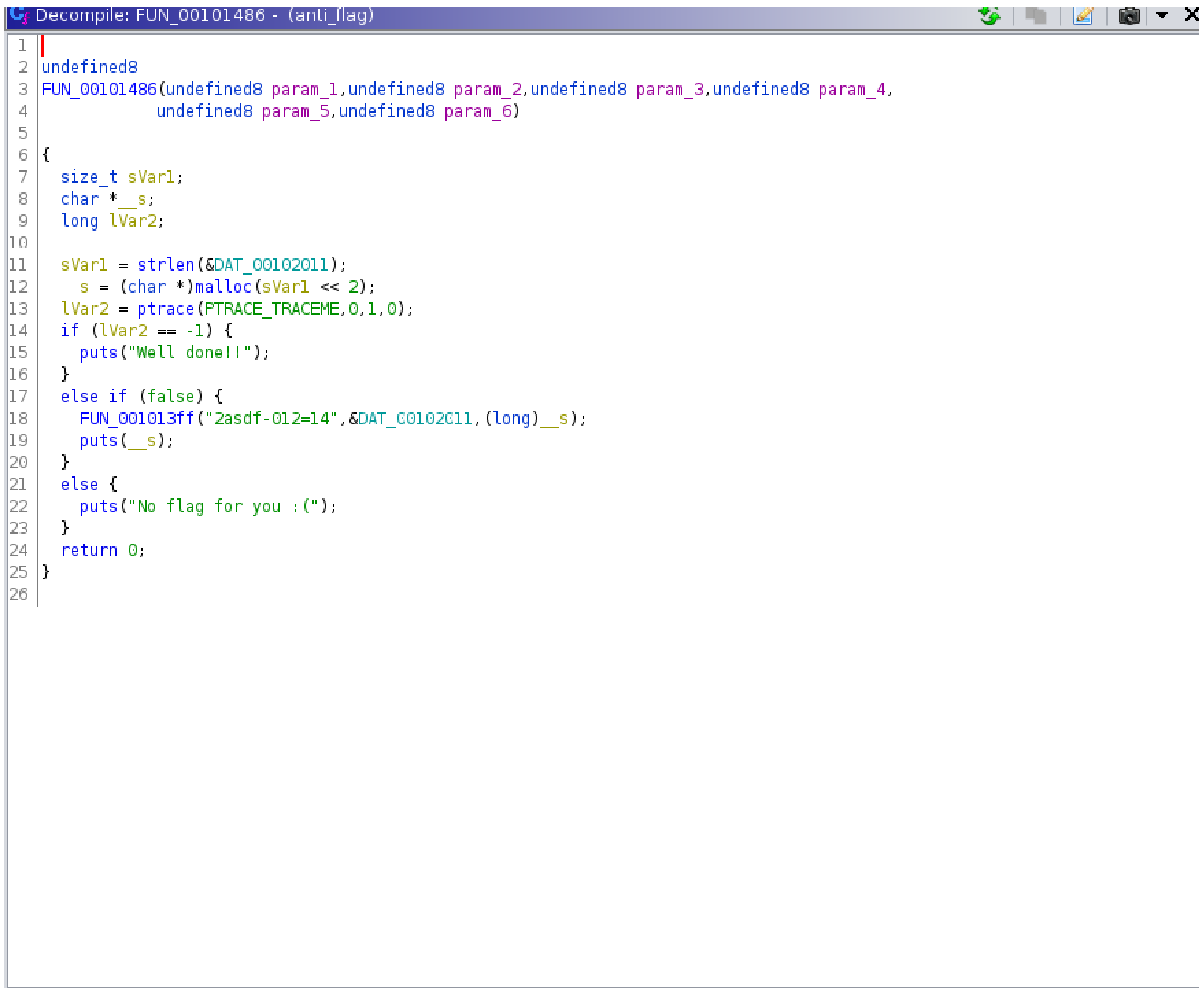
HTB Reverse Engineering: Anti Flag
Tools: Ghidra, GDB/GEF

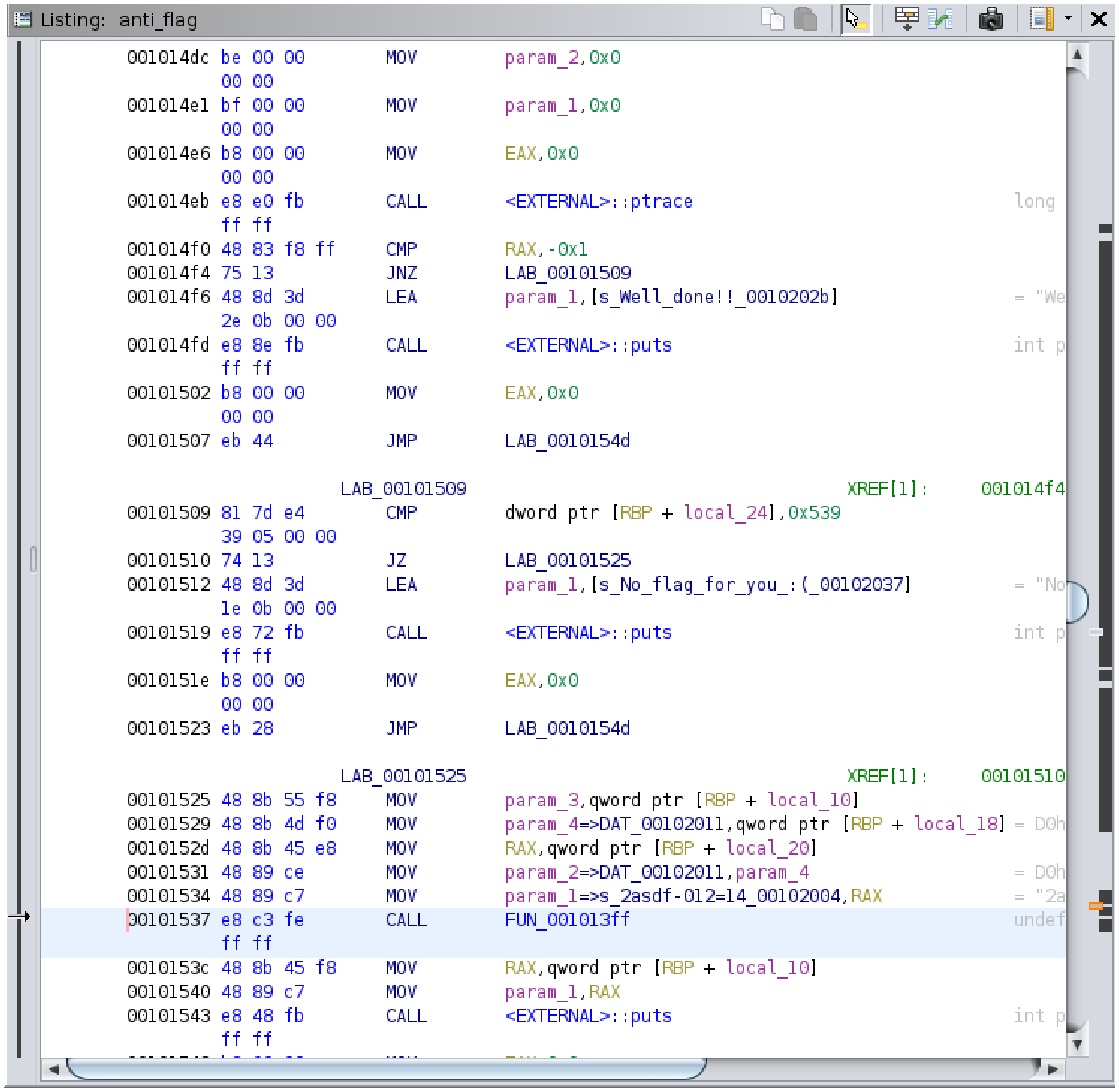
Using GDB and Ghidra we will debug the binary to find the flag.

Starting off we will open up the program in Ghidra and navigate to the entry function. Next in the taskbar at the top of Ghidra. Go to edit>tool options>decompiler>analysis and uncheck the eliminate unreachable code.This will allow us to see more in the decompiler.When you are done the decompile window should look like this.



```

undefined8
FUN_00101486(undefined8 param_1,undefined8 param_2,undefined8 param_3,undefined8 param_4,
            undefined8 param_5,undefined8 param_6)

{
  size_t sVar1;
  char *__s;
  long lVar2;

  sVar1 = strlen(&DAT_00102011);
  __s = (char *)malloc(sVar1 << 2);
  lVar2 = ptrace(PTRACE_TRACEME,0,1,0);
  if (lVar2 == -1) {
    puts("Well done!!");
  }
  else if (false) {
    FUN_001013ff("2asdf-012=14",&DAT_00102011,(long)__s);
    puts(__s);
  }
  else {
    puts("No flag for you :(");
  }
  return 0;
}

```

Now we can see line 17-20 and following that function we can see where our flag is hiding.



Listing: anti_flag

```
001014dc be 00 00       MOV       param_2,0x0
         00 00
001014e1 bf 00 00       MOV       param_1,0x0
         00 00
001014e6 b8 00 00       MOV       EAX,0x0
         00 00
001014eb e8 e0 fb       CALL      <EXTERNAL>::ptrace                          long
         ff ff
001014f0 48 83 f8 ff    CMP       RAX,-0x1
001014f4 75 13          JNZ       LAB_00101509
001014f6 48 8d 3d       LEA       param_1,[s_Well_done!!_0010202b]             = "We
         2e 0b 00 00
001014fd e8 8e fb       CALL      <EXTERNAL>::puts                            int p
         ff ff
00101502 b8 00 00       MOV       EAX,0x0
         00 00
00101507 eb 44          JMP       LAB_0010154d

                     LAB_00101509                              XREF[1]:     001014f4
00101509 81 7d e4       CMP       dword ptr [RBP + local_24],0x539
         39 05 00 00
00101510 74 13          JZ        LAB_00101525
00101512 48 8d 3d       LEA       param_1,[s_No_flag_for_you_:(_00102037]      = "No
         1e 0b 00 00
00101519 e8 72 fb       CALL      <EXTERNAL>::puts                            int p
         ff ff
0010151e b8 00 00       MOV       EAX,0x0
         00 00
00101523 eb 28          JMP       LAB_0010154d

                     LAB_00101525                              XREF[1]:     00101510
00101525 48 8b 55 f8    MOV       param_3,qword ptr [RBP + local_10]
00101529 48 8b 4d f0    MOV       param_4=>DAT_00102011,qword ptr [RBP + local_18] = D0h
0010152d 48 8b 45 e8    MOV       RAX,qword ptr [RBP + local_20]
00101531 48 89 ce       MOV       param_2=>DAT_00102011,param_4                = D0h
00101534 48 89 c7       MOV       param_1=>s_2asdf-012=14_00102004,RAX         = "2a
00101537 e8 c3 fe       CALL      FUN_001013ff                                undef
         ff ff
0010153c 48 8b 45 f8    MOV       RAX,qword ptr [RBP + local_10]
00101540 48 89 c7       MOV       param_1,RAX
00101543 e8 48 fb       CALL      <EXTERNAL>::puts                            int p
         ff ff
```

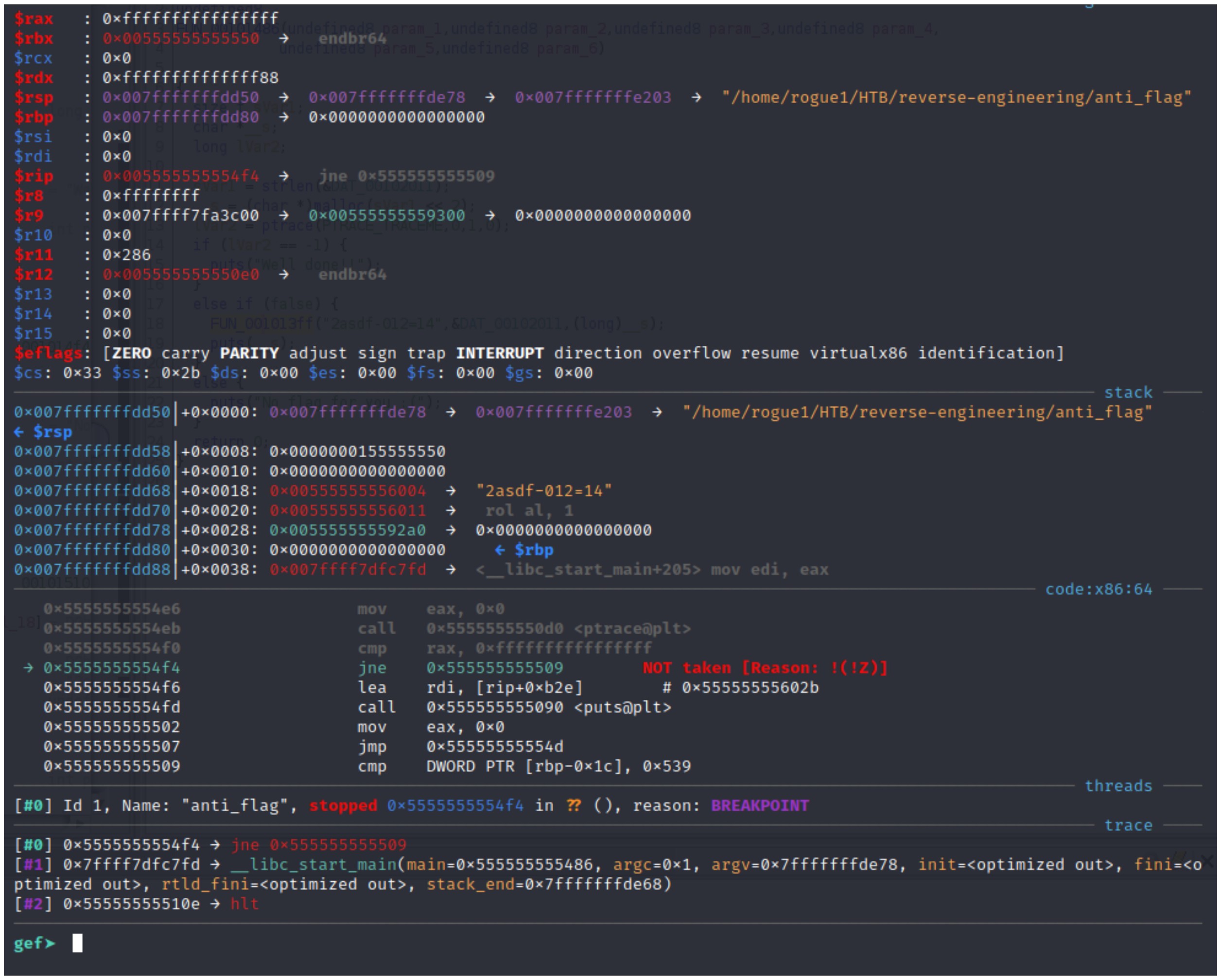
We want to get to address 1525 to read the flag but ptrace will not allow us. So we need to bypass ptrace by debugging with gdb. There are multiple methods of bypassing ptrace but I will only show this one.

$Gdb ./anti_flag
gef> starti (starts the binary at the 1st possible breakpoint)

Now we need to set a breakpoint just before the functions for the program start. We will do this by setting a breakpoint at an offset. In the picture above we are going to use the offset at 0x14f4, just before the LEA well done.

gef> pie breakpoint 0x14f4
gef> continue

Should look like this.



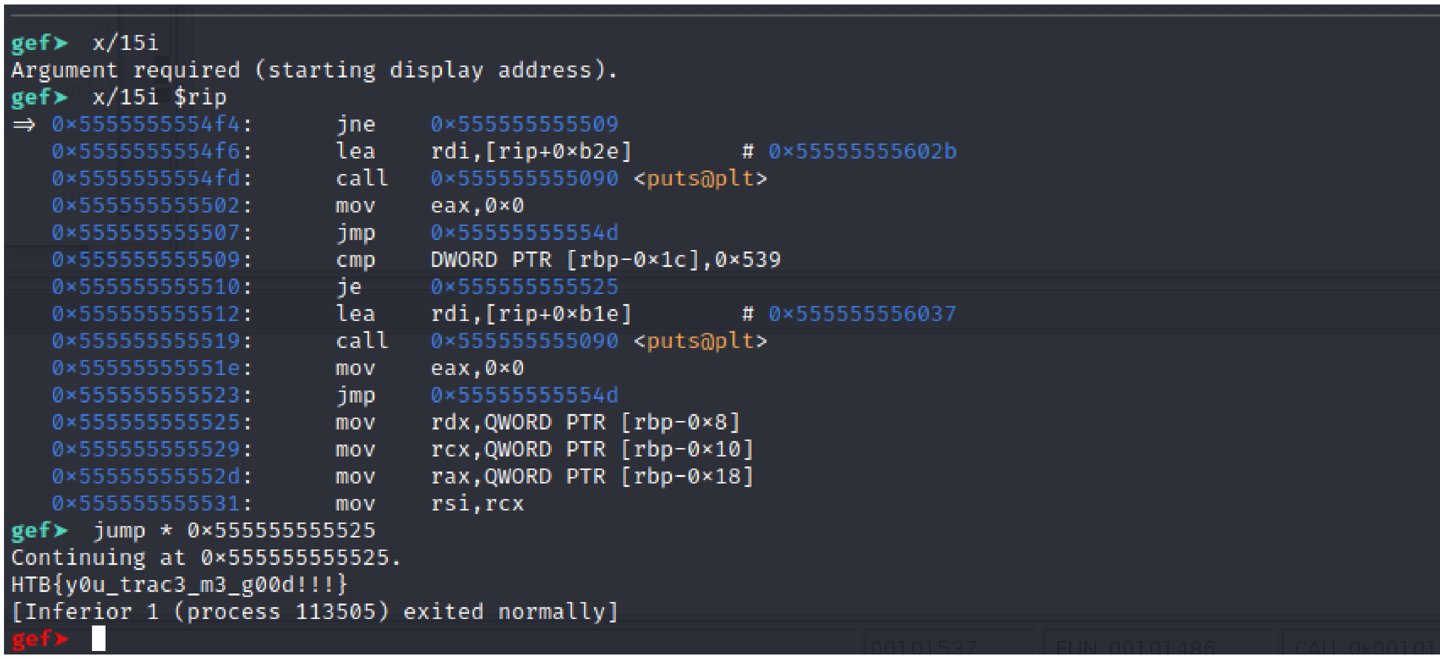
```
$rax   : 0×ffffffffffffffff
$rbx   : 0×00555555555550  →   endbr64
$rcx   : 0×0
$rdx   : 0×ffffffffffffff88
$rsp   : 0×007fffffffdd50  →  0×007fffffffde78  →  0×007fffffffe203  →  "/home/rogue1/HTB/reverse-engineering/anti_flag"
$rbp   : 0×007fffffffdd80  →  0×0000000000000000
$rsi   : 0×0
$rdi   : 0×0
$rip   : 0×005555555554f4  →   jne 0×555555555509
$r8    : 0×ffffffff
$r9    : 0×007ffff7fa3c00  →  0×00555555559300  →  0×0000000000000000
$r10   : 0×0
$r11   : 0×286
$r12   : 0×005555555550e0  →   endbr64
$r13   : 0×0
$r14   : 0×0
$r15   : 0×0
$eflags: [ZERO carry PARITY adjust sign trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0×33 $ss: 0×2b $ds: 0×00 $es: 0×00 $fs: 0×00 $gs: 0×00
──────────────────────────────────────────────────────────────────────────────────────────── stack ────
0×007fffffffdd50│+0×0000: 0×007fffffffde78  →  0×007fffffffe203  →  "/home/rogue1/HTB/reverse-engineering/anti_flag"
← $rsp
0×007fffffffdd58│+0×0008: 0×0000000155555550
0×007fffffffdd60│+0×0010: 0×0000000000000000
0×007fffffffdd68│+0×0018: 0×00555555556004  →  "2asdf-012=14"
0×007fffffffdd70│+0×0020: 0×00555555556011  →   rol al, 1
0×007fffffffdd78│+0×0028: 0×005555555592a0  →  0×0000000000000000
0×007fffffffdd80│+0×0030: 0×0000000000000000     ← $rbp
0×007fffffffdd88│+0×0038: 0×007ffff7dfc7fd  →  <__libc_start_main+205> mov edi, eax
─────────────────────────────────────────────────────────────────────────────────────── code:x86:64 ────
   0×5555555554e6                mov    eax, 0×0
   0×5555555554eb                call   0×5555555550d0 <ptrace@plt>
   0×5555555554f0                cmp    rax, 0×ffffffffffffffff
 → 0×5555555554f4                jne    0×555555555509        NOT taken [Reason: !(!Z)]
   0×5555555554f6                lea    rdi, [rip+0×b2e]        # 0×55555555602b
   0×5555555554fd                call   0×555555555090 <puts@plt>
   0×555555555502                mov    eax, 0×0
   0×555555555507                jmp    0×55555555554d
   0×555555555509                cmp    DWORD PTR [rbp-0×1c], 0×539
────────────────────────────────────────────────────────────────────────────────────────── threads ────
[#0] Id 1, Name: "anti_flag", stopped 0×5555555554f4 in ?? (), reason: BREAKPOINT
──────────────────────────────────────────────────────────────────────────────────────────── trace ────
[#0] 0×5555555554f4 → jne 0×555555555509
[#1] 0×7ffff7dfc7fd → __libc_start_main(main=0×555555555486, argc=0×1, argv=0×7fffffffde78, init=<optimized out>, fini=<o
ptimized out>, rtld_fini=<optimized out>, stack_end=0×7fffffffde68)
[#2] 0×55555555510e → hlt
───────────────────────────────────────────────────────────────────────────────────────────────────────
gef> 
```

Next we need the address for the flag. To get this address i will use the command.

gef> x/15i $rip

And now we can see the full address 0x555555555525 for the flag and it matches up with Ghidra’s 00101525 address. Next we set a jump command to skip the other functions and read the LAB_00101525.

gef> jump * 0x555555555525



```
gef➤  x/15i
Argument required (starting display address).
gef➤  x/15i $rip
⇒ 0x5555555554f4:      jne    0x555555555509
   0x5555555554f6:      lea    rdi,[rip+0xb2e]         # 0x55555555602b
   0x5555555554fd:      call   0x555555555090 <puts@plt>
   0x555555555502:      mov    eax,0x0
   0x555555555507:      jmp    0x55555555554d
   0x555555555509:      cmp    DWORD PTR [rbp-0x1c],0x539
   0x555555555510:      je     0x555555555525
   0x555555555512:      lea    rdi,[rip+0xb1e]         # 0x555555556037
   0x555555555519:      call   0x555555555090 <puts@plt>
   0x55555555551e:      mov    eax,0x0
   0x555555555523:      jmp    0x55555555554d
   0x555555555525:      mov    rdx,QWORD PTR [rbp-0x8]
   0x555555555529:      mov    rcx,QWORD PTR [rbp-0x10]
   0x55555555552d:      mov    rax,QWORD PTR [rbp-0x18]
   0x555555555531:      mov    rsi,rcx
gef➤  jump * 0x555555555525
Continuing at 0x555555555525.
HTB{y0u_trac3_m3_g00d!!!}
[Inferior 1 (process 113505) exited normally]
gef➤
```

That's all there is to it. There are some other methods but I found this one the simplest for me. The other had to do with using catch syscall ptrace. Theres also some other things you can try here.
https://seblau.github.io/posts/linux-anti-debugging