CSIE 5374 Assignment 4 (Due On 06/04 23:59)

In assignment 4, you are asked to add a new pass In LLVM based on LLVM 16.0.0. You should implement a V-Table pointer checker.

Assignment 4 is a group task. You should collaborate with your group members on this assignment.

Development Tips

Build LLVM toolchain

1. Get LLVM source code.

```
git clone -b llvmorg-16.0.0 --depth 1 https://github.com/llvm/llvm-project.git
```

2. Config and build it.

```
cmake -S llvm -B build -G Ninja \
-DLLVM_ENABLE_PROJECTS="clang" \
-DCMAKE_BUILD_TYPE=Release \
-DLLVM_PARALLEL_LINK_JOBS=1 \
-DLLVM_BUILD_TOOLS=on

cmake --build build
```

Build an LLVM pass plugin

You can build your pass as the LLVM pass plugin with the following commands:

```
cd hw4
mkdir build
cd build
export LLVM_DIR=/path/to/your/llvm-project/build/lib/cmake/llvm
cmake ..
make
```

After compilation, you can find the `passes` directory in the `build` and your plugin is compiled as `hw4.so`.

Compile programs with your runtime library and run your LLVM pass plugin

After compilation, you can run `hw4.so` with the following commands. Notably, you will implement the security check in a runtime library (`rtlib.*`). Therefore, you should compile test programs with your plugin and runtime library. You can compile your runtime library as a `.o` or `.so` file.

```
/path/to/your/clang \
    -fpass-plugin=`echo build/passes/hw4.so`\
    something.c rtlib.o
```

Requirement

Implementation (70%)

One of the most popular targets for attack is virtual table pointers, which point to virtual function tables (V-table) consisting of virtual function pointers.

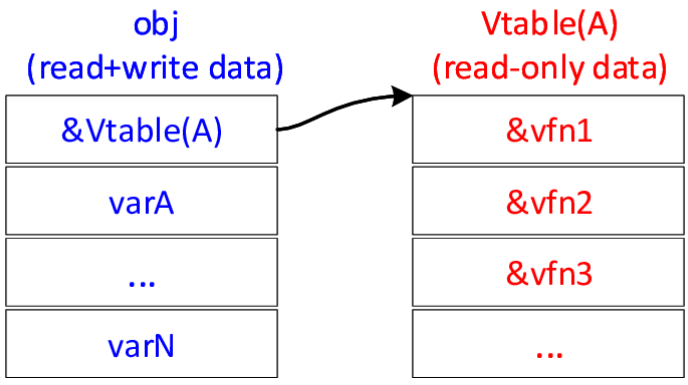
Exploiting vulnerabilities like use-after-free and heap overflow, attackers can tamper with the V-table or its pointer, enabling them to hijack subsequent virtual function calls (known as V-table hijacking).

In legitimate C++ programs, class objects' virtual tables are never changed, and thus any virtual table that is writable should never be used.

When a virtual function is called through a pointer or reference to a base class object, the compiler generates code to perform a virtual function dispatch. This involves looking up the correct function pointer in the object's V-table based on the actual type of the object at runtime. Once the correct function pointer is found, the corresponding function is invoked.

The following figure illustrates the concept of the virtual function dispatch. `Callsite` represents any virtual function callsite of the class `A` and the following instructions present how the virtual function dispatch works. Moreover, the structure in the left-bottom side is the layout of the class `A`, and the table in the right-bottom side is the V-table of `A`.

```
//Read-only Code
Callsite (A *obj):
  vptr = *(obj)
  vfnptr = *(vptr + offset)
  call vfnptr //Indirect call
```



In this assignment, you are asked to implement a function that performs the security check for V-table in a runtime library. In your program, before all virtual function dispatch, you should instrument a function by your pass to perform a security check to ensure that the V-table resides in a read-only memory region. More specifically, you should instrument the function immediately after the instruction that loads a V-table pointer to a register. On the other side, the function should find the address of the V-table and then validate if the V-table is located in a read-only memory region. If the validation fails, the program should be abort immediately.

You should provide a testing program. In this program, you have to define two classes that have a virtual function. After that, you should initialize a class object and then overwrite the V-table pointer of it with a writable memory region.

Write-up (30%)

Each of the groups is required to provide a write-up about the assignment. The write-up should include detailed explanation of your source code (e.g., how it works), a description of how you test the plugin and the runtime library, and contributions from each of you.

Homework submission

You should submit the assignment via NTU Cool.

Submission Format

You are required to submit source codes, a Makefile, and a write-up file. Compress all the files in hw4.zip, and upload the zip file to NTU Cool.

Your Makefile should compile the runtime library and the testing program.

```
.
├── testing program
├── Makefile
├── write-up.md
├── CMakeLists.txt
├── rtlib.c
├── passes
     ├── CMakeLists.txt
|    └── hw4.cpp
├── any extra file
```

Grading criteria

You will get zero points if your code fails to compile.

You are required to properly manage resources (free allocated memory), handle errors, and return error codes.

Plagiarism policy

You are allowed to reference sources from the internet. If you do, please attach all of your references in the write-up. If we find that your code is similar to other's from the internet, we will count it as plagiarism if we could not find the corresponding references. You will automatically get zero points for the assignment.

Late Policy

We do not accept late submissions for this assignment. Please start the assignment early.