软件安全与漏洞分析

3.5 返回导向编程的发展（4）

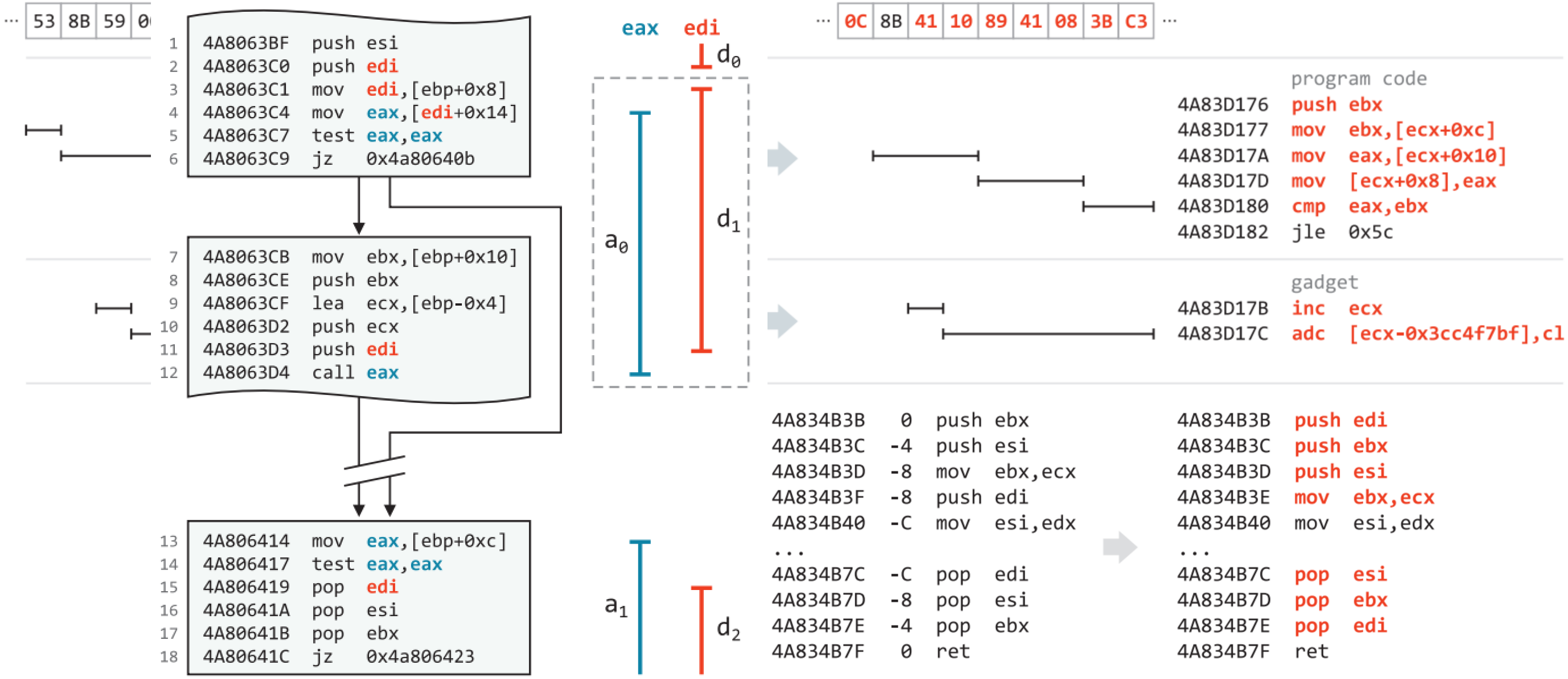
Previously in Software Security

- CFI的弱点及绕过
  - CFI的根本性弱点
  - 针对CFI的改进型返回导向编程编码方法
  - 返回导向编程的变种：数据导向编程

返回导向编程的发展（4）

- 本节主题 – 地址空间随机化（**A**ddress **S**pace **L**ayout **R**andomization）及其破解
  - 粗/粒度的ASLR方法
  - ASLR的无效化：Just-In-Time代码重用

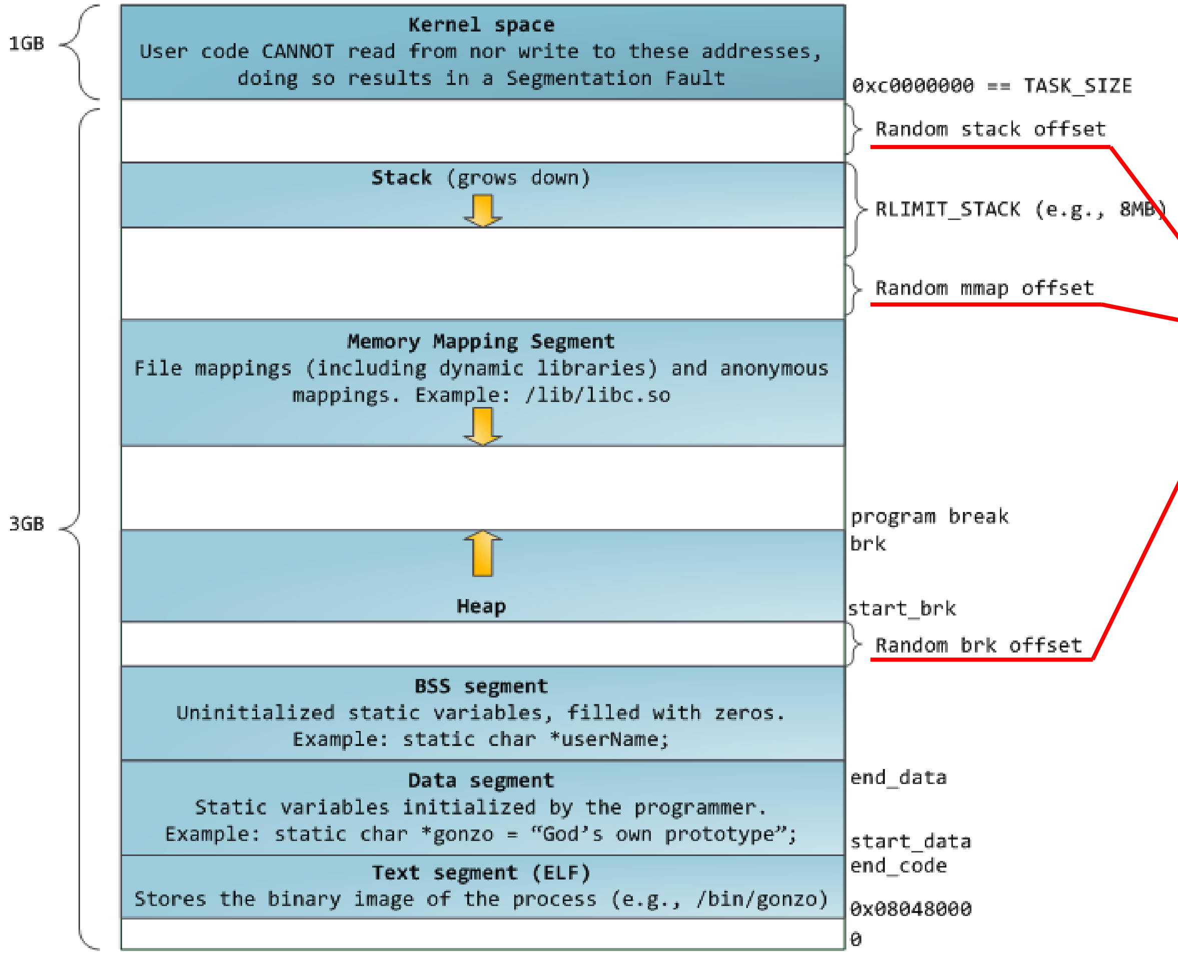
回顾：Smashing the gadgets



回顾：Smashing the gadgets

- 核心思路：阻止x86代码中出现意外的gadget
- 但是：ROP without Ret等新型ROP编码证明**消除gadget**过于困难
- ALSR对上述思想的发展：**令gadget的位置变得无法确定**

ASLR的基本思想



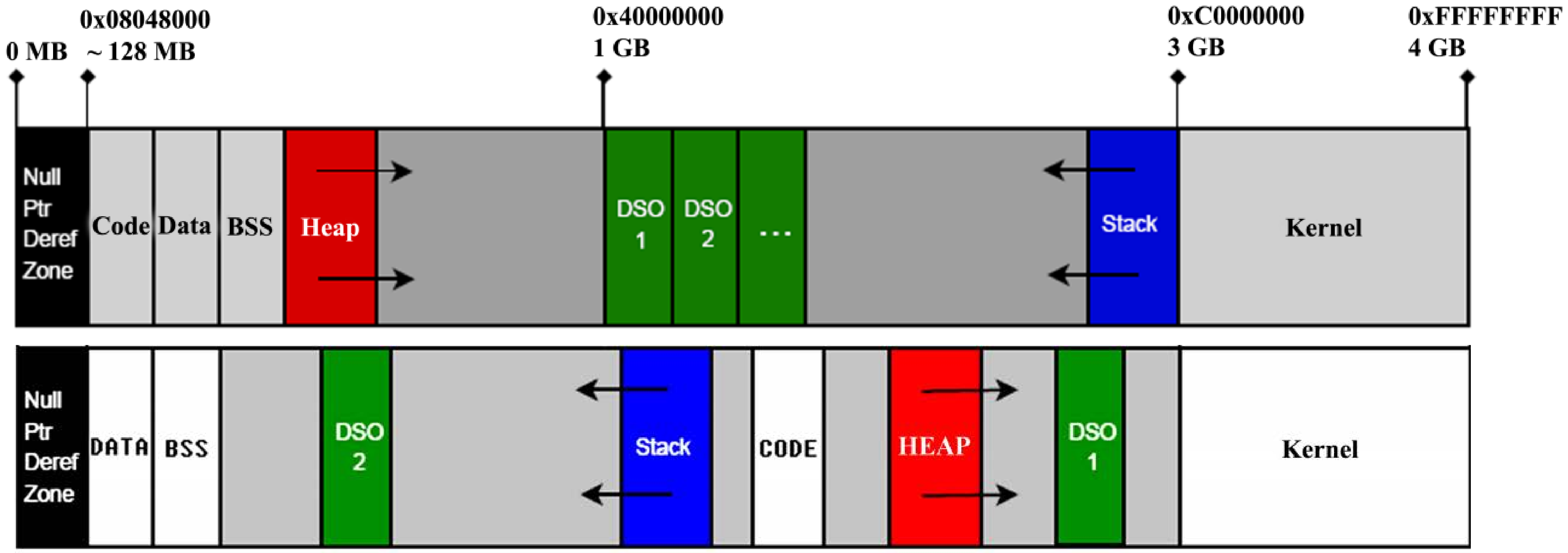
地址空间随机化
(ASLR)

粗粒度

已经广泛实装
(windows/linux等)

ASLR的发展

- 粗粒度ASLR的改进：偏移量+内存区段的位置置换

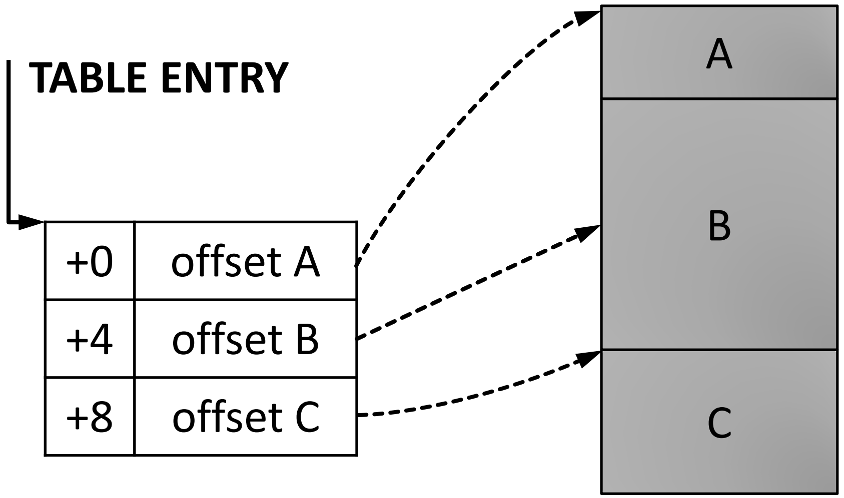


ASLR的发展

- 细粒度的ASLR
  - 区段间 → 区段内
  - 改变代码文本的具体内容
- 以switch-case结构为例

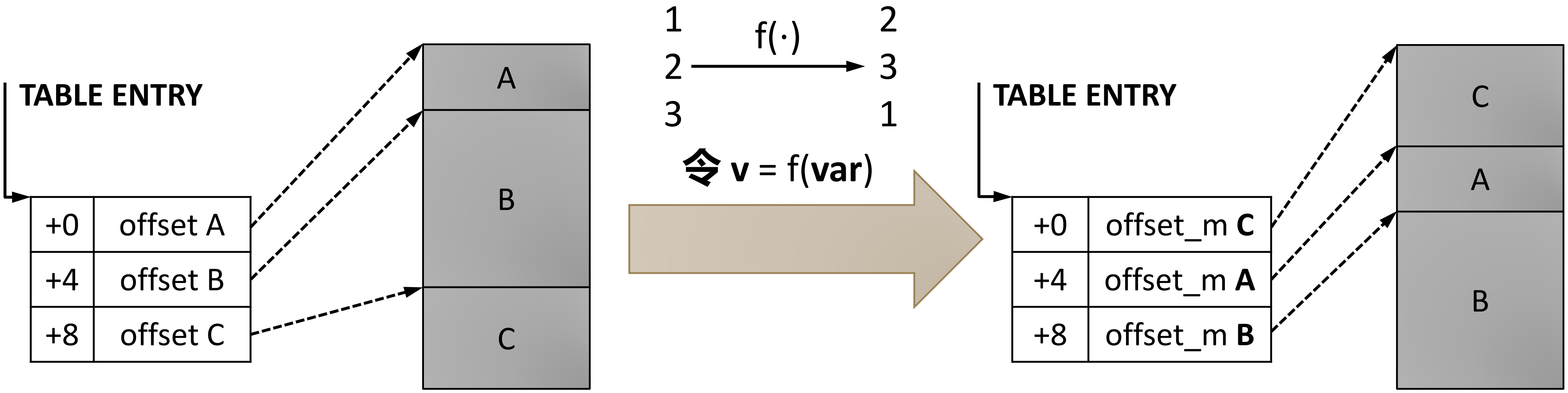
```
switch(var){
    case 1: {code A; break;}
    case 2: {code B; break;}
    case 3: {code C; break;}
}
```

TABLE ENTRY ⟶ **eax**;
var ⟶ **ebx**;
eax + (**ebx** -1)*4 ⟶ **eax**;
jmp **eax**;



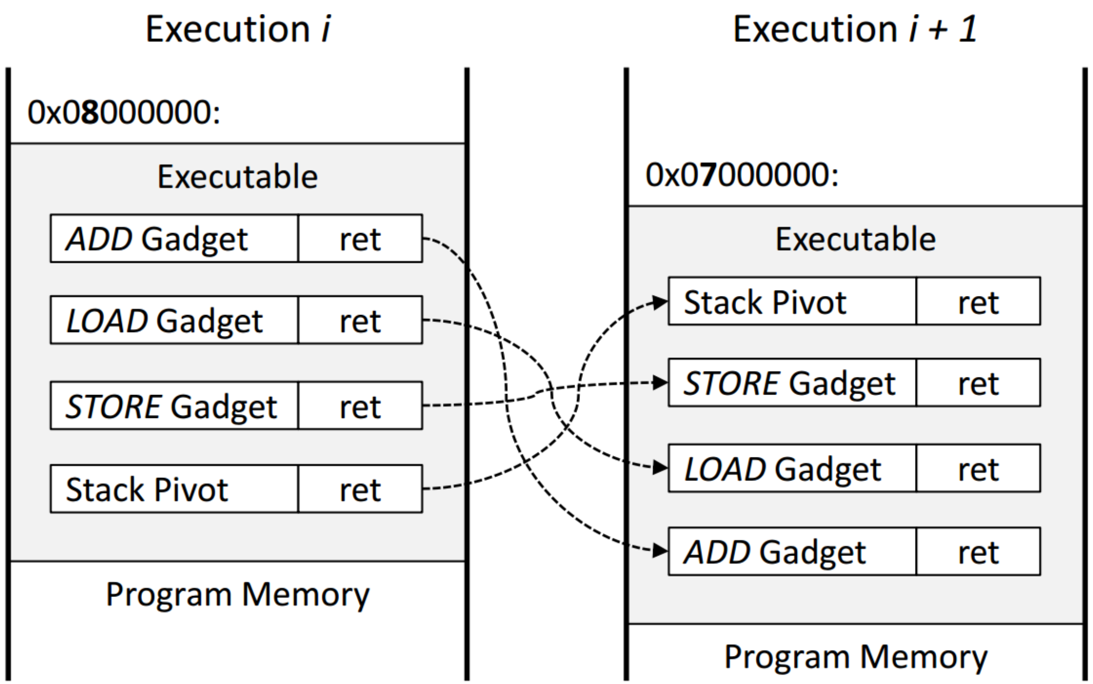
ASLR的发展

- 细粒度的ASLR



ASLR的发展

- 细粒度的ASLR

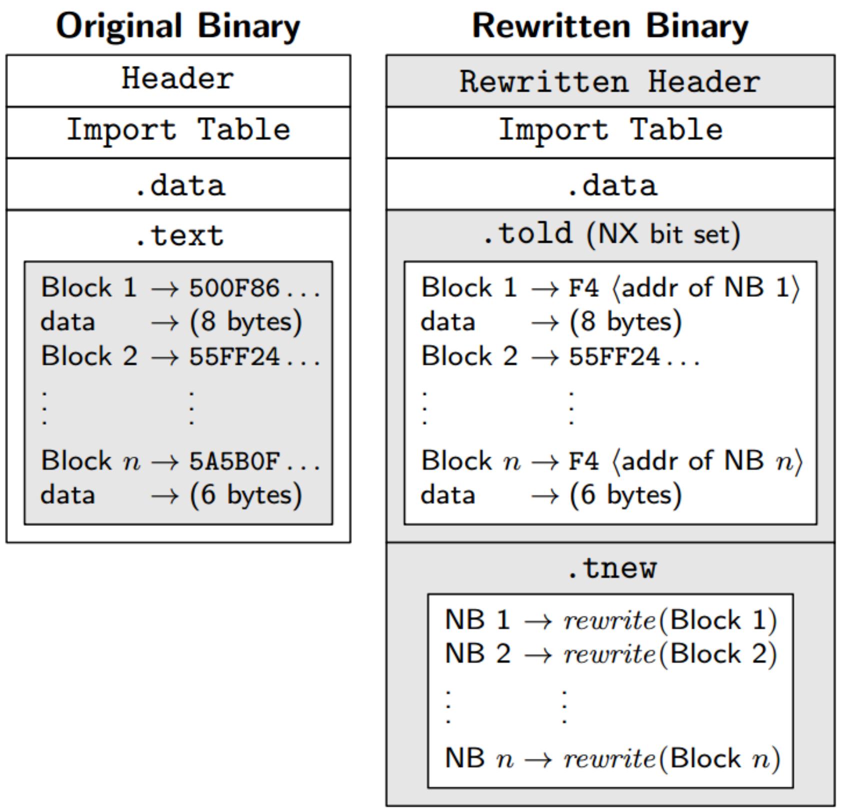


ASLR的发展

- 细粒度ASLR的实施
  - 程序在加载时自我随机化
  - 通过虚拟机进行动态随机化
  - 操作系统的随机化

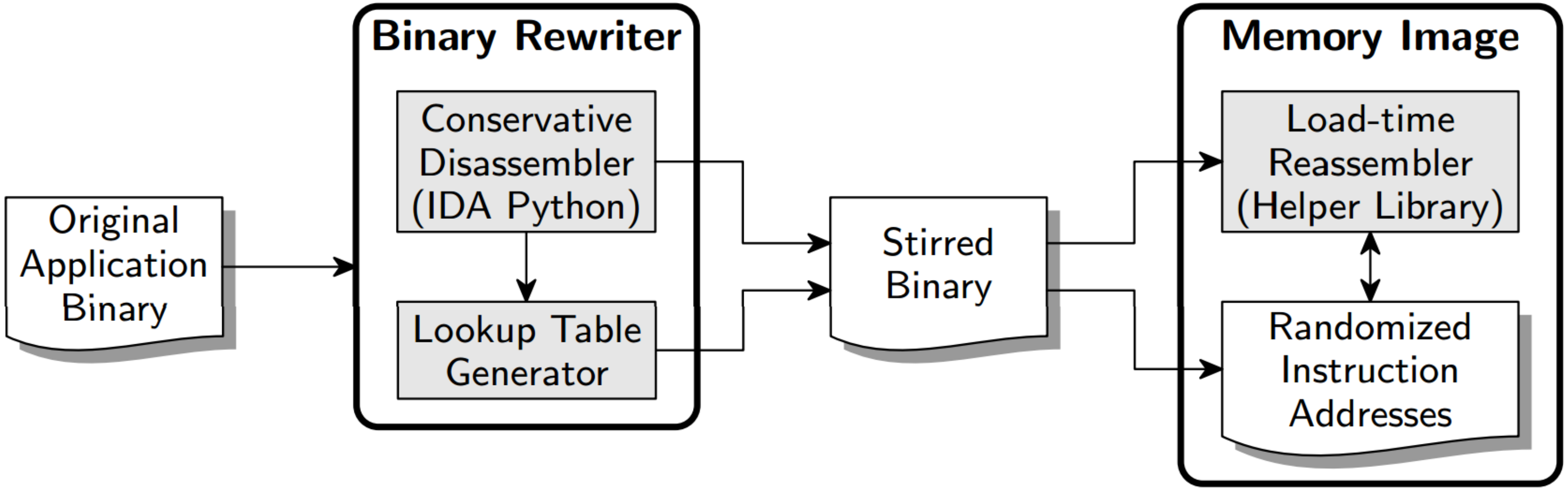
ASLR的发展

- 程序自我随机化：binary stirring (Self-Transforming Instruction Relocation)



ASLR的发展

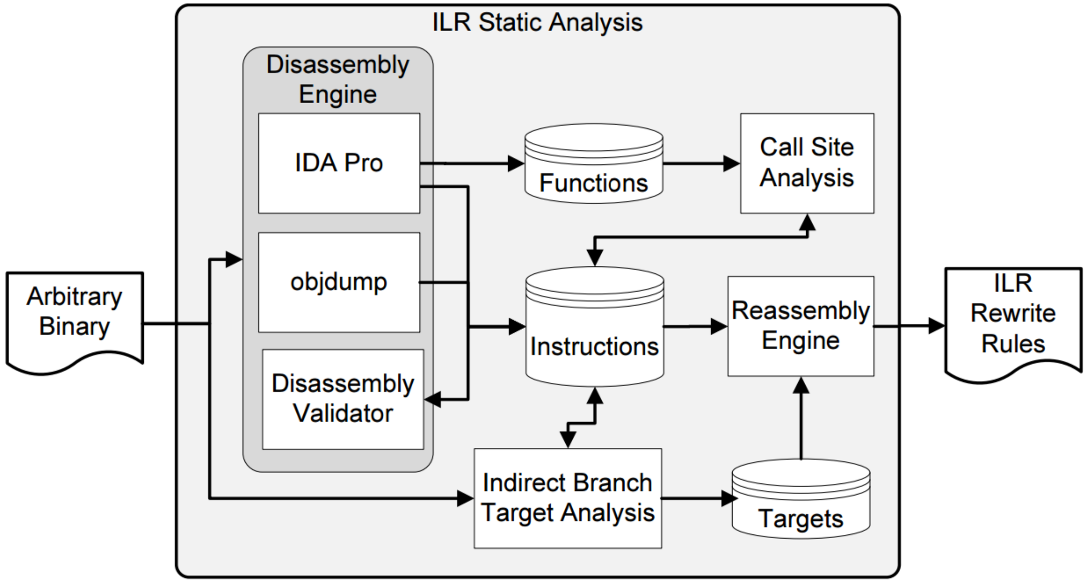
- 程序自我随机化：binary stirring (Self-Transforming Instruction Relocation)



- 参考文献：Wartell R, Mohan V, Hamlen K W, et al. Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code[C]//Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012: 157-168.

ASLR的发展

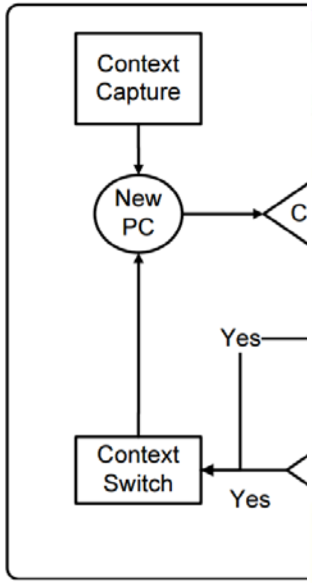
- 虚拟机动态随机化：Instruction Location Randomization (ILR)

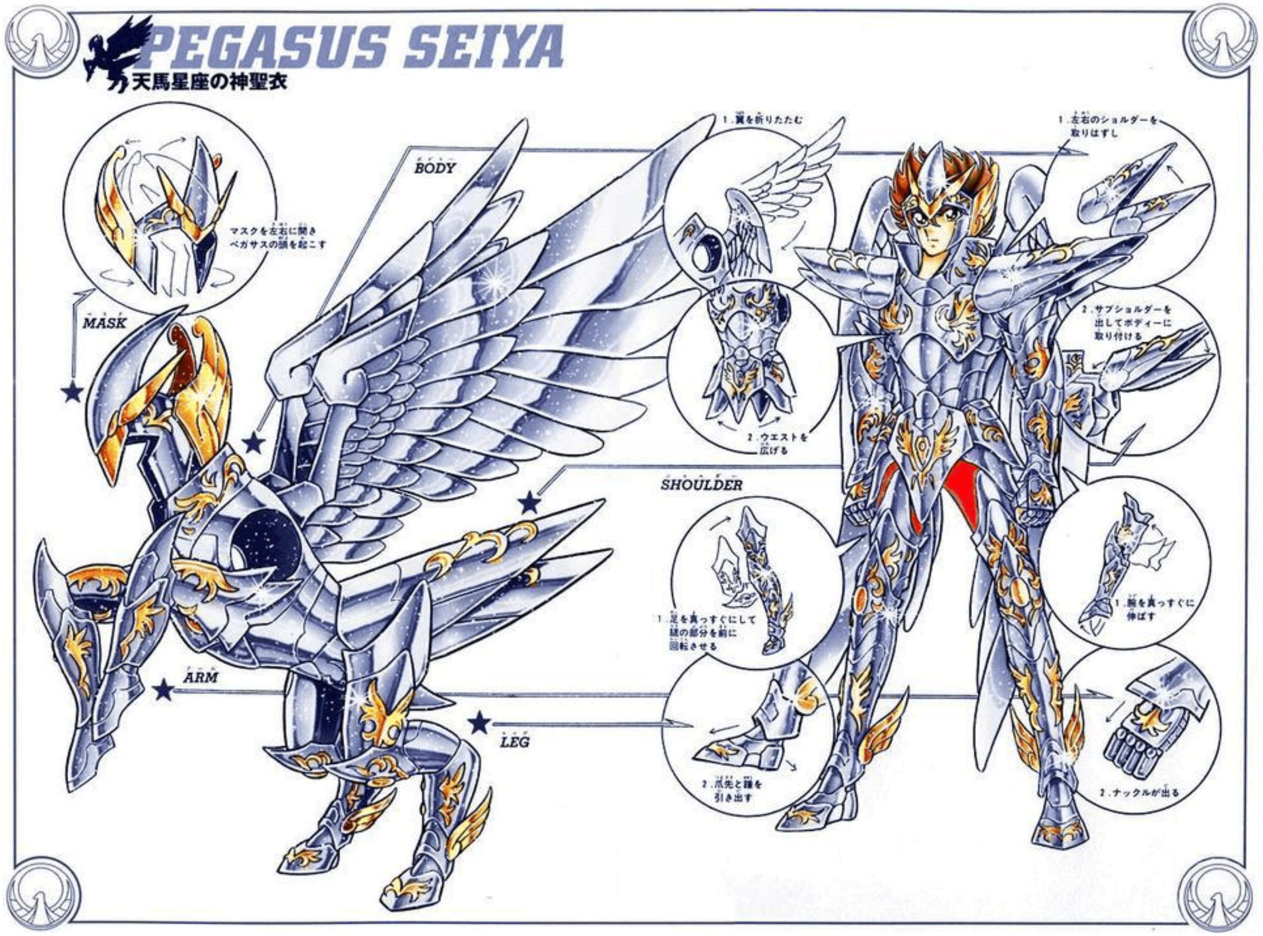



```
39bc ** cmp eax, #24
39bd -> d27e
d27e ** jeq a96b
d27f -> cb20
cb20 ** call 5f32
cb21 -> 67f3
67f3 ** mov [0x8000], 0
67f4 -> a96b
224a ** add eax, #1
224b -> 67f3
a96b ** ret
```

ASLR的发展

- 虚拟机动态随机



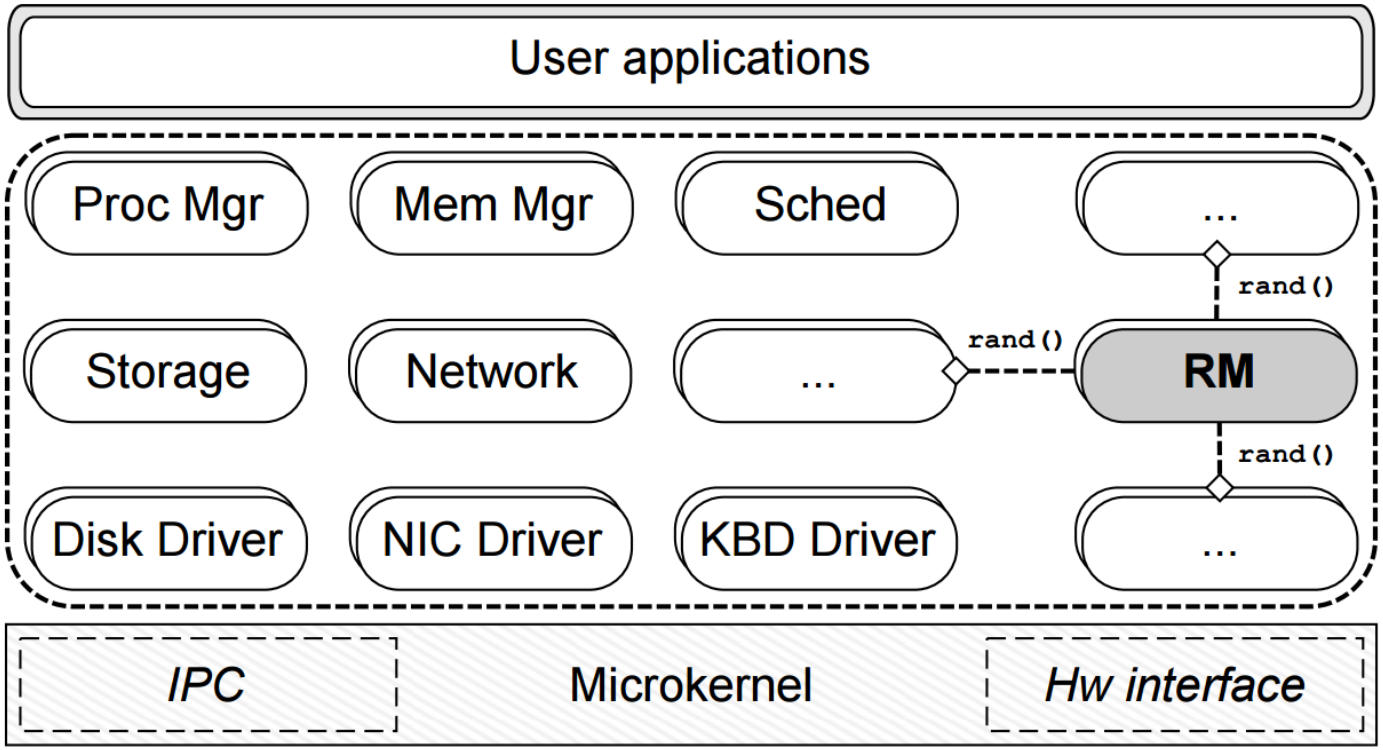



```
x, #24
6b
f32
x8000], 0
x, #1
```

- 参考文献：His dgets go?[C]//Securit 2012: 571-585.

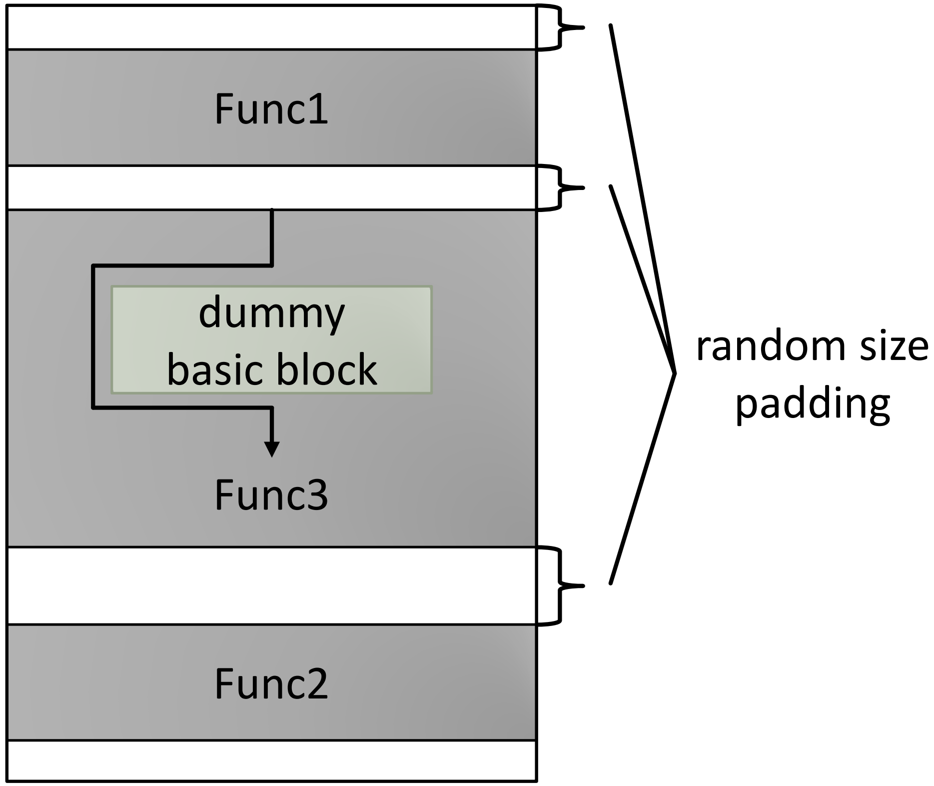
ASLR的发展

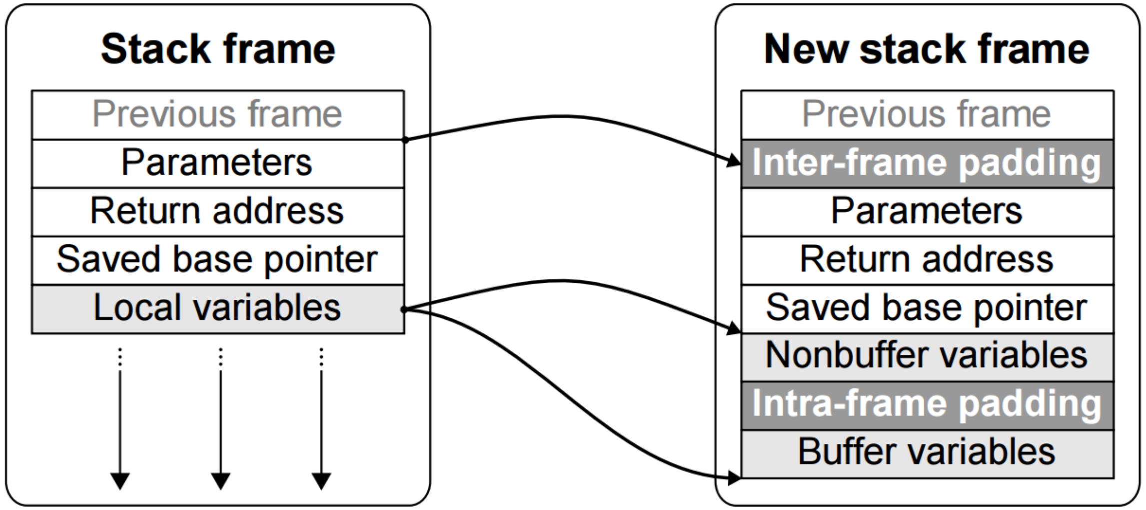
- 操作系统的链接时随机化/执行时重新随机化



ASLR的发展

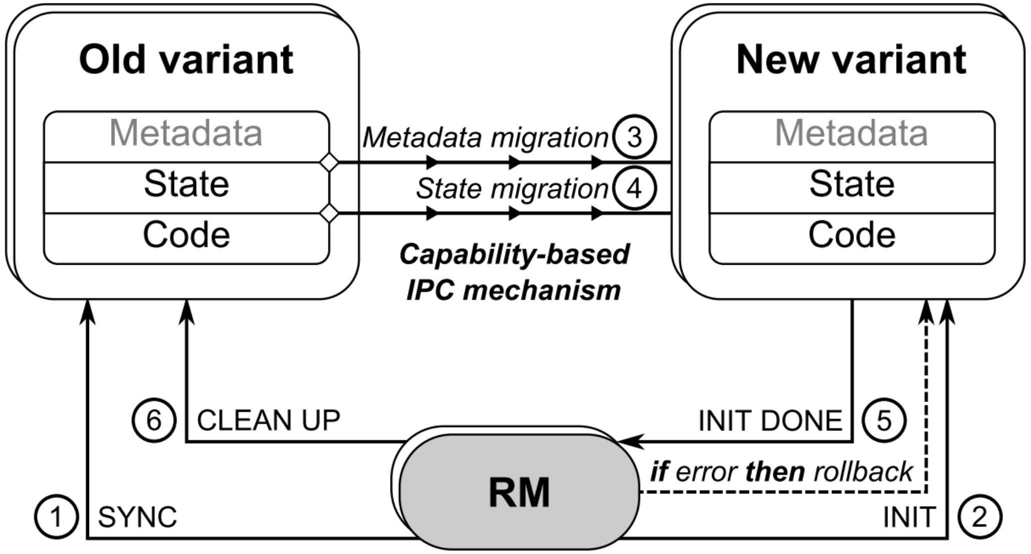
- 操作系统的链接时随机化/执行时重新随机化





ASLR的发展

- 操作系统的链接时随机化/执行时重新随机化



- 参考文献：Giuffrida C, Kuijsten A, Tanenbaum A S. Enhanced Operating System Security Through Efficient and Fine-grained Address Space Randomization[C]//USENIX Security Symposium. 2012: 475-490.

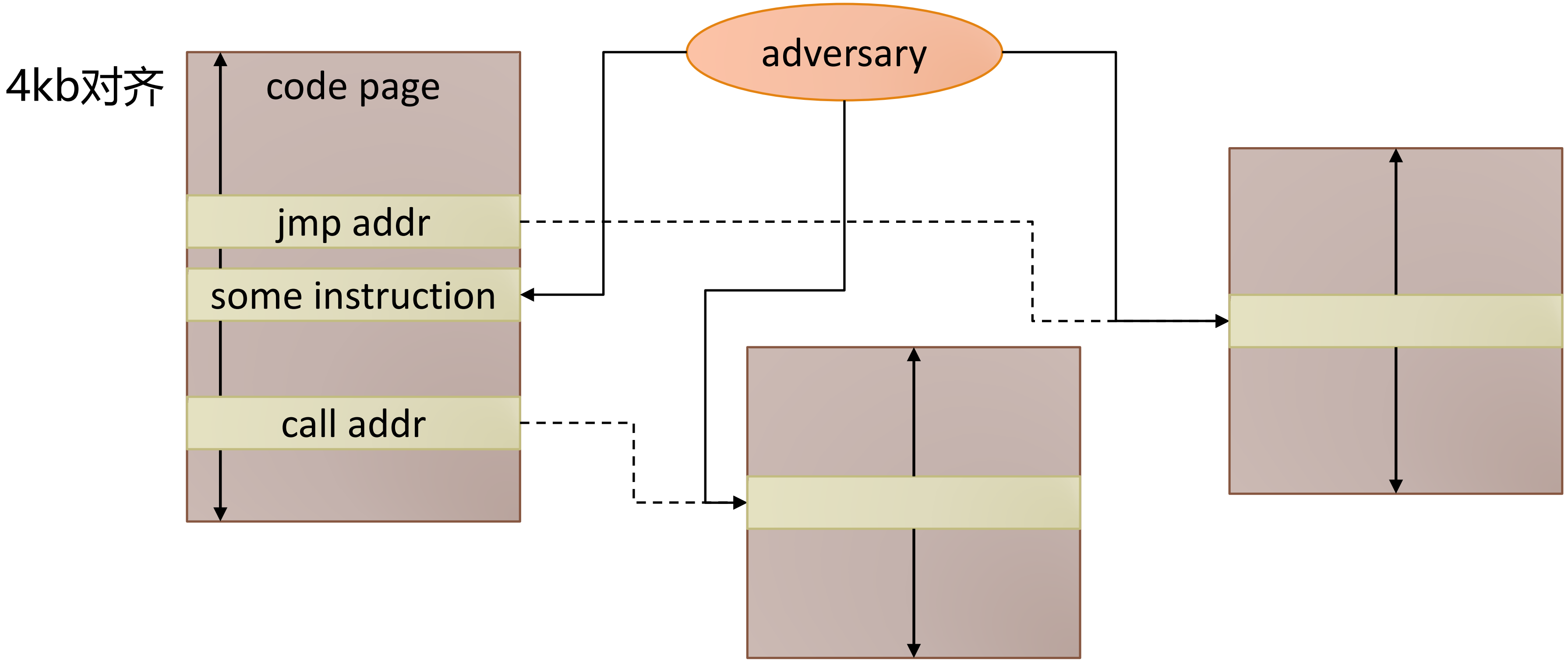
ASLR的无效化

- ASLR假设的攻击模型
  - Case 1：攻击者无法披露目标程序的内存空间
  - Case 2：攻击者可以实施内存空间披露，但只能获得一个代码指针

如果攻击者真的寻获了一个可以远程利用的内存披露漏洞，为何不反复利用之，以最大化漏洞价值?

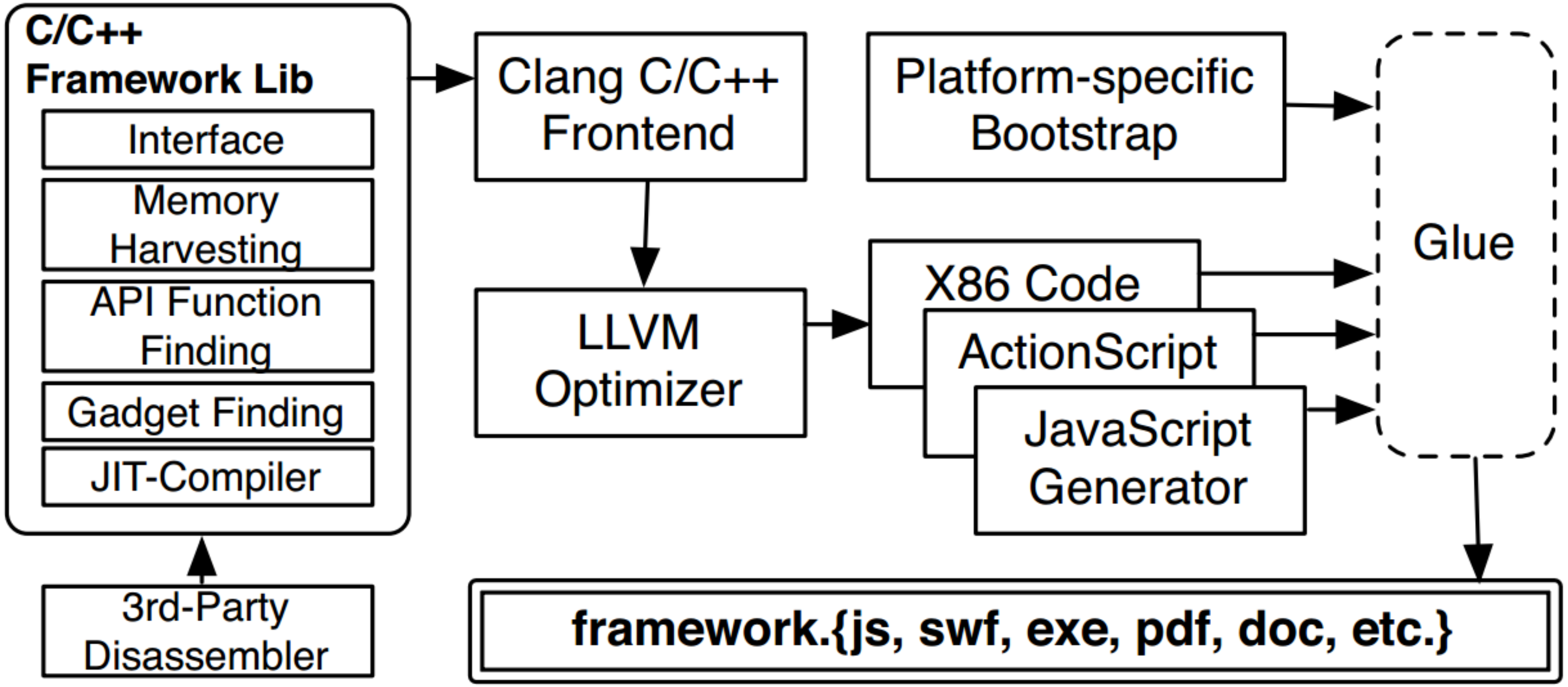
  - Limitation：对于任何ASLR，程序在执行时不再改变其内存空间结构

ASLR的无效化

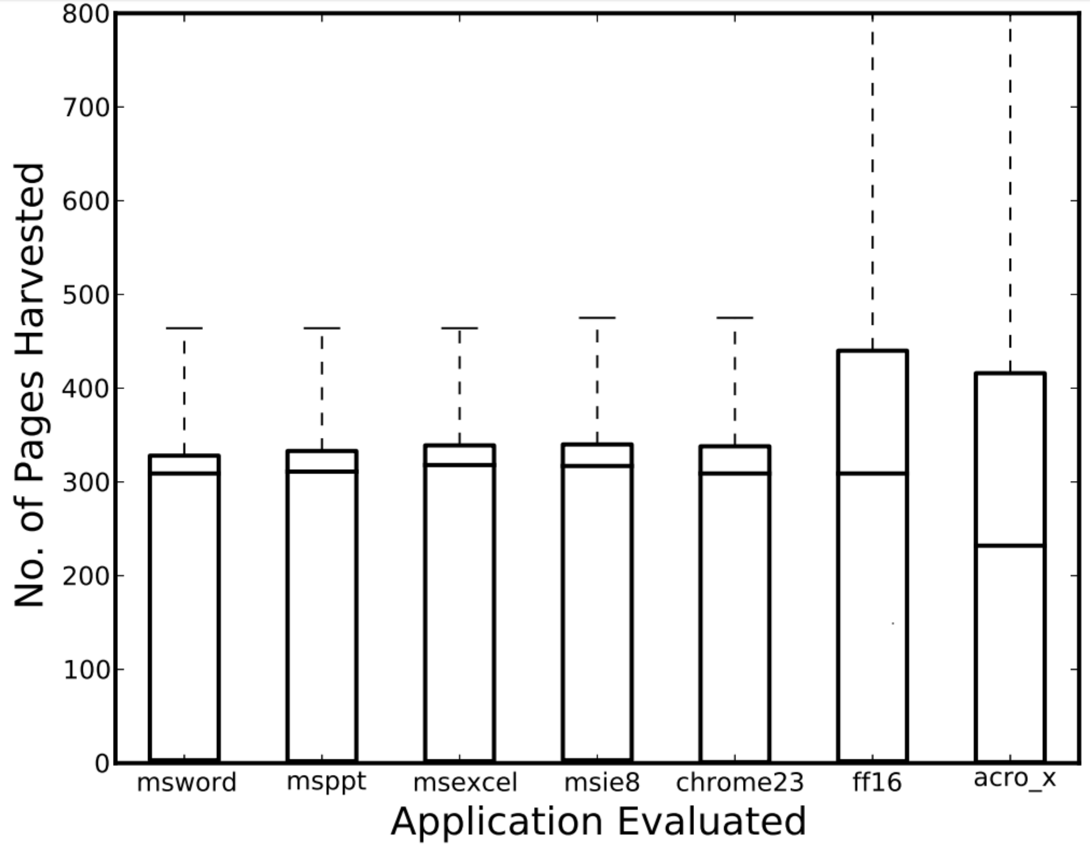


“页面收割” （page harvest），最终暴露进程中的全部代码、API接口等可利用资源

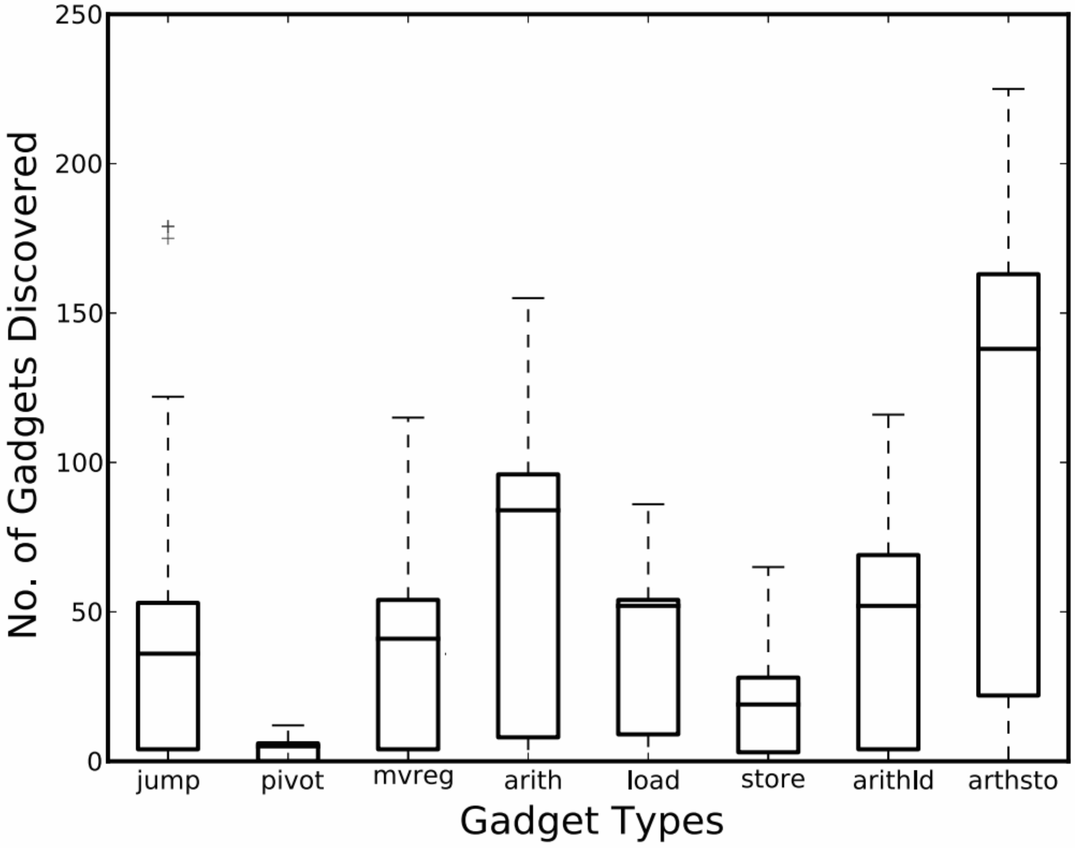
ASLR的无效化


C/C++
Framework Lib
Interface
Memory
Harvesting
API Function
Finding
Gadget Finding
JIT-Compiler
3rd-Party
Disassembler
Clang C/C++
Frontend
LLVM
Optimizer
Platform-specific
Bootstrap
X86 Code
ActionScript
JavaScript
Generator
Glue
framework.{js, swf, exe, pdf, doc, etc.}

ASLR的无效化

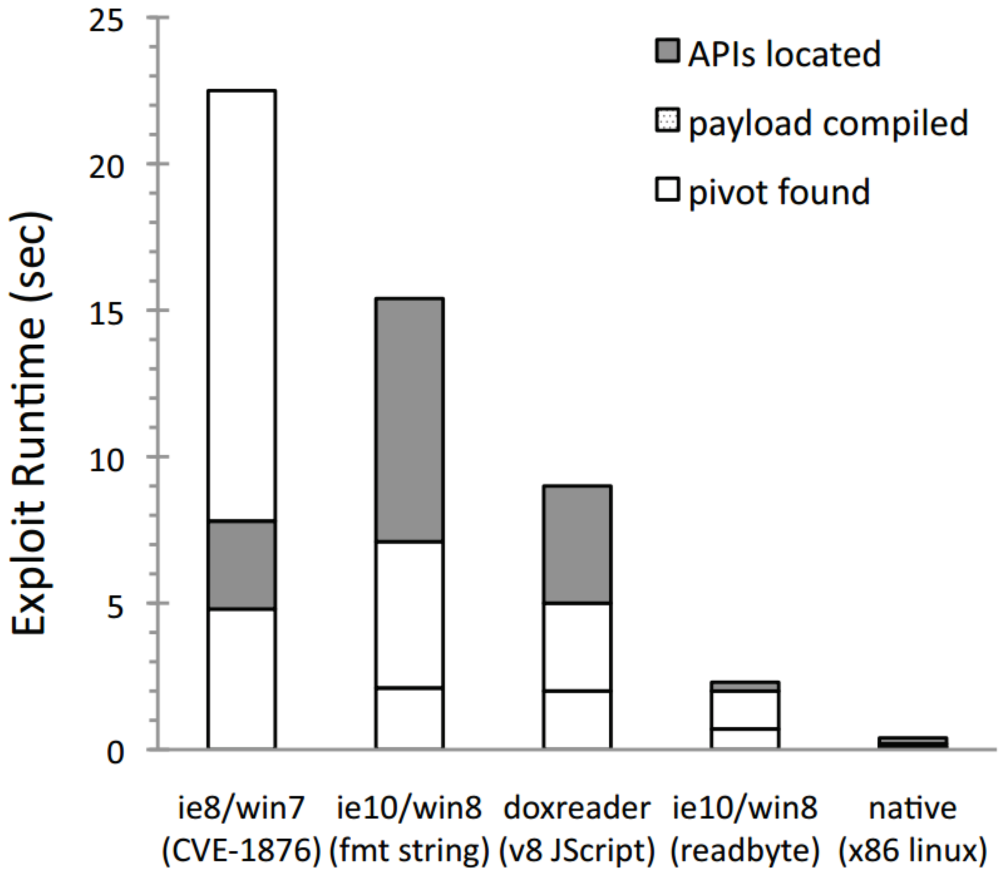


800
700
600
500
400
300
200
100
0
No. of Pages Harvested
msword
msppt
msexcel
msie8
chrome23
ff16
acro_x
Application Evaluated



250
200
150
100
50
0
No. of Gadgets Discovered
jump
pivot
mvreg
arith
load
store
arithld
arthsto
Gadget Types

ASLR的无效化



Exploit Runtime (sec)
25
20
15
10
5
0
APIs located
payload compiled
pivot found
ie8/win7 (CVE-1876)
ie10/win8 (fmt string)
doxreader (v8 JScript)
ie10/win8 (readbyte)
native (x86 linux)

ASLR的无效化

- 参考文献：Snow K Z, Monrose F, Davi L, et al. Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization[C]//Security and Privacy (SP), 2013 IEEE Symposium on. IEEE, 2013: 574-588.
- 扩展阅读：详细学习论文IV-E章节所述just-in-time code reuse的POC实现
- 思考：**目前看来，粗粒度CFI和细粒度ASLR的单独应用均没有令人满意的效果，但如果两者联合应用时，是否会有足够的防御强度？**

What’s next?

- 概述：其他软件/操作系统防御策略和技术
- 软件自我保护技术