

Pluto Obfuscator

Pluto obfuscator:-

- *Pluto is an **open-source** code obfuscator based on the **LLVM** compiler infrastructure. It is designed to protect software from being **reverse engineered** or **cracked** by making the code harder to understand and analyze. Obfuscation works by transforming the code into a functionally equivalent but more complex form, making it difficult for attackers to extract the original logic or functionality.*

Purpose:-

- *The purpose of this guide is to demonstrate how to obfuscate C code using **Pluto Obfuscator** on Ubuntu 20.04, targeting the **x86_64** architecture*

Pluto Obfuscator Installation and Usage Guide

Prerequisites:-

- Ubuntu 20.04.3 LTS
- CMake 3.16.3
- Ninja 1.10.0
- Git
- LLVM 12.0.1 and clang 12.0.1 (optional)



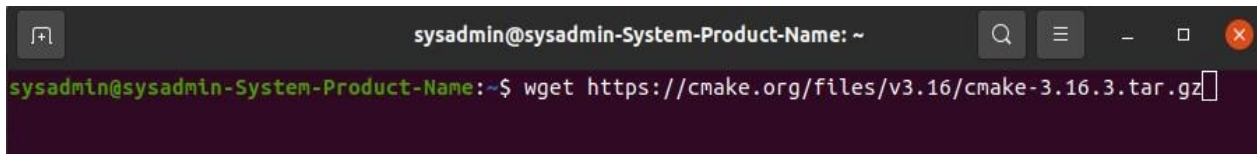
I. Installing CMake 3.16.3 on Ubuntu 20.04 **what is cmake?**

- CMake is an open-source cross-platform **build system** that manages the build process in a **compiler-independent manner**. It generates build files such as **Makefiles** or project files for various **Integrated Development Environments** (IDEs) based on a simple script format. With CMake, developers can define their build process in a **clear** and **platform-independent way**, making it easier to maintain and port projects across **different platforms** and **development environments**.

Step 1: Update Package Lists

A screenshot of a terminal window with a dark background. The prompt is 'sysadmin@sysadmin-System-Product-Name: ~'. The command 'sysadmin@sysadmin-System-Product-Name:~\$ sudo apt-get update' has been entered, and the cursor is at the end of the line.

```
sudo apt-get update
```



```
sysadmin@sysadmin-System-Product-Name: ~  
sysadmin@sysadmin-System-Product-Name:~$ wget https://cmake.org/files/v3.16/cmake-3.16.3.tar.gz
```

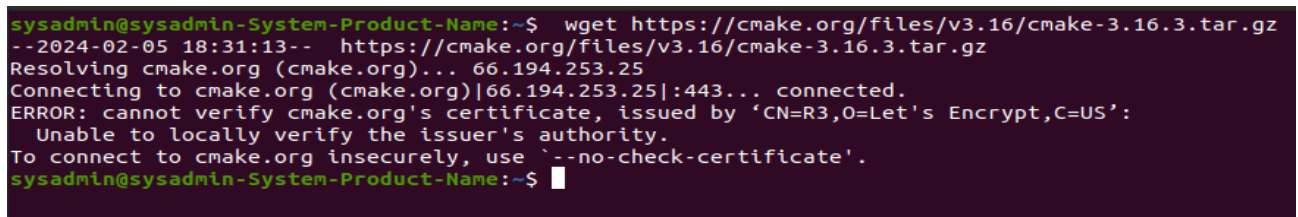
Step 2: Download CMake 3.16.3 Source Code

```
wget https://cmake.org/files/v3.16/cmake-3.16.3.tar.gz
```

Note: When you get an error like this below, just take the following steps.

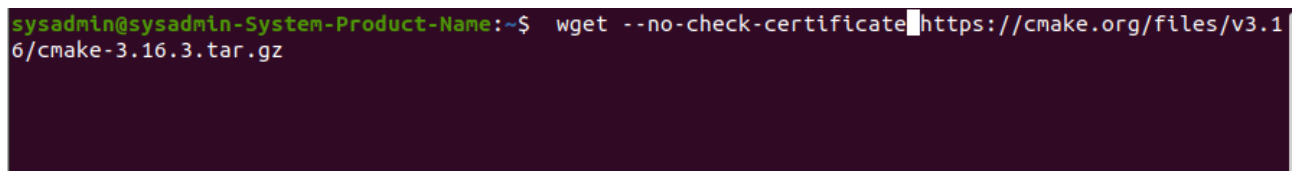
```
ERROR: cannot verify github.com's certificate, issued by 'CN=DigiCert TLS Hybrid  
ECC SHA384 2020 CA1,O=DigiCert Inc,C=US':
```

Unable to locally verify the issuer's authority.



```
sysadmin@sysadmin-System-Product-Name:~$ wget https://cmake.org/files/v3.16/cmake-3.16.3.tar.gz  
--2024-02-05 18:31:13-- https://cmake.org/files/v3.16/cmake-3.16.3.tar.gz  
Resolving cmake.org (cmake.org)... 66.194.253.25  
Connecting to cmake.org (cmake.org)|66.194.253.25|:443... connected.  
ERROR: cannot verify cmake.org's certificate, issued by 'CN=R3,O=Let's Encrypt,C=US':  
Unable to locally verify the issuer's authority.  
To connect to cmake.org insecurely, use '--no-check-certificate'.  
sysadmin@sysadmin-System-Product-Name:~$
```

-->To resolve the issue just run the below command.

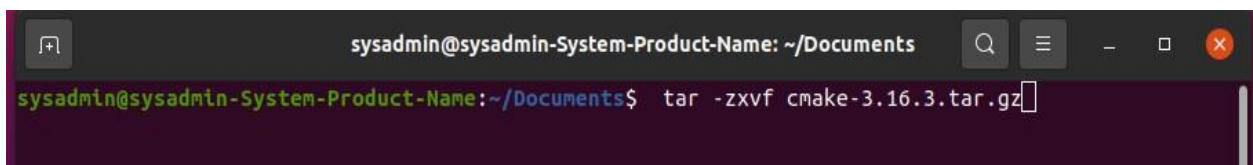


```
sysadmin@sysadmin-System-Product-Name:~$ wget --no-check-certificate https://cmake.org/files/v3.16/cmake-3.16.3.tar.gz
```

```
wget --no-check-certificate https://cmake.org/files/v3.16/cmake-3.16.3.tar.gz
```

--no-check-certificate -> if you encounter SSL certificate verification issues while using wget, you can bypass certificate checking by using the --no-check-certificate option

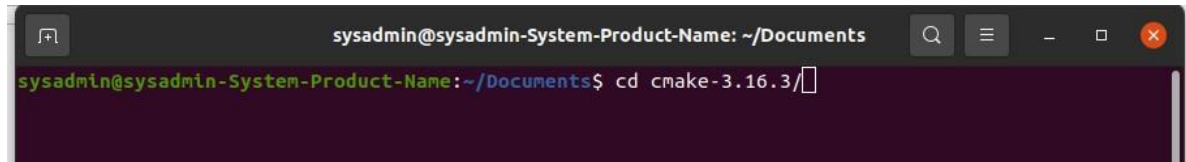
Step 3: Extract the Archive



```
sysadmin@sysadmin-System-Product-Name: ~/Documents  
sysadmin@sysadmin-System-Product-Name:~/Documents$ tar -zxvf cmake-3.16.3.tar.gz
```

```
tar -zxvf cmake-3.16.3.tar.gz
```

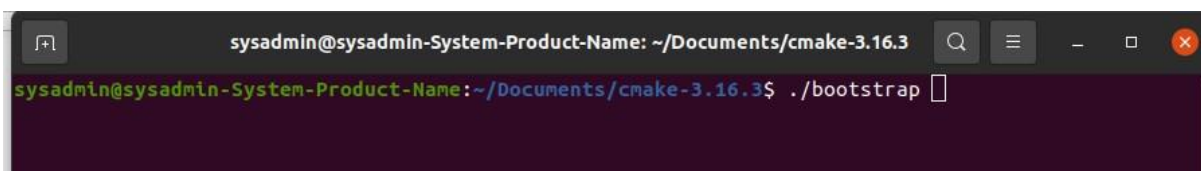
Step 4: Navigate to the CMake Directory

A terminal window with a dark background. The title bar shows 'sysadmin@sysadmin-System-Product-Name: ~/Documents'. The prompt is 'sysadmin@sysadmin-System-Product-Name:~/Documents\$'. The command 'cd cmake-3.16.3/' is entered and executed, with a cursor at the end of the line.

```
sysadmin@sysadmin-System-Product-Name:~/Documents$ cd cmake-3.16.3/
```

`cd cmake-3.16.3`

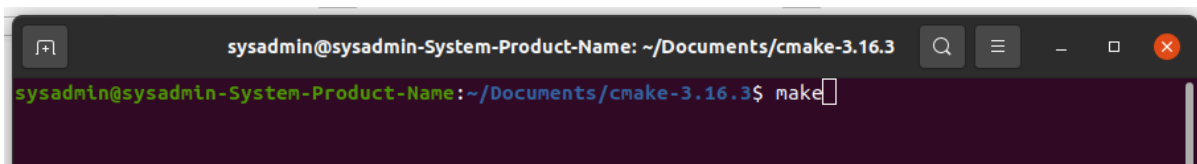
Step 5: Run the Bootstrap Script

A terminal window with a dark background. The title bar shows 'sysadmin@sysadmin-System-Product-Name: ~/Documents/cmake-3.16.3'. The prompt is 'sysadmin@sysadmin-System-Product-Name:~/Documents/cmake-3.16.3\$'. The command './bootstrap' is entered and executed, with a cursor at the end of the line.

```
sysadmin@sysadmin-System-Product-Name:~/Documents/cmake-3.16.3$ ./bootstrap
```

`./bootstrap`

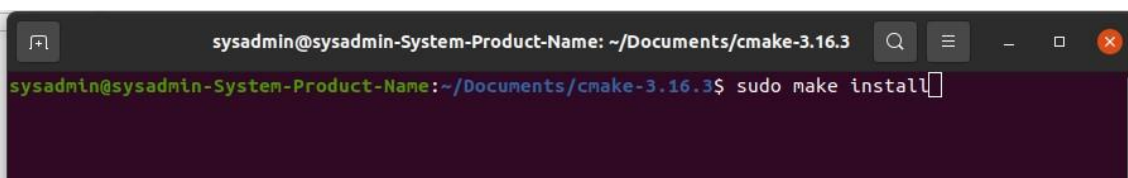
Step 6: Build Cmake

A terminal window with a dark background. The title bar shows 'sysadmin@sysadmin-System-Product-Name: ~/Documents/cmake-3.16.3'. The prompt is 'sysadmin@sysadmin-System-Product-Name:~/Documents/cmake-3.16.3\$'. The command 'make' is entered and executed, with a cursor at the end of the line.

```
sysadmin@sysadmin-System-Product-Name:~/Documents/cmake-3.16.3$ make
```

`make`

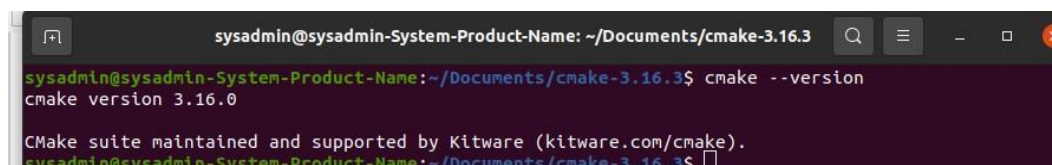
Step 7: Install Cmake

A terminal window with a dark background. The title bar shows 'sysadmin@sysadmin-System-Product-Name: ~/Documents/cmake-3.16.3'. The prompt is 'sysadmin@sysadmin-System-Product-Name:~/Documents/cmake-3.16.3\$'. The command 'sudo make install' is entered and executed, with a cursor at the end of the line.

```
sysadmin@sysadmin-System-Product-Name:~/Documents/cmake-3.16.3$ sudo make install
```

`sudo make install`

Step 8: Verify the Installation

A terminal window with a dark background. The title bar shows 'sysadmin@sysadmin-System-Product-Name: ~/Documents/cmake-3.16.3'. The prompt is 'sysadmin@sysadmin-System-Product-Name:~/Documents/cmake-3.16.3\$'. The command 'cmake --version' is entered and executed. The output shows 'cmake version 3.16.0' and 'CMake suite maintained and supported by Kitware (kitware.com/cmake)'.

```
sysadmin@sysadmin-System-Product-Name:~/Documents/cmake-3.16.3$ cmake --version
cmake version 3.16.0

CMake suite maintained and supported by Kitware (kitware.com/cmake).
```

`cmake --version`

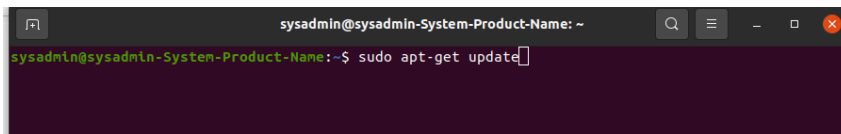
This should display the version number of CMake, confirming the successful installation.

II. Installing Ninja 1.10.0 on Ubuntu 20.04

what is Ninja ?

- Ninja is a fast and efficient **build system** designed for **speed** and **scalability**. It is commonly used as a replacement for other build tools like **Make** due to its superior performance. Ninja utilizes a simple build file format and parallelizes builds to take full advantage of modern hardware, making it particularly well-suited for **large projects** with complex dependencies.

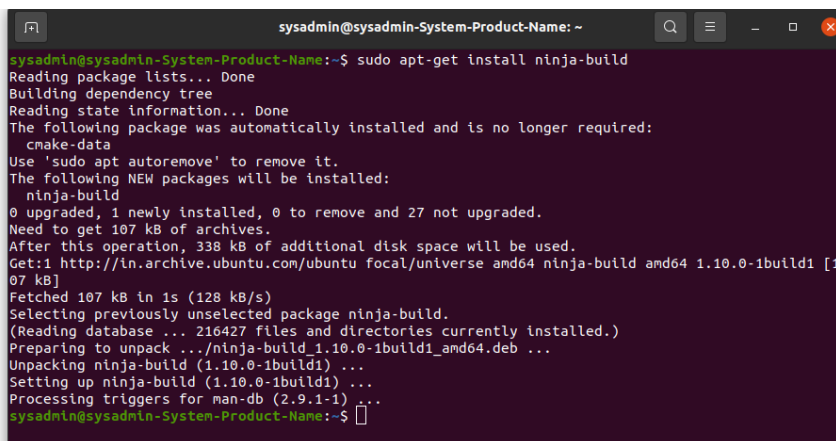
Step 1: Update Package Lists



```
sysadmin@sysadmin-System-Product-Name: ~  
sysadmin@sysadmin-System-Product-Name:~$ sudo apt-get update
```

sudo apt-get update

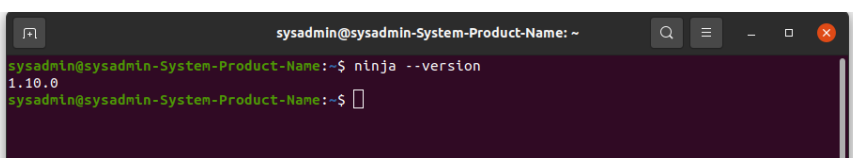
Step 2: Install the Ninja Build



```
sysadmin@sysadmin-System-Product-Name: ~  
sysadmin@sysadmin-System-Product-Name:~$ sudo apt-get install ninja-build  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following package was automatically installed and is no longer required:  
  cmake-data  
Use 'sudo apt autoremove' to remove it.  
The following NEW packages will be installed:  
  ninja-build  
0 upgraded, 1 newly installed, 0 to remove and 27 not upgraded.  
Need to get 107 kB of archives.  
After this operation, 338 kB of additional disk space will be used.  
Get:1 http://in.archive.ubuntu.com/ubuntu focal/universe amd64 ninja-build amd64 1.10.0-1build1 [107 kB]  
Fetched 107 kB in 1s (128 kB/s)  
Selecting previously unselected package ninja-build.  
(Reading database ... 216427 files and directories currently installed.)  
Preparing to unpack .../ninja-build_1.10.0-1build1_amd64.deb ...  
Unpacking ninja-build (1.10.0-1build1) ...  
Setting up ninja-build (1.10.0-1build1) ...  
Processing triggers for man-db (2.9.1-1) ...  
sysadmin@sysadmin-System-Product-Name:~$
```

sudo apt install ninja-build

Step 3: Verify the Installation



```
sysadmin@sysadmin-System-Product-Name: ~  
sysadmin@sysadmin-System-Product-Name:~$ ninja --version  
1.10.0  
sysadmin@sysadmin-System-Product-Name:~$
```

```
ninja --version
```

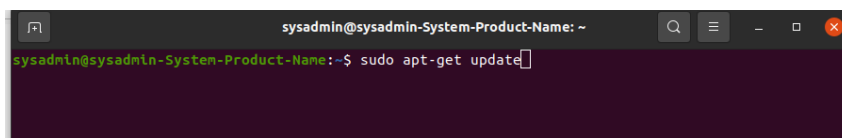
*This should display the version number of Ninja , confirming the successful installation.

III. Installing Git on Ubuntu 20.04

what is git?

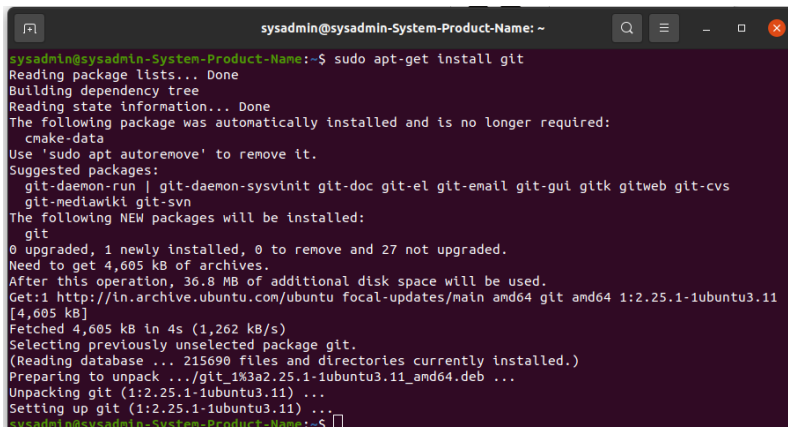
- Git is a **version control system** designed to track changes in files and coordinate work among multiple people. It allows developers to efficiently collaborate on projects by **managing revisions, branching, and merging** of code. With Git, users can track the history of changes, revert to previous versions, and work concurrently on different parts of a project. It has become a fundamental tool in software development, enabling teams to effectively manage and maintain codebases of any size.

Step 1: Update Package Lists

A terminal window with a dark background. The prompt is 'sysadmin@sysadmin-System-Product-Name: ~'. The command 'sudo apt-get update' has been entered and is followed by a cursor.

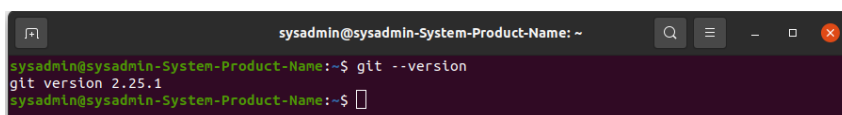
```
sudo apt-get update
```

Step 2: Install the Git

A terminal window showing the output of 'sudo apt-get install git'. The output includes: 'Reading package lists... Done', 'Building dependency tree', 'Reading state information... Done', 'The following package was automatically installed and is no longer required: cmake-data', 'Use \'sudo apt autoremove\' to remove it.', 'Suggested packages: git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitk gitweb git-cvs git-mediawiki git-svn', 'The following NEW packages will be installed: git', '0 upgraded, 1 newly installed, 0 to remove and 27 not upgraded.', 'Need to get 4,605 kB of archives.', 'After this operation, 36.8 MB of additional disk space will be used.', 'Get:1 http://in.archive.ubuntu.com/ubuntu focal-updates/main amd64 git amd64 1:2.25.1-1ubuntu3.11 [4,605 kB]', 'Fetched 4,605 kB in 4s (1,262 kB/s)', 'Selecting previously unselected package git.', '(Reading database ... 215690 files and directories currently installed.)', 'Preparing to unpack .../git_1%3a2.25.1-1ubuntu3.11_amd64.deb ...', 'Unpacking git (1:2.25.1-1ubuntu3.11) ...', 'Setting up git (1:2.25.1-1ubuntu3.11) ...', and the prompt 'sysadmin@sysadmin-System-Product-Name:~\$'.

```
sudo apt-get install git
```

Step 3: Verify the Installation

A terminal window showing the output of 'git --version'. The output is 'git version 2.25.1'. The prompt is 'sysadmin@sysadmin-System-Product-Name:~\$'.

git --version

IV. LLVM and Clang



what is LLVM ?

- LLVM is a versatile compiler infrastructure project initially developed at the University of Illinois. It offers a suite of tools and libraries for building compilers, code analyzers, and other programming tools. Known for its modular architecture and intermediate representation (IR), LLVM facilitates efficient code generation and optimization across different programming languages and platforms

what is clang?

- Clang is a high-performance compiler front end for C, C++, and Objective-C languages, part of the LLVM project. It's known for its speed, adherence to language standards, and advanced features like enhanced diagnostics and integration with development tools.

V. Download the source code from github

Step1: Update Package Lists

```
sysadmin@sysadmin-System-Product-Name: ~  
sysadmin@sysadmin-System-Product-Name:~$ sudo apt-get update
```

sudo apt-get update

Step1: Clone a Source Code from github:

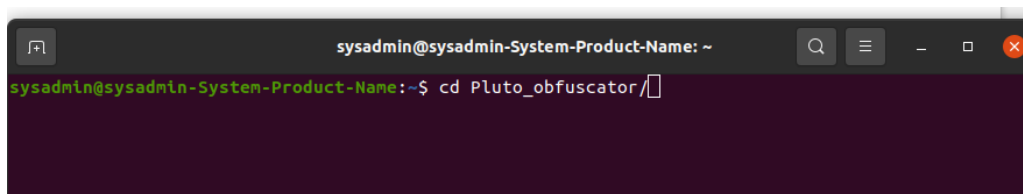
Clone the Pluto Obfuscator repository from GitHub by using below command in terminal:

```
sysadmin@sysadmin-System-Product-Name: ~  
sysadmin@sysadmin-System-Product-Name:~$ git clone https://github.com/XXTouchNG/Pluto-Obfuscator.  
git
```

git clone <https://github.com/XXTouchNG/Pluto-Obfuscator.git>

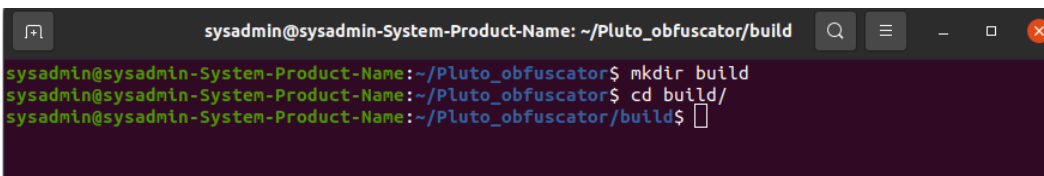
Step2: Build Directory:

Create a build directory within the Pluto Obfuscator source directory:



```
sysadmin@sysadmin-System-Product-Name: ~  
sysadmin@sysadmin-System-Product-Name:~$ cd Pluto_obfuscator/
```

cd Pluto_obfuscator

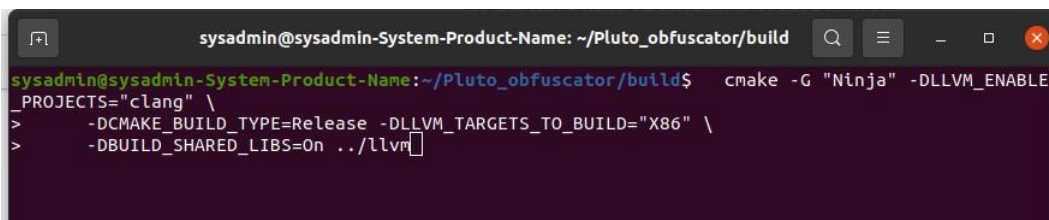


```
sysadmin@sysadmin-System-Product-Name: ~/Pluto_obfuscator/build  
sysadmin@sysadmin-System-Product-Name:~/Pluto_obfuscator$ mkdir build  
sysadmin@sysadmin-System-Product-Name:~/Pluto_obfuscator$ cd build/  
sysadmin@sysadmin-System-Product-Name:~/Pluto_obfuscator/build$
```

**mkdir build
cd build**

Step3: Generate Build System:

Configure the build system with Cmake:

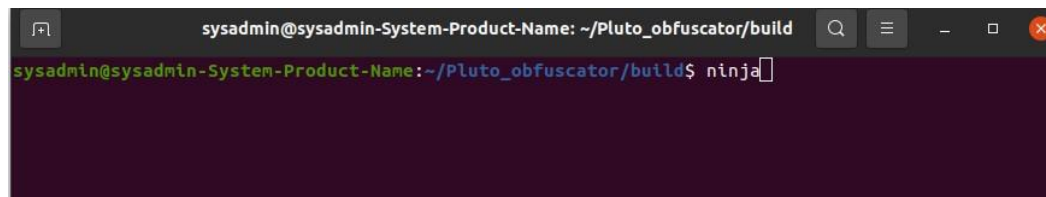


```
sysadmin@sysadmin-System-Product-Name: ~/Pluto_obfuscator/build  
sysadmin@sysadmin-System-Product-Name:~/Pluto_obfuscator/build$ cmake -G "Ninja" -DLLVM_ENABLE_PROJECTS="clang" \  
> -DCMAKE_BUILD_TYPE=Release -DLLVM_TARGETS_TO_BUILD="X86" \  
> -DBUILD_SHARED_LIBS=On ../llvm
```

```
cmake -G "Ninja" -DLLVM_ENABLE_PROJECTS="clang" \  
-DCMAKE_BUILD_TYPE=Release -DLLVM_TARGETS_TO_BUILD="X86" \  
-DBUILD_SHARED_LIBS=On ../llvm
```

Step4: Build Pluto Obfuscator:

Build the source code using Ninja:

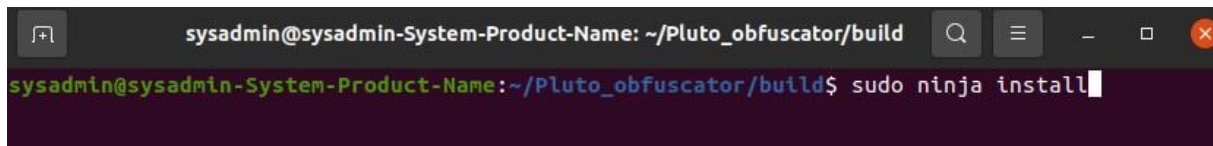


```
sysadmin@sysadmin-System-Product-Name: ~/Pluto_obfuscator/build
sysadmin@sysadmin-System-Product-Name:~/Pluto_obfuscator/build$ ninja
```

ninja

Step5: Install Pluto Obfuscator:

Install the Pluto Obfuscator binaries as root:



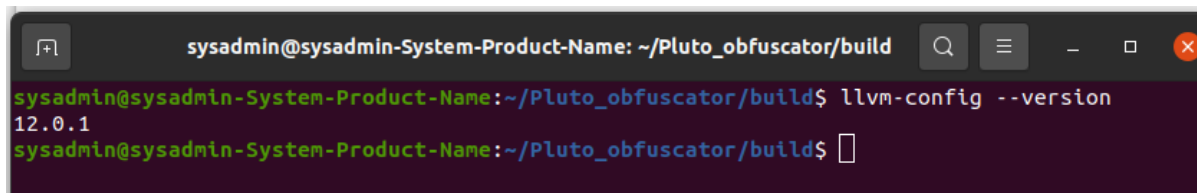
```
sysadmin@sysadmin-System-Product-Name: ~/Pluto_obfuscator/build
sysadmin@sysadmin-System-Product-Name:~/Pluto_obfuscator/build$ sudo ninja install
```

sudo ninja install

VI. Verify:

Verify installation by checking the version of llvm-config and clang:


i)verify the llvm installation



```
sysadmin@sysadmin-System-Product-Name: ~/Pluto_obfuscator/build
sysadmin@sysadmin-System-Product-Name:~/Pluto_obfuscator/build$ llvm-config --version
12.0.1
sysadmin@sysadmin-System-Product-Name:~/Pluto_obfuscator/build$
```

llvm -config --version

ii)verify the clang installation



```
sysadmin@sysadmin-System-Product-Name: ~/Pluto_obfuscator/build
sysadmin@sysadmin-System-Product-Name:~/Pluto_obfuscator/build$ clang --version
clang version 12.0.1 (https://github.com/XXTouchNG/Pluto-Obfuscator.git e7561e6a3207e801fd3d
b2dad986ab029348d26a)
Target: x86_64-unknown-linux-gnu
Thread model: posix
InstalledDir: /usr/local/bin
```

clang --version

*The output will tell you if the tools are installed and their versions.

Features:

- Control Flow Flattening
- Bogus Control Flow
- Instruction Substitution
- Random Control Flow
- Variable Substitution
- String Encryption
- Globals Encryption
- MBA Obfuscation

Obfuscation Flags:

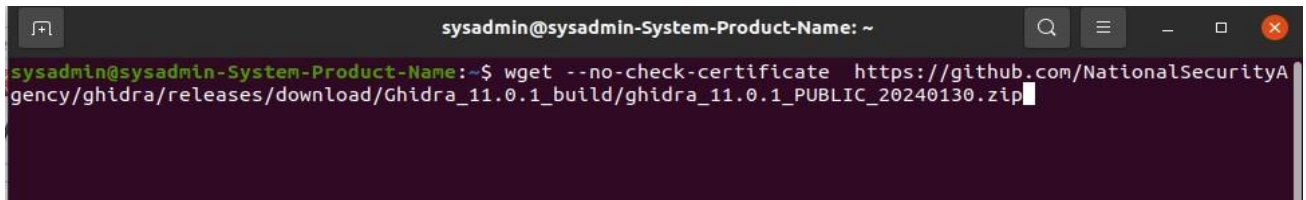
- Flattening: `-O2 -mllvm -fla`
- BogusControlFlow: `-O2 -mllvm -bcf`
- Substitution: `-O2 -mllvm -sub`
- GlobalsEncryption: `-O2 -mllvm -gle`
- MBAObfuscation: `-O2 -mllvm -mba -mllvm -mba-prob=100`
- FullProtection (**HIGHLY RECOMMENDED**): `-s -mllvm -mba -mllvm -mba-prob=50 -mllvm -fla -mllvm -gle`

Ghidra

- **Ghidra** is an open-source software **reverse engineering** suite developed by the NSA(National Security Agency). Released in 2019, it provides tools for analyzing **compiled code**, including **disassembly**, **decompilation**, and **debugging**. With a user-friendly interface, scripting support, and collaboration features, Ghidra is widely used in **cybersecurity** for tasks such as **malware analysis** and **vulnerability research**.

Install Ghidra:-

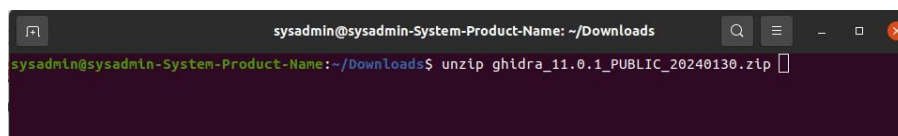
Step:1 Download Ghidra



```
sysadmin@sysadmin-System-Product-Name: ~  
sysadmin@sysadmin-System-Product-Name:~$ wget --no-check-certificate https://github.com/NationalSecurityAgency/ghidra/releases/download/Ghidra_11.0.1_build/ghidra_11.0.1_PUBLIC_20240130.zip
```

```
wget --no-check-certificate  
https://github.com/NationalSecurityAgency/ghidra/releases/download/Ghidra_11.0.1_build/ghidra_11.0.1_PUBLIC_20240130.zip
```

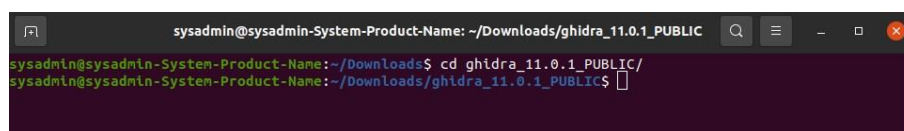
Step:2 Extract the ZIP File:



```
sysadmin@sysadmin-System-Product-Name: ~/Downloads  
sysadmin@sysadmin-System-Product-Name:~/Downloads$ unzip ghidra_11.0.1_PUBLIC_20240130.zip
```

- Open a terminal.
- Navigate to the directory containing the downloaded ZIP file.
- Run `unzip ghidra_11.0.1_PUBLIC_20240130.zip` to extract the contents.

Step:3 Navigate to Ghidra Directory:



```
sysadmin@sysadmin-System-Product-Name: ~/Downloads/ghidra_11.0.1_PUBLIC  
sysadmin@sysadmin-System-Product-Name:~/Downloads$ cd ghidra_11.0.1_PUBLIC/  
sysadmin@sysadmin-System-Product-Name:~/Downloads/ghidra_11.0.1_PUBLIC$
```

- Move into the Ghidra directory using `cd ghidra_*`.

Step:4 Run Ghidra

- Launch Ghidra with `./ghidraRun` in the terminal.

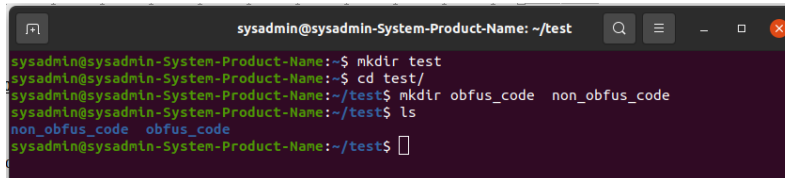
Test the obfuscation

1. Create a C file:

Write a simple C program (e.g., test.c) with a main function printing "Hello, World!".

Open terminal and run the following command

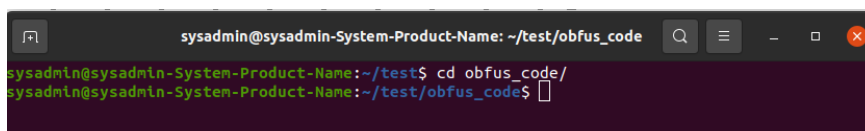
Step:1 create a directory and two subdirectory



```
sysadmin@sysadmin-System-Product-Name: ~/test
sysadmin@sysadmin-System-Product-Name:~$ mkdir test
sysadmin@sysadmin-System-Product-Name:~$ cd test/
sysadmin@sysadmin-System-Product-Name:~/test$ mkdir obfus_code non_obfus_code
sysadmin@sysadmin-System-Product-Name:~/test$ ls
non_obfus_code  obfus_code
sysadmin@sysadmin-System-Product-Name:~/test$
```

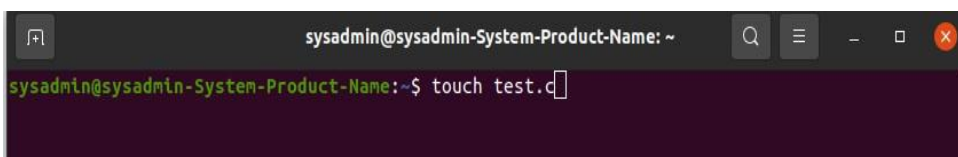
```
mkdir test
cd test
mkdir obfus_code nonobfus_code
```

Step:2 Create a c file in obfus_code directory



```
sysadmin@sysadmin-System-Product-Name: ~/test/obfus_code
sysadmin@sysadmin-System-Product-Name:~/test$ cd obfus_code/
sysadmin@sysadmin-System-Product-Name:~/test/obfus_code$
```

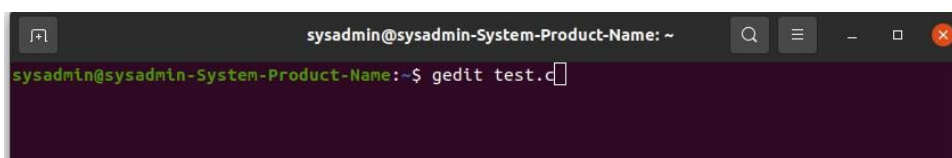
```
cd obfus_code
```



```
sysadmin@sysadmin-System-Product-Name: ~
sysadmin@sysadmin-System-Product-Name:~$ touch test.c
```

```
touch test.c
```

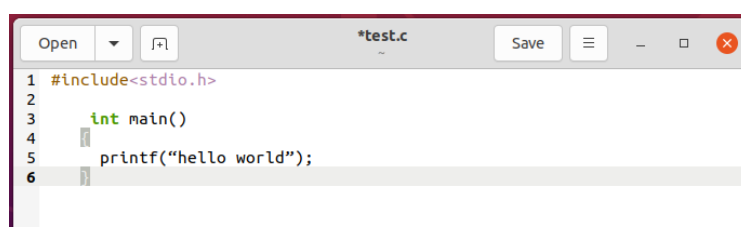
Step:2 Open a test.c file:-



```
sysadmin@sysadmin-System-Product-Name: ~
sysadmin@sysadmin-System-Product-Name:~$ gedit test.c
```

```
gedit test.c
```

Step:3 paste the below code and save it.

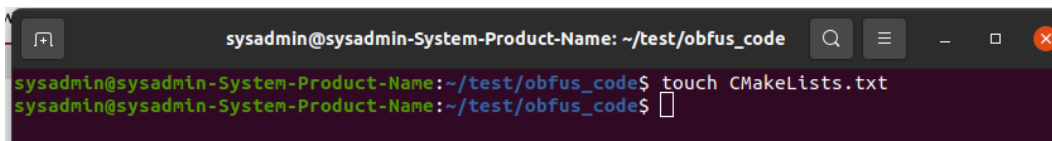


```
*test.c
1 #include<stdio.h>
2
3 int main()
4 {
5     printf("hello world");
6 }
```

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

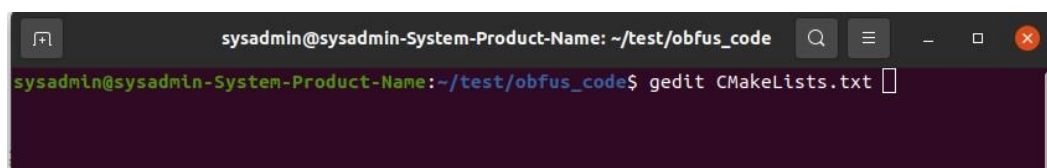
```
int main()  
{  
printf("hello world");  
}
```

Step:3 Create a Cmake file



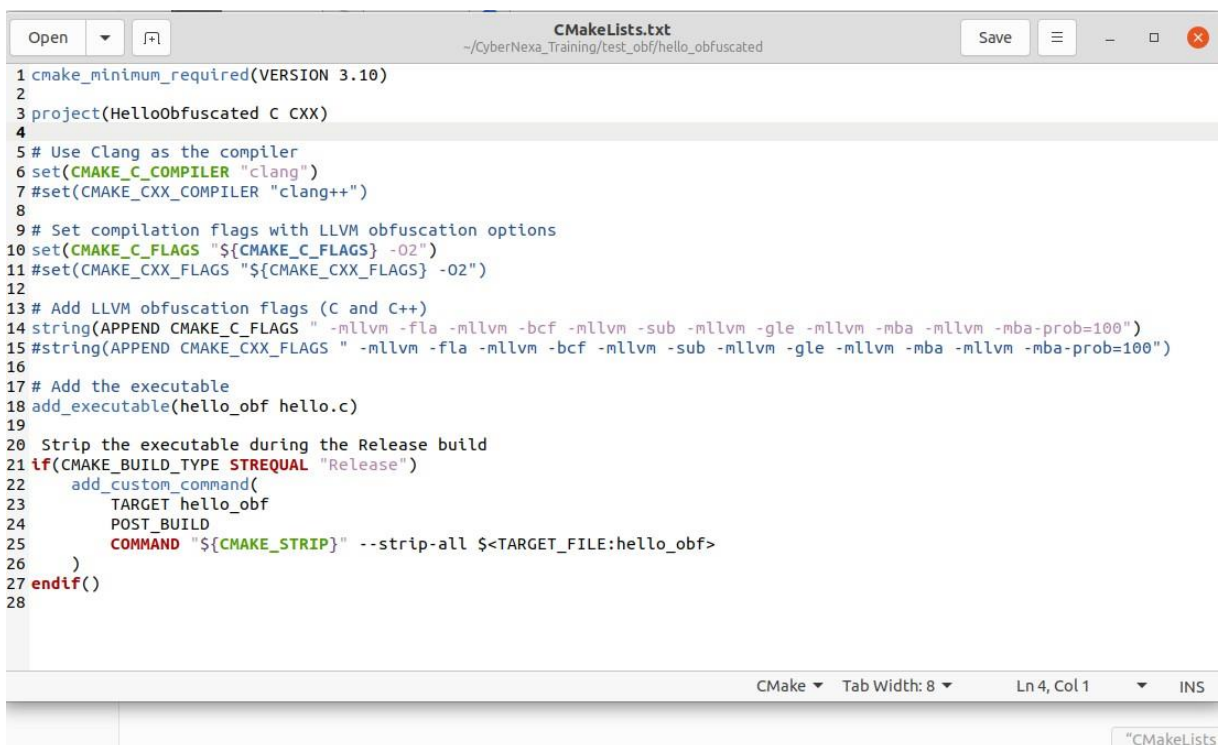
```
sysadmin@sysadmin-System-Product-Name: ~/test/obfus_code  
sysadmin@sysadmin-System-Product-Name:~/test/obfus_code$ touch CMakeLists.txt  
sysadmin@sysadmin-System-Product-Name:~/test/obfus_code$
```

touch CmakeLists.txt



```
sysadmin@sysadmin-System-Product-Name: ~/test/obfus_code  
sysadmin@sysadmin-System-Product-Name:~/test/obfus_code$ gedit CMakeLists.txt
```

gedit CmakeLists.txt



```
CMakeLists.txt  
~/CyberNexa_Training/test_obf/hello_obfuscated  
Save  
1 cmake_minimum_required(VERSION 3.10)  
2  
3 project(HelloObfuscated C CXX)  
4  
5 # Use Clang as the compiler  
6 set(CMAKE_C_COMPILER "clang")  
7 #set(CMAKE_CXX_COMPILER "clang++")  
8  
9 # Set compilation flags with LLVM obfuscation options  
10 set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -O2")  
11 #set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -O2")  
12  
13 # Add LLVM obfuscation flags (C and C++)  
14 string(APPEND CMAKE_C_FLAGS " -mllvm -fla -mllvm -bcf -mllvm -sub -mllvm -gle -mllvm -mba -mllvm -mba-prob=100")  
15 #string(APPEND CMAKE_CXX_FLAGS " -mllvm -fla -mllvm -bcf -mllvm -sub -mllvm -gle -mllvm -mba -mllvm -mba-prob=100")  
16  
17 # Add the executable  
18 add_executable(hello_obf hello.c)  
19  
20 Strip the executable during the Release build  
21 if(CMAKE_BUILD_TYPE STREQUAL "Release")  
22     add_custom_command(  
23         TARGET hello_obf  
24         POST_BUILD  
25         COMMAND "${CMAKE_STRIP}" --strip-all $<TARGET_FILE:hello_obf>  
26     )  
27 endif()  
28  
CMake Tab Width: 8 Ln 4, Col 1 INS  
"CMakeLists
```

```
cmake_minimum_required(VERSION 3.10)
project(HelloObfuscated C CXX)

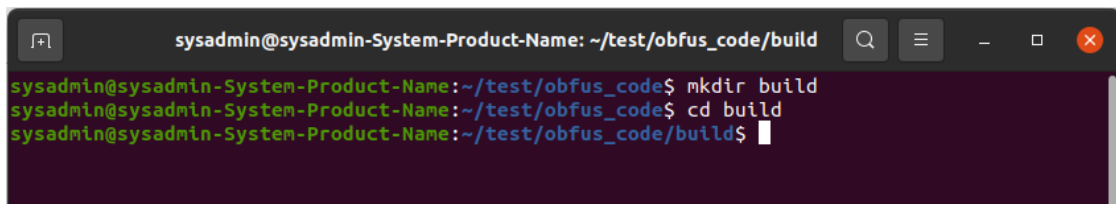
# Use Clang as the compiler
set(CMAKE_C_COMPILER "clang")
# Set compilation flags with LLVM obfuscation options
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -O2")

# Add LLVM obfuscation flags (C and C++)
string(APPEND CMAKE_C_FLAGS " -mllvm -fla -mllvm -bcf -mllvm -sub -mllvm -gle -mllvm -mba -mllvm -mba-
prob=100")

# Add the executable
add_executable(hello_obf hello.c)

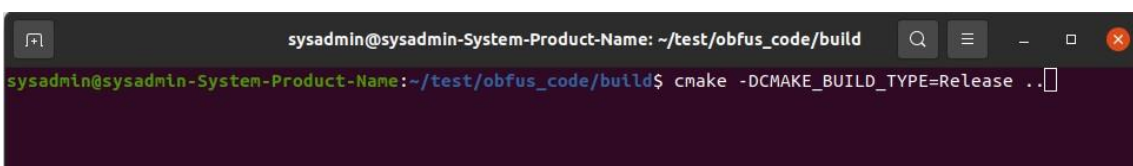
Strip the executable during the Release build
if(CMAKE_BUILD_TYPE STREQUAL "Release")
    add_custom_command(
        TARGET hello_obf
        POST_BUILD
        COMMAND "${CMAKE_STRIP}" --strip-all $<TARGET_FILE:hello_obf>
    )
endif()
```

Step:4 Create a Build Directory

A terminal window with a dark background. The title bar shows 'sysadmin@sysadmin-System-Product-Name: ~/test/obfus_code/build'. The terminal shows three commands being executed: 'mkdir build', 'cd build', and the prompt 'sysadmin@sysadmin-System-Product-Name:~/test/obfus_code/build\$'.

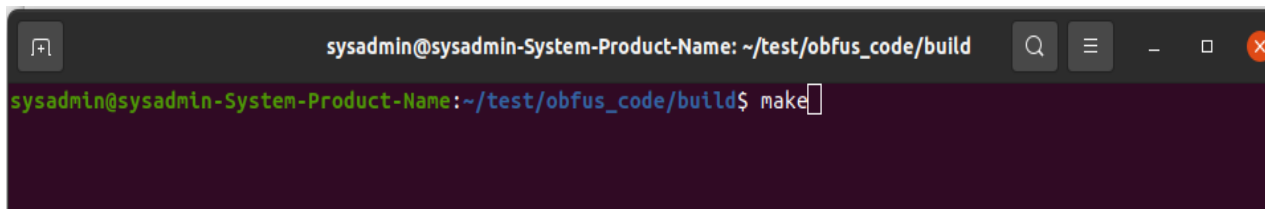
```
mkdir build
cd build
```

Step:5 compile the code

A terminal window with a dark background. The title bar shows 'sysadmin@sysadmin-System-Product-Name: ~/test/obfus_code/build'. The terminal shows the command 'cmake -DCMAKE_BUILD_TYPE=Release ..' being executed, followed by the prompt 'sysadmin@sysadmin-System-Product-Name:~/test/obfus_code/build\$'.

```
cmake -DCMAKE_BUILD_TYPE=Release ..
```

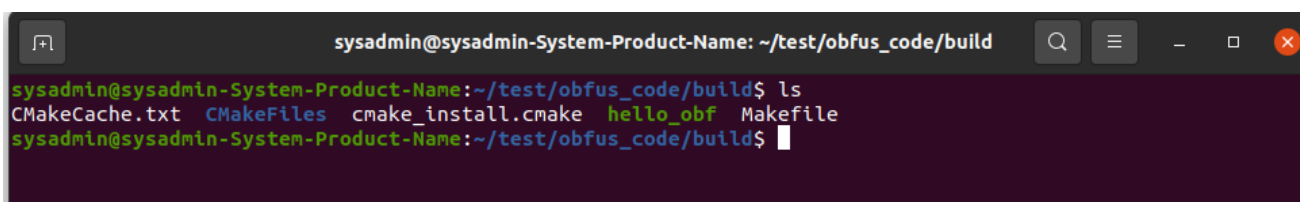
Step:6 Build



```
sysadmin@sysadmin-System-Product-Name: ~/test/obfus_code/build
sysadmin@sysadmin-System-Product-Name:~/test/obfus_code/build$ make
```

make

--> this will create a binary file hello_obf



```
sysadmin@sysadmin-System-Product-Name: ~/test/obfus_code/build
sysadmin@sysadmin-System-Product-Name:~/test/obfus_code/build$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  hello_obf  Makefile
sysadmin@sysadmin-System-Product-Name:~/test/obfus_code/build$
```

Step:7 Verify Obfuscation:

Check if the test_obfs file was created after compilation.

Use various methods to verify if the code is obfuscated:

Strings: Use the strings command to see if readable strings are present in the compiled binary.

Ghidra: Tools like Ghidra can help analyze the assembly code for obfuscation indicators.

Strings command

- The **strings** command in Linux is used to **extract human-readable** text strings from **binary files**. These strings could be anything from printable characters, such as error messages or variable names, to more **sensitive information** like hardcoded **passwords** or **cryptographic keys**.

Here's how the strings command is typically used:

strings [options] filename

Let's we check with our hello_obf binary file



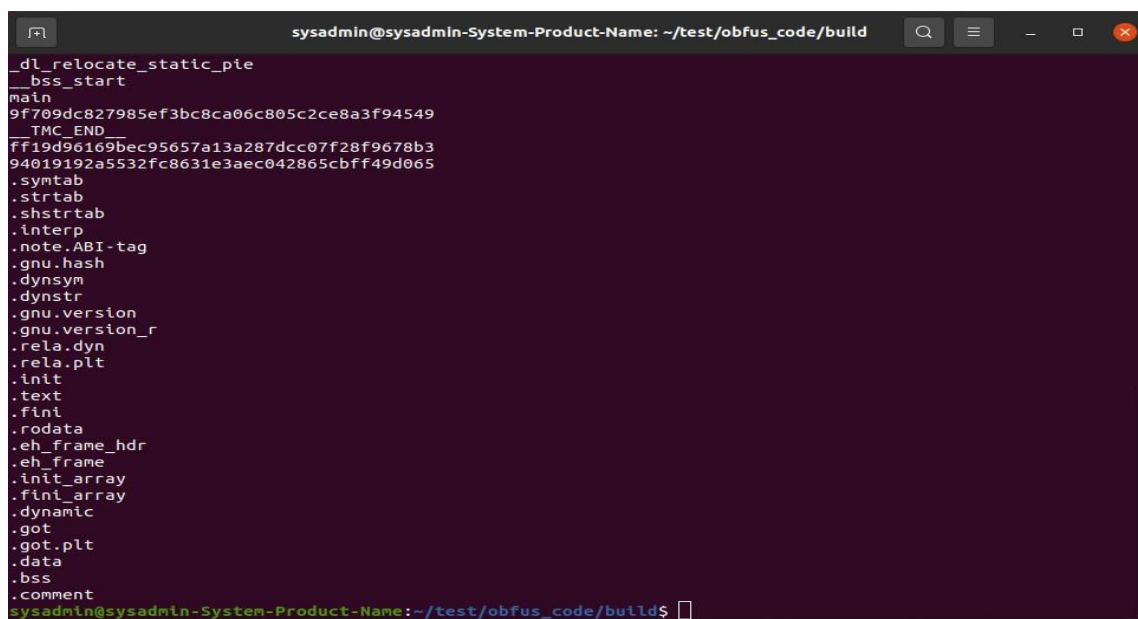
```
sysadmin@sysadmin-System-Product-Name: ~/test/obfus_code/build
sysadmin@sysadmin-System-Product-Name:~/test/obfus_code/build$ strings hello_obf
```

strings hello_obf



```
sysadmin@sysadmin-System-Product-Name: ~/test/obfus_code/build
sysadmin@sysadmin-System-Product-Name:~/test/obfus_code/build$ strings hello_obf
/lib64/ld-linux-x86-64.so.2
libc.so.6
printf
__libc_start_main
GLIBC_2.2.5
__gmon_start__
H=X@@
y\BZ
[JA\A]A^A_
;*3$
BOFFE*
<yxGCC: (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0
clang version 12.0.1 (https://github.com/XXTouchNG/Pluto-Obfuscator.git e7561e6a3207e801fd3db2dad986ab029348d26a)
crtstuff.c
deregister_tm_clones
_do_global_dtors_aux
```

*The continuation of the terminal image is below.



```
sysadmin@sysadmin-System-Product-Name: ~/test/obfus_code/build
_dl_relocate_static_pie
_bss_start
main
9f709dc827985ef3bc8ca06c805c2ce8a3f94549
_TMC_END_
ff19d96169bec95657a13a287dcc07f28f9678b3
94019192a5532fc8631e3aec042865cbff49d065
.symtab
.strtab
.shstrtab
.interp
.note.ABI-tag
.gnu.hash
.dynsym
.dynstr
.gnu.version
.gnu.version_r
.rela.dyn
.rela.plt
.init
.text
.fini
.rodata
.eh_frame_hdr
.eh_frame
.init_array
.fini_array
.dynamic
.got
.got.plt
.data
.bss
.comment
sysadmin@sysadmin-System-Product-Name:~/test/obfus_code/build$
```

*We can't see any "hello world" strings right, So we can ensure its obfuscated.
If you test this with non obfuscated binary, you can see the strings like "hello world".

Ghidra Tool:-

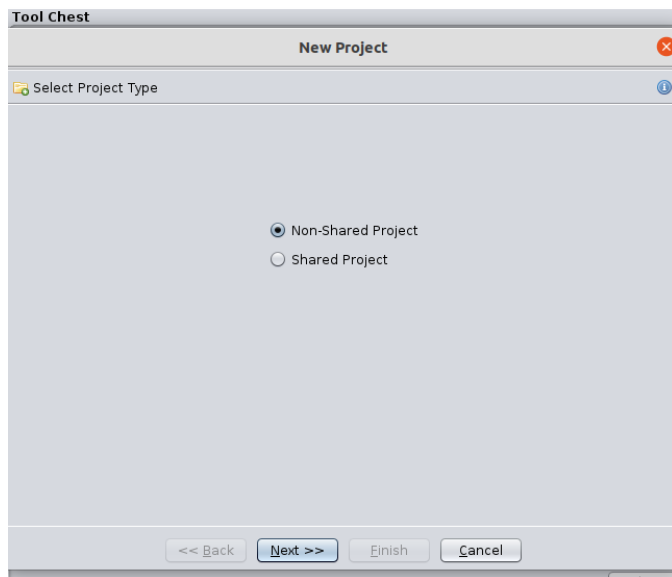
Step:1 Open the Ghidra tool

->Navigate to the Ghidra directory.

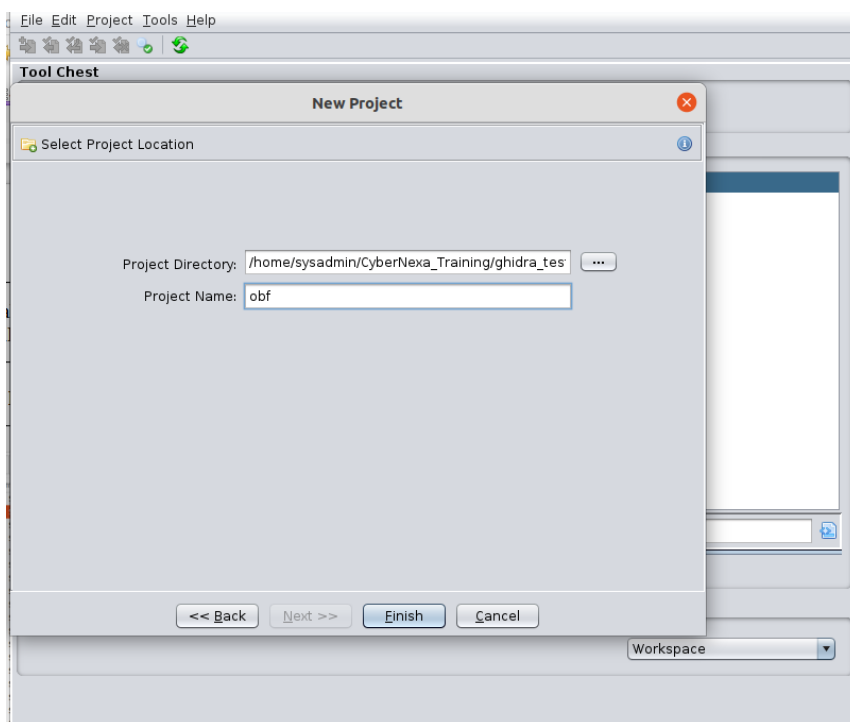
->Run the following command
`./ghidraRun`

Step:2 Create a New Project

i) select project type

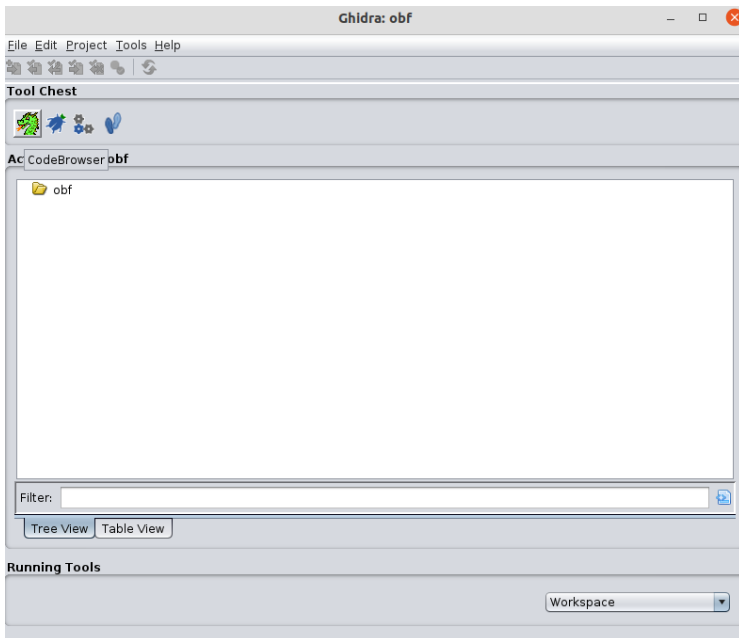


ii) Project name



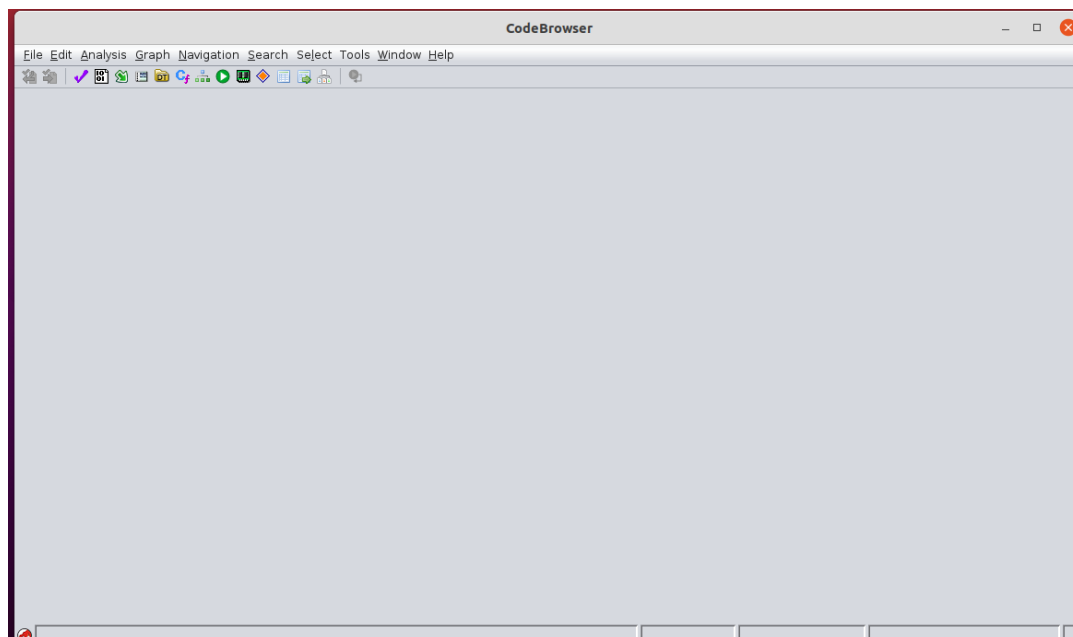
--> give a name for your project

iii) open code browser



---> click the code browser button (dragon symbol).
few seconds after new window will be open.

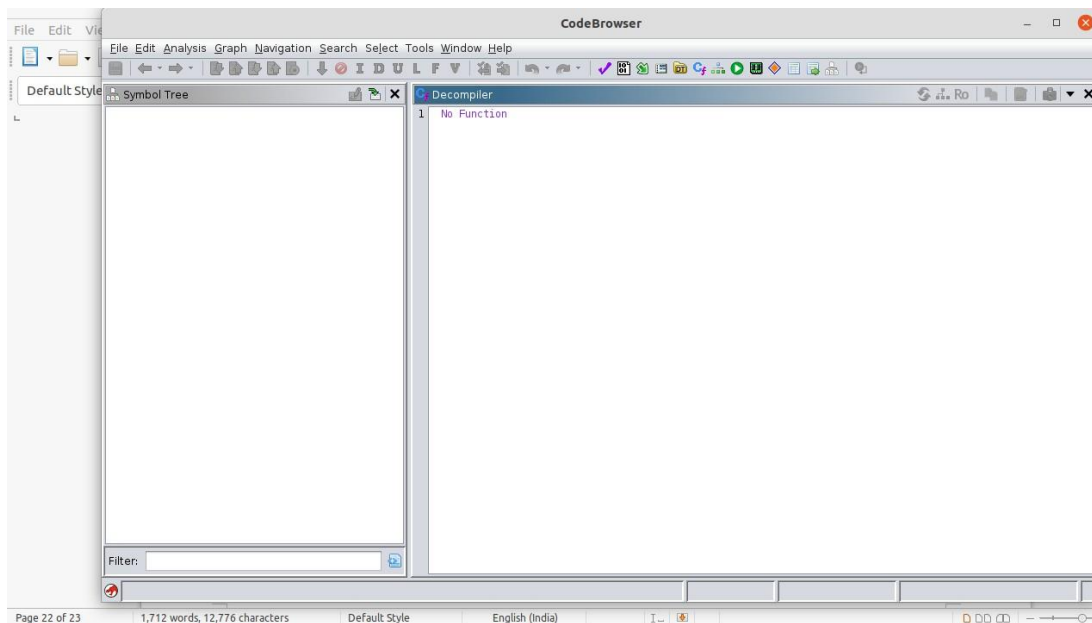
Code browser windows



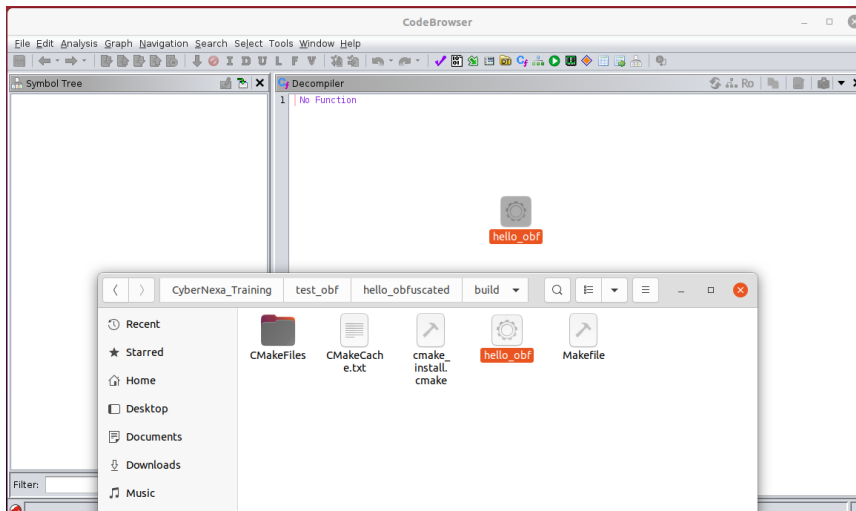


--->click the display symbol tree button
--->click the Display decompiler button

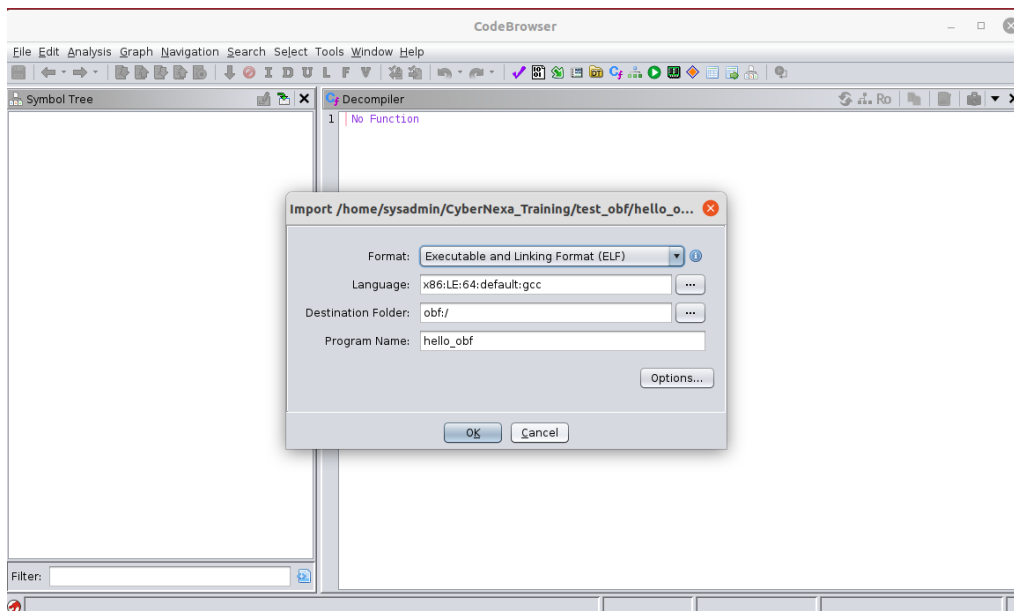
*** After click the above buttons you see the window like below**



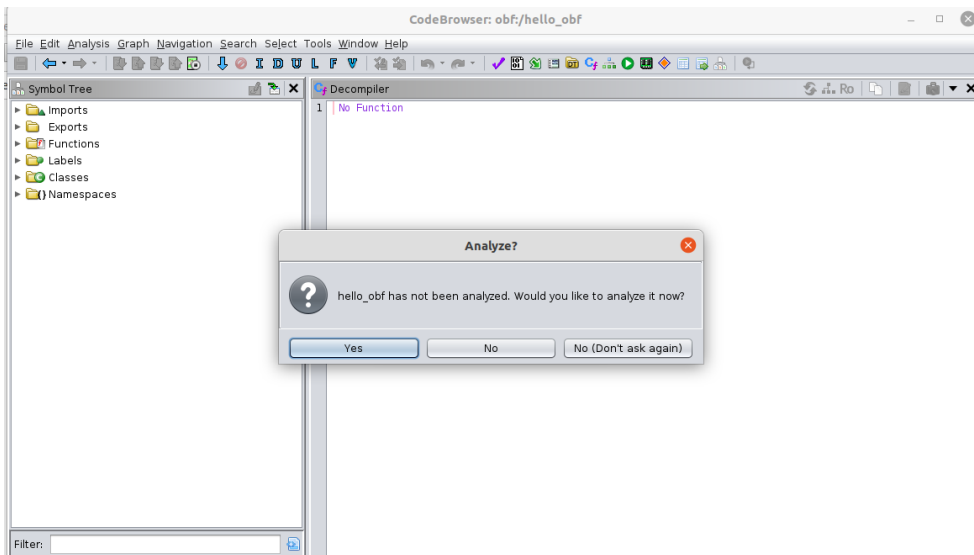
*** drag and drop your binary from your files**



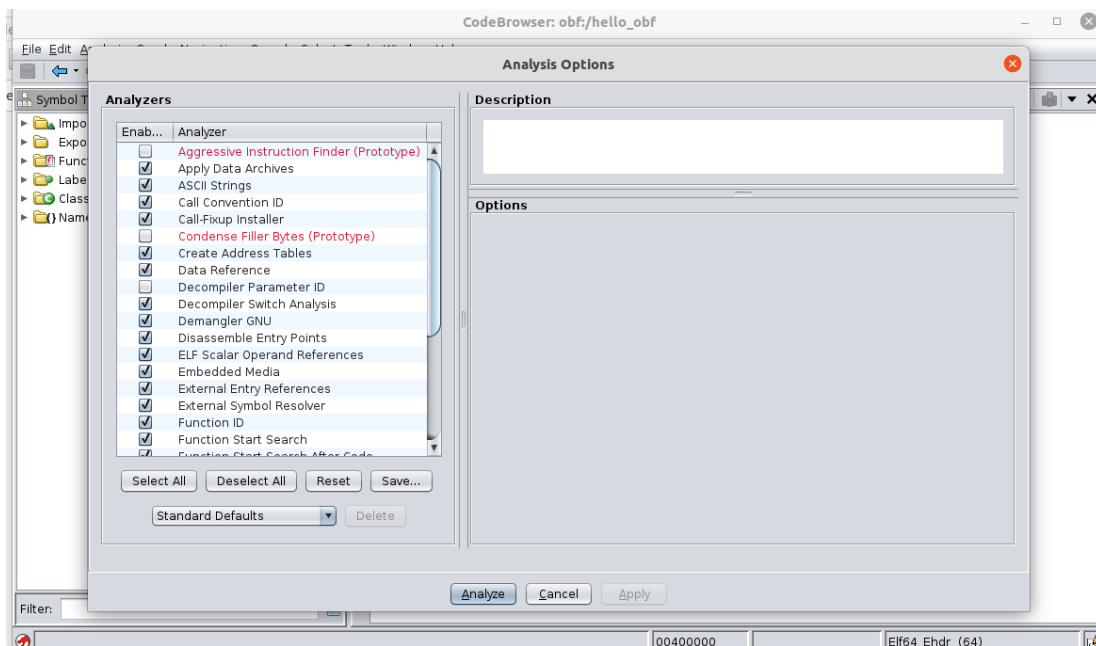
--> After drag and drop your binary file its shows like below



-->click **ok button**



-->click **Yes** button.



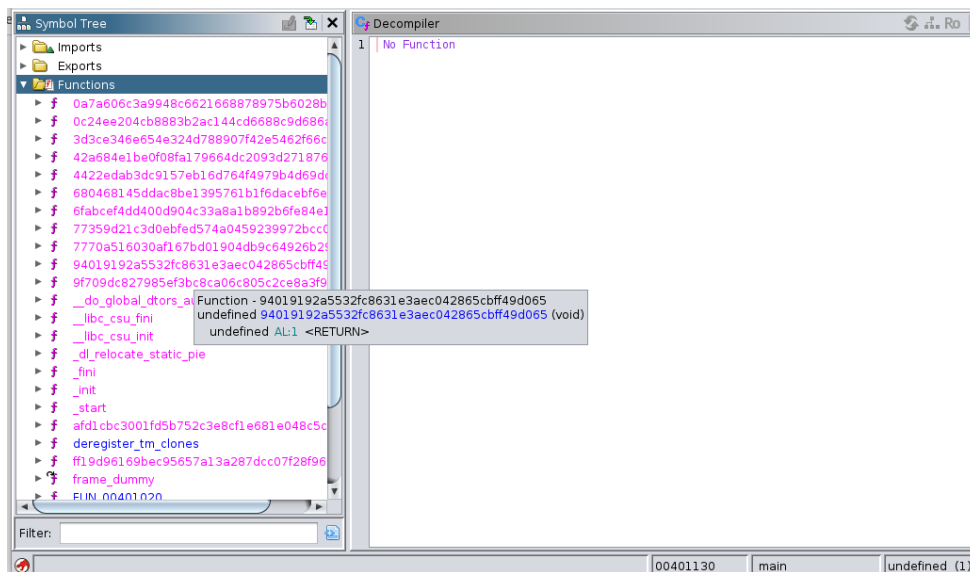
-->click **Analyze** button

Author: L.Radhakrishnan
Reviewed By: KMV.Subramanyam

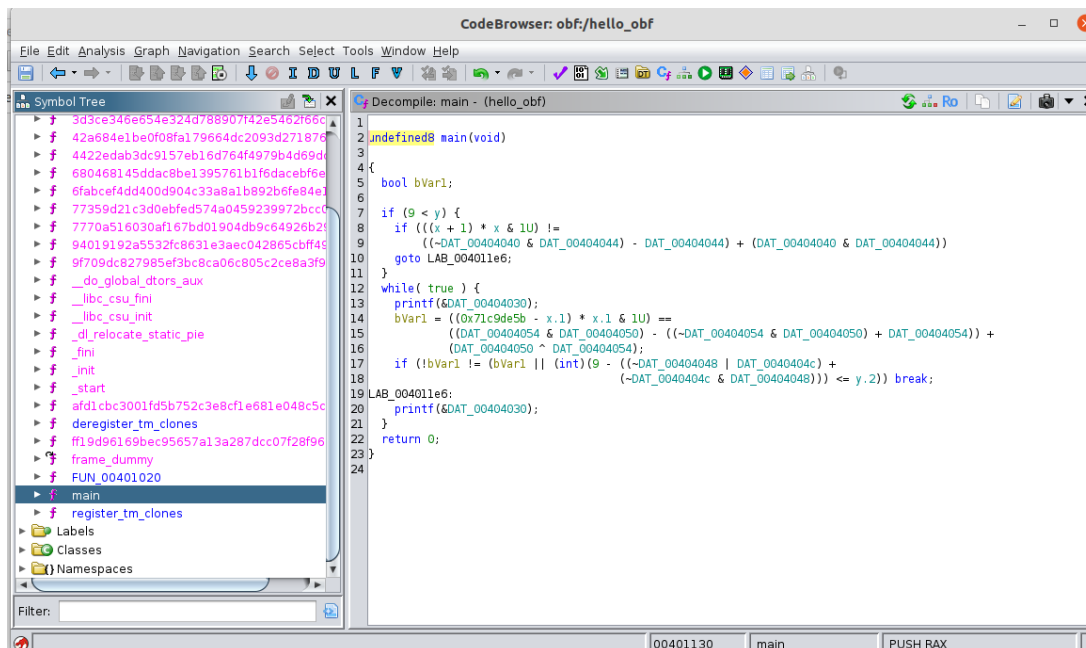
-->after analyze click a **ok** button

Test the code is obfuscate or not

-->we just write a hello world code ,so start with main function



--> code browser click a function button,it will show the functions list

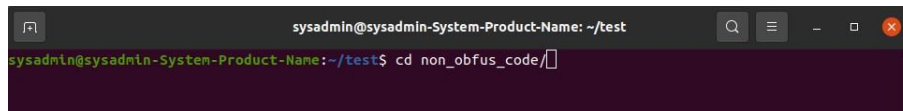


--> we write main function only, so click the main function ,it will show the code.

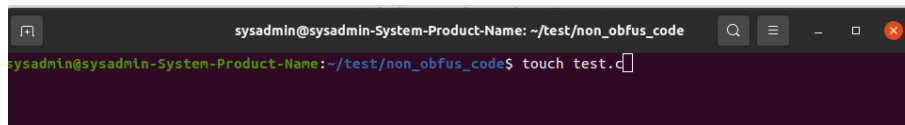
-->its not show any strings and original code.

Let's test the non obfuscated code

Step:1 create a c file

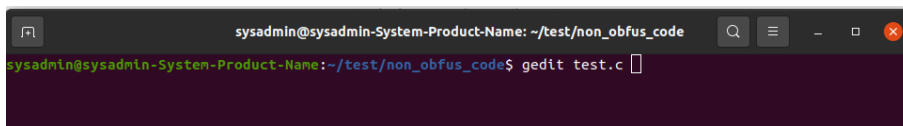


```
sysadmin@sysadmin-System-Product-Name: ~/test
sysadmin@sysadmin-System-Product-Name:~/test$ cd non_obfus_code/
```



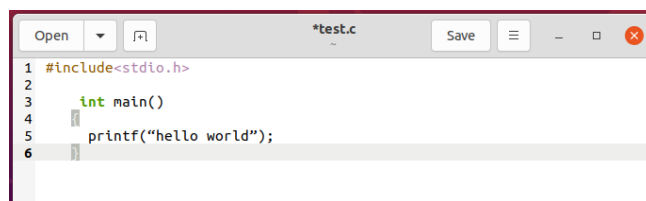
```
sysadmin@sysadmin-System-Product-Name: ~/test/non_obfus_code
sysadmin@sysadmin-System-Product-Name:~/test/non_obfus_code$ touch test.c
```

Step:2 Open the file



```
sysadmin@sysadmin-System-Product-Name: ~/test/non_obfus_code
sysadmin@sysadmin-System-Product-Name:~/test/non_obfus_code$ gedit test.c
```

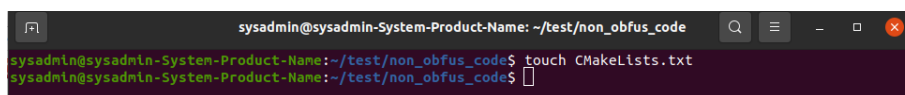
--> paste the below code and save it.



```
*test.c
1 #include<stdio.h>
2
3 int main()
4 {
5     printf("hello world");
6 }
```

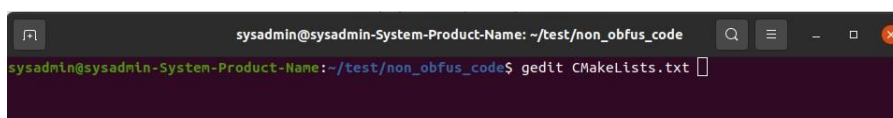
```
#include<stdio.h>
int main()
{
    printf("hello world");
}
```

Step:3 Create a Cmakefile



```
sysadmin@sysadmin-System-Product-Name: ~/test/non_obfus_code
sysadmin@sysadmin-System-Product-Name:~/test/non_obfus_code$ touch CMakeLists.txt
sysadmin@sysadmin-System-Product-Name:~/test/non_obfus_code$
```

Step:4 Open the File and write a code



```
sysadmin@sysadmin-System-Product-Name: ~/test/non_obfus_code
sysadmin@sysadmin-System-Product-Name:~/test/non_obfus_code$ gedit CMakeLists.txt
```

A screenshot of a text editor window titled '*CMakeLists.txt' with the path '~/.CyberNexa_Training/test_obf/hello_non_obfuscation'. The editor contains the following CMake code:

```
1 cmake_minimum_required(VERSION 3.10)
2
3 project(HelloObfuscated C CXX)
4
5 # Use Clang as the compiler
6 set(CMAKE_C_COMPILER "clang")
7 #set(CMAKE_CXX_COMPILER "clang++")
8
9 # Set compilation flags with LLVM obfuscation options
10 set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -O2")
11 #set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -O2")
12
13 # Add the executable
14 add_executable(hello hello.c)
15
```

The status bar at the bottom indicates 'CMake', 'Tab Width: 8', 'Ln 12, Col 1', and 'INS'.

```
cmake_minimum_required(VERSION 3.10)
```

```
project(HelloObfuscated C CXX)
```

```
# Use Clang as the compiler
```

```
set(CMAKE_C_COMPILER "clang")
```

```
#set(CMAKE_CXX_COMPILER "clang++")
```

```
# Set compilation flags with LLVM obfuscation options
```

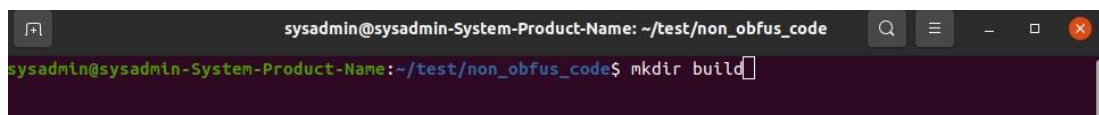
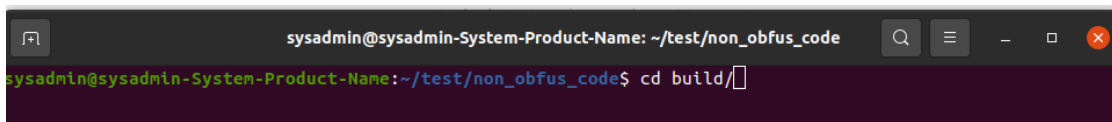
```
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -O2")
```

```
#set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -O2")
```

```
# Add the executable
```

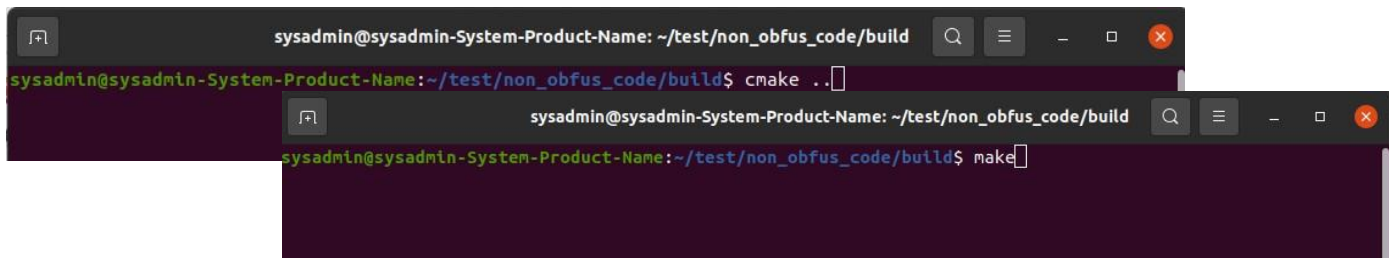
```
add_executable(hello hello.c)
```

Step:5 Create a Build directory

A terminal window screenshot showing the command 'mkdir build' being executed in the directory '~/.test/non_obfus_code'. The prompt is 'sysadmin@sysadmin-System-Product-Name: ~/.test/non_obfus_code\$'.A terminal window screenshot showing the command 'cd build/' being executed. The prompt is 'sysadmin@sysadmin-System-Product-Name: ~/.test/non_obfus_code\$'.

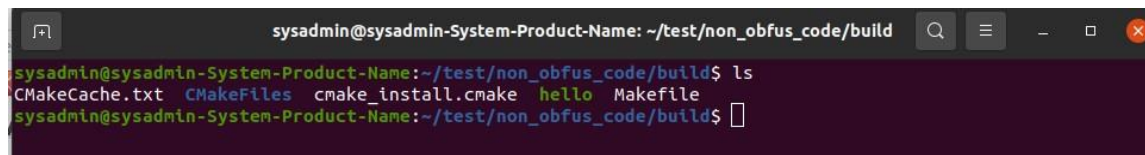
```
mkdir build
cd build
```

Step:6 Compile the code



```
sysadmin@sysadmin-System-Product-Name: ~/test/non_obfus_code/build
sysadmin@sysadmin-System-Product-Name:~/test/non_obfus_code/build$ cmake ..
sysadmin@sysadmin-System-Product-Name: ~/test/non_obfus_code/build
sysadmin@sysadmin-System-Product-Name:~/test/non_obfus_code/build$ make
```

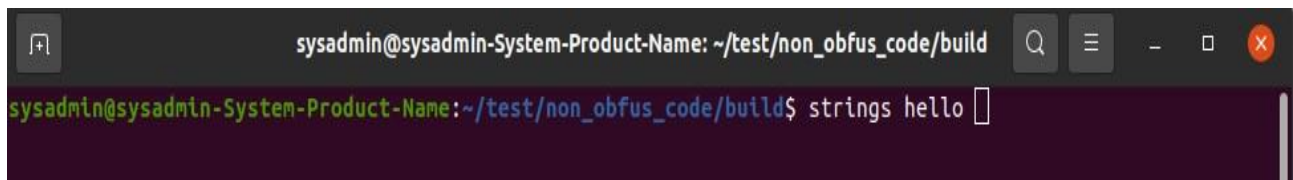
--> this will create a binary file hello



```
sysadmin@sysadmin-System-Product-Name: ~/test/non_obfus_code/build
sysadmin@sysadmin-System-Product-Name:~/test/non_obfus_code/build$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  hello  Makefile
sysadmin@sysadmin-System-Product-Name:~/test/non_obfus_code/build$
```

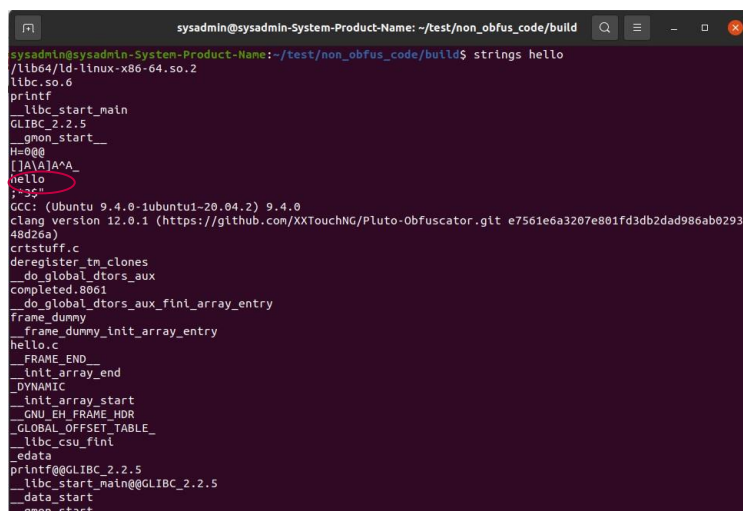
Step:7 Verify Obfuscation

Strings command:



```
sysadmin@sysadmin-System-Product-Name: ~/test/non_obfus_code/build
sysadmin@sysadmin-System-Product-Name:~/test/non_obfus_code/build$ strings hello
```

strings hello



```
sysadmin@sysadmin-System-Product-Name: ~/test/non_obfus_code/build
sysadmin@sysadmin-System-Product-Name:~/test/non_obfus_code/build$ strings hello
/lib64/ld-linux-x86-64.so.2
libc.so.6
printf
__libc_start_main
GLIBC_2.2.5
__gmon_start__
H=000
[]A[A]A^A_
hello
;=36
GCC: (Ubuntu 9.4.0-1ubuntu1-20.04.2) 9.4.0
clang version 12.0.1 (https://github.com/XXTouchNG/Pluto-Obfuscator.git e7561e6a3207e801fd3db2dad986ab029348d26a)
crtstuff.c
deregister_tm_clones
__do_global_dtors_aux
completed.8061
__do_global_dtors_aux_fini_array_entry
frame_dummy
frame_dummy_init_array_entry
hello.c
__FRAME_END__
__init_array_end
DYNAMIC
__init_array_start
GNU_EH_FRAME_HDR
_GLOBAL_OFFSET_TABLE_
__libc_csu_fini
edata
printf@GLIBC_2.2.5
__libc_start_main@GLIBC_2.2.5
__data_start
__gmon_start__
```

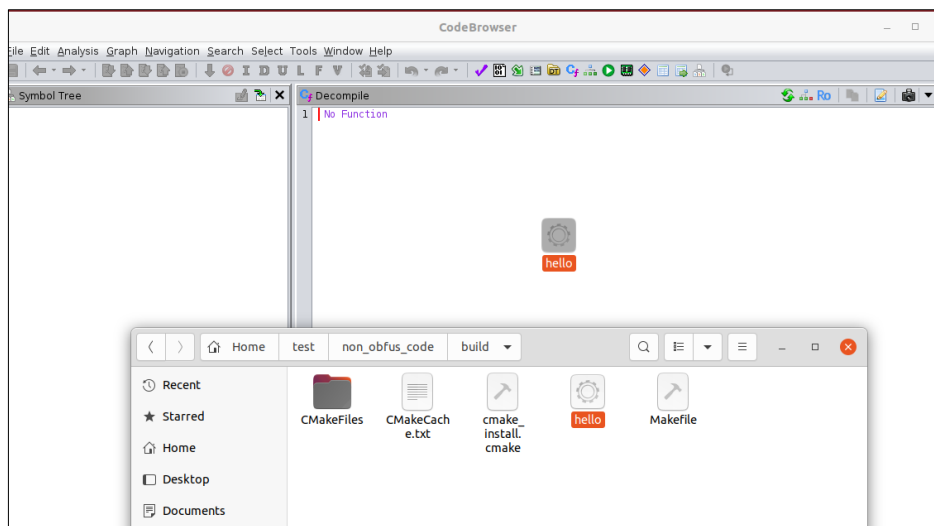
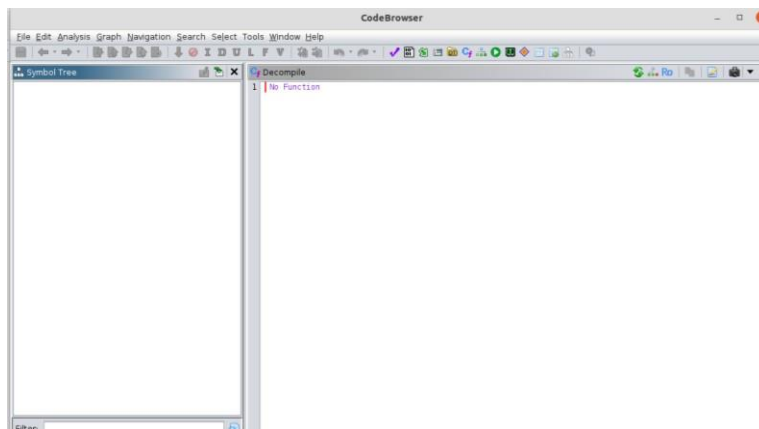

Author: L.Radhakrishnan
Reviewed By: KMV.Subramanyam

*we can see the hello string, so its not obfuscated.

Ghidra Tool

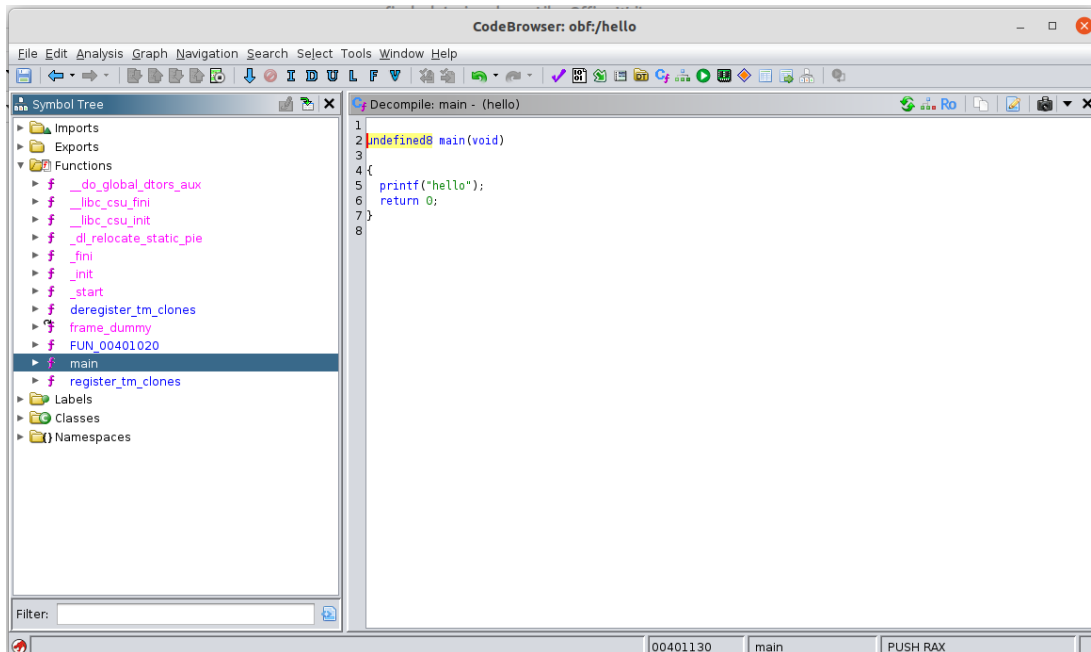
---> you know how do use the ghidra tool by previous guide.

Let's check with ghidra



Author: L.Radhakrishnan
Reviewed By: KMV.Subramanyam

-->drag and drop your non obfuscated binary file



-->we can see the original code,so it's not obfuscated.