2018-04-21

By -Sky3-

Index

- ~~Data-Oriented-Programming~~
- ASLR & …
- LLVM

ASLR

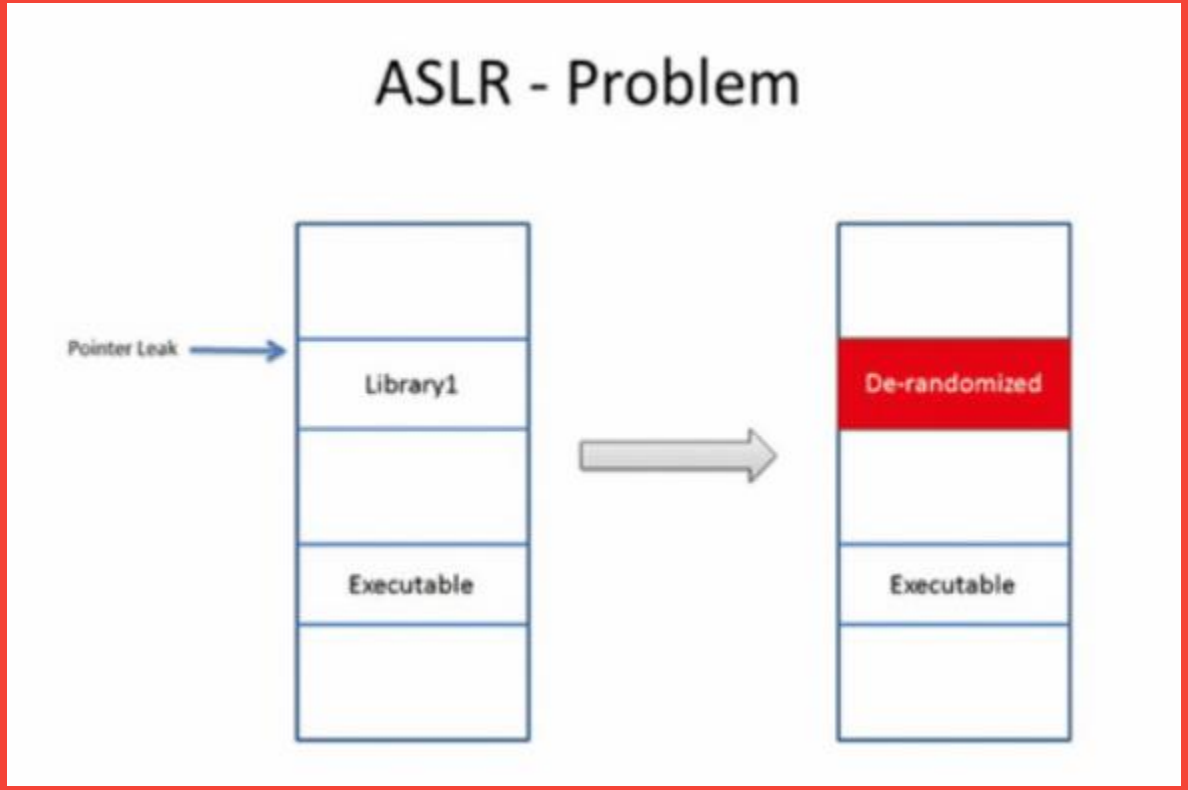
And more…

ASLR

- 一些攻击，比如ROP之类的代码复用攻击，会试图得到被攻击者的内存布局信息。这样就可以知道代码或者数据放在哪里，来定位并进行攻击。比如可以找到ROP里面的gadget。
- 而ASLR让这些内存区域随机分布，来提高攻击者成功难度，让他们只能通过猜测来进行不断试错的攻击

ASLR

- 在出现了某些漏洞，比如内存信息泄露的情况下，攻击者会得到部分内存信息，比如某些代码指针
- 传统的ASLR只能随机化整个segment，比如栈、堆、或者代码区。这时攻击者可以通过泄露的地址信息来推导别的信息



ASLR

- 如何改进?

- 防止内存泄露
- 增强ASLR

ASLR

- 增强ASLR的粒度：

- ASLP在函数级进行随机化
- Binary stirring在basic block级进行随机化
- ILR和IPR在指令级

ASLR

- 随机化的方式可以改进：
- Oxymoron解决了库函数随机化的重复问题：假如每个进程的库函数都经过的ASLR，会导致内存开销很大，该文用了X86的分段巧妙地解决了这个问题

ASLR

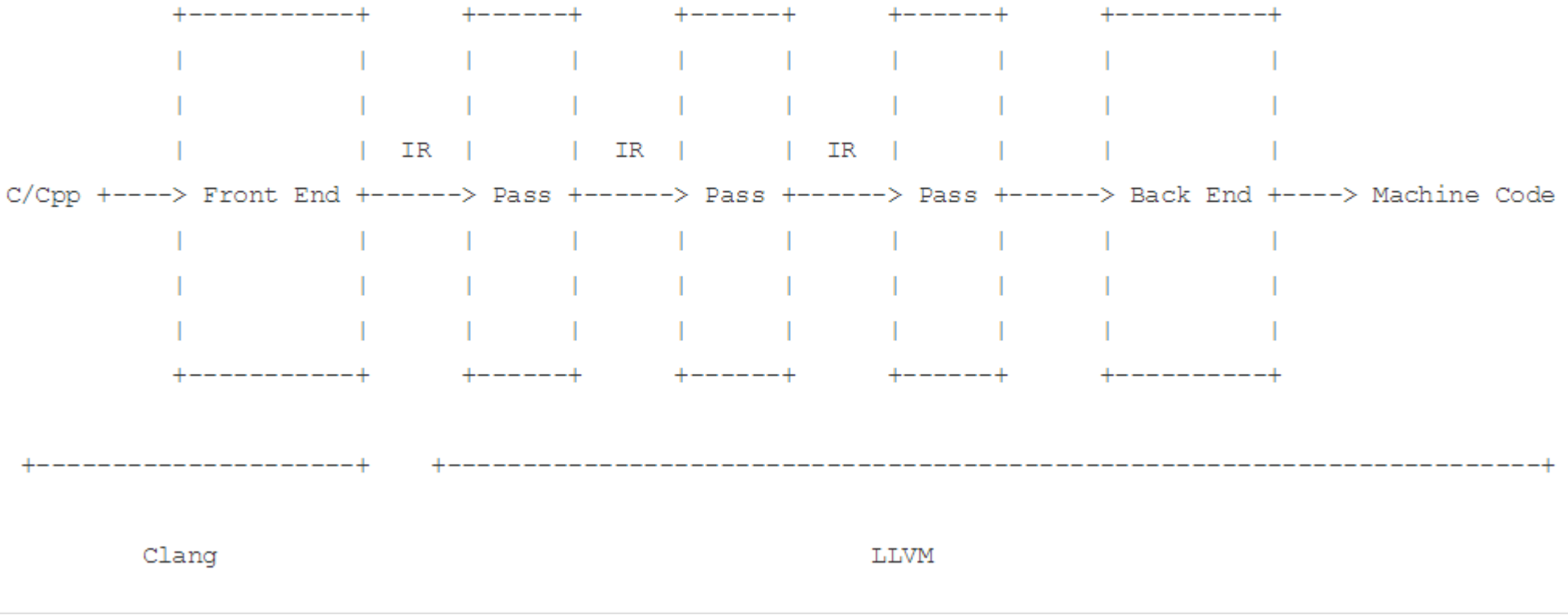
- 随机化的时间可以改进：

- TASR：Timely Address Space Randomization
  - Timely Rerandomization for Mitigating Memory Disclosures
  - 通过在每次产生输出时对进程的内存布局应用重随机化，这种方法会使得利用泄露信息的攻击者在劫持控制流的时候失效。
  - Paper原型运行于C代码，重编译了程序且使用了一组增强信息来跟踪指针位置。

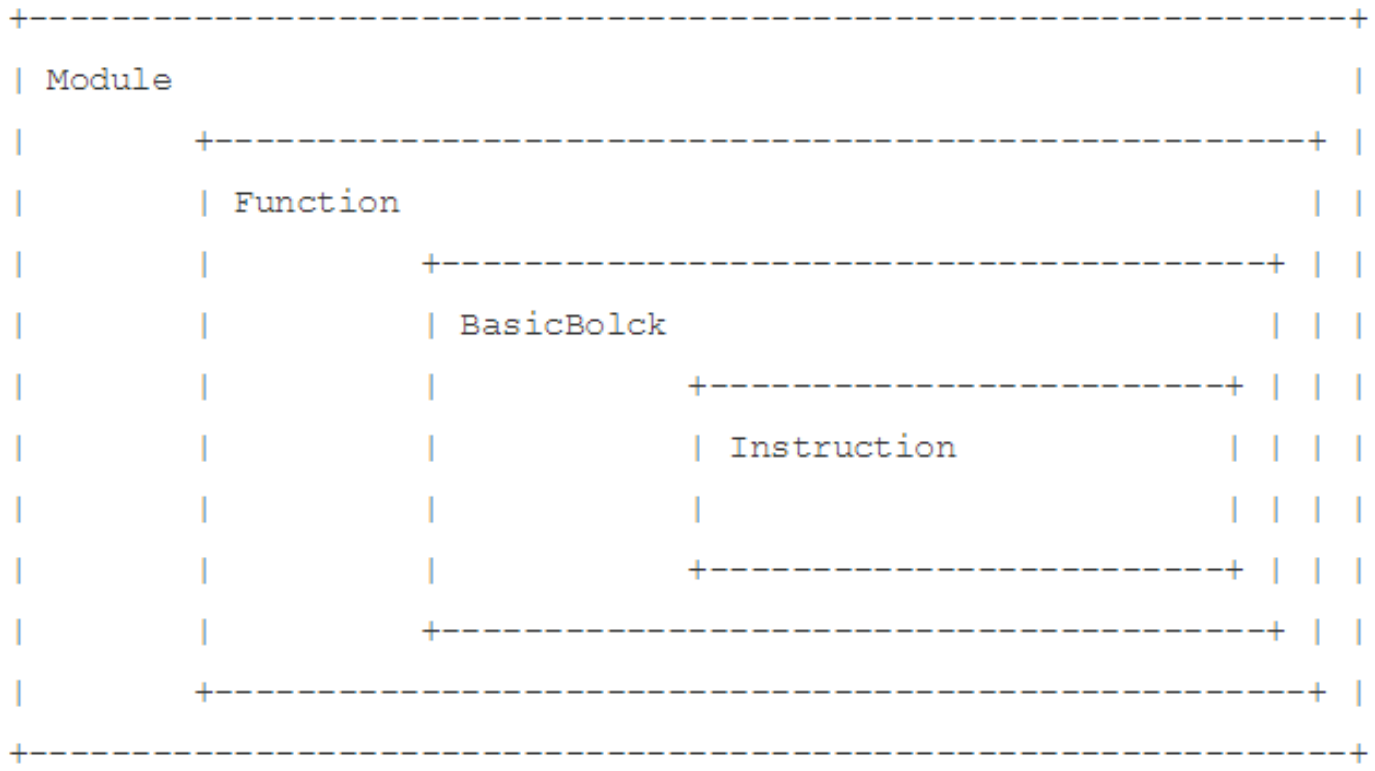
LLVM

Things you don’t know

LLVM

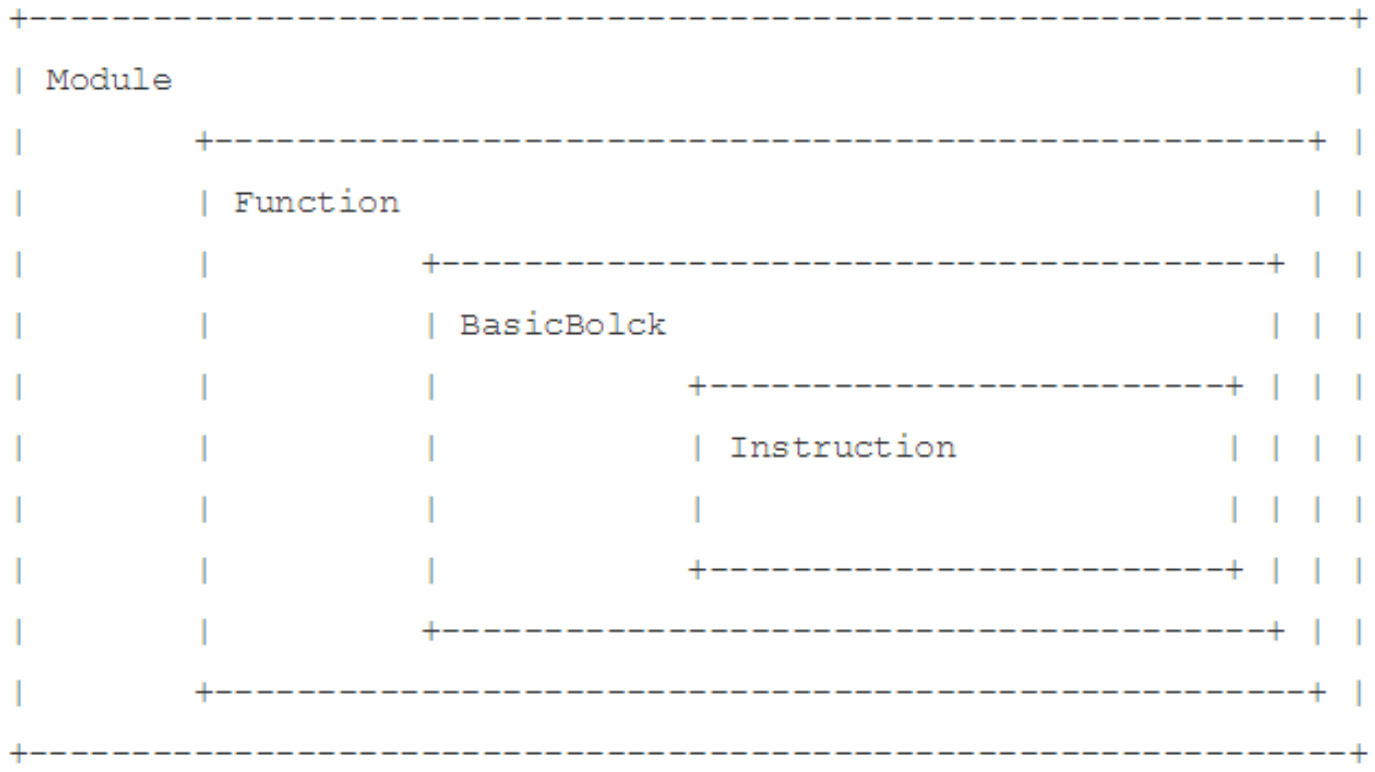

IR
IR
IR
C/Cpp
Front End
Pass
Pass
Pass
Back End
Machine Code
Clang
LLVM

LLVM



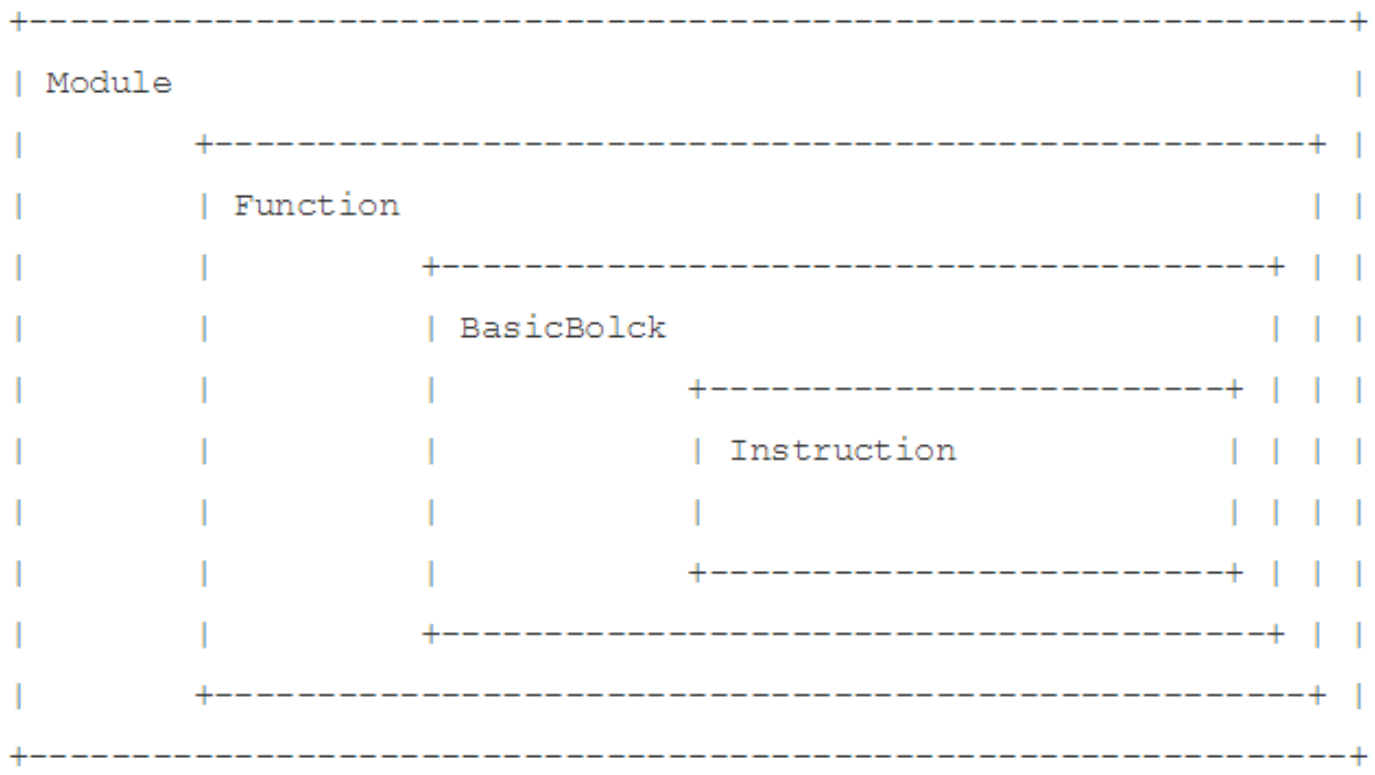
模块包含了函数，函数又包含了代码块，后者又是由指令组成。除了模块以外，所有结构都是从值产生而来的。

LLVM



- 粗略地说，模块表示了一个源文件，或者学术一点讲叫翻译单元。其他所有东西都被包含在模块之中。
- 最值得注意的是，模块容纳了函数，顾名思义，后者就是一段段被命名的可执行代码。（在C++中，函数function和方法method都相应于LLVM中的函数。）

LLVM



- 除了声明名字和参数之外，函数主要会做为代码块的容器。代码块和它在编译器中的概念差不多，不过目前我们把它看做是一段连续的指令。
- 而说到指令，就是一条单独的代码命令。这一种抽象基本上和RISC机器码是类似的：比如一个指令可能是一次整数加法，可能是一次浮点数除法，也可能是向内存写入。

LLVM

```
; ModuleID = 'hello.c'
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

@.str = private unnamed_addr constant [3 x i8] c"%s\00", align 1
@.str.1 = private unnamed_addr constant [13 x i8] c"Hello, world\00", align 1

; Function Attrs: nounwind uwtable
define i32 @main() #0 {
%1 = alloca i32, align 4
store i32 0, i32* %1, align 4
%2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]*
@.str, i32 0, i32 0), i8* getelementptr inbounds ([13 x i8], [13 x i8]* @.str.1,
i32 0, i32 0))
ret i32 0
}

declare i32 @printf(i8*, ...) #1
```

LLVM

```
attributes #0 = { nounwind uwtable "disable-tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false" "no-nans-fp-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2" "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #1 = { "disable-tail-calls"="false" "less-precise-fpmad"="false" "no-frame-pointer-elim"="true" "no-frame-pointer-elim-non-leaf" "no-infs-fp-math"="false" "no-nans-fp-math"="false" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "target-features"="+fxsr,+mmx,+sse,+sse2" "unsafe-fp-math"="false" "use-soft-float"="false" }

!llvm.ident = !{!0}

!0 = !{!"clang version 3.8.0-2ubuntu4 (tags/RELEASE_380/final)"}
```

LLVM

Thanks