软件安全与漏洞分析

3.1 代码注入 --- 返回导向编程

Previously in Software Security

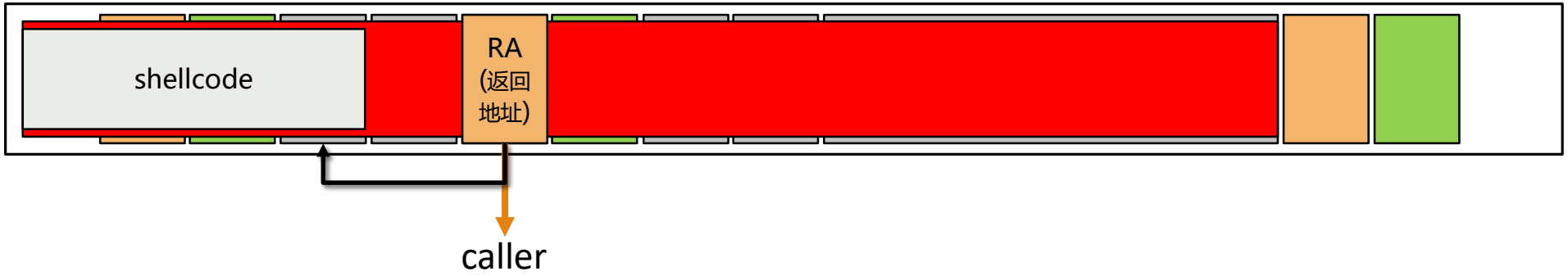
- 缓冲区溢出、格式化字符串漏洞等典型软件漏洞
  - 原理
  - 利用方式
- SQL注入、跨站脚本等典型的Web应用漏洞

代码注入

- 本节主题 -- 1.代码注入的基本原理回顾
- 本节主题 -- 2.返回导向编程
  - 动机
  - 前身（Ret2Libc）
  - 原理
  - “图灵完全”

回顾：代码注入的基本形式

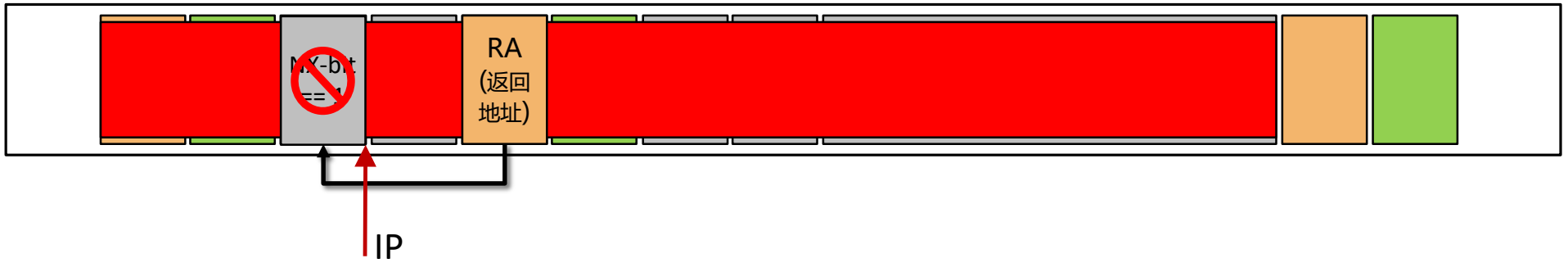
- 基于栈溢出的典型代码注入攻击



  - 通过栈溢出，劫持位于栈帧中的函数返回地址
  - 直接将被劫持的返回地址指向被溢出的栈区
  - 溢出过程在被指向的位置上写入shellcode

回顾：针对代码注入的防护机制

- 数据执行保护（**D**ata **E**xecution **P**revention）
  - 一些别名：W⊕X，NX-bit
  - 机制：对内存页面增加一个标识bit，使之**要么可改写，要么可执行**



回顾：针对代码注入的防护机制

- 经典代码注入的核心思想
  - 利用逻辑异常，在程序**数据中混入代码**
  - 劫持控制流，**使得指令指针从数据段读取指令**
  - 所需弱点：读取指令时，CPU无法区分目标内存区域的性质（代码？数据？）
- 针对上述特点，DEP的防御思路：
  - 引入新的硬件安全属性，支持CPU在执行时区分代码和数据区域
  - **但是**：若不试图执行数据区内容，则不能发现/阻止溢出，**也无法防范溢出的其他后果**

目标：绕过DEP

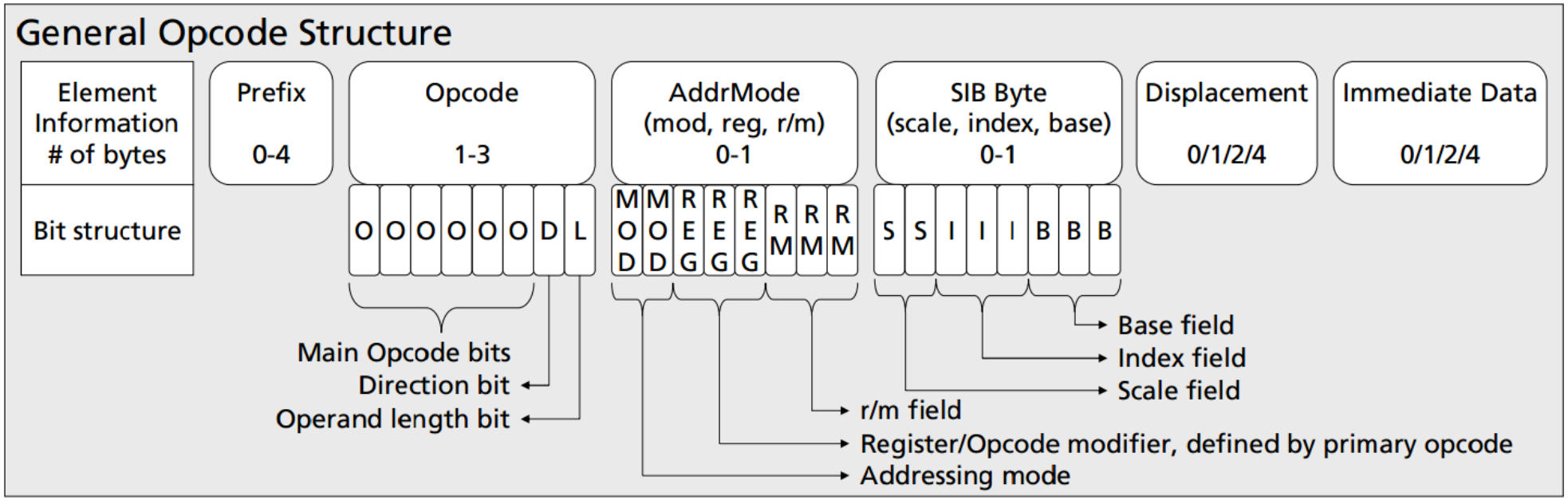
- DEP客观存在，注入类攻击如何下手？--- **代码重用（code reuse）攻击**
  - 异常数据的注入仍然可以实施
  - 既然代码无法直接注入，那么就**利用进程空间中已经存在的代码**
  - 不管所利用代码原本的用途为何，**通过注入的异常数据控制其为攻击者服务**
- 最先出现的此类手段：return-to-libc攻击

Top of stack	EBP	EIP	Dummy return addr	address of /bin/sh string
AAAAAAAAA	AAAA	Addr of system function (0xb7e9ef10)	DUMM	address of /bin/sh string

目标：绕过DEP

- 新的利用点：复杂指令集（complex instruction set computing，CISC）
  - **指令变长**（目的：以最短的代码长度压缩最多的指令内容）
  - CPU串行地读取指令，**依据上下文区分操作码/操作数等成分**
  - 典型：x86指令构架

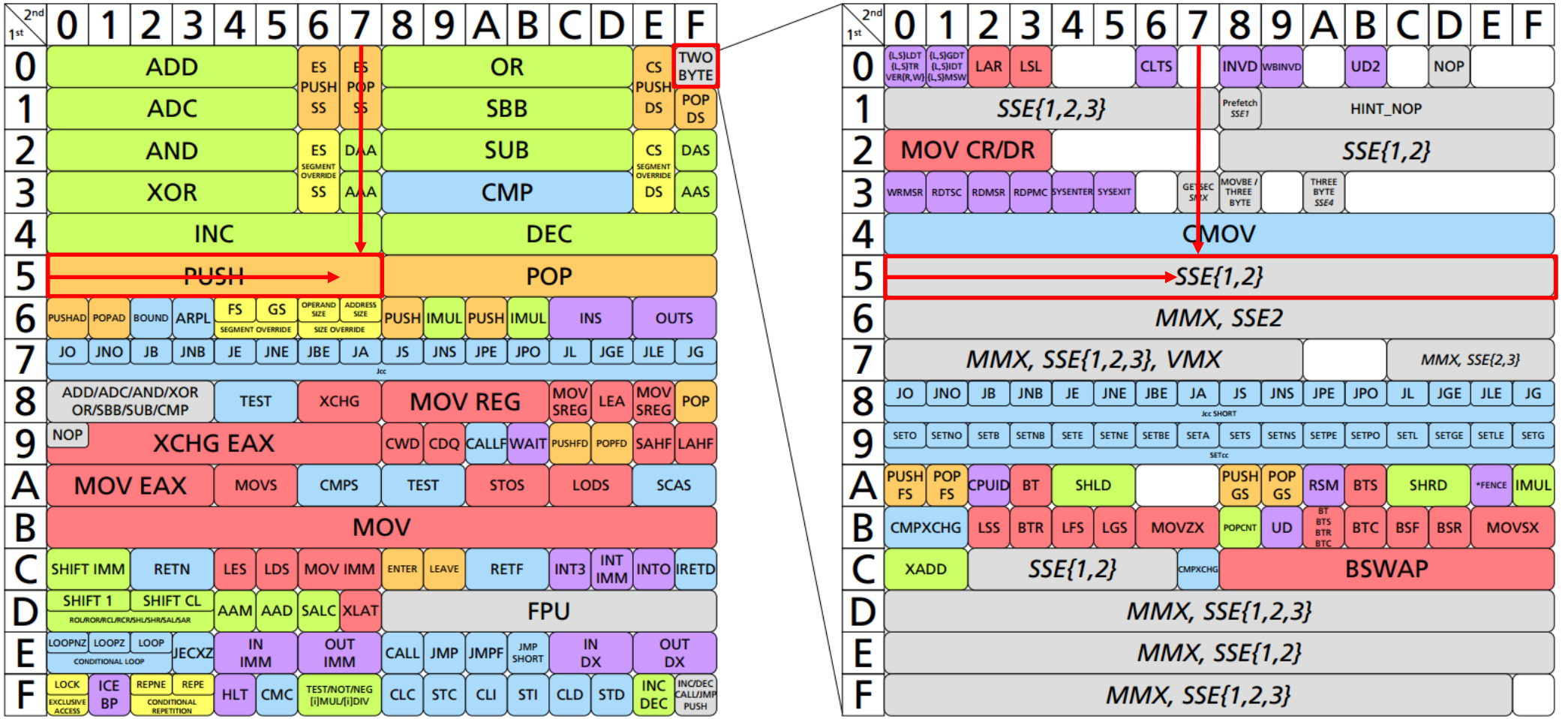
目标：绕过DEP


General Opcode Structure
Element Information # of bytes
Bit structure
Prefix
0-4
Opcode
1-3
O O O O O O D L
AddrMode (mod, reg, r/m)
0-1
MOD MOD REG REG REG RM RM RM
SIB Byte (scale, index, base)
0-1
S S I I I B B B
Displacement
0/1/2/4
Immediate Data
0/1/2/4
Main Opcode bits
Direction bit
Operand length bit
Base field
Index field
Scale field
r/m field
Register/Opcode modifier, defined by primary opcode
Addressing mode

目标：绕过DEP

- CISC所存在的问题：**代码中一个字节的含义取决于上下文**
  - 同一数值在单字节/双字节操作码中出现，意义不同
  - 同一数值作为指令中的不同元素（如操作码/立即数）时，意义不同
  - **原则上，任何一个字节都可能是一条指令的起始**

目标：绕过DEP


2nd
1st
0 1 2 3 4 5 6 7 8 9 A B C D E F
0 ADD ES PUSH SS ES POP SS OR CS PUSH DS TWO BYTE
1 ADC SBB POP DS
2 AND ES SEGMENT OVERRIDE DAA SUB CS SEGMENT OVERRIDE DAS
3 XOR SS AAA CMP DS AAS
4 INC DEC
5 PUSH POP
6 PUSHAD POPAD BOUND ARPL FS GS SEGMENT OVERRIDE OPERAND SIZE ADDRESS SIZE SIZE OVERRIDE PUSH IMUL PUSH IMUL INS OUTS
7 JO JNO JB JNB JE JNE JBE JA JS JNS JPE JPO JL JGE JLE JG Jcc
8 ADD/ADC/AND/XOR OR/SBB/SUB/CMP TEST XCHG MOV REG MOV SREG LEA MOV SREG POP
9 NOP XCHG EAX CWD CDQ CALLF WAIT PUSHFD POPFD SAHF LAHF
A MOV EAX MOVS CMPS TEST STOS LODS SCAS
B MOV
C SHIFT IMM RETN LES LDS MOV IMM ENTER LEAVE RETF INT3 INT IMM INTO IRETD
D SHIFT 1 SHIFT CL ROL/ROR/RCL/RCR/SHL/SHR/SAL/SAR AAM AAD SALC XLAT FPU
E LOOPNZ LOOPZ LOOP CONDITIONAL LOOP JECXZ IN IMM OUT IMM CALL JMP JMPF JMP SHORT IN DX OUT DX
F LOCK EXCLUSIVE ACCESS ICE BP REPNE REPE CONDITIONAL REPETITION HLT CMC TEST/NOT/NEG [i]MUL/[i]DIV CLC STC CLI STI CLD STD INC DEC INC/DEC CALL/JMP PUSH
2nd
1st
0 1 2 3 4 5 6 7 8 9 A B C D E F
0 {L,S}LDT {L,S}TR VER{R,W} {L,S}GDT {L,S}IDT {L,S}MSW LAR LSL CLTS INVD WBINVD UD2 NOP
1 SSE{1,2,3} Prefetch SSE1 HINT_NOP
2 MOV CR/DR SSE{1,2}
3 WRMSR RDTSC RDMSR RDPMC SYSENTER SYSEXIT GETSEC SMX MOVBE / THREE BYTE THREE BYTE SSE4
4 CMOV
5 SSE{1,2}
6 MMX, SSE2
7 MMX, SSE{1,2,3}, VMX MMX, SSE{2,3}
8 JO JNO JB JNB JE JNE JBE JA JS JNS JPE JPO JL JGE JLE JG Jcc SHORT
9 SETO SETNO SETB SETNB SETE SETNE SETBE SETA SETS SETNS SETPE SETPO SETL SETGE SETLE SETG SETcc
A PUSH FS POP FS CPUID BT SHLD PUSH GS POP GS RSM BTS SHRD *FENCE IMUL
B CMPXCHG LSS BTR LFS LGS MOVZX POPCNT UD BT BTS BTR BTC BTC BSF BSR MOVSX
C XADD SSE{1,2} CMPXCHG BSWAP
D MMX, SSE{1,2,3}
E MMX, SSE{1,2}
F MMX, SSE{1,2,3}

目标：绕过DEP

```
f7 c7 07 00 00 00        test $0x00000007, %edi
0f 95 45 c3              setnzb -61(%ebp)

c7 07 00 00 00 0f        movl $0x0f000000, (%edi)
95                       xchg %ebp, %eax
45                       inc %ebp
c3                       ret
```

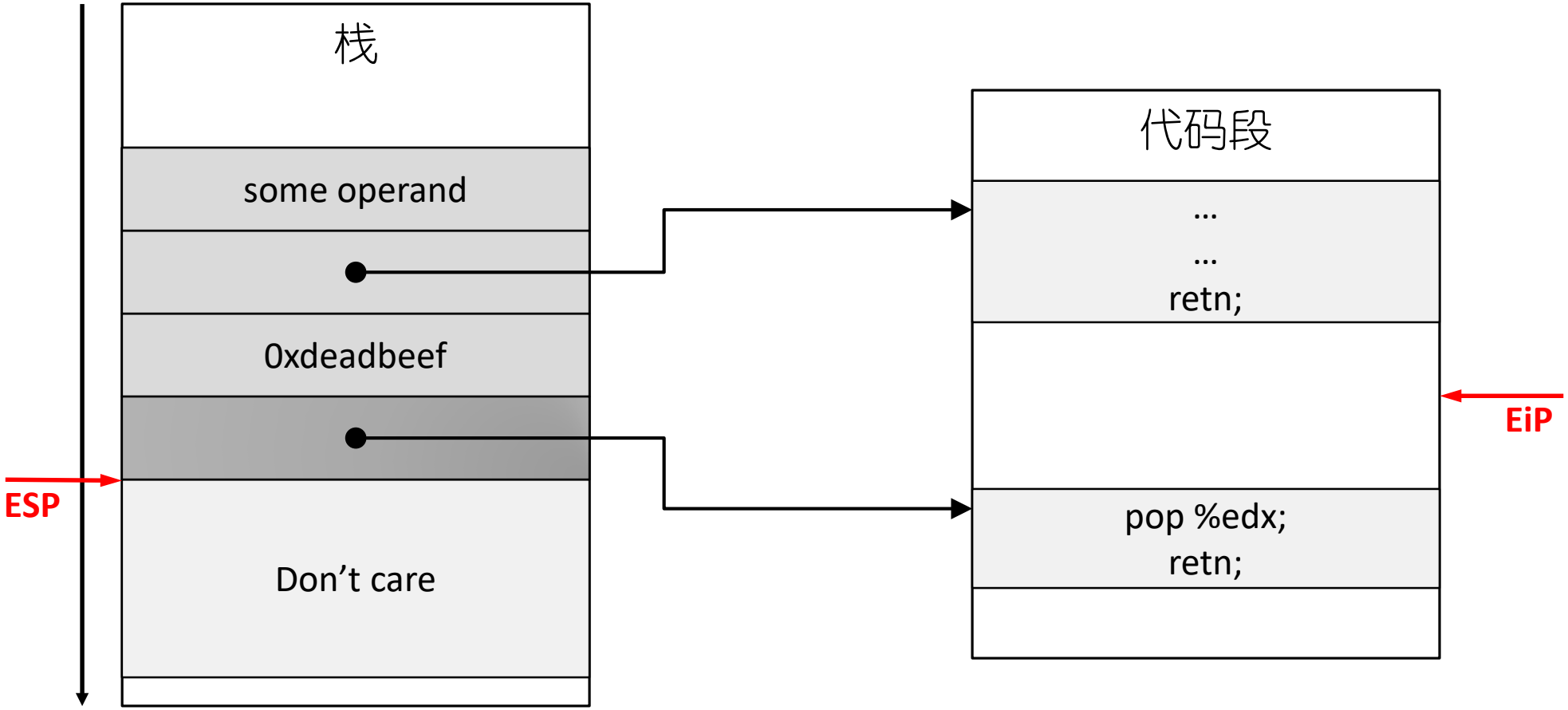
意义：通过劫持控制流，攻击者可以**令CPU按照其意愿重新解读已经存在的代码**

返回导向编程（return-oriented programming）

- 一个问题：曲解代码并不能直接给攻击者提供其想要的指令序列
  - 曲解代码的后果不完全受攻击者的控制，形成的有用指令数量往往不多
  - 攻击者的目标指令串可能比较长
  - 因此，攻击者需要一种**非常规**方式**将短小指令片段串联成完整的代码流**
- 回顾：ret指令的执行效果
  - 根据当前栈顶内容，修改指令寄存器所指向的地址
  - 将栈指针位置**向栈底方向**移动4字节

与栈溢出方向吻合，具有指令寻址能力

返回导向编程（return-oriented programming）

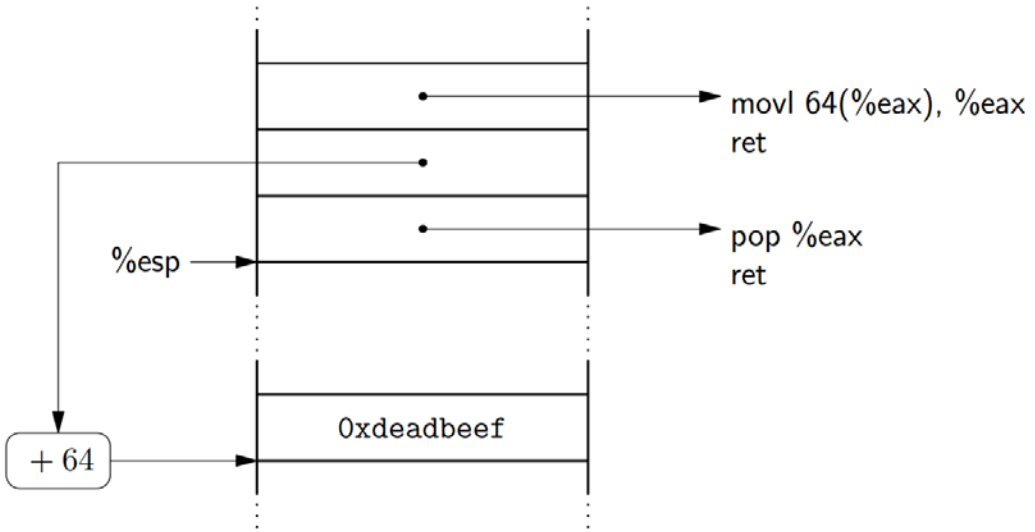


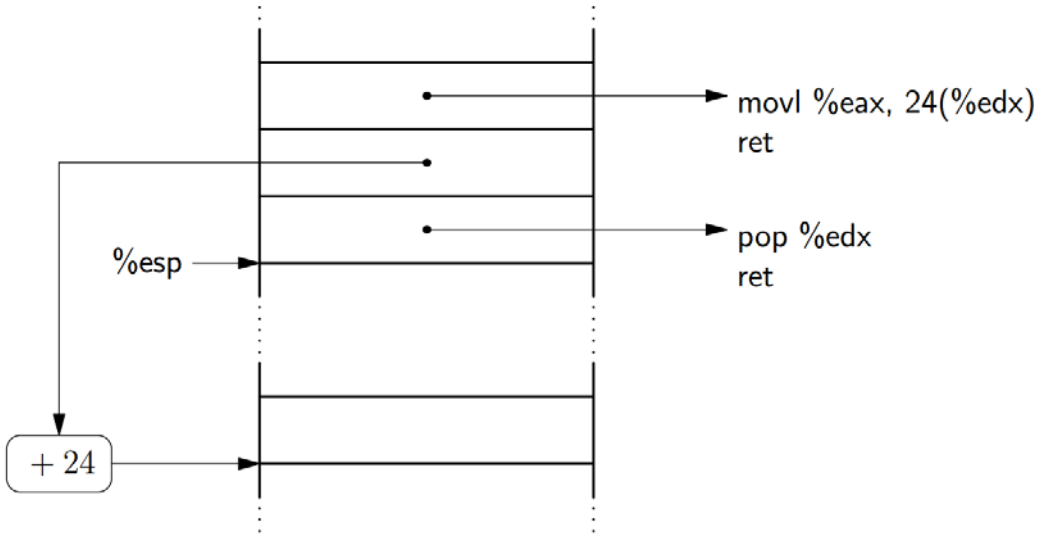
返回导向编程（return-oriented programming）

- 指令集与“图灵完全”
  - 通用图灵机：能够模拟其它所有图灵机的图灵机
  - 图灵完全的指令集能够模拟通用图灵机
  - 即：图灵完全的指令集能够实现**任意编码**
- 图灵完全的指令集应当包括
  - 加载/存储
  - 算术与逻辑
  - 控制流

返回导向编程（return-oriented programming）

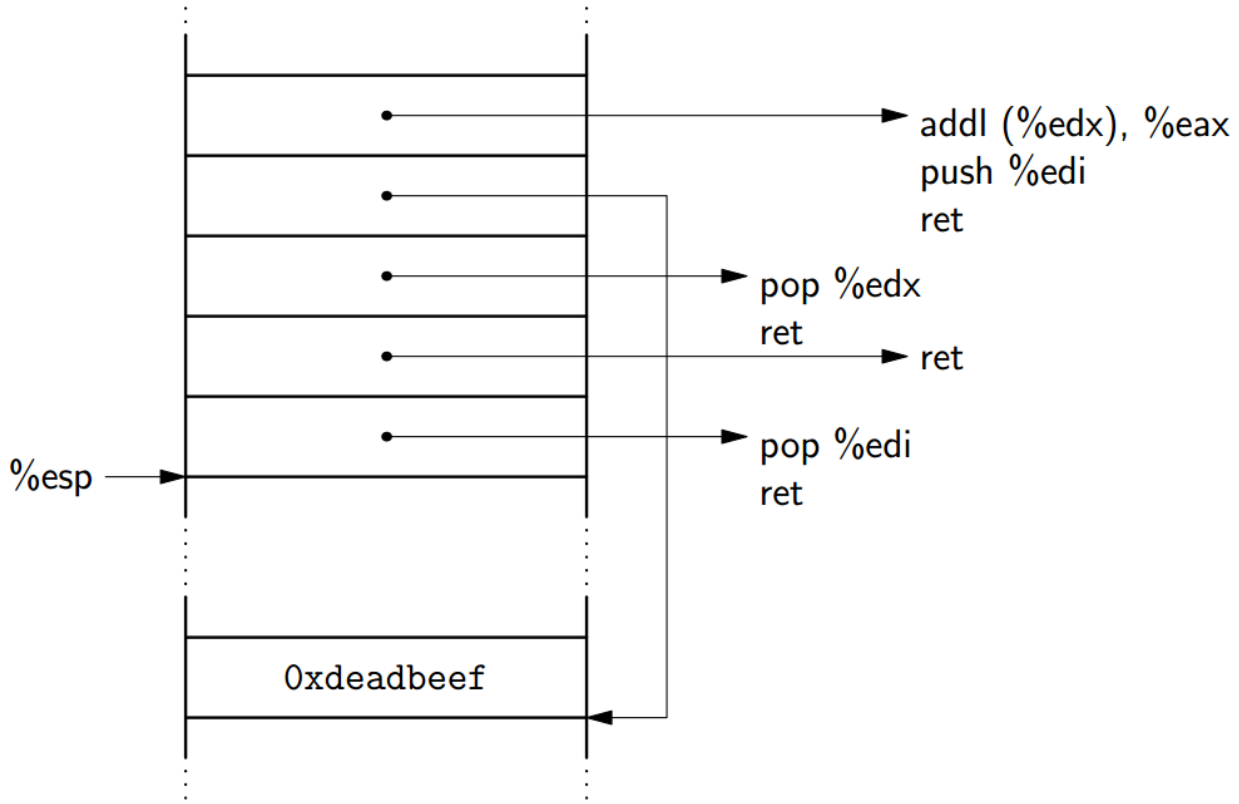
- 返回导向编程的“图灵完全” --- 加载/存储





返回导向编程（return-oriented programming）

- 返回导向编程的“图灵完全” --- 加法



返回导向编程（return-oriented programming）

- 返回导向编程的“图灵完全” --- 逻辑运算

andb %al, 0x5d5e0cc4(%ebx); ret　　and

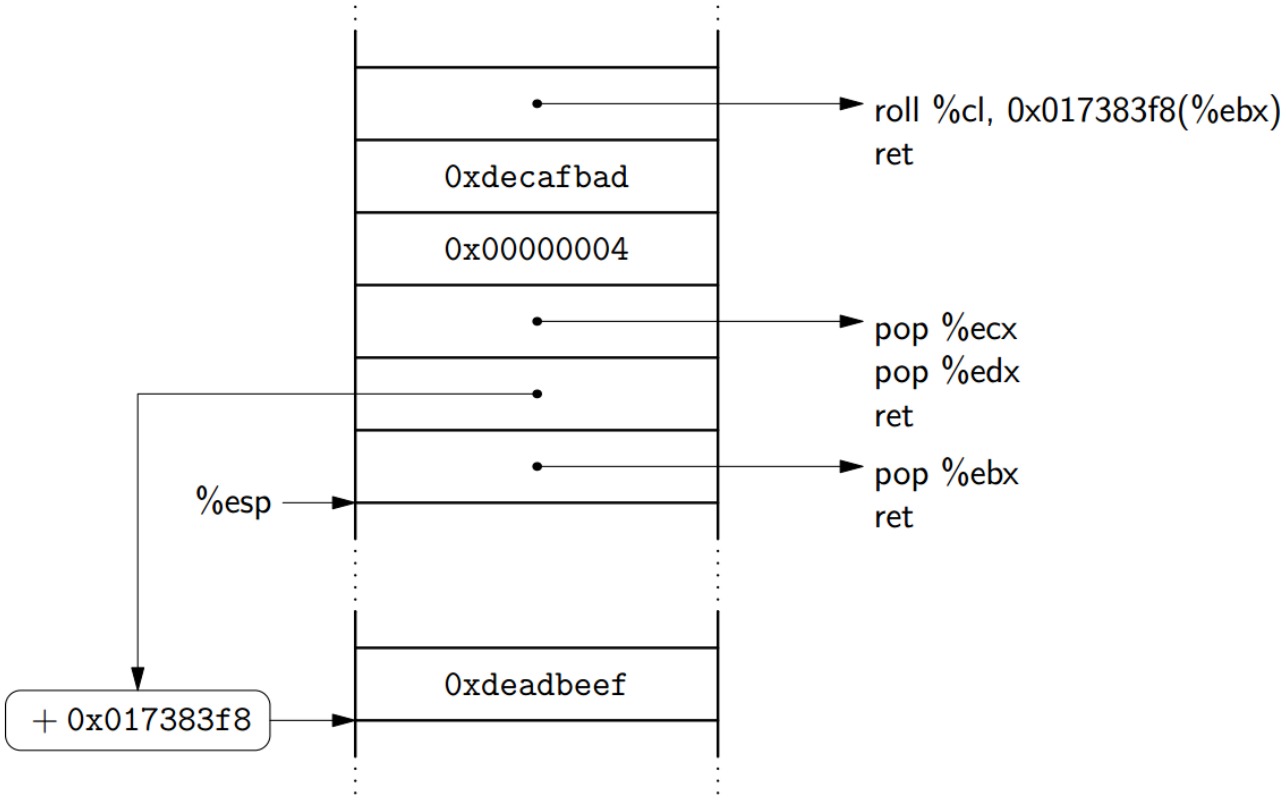
orb %al, 0x40e4602(%ebx); ret.

xorb %al, 0x48908c0(%ebx);　　and $0xff, %al;

push %ebp;　　or $0xc9, %al;　　ret.

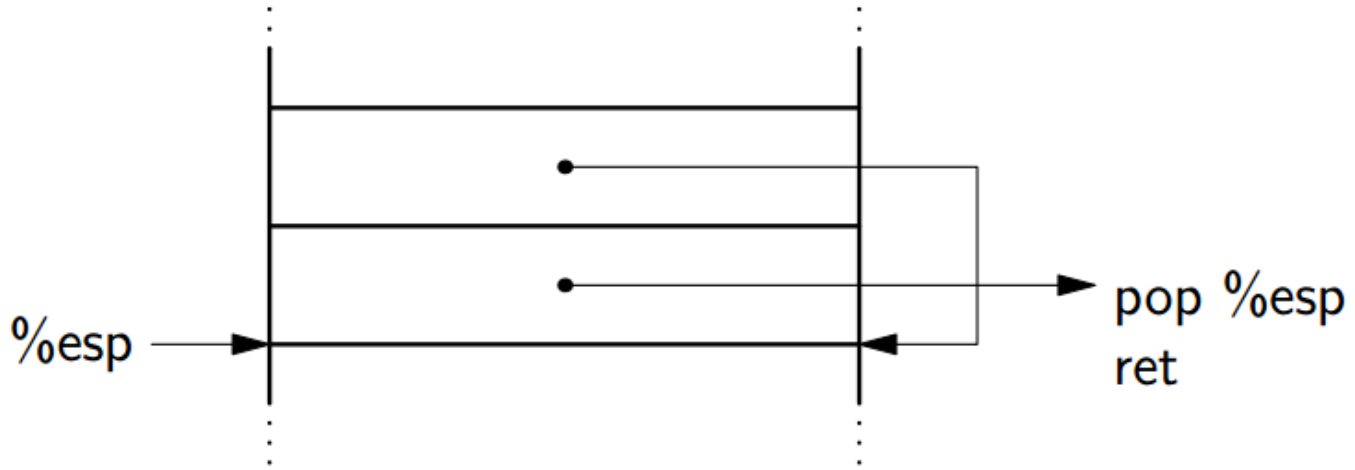
返回导向编程（return-oriented programming）

- 返回导向编程的“图灵完全” --- 位移



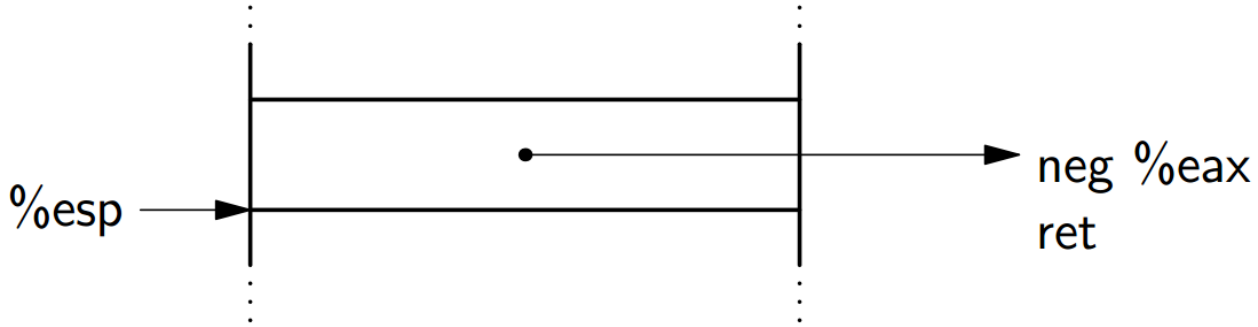
返回导向编程（return-oriented programming）

- 返回导向编程的“图灵完全” --- 控制流
  - 无条件跳转（由于ROP过程中栈指针代替了指令指针的作用）：



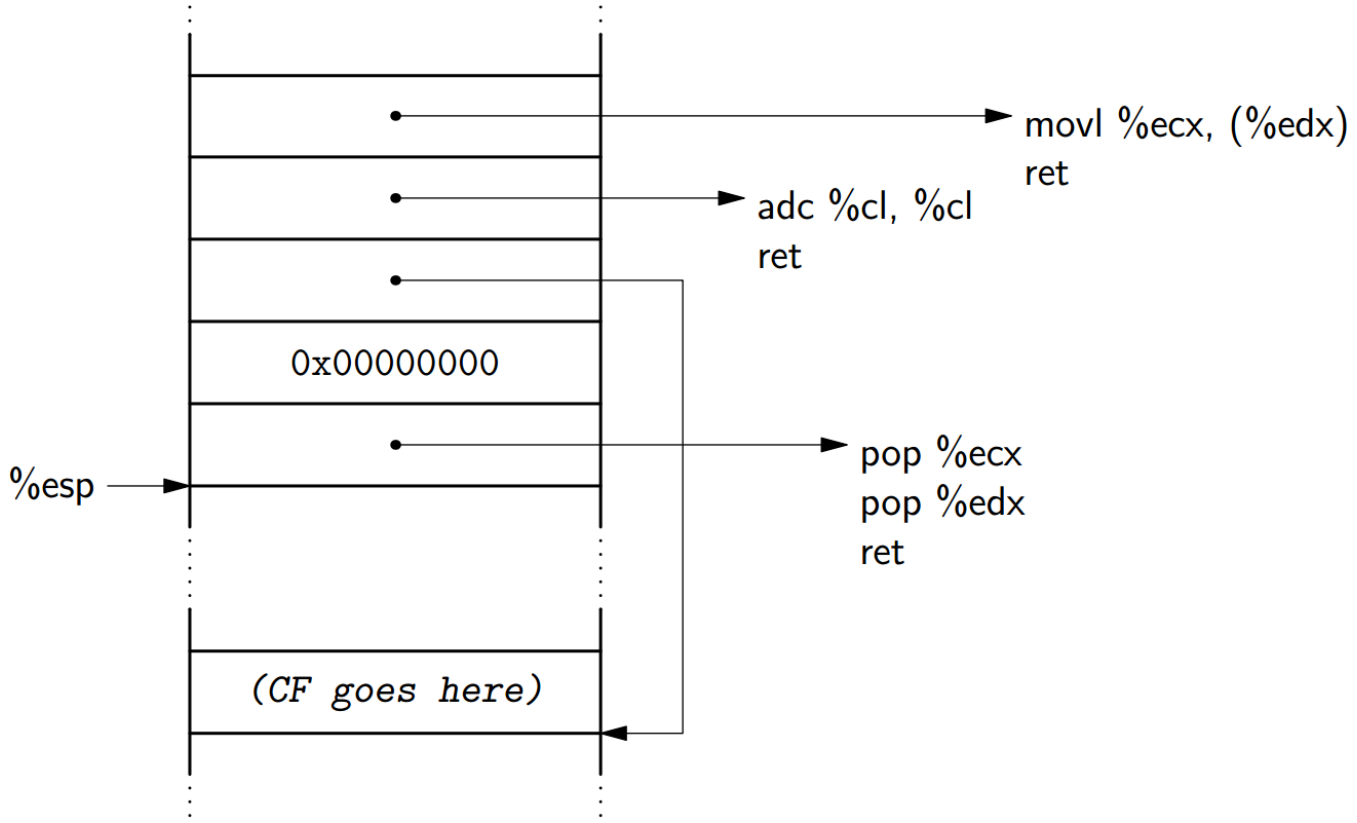
返回导向编程（return-oriented programming）

- 返回导向编程的“图灵完全” --- 控制流
  - 条件跳转：



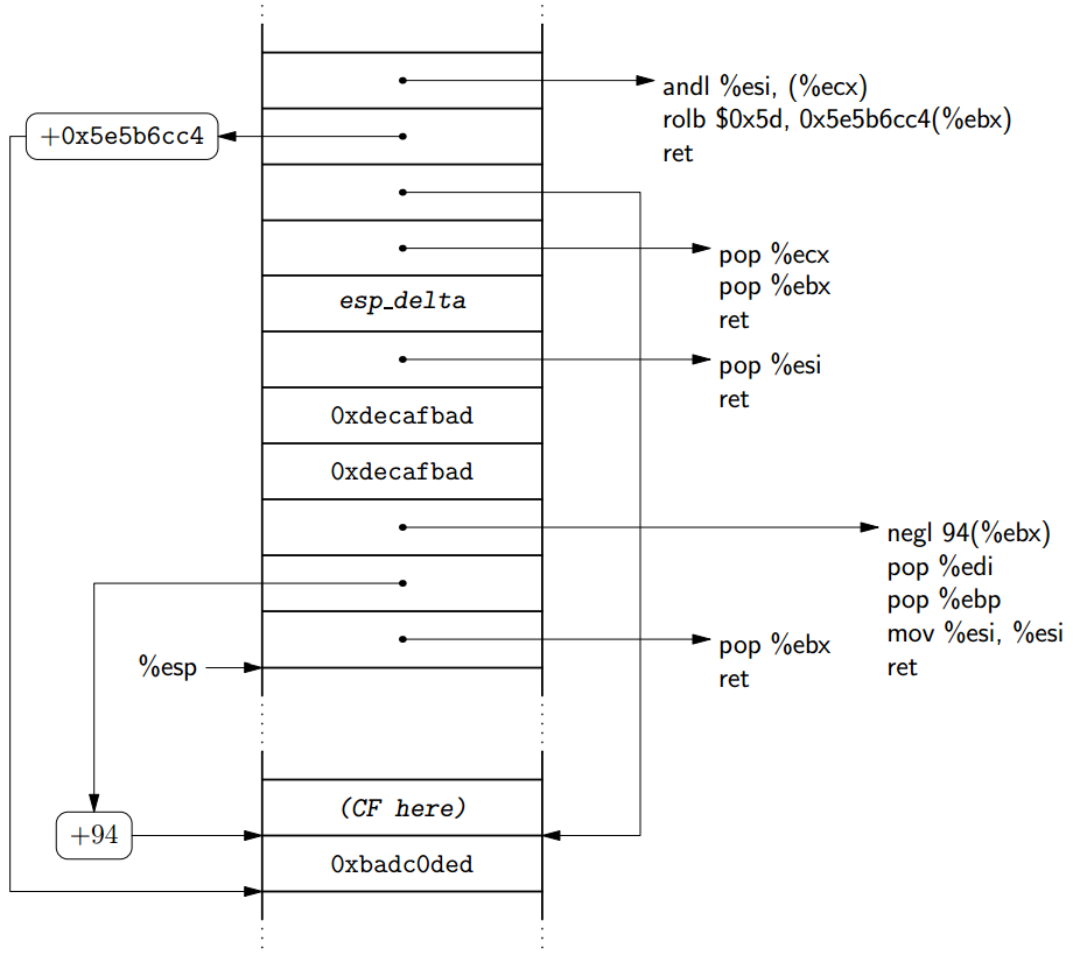
返回导向编程（return-oriented programming）

- 返回导向编程的“图灵完全” --- 控制流
  - 条件跳转：



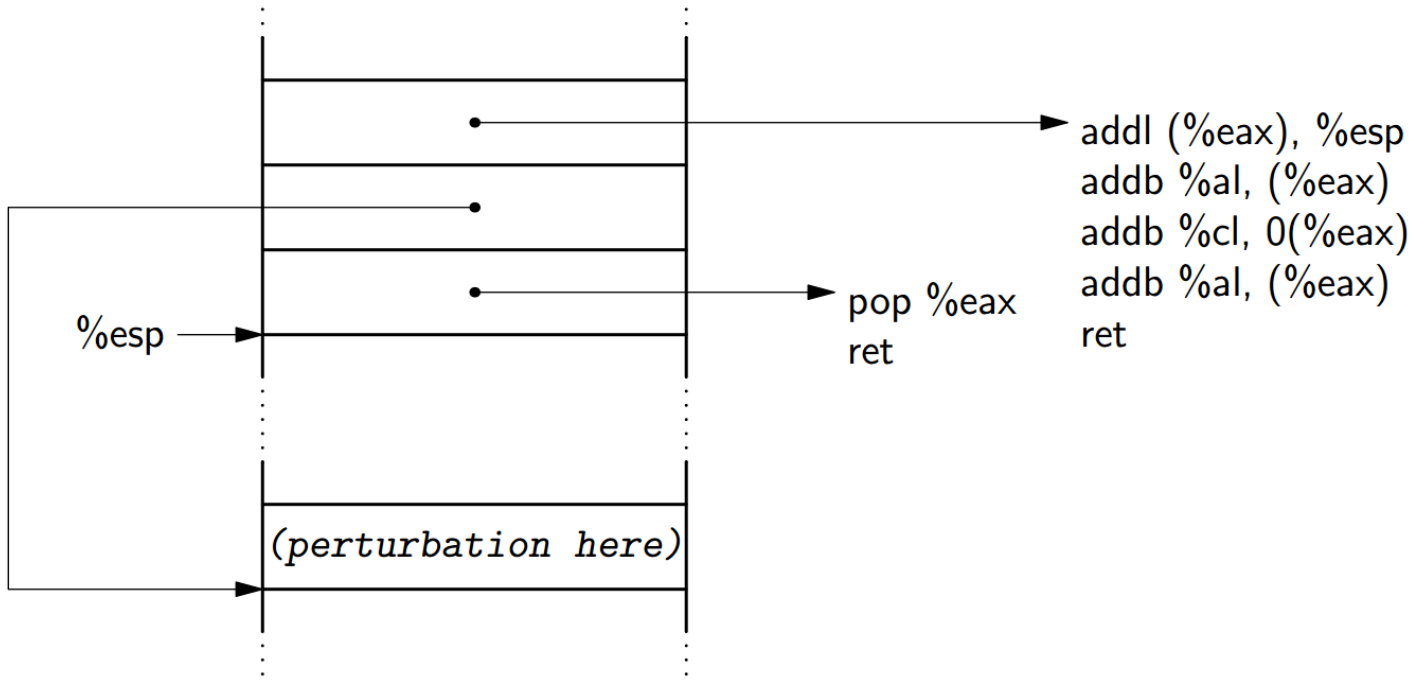
返回导向编程（return-oriented programming）

- 返回导向编程的“
  - 条件跳转：



返回导向编程（return-oriented programming）

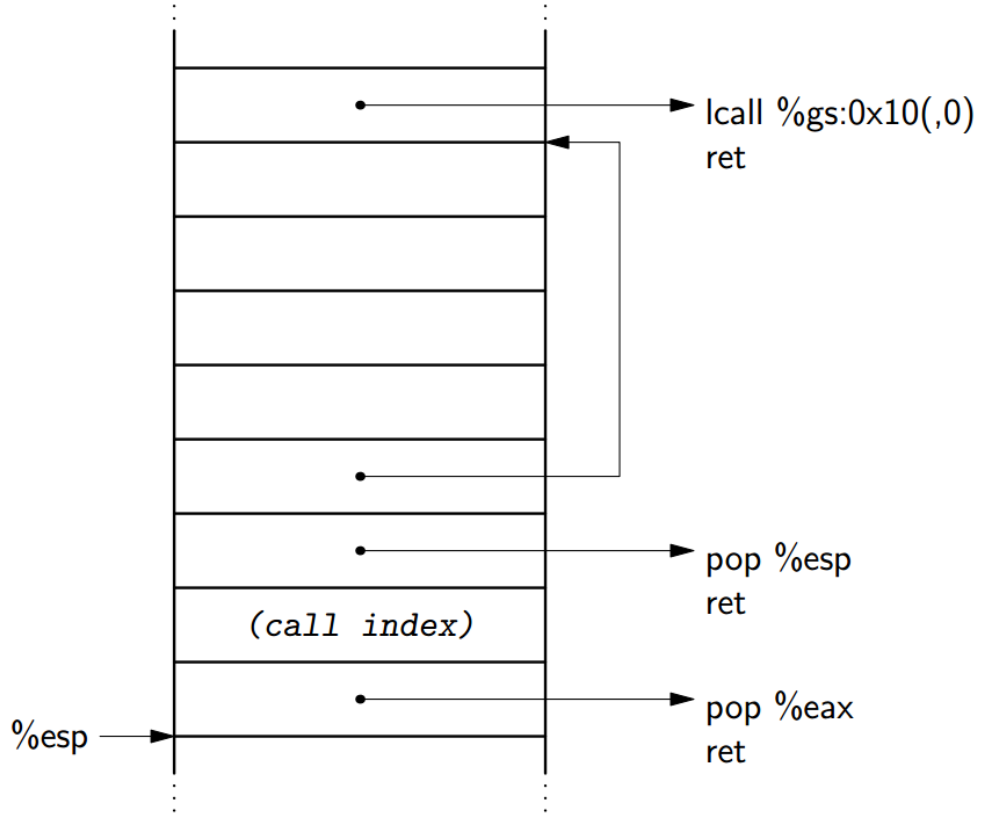
- 返回导向编程的“图灵完全” --- 控制流
  - 条件跳转：



返回导向编程（return-oriented programming）

- 返回导向编程的“图灵完全” --- 系统调用

```
89 da                   mov %ebx, %edx
8b 5c 24 04             movl 4(%esp), %ebx
b8 3c 00 00 00          mov $0x0000003C, %eax
65 ff 15 10 00 00 00    lcall %gs:0x10(,0)
89 d3                   mov %edx, %ebx
c3                      ret
```



返回导向编程（return-oriented programming）

- 为返回导向编程搜索可用的指令资源 --- Galileo算法

Algorithm GALILEO:

create a node, *root*, representing the ret instruction;
place *root* in the trie;
for *pos* **from** 1 **to** *textseg_len* **do**:
if the byte at *pos* is c3, i.e., a ret instruction, **then**:
call BUILDFROM(*pos*, *root*).

Procedure BUILDFROM(index *pos*, instruction *parent_insn*):
for *step* **from** 1 **to** *max_insn_len* **do**:
if bytes $[(pos - step) \ldots (pos - 1)]$ decode as a valid instruction *insn* **then**:
ensure *insn* is in the trie as a child of *parent_insn*;
if *insn* isn't boring **then**:
call BUILDFROM($pos - step$, *insn*).

What’s next?

- 针对返回导向编程的防御技术
- 返回导向编程的变种