structure named fb

Case Study: libl

- What is the problem here?

When the while loop is first called, wused == 0, and fb->len is 0

So we go into the if stmt
fb->len = 128 and fb->wbuf = malloc(128)

```
#define FILEWBUF_INIT_LEN   128
#define FILEWBUF_POOL_ITEMS 32

static struct filewbuf fb_pool[FILEWBUF_POOL_ITEMS];
static int fb_pool_cur;
```

wbuf[wused] will be assigned to the wchar read from the stream

So for wused == 0 → 127, the if stmt will be skipped and just this line will be executed

But what happens when wused == 128?
It will skip the if stmt and go to fb->wbuf[128] = wc, writing a character into an uninitialized part of memory

```
wchar_t *
fgetwln(FILE *stream, size_t *lenp)
{
    struct filewbuf *fb;
    wint_t wc;
    size_t wused = 0;

    /* Try to diminish the possibility of several fgetwln() calls being
     * used on different streams, by using a pool of buffers per file. */
    fb = &fb_pool[fb_pool_cur];
    if (fb->fp != stream && fb->fp != NULL) {
        fb_pool_cur++;
        fb_pool_cur %= FILEWBUF_POOL_ITEMS;
        fb = &fb_pool[fb_pool_cur];
    }
    fb->fp = stream;

    while ((wc = fgetwc(stream)) != WEOF) {
        if (!fb->len || wused > fb->len) {
            wchar_t *wp;

            if (fb->len)
                fb->len *= 2;
            else
                fb->len = FILEWBUF_INIT_LEN;

            wp = reallocarray(fb->wbuf, fb->len, sizeof(wchar_t));
            if (wp == NULL) {
                wused = 0;
                break;
            }
            fb->wbuf = wp;
        }

        fb->wbuf[wused++] = wc;

        if (wc == L'\n')
            break;
    }

    *lenp = wused;
    return wused ? fb->wbuf : NULL;
}
```

Case Study: libbsd (CVE-2016-2090)

- https://github.com/freedesktop/libbsd/blob/0.8.1/src/fgetwln.c
- https://github.com/freedesktop/libbsd/blob/0.8.2/src/fgetwln.c