Lab 5: LLVM Pass

Software Testing 2023

2023/03/23

whoami

- Software Quality Lab @ EC547
- TA
  a. 蔡惠喬
     - hctsai.cs10@nycu.edu.tw
  b. 陳舜寧
     - xdev11.cs11@nycu.edu.tw
  c. 張書銘
     - ljp.cs10@nycu.edu.tw

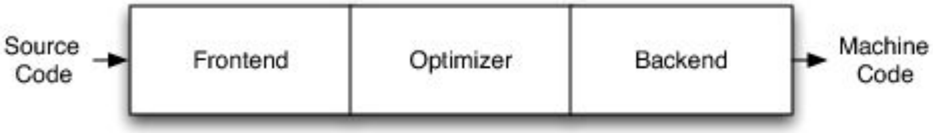
GitHub Repo

- <student_id>-ST-2023
- Add collaborators
  - XDEv11, chameleon10712, skhuang, LJP-TW

LLVM

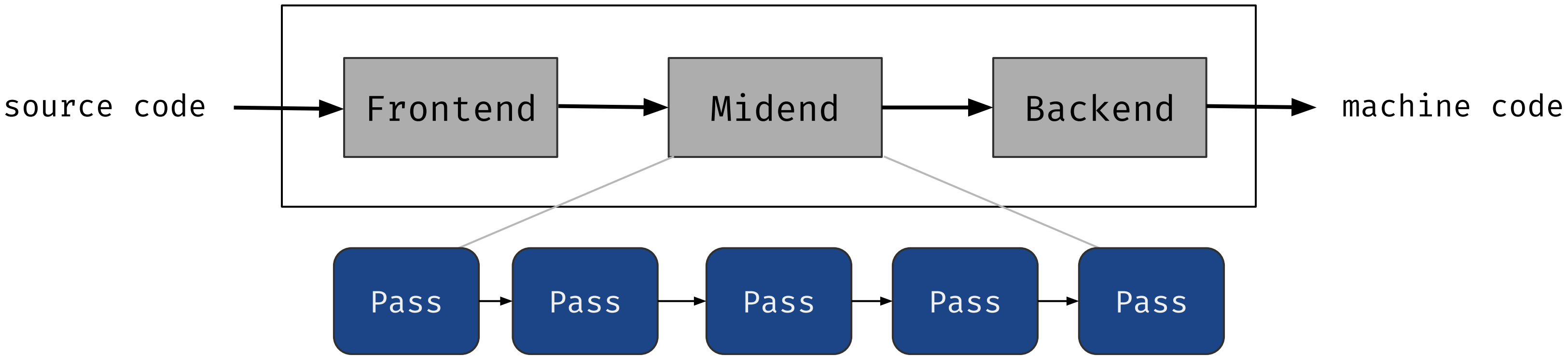
Three-Phase Compiler

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



LLVM

clang - llvm



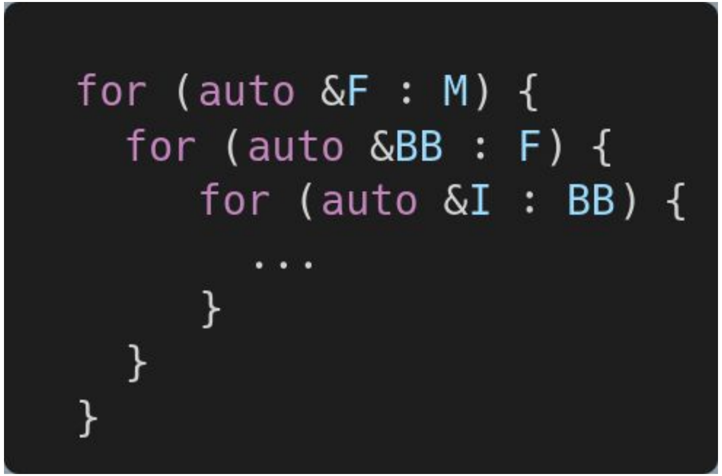
Why LLVM?

- Analysis
  - control flow graph
- Instrumentation
  - sanitizer

LLVM Pass

LLVM Structure

- Module
- Function
- BasicBlock
- Instruction

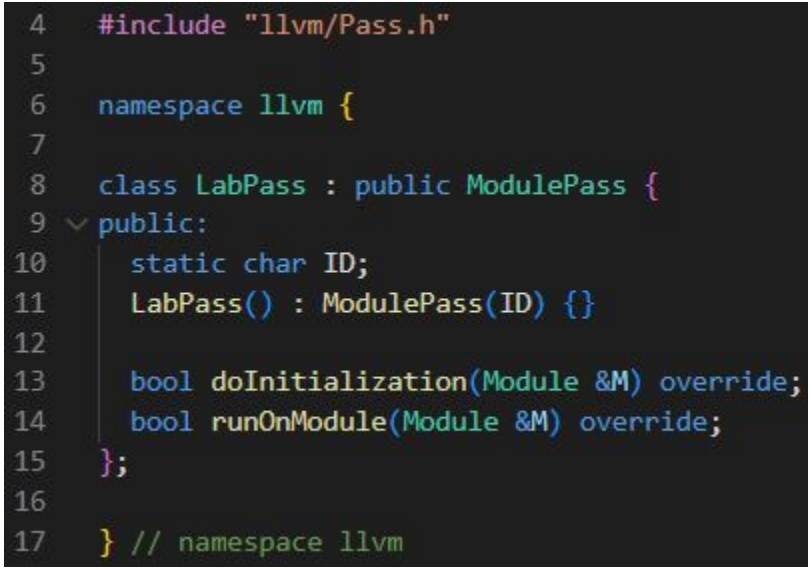


```
for (auto &F : M) {
  for (auto &BB : F) {
     for (auto &I : BB) {
       ...
     }
  }
}
```

Create LLVM Pass

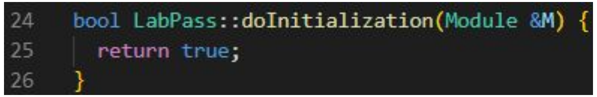
- Inherit from Pass class
- Depending on how your pass works, you should inherit from:
    - ModulePass
    - CallGraphSCCPass
    - FunctionPass
    - …

Create LLVM Pass

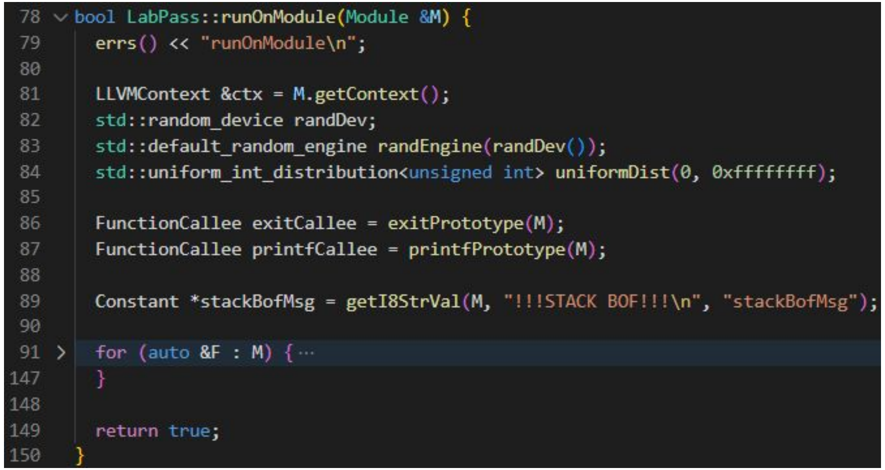


```
#include "llvm/Pass.h"

namespace llvm {

class LabPass : public ModulePass {
public:
  static char ID;
  LabPass() : ModulePass(ID) {}

  bool doInitialization(Module &M) override;
  bool runOnModule(Module &M) override;
};

} // namespace llvm
```

example/llvmpass/lab-pass.h



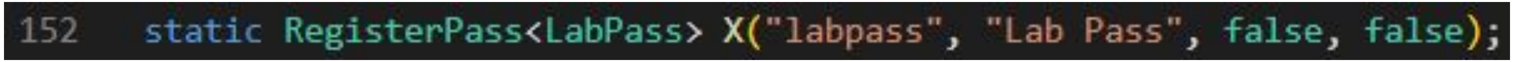
```
bool LabPass::doInitialization(Module &M) {
  return true;
}
```



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

  LLVMContext &ctx = M.getContext();
  std::random_device randDev;
  std::default_random_engine randEngine(randDev());
  std::uniform_int_distribution<unsigned int> uniformDist(0, 0xffffffff);

  FunctionCallee exitCallee = exitPrototype(M);
  FunctionCallee printfCallee = printfPrototype(M);

  Constant *stackBofMsg = getI8StrVal(M, "!!!STACK BOF!!!\n", "stackBofMsg");

  for (auto &F : M) {···
  }

  return true;
}
```

example/llvmpass/lab-pass.cc

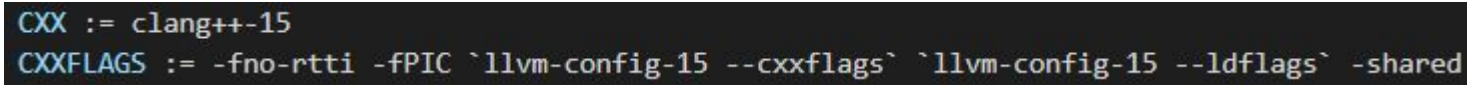
Register LLVM Pass



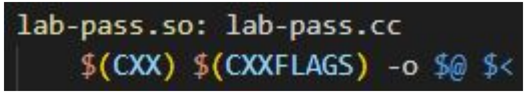
```
static RegisterPass<LabPass> X("labpass", "Lab Pass", false, false);
```

example/llvmpass/lab-pass.cc

Compile LLVM Pass



```
CXX := clang++-15
CXXFLAGS := -fno-rtti -fPIC `llvm-config-15 --cxxflags` `llvm-config-15 --ldflags` -shared
```

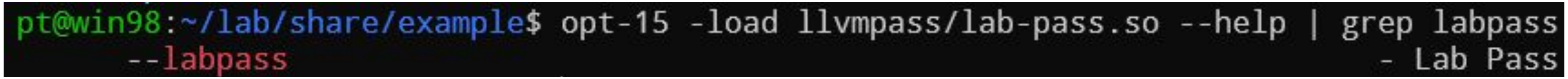


```
lab-pass.so: lab-pass.cc
	$(CXX) $(CXXFLAGS) -o $@ $<
```

example/llvmpass/Makefile

Use LLVM Pass

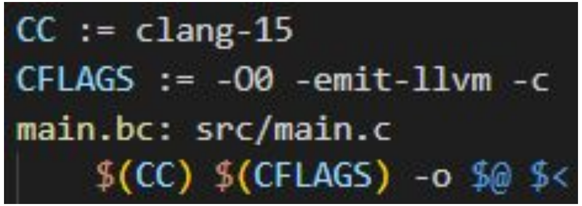
- Use “opt” to load and apply LLVM pass



```
pt@win98:~/lab/share/example$ opt-15 -load llvmpass/lab-pass.so --help | grep labpass
      --labpass                                                            - Lab Pass
```

Use LLVM Pass

- Compile source code to LLVM IR binary file (.bc)

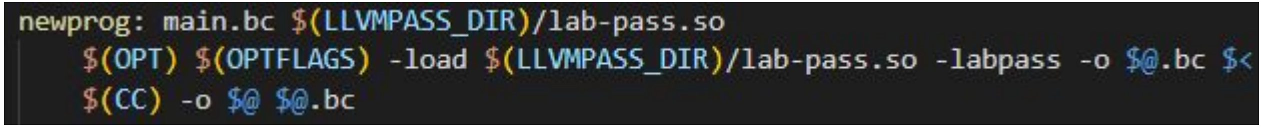


```
CC := clang-15
CFLAGS := -O0 -emit-llvm -c
main.bc: src/main.c
	$(CC) $(CFLAGS) -o $@ $<
```

example/Makefile

Use LLVM Pass

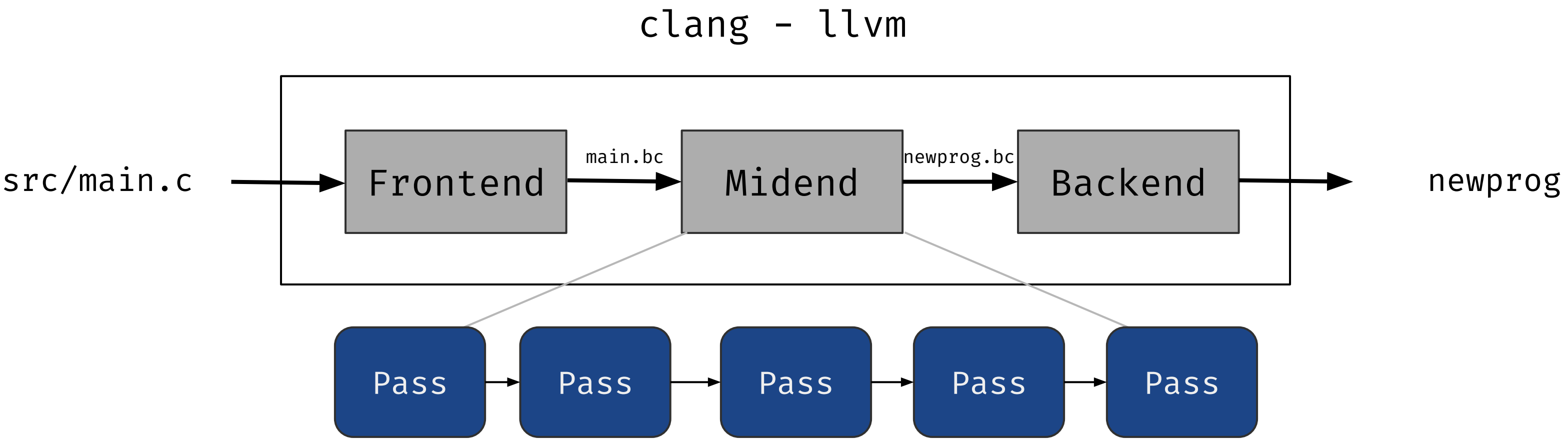
- Compile source code to LLVM IR binary file (.bc)
- Run LLVM Pass
- Generate executable



```
newprog: main.bc $(LLVMPASS_DIR)/lab-pass.so
    $(OPT) $(OPTFLAGS) -load $(LLVMPASS_DIR)/lab-pass.so -labpass -o $@.bc $<
    $(CC) -o $@ $@.bc
```

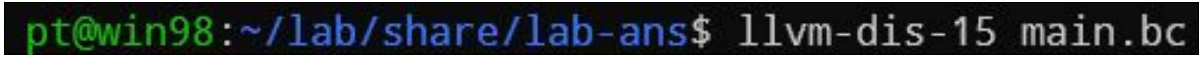
example/Makefile

LLVM

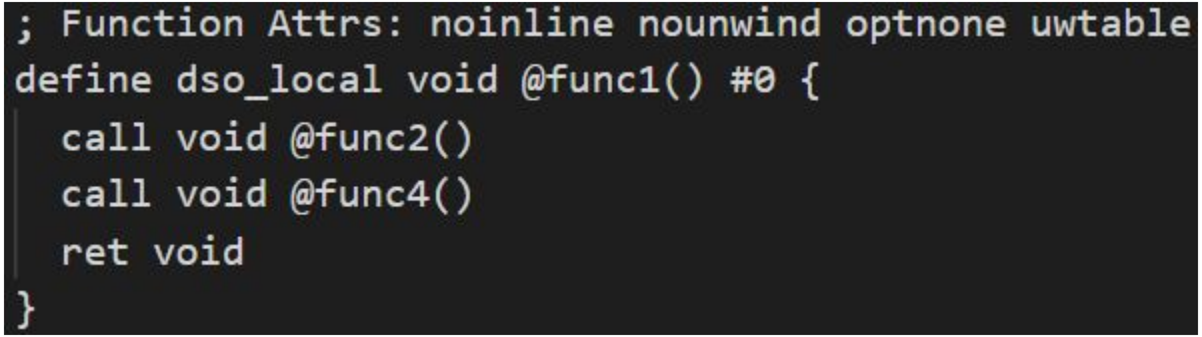

clang - llvm
src/main.c
Frontend
main.bc
Midend
newprog.bc
Backend
newprog
Pass
Pass
Pass
Pass
Pass

BC to IR

- You can use llvm-dis to disassemble .bc to .ll (LLVM IR)



```
pt@win98:~/lab/share/lab-ans$ llvm-dis-15 main.bc
```



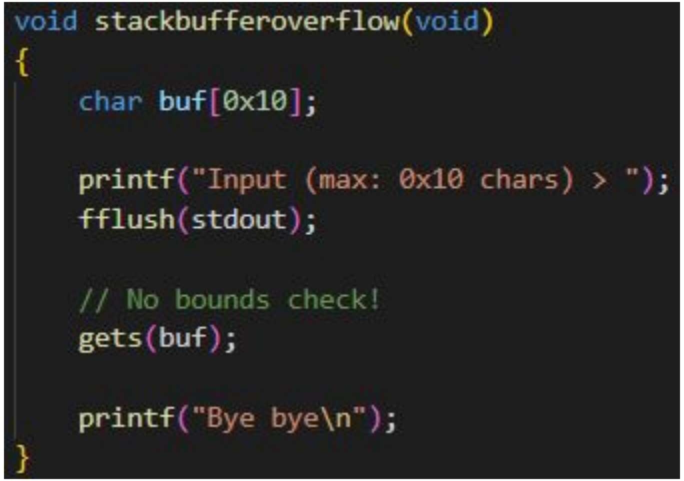
```
; Function Attrs: noinline nounwind optnone uwtable
define dso_local void @func1() #0 {
  call void @func2()
  call void @func4()
  ret void
}
```

main.ll

Example

Stack-based Buffer Overflow

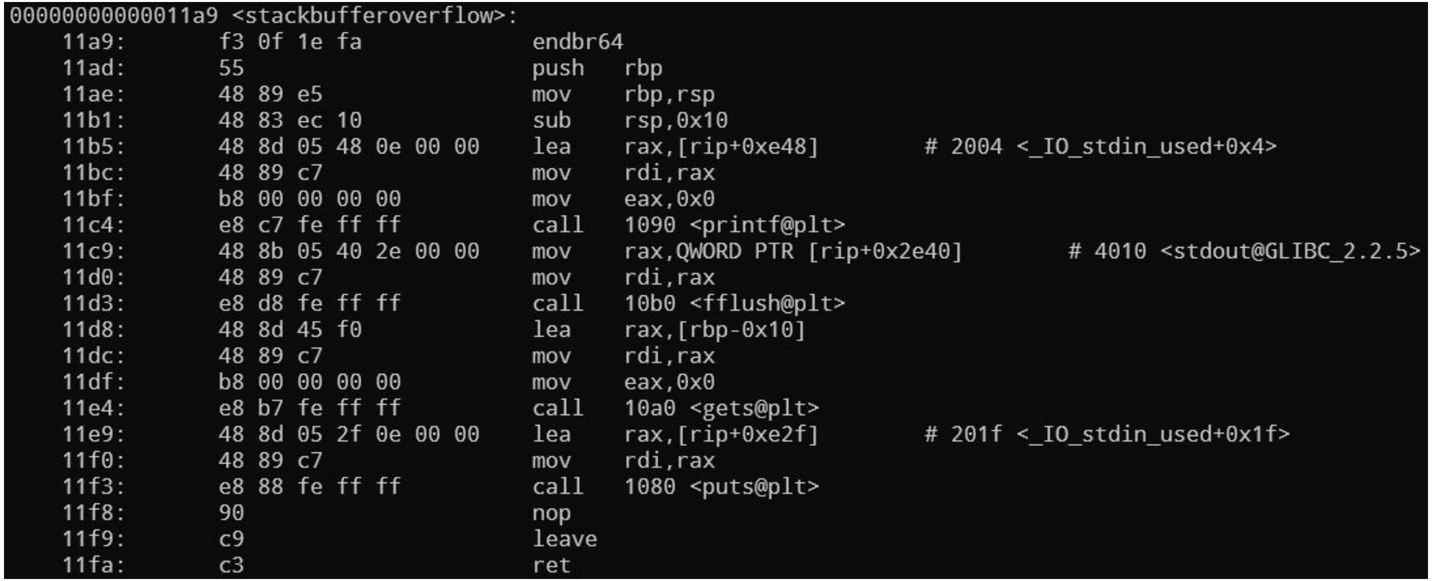
- Writes to a memory address on the program's call stack outside of the intended data structure



```
void stackbufferoverflow(void)
{
    char buf[0x10];

    printf("Input (max: 0x10 chars) > ");
    fflush(stdout);

    // No bounds check!
    gets(buf);

    printf("Bye bye\n");
}
```

Stack-based Buffer Overflow

- The first argument of “gets” is “rbp-0x10”, which is “buf”

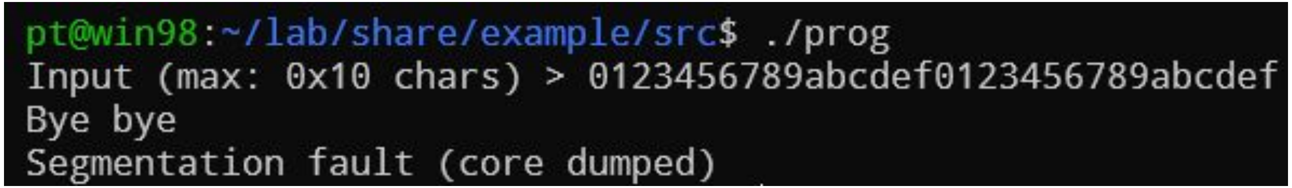


```
0000000000011a9 <stackbufferoverflow>:
    11a9:       f3 0f 1e fa             endbr64
    11ad:       55                      push   rbp
    11ae:       48 89 e5                mov    rbp,rsp
    11b1:       48 83 ec 10             sub    rsp,0x10
    11b5:       48 8d 05 48 0e 00 00    lea    rax,[rip+0xe48]        # 2004 <_IO_stdin_used+0x4>
    11bc:       48 89 c7                mov    rdi,rax
    11bf:       b8 00 00 00 00          mov    eax,0x0
    11c4:       e8 c7 fe ff ff          call   1090 <printf@plt>
    11c9:       48 8b 05 40 2e 00 00    mov    rax,QWORD PTR [rip+0x2e40]        # 4010 <stdout@GLIBC_2.2.5>
    11d0:       48 89 c7                mov    rdi,rax
    11d3:       e8 d8 fe ff ff          call   10b0 <fflush@plt>
    11d8:       48 8d 45 f0             lea    rax,[rbp-0x10]
    11dc:       48 89 c7                mov    rdi,rax
    11df:       b8 00 00 00 00          mov    eax,0x0
    11e4:       e8 b7 fe ff ff          call   10a0 <gets@plt>
    11e9:       48 8d 05 2f 0e 00 00    lea    rax,[rip+0xe2f]        # 201f <_IO_stdin_used+0x1f>
    11f0:       48 89 c7                mov    rdi,rax
    11f3:       e8 88 fe ff ff          call   1080 <puts@plt>
    11f8:       90                      nop
    11f9:       c9                      leave
    11fa:       c3                      ret
```

cd example/src; make; objdump -d -M intel prog

Stack-based Buffer Overflow

- Overwriting important data (e.g. return address) can lead to program crashes or more serious security issues



```
pt@win98:~/lab/share/example/src$ ./prog
Input (max: 0x10 chars) > 0123456789abcdef0123456789abcdef
Bye bye
Segmentation fault (core dumped)
```

(buf) rbp-0x10	
rbp-0x8	
rbp+0x0	
rbp+0x8	return address

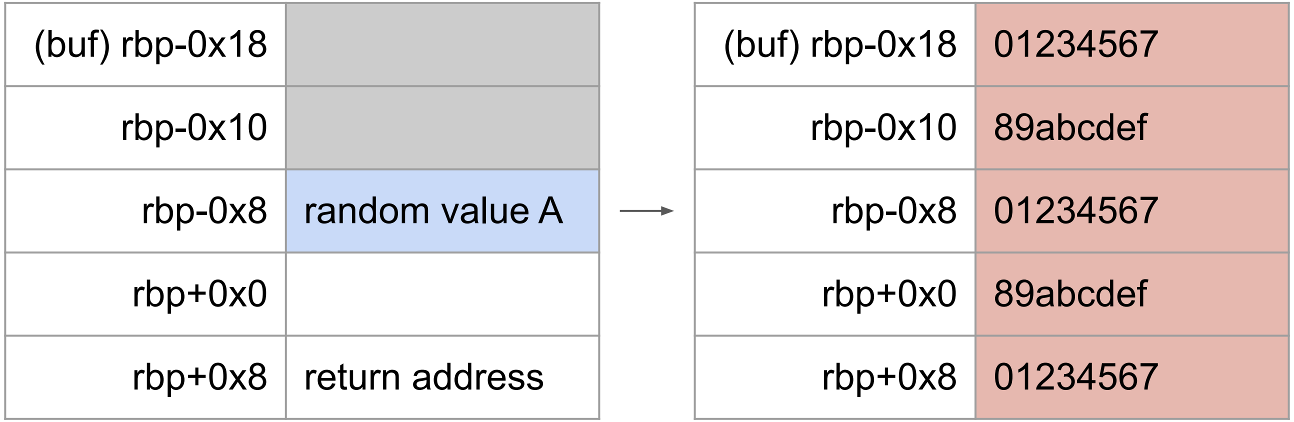
→

(buf) rbp-0x10	01234567
rbp-0x8	89abcdef
rbp+0x0	01234567
rbp+0x8	89abcdef

Example LLVM Pass

- Create a local variable at the bottom of the stack
- Store a random value in this local variable
- Verify whether the local variable has been changed
- If it has changed, we can infer that a stack-based buffer overflow has occurred.
    - Print "!!!STACK BOF!!!" message
    - Terminate the process to prevent further exploitation

Example LLVM Pass

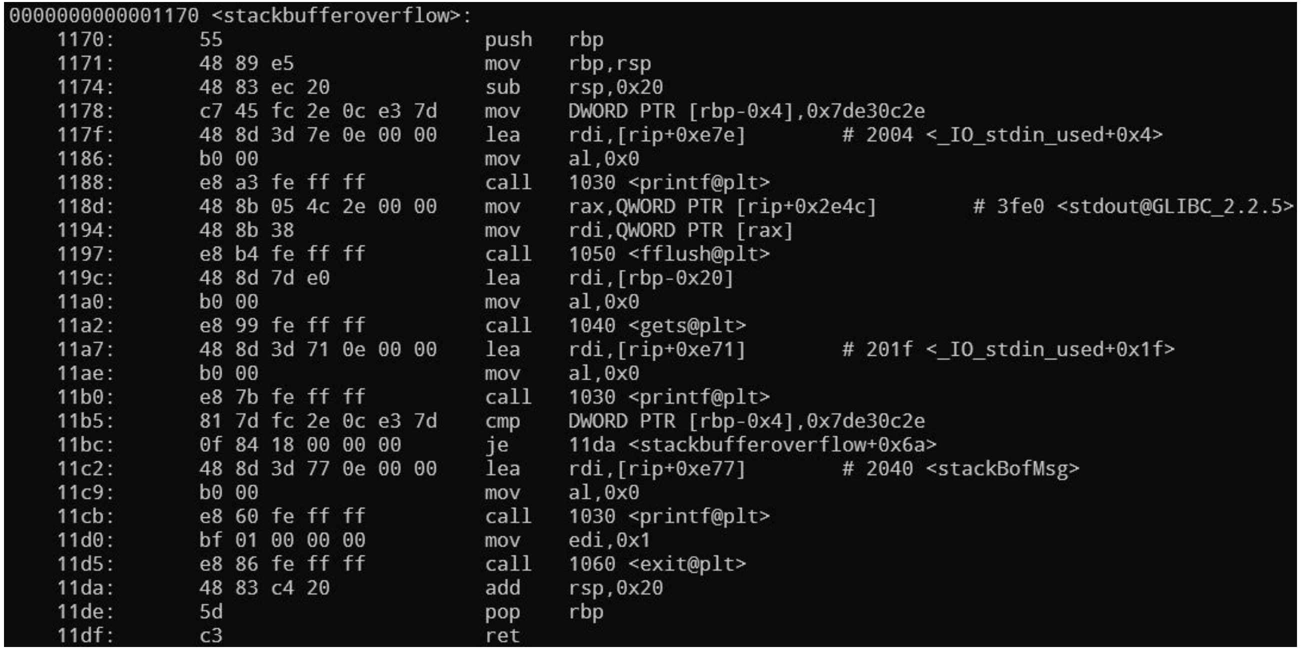


(buf) rbp-0x18	
rbp-0x10	
rbp-0x8	random value A
rbp+0x0	
rbp+0x8	return address

(buf) rbp-0x18	01234567
rbp-0x10	89abcdef
rbp-0x8	01234567
rbp+0x0	89abcdef
rbp+0x8	01234567

Stack-based buffer overflow detected!

Example LLVM Pass

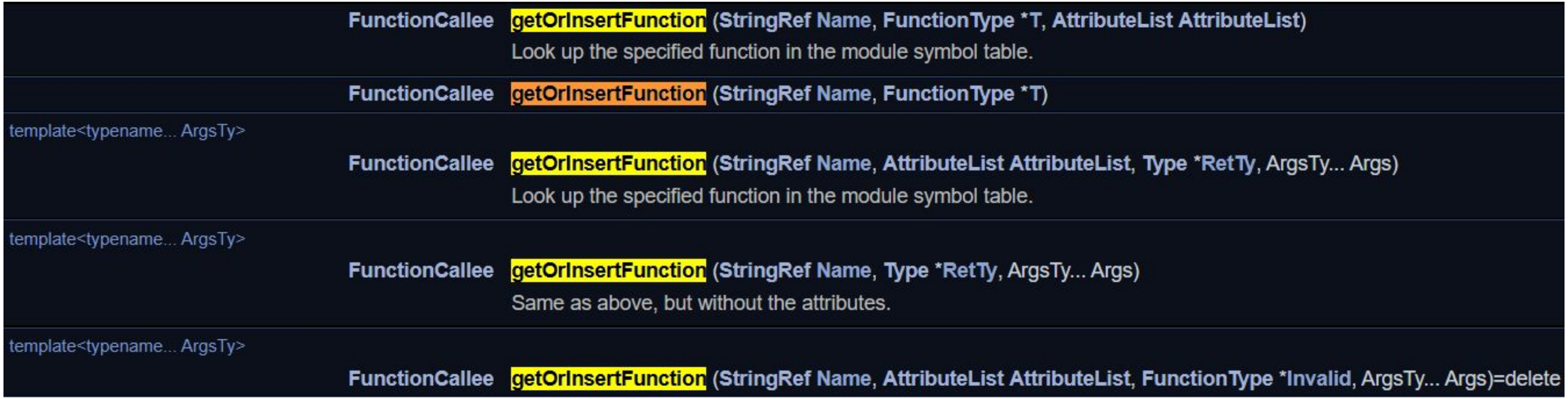


```
0000000000001170 <stackbufferoverflow>:
    1170:       55                      push   rbp
    1171:       48 89 e5                mov    rbp,rsp
    1174:       48 83 ec 20             sub    rsp,0x20
    1178:       c7 45 fc 2e 0c e3 7d    mov    DWORD PTR [rbp-0x4],0x7de30c2e
    117f:       48 8d 3d 7e 0e 00 00    lea    rdi,[rip+0xe7e]        # 2004 <_IO_stdin_used+0x4>
    1186:       b0 00                   mov    al,0x0
    1188:       e8 a3 fe ff ff          call   1030 <printf@plt>
    118d:       48 8b 05 4c 2e 00 00    mov    rax,QWORD PTR [rip+0x2e4c]        # 3fe0 <stdout@GLIBC_2.2.5>
    1194:       48 8b 38                mov    rdi,QWORD PTR [rax]
    1197:       e8 b4 fe ff ff          call   1050 <fflush@plt>
    119c:       48 8d 7d e0             lea    rdi,[rbp-0x20]
    11a0:       b0 00                   mov    al,0x0
    11a2:       e8 99 fe ff ff          call   1040 <gets@plt>
    11a7:       48 8d 3d 71 0e 00 00    lea    rdi,[rip+0xe71]        # 201f <_IO_stdin_used+0x1f>
    11ae:       b0 00                   mov    al,0x0
    11b0:       e8 7b fe ff ff          call   1030 <printf@plt>
    11b5:       81 7d fc 2e 0c e3 7d    cmp    DWORD PTR [rbp-0x4],0x7de30c2e
    11bc:       0f 84 18 00 00 00       je     11da <stackbufferoverflow+0x6a>
    11c2:       48 8d 3d 77 0e 00 00    lea    rdi,[rip+0xe77]        # 2040 <stackBofMsg>
    11c9:       b0 00                   mov    al,0x0
    11cb:       e8 60 fe ff ff          call   1030 <printf@plt>
    11d0:       bf 01 00 00 00          mov    edi,0x1
    11d5:       e8 86 fe ff ff          call   1060 <exit@plt>
    11da:       48 83 c4 20             add    rsp,0x20
    11de:       5d                      pop    rbp
    11df:       c3                      ret
```

cd example; make; objdump -d -M intel newprog

Module

- getOrInsertFunction

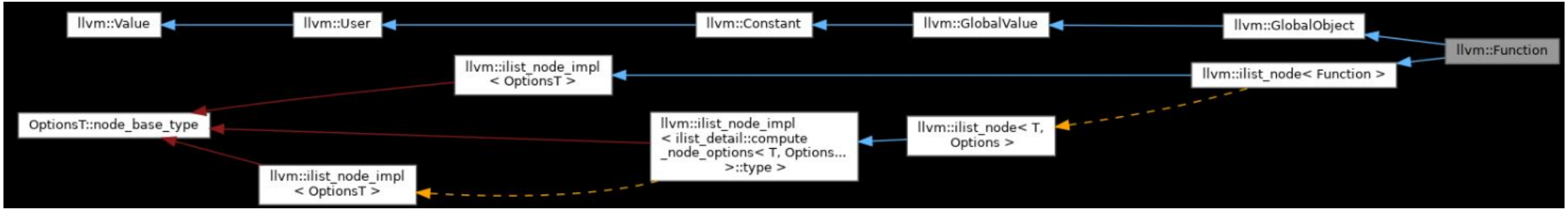


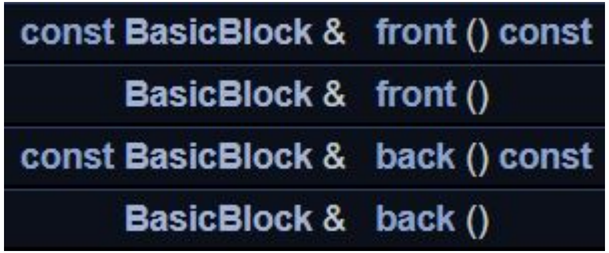
FunctionCallee	getOrInsertFunction (StringRef Name, FunctionType *T, AttributeList AttributeList) Look up the specified function in the module symbol table.
FunctionCallee	getOrInsertFunction (StringRef Name, FunctionType *T)
template<typename... ArgsTy> FunctionCallee	getOrInsertFunction (StringRef Name, AttributeList AttributeList, Type *RetTy, ArgsTy... Args) Look up the specified function in the module symbol table.
template<typename... ArgsTy> FunctionCallee	getOrInsertFunction (StringRef Name, Type *RetTy, ArgsTy... Args) Same as above, but without the attributes.
template<typename... ArgsTy> FunctionCallee	getOrInsertFunction (StringRef Name, AttributeList AttributeList, FunctionType *Invalid, ArgsTy... Args)=delete

https://llvm.org/doxygen/classllvm_1_1Module.html

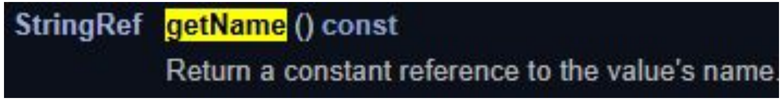
Function

- getName
- front
- back


llvm::Value
llvm::User
llvm::Constant
llvm::GlobalValue
llvm::GlobalObject
llvm::Function
llvm::ilist_node_impl < OptionsT >
llvm::ilist_node< Function >
OptionsT::node_base_type
llvm::ilist_node_impl < ilist_detail::compute _node_options< T, Options... >::type >
llvm::ilist_node< T, Options >
llvm::ilist_node_impl < OptionsT >



```
const BasicBlock &   front () const
      BasicBlock &   front ()
const BasicBlock &   back () const
      BasicBlock &   back ()
```

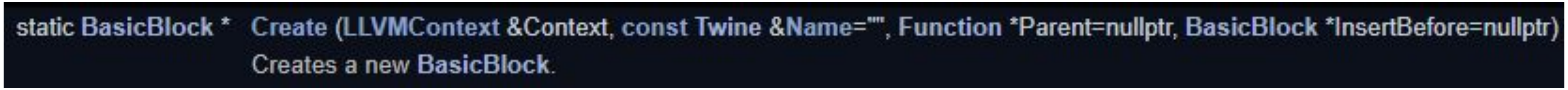


StringRef getName () const
Return a constant reference to the value's name.

https://llvm.org/doxygen/classllvm_1_1Function.html https://llvm.org/doxygen/classllvm_1_1Value.html

BasicBlock

- Create
- front
- back

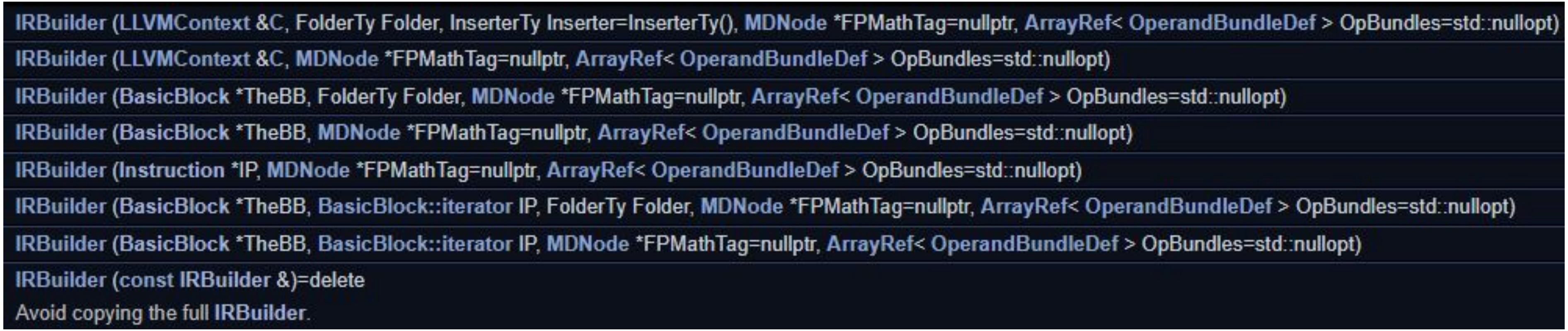


```
static BasicBlock *   Create (LLVMContext &Context, const Twine &Name="", Function *Parent=nullptr, BasicBlock *InsertBefore=nullptr)
                      Creates a new BasicBlock.
```

https://llvm.org/doxygen/classllvm_1_1BasicBlock.html

IRBuilder

- Constructor

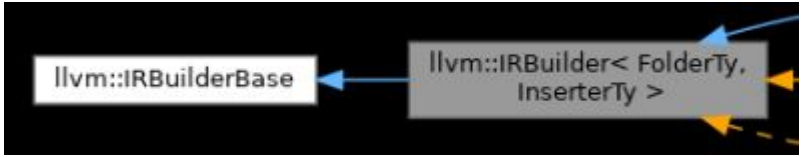


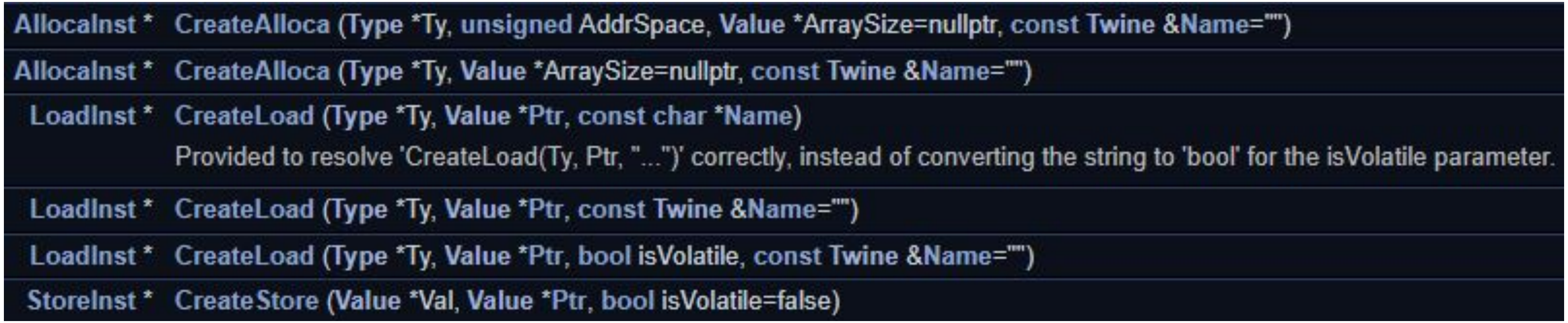
```
IRBuilder (LLVMContext &C, FolderTy Folder, InserterTy Inserter=InserterTy(), MDNode *FPMathTag=nullptr, ArrayRef< OperandBundleDef > OpBundles=std::nullopt)
IRBuilder (LLVMContext &C, MDNode *FPMathTag=nullptr, ArrayRef< OperandBundleDef > OpBundles=std::nullopt)
IRBuilder (BasicBlock *TheBB, FolderTy Folder, MDNode *FPMathTag=nullptr, ArrayRef< OperandBundleDef > OpBundles=std::nullopt)
IRBuilder (BasicBlock *TheBB, MDNode *FPMathTag=nullptr, ArrayRef< OperandBundleDef > OpBundles=std::nullopt)
IRBuilder (Instruction *IP, MDNode *FPMathTag=nullptr, ArrayRef< OperandBundleDef > OpBundles=std::nullopt)
IRBuilder (BasicBlock *TheBB, BasicBlock::iterator IP, FolderTy Folder, MDNode *FPMathTag=nullptr, ArrayRef< OperandBundleDef > OpBundles=std::nullopt)
IRBuilder (BasicBlock *TheBB, BasicBlock::iterator IP, MDNode *FPMathTag=nullptr, ArrayRef< OperandBundleDef > OpBundles=std::nullopt)
IRBuilder (const IRBuilder &)=delete
Avoid copying the full IRBuilder.
```

https://llvm.org/doxygen/classllvm_1_1IRBuilder.html

IRBuilder

- Create IR command



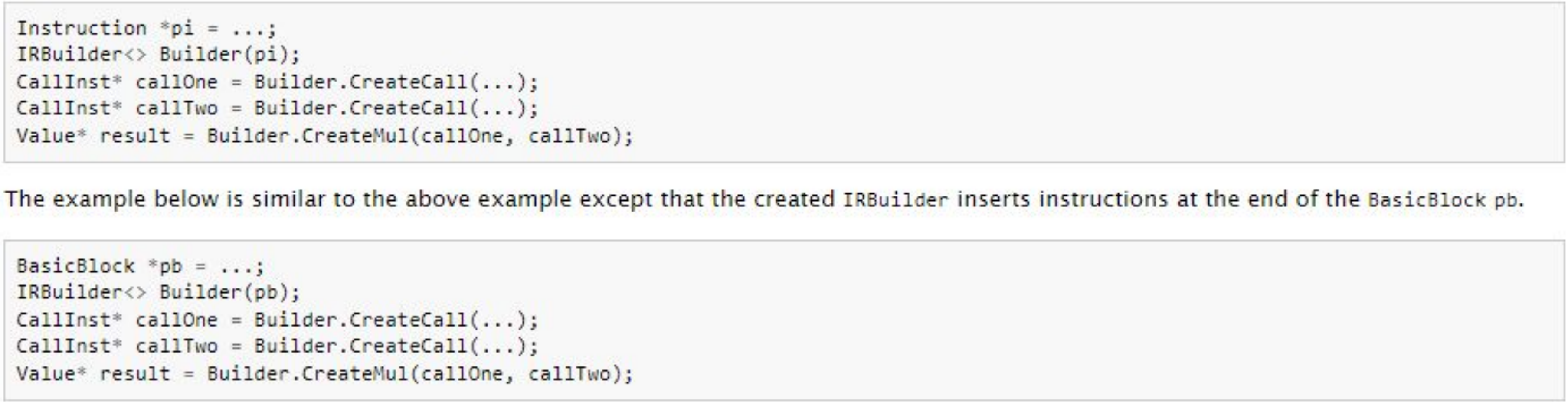


AllocaInst *	CreateAlloca (Type *Ty, unsigned AddrSpace, Value *ArraySize=nullptr, const Twine &Name="")
AllocaInst *	CreateAlloca (Type *Ty, Value *ArraySize=nullptr, const Twine &Name="")
LoadInst *	CreateLoad (Type *Ty, Value *Ptr, const char *Name) Provided to resolve 'CreateLoad(Ty, Ptr, "...")' correctly, instead of converting the string to 'bool' for the isVolatile parameter.
LoadInst *	CreateLoad (Type *Ty, Value *Ptr, const Twine &Name="")
LoadInst *	CreateLoad (Type *Ty, Value *Ptr, bool isVolatile, const Twine &Name="")
StoreInst *	CreateStore (Value *Val, Value *Ptr, bool isVolatile=false)

https://llvm.org/doxygen/classllvm_1_1IRBuilderBase.html

IRBuilder

- Please take a look at the official tutorial



```
Instruction *pi = ...;
IRBuilder<> Builder(pi);
CallInst* callOne = Builder.CreateCall(...);
CallInst* callTwo = Builder.CreateCall(...);
Value* result = Builder.CreateMul(callOne, callTwo);
```

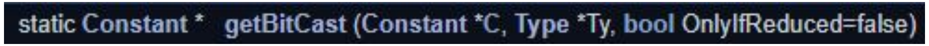
The example below is similar to the above example except that the created `IRBuilder` inserts instructions at the end of the `BasicBlock` `pb`.

```
BasicBlock *pb = ...;
IRBuilder<> Builder(pb);
CallInst* callOne = Builder.CreateCall(...);
CallInst* callTwo = Builder.CreateCall(...);
Value* result = Builder.CreateMul(callOne, callTwo);
```

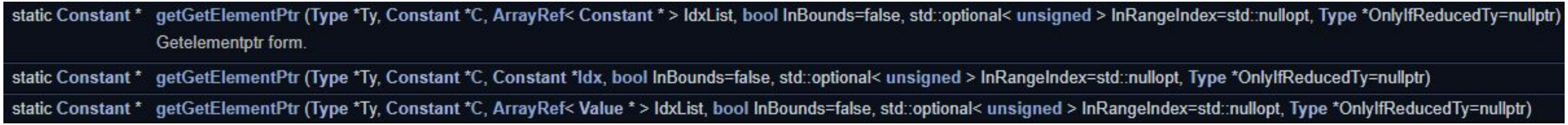
https://llvm.org/docs/ProgrammersManual.html#creating-and-inserting-new-instructions

ConstantExpr

- getBitCast
- getGetElementPtr



static Constant * getBitCast (Constant *C, Type *Ty, bool OnlyIfReduced=false)



static Constant * getGetElementPtr (Type *Ty, Constant *C, ArrayRef< Constant * > IdxList, bool InBounds=false, std::optional< unsigned > InRangeIndex=std::nullopt, Type *OnlyIfReducedTy=nullptr)
Getelementptr form.

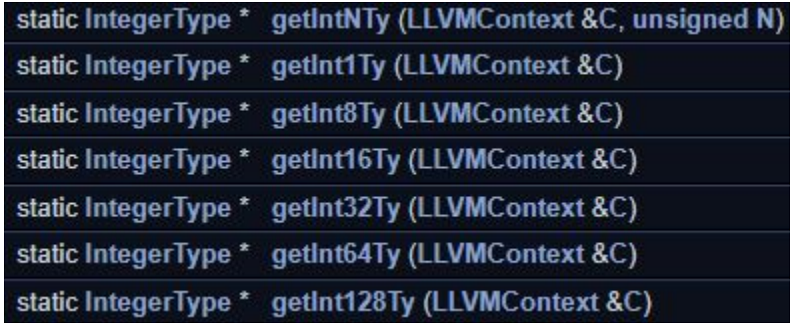
static Constant * getGetElementPtr (Type *Ty, Constant *C, Constant *Idx, bool InBounds=false, std::optional< unsigned > InRangeIndex=std::nullopt, Type *OnlyIfReducedTy=nullptr)

static Constant * getGetElementPtr (Type *Ty, Constant *C, ArrayRef< Value * > IdxList, bool InBounds=false, std::optional< unsigned > InRangeIndex=std::nullopt, Type *OnlyIfReducedTy=nullptr)

https://llvm.org/doxygen/classllvm_1_1ConstantExpr.html

Type

- getInt8PtrTy
- getInt32Ty
- getVoidTy



static IntegerType *	getIntNTy (LLVMContext &C, unsigned N)
static IntegerType *	getInt1Ty (LLVMContext &C)
static IntegerType *	getInt8Ty (LLVMContext &C)
static IntegerType *	getInt16Ty (LLVMContext &C)
static IntegerType *	getInt32Ty (LLVMContext &C)
static IntegerType *	getInt64Ty (LLVMContext &C)
static IntegerType *	getInt128Ty (LLVMContext &C)

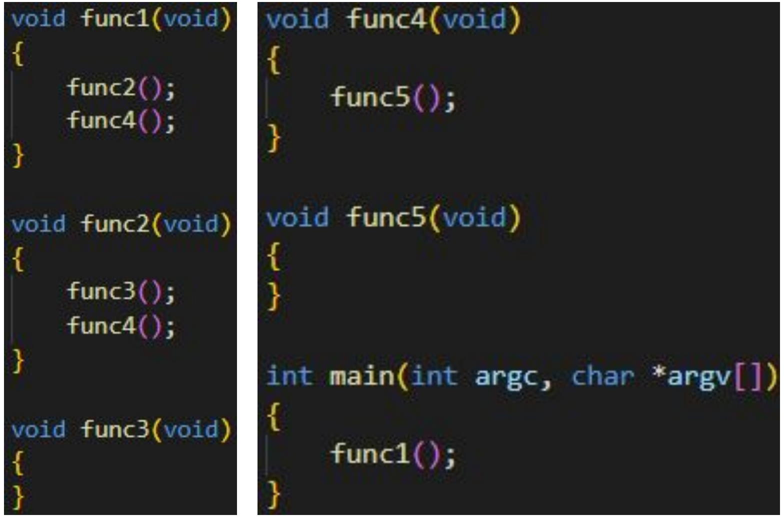
https://llvm.org/doxygen/classllvm_1_1Type.html

Lab

Lab 5

- Create an LLVM Pass that can log the following message when a function is invoked:
    - Indent N spaces before outputting, where N is the function depth, and define the depth of the main function as 0
    - Function name
    - Function address
- You can only modify the files listed below:
    - lab/llvmpass/lab-pass.cc
    - lab/llvmpass/lab-pass.h
- Due to ASLR and PIE, the addresses of the functions will be different in each execution, but the offset will remain the same.

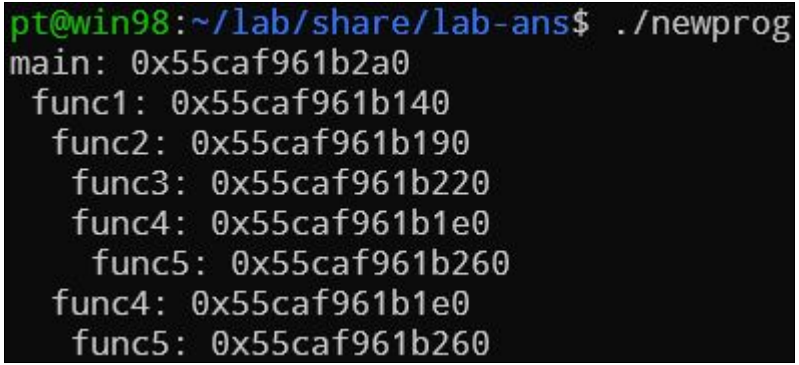
Lab 5



```c
void func1(void)
{
    func2();
    func4();
}

void func2(void)
{
    func3();
    func4();
}

void func3(void)
{
}
```

```c
void func4(void)
{
    func5();
}

void func5(void)
{
}

int main(int argc, char *argv[])
{
    func1();
}
```

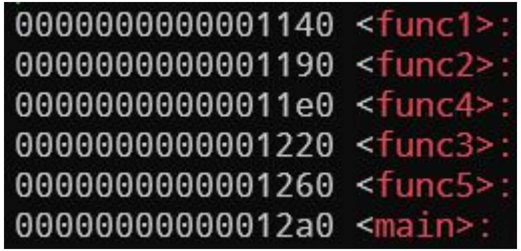
lab/src/main.c



```
pt@win98:~/lab/share/lab-ans$ ./newprog
main: 0x55caf961b2a0
 func1: 0x55caf961b140
  func2: 0x55caf961b190
   func3: 0x55caf961b220
   func4: 0x55caf961b1e0
    func5: 0x55caf961b260
  func4: 0x55caf961b1e0
   func5: 0x55caf961b260
```

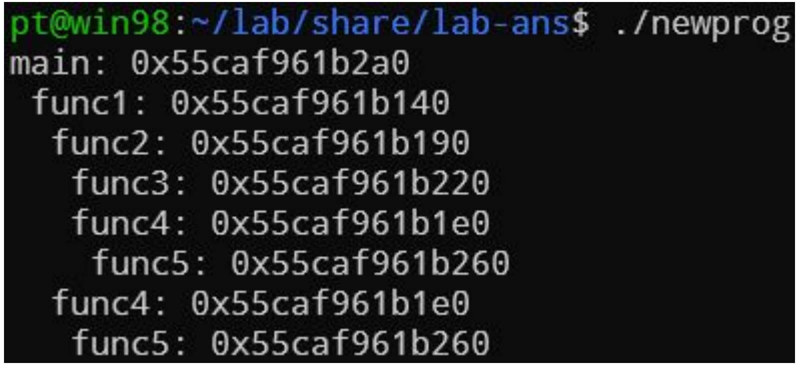
expected output

Lab 5



```
0000000000001140 <func1>:
0000000000001190 <func2>:
00000000000011e0 <func4>:
0000000000001220 <func3>:
0000000000001260 <func5>:
00000000000012a0 <main>:
```

objdump -d -M intel newprog | grep -E "func.>:|main>:"



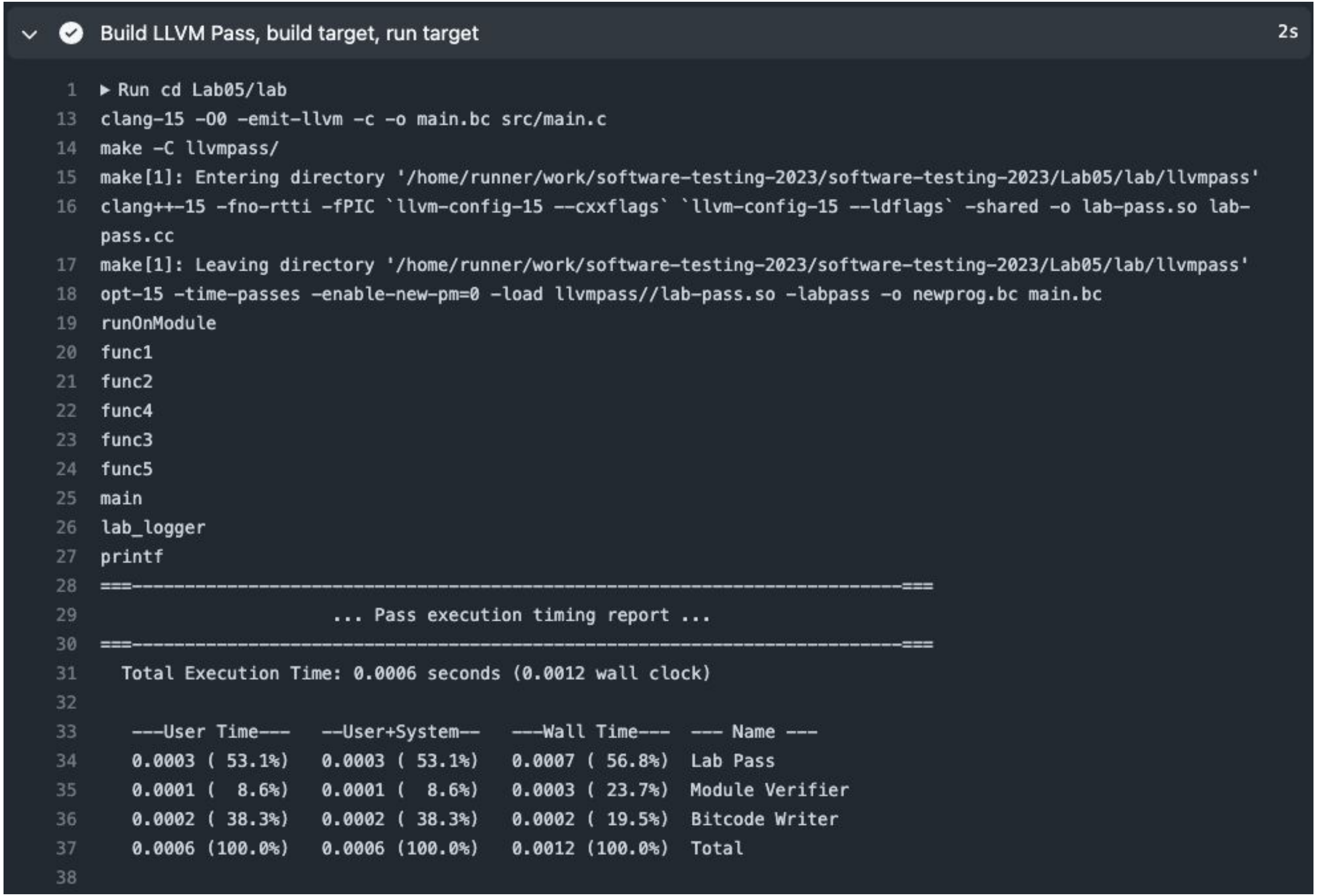
```
pt@win98:~/lab/share/lab-ans$ ./newprog
main: 0x55caf961b2a0
 func1: 0x55caf961b140
  func2: 0x55caf961b190
   func3: 0x55caf961b220
   func4: 0x55caf961b1e0
    func5: 0x55caf961b260
  func4: 0x55caf961b1e0
   func5: 0x55caf961b260
```

expected output

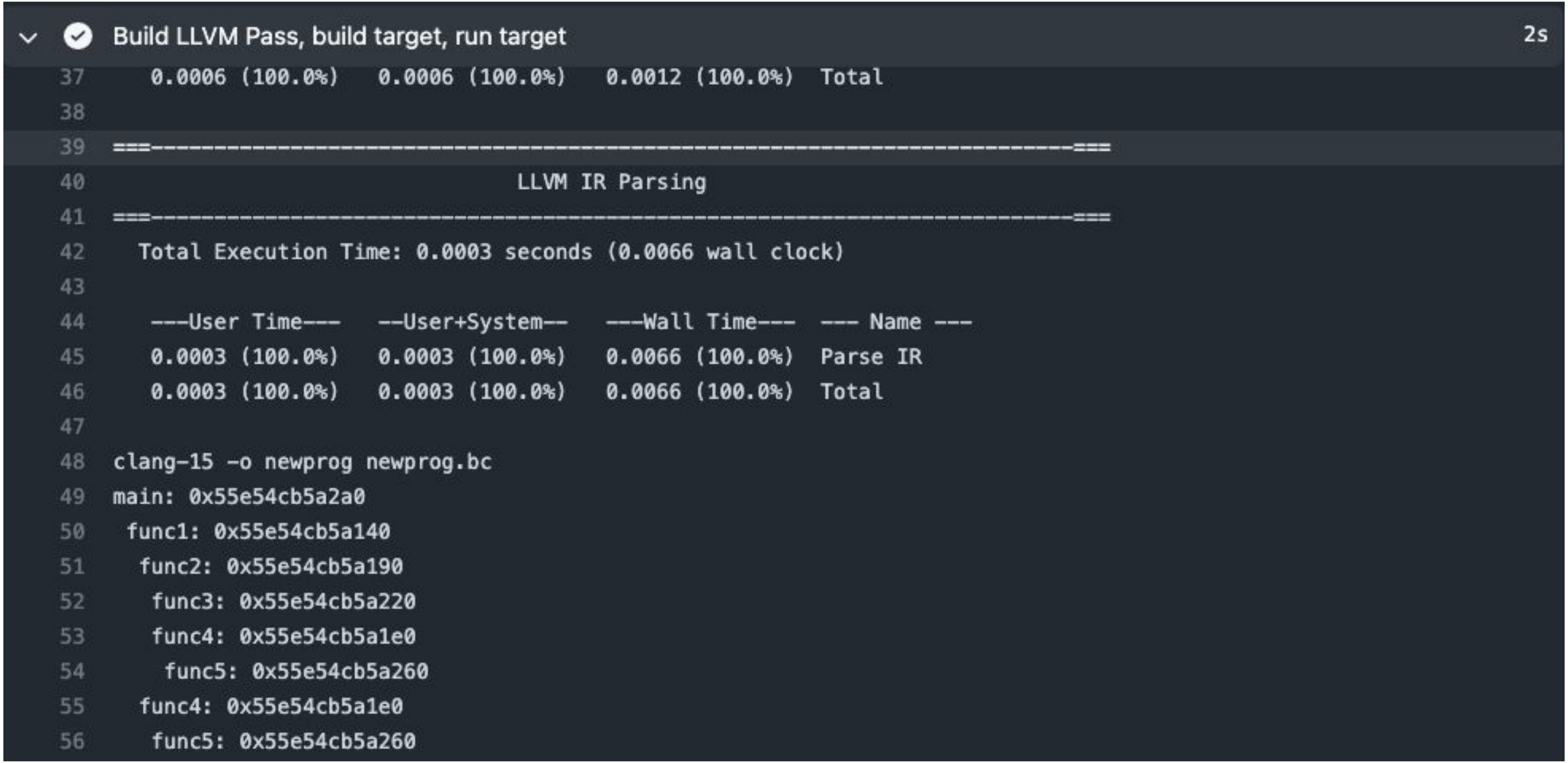
Lab Environment

- Ubuntu 22.04
- sudo apt-get install clang-15 llvm-15

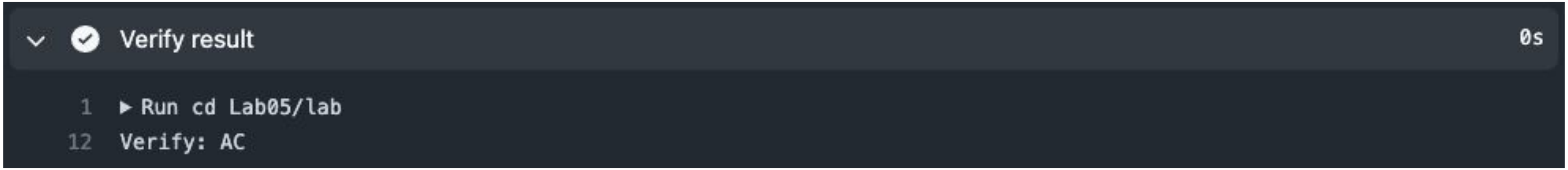
Example Output



```
▶ Run cd Lab05/lab
clang-15 -O0 -emit-llvm -c -o main.bc src/main.c
make -C llvmpass/
make[1]: Entering directory '/home/runner/work/software-testing-2023/software-testing-2023/Lab05/lab/llvmpass'
clang++-15 -fno-rtti -fPIC `llvm-config-15 --cxxflags` `llvm-config-15 --ldflags` -shared -o lab-pass.so lab-pass.cc
make[1]: Leaving directory '/home/runner/work/software-testing-2023/software-testing-2023/Lab05/lab/llvmpass'
opt-15 -time-passes -enable-new-pm=0 -load llvmpass//lab-pass.so -labpass -o newprog.bc main.bc
runOnModule
func1
func2
func4
func3
func5
main
lab_logger
printf
===-------------------------------------------------------------------------===
                      ... Pass execution timing report ...
===-------------------------------------------------------------------------===
  Total Execution Time: 0.0006 seconds (0.0012 wall clock)

   ---User Time---   --User+System--   ---Wall Time---  --- Name ---
   0.0003 ( 53.1%)   0.0003 ( 53.1%)   0.0007 ( 56.8%)  Lab Pass
   0.0001 (  8.6%)   0.0001 (  8.6%)   0.0003 ( 23.7%)  Module Verifier
   0.0002 ( 38.3%)   0.0002 ( 38.3%)   0.0002 ( 19.5%)  Bitcode Writer
   0.0006 (100.0%)   0.0006 (100.0%)   0.0012 (100.0%)  Total

```



```
   0.0006 (100.0%)   0.0006 (100.0%)   0.0012 (100.0%)  Total

===-------------------------------------------------------------------------===
                                LLVM IR Parsing
===-------------------------------------------------------------------------===
  Total Execution Time: 0.0003 seconds (0.0066 wall clock)

   ---User Time---   --User+System--   ---Wall Time---  --- Name ---
   0.0003 (100.0%)   0.0003 (100.0%)   0.0066 (100.0%)  Parse IR
   0.0003 (100.0%)   0.0003 (100.0%)   0.0066 (100.0%)  Total

clang-15 -o newprog newprog.bc
main: 0x55e54cb5a2a0
 func1: 0x55e54cb5a140
  func2: 0x55e54cb5a190
   func3: 0x55e54cb5a220
   func4: 0x55e54cb5a1e0
    func5: 0x55e54cb5a260
  func4: 0x55e54cb5a1e0
   func5: 0x55e54cb5a260
```



```
▶ Run cd Lab05/lab
Verify: AC
```

Submission

Submission

- Add Lab05 status badge in your README file
- Please submit your Github repo `<student_id>-ST-2023`, and upload these to E3:
  - commit URL
    - refer to Lab 1 submission
  - github action job URL
    - refer to Lab 3 submission

Reference

- https://llvm.org/doxygen/
- https://llvm.org/doxygen/classes.html
- https://llvm.org/docs/GettingStarted.html
- https://llvm.org/docs/WritingAnLLVMPass.html
- https://llvm.org/docs/ProgrammersManual.html