

## ANALYSIS OF THE BINARY:

We first run the binary and face a segmentation fault. When we look into the disassembly, we find that a flag.txt file is needed. So I create a flag.txt file with 'aaaaaaaaaaaaaaaa'. So that it is easily recognisable in the stack when we analyse the stack.

We load the binary in Cutter and get the decompilation:



```
// WARNING: [r2ghidra] Failed to match type file* for variable stream to Decompiler type: Unknown type identifier file
// WARNING: [r2ghidra] Detected overlap for variable var_84h
// WARNING: [r2ghidra] Detected overlap for variable var_8ch
// WARNING: [r2ghidra] Detected overlap for variable var_8dh

undefined8 main(void)
{
    char cVar1;
    undefined8 uVar2;
    int64_t in_FS_OFFSET;
    int32_t var_8ch;
    int64_t var_88h;
    undefined8 stream;
    int64_t var_78h;
    char acStack120 [32];
    char *format;
    int64_t canary;

    canary = *(int64_t *) (in_FS_OFFSET + 0x28);
    setvbuf(_reloc.stdin, 0, 2, 0);
    setvbuf(_reloc.stdout, 0, 2, 0);
    puts(0xa68);
    uVar2 = fopen(0xa88, 0xa86);
    var_8ch = 0;
    puts(0xa98);
    while( true ) {
        cVar1 = fgetc(uVar2);
        if (cVar1 == -1) break;
        acStack120[var_8ch] = cVar1;
        var_8ch = var_8ch + 1;
    }
    acStack120[var_8ch] = '\0';
    puts(0xabf);
    fgets(&format, 0x40, _reloc.stdin);
    printf(&format);
    uVar2 = 0;
    if (canary != *(int64_t *) (in_FS_OFFSET + 0x28)) {
        uVar2 = __stack_chk_fail();
    }
    return uVar2;
}
```

Auto Refresh Refresh

Decompiler: Ghidra

Dashboard **Decompiler** Strings Imports Search

At once we notice a format string vulnerability. We also notice that the flag from 'flag.txt is loaded into the stack one by one.

Next we load the binary and attach it to gdb

```
Terminal
[?] Q ⋮ - + x

$R14 : 0x0
$R15 : 0x0
gef> flags: [ZERO carry PARITY adjust sign trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0x0033 $ss: 0x002b $ds: 0x0000 $es: 0x0000 $fs: 0x0000 $gs: 0x0000

0x00007fff7768c40 +0x0000: 0x00000027ff000000 ←$rsp
0x00007fff7768c48 +0x0000: 0xdeadbeef00000000
0x00007fff7768c50 +0x0010: 0x000056301b1172a0 → 0x00000000fbad2498
0x00007fff7768c58 +0x0018: 0x400911d14e3bcd36
0x00007fff7768c60 +0x0020: "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\n"
0x00007fff7768c68 +0x0028: "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\n"
0x00007fff7768c70 +0x0030: "aaaaaaaaaaaaaaaaaaaaaaaa\n"
0x00007fff7768c78 +0x0038: "aaaaaaaaaaaaa\n"

0x56301ae4b98b <main+241> mov     BYTE PTR [rbp+rax*1-0x70], 0x0
0x56301ae4b990 <main+246> lea     rdi, [rip+0x128] # 0x56301ae4babf
0x56301ae4b997 <main+253> call    0x56301ae4b710 <puts@plt>
→ 0x56301ae4b99c <main+258> mov     rdx, QWORD PTR [rip+0x20067d] # 0x56301b04c020 <stdin@@GLIBC_2.2.5>
0x56301ae4b9a3 <main+265> lea     rax, [rbp-0x50]
0x56301ae4b9a7 <main+269> mov     esi, 0x40
0x56301ae4b9ac <main+274> mov     rdi, rax
0x56301ae4b9af <main+277> call    0x56301ae4b750 <fgets@plt>
0x56301ae4b9b4 <main+282> lea     rax, [rbp-0x50]

[#0] Id 1, Name: "jumpdrive", Stopped 0x56301ae4b99c in main (), reason: BREAKPOINT

[#0] 0x56301ae4b99c → main()

gef> x/50gx $rsp
0x7fff7768c40: 0x00000027ff000000 0xdeadbeef00000000
0x7fff7768c50: 0x000056301b1172a0 0x400911d14e3bcd36
0x7fff7768c60: 0x6161616161616161 0x6161616161616161
0x7fff7768c70: 0x6161616161616161 0x6161616161616161
0x7fff7768c80: 0x0000a61616161616 0x00007fbaf28b77e5
0x7fff7768c90: 0x0000000000000001 0x000056301ae4ba2d
0x7fff7768ca0: 0x00007fbaf29e8008 0x0000000000000000
0x7fff7768cb0: 0x000056301ae4b9e0 0x000056301ae4b790
0x7fff7768cc0: 0x00007fff7768db0 0x1872d4363bac1200
0x7fff7768cd0: 0x000056301ae4b9e0 0x00007fbaf281f1e3
0x7fff7768ce0: 0x0000000000000000 0x00007fff7768db8
0x7fff7768cf0: 0x0000000100040000 0x000056301ae4b89a
0x7fff7768d00: 0x0000000000000000 0x91806780803b026f
0x7fff7768d10: 0x000056301ae4b790 0x00007fff7768db0
0x7fff7768d20: 0x0000000000000000 0x0000000000000000
0x7fff7768d30: 0xc21fdca4ea3b026f 0xc295b74a10d5026f
0x7fff7768d40: 0x0000000000000000 0x0000000000000000
0x7fff7768d50: 0x0000000000000000 0x00007fff7768dc8
0x7fff7768d60: 0x00007fbaf2a2c190 0x00007fbaf2a0f131
0x7fff7768d70: 0x0000000000000000 0x0000000000000000
0x7fff7768d80: 0x000056301ae4b790 0x00007fff7768db0
0x7fff7768d90: 0x0000000000000000 0x000056301ae4b7ba
0x7fff7768da0: 0x00007fff7768da8 0x000000000000001c
0x7fff7768db0: 0x0000000000000001 0x00007fff776a341
0x7fff7768dc0: 0x0000000000000000 0x00007fff776a34d
gef> 
```

We can see that our flag is loaded into the stack.

## EXPLOITATION:

When we randomly pass a format string specifier ‘%10\$s’, we get a portion of the flag leaked. We can use a cyclic pattern as well or some know words, in order to find out which portion of the stack is being leaked. We can also make a loop in our exploit script to loop over various positions of the stack. However I did it manually by leaking portions of the flag. A POC exploit is added to this.

Writeup though.