- **Name**: Iskander Nafikov
- **E-mail**: i.nafikov@innopolis.university
- **GitHub**: https://github.com/iskanred
- **Username**: `iskanred` / `i.nafikov`
- **Hostname**: `lenovo` / `macbook-KN70WX2HPH`

Understanding assembly

1. Preparation

> **Task description**
>
> a. You can use any debugger/disassembler as long as it supports the given task architecture (32bit or 64bit)
> b. Try to stay away from decompilers, as this will help you to better understand the nature of your task based on assembly only, remember in real-world tasks you will have to deal with much larger binaries.
> c. **IMPORTANT** : Please check what each binary does before running it (don't trust us 😈)
> d. Check some writeups about some CTF to see what you should/shouldn't include in your report
> e. Try to do the lab in a Linux VM, as you might need to disable ASLR

2. Theory

---

a.

> **Task description**
>
> a. What is ASLR, and why do we need it?

Description

- **ASLR** (Address Space Layout Randomization) is a security technique used in modern operating systems to protect against certain types of attacks, such as buffer overflow attacks.
- When you run a program on your computer, the operating system allocates memory for it, where it can store its data and execute commands. In traditional systems, this memory would always be located at the same place. For example, the program's code would start at a specific address, and its data and other important components would have predictable addresses.
- If an attacker knows these addresses, they might exploit a vulnerability to manipulate the program's execution flow and run their own code.
- ASLR addresses this problem by randomly changing the locations of different parts of the program's memory (such as code, data, and stack) each time it starts. This makes it more difficult for attackers to guess where the needed code is located.

Example

- **Without ASLR**:
  - The program always loads at address `0x400000`.
  - The attacker knows that if they can input specific commands, they can redirect execution to this address to run their malicious code.
- **With ASLR**:
  - Each time the program runs, it may load at address `0x7A6000` the first time, and then at `0x5B9000` the next time.
  - The attacker does not know in advance where the code is located and cannot simply redirect the program execution to that address.

b.

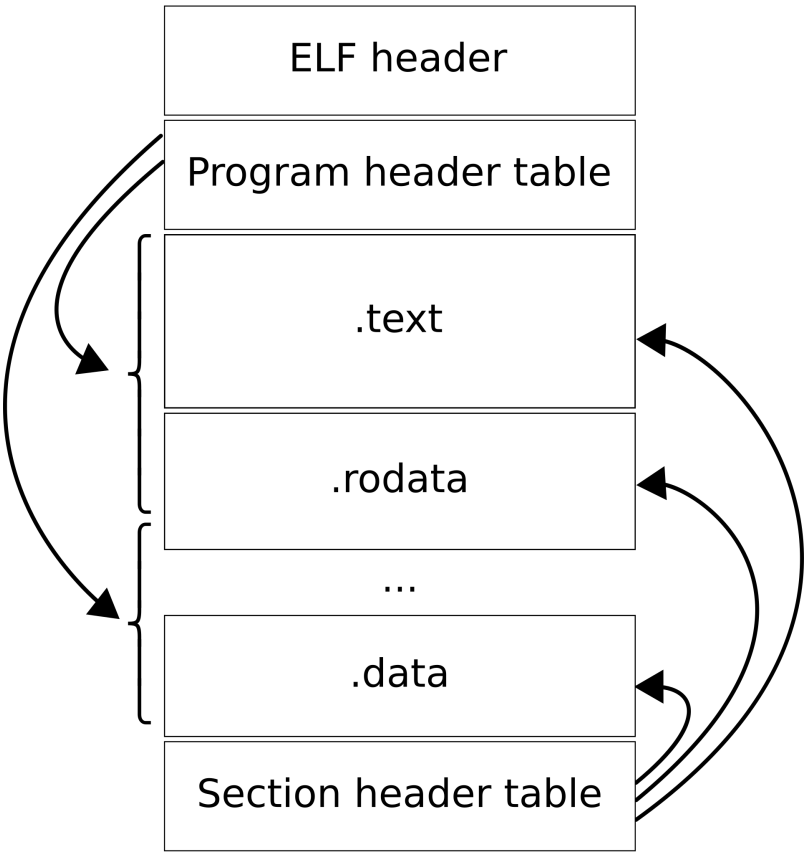
Task description

b. What kind of file did you receive (which arch? 32bit or 64bit?)?

- `sample32`

```
iskanred@lenovo:~/Study/iu-ot-course/lab-01/task-3$ file sample32
sample32: ELF 32-bit LSB pie executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 3.2.0, B
uildID[sha1]=14e300cb9d36dfdb6b23833164ecb1c1339d814c, with debug_info, not stripped
iskanred@lenovo:~/Study/iu-ot-course/lab-01/task-3$ binwalk -v sample32

Scan Time:     2025-03-30 16:18:05
Target File:   /home/iskanred/Study/iu-ot-course/lab-01/task-3/sample32
MD5 Checksum:  8087e213bc03bb7acf9e50764de3ded3
Signatures:    411

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             ELF, 32-bit LSB shared object, Intel 80386, version 1 (SYSV)
5411          0x1523          Unix path: /usr/lib/gcc/x86_64-linux-gnu/7/include
6227          0x1853          Unix path: /home/syscaller/innopolis/AS/lab3

iskanred@lenovo:~/Study/iu-ot-course/lab-01/task-3$
```

  - **File name**: `sample32`
  - **File format**: ELF (Executable and Linkable Format) is a format for executable files in Unix-like systems



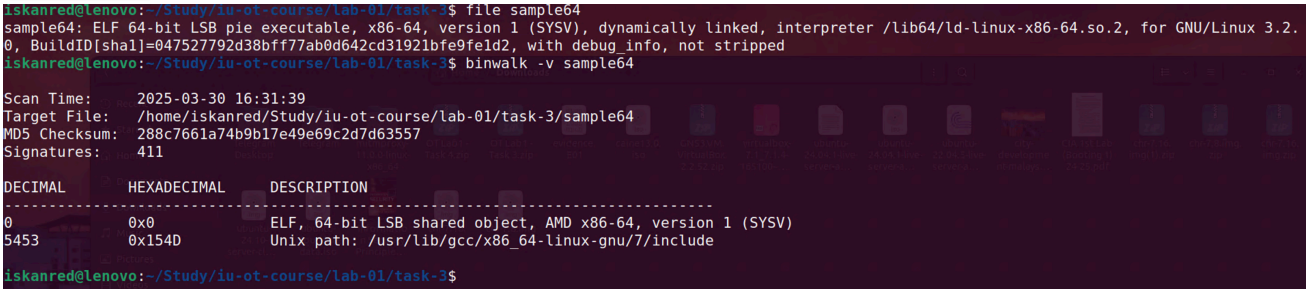
  - **Architecture type**: 32-bit
  - **Bit order** : LSB (Least Significant Byte first) which is default for Intel processors
  - **PIE**: PIE (Position Independent Executable) enabled which means the executable can be loaded to any place in memory. That is important because PIE is required for ASLR:

    > [1] If a program is compiled without PIE its text and data sections cannot be relocated in memory, however, ASLR can be applied to the stack, heap, and dynamic libraries that it uses, such as libc.

  - **Architecture**: Intel 80386 (i386)
  - **ELF version**: 1 which is compatible with System V

- **Linking**: Dynamic which means the load of necessary libraries into memory occurs during runtime rather than including it into the executable in compile-time
- **Interpreter**: `/lib/ld-linux.so.2` which is necessary for dynamic linking during runtime
- **OS version**: GNU/Linux 3.2.0 is a minimal version of OS to be able to run the executable
- **Build ID**: `14e300cb9d36dfdb6b23833164ecb1c1339d814c` which is SHA1 hash
- **Debug info**: Present. This option tells that debug info is present inside the executable.
- **Stripped**: No. This option tells whether the debug info and other associated metadata is absent inside an executable which makes debugging and reverse engineering harder.
- **Dependencies**:
  - `/usr/lib/gcc/x86_64-linux-gnu/7/include` : This includes the standard libraries used by programs compiled with GCC compiler version 7. The offset of this section is `0x1523` .
  - `/home/syscaller/innopolis/AS/lab3` : This includes the user-defined libraries. The offset of this section is `0x1853`

- **`sample64`**



  - **File name**: `sample64`
  - **File format**: ELF (Executable and Linkable Format)
  - **Architecture type**: 64-bit
  - **Bit order** : LSB (Least Significant Byte first)
  - **PIE**: PIE (Position Independent Executable) enabled.
  - **Architecture**: x86-64
  - **ELF version**: 1 which is compatible with System V
  - **Linking**: Dynamic
  - **Interpreter**: `/lib/ld-linux-x86-64.so.2` which is necessary for dynamic linking during runtime.
  - **OS version**: GNU/Linux 3.2.0
  - **Build ID**: `047527792d38bff77ab0d642cd31921bfe9fe1d2`
  - **Debug info**: Present
  - **Stripped**: No

- **Dependencies**:
  - `/usr/lib/gcc/x86_64-linux-gnu/7/include` : This includes the standard libraries used by programs compiled with GCC compiler version 7. The offset of this section is `0x1523` .

- `sample64-2`

  - **File name**: `sample64-2`
  - **File format**: ELF (Executable and Linkable Format)
  - **Architecture type**: 64-bit
  - **Bit order** : LSB (Least Significant Byte first)
  - **PIE**: PIE (Position Independent Executable) enabled.
  - **Architecture**: x86-64
  - **ELF version**: 1 which is compatible with System V
  - **Linking**: Dynamic
  - **Interpreter**: `/lib/ld-linux-x86-64.so.2` which is necessary for dynamic linking during runtime.
  - **OS version**: GNU/Linux 3.2.0
  - **Build ID**: `cb3d8fd741fd67fbdcf029696261b95faa9fd513`
  - **Debug info**: Absent ❗
  - **Stripped**: No
  - **Dependencies**: No dependencies

c.

> **Task description**
>
> c. What do stripped binaries mean?

- A **stripped executable** is one from which debug symbols and non-essential information have been removed. This process reduces the file size and removes information that isn't necessary for the executable to run. Stripping is often done to protect the source code from reverse engineering or to make the executable smaller for distribution.
- Characteristics of a Stripped Executable:

- **No Debug Symbols**: The debug information (like variable names, function names, line numbers, and other metadata) is removed, making it harder to perform debugging or reverse engineering.
- **Smaller Size**: Since the extra information is removed, the file size is typically smaller.
- **Difficult to Debug**: Without debug symbols, it can be challenging to perform post-mortem analysis or debugging with tools like GDB.

d.

> **Task description**
>
> d. What are GOT and PLT?

- **GOT (Global Offset Table)**: GOT is a table that is used to store the addresses of global variables and functions in dynamically compiled executable files. It allows you to support working with dynamic libraries, since the addresses of functions and variables in such libraries can change every time the program is started.
- **PLT (Procedure Linkage Table)**: A PLT is a table that is used to support linking function calls in dynamically compiled executable files. PLT helps you manage function calls from dynamic libraries. It contains instructions for function calls, providing a mechanism for accessing dynamically loaded functions.
- **Description of a process:**
  When a program calls a function from a dynamic library (for example, printf), it makes a call to PLT. The first function call goes through PLT, which finds the corresponding address in GOT. After the first call, the data in GOT is updated so that the next time the function can be called directly, without having to access the PLT.
- **Why do we need PLT?**
  - **Lazy Binding**: PLT allows you to use the "lazy loading" mechanism. This means that a function from a dynamic library is not loaded until it is called for the first time. If the program tried to access GOT directly without PLT, it would require static resolution of the addresses of all functions at the loading stage, which could lead to performance loss.Example: If your program uses many dynamic libraries, but calls only a few functions, PLT allows you to load addresses only as needed. This can reduce program loading time and memory consumption.
  - **Ensuring compatibility**: PLT provides an abstraction layer between program code and dynamic libraries. Each function called from the dynamic library can perform additional actions (for example, initialization or verification) before the actual function call. PLT can handle calls to both functions, both from dynamic libraries and from static ones.

- **Address updates and changes**:
  When the program starts, the addresses of functions in dynamic libraries may not be known, and PLT allows you to first send control to a single point, which then determines the correct address in GOT. This is critical in cases where a library version with changed addresses is used, or if the same library (or function) is called from multiple processes.

e.

> **Task description**
>
> e. How can the debugger insert a breakpoint in the debugged binary/application?

- First, debugger can insert breakpoints in two different ways
  1. **Software** approach which is old but actively used
  2. **Hardware** approach which is supported by all modern processors

Software breakpoints

Description

- Software breakpoints are instructions that are inserted into the program code to stop its execution at a specific point. They are implemented by replacing a specific instruction (or its first byte) with a special interrupt instruction.

Algorithm

1. **Replacing the Instructions**: When you set a software breakpoint in the debugger, it changes the program code. For example, the standard instruction can be replaced with INT 3 (for x86 architecture) or a similar instruction.
2. **Interrupt Call**: When the program executes this command, control is transferred to the debugger, which stops program execution and allows the developer to examine the status.

Example

```
mov eax, 42         ; Instruction
int 3               ; Inserted breakpoint interruption
mov ebx, 7          ; This instruction will not be executed until the
debugger returns execution flow to the program
```

Hardware breakpoints

Description

- Hardware breakpoints use special processor capabilities to stop program execution without changing the code. This means that instead of changing instructions, the processor remembers the address where the stop should occur.

Algorithm

1. **Using Special Registers**: Most modern processors have special registers that can store addresses for breakpoints. For example, the registers `DR0–DR3` can be used in the x86 architecture.
2. **Interception of execution**: If the program tries to execute code at the address that is stored in this special register, an interrupt is triggered and execution stops.

Comparison

	Software	Hardware
Change of code	Yes, program's source code is changed	No, program's source remains unchanged
Number of breakpoints	Almost unlimited, depends on memory size	Limited to the numbers of special registers (typically, 4)
Performance impact	Can impact on performance due to code change	Has no impact
Usage	Useful for dymaic injection of breakpoints	Useful for low-level code and system programming

3. Disassembly

---

a.

Task description

a. Disable ASLR using the following command `sudo sysctl -w kernel.randomize_va_space=0`

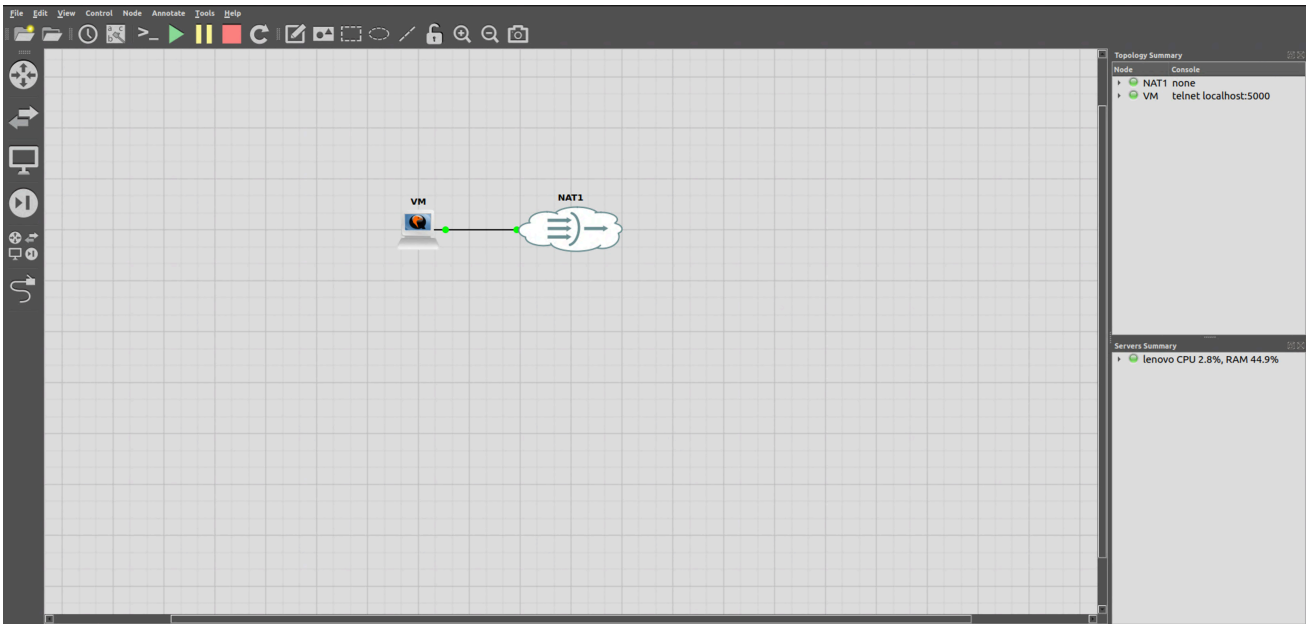
Description

The variable `kernel.randomize_va_space` can take the following values:

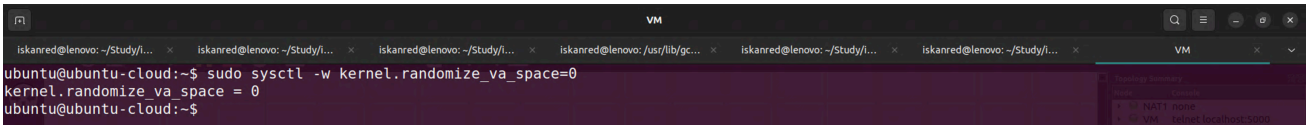
- `0` : Disables address space randomization. In this mode, the addresses where memory segments are located (for example, stack, heap, dynamically loaded libraries) remain fixed. This increases the predictability of data placement in memory, which can be vulnerable to attacks.
- `1` : Enables randomization for stacks and dynamically loaded libraries, but not for precomputed addresses. This means that the stack and library addresses will change, making attacks more difficult.
- `2` : Enables full randomization of the address space. All memory segments (stack, heap, libraries, and other important data) will be randomly allocated to the process's memory each time it is started. This provides maximum protection against predictable attacks.
- `3` : In some versions of Linux, the value 3 may be available, which represents a larger and stricter security measure. However, support for this value may vary depending on the kernel version and distribution.

Reproduction

- I have created a new Ubuntu Cloud VM in GNS3



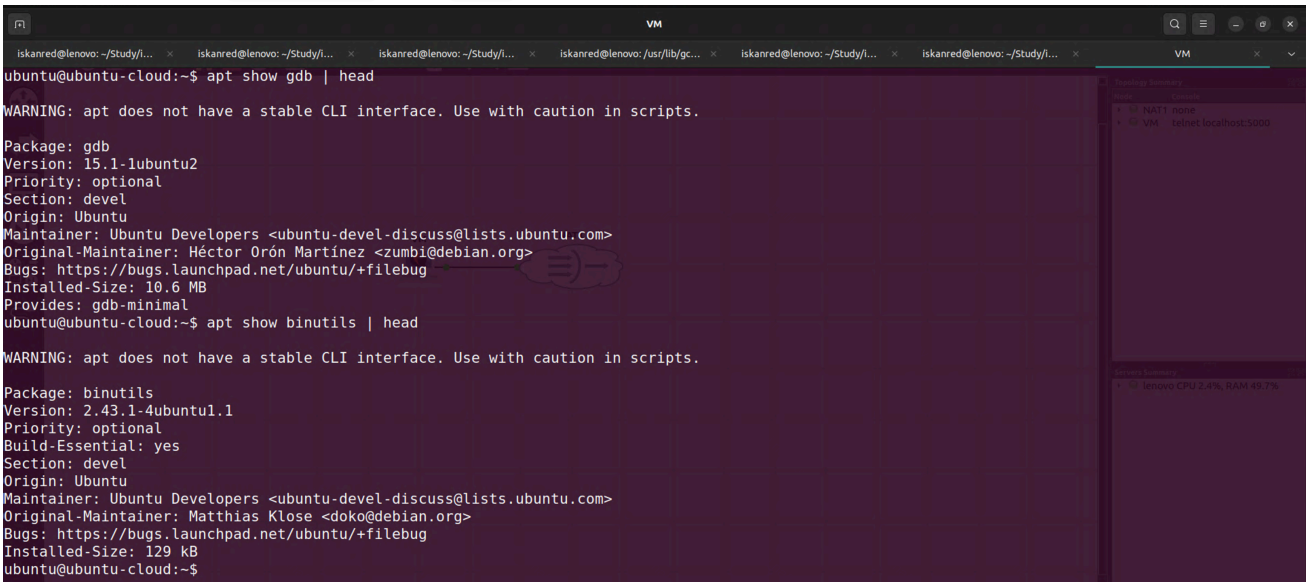
- I have disabled ASLR on my VM



b.

> **Task description**
>
> b. Load the binaries **from the Task 3 folder** into a disassembler/debugger

- I have installed `binutils` and `gdb` to my VM



Disassembly

- I used `objdump -d` as a disassembler and `gdb` as a debugger
- So below is the output of `objdumd -d` for the executables
  - **`sample32`**

```
sample32:     file format elf32-i386


Disassembly of section .init:

00000398 <_init>:
 398:   53                      push   %ebx
 399:   83 ec 08                sub    $0x8,%esp
 39c:   e8 af 00 00 00          call   450 <__x86.get_pc_thunk.bx>
 3a1:   81 c3 33 1c 00 00       add    $0x1c33,%ebx
 3a7:   8b 83 20 00 00 00       mov    0x20(%ebx),%eax
 3ad:   85 c0                   test   %eax,%eax
 3af:   74 05                   je     3b6 <_init+0x1e>
```

```
 3b1:   e8 52 00 00 00          call   408 <__gmon_start__@plt>
 3b6:   83 c4 08                add    $0x8,%esp
 3b9:   5b                      pop    %ebx
 3ba:   c3                      ret

Disassembly of section .plt:

000003c0 <.plt>:
 3c0:   ff b3 04 00 00 00       push   0x4(%ebx)
 3c6:   ff a3 08 00 00 00       jmp    *0x8(%ebx)
 3cc:   00 00                   add    %al,(%eax)
    ...

000003d0 <printf@plt>:
 3d0:   ff a3 0c 00 00 00       jmp    *0xc(%ebx)
 3d6:   68 00 00 00 00          push   $0x0
 3db:   e9 e0 ff ff ff          jmp    3c0 <.plt>

000003e0 <gets@plt>:
 3e0:   ff a3 10 00 00 00       jmp    *0x10(%ebx)
 3e6:   68 08 00 00 00          push   $0x8
 3eb:   e9 d0 ff ff ff          jmp    3c0 <.plt>

000003f0 <__libc_start_main@plt>:
 3f0:   ff a3 14 00 00 00       jmp    *0x14(%ebx)
 3f6:   68 10 00 00 00          push   $0x10
 3fb:   e9 c0 ff ff ff          jmp    3c0 <.plt>

Disassembly of section .plt.got:

00000400 <__cxa_finalize@plt>:
 400:   ff a3 1c 00 00 00       jmp    *0x1c(%ebx)
 406:   66 90                   xchg   %ax,%ax

00000408 <__gmon_start__@plt>:
 408:   ff a3 20 00 00 00       jmp    *0x20(%ebx)
 40e:   66 90                   xchg   %ax,%ax

Disassembly of section .text:
```

```
00000410 <_start>:
 410:   31 ed                   xor    %ebp,%ebp
 412:   5e                      pop    %esi
 413:   89 e1                   mov    %esp,%ecx
 415:   83 e4 f0                and    $0xfffffff0,%esp
 418:   50                      push   %eax
 419:   54                      push   %esp
 41a:   52                      push   %edx
 41b:   e8 22 00 00 00          call   442 <_start+0x32>
 420:   81 c3 b4 1b 00 00       add    $0x1bb4,%ebx
 426:   8d 83 ac e6 ff ff       lea    -0x1954(%ebx),%eax
 42c:   50                      push   %eax
 42d:   8d 83 4c e6 ff ff       lea    -0x19b4(%ebx),%eax
 433:   50                      push   %eax
 434:   51                      push   %ecx
 435:   56                      push   %esi
 436:   ff b3 24 00 00 00       push   0x24(%ebx)
 43c:   e8 af ff ff ff          call   3f0 <__libc_start_main@plt>
 441:   f4                      hlt
 442:   8b 1c 24                mov    (%esp),%ebx
 445:   c3                      ret
 446:   66 90                   xchg   %ax,%ax
 448:   66 90                   xchg   %ax,%ax
 44a:   66 90                   xchg   %ax,%ax
 44c:   66 90                   xchg   %ax,%ax
 44e:   66 90                   xchg   %ax,%ax

00000450 <__x86.get_pc_thunk.bx>:
 450:   8b 1c 24                mov    (%esp),%ebx
 453:   c3                      ret
 454:   66 90                   xchg   %ax,%ax
 456:   66 90                   xchg   %ax,%ax
 458:   66 90                   xchg   %ax,%ax
 45a:   66 90                   xchg   %ax,%ax
 45c:   66 90                   xchg   %ax,%ax
 45e:   66 90                   xchg   %ax,%ax

00000460 <deregister_tm_clones>:
 460:   e8 e4 00 00 00          call   549 <__x86.get_pc_thunk.dx>
 465:   81 c2 6f 1b 00 00       add    $0x1b6f,%edx
```

```
 46b:   8d 8a 34 00 00 00       lea    0x34(%edx),%ecx
 471:   8d 82 34 00 00 00       lea    0x34(%edx),%eax
 477:   39 c8                   cmp    %ecx,%eax
 479:   74 1d                   je     498
<deregister_tm_clones+0x38>
 47b:   8b 82 18 00 00 00       mov    0x18(%edx),%eax
 481:   85 c0                   test   %eax,%eax
 483:   74 13                   je     498
<deregister_tm_clones+0x38>
 485:   55                      push   %ebp
 486:   89 e5                   mov    %esp,%ebp
 488:   83 ec 14                sub    $0x14,%esp
 48b:   51                      push   %ecx
 48c:   ff d0                   call   *%eax
 48e:   83 c4 10                add    $0x10,%esp
 491:   c9                      leave
 492:   c3                      ret
 493:   90                      nop
 494:   8d 74 26 00             lea    0x0(%esi,%eiz,1),%esi
 498:   f3 c3                   repz ret
 49a:   8d b6 00 00 00 00       lea    0x0(%esi),%esi

000004a0 <register_tm_clones>:
 4a0:   e8 a4 00 00 00          call   549 <__x86.get_pc_thunk.dx>
 4a5:   81 c2 2f 1b 00 00       add    $0x1b2f,%edx
 4ab:   55                      push   %ebp
 4ac:   8d 8a 34 00 00 00       lea    0x34(%edx),%ecx
 4b2:   8d 82 34 00 00 00       lea    0x34(%edx),%eax
 4b8:   29 c8                   sub    %ecx,%eax
 4ba:   89 e5                   mov    %esp,%ebp
 4bc:   53                      push   %ebx
 4bd:   c1 f8 02                sar    $0x2,%eax
 4c0:   89 c3                   mov    %eax,%ebx
 4c2:   83 ec 04                sub    $0x4,%esp
 4c5:   c1 eb 1f                shr    $0x1f,%ebx
 4c8:   01 d8                   add    %ebx,%eax
 4ca:   d1 f8                   sar    $1,%eax
 4cc:   74 14                   je     4e2 <register_tm_clones+0x42>
 4ce:   8b 92 28 00 00 00       mov    0x28(%edx),%edx
 4d4:   85 d2                   test   %edx,%edx
```

```
 4d6:   74 0a                   je     4e2 <register_tm_clones+0x42>
 4d8:   83 ec 08                sub    $0x8,%esp
 4db:   50                      push   %eax
 4dc:   51                      push   %ecx
 4dd:   ff d2                   call   *%edx
 4df:   83 c4 10                add    $0x10,%esp
 4e2:   8b 5d fc                mov    -0x4(%ebp),%ebx
 4e5:   c9                      leave
 4e6:   c3                      ret
 4e7:   89 f6                   mov    %esi,%esi
 4e9:   8d bc 27 00 00 00 00    lea    0x0(%edi,%eiz,1),%edi

000004f0 <__do_global_dtors_aux>:
 4f0:   55                      push   %ebp
 4f1:   89 e5                   mov    %esp,%ebp
 4f3:   53                      push   %ebx
 4f4:   e8 57 ff ff ff          call   450 <__x86.get_pc_thunk.bx>
 4f9:   81 c3 db 1a 00 00       add    $0x1adb,%ebx
 4ff:   83 ec 04                sub    $0x4,%esp
 502:   80 bb 34 00 00 00 00    cmpb   $0x0,0x34(%ebx)
 509:   75 27                   jne    532
<__do_global_dtors_aux+0x42>
 50b:   8b 83 1c 00 00 00       mov    0x1c(%ebx),%eax
 511:   85 c0                   test   %eax,%eax
 513:   74 11                   je     526
<__do_global_dtors_aux+0x36>
 515:   83 ec 0c                sub    $0xc,%esp
 518:   ff b3 30 00 00 00       push   0x30(%ebx)
 51e:   e8 dd fe ff ff          call   400 <__cxa_finalize@plt>
 523:   83 c4 10                add    $0x10,%esp
 526:   e8 35 ff ff ff          call   460 <deregister_tm_clones>
 52b:   c6 83 34 00 00 00 01    movb   $0x1,0x34(%ebx)
 532:   8b 5d fc                mov    -0x4(%ebp),%ebx
 535:   c9                      leave
 536:   c3                      ret
 537:   89 f6                   mov    %esi,%esi
 539:   8d bc 27 00 00 00 00    lea    0x0(%edi,%eiz,1),%edi

00000540 <frame_dummy>:
 540:   55                      push   %ebp
```

```
 541:   89 e5                   mov    %esp,%ebp
 543:   5d                      pop    %ebp
 544:   e9 57 ff ff ff          jmp    4a0 <register_tm_clones>

00000549 <__x86.get_pc_thunk.dx>:
 549:   8b 14 24                mov    (%esp),%edx
 54c:   c3                      ret

0000054d <sample_function>:
 54d:   55                      push   %ebp
 54e:   89 e5                   mov    %esp,%ebp
 550:   53                      push   %ebx
 551:   83 ec 14                sub    $0x14,%esp
 554:   e8 f7 fe ff ff          call   450 <__x86.get_pc_thunk.bx>
 559:   81 c3 7b 1a 00 00       add    $0x1a7b,%ebx
 55f:   c7 45 f4 ff ff ff ff    movl   $0xffffffff,-0xc(%ebp)
 566:   8d 45 f4                lea    -0xc(%ebp),%eax
 569:   83 ec 08                sub    $0x8,%esp
 56c:   50                      push   %eax
 56d:   8d 83 cc e6 ff ff       lea    -0x1934(%ebx),%eax
 573:   50                      push   %eax
 574:   e8 57 fe ff ff          call   3d0 <printf@plt>
 579:   83 c4 10                add    $0x10,%esp
 57c:   8d 45 ea                lea    -0x16(%ebp),%eax
 57f:   83 ec 08                sub    $0x8,%esp
 582:   50                      push   %eax
 583:   8d 83 fc e6 ff ff       lea    -0x1904(%ebx),%eax
 589:   50                      push   %eax
 58a:   e8 41 fe ff ff          call   3d0 <printf@plt>
 58f:   83 c4 10                add    $0x10,%esp
 592:   8b 45 f4                mov    -0xc(%ebp),%eax
 595:   83 ec 08                sub    $0x8,%esp
 598:   50                      push   %eax
 599:   8d 83 30 e7 ff ff       lea    -0x18d0(%ebx),%eax
 59f:   50                      push   %eax
 5a0:   e8 2b fe ff ff          call   3d0 <printf@plt>
 5a5:   83 c4 10                add    $0x10,%esp
 5a8:   83 ec 0c                sub    $0xc,%esp
 5ab:   8d 45 ea                lea    -0x16(%ebp),%eax
 5ae:   50                      push   %eax
```

```
 5af:   e8 2c fe ff ff          call   3e0 <gets@plt>
 5b4:   83 c4 10                add    $0x10,%esp
 5b7:   8b 45 f4                mov    -0xc(%ebp),%eax
 5ba:   83 ec 08                sub    $0x8,%esp
 5bd:   50                      push   %eax
 5be:   8d 83 5c e7 ff ff       lea    -0x18a4(%ebx),%eax
 5c4:   50                      push   %eax
 5c5:   e8 06 fe ff ff          call   3d0 <printf@plt>
 5ca:   83 c4 10                add    $0x10,%esp
 5cd:   90                      nop
 5ce:   8b 5d fc                mov    -0x4(%ebp),%ebx
 5d1:   c9                      leave
 5d2:   c3                      ret

000005d3 <main>:
 5d3:   8d 4c 24 04             lea    0x4(%esp),%ecx
 5d7:   83 e4 f0                and    $0xfffffff0,%esp
 5da:   ff 71 fc                push   -0x4(%ecx)
 5dd:   55                      push   %ebp
 5de:   89 e5                   mov    %esp,%ebp
 5e0:   53                      push   %ebx
 5e1:   51                      push   %ecx
 5e2:   83 ec 10                sub    $0x10,%esp
 5e5:   e8 31 00 00 00          call   61b <__x86.get_pc_thunk.ax>
 5ea:   05 ea 19 00 00          add    $0x19ea,%eax
 5ef:   8d 55 f4                lea    -0xc(%ebp),%edx
 5f2:   83 ec 08                sub    $0x8,%esp
 5f5:   52                      push   %edx
 5f6:   8d 90 88 e7 ff ff       lea    -0x1878(%eax),%edx
 5fc:   52                      push   %edx
 5fd:   89 c3                   mov    %eax,%ebx
 5ff:   e8 cc fd ff ff          call   3d0 <printf@plt>
 604:   83 c4 10                add    $0x10,%esp
 607:   e8 41 ff ff ff          call   54d <sample_function>
 60c:   b8 00 00 00 00          mov    $0x0,%eax
 611:   8d 65 f8                lea    -0x8(%ebp),%esp
 614:   59                      pop    %ecx
 615:   5b                      pop    %ebx
 616:   5d                      pop    %ebp
 617:   8d 61 fc                lea    -0x4(%ecx),%esp
```

```
 61a:   c3                      ret

0000061b <__x86.get_pc_thunk.ax>:
 61b:   8b 04 24                mov    (%esp),%eax
 61e:   c3                      ret
 61f:   90                      nop

00000620 <__libc_csu_init>:
 620:   55                      push   %ebp
 621:   57                      push   %edi
 622:   56                      push   %esi
 623:   53                      push   %ebx
 624:   e8 27 fe ff ff          call   450 <__x86.get_pc_thunk.bx>
 629:   81 c3 ab 19 00 00       add    $0x19ab,%ebx
 62f:   83 ec 0c                sub    $0xc,%esp
 632:   8b 6c 24 28             mov    0x28(%esp),%ebp
 636:   8d b3 04 ff ff ff       lea    -0xfc(%ebx),%esi
 63c:   e8 57 fd ff ff          call   398 <_init>
 641:   8d 83 00 ff ff ff       lea    -0x100(%ebx),%eax
 647:   29 c6                   sub    %eax,%esi
 649:   c1 fe 02                sar    $0x2,%esi
 64c:   85 f6                   test   %esi,%esi
 64e:   74 25                   je     675 <__libc_csu_init+0x55>
 650:   31 ff                   xor    %edi,%edi
 652:   8d b6 00 00 00 00       lea    0x0(%esi),%esi
 658:   83 ec 04                sub    $0x4,%esp
 65b:   55                      push   %ebp
 65c:   ff 74 24 2c             push   0x2c(%esp)
 660:   ff 74 24 2c             push   0x2c(%esp)
 664:   ff 94 bb 00 ff ff ff    call   *-0x100(%ebx,%edi,4)
 66b:   83 c7 01                add    $0x1,%edi
 66e:   83 c4 10                add    $0x10,%esp
 671:   39 fe                   cmp    %edi,%esi
 673:   75 e3                   jne    658 <__libc_csu_init+0x38>
 675:   83 c4 0c                add    $0xc,%esp
 678:   5b                      pop    %ebx
 679:   5e                      pop    %esi
 67a:   5f                      pop    %edi
 67b:   5d                      pop    %ebp
 67c:   c3                      ret
```

```
 67d:   8d 76 00                lea    0x0(%esi),%esi

00000680 <__libc_csu_fini>:
 680:   f3 c3                   repz ret

Disassembly of section .fini:

00000684 <_fini>:
 684:   53                      push   %ebx
 685:   83 ec 08                sub    $0x8,%esp
 688:   e8 c3 fd ff ff          call   450 <__x86.get_pc_thunk.bx>
 68d:   81 c3 47 19 00 00       add    $0x1947,%ebx
 693:   83 c4 08                add    $0x8,%esp
 696:   5b                      pop    %ebx
 697:   c3                      ret
```

- **sample64**

```
sample64:     file format elf64-x86-64


Disassembly of section .init:

0000000000000528 <_init>:
 528:   48 83 ec 08             sub    $0x8,%rsp
 52c:   48 8b 05 b5 0a 20 00    mov    0x200ab5(%rip),%rax        #
200fe8 <__gmon_start__>
 533:   48 85 c0                test   %rax,%rax
 536:   74 02                   je     53a <_init+0x12>
 538:   ff d0                   call   *%rax
 53a:   48 83 c4 08             add    $0x8,%rsp
 53e:   c3                      ret

Disassembly of section .plt:

0000000000000540 <.plt>:
 540:   ff 35 72 0a 20 00       push   0x200a72(%rip)        # 200fb8
<_GLOBAL_OFFSET_TABLE_+0x8>
 546:   ff 25 74 0a 20 00       jmp    *0x200a74(%rip)        #
200fc0 <_GLOBAL_OFFSET_TABLE_+0x10>
```

```
 54c:   0f 1f 40 00             nopl   0x0(%rax)

0000000000000550 <printf@plt>:
 550:   ff 25 72 0a 20 00       jmp    *0x200a72(%rip)        #
200fc8 <printf@GLIBC_2.2.5>
 556:   68 00 00 00 00          push   $0x0
 55b:   e9 e0 ff ff ff          jmp    540 <.plt>

0000000000000560 <gets@plt>:
 560:   ff 25 6a 0a 20 00       jmp    *0x200a6a(%rip)        #
200fd0 <gets@GLIBC_2.2.5>
 566:   68 01 00 00 00          push   $0x1
 56b:   e9 d0 ff ff ff          jmp    540 <.plt>

Disassembly of section .plt.got:

0000000000000570 <__cxa_finalize@plt>:
 570:   ff 25 82 0a 20 00       jmp    *0x200a82(%rip)        #
200ff8 <__cxa_finalize@GLIBC_2.2.5>
 576:   66 90                   xchg   %ax,%ax

Disassembly of section .text:

0000000000000580 <_start>:
 580:   31 ed                   xor    %ebp,%ebp
 582:   49 89 d1                mov    %rdx,%r9
 585:   5e                      pop    %rsi
 586:   48 89 e2                mov    %rsp,%rdx
 589:   48 83 e4 f0             and    $0xfffffffffffffff0,%rsp
 58d:   50                      push   %rax
 58e:   54                      push   %rsp
 58f:   4c 8d 05 1a 02 00 00    lea    0x21a(%rip),%r8        # 7b0
<__libc_csu_fini>
 596:   48 8d 0d a3 01 00 00    lea    0x1a3(%rip),%rcx        # 740
<__libc_csu_init>
 59d:   48 8d 3d 6b 01 00 00    lea    0x16b(%rip),%rdi        # 70f
<main>
 5a4:   ff 15 36 0a 20 00       call   *0x200a36(%rip)        #
200fe0 <__libc_start_main@GLIBC_2.2.5>
 5aa:   f4                      hlt
```

```
 5ab:   0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

00000000000005b0 <deregister_tm_clones>:
 5b0:   48 8d 3d 59 0a 20 00    lea    0x200a59(%rip),%rdi        #
201010 <__TMC_END__>
 5b7:   55                      push   %rbp
 5b8:   48 8d 05 51 0a 20 00    lea    0x200a51(%rip),%rax        #
201010 <__TMC_END__>
 5bf:   48 39 f8                cmp    %rdi,%rax
 5c2:   48 89 e5                mov    %rsp,%rbp
 5c5:   74 19                   je     5e0
<deregister_tm_clones+0x30>
 5c7:   48 8b 05 0a 0a 20 00    mov    0x200a0a(%rip),%rax        #
200fd8 <_ITM_deregisterTMCloneTable>
 5ce:   48 85 c0                test   %rax,%rax
 5d1:   74 0d                   je     5e0
<deregister_tm_clones+0x30>
 5d3:   5d                      pop    %rbp
 5d4:   ff e0                   jmp    *%rax
 5d6:   66 2e 0f 1f 84 00 00    cs nopw 0x0(%rax,%rax,1)
 5dd:   00 00 00
 5e0:   5d                      pop    %rbp
 5e1:   c3                      ret
 5e2:   0f 1f 40 00             nopl   0x0(%rax)
 5e6:   66 2e 0f 1f 84 00 00    cs nopw 0x0(%rax,%rax,1)
 5ed:   00 00 00

00000000000005f0 <register_tm_clones>:
 5f0:   48 8d 3d 19 0a 20 00    lea    0x200a19(%rip),%rdi        #
201010 <__TMC_END__>
 5f7:   48 8d 35 12 0a 20 00    lea    0x200a12(%rip),%rsi        #
201010 <__TMC_END__>
 5fe:   55                      push   %rbp
 5ff:   48 29 fe                sub    %rdi,%rsi
 602:   48 89 e5                mov    %rsp,%rbp
 605:   48 c1 fe 03             sar    $0x3,%rsi
 609:   48 89 f0                mov    %rsi,%rax
 60c:   48 c1 e8 3f             shr    $0x3f,%rax
 610:   48 01 c6                add    %rax,%rsi
 613:   48 d1 fe                sar    $1,%rsi
```

```
 616:   74 18                   je     630 <register_tm_clones+0x40>
 618:   48 8b 05 d1 09 20 00    mov    0x2009d1(%rip),%rax        #
200ff0 <_ITM_registerTMCloneTable>
 61f:   48 85 c0                test   %rax,%rax
 622:   74 0c                   je     630 <register_tm_clones+0x40>
 624:   5d                      pop    %rbp
 625:   ff e0                   jmp    *%rax
 627:   66 0f 1f 84 00 00 00    nopw   0x0(%rax,%rax,1)
 62e:   00 00
 630:   5d                      pop    %rbp
 631:   c3                      ret
 632:   0f 1f 40 00             nopl   0x0(%rax)
 636:   66 2e 0f 1f 84 00 00    cs nopw 0x0(%rax,%rax,1)
 63d:   00 00 00

0000000000000640 <__do_global_dtors_aux>:
 640:   80 3d c9 09 20 00 00    cmpb   $0x0,0x2009c9(%rip)        #
201010 <__TMC_END__>
 647:   75 2f                   jne    678
<__do_global_dtors_aux+0x38>
 649:   48 83 3d a7 09 20 00    cmpq   $0x0,0x2009a7(%rip)        #
200ff8 <__cxa_finalize@GLIBC_2.2.5>
 650:   00
 651:   55                      push   %rbp
 652:   48 89 e5                mov    %rsp,%rbp
 655:   74 0c                   je     663
<__do_global_dtors_aux+0x23>
 657:   48 8b 3d aa 09 20 00    mov    0x2009aa(%rip),%rdi        #
201008 <__dso_handle>
 65e:   e8 0d ff ff ff          call   570 <__cxa_finalize@plt>
 663:   e8 48 ff ff ff          call   5b0 <deregister_tm_clones>
 668:   c6 05 a1 09 20 00 01    movb   $0x1,0x2009a1(%rip)        #
201010 <__TMC_END__>
 66f:   5d                      pop    %rbp
 670:   c3                      ret
 671:   0f 1f 80 00 00 00 00    nopl   0x0(%rax)
 678:   f3 c3                   repz ret
 67a:   66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)

0000000000000680 <frame_dummy>:
```

```
 680:   55                      push   %rbp
 681:   48 89 e5                mov    %rsp,%rbp
 684:   5d                      pop    %rbp
 685:   e9 66 ff ff ff          jmp    5f0 <register_tm_clones>

000000000000068a <sample_function>:
 68a:   55                      push   %rbp
 68b:   48 89 e5                mov    %rsp,%rbp
 68e:   48 83 ec 20             sub    $0x20,%rsp
 692:   b8 ff ff ff ff          mov    $0xffffffff,%eax
 697:   48 89 45 f8             mov    %rax,-0x8(%rbp)
 69b:   48 8d 45 f8             lea    -0x8(%rbp),%rax
 69f:   48 89 c6                mov    %rax,%rsi
 6a2:   48 8d 3d 1f 01 00 00    lea    0x11f(%rip),%rdi        # 7c8
<_IO_stdin_used+0x8>
 6a9:   b8 00 00 00 00          mov    $0x0,%eax
 6ae:   e8 9d fe ff ff          call   550 <printf@plt>
 6b3:   48 8d 45 ee             lea    -0x12(%rbp),%rax
 6b7:   48 89 c6                mov    %rax,%rsi
 6ba:   48 8d 3d 37 01 00 00    lea    0x137(%rip),%rdi        # 7f8
<_IO_stdin_used+0x38>
 6c1:   b8 00 00 00 00          mov    $0x0,%eax
 6c6:   e8 85 fe ff ff          call   550 <printf@plt>
 6cb:   48 8b 45 f8             mov    -0x8(%rbp),%rax
 6cf:   48 89 c6                mov    %rax,%rsi
 6d2:   48 8d 3d 4f 01 00 00    lea    0x14f(%rip),%rdi        # 828
<_IO_stdin_used+0x68>
 6d9:   b8 00 00 00 00          mov    $0x0,%eax
 6de:   e8 6d fe ff ff          call   550 <printf@plt>
 6e3:   48 8d 45 ee             lea    -0x12(%rbp),%rax
 6e7:   48 89 c7                mov    %rax,%rdi
 6ea:   b8 00 00 00 00          mov    $0x0,%eax
 6ef:   e8 6c fe ff ff          call   560 <gets@plt>
 6f4:   48 8b 45 f8             mov    -0x8(%rbp),%rax
 6f8:   48 89 c6                mov    %rax,%rsi
 6fb:   48 8d 3d 56 01 00 00    lea    0x156(%rip),%rdi        # 858
<_IO_stdin_used+0x98>
 702:   b8 00 00 00 00          mov    $0x0,%eax
 707:   e8 44 fe ff ff          call   550 <printf@plt>
 70c:   90                      nop
```

```
 70d:   c9                      leave
 70e:   c3                      ret

000000000000070f <main>:
 70f:   55                      push   %rbp
 710:   48 89 e5                mov    %rsp,%rbp
 713:   48 83 ec 10             sub    $0x10,%rsp
 717:   48 8d 45 fc             lea    -0x4(%rbp),%rax
 71b:   48 89 c6                mov    %rax,%rsi
 71e:   48 8d 3d 63 01 00 00    lea    0x163(%rip),%rdi        # 888
<_IO_stdin_used+0xc8>
 725:   b8 00 00 00 00          mov    $0x0,%eax
 72a:   e8 21 fe ff ff          call   550 <printf@plt>
 72f:   b8 00 00 00 00          mov    $0x0,%eax
 734:   e8 51 ff ff ff          call   68a <sample_function>
 739:   b8 00 00 00 00          mov    $0x0,%eax
 73e:   c9                      leave
 73f:   c3                      ret

0000000000000740 <__libc_csu_init>:
 740:   41 57                   push   %r15
 742:   41 56                   push   %r14
 744:   49 89 d7                mov    %rdx,%r15
 747:   41 55                   push   %r13
 749:   41 54                   push   %r12
 74b:   4c 8d 25 5e 06 20 00    lea    0x20065e(%rip),%r12        #
200db0 <__frame_dummy_init_array_entry>
 752:   55                      push   %rbp
 753:   48 8d 2d 5e 06 20 00    lea    0x20065e(%rip),%rbp        #
200db8 <__do_global_dtors_aux_fini_array_entry>
 75a:   53                      push   %rbx
 75b:   41 89 fd                mov    %edi,%r13d
 75e:   49 89 f6                mov    %rsi,%r14
 761:   4c 29 e5                sub    %r12,%rbp
 764:   48 83 ec 08             sub    $0x8,%rsp
 768:   48 c1 fd 03             sar    $0x3,%rbp
 76c:   e8 b7 fd ff ff          call   528 <_init>
 771:   48 85 ed                test   %rbp,%rbp
 774:   74 20                   je     796 <__libc_csu_init+0x56>
 776:   31 db                   xor    %ebx,%ebx
```

```
 778:   0f 1f 84 00 00 00 00    nopl   0x0(%rax,%rax,1)
 77f:   00
 780:   4c 89 fa                mov    %r15,%rdx
 783:   4c 89 f6                mov    %r14,%rsi
 786:   44 89 ef                mov    %r13d,%edi
 789:   41 ff 14 dc             call   *(%r12,%rbx,8)
 78d:   48 83 c3 01             add    $0x1,%rbx
 791:   48 39 dd                cmp    %rbx,%rbp
 794:   75 ea                   jne    780 <__libc_csu_init+0x40>
 796:   48 83 c4 08             add    $0x8,%rsp
 79a:   5b                      pop    %rbx
 79b:   5d                      pop    %rbp
 79c:   41 5c                   pop    %r12
 79e:   41 5d                   pop    %r13
 7a0:   41 5e                   pop    %r14
 7a2:   41 5f                   pop    %r15
 7a4:   c3                      ret
 7a5:   90                      nop
 7a6:   66 2e 0f 1f 84 00 00    cs nopw 0x0(%rax,%rax,1)
 7ad:   00 00 00

00000000000007b0 <__libc_csu_fini>:
 7b0:   f3 c3                   repz ret

Disassembly of section .fini:

00000000000007b4 <_fini>:
 7b4:   48 83 ec 08             sub    $0x8,%rsp
 7b8:   48 83 c4 08             add    $0x8,%rsp
 7bc:   c3                      ret
```

- **sample64-2**

```
sample64-2:     file format elf64-x86-64


Disassembly of section .init:

0000000000000580 <_init>:
 580:   48 83 ec 08             sub    $0x8,%rsp
```

```
 584:   48 8b 05 5d 0a 20 00    mov    0x200a5d(%rip),%rax        #
200fe8 <__gmon_start__>
 58b:   48 85 c0                test   %rax,%rax
 58e:   74 02                   je     592 <_init+0x12>
 590:   ff d0                   call   *%rax
 592:   48 83 c4 08             add    $0x8,%rsp
 596:   c3                      ret

Disassembly of section .plt:

00000000000005a0 <.plt>:
 5a0:   ff 35 0a 0a 20 00       push   0x200a0a(%rip)        # 200fb0
<_GLOBAL_OFFSET_TABLE_+0x8>
 5a6:   ff 25 0c 0a 20 00       jmp    *0x200a0c(%rip)        #
200fb8 <_GLOBAL_OFFSET_TABLE_+0x10>
 5ac:   0f 1f 40 00             nopl   0x0(%rax)

00000000000005b0 <__stack_chk_fail@plt>:
 5b0:   ff 25 0a 0a 20 00       jmp    *0x200a0a(%rip)        #
200fc0 <__stack_chk_fail@GLIBC_2.4>
 5b6:   68 00 00 00 00          push   $0x0
 5bb:   e9 e0 ff ff ff          jmp    5a0 <.plt>

00000000000005c0 <printf@plt>:
 5c0:   ff 25 02 0a 20 00       jmp    *0x200a02(%rip)        #
200fc8 <printf@GLIBC_2.2.5>
 5c6:   68 01 00 00 00          push   $0x1
 5cb:   e9 d0 ff ff ff          jmp    5a0 <.plt>

00000000000005d0 <gets@plt>:
 5d0:   ff 25 fa 09 20 00       jmp    *0x2009fa(%rip)        #
200fd0 <gets@GLIBC_2.2.5>
 5d6:   68 02 00 00 00          push   $0x2
 5db:   e9 c0 ff ff ff          jmp    5a0 <.plt>

Disassembly of section .plt.got:

00000000000005e0 <__cxa_finalize@plt>:
 5e0:   ff 25 12 0a 20 00       jmp    *0x200a12(%rip)        #
200ff8 <__cxa_finalize@GLIBC_2.2.5>
```

```
 5e6:   66 90                   xchg   %ax,%ax

Disassembly of section .text:

00000000000005f0 <_start>:
 5f0:   31 ed                   xor    %ebp,%ebp
 5f2:   49 89 d1                mov    %rdx,%r9
 5f5:   5e                      pop    %rsi
 5f6:   48 89 e2                mov    %rsp,%rdx
 5f9:   48 83 e4 f0             and    $0xfffffffffffffff0,%rsp
 5fd:   50                      push   %rax
 5fe:   54                      push   %rsp
 5ff:   4c 8d 05 6a 02 00 00    lea    0x26a(%rip),%r8        # 870
<__libc_csu_fini>
 606:   48 8d 0d f3 01 00 00    lea    0x1f3(%rip),%rcx        # 800
<__libc_csu_init>
 60d:   48 8d 3d 8e 01 00 00    lea    0x18e(%rip),%rdi        # 7a2
<main>
 614:   ff 15 c6 09 20 00       call   *0x2009c6(%rip)        #
200fe0 <__libc_start_main@GLIBC_2.2.5>
 61a:   f4                      hlt
 61b:   0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

0000000000000620 <deregister_tm_clones>:
 620:   48 8d 3d e9 09 20 00    lea    0x2009e9(%rip),%rdi        #
201010 <__TMC_END__>
 627:   55                      push   %rbp
 628:   48 8d 05 e1 09 20 00    lea    0x2009e1(%rip),%rax        #
201010 <__TMC_END__>
 62f:   48 39 f8                cmp    %rdi,%rax
 632:   48 89 e5                mov    %rsp,%rbp
 635:   74 19                   je     650
<deregister_tm_clones+0x30>
 637:   48 8b 05 9a 09 20 00    mov    0x20099a(%rip),%rax        #
200fd8 <_ITM_deregisterTMCloneTable>
 63e:   48 85 c0                test   %rax,%rax
 641:   74 0d                   je     650
<deregister_tm_clones+0x30>
 643:   5d                      pop    %rbp
 644:   ff e0                   jmp    *%rax
```

```
 646:   66 2e 0f 1f 84 00 00    cs nopw 0x0(%rax,%rax,1)
 64d:   00 00 00
 650:   5d                      pop    %rbp
 651:   c3                      ret
 652:   0f 1f 40 00             nopl   0x0(%rax)
 656:   66 2e 0f 1f 84 00 00    cs nopw 0x0(%rax,%rax,1)
 65d:   00 00 00

0000000000000660 <register_tm_clones>:
 660:   48 8d 3d a9 09 20 00    lea    0x2009a9(%rip),%rdi        #
201010 <__TMC_END__>
 667:   48 8d 35 a2 09 20 00    lea    0x2009a2(%rip),%rsi        #
201010 <__TMC_END__>
 66e:   55                      push   %rbp
 66f:   48 29 fe                sub    %rdi,%rsi
 672:   48 89 e5                mov    %rsp,%rbp
 675:   48 c1 fe 03             sar    $0x3,%rsi
 679:   48 89 f0                mov    %rsi,%rax
 67c:   48 c1 e8 3f             shr    $0x3f,%rax
 680:   48 01 c6                add    %rax,%rsi
 683:   48 d1 fe                sar    $1,%rsi
 686:   74 18                   je     6a0 <register_tm_clones+0x40>
 688:   48 8b 05 61 09 20 00    mov    0x200961(%rip),%rax        #
200ff0 <_ITM_registerTMCloneTable>
 68f:   48 85 c0                test   %rax,%rax
 692:   74 0c                   je     6a0 <register_tm_clones+0x40>
 694:   5d                      pop    %rbp
 695:   ff e0                   jmp    *%rax
 697:   66 0f 1f 84 00 00 00    nopw   0x0(%rax,%rax,1)
 69e:   00 00
 6a0:   5d                      pop    %rbp
 6a1:   c3                      ret
 6a2:   0f 1f 40 00             nopl   0x0(%rax)
 6a6:   66 2e 0f 1f 84 00 00    cs nopw 0x0(%rax,%rax,1)
 6ad:   00 00 00

00000000000006b0 <__do_global_dtors_aux>:
 6b0:   80 3d 59 09 20 00 00    cmpb   $0x0,0x200959(%rip)        #
201010 <__TMC_END__>
 6b7:   75 2f                   jne    6e8
```

```
<__do_global_dtors_aux+0x38>
 6b9:   48 83 3d 37 09 20 00    cmpq   $0x0,0x200937(%rip)        #
200ff8 <__cxa_finalize@GLIBC_2.2.5>
 6c0:   00
 6c1:   55                      push   %rbp
 6c2:   48 89 e5                mov    %rsp,%rbp
 6c5:   74 0c                   je     6d3
<__do_global_dtors_aux+0x23>
 6c7:   48 8b 3d 3a 09 20 00    mov    0x20093a(%rip),%rdi        #
201008 <__dso_handle>
 6ce:   e8 0d ff ff ff          call   5e0 <__cxa_finalize@plt>
 6d3:   e8 48 ff ff ff          call   620 <deregister_tm_clones>
 6d8:   c6 05 31 09 20 00 01    movb   $0x1,0x200931(%rip)        #
201010 <__TMC_END__>
 6df:   5d                      pop    %rbp
 6e0:   c3                      ret
 6e1:   0f 1f 80 00 00 00 00    nopl   0x0(%rax)
 6e8:   f3 c3                   repz ret
 6ea:   66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)

00000000000006f0 <frame_dummy>:
 6f0:   55                      push   %rbp
 6f1:   48 89 e5                mov    %rsp,%rbp
 6f4:   5d                      pop    %rbp
 6f5:   e9 66 ff ff ff          jmp    660 <register_tm_clones>

00000000000006fa <sample_function>:
 6fa:   55                      push   %rbp
 6fb:   48 89 e5                mov    %rsp,%rbp
 6fe:   48 83 ec 20             sub    $0x20,%rsp
 702:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
 709:   00 00
 70b:   48 89 45 f8             mov    %rax,-0x8(%rbp)
 70f:   31 c0                   xor    %eax,%eax
 711:   b8 ff ff ff ff          mov    $0xffffffff,%eax
 716:   48 89 45 e0             mov    %rax,-0x20(%rbp)
 71a:   48 8d 45 e0             lea    -0x20(%rbp),%rax
 71e:   48 89 c6                mov    %rax,%rsi
 721:   48 8d 3d 60 01 00 00    lea    0x160(%rip),%rdi        # 888
<_IO_stdin_used+0x8>
```

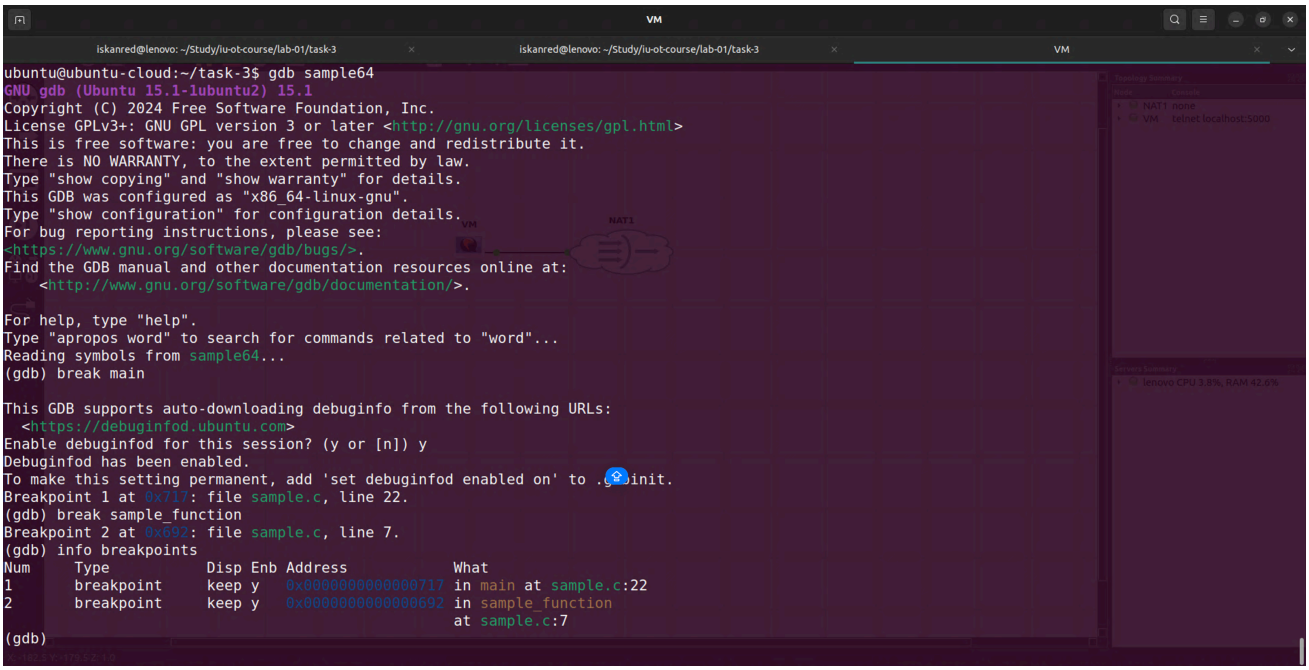
```
 728:   b8 00 00 00 00          mov    $0x0,%eax
 72d:   e8 8e fe ff ff          call   5c0 <printf@plt>
 732:   48 8d 45 ee             lea    -0x12(%rbp),%rax
 736:   48 89 c6                mov    %rax,%rsi
 739:   48 8d 3d 78 01 00 00    lea    0x178(%rip),%rdi        # 8b8
<_IO_stdin_used+0x38>
 740:   b8 00 00 00 00          mov    $0x0,%eax
 745:   e8 76 fe ff ff          call   5c0 <printf@plt>
 74a:   48 8b 45 e0             mov    -0x20(%rbp),%rax
 74e:   48 89 c6                mov    %rax,%rsi
 751:   48 8d 3d 90 01 00 00    lea    0x190(%rip),%rdi        # 8e8
<_IO_stdin_used+0x68>
 758:   b8 00 00 00 00          mov    $0x0,%eax
 75d:   e8 5e fe ff ff          call   5c0 <printf@plt>
 762:   48 8d 45 ee             lea    -0x12(%rbp),%rax
 766:   48 89 c7                mov    %rax,%rdi
 769:   b8 00 00 00 00          mov    $0x0,%eax
 76e:   e8 5d fe ff ff          call   5d0 <gets@plt>
 773:   48 8b 45 e0             mov    -0x20(%rbp),%rax
 777:   48 89 c6                mov    %rax,%rsi
 77a:   48 8d 3d 97 01 00 00    lea    0x197(%rip),%rdi        # 918
<_IO_stdin_used+0x98>
 781:   b8 00 00 00 00          mov    $0x0,%eax
 786:   e8 35 fe ff ff          call   5c0 <printf@plt>
 78b:   90                      nop
 78c:   48 8b 45 f8             mov    -0x8(%rbp),%rax
 790:   64 48 33 04 25 28 00    xor    %fs:0x28,%rax
 797:   00 00
 799:   74 05                   je     7a0 <sample_function+0xa6>
 79b:   e8 10 fe ff ff          call   5b0 <__stack_chk_fail@plt>
 7a0:   c9                      leave
 7a1:   c3                      ret

00000000000007a2 <main>:
 7a2:   55                      push   %rbp
 7a3:   48 89 e5                mov    %rsp,%rbp
 7a6:   48 83 ec 10             sub    $0x10,%rsp
 7aa:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
 7b1:   00 00
 7b3:   48 89 45 f8             mov    %rax,-0x8(%rbp)
```

```
 7b7:   31 c0                   xor    %eax,%eax
 7b9:   48 8d 45 f4             lea    -0xc(%rbp),%rax
 7bd:   48 89 c6                mov    %rax,%rsi
 7c0:   48 8d 3d 81 01 00 00    lea    0x181(%rip),%rdi        # 948
<_IO_stdin_used+0xc8>
 7c7:   b8 00 00 00 00          mov    $0x0,%eax
 7cc:   e8 ef fd ff ff          call   5c0 <printf@plt>
 7d1:   b8 00 00 00 00          mov    $0x0,%eax
 7d6:   e8 1f ff ff ff          call   6fa <sample_function>
 7db:   b8 00 00 00 00          mov    $0x0,%eax
 7e0:   48 8b 55 f8             mov    -0x8(%rbp),%rdx
 7e4:   64 48 33 14 25 28 00    xor    %fs:0x28,%rdx
 7eb:   00 00
 7ed:   74 05                   je     7f4 <main+0x52>
 7ef:   e8 bc fd ff ff          call   5b0 <__stack_chk_fail@plt>
 7f4:   c9                      leave
 7f5:   c3                      ret
 7f6:   66 2e 0f 1f 84 00 00    cs nopw 0x0(%rax,%rax,1)
 7fd:   00 00 00

0000000000000800 <__libc_csu_init>:
 800:   41 57                   push   %r15
 802:   41 56                   push   %r14
 804:   49 89 d7                mov    %rdx,%r15
 807:   41 55                   push   %r13
 809:   41 54                   push   %r12
 80b:   4c 8d 25 96 05 20 00    lea    0x200596(%rip),%r12        #
200da8 <__frame_dummy_init_array_entry>
 812:   55                      push   %rbp
 813:   48 8d 2d 96 05 20 00    lea    0x200596(%rip),%rbp        #
200db0 <__do_global_dtors_aux_fini_array_entry>
 81a:   53                      push   %rbx
 81b:   41 89 fd                mov    %edi,%r13d
 81e:   49 89 f6                mov    %rsi,%r14
 821:   4c 29 e5                sub    %r12,%rbp
 824:   48 83 ec 08             sub    $0x8,%rsp
 828:   48 c1 fd 03             sar    $0x3,%rbp
 82c:   e8 4f fd ff ff          call   580 <_init>
 831:   48 85 ed                test   %rbp,%rbp
 834:   74 20                   je     856 <__libc_csu_init+0x56>
```

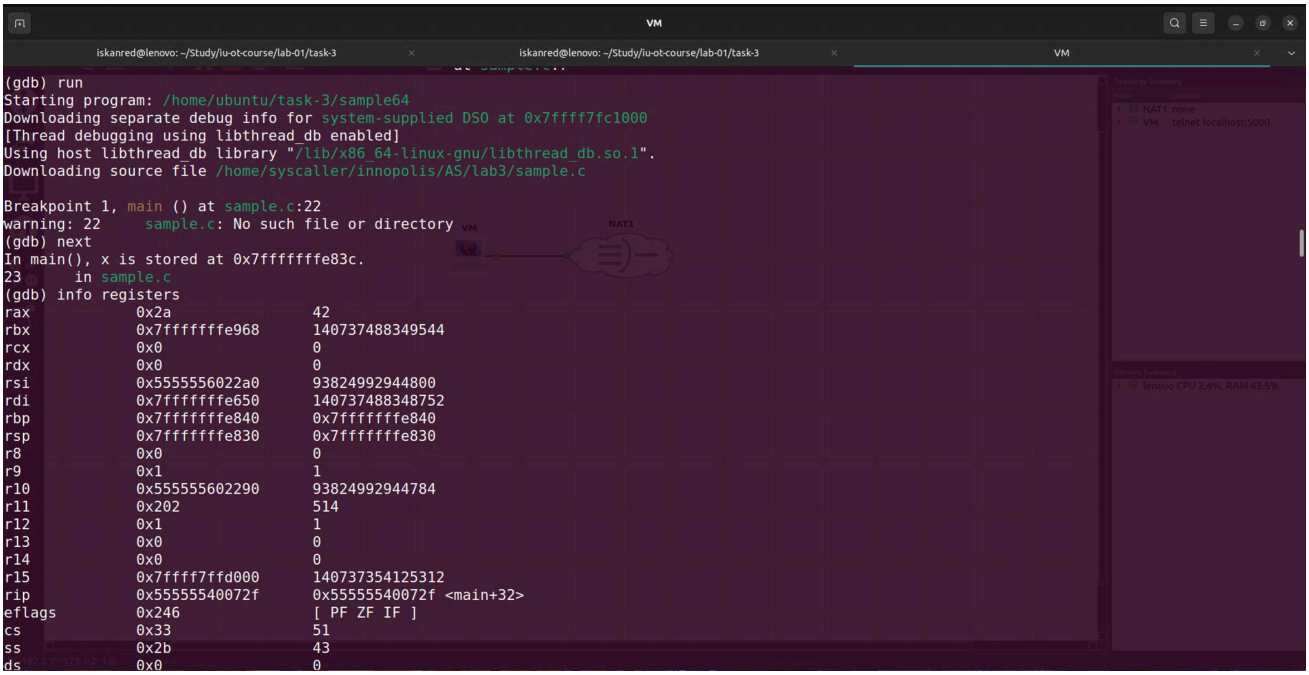
```
 836:   31 db                   xor    %ebx,%ebx
 838:   0f 1f 84 00 00 00 00    nopl   0x0(%rax,%rax,1)
 83f:   00
 840:   4c 89 fa                mov    %r15,%rdx
 843:   4c 89 f6                mov    %r14,%rsi
 846:   44 89 ef                mov    %r13d,%edi
 849:   41 ff 14 dc             call   *(%r12,%rbx,8)
 84d:   48 83 c3 01             add    $0x1,%rbx
 851:   48 39 dd                cmp    %rbx,%rbp
 854:   75 ea                   jne    840 <__libc_csu_init+0x40>
 856:   48 83 c4 08             add    $0x8,%rsp
 85a:   5b                      pop    %rbx
 85b:   5d                      pop    %rbp
 85c:   41 5c                   pop    %r12
 85e:   41 5d                   pop    %r13
 860:   41 5e                   pop    %r14
 862:   41 5f                   pop    %r15
 864:   c3                      ret
 865:   90                      nop
 866:   66 2e 0f 1f 84 00 00    cs nopw 0x0(%rax,%rax,1)
 86d:   00 00 00

0000000000000870 <__libc_csu_fini>:
 870:   f3 c3                   repz ret

Disassembly of section .fini:

0000000000000874 <_fini>:
 874:   48 83 ec 08             sub    $0x8,%rsp
 878:   48 83 c4 08             add    $0x8,%rsp
 87c:   c3                      ret
```

Debugging

- From disassembling I figured out there are at least two user-defined function
- Therefore, I inserted two breakpoints: to enter the function `main` and to enter the function `sample_function` in `sample64` executable



- And started discovering the program



- I used different commands such as
  - `info` : `registers` , `locals` , `variables` , `functions` , `source` , `stack` , `line` , `scope`
  - `bt` , `next` , `step` , `print`

Evaluation

- After hours spent on figuring out what's going in the given binaries I can conclude that
  - `sample32` : 32-bit program that does the following:
    1. In `main` we create 4-byte (`int32`) `x` integer variable.

2. We print its memory address using pointer to it or referencing `&x`: `In main(), x is stored at 0x7fffffffe83c`
3. Than we enter `sample_function` with no arguments
4. Inside `sample_function` we create 8-byte integer `i` (`int64`) and initialise it with `0xffffffff` value
5. Also, we create a `char` array `buffer` with length of 10 bytes (9 meaningful bytes and the last `\0`)
6. Then we print `i`'s memory address: `In sample_function(), i is stored at 0x7fffffffe818`
7. Then we print `buffer`'s memory address: `In sample_function(), buffer is stored at 0x7fffffffe80e`
8. Then we print the value of `i` in a hexadecimal format: `Value of i before calling gets(): 0xffffffff`
9. Then we call `gets(&buffer)` to save input line to the `buffer` which can trigger possible buffer overflow and replacement of `i`'s value
10. Then we run print `i` 's value after `gets`: `Value of i after calling gets(): 0xffffffff`
11. The functions ends and returns to the `main`
12. The program finishes

- `samp164`: 64-bit program does the same as 32-bit but on 64-bit architecture
- `sample64-2` 64-bit program that does the same as `sample64` but seems that is was compiled with stack overflow protection using the `-fstack-protector` flag which detects buffer overflow and finishes programs preventing an attack using this vulnerability. Also, the executable does not contain any debug info which makes it harder to figure out its behaviour.

```
Value of i before calling gets(): 0xffffffff
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Value of i after calling gets(): 0xffffffff
*** stack smashing detected ***: terminated
Aborted (core dumped)
```

c.

> ✎ **Task description**
>
> c. Do the function prologue and epilogue differ in 32bit and 64bit?

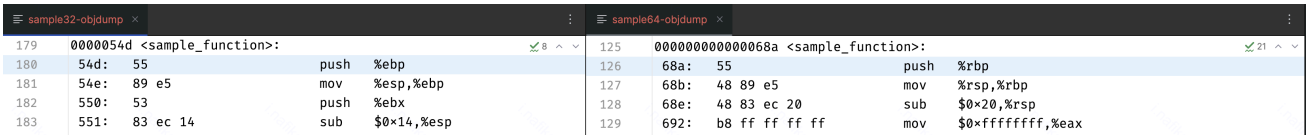
Description

- **Function prologue** is a code generated at the beginning of a function that performs the following tasks:

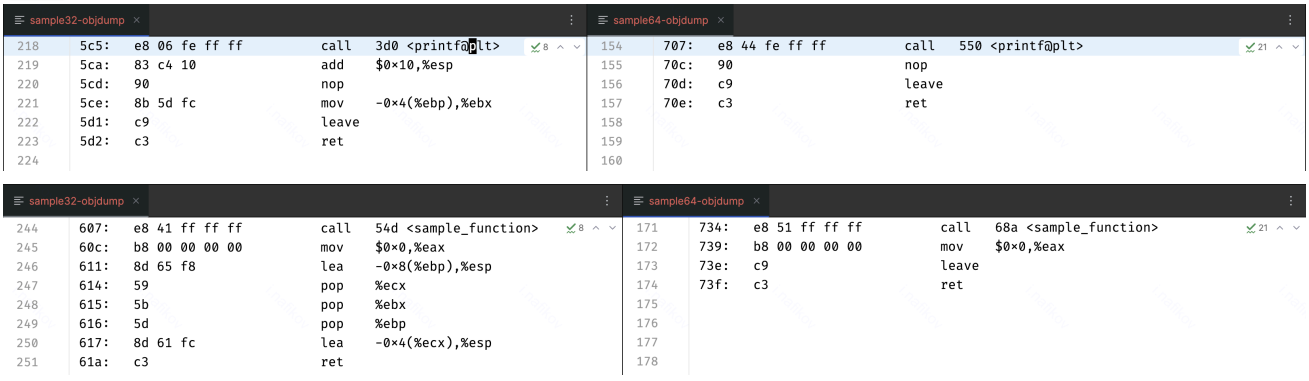
    - **Creating stack space for local variables**: This usually involves allocating stack space for variables used in a function.
    - **Context preservation**: If a function uses registers that can be changed (for example, general-purpose registers), then prolog can save the current state of these registers on the stack in order to restore it later. This is important so that the function can complete its work correctly and not damage the data that could be used by the functions that called it.
    - **Base Pointer installation**: The usual practice is to save the current base pointer (BP, Base Pointer) onto the stack, and then set it to a new value to manipulate local variables within the function.
- **Function epilogue** is a code added at the end of a function that performs the following tasks:
    - **Context Recovery**: Restores registers and other stack data that were saved in the prolog so that the current system state is the same as before the function was called.
    - **Stack ordering**: Frees up the space allocated for local variables and function arguments by restoring the stack pointer to its original state.
    - **Return control**: At the end of the epilogue, the code returns control to the calling function using the ret command, which may also include clearing the stack.

Answer

- **Function prologue** does not differ much: Only name of registers are different since in 64-bit architecture these registers are 64-bit length



- **Function epilogue** differs much: Here we see that in `main` function `leave` instruction is used as an alternative for long restoring pointers in 32-bit: `lea -0x8(%ebp),%esp; pop ...`

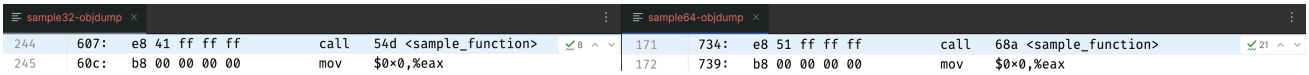


d.

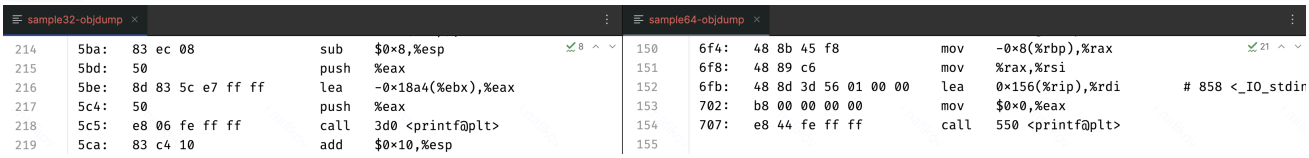
Task description

d. Do function calls differ in 32bit and 64bit? What about argument passing?

- **Function calls** look similar using `call` instruction



- **Argument passing** differs: for 32-bit arguments are pushed into the stack in reverse order while for 64-bit special registers used as the first six arguments in the following order: `%rdi` , `%rsi` , `%rdx` , `%rcx` , `%r8` , `%r9` . Also, for 32-bit we need to cleanup stack (`add $0x10, %esp`) after function call.

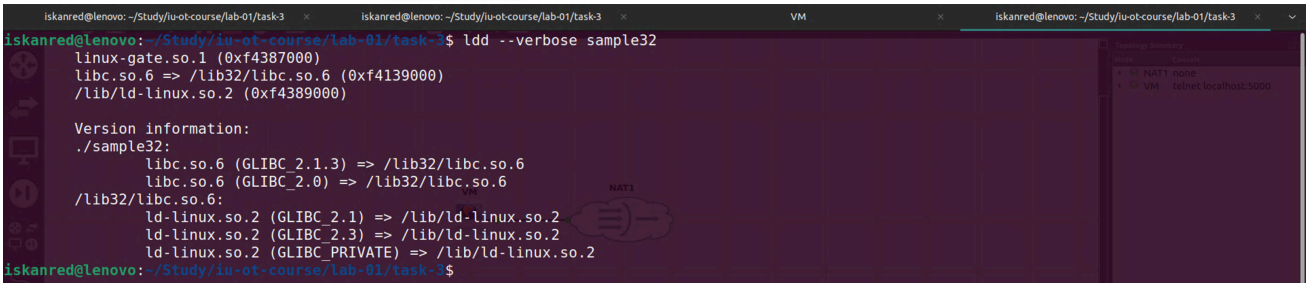


e.

Task description

e. What does the command **ldd** do? “ldd BINARY-NAME”

- The command `ldd` is a utility used to print the shared library dependencies of executables. It will display the paths of the shared libraries that the specified executable or library relies on at runtime. This command can be particularly useful for debugging issues related to missing libraries, ensuring that the correct versions of libraries are being used, and understanding how a program interacts with its dynamic dependencies.
- `ldd` is useful for dynamically linked programs and uses linker (`/lib/ld-linux-x86-64.so.2`) for discovering dependencies
- Dynamic dependencies of `sample32`



- Dynamic dependencies of `sample64`

```
iskanred@lenovo:~/Study/iu-ot-course/lab-01/task-3$ ldd --verbose sample64
        linux-vdso.so.1 (0x00007ffebd086000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ca608400000)
        /lib64/ld-linux-x86-64.so.2 (0x00007ca608cd2000)

        Version information:
        ./sample64:
                libc.so.6 (GLIBC_2.2.5) => /lib/x86_64-linux-gnu/libc.so.6
        /lib/x86_64-linux-gnu/libc.so.6:
                ld-linux-x86-64.so.2 (GLIBC_2.2.5) => /lib64/ld-linux-x86-64.so.2
                ld-linux-x86-64.so.2 (GLIBC_2.3) => /lib64/ld-linux-x86-64.so.2
                ld-linux-x86-64.so.2 (GLIBC_PRIVATE) => /lib64/ld-linux-x86-64.so.2
iskanred@lenovo:~/Study/iu-ot-course/lab-01/task-3$
```

- Dynamic dependencies of `sample64-2`

```
iskanred@lenovo:~/Study/iu-ot-course/lab-01/task-3$ ldd --verbose sample64-2
        linux-vdso.so.1 (0x00007ffddf026000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x0000792498a00000)
        /lib64/ld-linux-x86-64.so.2 (0x0000792499353000)

        Version information:
        ./sample64-2:
                libc.so.6 (GLIBC_2.2.5) => /lib/x86_64-linux-gnu/libc.so.6
                libc.so.6 (GLIBC_2.4) => /lib/x86_64-linux-gnu/libc.so.6
        /lib/x86_64-linux-gnu/libc.so.6:
                ld-linux-x86-64.so.2 (GLIBC_2.2.5) => /lib64/ld-linux-x86-64.so.2
                ld-linux-x86-64.so.2 (GLIBC_2.3) => /lib64/ld-linux-x86-64.so.2
                ld-linux-x86-64.so.2 (GLIBC_PRIVATE) => /lib64/ld-linux-x86-64.so.2
iskanred@lenovo:~/Study/iu-ot-course/lab-01/task-3$
```

- We see that used dynamic libraries are pretty similar for the different architectures and are the same for the same architecture.

f.

Task description

f. Why in the “sample64-2“ binary, the value of **i** didn’t change even if our input was very long?

- This is because enabled stack protection canaries which makes it impossible

```
sample64-objdump
149   6ef:   e8 6c fe ff ff          call   560 <gets@plt>
150   6f4:   48 8b 45 f8             mov    -0x8(%rbp),%rax
151   6f8:   48 89 c6                mov    %rax,%rsi
152   6fb:   48 8d 3d 56 01 00 00    lea    0x156(%rip),%rdi        # 858 <_IO_s
153   702:   b8 00 00 00 00          mov    $0x0,%eax
154   707:   e8 44 fe ff ff          call   550 <printf@plt>
155   70c:   90                      nop
156   70d:   c9                      leave
157   70e:   c3                      ret

sample64-2-objdump
163   786:   e8 35 fe ff ff          call   5c0 <printf@plt>
164   78b:   90                      nop
165   78c:   48 8b 45 f8             mov    -0x8(%rbp),%rax
166   790:   64 48 33 04 25 28 00    xor    %fs:0x28,%rax
167   797:   00 00
168   799:   74 05                   je     7a0 <sample_function+0xa6>
169   79b:   e8 10 fe ff ff          call   5b0 <__stack_chk_fail@plt>
170   7a0:   c9                      leave
171   7a1:   c3                      ret
```

- To prove it I tried to compile a similar program with flag `-fstack-protector` and with flag `-fno-stack-protector`

```c
#include <stdio.h>
int main() {
  char buffer[10];
  long long i = 0xffffff;

  printf("%llx\n", i);
  gets(&buffer);
  printf("%llx\n", i);
```

```
    return 0;
}
```

```
iskanred@lenovo:~/Study/iu-ot-course/lab-01/task-3$ cat my.c
#include <stdio.h>

int main() {
  char buffer[10];
  long long i = 0xffffff;

  printf("%llx\n", i);
  gets(&buffer);
  printf("%llx\n", i);

  return 0;
}
iskanred@lenovo:~/Study/iu-ot-course/lab-01/task-3$
```

- Then I compiled it with `-fstack-protector` flag and ran with input that triggers buffer overflow

```
iskanred@lenovo:~/Study/iu-ot-course/lab-01/task-3$ gcc my.c -o my -fstack-protector
my.c: In function ‘main’:
my.c:8:3: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’? [-Wimplicit-function-declaration]
    8 |   gets(&buffer);
      |   ^~~~
      |   fgets
/usr/bin/ld: /tmp/ccUUpPWN.o: in function `main':
my.c:(.text+0x4b): warning: the `gets' function is dangerous and should not be used.
iskanred@lenovo:~/Study/iu-ot-course/lab-01/task-3$ ./my
ffffff
aaaaaaaaaaaaaaaaaaaaaa
ffffff
*** stack smashing detected ***: terminated
Aborted (core dumped)
iskanred@lenovo:~/Study/iu-ot-course/lab-01/task-3$
```

- Then I compiled it with `-fno-stack-protector` flag and ran with input that triggers buffer overflow

```
iskanred@lenovo:~/Study/iu-ot-course/lab-01/task-3$ gcc my.c -o my -fno-stack-protector
my.c: In function ‘main’:
my.c:8:3: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’? [-Wimplicit-function-declaration]
    8 |   gets(&buffer);
      |   ^~~~
      |   fgets
/usr/bin/ld: /tmp/ccufdcUn.o: in function `main':
my.c:(.text+0x3c): warning: the `gets' function is dangerous and should not be used.
iskanred@lenovo:~/Study/iu-ot-course/lab-01/task-3$ ./my
ffffff
aaaaaaaaaaaaaaaaaaaaaa
6161616161616161
iskanred@lenovo:~/Study/iu-ot-course/lab-01/task-3$
```

4. Reversing

---

a.

> **Task description**
>
> a. You will have multiple binaries **in the Task 4 folder**, Try to reverse them by recreating them using any programming language of your choice (C is preferred)

- Using the learned techniques (scanning binaries, debugging and disassembling) I tried to recreate the C source for the given binaries

- Although the programs are difficult to debug (they are stripped with no debug info), I got the following results:
  - `bin1`

```c
#include <stdio.h>
#include <time.h>

int main() {
    time_t t;
    time(&t);
    struct tm *tm_info = localtime(&t);

    printf("%04d-%02d-%02d %02d:%02d:%02d\n",
           tm_info->tm_year + 1900,
           tm_info->tm_mon + 1,
           tm_info->tm_mday,
           tm_info->tm_hour,
           tm_info->tm_min,
           tm_info->tm_sec);

    return 0;
}
```

  - `bin2`

```c
#include <stdio.h>

int main() {
    int a[20];

    for (int i = 0; i < 20; i++) {
        a[i] = i * 2;
    }

    for (int i = 0; i < 20; i++) {
        printf("a[%d]=%d\n", i, a[i]);
    }

    return 0;
}
```

  - `bin3` : identical to `bin2`
  - `bin4`

```c
#include <stdio.h>

int main() {
    int number;
    printf("Enter an integer: ");
    scanf("%d", &number);

    if (number % 2 != 0) {
        printf("%d is odd.", number);
    } else {
        printf("%d is even.", number);
    }

    return 0;
}
```

- `bin5`

```c
#include <stdio.h>
int main() {
    int number;
    long long factorial = 1;

    printf("Enter an integer: ");
    scanf("%d", &number);

    if (number < 0) {
        printf("Error!");
    } else {
        for (int i = 1; i <= number; i++) {
            factorial *= i;
        }
        printf("Result is %d = %lld", number, factorial);
    }

    return 0;
}
```

- `bin6`

```c
#include <stdio.h>

int main() {
    int a, b;
```

```
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    printf("%d + %d = %d", a, b, a + b);

    return 0;
}
    ```
```

# References


1. https://stackoverflow.com/questions/54747917/difference-between-aslr-and-pie ↩
```