

Lab 5: LLVM Pass

Software Testing 2021

2022/03/31

This lab will cover

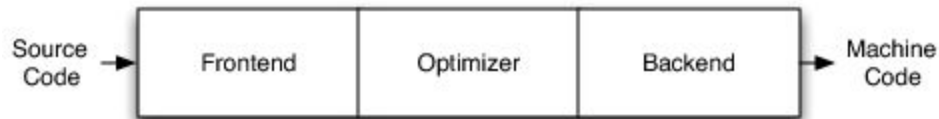
- a little bit compiler concept
- llvm pass
- some assembly concept
- some C++ OOP concept



LLVM

Three-Phase Compiler

- Frontend, Optimizer, Backend
 - 解析、優化、輸出
 - source code -> IR -> machine code



Why LLVM?

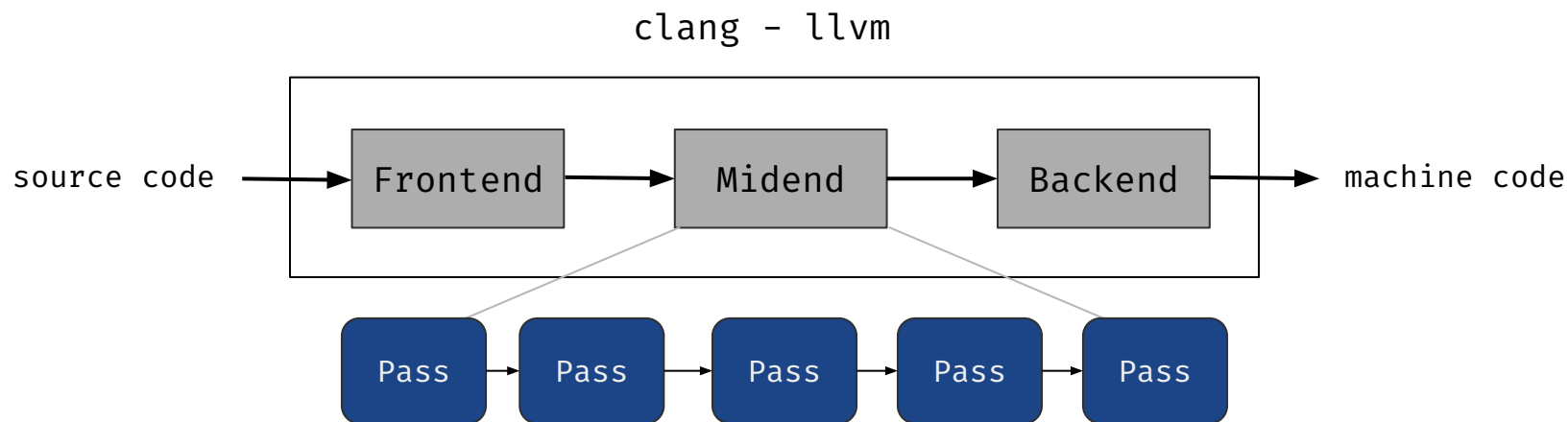
- Analysis
 - control flow graph
- Instrumentation
 - sanitizer

LLVM Pass

Caution!

- All following content works on **llvm-11**, other versions are not guaranteed!
- All API can be found in **<https://llvm.org/doxygen/index.html>**

Pass



Pass

- inherit from Pass class
 - ModulePass
 - FunctionPass
 - ...

```
class ExamplePass : public ModulePass {  
  
public:  
    static char ID;  
    ExamplePass() : ModulePass(ID) { }  
  
    bool doInitialization(Module &M) override;  
    bool runOnModule(Module &M) override;  
};
```

```
bool ExamplePass::runOnModule(Module &M) {  
  
    /*Do things here */  
  
    return true;  
}
```

```
static void registerExamplePass(const PassManagerBuilder &,  
                                Legacy::PassManagerBase &PM) {  
  
    PM.add(new ExamplePass());  
}  
  
static RegisterStandardPasses RegisterExamplePass(  
    PassManagerBuilder::EP_OptimizerLast, registerExamplePass);  
  
static RegisterStandardPasses RegisterExamplePass0(  
    PassManagerBuilder::EP_EnabledOnOptLevel0, registerExamplePass);
```

Pass

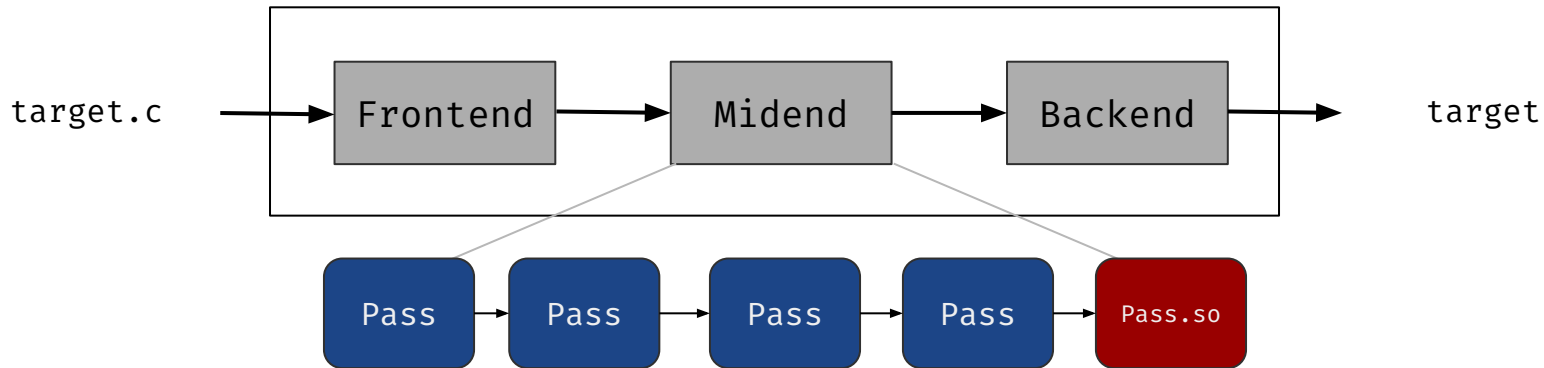
```
CXXFLAGS := -fno-rtti -fPIC `llvm-config --cxxflags` `llvm-config --ldflags` -shared  
clang++ $(CXXFLAGS) ./Pass.cc -o Pass.so
```



Pass

```
CFLAGS := `llvm-config --ldflags`  
clang $(CFLAGS) -Xclang -load -Xclang Pass.so target.c -o target
```

clang - llvm



LLVM Structure

- Module
- Function
- BasicBlock
- Instruction (IR)

```
for (auto &F : M) {  
    for (auto &BB : F) {  
        for (auto &I : BB) {  
            ...  
        }  
    }  
}
```

LLVM Structure

- Module
 - top-level structure in an LLVM program

```
// Look up the specified global variable in the module symbol table.  
GlobalVariable* llvm::Module::getGlobalVariable(StringRef Name)  
  
// Look up the specified function in the module symbol table.  
FunctionCallee getOrInsertFunction(StringRef Name, FunctionType *T)
```

LLVM Structure

- Function

```
// get entry block of function
BasicBlock& getEntryBlock()

// get function arguments through index
Argument * getArg(unsigned i) const
// get function arguments through iterator
for(auto it = F.arg_begin(); it != F.arg_end(); it++) {

}
```

LLVM Structure

- BasicBlock

```
const Function * getParent() const

// get first place to insert IR
iterator getFirstInsertionPt()

// Unlink 'this' from the containing function,
// but do not delete it.
void removeFromParent()

// Insert unlinked basic block into a function.
void insertInto(Function *Parent, BasicBlock *InsertBefore=nullptr)
```

LLVM Structure

- Instruction (IR)

```
const BasicBlock * getParent() const

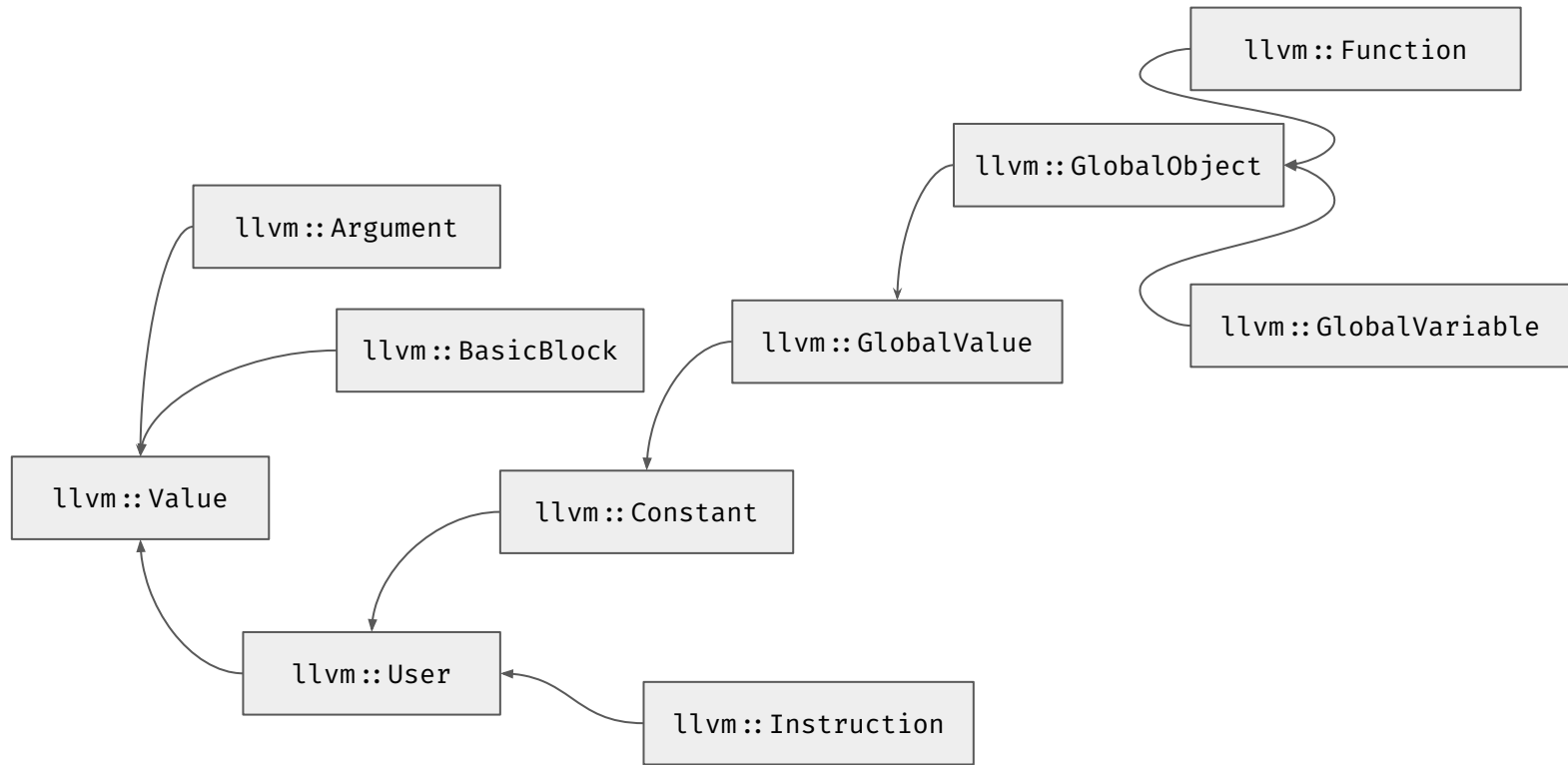
// This method unlinks 'this' from the containing basic block,
// but does not delete it.
void removeFromParent()

// Insert an unlinked instruction into a basic block immediately
// before the specified instruction.
void insertBefore(Instruction *InsertPos)

// Get the metadata of given kind attached to this Instruction.
MDNode * getMetadata(StringRef Kind) const

// Set the metadata of the specified kind to the specified node.
void setMetadata(StringRef Kind, MDNode *Node)
```

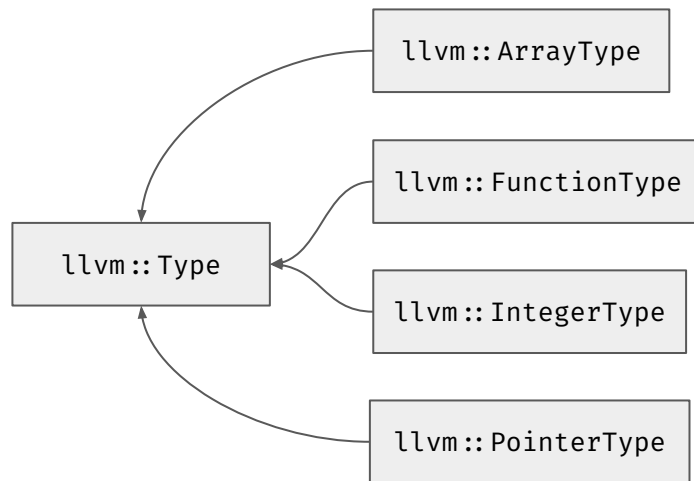

Value



Value

```
// Change all uses of this to point to a new Value.  
void replaceAllUsesWith(Value *V)  
  
// Change non-metadata uses of this to point to a new Value.  
void replaceNonMetadataUsesWith(Value *V)
```

Type



ConstantInt

- get instance of int32 ConstantInt

```
/* get instance of int32 ConstantInt with value 87 */  
  
/* static IntegerType * getInt32Ty (LLVMContext &C) */  
IntegerType *Int32Ty = IntegerType::getInt32Ty(M.getContext());  
  
/* static Constant * get (Type *Ty, uint64_t V, bool IsSigned=false) */  
ConstantInt *Val = ConstantInt::get(Int32Ty, 87);
```

getOrInsertFunction

```
// used for debug
void debug(int id) {

    if (id == 9527)
        fprintf(stderr, "debug mode\n");
    else
        fprintf(stderr, "bad id !\n");
}
```

```
/* Insert a debug function into Module M */

/* static FunctionType * get (Type *Result, ArrayRef< Type * > Params, bool isVarArg) */
std::vector<Type *>FnArgs;
FnArgs.push_back(Int32Ty);

FunctionType *FnTy = FunctionType::get(VoidTy, FnArgs, false);
// or
FunctionType *FnTy = FunctionType::get(VoidTy, {Int32Ty}, false);

/// Look up the specified function in the module symbol table. If it does not
/// exist, add a prototype for the function and return it.
/* FunctionCallee getOrInsertFunction(StringRef Name, FunctionType *T) */
FunctionCallee Fn = M.getOrInsertFunction("debug", FnTy);
```

IRBuilder

- insert new IR at specific position

```
/* Insert IR at beginning of BasicBlock */

BasicBlock::iterator IP = BB.getFirstInsertionPt();
IRBuilder<> IRB(&(*IP));

/* Insert function call */
/* CallInst *CreateCall(FunctionCallee Callee, ArrayRef<Value *> Args = None,
                        const Twine &Name = "", MDNode *FPMathTag = nullptr) */
IRB.CreateCall(Fn, ConstantInt::get(Int32Ty, 87));
```

IRBuilder

- insert new IR at specific position

```
/* Insert IR at the beginning of BasicBlock */  
  
BasicBlock::iterator IP = BB.getFirstInsertionPt();  
IRBuilder<> IRB(&(*IP));  
  
/* Make a new global variable with initializer type i8*. */  
Value *Argv1 = IRB.CreateGlobalStringPtr("function");  
  
/* Insert function call with string argument */  
IRB.CreateCall(Fn, Argv1);
```

LLVM IR

- **getelementptr** Instruction

- used to get the address of a subelement of an **aggregate** data structure.

```
struct RT {  
    char A;  
    int B[10][20];  
    char C;  
};  
  
struct ST {  
    int X;  
    double Y;  
    struct RT Z;  
};  
  
int *foo(struct ST *s) {  
    return &s[1].Z.B[5][13];  
}
```

```
%struct.RT = type { i8, [10 x [20 x i32]], i8 }  
%struct.ST = type { i32, double, %struct.RT }  
  
define i32* @foo(%struct.ST* %s) nounwind uwtable readnone optsize ssp {  
entry:  
    %arrayidx = getelementptr inbounds %struct.ST, %struct.ST* %s, i64 1, i32 2, i32 1, i64 5, i64 13  
    ret i32* %arrayidx  
}
```


Example 1: AFL LLVM Pass

https://github.com/google/AFL/blob/master/llvm_mode/afl-llvm-pass.so.cc

Coverage Instrumentation

- Instrument following code into each BasicBlock

```
cur_location = <COMPILE_TIME_RANDOM>;  
shared_mem[cur_location ^ prev_location]++;  
prev_location = cur_location >> 1;
```

Coverage Instrumentation

- Get Global Variable `shared_mem`, `prev_loc`

```
cur_location = <COMPILE_TIME_RANDOM>;
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

Pass.cc

```
/* Get globals for the SHM region and the previous location. Note that
   __afl_prev_loc is thread-local. */

GlobalVariable *AFLMapPtr =
    new GlobalVariable(M, PointerType::get(Int8Ty, 0), false,
                      GlobalValue::ExternalLinkage, 0, "__afl_area_ptr");

GlobalVariable *AFLPrevLoc = new GlobalVariable(
    M, Int32Ty, false, GlobalValue::ExternalLinkage, 0, "__afl_prev_loc",
    0, GlobalVariable::GeneralDynamicTLSModel, 0, false);
```

In afl-llvm-rt.o.c


```
u8* __afl_area_ptr = __afl_area_initial;
__thread u32 __afl_prev_loc;
```

Coverage Instrumentation

- Make up cur_loc

- ```
cur_location = <COMPILE_TIME_RANDOM>;
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

Pass.cc



```
/* Make up cur_loc */

unsigned int cur_loc = random(MAP_SIZE);

ConstantInt *CurLoc = ConstantInt::get(Int32Ty, cur_loc);
```

# Coverage Instrumentation

- Load prev\_loc

- ```
cur_location = <COMPILE_TIME_RANDOM>;
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

In LLVM, to access global variable value or pointer value, we need to use LoadInst

Pass.cc

```
/* Load prev_loc */
LoadInst *PrevLoc = IRB.CreateLoad(AFLPrevLoc);

/* The 'zext' instruction zero extends its operand to type ty2. */
Value *PrevLocCasted = IRB.CreateZExt(PrevLoc, IRB.getInt32Ty());
```

Coverage Instrumentation

- Load SHM pointer

-

```
cur_location = <COMPILE_TIME_RANDOM>;  
shared_mem[cur_location ^ prev_location]++;  
prev_location = cur_location >> 1;
```

In LLVM, getElementPtr only calculates address, it does not access memory

Pass.cc

```
/* Load SHM pointer */  
  
LoadInst *MapPtr = IRB.CreateLoad(AFLMapPtr);  
  
/* GEP: getElementPtr instruction is used to get  
the address of a subelement of an aggregate data structure. */  
Value *MapPtrIdx =  
    IRB.CreateGEP(MapPtr, IRB.CreateXor(PrevLocCasted, CurLoc));
```

Coverage Instrumentation

- Update bitmap

-

```
cur_location = <COMPILE_TIME_RANDOM>;  
shared_mem[cur_location ^ prev_location]++;  
prev_location = cur_location >> 1;
```

Pass.cc

```
/* Update bitmap */  
  
/* actually access memory */  
LoadInst *Counter = IRB.CreateLoad(MapPtrIdx);  
  
/* increase 1 */  
Value *Incr = IRB.CreateAdd(Counter, ConstantInt::get(Int8Ty, 1));  
  
/* Store counter back to bitmap */  
IRB.CreateStore(Incr, MapPtrIdx);
```

Coverage Instrumentation

- Set prev_loc to cur_loc >> 1

- ```
cur_location = <COMPILE_TIME_RANDOM>;
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

Pass.cc

```
/* Set prev_loc to cur_loc >> 1 */

StoreInst *Store =
 IRB.CreateStore(ConstantInt::get(Int32Ty, cur_loc >> 1), AFLPrevLoc);
```





Lab

# Lab

1. Given a target.c file, write a llvm pass to satisfy following requirements **without modifying** source code.
  - a. **Invoke debug function** with the first argument is **9527** in main function. (40%)
  - b. Let **argv[1] = "aesophor is ghost !!!"** before checking. (30%)
  - c. Let **argc = 9487** before checking. (30%)
2. Upgrade your llvm-lab-pass.cc to new e3.
  - a. We will compile your code and use it to instrument target.c.
  - b. We will execute ./target 1 and see the output result.
  - c. We may modify the output message after checking, so do not just instrument the output message.
3. We provide Pass.cc template, Makefile, and a docker file as testing environment.

*Do not compress the files and plagiarism.*

# target.c

```
int main(int argc, char *argv[]) {

 printf("argc: %d\n", argc);

 if (argc >= 2)
 printf("argv[1]: %s\n", argv[1]);
 else
 return 0;

 if (argc == 9487) {
 printf("Omg! Why your argc is so large ?\n");
 }
 else {
 printf("Looks like your argc is not large enough !\n");
 }

 if (!strcmp(argv[1], "aesophor is ghost !!!", 21)) {
 printf("Yes, aesophor is ghost !\n");
 }
 else {
 printf("Do you know aesophor ?\n");
 }

 return 0;
}
```

```
// used for debug
void debug(int id) {

 if (id == 9527)
 fprintf(stderr, "debug mode\n");
 else
 fprintf(stderr, "bad id !\n");
}
```

# llvm-pass.cc

```
bool ExamplePass::runOnModule(Module &M) {
 errs() << "runOnModule\n";

 for (auto &F : M) {
 /* add you code here */
 errs() << F.getName() << "\n";
 }

 return true;
}
```

# Demo

## Original

```
root@faadbe05ca04:/# cd /home/llvm-lab/share/
root@faadbe05ca04:/home/llvm-lab/share# make
clang++-11 -fno-rtti -fPIC `llvm-config-11 --cxxflags`
clang-11 `llvm-config-11 --ldflags` -g -ggdb -Xclang
runOnModule
debug
llvm.dbg.declare
fprintf
main
printf
strcmp
root@faadbe05ca04:/home/llvm-lab/share# ./target 1
argc: 2
argv[1]: 1
Looks like your argc is not large enough !
Do you know aesophor ?
root@faadbe05ca04:/home/llvm-lab/share# █
```

## Instrumentation

```
root@faadbe05ca04:/home/llvm-lab/share# make
clang++-11 -fno-rtti -fPIC `llvm-config-11 --cxxflags`
clang-11 `llvm-config-11 --ldflags` -g -ggdb -Xclang
root@faadbe05ca04:/home/llvm-lab/share# ./target 1
debug mode
argc: 9487
argv[1]: aesophor is ghost !!!
Omg! Why your argc is so large ?
Yes, aesophor is ghost !
root@faadbe05ca04:/home/llvm-lab/share# █
```

# Environment

1. we recommend you to test on linux, such as ubuntu, and install docker
2. environment in docker file:
  - a. Download llvm-lab.zip from github
  - b. unzip and cd to distribute
  - c. `sudo docker build -t llvm-lab . --no-cache`
  - d. `sudo docker run -v $PWD/share:/home/llvm-lab/share -it llvm-lab /bin/bash`
  - e. `cd /home/llvm-lab/share`
  - f. `make`
  - g. `./target 1`

# Debug

- compile error
  - clang error message
- Instrumentation debug
  - target output
  - gdb

Original

```
pwndbg> disas main
Dump of assembler code for function main:
0x00000000004011b0 <+0>: push rbp
0x00000000004011b1 <+1>: mov rbp, rsp
0x00000000004011b4 <+4>: sub rsp, 0x10
0x00000000004011b8 <+8>: mov DWORD PTR [rbp-0x4], 0x0
0x00000000004011bf <+15>: mov DWORD PTR [rbp-0x8], edi
0x00000000004011c2 <+18>: mov QWORD PTR [rbp-0x10], rsi
0x00000000004011c6 <+22>: mov esi, DWORD PTR [rbp-0x8]
0x00000000004011c9 <+25>: movabs rdi, 0x40201a
0x00000000004011d3 <+35>: mov al, 0x0
0x00000000004011d5 <+37>: call 0x401040 <printf@plt>
```

Instrumentation

```
pwndbg> disas main
Dump of assembler code for function main:
0x00000000004011b0 <+0>: push rbp
0x00000000004011b1 <+1>: mov rbp, rsp
0x00000000004011b4 <+4>: sub rsp, 0x20
0x00000000004011b8 <+8>: mov eax, 0x2537
0x00000000004011bd <+13>: mov DWORD PTR [rbp-0x14], edi
0x00000000004011c0 <+16>: mov edi, eax
0x00000000004011c2 <+18>: mov QWORD PTR [rbp-0x20], rsi
0x00000000004011c6 <+22>: call 0x401150 <debug>
0x00000000004011cb <+27>: movabs rcx, 0x40207f
0x00000000004011d5 <+37>: mov rdx, QWORD PTR [rbp-0x20]
0x00000000004011d9 <+41>: mov QWORD PTR [rdx+0x8], rcx
0x00000000004011dd <+45>: mov DWORD PTR [rbp-0x4], 0x0
0x00000000004011e4 <+52>: mov DWORD PTR [rbp-0x8], 0x250f
0x00000000004011eb <+59>: mov QWORD PTR [rbp-0x10], rdx
0x00000000004011ef <+63>: mov esi, DWORD PTR [rbp-0x8]
0x00000000004011f2 <+66>: movabs rdi, 0x40201a
0x00000000004011fc <+76>: mov al, 0x0
0x00000000004011fe <+78>: call 0x401040 <printf@plt>
```

# Debug

- call debug with arg1=9527=0x2537
- x64 calling convention
  - **arg1, arg2, arg3** will be set in **rdi, rsi, rdx**

## Original

```
pwndbg> disas main
Dump of assembler code for function main:
0x00000000004011b0 <+0>: push rbp
0x00000000004011b1 <+1>: mov rbp, rsp
0x00000000004011b4 <+4>: sub rsp, 0x10
0x00000000004011b8 <+8>: mov DWORD PTR [rbp-0x4], 0x0
0x00000000004011bf <+15>: mov DWORD PTR [rbp-0x8], edi
0x00000000004011c2 <+18>: mov QWORD PTR [rbp-0x10], rsi
0x00000000004011c6 <+22>: mov esi, DWORD PTR [rbp-0x8]
0x00000000004011c9 <+25>: movabs rdi, 0x40201a
0x00000000004011d3 <+35>: mov al, 0x0
0x00000000004011d5 <+37>: call 0x401040 <printf@plt>
```

## Instrumentation

```
pwndbg> disas main
Dump of assembler code for function main:
0x00000000004011b0 <+0>: push rbp
0x00000000004011b1 <+1>: mov rbp, rsp
0x00000000004011b4 <+4>: sub rsp, 0x20
0x00000000004011b8 <+8>: mov eax, 0x2537
0x00000000004011bd <+13>: mov DWORD PTR [rbp-0x14], edi
0x00000000004011c0 <+16>: mov edi, eax
0x00000000004011c2 <+18>: mov QWORD PTR [rbp-0x20], rsi
0x00000000004011c6 <+22>: call 0x401150 <debug>
0x00000000004011cb <+27>: movabs rcx, 0x40207f
0x00000000004011d5 <+37>: mov rdx, QWORD PTR [rbp-0x20]
0x00000000004011d9 <+41>: mov QWORD PTR [rdx+0x8], rcx
0x00000000004011dd <+45>: mov DWORD PTR [rbp-0x4], 0x0
0x00000000004011e4 <+52>: mov DWORD PTR [rbp-0x8], 0x250f
0x00000000004011eb <+59>: mov QWORD PTR [rbp-0x10], rdx
0x00000000004011ef <+63>: mov esi, DWORD PTR [rbp-0x8]
0x00000000004011f2 <+66>: movabs rdi, 0x40201a
0x00000000004011fc <+76>: mov al, 0x0
0x00000000004011fe <+78>: call 0x401040 <printf@plt>
```



# Debug

- let argc=9487=0x250f

- 

Original

|       |                   |            |
|-------|-------------------|------------|
| stack | ret               |            |
|       | rbp               |            |
|       | arg1: <b>argc</b> | [rbp-0x8]  |
|       | arg2              | [rbp-0x10] |

Instrumentation

|       |                   |            |
|-------|-------------------|------------|
| stack | ret               |            |
|       | rbp               |            |
|       | arg1: <b>9487</b> | [rbp-0x8]  |
|       | arg2              | [rbp-0x10] |

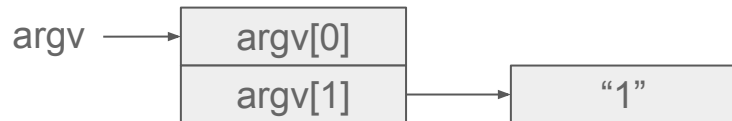
Instrumentation

```
pwndbg> disas main
Dump of assembler code for function main:
0x00000000004011b0 <+0>: push rbp
0x00000000004011b1 <+1>: mov rbp, rsp
0x00000000004011b4 <+4>: sub rsp, 0x20
0x00000000004011b8 <+8>: mov eax, 0x2537
0x00000000004011bd <+13>: mov DWORD PTR [rbp-0x14], edi
0x00000000004011c0 <+16>: mov edi, eax
0x00000000004011c2 <+18>: mov QWORD PTR [rbp-0x20], rsi
0x00000000004011c6 <+22>: call 0x401150 <debug>
0x00000000004011cb <+27>: movabs rcx, 0x40207f
0x00000000004011d5 <+37>: mov rdx, QWORD PTR [rbp-0x20]
0x00000000004011d9 <+41>: mov QWORD PTR [rdx+0x8], rcx
0x00000000004011dd <+45>: mov DWORD PTR [rbp-0x4], 0x0
0x00000000004011e4 <+52>: mov DWORD PTR [rbp-0x8], 0x250f
0x00000000004011eb <+59>: mov QWORD PTR [rbp-0x10], rdx
0x00000000004011ef <+63>: mov esi, DWORD PTR [rbp-0x8]
0x00000000004011f2 <+66>: movabs rdi, 0x40201a
0x00000000004011fc <+76>: mov al, 0x0
0x00000000004011fe <+78>: call 0x401040 <printf@plt>
```

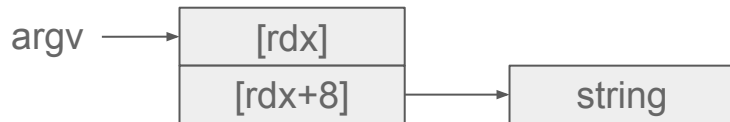
# Debug

- let argv[1]="aesophor is ghost !!!"

Original



Instrumentation



```
pwndbg> x/s 0x40207f
0x40207f: "aesophor is ghost !!!"
```

Instrumentation

```
pwndbg> disas main
Dump of assembler code for function main:
0x00000000004011b0 <+0>: push rbp
0x00000000004011b1 <+1>: mov rbp, rsp
0x00000000004011b4 <+4>: sub rsp, 0x20
0x00000000004011b8 <+8>: mov eax, 0x2537
0x00000000004011bd <+13>: mov DWORD PTR [rbp-0x14], edi
0x00000000004011c0 <+16>: mov edi, eax
0x00000000004011c2 <+18>: mov QWORD PTR [rbp-0x20], rsi
0x00000000004011c6 <+22>: call 0x401150 <debug>
0x00000000004011cb <+27>: movabs rcx, 0x40207f
0x00000000004011d5 <+37>: mov rdx, QWORD PTR [rbp-0x20]
0x00000000004011d9 <+41>: mov QWORD PTR [rdx+0x8], rcx
0x00000000004011dd <+45>: mov DWORD PTR [rbp-0x4], 0x0
0x00000000004011e4 <+52>: mov DWORD PTR [rbp-0x8], 0x250f
0x00000000004011eb <+59>: mov QWORD PTR [rbp-0x10], rdx
0x00000000004011ef <+63>: mov esi, DWORD PTR [rbp-0x8]
0x00000000004011f2 <+66>: movabs rdi, 0x40201a
0x00000000004011fc <+76>: mov al, 0x0
0x00000000004011fe <+78>: call 0x401040 <printf@plt>
```

# Questions

- `tl455047.cs09@nycu.edu.tw`

# Reference

- <https://llvm.org/docs/LangRef.html>
- <https://llvm.org/doxygen/index.html>
- <https://github.com/google/AFL>