ANALYSIS OF THE BINARY:

We first run the binary and face a sementation 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:

Decompiler

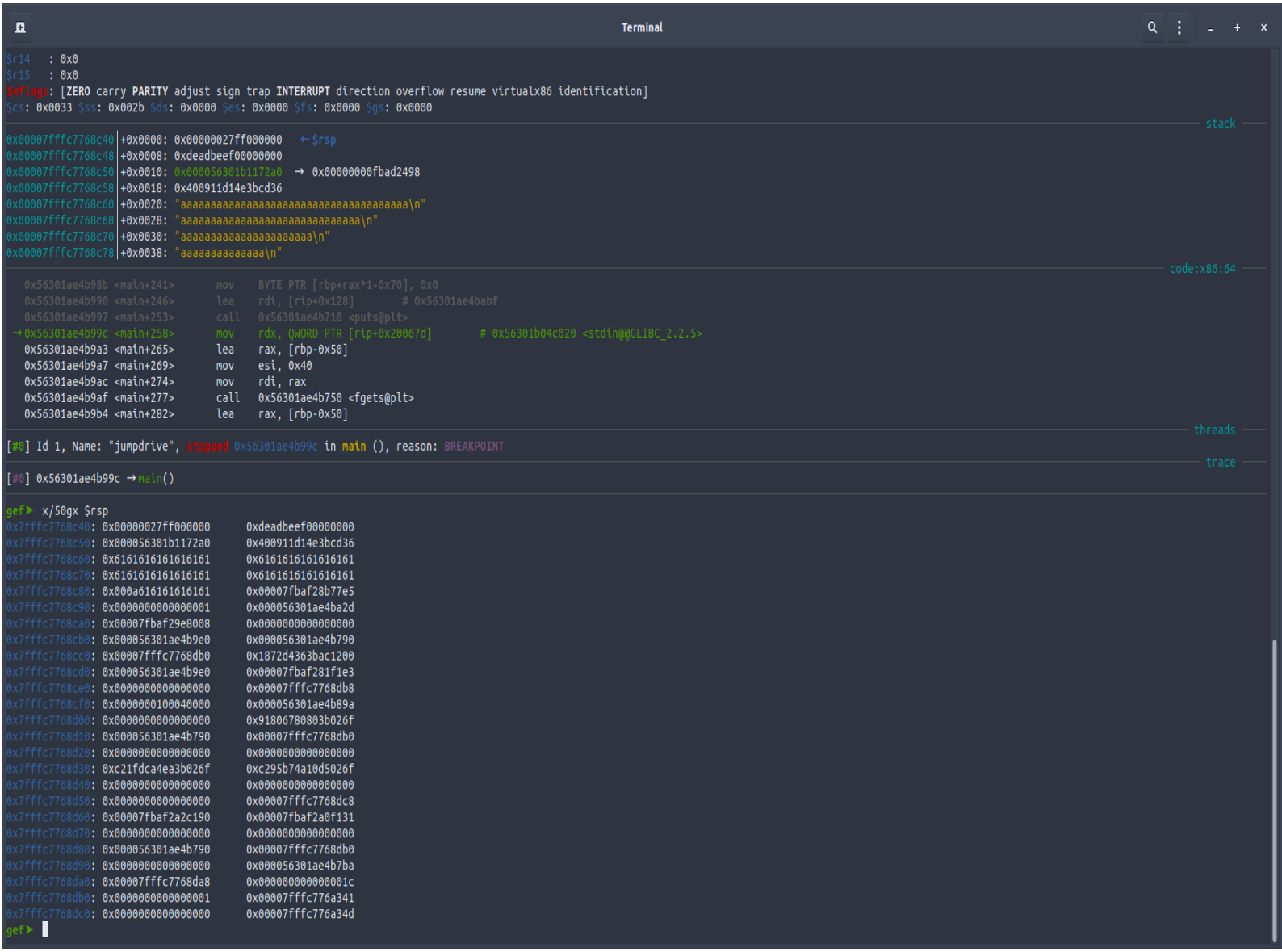
```
// WARNING: [r2ghidra] Failed to match type file* for variable stream to Decompiler type: Unknown type identifier file
// WARNING: [r2ghidra] Detected overlap for variable var_84h
// WARNING: [r2ghidra] Detected overlap for variable var_8ch
// WARNING: [r2ghidra] Detected overlap for variable var_8dh

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



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.