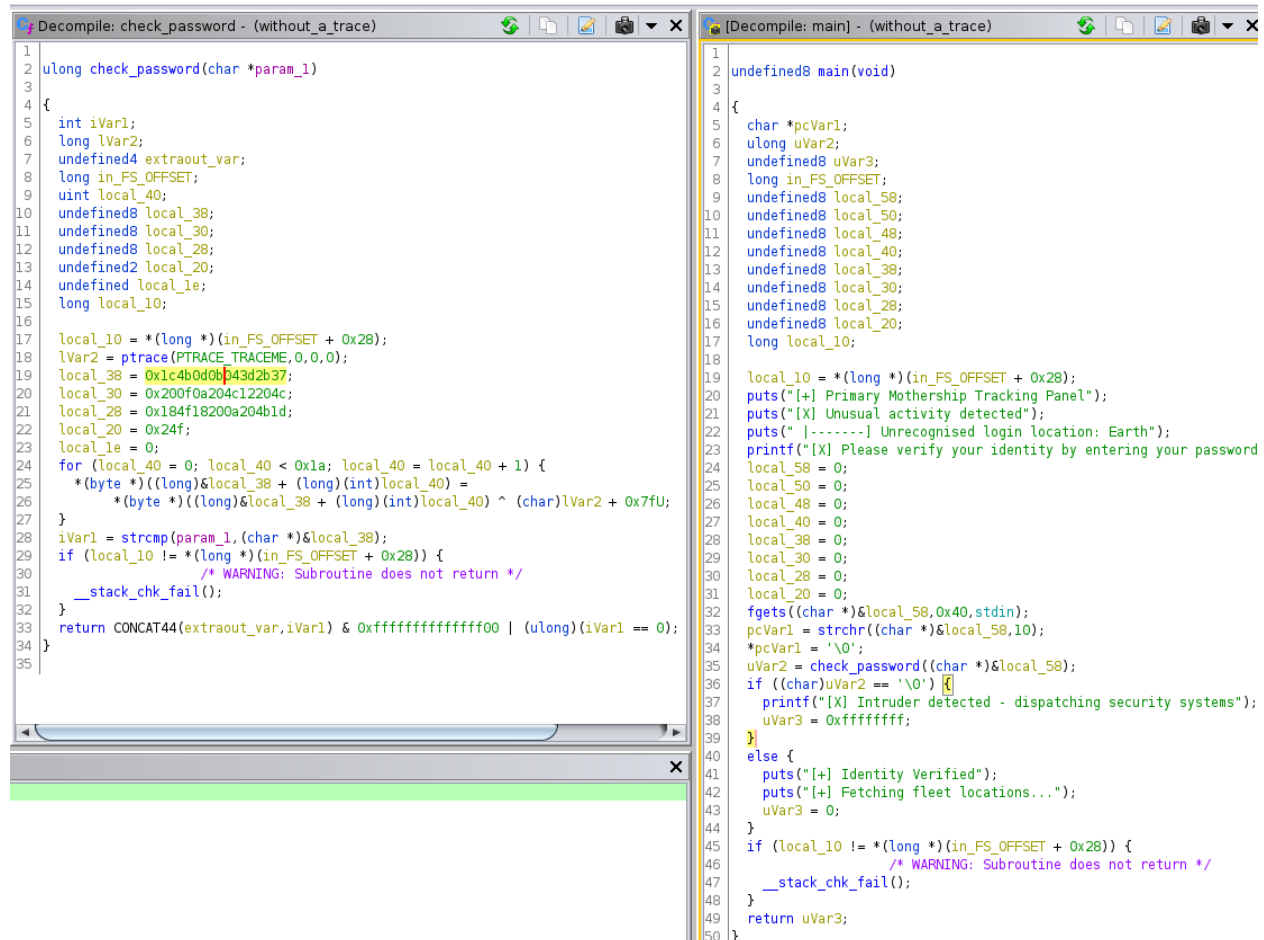


## HTB Reverse Engineering: Without a Trace

Tools: Ghidra, ltrace

With this one I was stumped for awhile and became convinced ptrace was covering the password somehow and all i needed to do was decipher it.



The image shows two side-by-side decompilation windows from Ghidra. The left window, titled 'Decompile: check\_password - (without\_a\_trace)', shows a function with a loop that iterates over a password buffer, comparing it to a target value. The right window, titled 'Decompile: main - (without\_a\_trace)', shows the main function which calls the password check function. The code in the right window is more verbose, showing the main function's logic and the password check function's implementation.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
ulong check_password(char *param_1)
{
    int iVar1;
    long lVar2;
    undefined4 extraout_var;
    long in_FS_OFFSET;
    uint local_40;
    undefined8 local_38;
    undefined8 local_30;
    undefined8 local_28;
    undefined2 local_20;
    undefined local_1e;
    long local_10;

    local_10 = *(long *) (in_FS_OFFSET + 0x28);
    lVar2 = ptrace(PTRACE_TRACEME, 0, 0, 0);
    local_38 = 0x1c4b0d0b543d2b37;
    local_30 = 0x200f0a204c12204c;
    local_28 = 0x184f18200a204b1d;
    local_20 = 0x24f;
    local_1e = 0;
    for (local_40 = 0; local_40 < 0x1a; local_40 = local_40 + 1) {
        *(byte *) ((long) &local_38 + (long) (int) local_40) =
            *(byte *) ((long) &local_38 + (long) (int) local_40) ^ (char) lVar2 + 0x7fU;
    }
    iVar1 = strcmp(param_1, (char *) &local_38);
    if (local_10 != *(long *) (in_FS_OFFSET + 0x28)) {
        /* WARNING: Subroutine does not return */
        __stack_chk_fail();
    }
    return CONCAT44(extraout_var, iVar1) & 0xfffffffffff00 | (ulong) (iVar1 == 0);
}

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
undefined8 main(void)
{
    char *pcVar1;
    ulong uVar2;
    undefined8 uVar3;
    long in_FS_OFFSET;
    undefined8 local_58;
    undefined8 local_50;
    undefined8 local_48;
    undefined8 local_40;
    undefined8 local_38;
    undefined8 local_30;
    undefined8 local_28;
    undefined8 local_20;
    long local_10;

    local_10 = *(long *) (in_FS_OFFSET + 0x28);
    puts("[+] Primary Mothership Tracking Panel");
    puts("[X] Unusual activity detected");
    puts(" [-----] Unrecognised login location: Earth");
    printf("[X] Please verify your identity by entering your password\n");
    local_58 = 0;
    local_50 = 0;
    local_48 = 0;
    local_40 = 0;
    local_38 = 0;
    local_30 = 0;
    local_28 = 0;
    local_20 = 0;
    fgets((char *) &local_58, 0x40, stdin);
    pcVar1 = strchr((char *) &local_58, 10);
    *pcVar1 = '\0';
    uVar2 = check_password((char *) &local_58);
    if ((char) uVar2 == '\0') {
        printf("[X] Intruder detected - dispatching security systems");
        uVar3 = 0xffffffff;
    }
    else {
        puts("[+] Identity Verified");
        puts("[+] Fetching fleet locations...");
        uVar3 = 0;
    }
    if (local_10 != *(long *) (in_FS_OFFSET + 0x28)) {
        /* WARNING: Subroutine does not return */
        __stack_chk_fail();
    }
    return uVar3;
}
```

I believed the password was 8 characters in length due to the decompilation on the right asking to verify the password with 8 different values. Then in the left side decompilation ptrace is showing other values that I was unable to decode.

However running `$ ltrace -i -C` on the file revealed this.

```
(base) (rogue1@rogue1)~/HTB/CTF/Apocalypse2022/rev_without_a_trace
$ ltrace -i -C ./without_a_trace
[0x55e8ffa0094b] puts("[+] Primary Mothership Tracking " ... [+] Primary Mothership Tracking Panel
)
= 38
[0x55e8ffa00957] puts("[X] Unusual activity detected"[X] Unusual activity detected
)
= 30
[0x55e8ffa00963] puts(" |-----] Unrecognised login lo" ... |-----] Unrecognised login location: Earth
)
= 46
[0x55e8ffa00974] printf("[X] Please verify your identity " ... )
= 60
[0x55e8ffa009cc] fgets([X] Please verify your identity by entering your password > whateverifeellikeputting
"whateverifeellikeputting\n", 64, 0x7ff2ce1c39a0) = 0x7ffc03f2ef20
[0x55e8ffa009dd] strchr("whateverifeellikeputting\n", '\n')
= "\n"
[0x55e8ffa00893] ptrace(0, 0, 0, 0)
= -1
[0x55e8ffa0090d] strcmp("whateverifeellikeputting", "IUCzus5b2^l2^tq^c5^t^f1f1|") = 46
[0x55e8ffa00a20] printf("[X] Intruder detected - dispatch" ... )
= 52
[X] Intruder detected - dispatching security systems[0xffffffffff] ++ exited (status 255) ++
```

There is an interesting output just after the password I entered. `IUCzus5b2^l2^tq^c5^t^f1f1|` is xored and I can tell this from the `^` symbol. So I am going to take that value and input into the online Xor decoder.

SEARCH A TOOL ON DCODE BY KEYWORDS:  
e.g. type 'sudoku'

BROWSE THE FULL DCODE TOOLS' LIST

Results

route force attempt. Only relevant results are displayed.

[ Hexadecimal key | Plain text ]

↑↓	↑↓
1111	HTB{tr4c3_m3_up_b4_u_g0g0}
1113	HTB{ytr4a3_m1_up}b4_w_g0e0}
1117	HTB{tr4e3_m5_upYb4_s_g0a0}
1121	HTA{tr7c3_n3_us_b4u_g3g0}
1123	HTA{ytr7a3_n1_us}b4w_g3e0}
1127	HTA{tr7e3_n5_usYb4s_g3a0}
1131	HT@{tr6c3_o3_ur_b4}u_g2g0}
1133	HT@ytr6a3_o1_ur}b4}w_g2e0}
1137	HT@}tr6e3_o5_urYb4}s_g2a0}
1151	HTF{tr0c3_i3_ut_b4[u_g4g0}
1153	HTFytr0a3_i1_ut}b4[w_g4e0}

ASCII Printable Characters (Automatic Detection)

IUCzus5b2^l2^tq^c5^t^f1f1|

ENCRYPTION/DECRYPTION METHOD

☐ AUTOMATIC (BRUTEFORCE 1 TO 16 BYTES)

☐ USE THE BINARY KEY

☐ USE THE HEXADECEMAL KEY

☐ USE THE ASCII KEY

☒ KNOWING THE KEY SIZE (IN BYTES) 4

RESULTS FORMAT

☒ ASCII (PRINTABLE) CHARACTERS

☐ HEXADECEMAL 00-7F-FF

☐ DECIMAL 0-127-255

☐ OCTAL 000-177-377

☐ BINARY 00000000-11111111

☐ INTEGER NUMBER

☐ FILE TO DOWNLOAD

► ENCRYPT / DECRYPT

Awesome, we got the flag!

I was pretty surprised at how easy this one turned out to be and even messaged an admin to see if this was the correct way to do it. They told me yes but I still feel like there was another way to solve it. Also not knowing the true key size is something I probably missed. I had tried 8, 7, and 6, before trying 4. But if you know to run `ltrace` on all of your binaries then this one can be one of the easiest challenges.