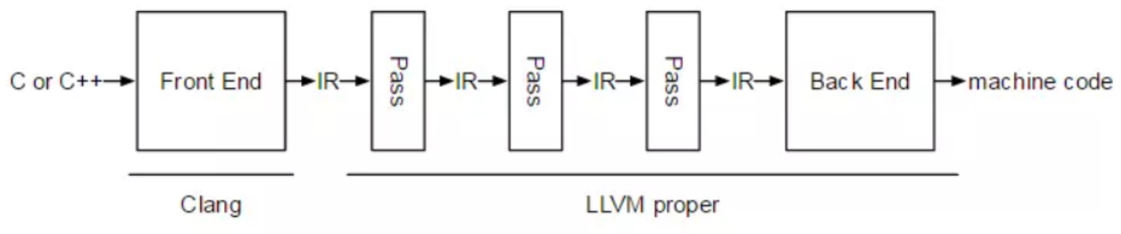
Obfuscator-llvm

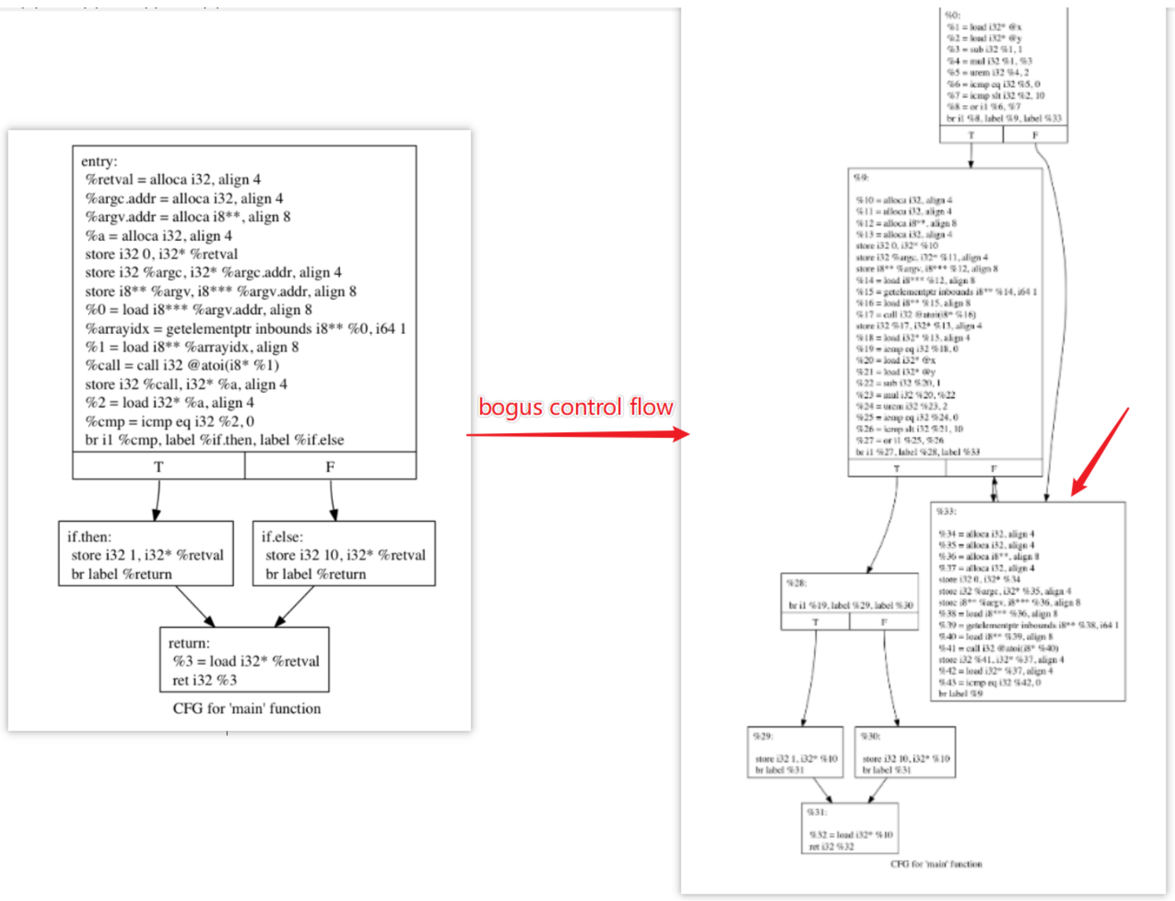
overall architecture



bogus control flow

- This method modifies a function call graph by adding a basic block before the current basic block. This new basic block contains an opaque predicate and then makes a conditional jump to the original basic block.

  The original basic block is also cloned and filled up with junk instructions chosen at random.



```
entry:
 %retval = alloca i32, align 4
 %argc.addr = alloca i32, align 4
 %argv.addr = alloca i8**, align 8
 %a = alloca i32, align 4
 store i32 0, i32* %retval
 store i32 %argc, i32* %argc.addr, align 4
 store i8** %argv, i8*** %argv.addr, align 8
 %0 = load i8*** %argv.addr, align 8
 %arrayidx = getelementptr inbounds i8** %0, i64 1
 %1 = load i8** %arrayidx, align 8
 %call = call i32 @atoi(i8* %1)
 store i32 %call, i32* %a, align 4
 %2 = load i32* %a, align 4
 %cmp = icmp eq i32 %2, 0
 br i1 %cmp, label %if.then, label %if.else

if.then:
 store i32 1, i32* %retval
 br label %return

if.else:
 store i32 10, i32* %retval
 br label %return

return:
 %3 = load i32* %retval
 ret i32 %3
```

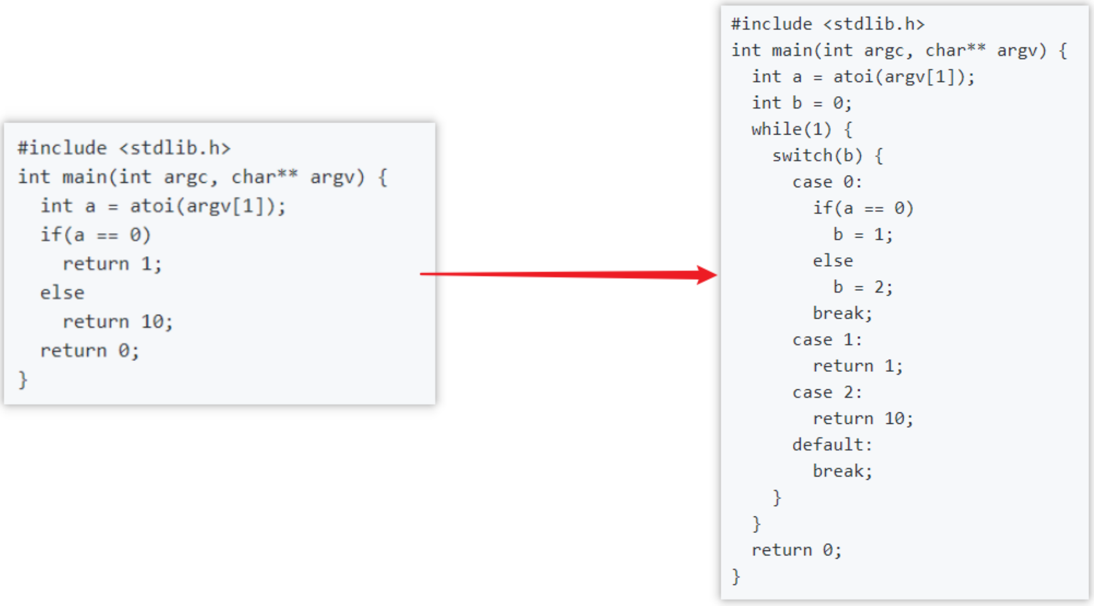
CFG for 'main' function

```
%0:
 %1 = load i32* @x
 %2 = load i32* @y
 %3 = sub i32 %1, 1
 %4 = mul i32 %1, %3
 %5 = urem i32 %4, 2
 %6 = icmp eq i32 %5, 0
 %7 = icmp slt i32 %2, 10
 %8 = or i1 %6, %7
 br i1 %8, label %9, label %33

%9:

 %10 = alloca i32, align 4
 %11 = alloca i32, align 4
 %12 = alloca i8**, align 8
 %13 = alloca i32, align 4
 store i32 0, i32* %10
 store i32 %argc, i32* %11, align 4
 store i8** %argv, i8*** %12, align 8
 %14 = load i8*** %12, align 8
 %15 = getelementptr inbounds i8** %14, i64 1
 %16 = load i8** %15, align 8
 %17 = call i32 @atoi(i8* %16)
 store i32 %17, i32* %13, align 4
 %18 = load i32* %13, align 4
 %19 = icmp eq i32 %18, 0
 %20 = load i32* @x
 %21 = load i32* @y
 %22 = sub i32 %20, 1
 %23 = mul i32 %20, %22
 %24 = urem i32 %23, 2
 %25 = icmp eq i32 %24, 0
 %26 = icmp slt i32 %21, 10
 %27 = or i1 %25, %26
 br i1 %27, label %28, label %33

%33:

 %34 = alloca i32, align 4
 %35 = alloca i32, align 4
 %36 = alloca i8**, align 8
 %37 = alloca i32, align 4
 store i32 0, i32* %34
 store i32 %argc, i32* %35, align 4
 store i8** %argv, i8*** %36, align 8
 %38 = load i8*** %36, align 8
 %39 = getelementptr inbounds i8** %38, i64 1
 %40 = load i8** %39, align 8
 %41 = call i32 @atoi(i8* %40)
 store i32 %41, i32* %37, align 4
 %42 = load i32* %37, align 4
 %43 = icmp eq i32 %42, 0
 br label %9

%28:

 br i1 %19, label %29, label %30

%29:

 store i32 1, i32* %10
 br label %31

%30:

 store i32 10, i32* %10
 br label %31

%31:

 %32 = load i32* %10
 ret i32 %32
```

CFG for 'main' function

control flow flattening

- all basic blocks are split and put into an infinite loop



```c
#include <stdlib.h>
int main(int argc, char** argv) {
  int a = atoi(argv[1]);
  if(a == 0)
    return 1;
  else
    return 10;
  return 0;
}
```

```c
#include <stdlib.h>
int main(int argc, char** argv) {
  int a = atoi(argv[1]);
  int b = 0;
  while(1) {
    switch(b) {
      case 0:
        if(a == 0)
          b = 1;
        else
          b = 2;
        break;
      case 1:
        return 1;
      case 2:
        return 10;
      default:
        break;
    }
  }
  return 0;
}
```

instruction substitution

- replacing standard binary operators (+ , – , & , | , ^) to make it more complicated
- only operators on integers are available
- does not add a lot of security
- bring diversity
- it can easily be removed by re-optimizing the generated code

- Addition
    - `a = b - (-c)`

```
%0 = load i32* %a, align 4
%1 = load i32* %b, align 4
%2 = sub i32 0, %1
%3 = sub nsw i32 %0, %2
```

- `a = -(-b + (-c))`

```
%0 = load i32* %a, align 4
%1 = load i32* %b, align 4
%2 = sub i32 0, %0
%3 = sub i32 0, %1
%4 = add i32 %2, %3
%5 = sub nsw i32 0, %4
```

- `r = rand (); a = b + r; a = a + c; a = a - r`

```
%0 = load i32* %a, align 4
%1 = load i32* %b, align 4
%2 = add i32 %0, 1107414009
%3 = add i32 %2, %1
%4 = sub nsw i32 %3, 1107414009
```

p.s.

- OLLVM的整体架构可扩展性特别好，针对IR进行Obfuscation，这样对于所有不同的编程语言都可以转化成IR，再用此平台进行混淆。
- OLLVM的代码混淆工具比较原始，现有很多de-obfuscation的方案。