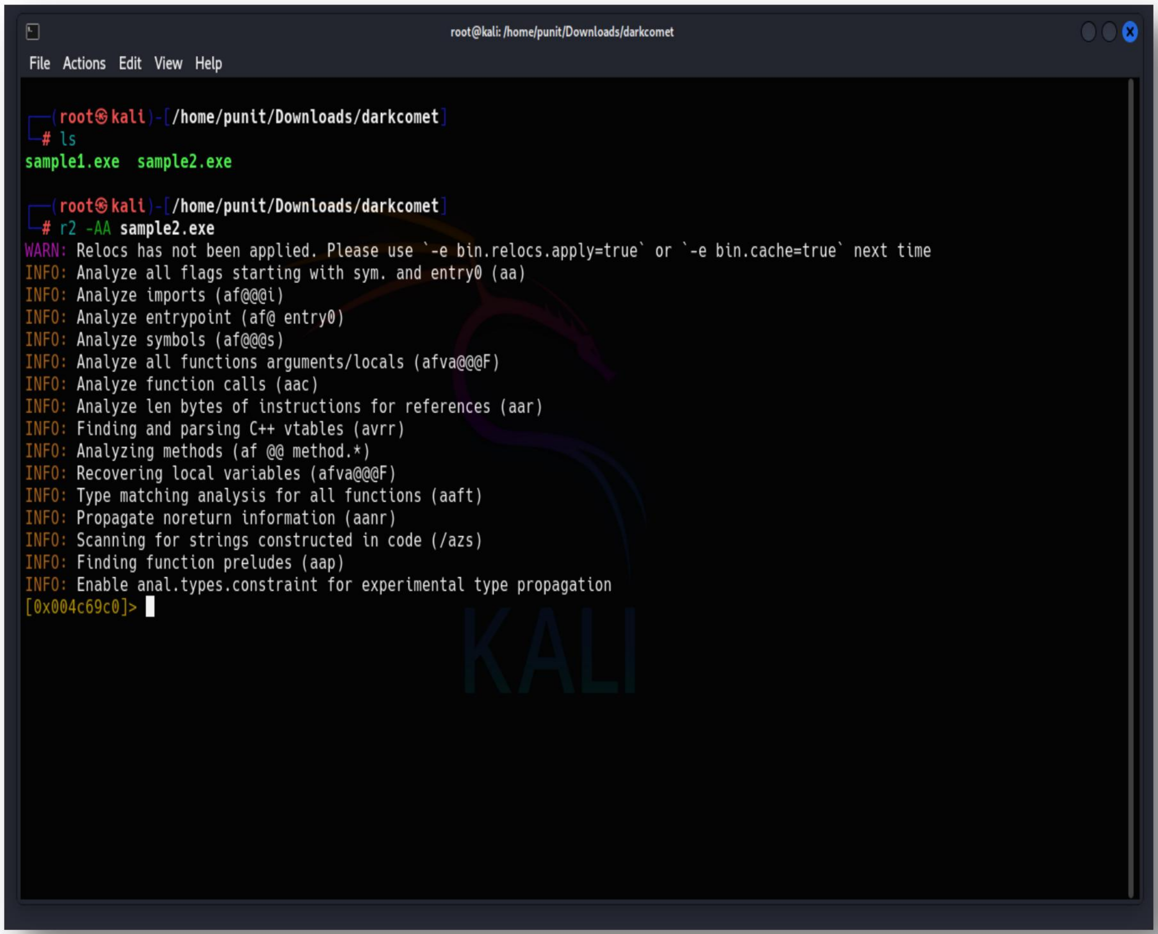
AIM: Reverse engineering malware using reverse engineering frameworks such as Radare2.

Step 1: Install Radare2 on your system using the following command:

- sudo apt install radare2

Step 2: Set up a secure analysis environment (VM or sandbox) to avoid executing malicious code.

- Loading the Malware Sample in Radare2
  1. Open the terminal and navigate to the folder containing the malware binary.
  2. Run the following command to load the binary into Radare2:
     - r2 -AA malware_sample.exe

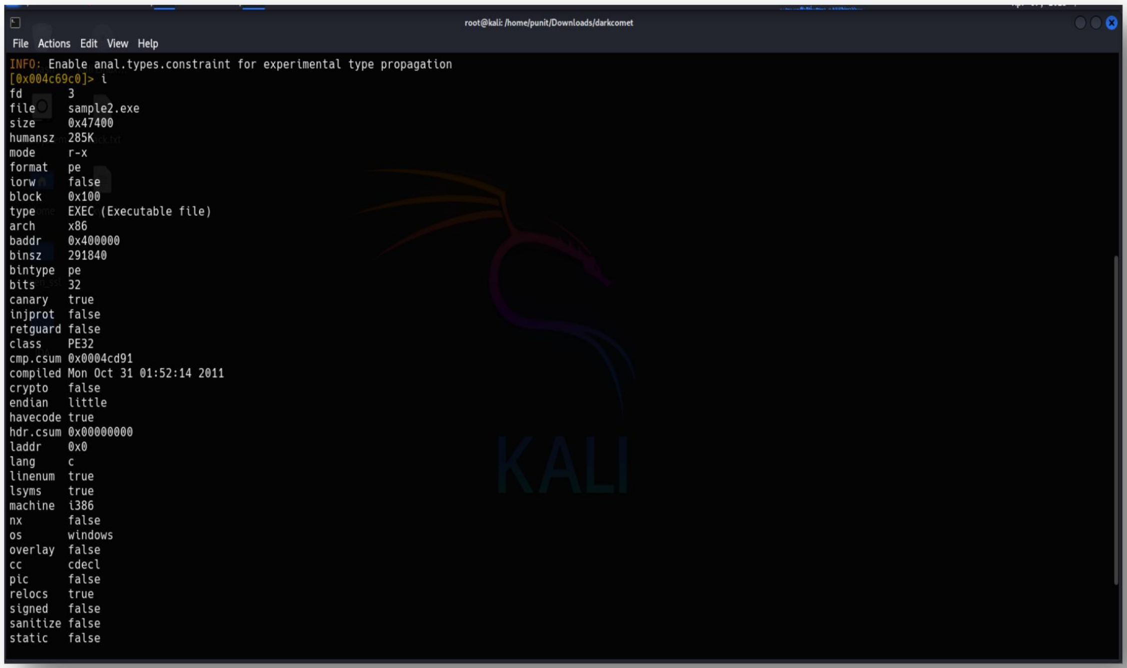

root@kali: /home/punit/Downloads/darkcomet
File Actions Edit View Help

```
┌──(root㉿kali)-[/home/punit/Downloads/darkcomet]
└─# ls
sample1.exe  sample2.exe

┌──(root㉿kali)-[/home/punit/Downloads/darkcomet]
└─# r2 -AA sample2.exe
WARN: Relocs has not been applied. Please use `-e bin.relocs.apply=true` or `-e bin.cache=true` next time
INFO: Analyze all flags starting with sym. and entry0 (aa)
INFO: Analyze imports (af@@@i)
INFO: Analyze entrypoint (af@ entry0)
INFO: Analyze symbols (af@@@s)
INFO: Analyze all functions arguments/locals (afva@@@F)
INFO: Analyze function calls (aac)
INFO: Analyze len bytes of instructions for references (aar)
INFO: Finding and parsing C++ vtables (avrr)
INFO: Analyzing methods (af @@ method.*)
INFO: Recovering local variables (afva@@@F)
INFO: Type matching analysis for all functions (aaft)
INFO: Propagate noreturn information (aanr)
INFO: Scanning for strings constructed in code (/azs)
INFO: Finding function preludes (aap)
INFO: Enable anal.types.constraint for experimental type propagation
[0x004c69c0]> 
```

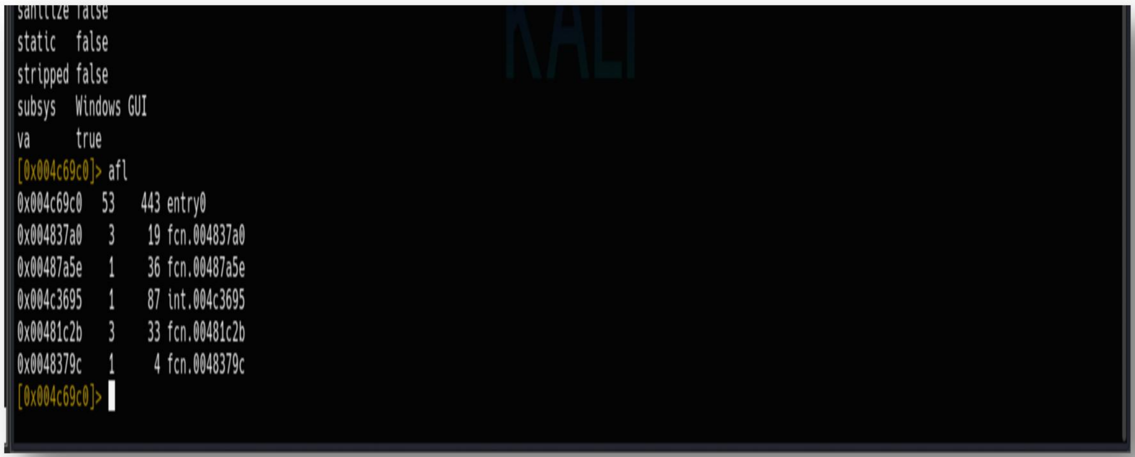
Step 3: Analyzing the Binary Structure

- Use the `i` command to display basic binary information:


root@kali: /home/punit/Downloads/darkcomet
File Actions Edit View Help

```
INFO: Enable anal.types.constraint for experimental type propagation
[0x004c69c0]> i
fd       3
file     sample2.exe
size     0x47400
humansz  285K
mode     r-x
format   pe
iorw     false
block    0x100
type     EXEC (Executable file)
arch     x86
baddr    0x400000
binsz    291840
bintype  pe
bits     32
canary   true
injprot  false
retguard false
class    PE32
cmp.csum 0x0004cd91
compiled Mon Oct 31 01:52:14 2011
crypto   false
endian   little
havecode true
hdr.csum 0x00000000
laddr    0x0
lang     c
linenum  true
lsyms    true
machine  i386
nx       false
os       windows
overlay  false
cc       cdecl
pic      false
relocs   true
signed   false
sanitize false
static   false
```

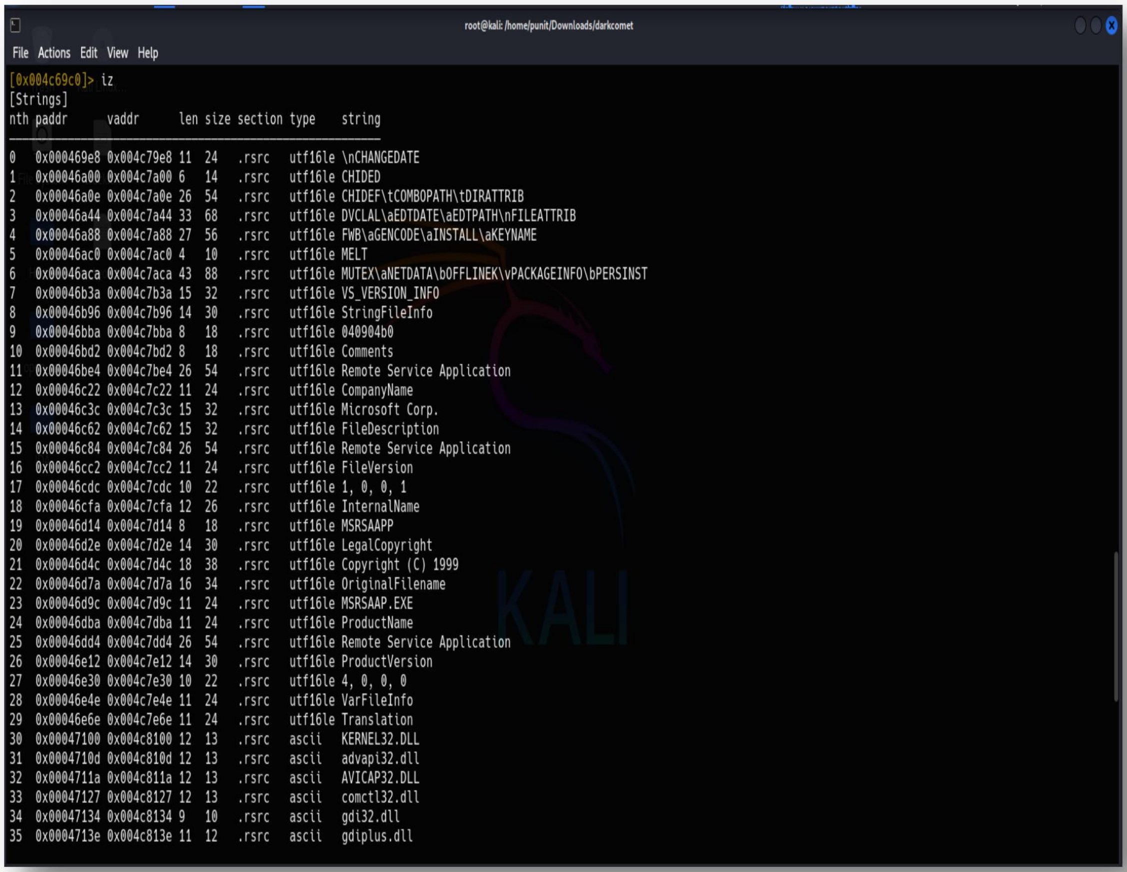
- List the functions detected in the binary using: **afl**



```
sanitize false
static   false
stripped false
subsys   Windows GUI
va       true
[0x004c69c0]> afl
0x004c69c0   53    443 entry0
0x004837a0    3     19 fcn.004837a0
0x00487a5e    1     36 fcn.00487a5e
0x004c3695    1     87 int.004c3695
0x00481c2b    3     33 fcn.00481c2b
0x0048379c    1      4 fcn.0048379c
[0x004c69c0]>
```

Step 4: Disassembling and Identifying Code Flow

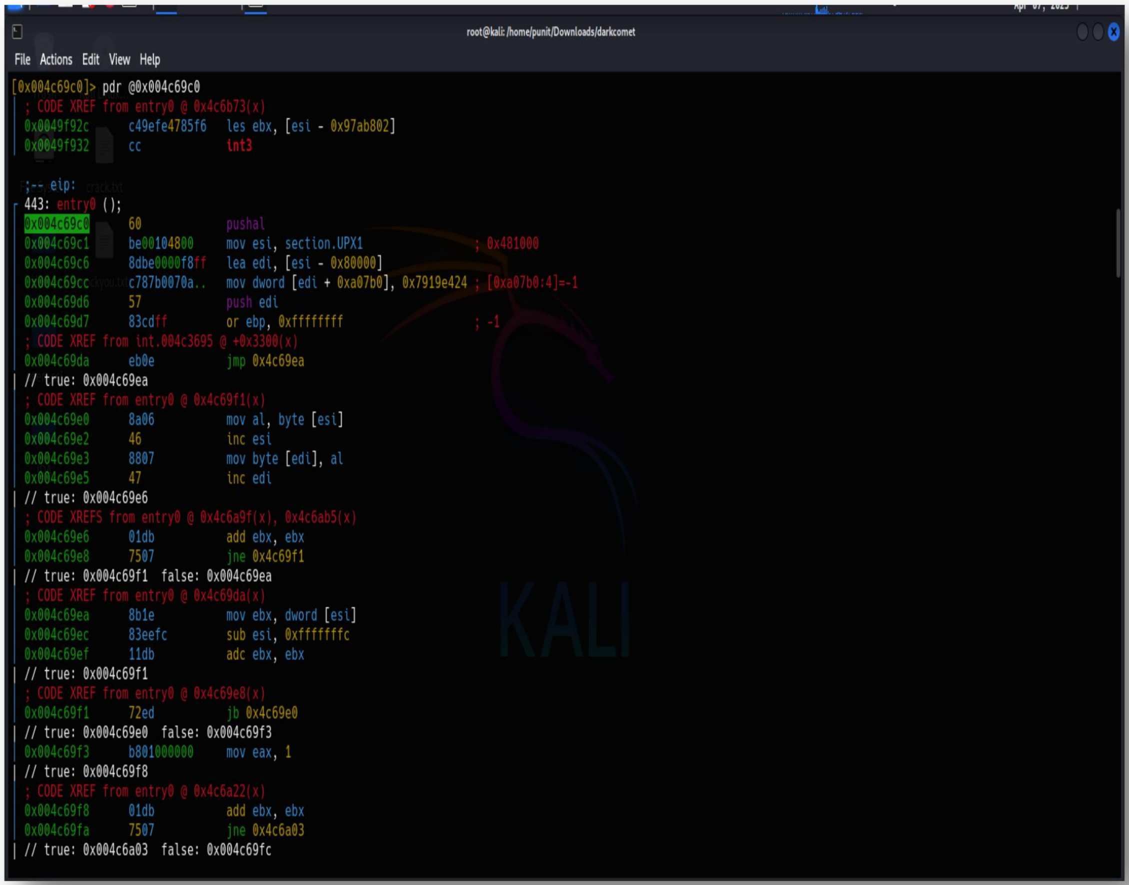
- Run the following command to extract readable strings: **iz**


root@kali: /home/punit/Downloads/darkcomet
File Actions Edit View Help

```
[0x004c69c0]> iz
[Strings]
nth paddr      vaddr      len size section type    string
―――――――――――――――――――――――――――――――――――――――――――――――――――――――――
0   0x000469e8 0x004c79e8 11  24   .rsrc   utf16le \nCHANGEDATE
1   0x00046a00 0x004c7a00 6   14   .rsrc   utf16le CHIDED
2   0x00046a0e 0x004c7a0e 26  54   .rsrc   utf16le CHIDEF\tCOMBOPATH\tDIRATTRIB
3   0x00046a44 0x004c7a44 33  68   .rsrc   utf16le DVCLAL\aEDTDATE\aEDTPATH\nFILEATTRIB
4   0x00046a88 0x004c7a88 27  56   .rsrc   utf16le FWB\aGENCODE\aINSTALL\aKEYNAME
5   0x00046ac0 0x004c7ac0 4   10   .rsrc   utf16le MELT
6   0x00046aca 0x004c7aca 43  88   .rsrc   utf16le MUTEX\aNETDATA\bOFFLINEK\vPACKAGEINFO\bPERSINST
7   0x00046b3a 0x004c7b3a 15  32   .rsrc   utf16le VS_VERSION_INFO
8   0x00046b96 0x004c7b96 14  30   .rsrc   utf16le StringFileInfo
9   0x00046bba 0x004c7bba 8   18   .rsrc   utf16le 040904b0
10  0x00046bd2 0x004c7bd2 8   18   .rsrc   utf16le Comments
11  0x00046be4 0x004c7be4 26  54   .rsrc   utf16le Remote Service Application
12  0x00046c22 0x004c7c22 11  24   .rsrc   utf16le CompanyName
13  0x00046c3c 0x004c7c3c 15  32   .rsrc   utf16le Microsoft Corp.
14  0x00046c62 0x004c7c62 15  32   .rsrc   utf16le FileDescription
15  0x00046c84 0x004c7c84 26  54   .rsrc   utf16le Remote Service Application
16  0x00046cc2 0x004c7cc2 11  24   .rsrc   utf16le FileVersion
17  0x00046cdc 0x004c7cdc 10  22   .rsrc   utf16le 1, 0, 0, 1
18  0x00046cfa 0x004c7cfa 12  26   .rsrc   utf16le InternalName
19  0x00046d14 0x004c7d14 8   18   .rsrc   utf16le MSRSAAPP
20  0x00046d2e 0x004c7d2e 14  30   .rsrc   utf16le LegalCopyright
21  0x00046d4c 0x004c7d4c 18  38   .rsrc   utf16le Copyright (C) 1999
22  0x00046d7a 0x004c7d7a 16  34   .rsrc   utf16le OriginalFilename
23  0x00046d9c 0x004c7d9c 11  24   .rsrc   utf16le MSRSAAP.EXE
24  0x00046dba 0x004c7dba 11  24   .rsrc   utf16le ProductName
25  0x00046dd4 0x004c7dd4 26  54   .rsrc   utf16le Remote Service Application
26  0x00046e12 0x004c7e12 14  30   .rsrc   utf16le ProductVersion
27  0x00046e30 0x004c7e30 10  22   .rsrc   utf16le 4, 0, 0, 0
28  0x00046e4e 0x004c7e4e 11  24   .rsrc   utf16le VarFileInfo
29  0x00046e6e 0x004c7e6e 11  24   .rsrc   utf16le Translation
30  0x00047100 0x004c8100 12  13   .rsrc   ascii   KERNEL32.DLL
31  0x0004710d 0x004c810d 12  13   .rsrc   ascii   advapi32.dll
32  0x0004711a 0x004c811a 12  13   .rsrc   ascii   AVICAP32.DLL
33  0x00047127 0x004c8127 12  13   .rsrc   ascii   comctl32.dll
34  0x00047134 0x004c8134 9   10   .rsrc   ascii   gdi32.dll
35  0x0004713e 0x004c813e 11  12   .rsrc   ascii   gdiplus.dll
```

Step 5: Disassembling and Identifying Code Flow

- Open the disassembler view using: **pdr @entry**



root@kali: /home/punit/Downloads/darkcomet
File Actions Edit View Help

```
[0x004c69c0]> pdr @0x004c69c0
| ; CODE XREF from entry0 @ 0x4c6b73(x)
| 0x0049f92c      c49efe4785f6   les ebx, [esi - 0x97ab802]
| 0x0049f932      cc             int3

  ;-- eip:
┌ 443: entry0 ();
│ 0x004c69c0      60             pushal
│ 0x004c69c1      be00104800     mov esi, section.UPX1              ; 0x481000
│ 0x004c69c6      8dbe0000f8ff   lea edi, [esi - 0x80000]
│ 0x004c69cc      c787b0070a..   mov dword [edi + 0xa07b0], 0x7919e424 ; [0xa07b0:4]=-1
│ 0x004c69d6      57             push edi
│ 0x004c69d7      83cdff         or ebp, 0xffffffff                 ; -1
│ ; CODE XREF from int.004c3695 @ +0x3300(x)
│ 0x004c69da      eb0e           jmp 0x4c69ea
| // true: 0x004c69ea
│ ; CODE XREF from entry0 @ 0x4c69f1(x)
│ 0x004c69e0      8a06           mov al, byte [esi]
│ 0x004c69e2      46             inc esi
│ 0x004c69e3      8807           mov byte [edi], al
│ 0x004c69e5      47             inc edi
| // true: 0x004c69e6
│ ; CODE XREFS from entry0 @ 0x4c6a9f(x), 0x4c6ab5(x)
│ 0x004c69e6      01db           add ebx, ebx
│ 0x004c69e8      7507           jne 0x4c69f1
| // true: 0x004c69f1  false: 0x004c69ea
│ ; CODE XREF from entry0 @ 0x4c69da(x)
│ 0x004c69ea      8b1e           mov ebx, dword [esi]
│ 0x004c69ec      83eefc         sub esi, 0xfffffffc
│ 0x004c69ef      11db           adc ebx, ebx
| // true: 0x004c69f1
│ ; CODE XREF from entry0 @ 0x4c69e8(x)
│ 0x004c69f1      72ed           jb 0x4c69e0
| // true: 0x004c69e0  false: 0x004c69f3
│ 0x004c69f3      b801000000     mov eax, 1
| // true: 0x004c69f8
│ ; CODE XREF from entry0 @ 0x4c6a22(x)
│ 0x004c69f8      01db           add ebx, ebx
│ 0x004c69fa      7507           jne 0x4c6a03
| // true: 0x004c6a03  false: 0x004c69fc
```

Conclusion:

By following these steps, you can successfully reverse engineer a malware sample using Radare2. Reverse engineering helps security professionals understand malware behavior, detect threats early, and develop strategies to defend against cyberattacks. Always remember to work in a secure and isolated environment to avoid any risk of accidental execution of the malware.