Assignment 1– LLVM Playground (Part 1)

HSS
Fall 2025

The goal of this assignment is to get your hands dirty using LLVM and gain experience developing static analysis and dynamic instrumentation techniques that can work on real-world code. In this part, you goal is to understand different LLVM IR instructions and how they are produced from C code.
Logistics:

- **LLVM Primer:** Please make sure that you have skimmed the LLVM Primer presentation (access it from the course webpage) to know the capabilities of LLVM. Also refer to `https://mapping-high-level-constructs-to-llvm-ir.readthedocs.io/en/latest/index.html` to know the mapping from C to LLVM IR.
- **Setup Repo:** I have created a `github` repo with all the necessary scripts to install LLVM, Z3 and starter code to write a pass. You can access it at: `https://github.com/HolisticSoftwareSecurity/hssllvmsetup`. The repo has examples of analysis (i.e., the passes that do not modify the IR) and instrumentation (i.e., the passes that modify the IR) passes.
- **Development Environment:** I use CLion (`https://www.jetbrains.com/clion/`) while working with LLVM and strongly suggest you to use it. You can get unlimited access using your `@purdue.edu` email.

1 Generating Bitcode file

You can generate a bitcode file for a given C file by following the below instructions:

```
clang -c -emit-llvm <your_c_file> -o <path_to_output_bitcode>
```

Example:

```
clang -c -emit-llvm simple_log.c -o simple_log.bc
```

The file `simple_log.bc` will be in binary format. You can get human readable bitcode file from the binary format using the following command:

```
llvm-dis <path_to_bitcode_file>
```

Example:

```
llvm-dis simple_log.bc
```

The above command will generate `simple_log.ll` which is a text file containing human-readable LLVM IR.

You can generate bitcode for your entire project (i.e., multiple C files) using `wllvm` [1].

Part 1 - Understanding the LLVM IR

Repo

`https://github.com/HolisticSoftwareSecurity/LLVMPlayground`

Step 1

Study the LLVM Primer from the course webpage to understand the structure of the LLVM IR. The primer shows how to run LLVM on a sample C program to generate the corresponding LLVM IR program. You can use the `part1_learningir/` directory in the repo for this purpose:

```
cd test
clang -emit-llvm -S -fno-discard-value-names -c simple0.c
```

Step 2

Write by hand the C programs corresponding to the LLVM IR programs under the `part1_learningir/ir_programs` directory and place them under the `part1_learningir/c_programs/` directory. Ensure that running the above command on your hand-written C programs generates the exact LLVM IR programs provided as we will auto-grade them. You can do so by using the diff command-line utility to check if your files are the same. As shown in the following example:

```
cd part1_learningir/c_programs
clang -emit-llvm -S -fno-discard-value-names -c test1.c
diff test1.ll ../ir_programs/test1.ll
```

You can ignore some differences in clang attributes. Just make sure that the code is same.

2 Submission

Run the below command to produce file `submission.zip` and submit that file to brightspace.

```
make submit
mkdir -p submission
rm -rf submission/*
cp -r c_programs submission
zip -r submission.zip submission/ 2 >&1>/dev/null
```

References

[1] https://github.com/travitch/whole-program-llvm.