INTRODUCTION
CVE-2017-13872 is an authentication flaw that affects MacOS v.10.13 (High Sierra). It allows any user to log in as root without a password (using any or a blank password). First noticed Nov 13th 2017. First patch provided by Apple Nov 29th 2017.
The following work is based on public information found online but mainly based on the analysis and debugging done by Patric Wardle in his blog [1].

VULNERABILITY
If a user who already has access to a vulnerable MacOS computer tries to log in as root then this currently active user has to do two login attempts. The first attempt is for enabling the root account with a blank password and the second attempt is for authentication and logging in as root. For example in Users & Groups settings > select desired account > use 'root' user name and leave pwd field blank > click Unlock > repeat if necessary and that’s it, you are logged in as root:
Or at startup > click Other > use root user name and leave pwd field blank > enter > repeat if necessary.

If remote access (System Preferences > Sharing) is enabled then it works over remote connection as well.

The Cause

In MacOS the user authentication goes through the daemon process ‘opendirectoryd’ which is the main process of a directory service Open Directory. Shortly about Open Directory [2]:

Open Directory forms the foundation of how Mac OS X accesses all authoritative configuration information (users, groups, mounts, managed desktop data, etc.).

According to debugging of this process done by Patrick Wardle and who used a fake made up password ‘hunter2’, we can take a look at exact function calls during the authentication. As detailed step by step process with pictures is presented in Patric’s blog I will sum it up a bit more shortly:

When the ‘opendirectoryd’ receives an authentication message, the function ‘odm_RecordVerifyPassword’ is called which then calls another function ‘sub_18f1’. In ‘sub_18f1’ there is a parameter R13 which is a dictionary that holds the target user account info (the account what the active user wants to activate). In this dictionary there is a key ‘dsAttrTypeStandard:Password’ which value is ‘*’ (asterisk).
Continuing, another function ‘sub_826b’ is called which calls for function ‘sub_5192’. In ‘sub_5192’ a user data ‘shadowhash’ is read from the target account.

```c
if (os_log_type_enabled(0x29ad8, 0x2) != 0x0) {
    r15 = 0x29ad8;
    rbx = rsp;
    *((int8_t *)(rsp + 0xfffffffffffffff0) = 0x2;
    *((int8_t *)(rsp + 0xfffffffffffffff1) = 0x1;
    *(int8_t *)(rsp + 0xfffffffffffffff2) = 0x42;
    *(int8_t *)(rsp + 0xfffffffffffffff3) = 0x8;
    *(rsp + 0xfffffffffffffff4) = e"dsAttrTypeNative:ShadowHashData";
    _os_log_impl(rsp + 0xfffffffffffffff5099e, r15, 0x2, "read shadowhash data"
}
```

The ‘shadowhash’ data exists if the target account is enabled (for example for the active user). If the target account is not enabled then this ‘shadowhash’ does not exist and returns zero. In case of this zero is returned an ‘else’ branch is executed in ‘sub_826b’.

```c
rax = sub_5192(var_98, r13, r12, r14, &var_88, &var_60);
if (rax != 0x0) {
    //found shadow hash data
}
//no shadow hash data found
else {
    //read 'dsAttrTypeStandard:Password'
    rax = cdproplist_get_array(r12, "_kODAAttributeTypePassword");
    ...

    var_41 = 0x0;
    var_34 = 0x1234;
    if (od_verify_crypt_password(var_70, rax, var_60, &var_54, &var_41) != 0x0) {
        //upgrade password
        sub_13d00(arg7, var_60);
        sub_14324(var_70, var_60, var_68, var_50, r15, var_60, arg7);
    }
```
In this else-branch, if for example a fake randomly made up ‘hunter2’ pwd is entered then in the dictionary (R13) the 'dsAttrTypeStandard:Password' key '*' is used and a non-zero value is returned by 'od_verify_crypt_password' in 'sub_826b'. We will return to this detailed meaning below but for now it can be said that the non-zero value means ‘no match’ between target account pwd and provided fake pwd. But the process does not stop in failure.

After the non-zero is returned an upgrade from this crypt pwd to 'shadowhash' data is done (with the fake pwd). But no other checks are done after this non-zero value. Like it was a correct password the process continues. This made up pwd is converted to 'shadowhash' data (or securetoken) and saved for target user account (for example root account). So the active user can be logged in now, with this made up pwd (or a blank one).

"found crypt password in user-record - upgrading to shadowhash or securetoken"

Why the 'od_verify_crypt_password' function lets this made up fake pwd through?

The 'od_verify_crypt_password' function is called with several parameters. According to the debugging, before the function call, one of those parameters, 'var_54', is a pointer to a parameter that is set to zero if the target's account pwd and the provided pwd match.

```c
//sub_826b
//check password and upgrade if necessary
if (od_verify_crypt_password(var_70, rsx, var_60, &var_54, &var_41) != 0x0)
{
    //upgrade password
    sub_13d00(arg7, var_60);
    sub_14324(var_70, var_A0, var_68, var_50, r15, var_60, arg7);
}
```

Closer look at the pwd verification:
Before the function call 'od_verify_crypt_password' this 'var_54' value is set to 0x1388 which means 'kODErrorCredentialsInvalid'.

```c
var_54 = 0x1388;
if (od_verify_crypt_password(var_70, rsx, var_60, &var_54, &var_41) != 0x0)
{
...
```

<table>
<thead>
<tr>
<th>Platform</th>
<th>Framework</th>
<th>Error Name</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CFOOpenDirectory</td>
<td>kODErrorCredentialsInvalid</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>CFOOpenDirectoryConstants.h</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The actual pwd verification is done when 'od_verify_crypt_password' calls 'crypt_verify' which takes also parameters like the target account pwd hash (in this case this asterisk ""), the fake pwd and the var_54, and then verifies if those named pwd-s match. In this case it's comparing '""' against 'hunter2'. But as discovered earlier, after the 'od_verify_crypt_password' returns a non-zero result, no other checks are done, so the 'var_54' pointed match-result is not checked.
after the function call. Upgrade functions that convert this provided fake pwd to ‘shadowhash’
(or securetoken) are now called, against all logic.

```c
//verify
// 'match' will be set to 0x0 if verification is ok!
int match = kODDBadCredentialsInvalid;

od_verify_crypt_password(accountHash, providedPassword, &match, ...);
......

//verify by checking hashes
// 'match' will be set to 0x0 if verification is ok!
if(strcmp(providedTWRHash, accountTWRHash, user) == 0x0) {
    *match = 0x0;
}
```

```
var_54 = 0x1388;
if (od_verify_crypt_password(var_70, var_68, var_60, var_54, var_41) != 0x0) {
    //dispacth_oncc(var_29a90, ^ { /* block implemented at sub_16535 */ })
}
if (var_13d00(var_68, var_60, var_58, r15, var_68, arg7) != 0x0) {
    os_log_type_enabled(+0x29000, 0x1) != 0x0) {
        r12 = 0x29b88;
        int1b_t_t |= r12 + 0xffffffffff0f0 = 0x0;
        _os_log_impl(r12 + 0xffffffffff0e023, r12, 0x1, "found crypt password in user-record — upgrading
        rsp = rsp;
        r15 = var_78;
    }
    var_13d00(var_68, var_60, var_58, r15, var_68, arg7);    
    r15 = (rsp - 0x10) + 0x10;
}
```

**EXPLOIT**

Any user can bypass authentication and perform unauthorized activities in target computer while having the root privileges.

**FIX**

After Apple released its patches the author of this debugging checked that the analysis and debugging done was correct and that Apple added code to detect those mismatching pwd-s, which means this ‘var_54’ is now taken into account and checked also after the ‘od_verify_crypt_password’ function call. Now, only if the target account pwd and provided pwd match, target account can be activated by the user who attempts this action.
The fix was basically a 'typo' fix and not a massive code redesign. Apple called it formally a logic error which it is, by its nature.

This update patch caused some new problems (broke file sharing) but this is another subject and will not be discussed in this paper.
References

1. https://objective-see.com/blog/blog_0x24.html
3. https://www.youtube.com/watch?v=HxmjhXEkkrs