LLVM Demystified

Building a Feedback Fuzzer

By
Fady Othman

Who Am I

- Co-Founder @ ZINAD IT.
- Security Analyst at Hackerone.
- OSCP, OSCE.
- Love fuzzing and exploit writing.

Agenda

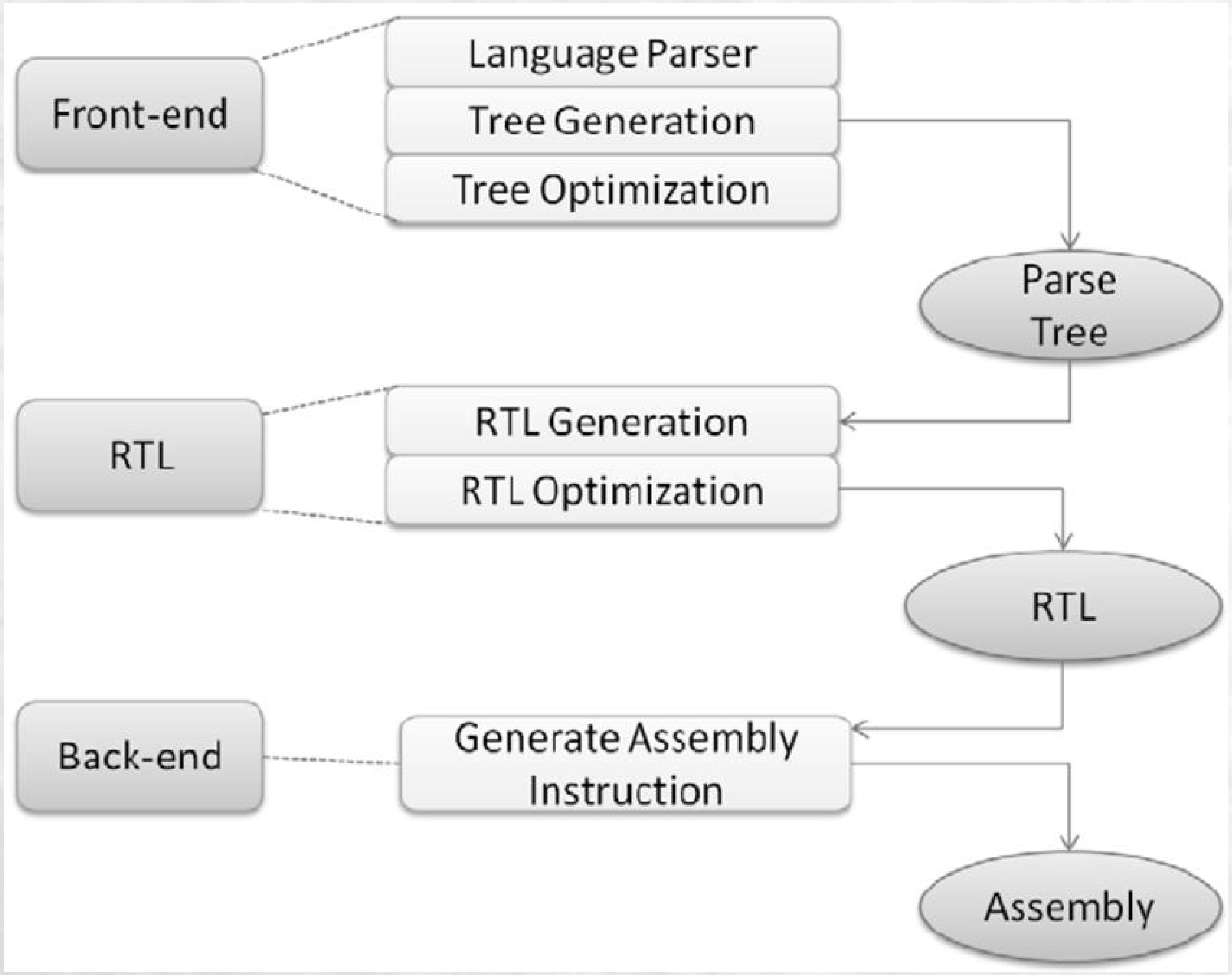
- What is LLVM？
- Why should I care？
- Is it hard to learn？
- LLVM Passes.
- Building the taint analysis pass.

What is LLVM？

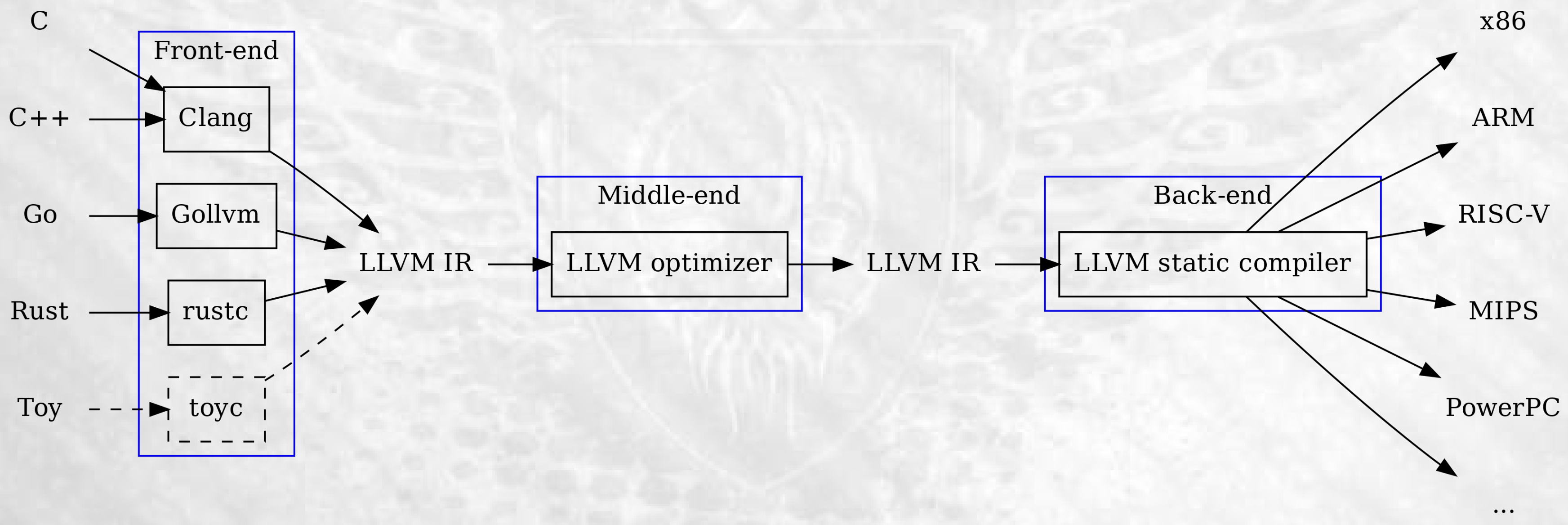
What is LLVM？

- A set of compiler and toolchain technologies, which can be used to develop a front end for any programming language and a back end for any instruction set architecture.

Classic Compiler Archietecture



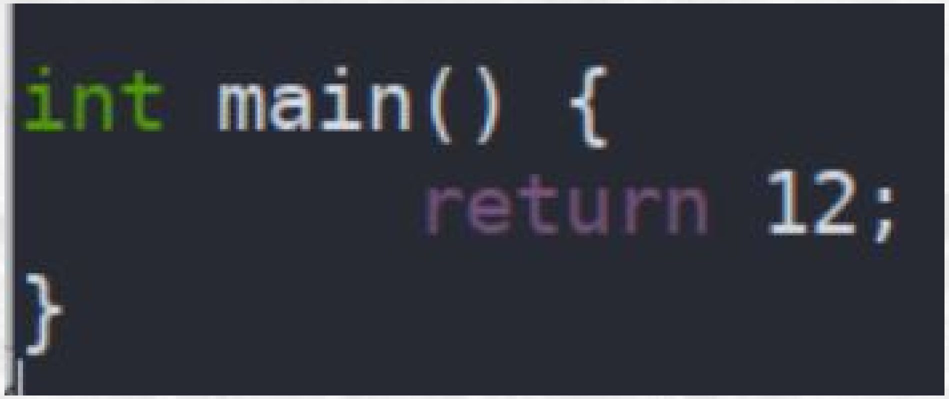
LLVM Archietecture



LLVM IR

- **LLVM uses IR to represent the code.**
- **Optmization passes can run on IR.**
- **You can write your own passes.**

IR Example



```
int main() {
        return 12;
}
```

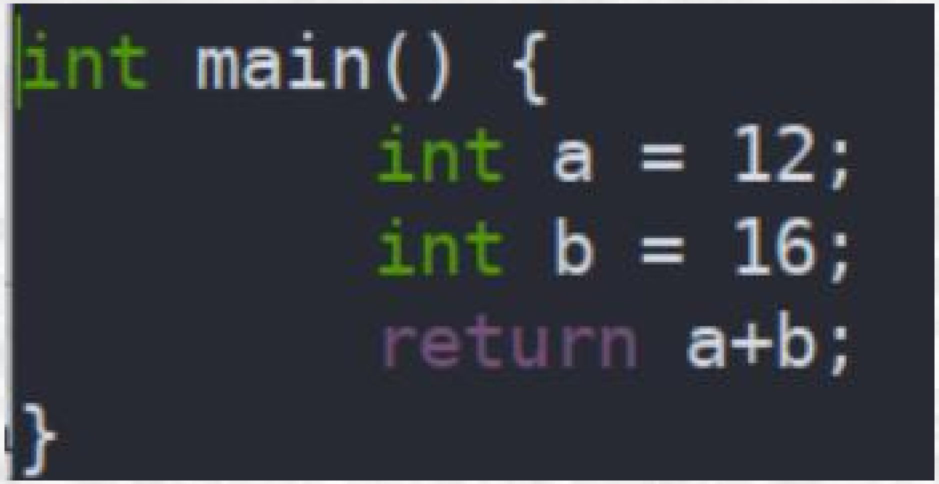
Compile With:

clang -S -emit-llvm -O3 hello.c

IR Example : IR Code

```
define @main() {
  ret i32 12
}
```

IR Second Example

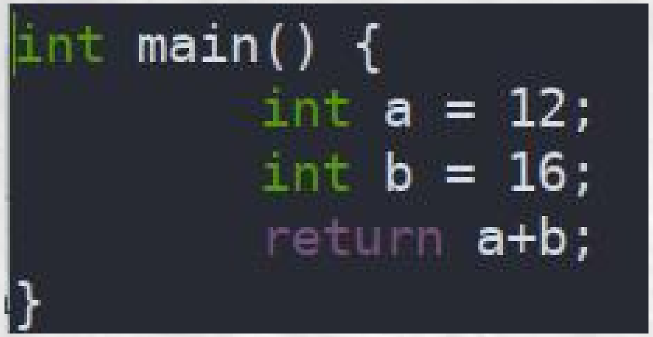


```
int main() {
        int a = 12;
        int b = 16;
        return a+b;
}
```

Compile With:

```
clang -S -emit-llvm -O0 hello2.c
```

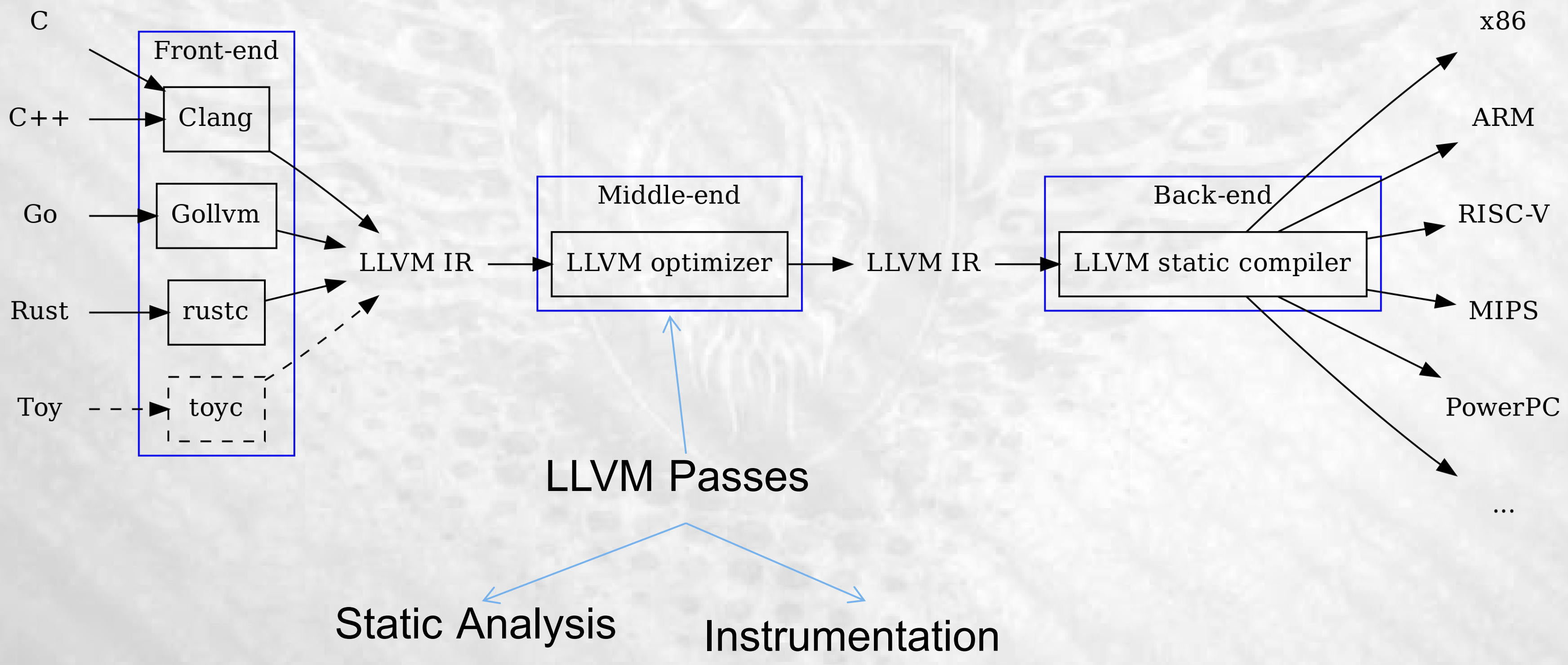
IR Second Example : IR Code



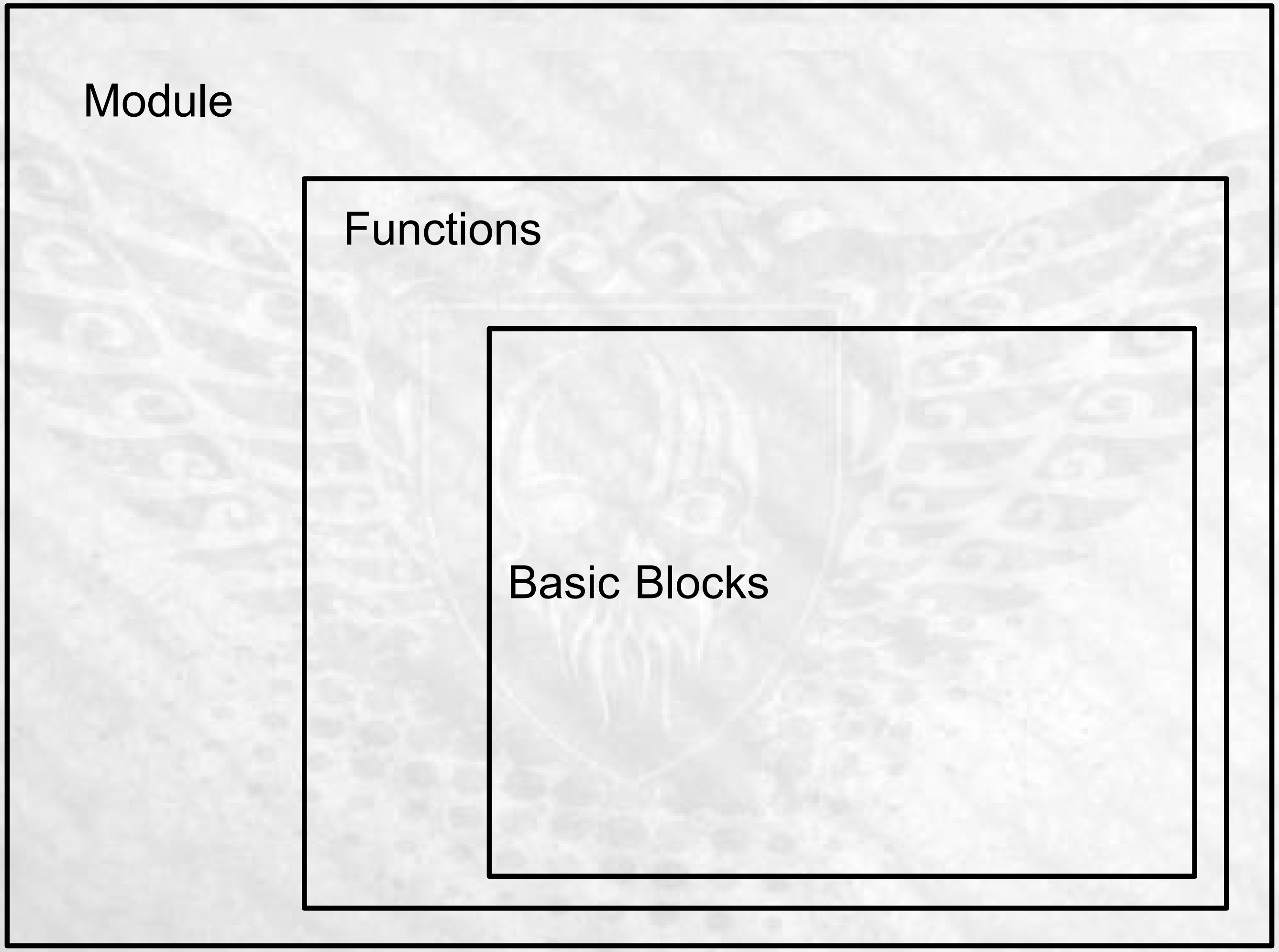
```c
int main() {
        int a = 12;
        int b = 16;
        return a+b;
}
```

```
define i32 @main() {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4

  store i32 0, i32* %1, align 4
  store i32 12, i32* %2, align 4
  store i32 16, i32* %3, align 4

  %4 = load i32, i32* %2, align 4
  %5 = load i32, i32* %3, align 4
  %6 = add nsw i32 %4, %5
  ret i32 %6
}
```

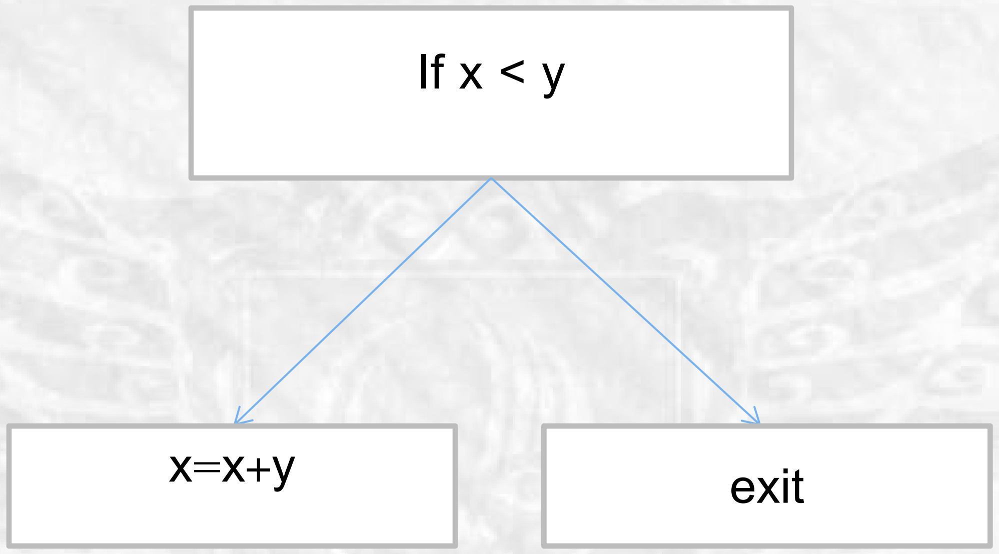
Why Should I Care？



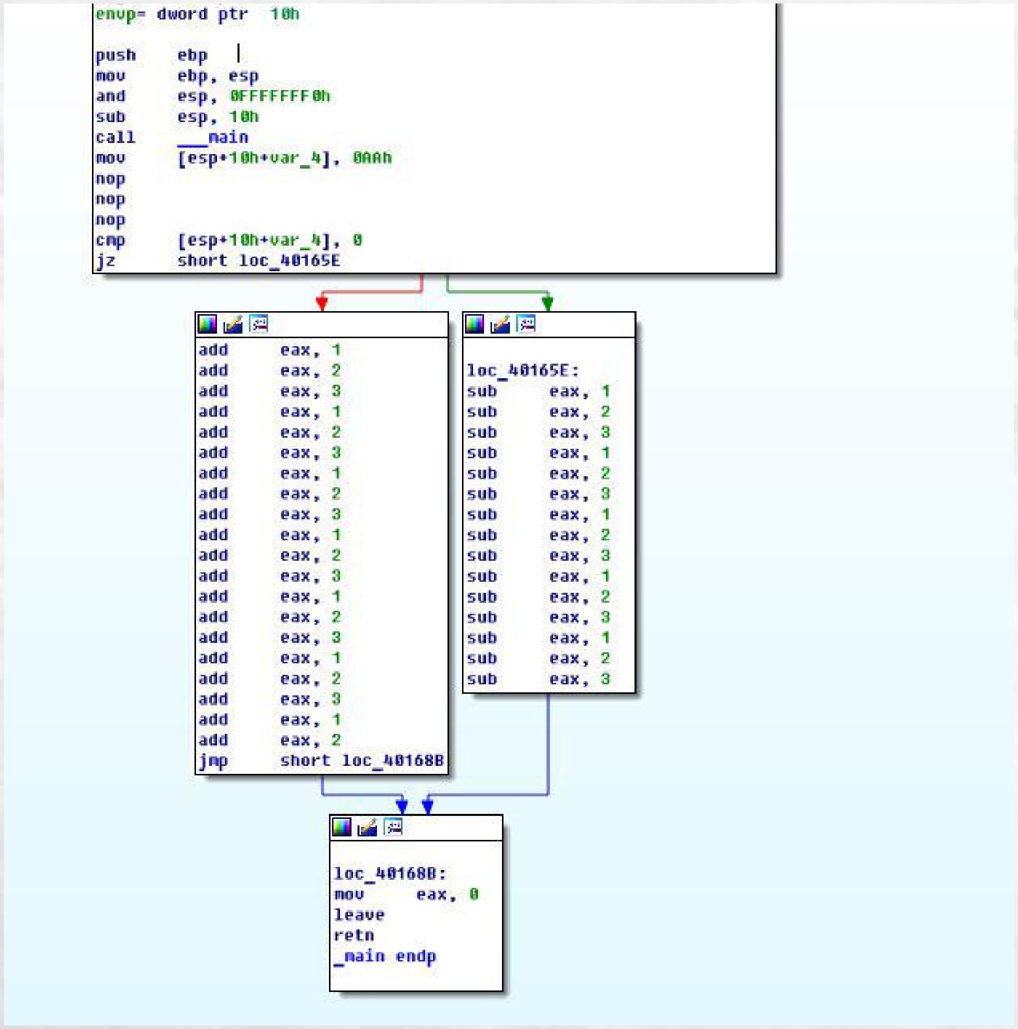
How Passes See the Code



Basic Blocks



Familiar Right



```
envp= dword ptr  10h

push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF0h
sub     esp, 10h
call    ___main
mov     [esp+10h+var_4], 0AAh
nop
nop
nop
cmp     [esp+10h+var_4], 0
jz      short loc_40165E
```

```
add     eax, 1
add     eax, 2
add     eax, 3
add     eax, 1
add     eax, 2
add     eax, 3
add     eax, 1
add     eax, 2
add     eax, 3
add     eax, 1
add     eax, 2
add     eax, 3
add     eax, 1
add     eax, 2
add     eax, 3
add     eax, 1
add     eax, 2
add     eax, 3
add     eax, 1
add     eax, 2
jmp     short loc_40168B
```

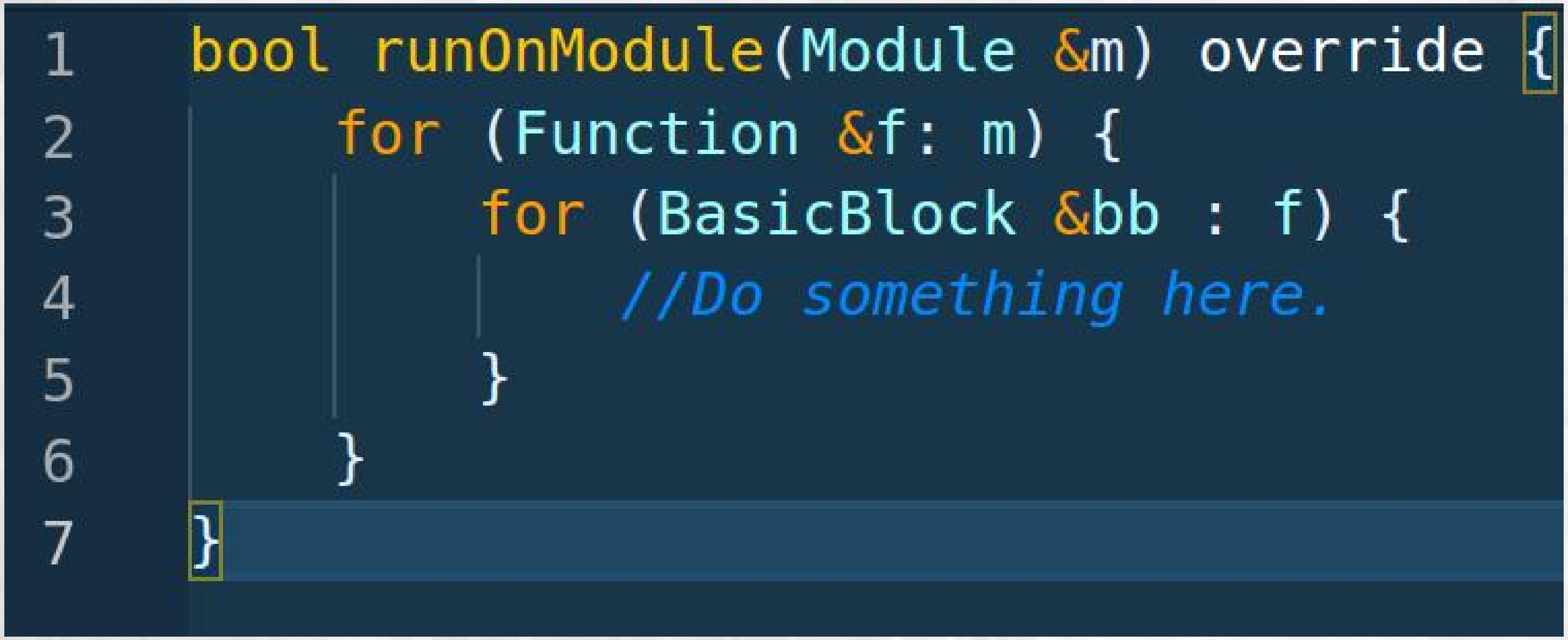
```
loc_40165E:
sub     eax, 1
sub     eax, 2
sub     eax, 3
sub     eax, 1
sub     eax, 2
sub     eax, 3
sub     eax, 1
sub     eax, 2
sub     eax, 3
sub     eax, 1
sub     eax, 2
sub     eax, 3
sub     eax, 1
sub     eax, 2
sub     eax, 3
```

```
loc_40168B:
mov     eax, 0
leave
retn
_main endp
```

LLVM Passes

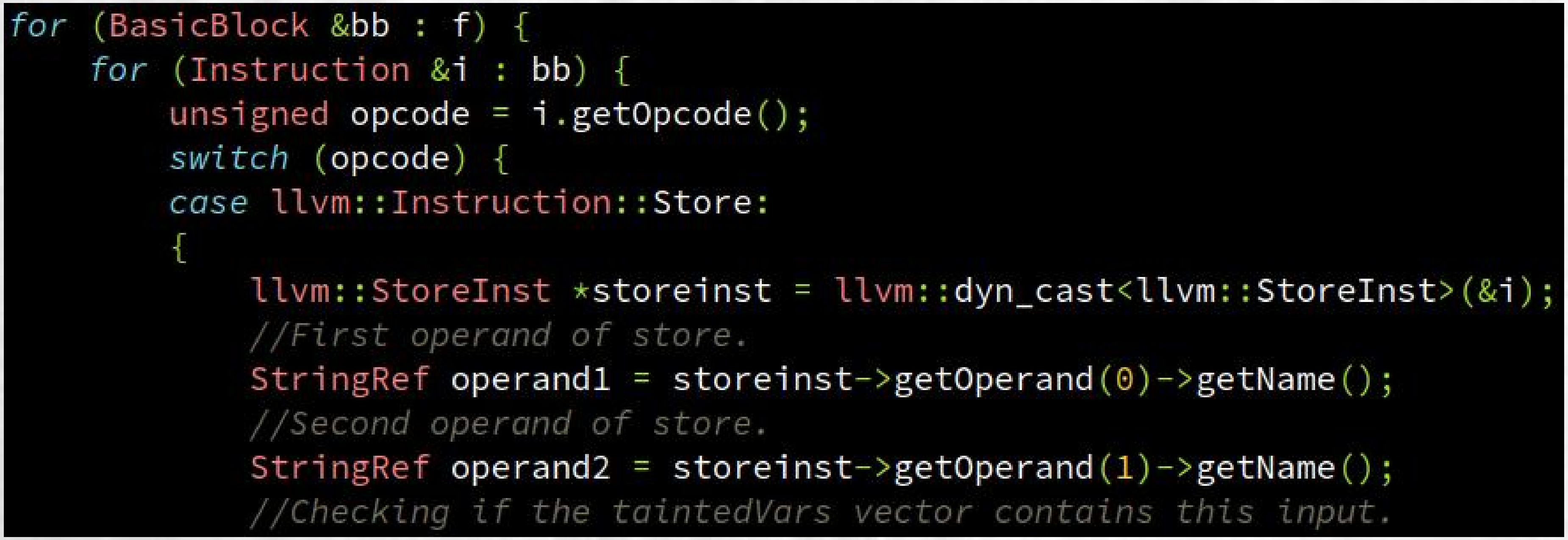
- **Module Pass.**
- **Function Pass.**
- **Basic Block Pass.**

Pass Skeleton



```
bool runOnModule(Module &m) override {
    for (Function &f: m) {
        for (BasicBlock &bb : f) {
            //Do something here.
        }
    }
}
```

Parsing Instructions



```
for (BasicBlock &bb : f) {
    for (Instruction &i : bb) {
        unsigned opcode = i.getOpcode();
        switch (opcode) {
        case llvm::Instruction::Store:
        {
            llvm::StoreInst *storeinst = llvm::dyn_cast<llvm::StoreInst>(&i);
            //First operand of store.
            StringRef operand1 = storeinst->getOperand(0)->getName();
            //Second operand of store.
            StringRef operand2 = storeinst->getOperand(1)->getName();
            //Checking if the taintedVars vector contains this input.
```

Is it hard to learn？

RTFM

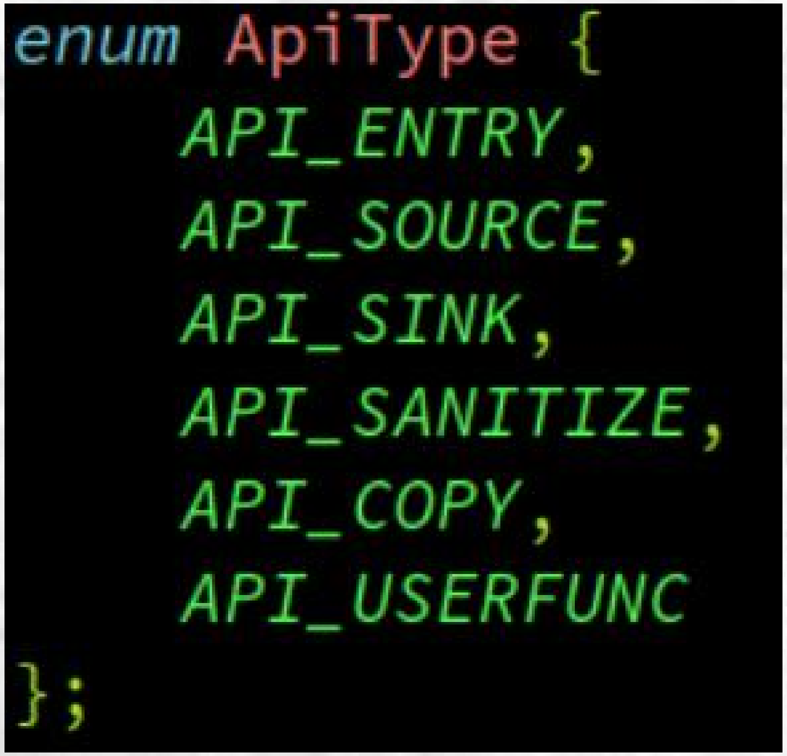
Vs

RTFC

Building the Taint Analysis Pass

How it works？

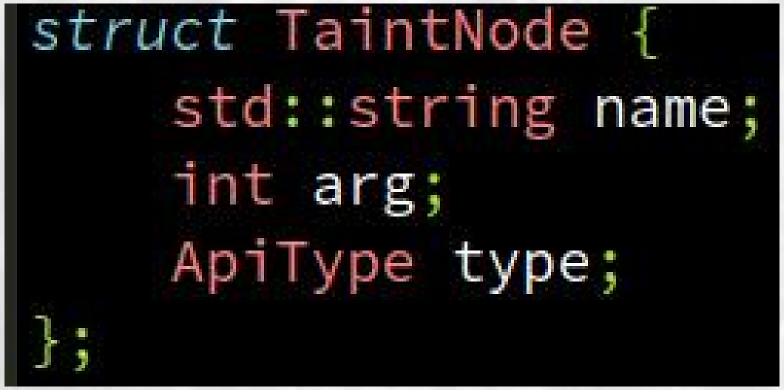
- **We start by classify functions by thier type**:



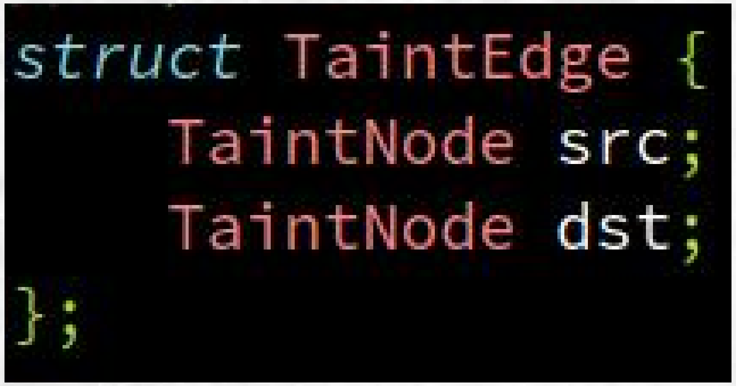
```
enum ApiType {
    API_ENTRY,
    API_SOURCE,
    API_SINK,
    API_SANITIZE,
    API_COPY,
    API_USERFUNC
};
```

How it Works

- **We loop through all the functions and mark the tainted ones as such**

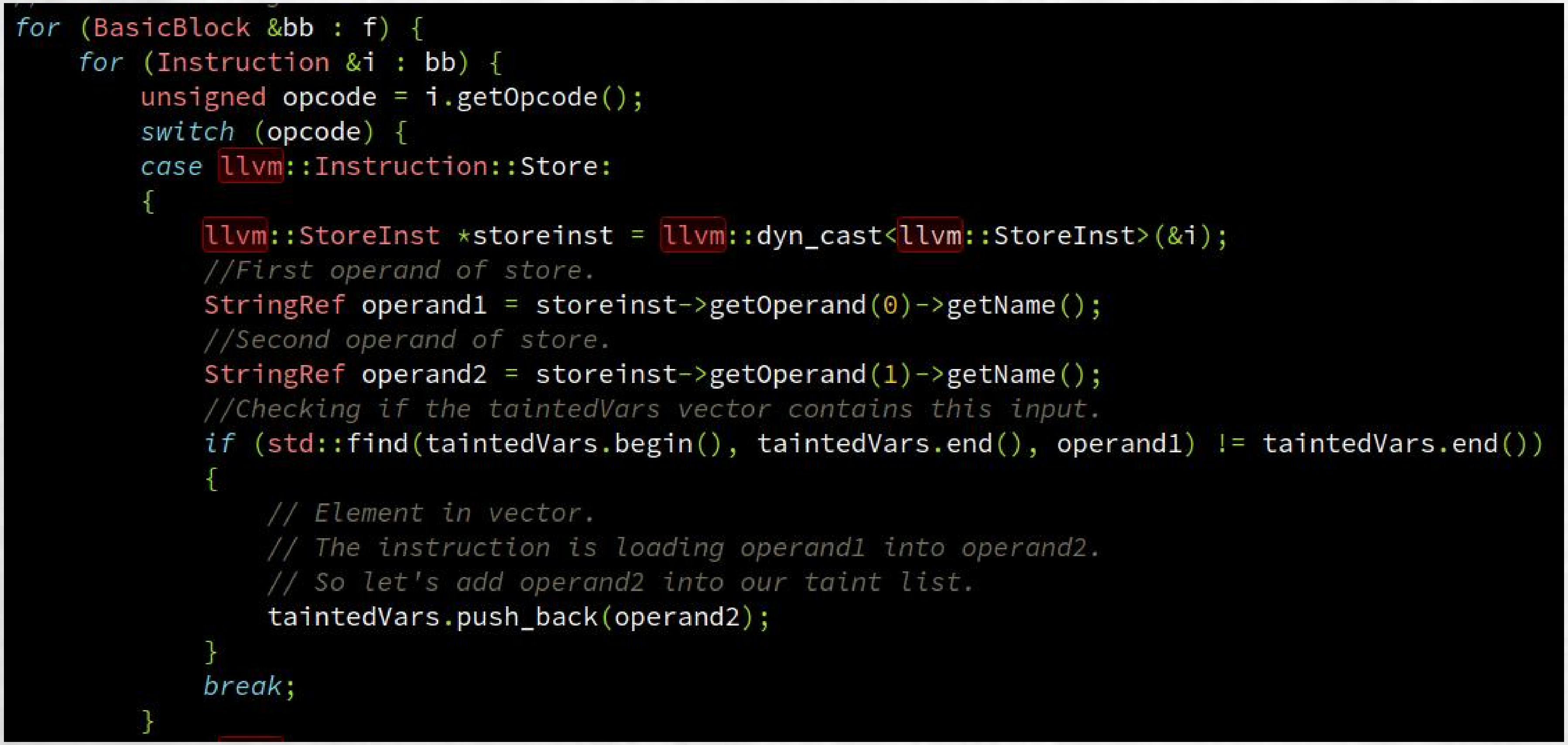


```
struct TaintNode {
    std::string name;
    int arg;
    ApiType type;
};
```



```
struct TaintEdge {
    TaintNode src;
    TaintNode dst;
};
```

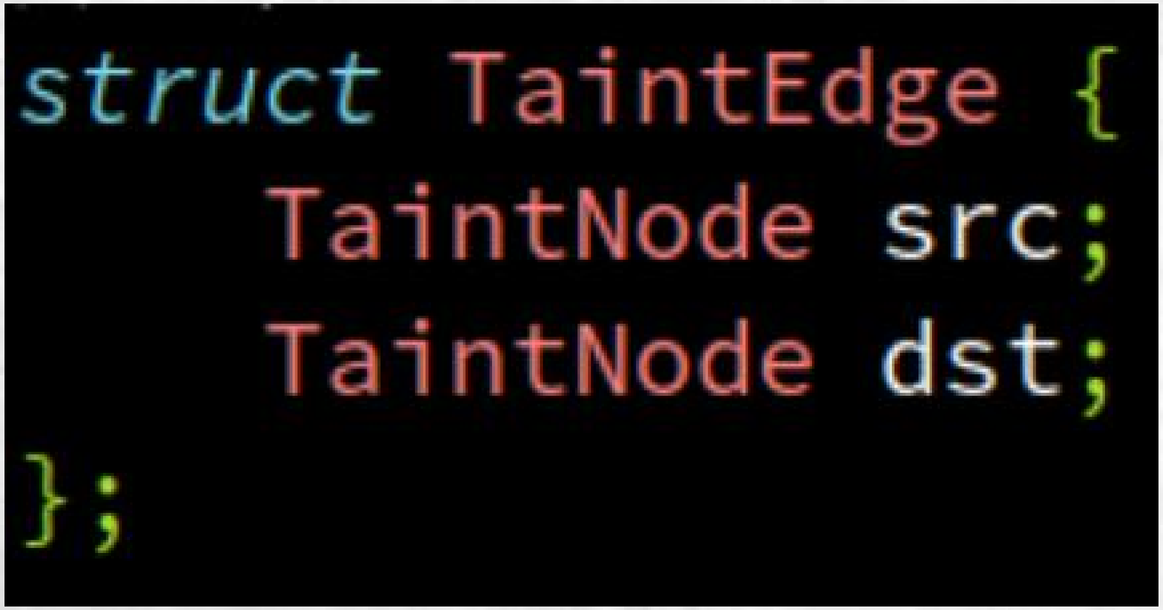
Taint Analysis



```
for (BasicBlock &bb : f) {
    for (Instruction &i : bb) {
        unsigned opcode = i.getOpcode();
        switch (opcode) {
        case llvm::Instruction::Store:
        {
            llvm::StoreInst *storeinst = llvm::dyn_cast<llvm::StoreInst>(&i);
            //First operand of store.
            StringRef operand1 = storeinst->getOperand(0)->getName();
            //Second operand of store.
            StringRef operand2 = storeinst->getOperand(1)->getName();
            //Checking if the taintedVars vector contains this input.
            if (std::find(taintedVars.begin(), taintedVars.end(), operand1) != taintedVars.end())
            {
                // Element in vector.
                // The instruction is loading operand1 into operand2.
                // So let's add operand2 into our taint list.
                taintedVars.push_back(operand2);
            }
            break;
        }
```

Printing the Stack Trace

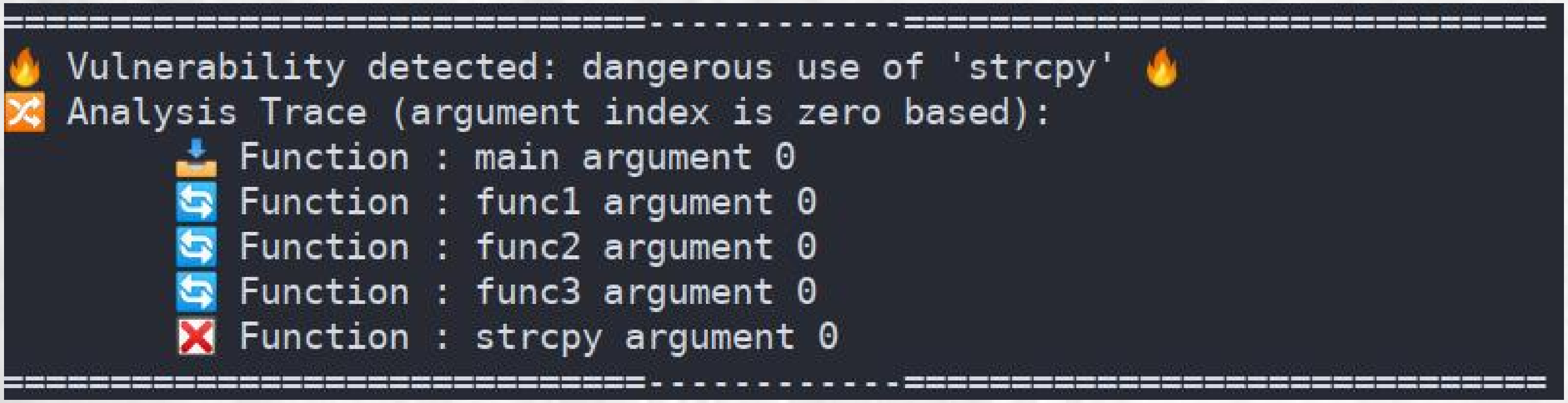
- **We keep track if the taints as edges.**
- **We follow the edges and see if a source is reaching a dangerous sink.**



```
struct TaintEdge {
    TaintNode src;
    TaintNode dst;
};
```

Running the Module

opt -load=libLLVM_TaintPass.so' -taintanalyzer -instnamer < '/media/fady/Data/Work/Defcon/LLVM/llvmPass/testcases/bof.ll'



```
==============================-------------=================================
🔥 Vulnerability detected: dangerous use of 'strcpy' 🔥
🔀 Analysis Trace (argument index is zero based):
        📥 Function : main argument 0
        🔄 Function : func1 argument 0
        🔄 Function : func2 argument 0
        🔄 Function : func3 argument 0
        ❌ Function : strcpy argument 0
==============================-------------=================================
```

Q & A