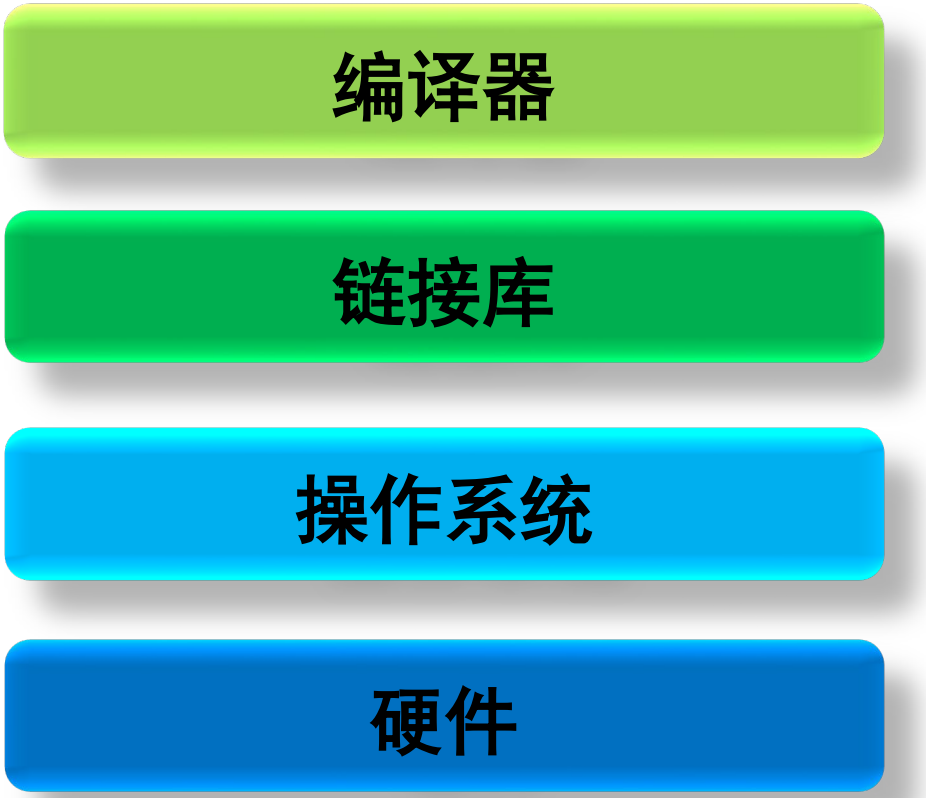
内容概要

01 内存安全防护体系与防护技术

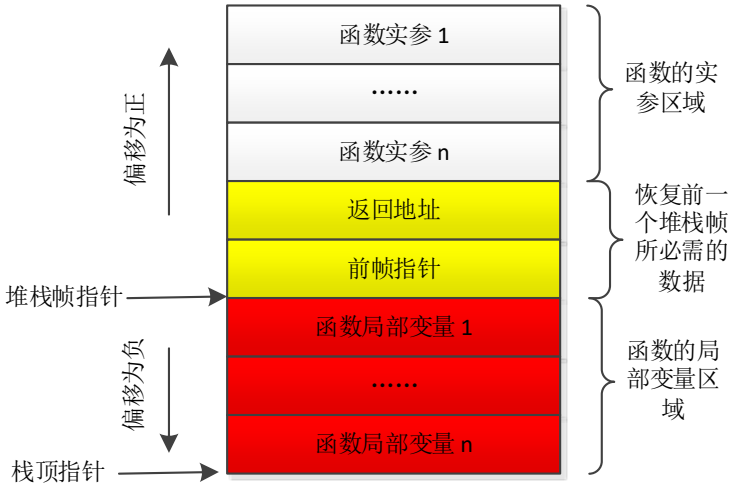
02 内存安全机制实现案例

内存防护技术的四个层次



溢出检测技术

- “Canaries” 探测技术
- 在函数栈缓冲区和控制信息（如 EBP 等）间插入 canary word
- 函数返回时检查函数栈中的canary word是否被修改



“canary word”形式：

- Terminator canaries
  - 0x00000000，C字符串遇NULL结束
  - 0x000aff0d字符串作为 canary word，NULL（0x00），CR（0x0d），LF（0x0a）和 EOF（0xff）四个字符，0x00 使 strcpy() 结束，0x0a 会使 gets() 结束
  - 固定，容易在shellcode构造回去

- Random canaries
  - 随机产生Canary word，程序初始化时产生，保存到特定地方
  - 优点：不同程序canary word不同
  - 缺陷：同个程序canary word 相同，最终会被猜测出
- Random XOR canaries
  - 由一个随机数和函数栈中的所有控制信息、返回地址通过异或运算得到
  - 优点：不易伪造，数栈中的 canaries 或者任何控制信息、返回地址被修改就都能被检测

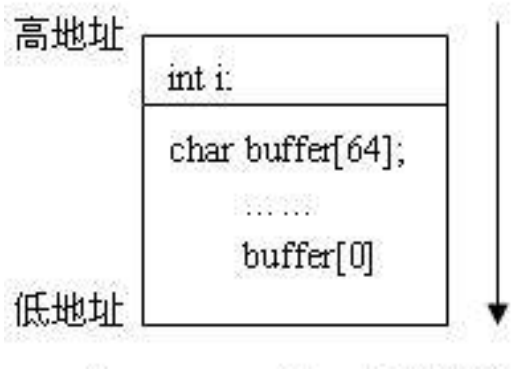
- 主流编译器（GCC等）栈保护技术
  - Stack Guard
  - Stack-smashing Protection(SSP，ProPolice)
  - Canaries 探测作为它们主要的保护技术

- Stack-smashing Protection（SSP）
  - 保护返回地址
  - 保护栈EBP等控制信息
  - 局部变量中的数组放在函数栈的高地址，其他变量放在低地址
  - 通过溢出一个数组来修改其他变量（如一个函数指针）变得困难

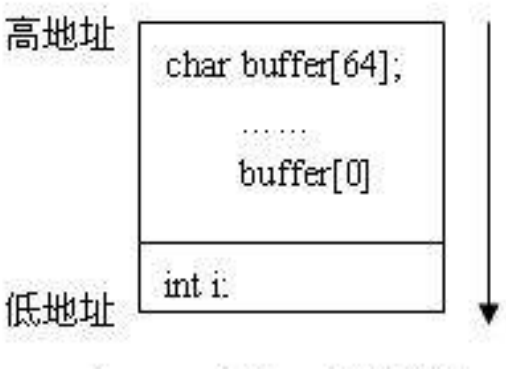
启用Stack-smashing Protection函数栈变化

```
int main()
{
int i;
 char buffer[64];
 i = 1;
buffer[0] = 'a';
 return 0;
}
```

没启用SSP



启用SSP



GCC （4.1+） 栈保护有关的编译选项

- -fstack-protector

  启用堆栈保护，保护函数中通过alloca()分配缓存以及存在大于8字节的缓存的函数

- -fstack-protector-all

  启用堆栈保护，为所有函数插入保护代码

- stack-protector-strong

  在stack-protector基础上，增加本地数组、指向本地帧栈地址空间保护

- stack-protector-explicit

  只对有含有stack_protect属性标识的函数提供保护

- -fno-stack-protector

  禁用堆栈保护

溢出检查——Stackshield

- 创建一个特别的堆栈用来储存函数返回地址的一份拷贝
- 受保护的函数的开头和结尾分别增加一段代码，开头处的代码用来将函数返回地址拷贝到一个特殊的表中，而结尾处的代码用来将返回地址从表中拷贝回堆栈

边界检查

- 运行时对（数组、指针）边界进行检查
- 描述了每个分配内存块的中央数据块
- 包含了指针以及描述它们指向区域的额外数据的胖指针

Formatguard

- 是Glibc的补丁，遵循GPL
- 它使用特殊的CPP（gcc预编译程序）宏取代原有的*printf()的参数统计方式，比较传递给*printf的参数的个数和格式窜的个数
- 格式窜的个数大于实际参数的个数，判定为攻击行为，向syslogd发送消息并终止进程
- 缺陷：程序调用Glibc以外的库，formatguard就无法保护

Libsafe

- 是一个动态链接库
- 在标准的C库之前被加载
- 主要加固gets()，strcpy()，strcat()，sprintf()……等容易发生安全问题的C函数
- 针对stack smashing和 format string类型的攻击

安全库（函数）

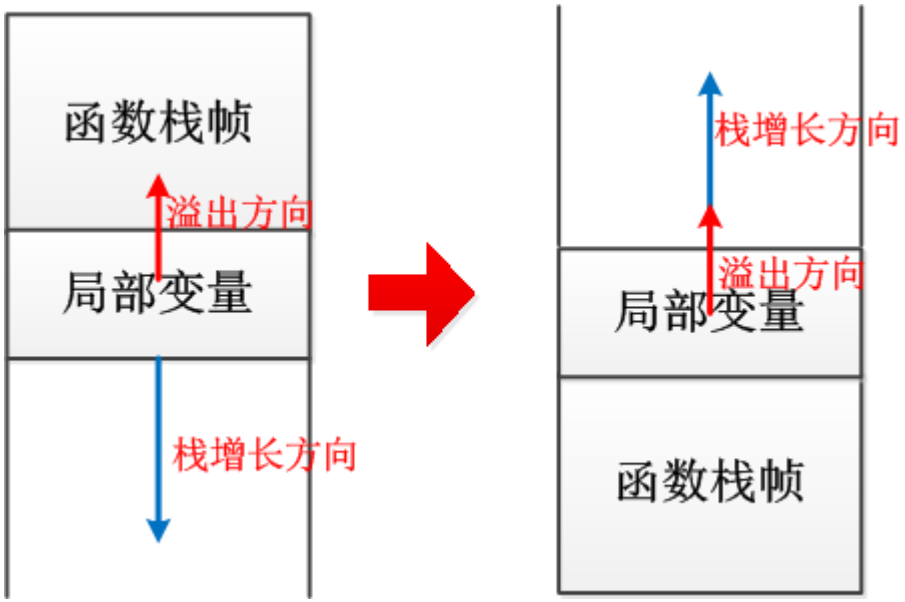
- 静态分配的缓冲区方法
  - 当缓冲区用完时，拒绝为缓冲区增加任何空间
- 标准 C 库方法：
  - 标准 C strncpy/strncat 和 OpenBSD 的 strlcpy/strlcat
- 动态分配的缓冲区方法
  - 当缓冲区用完时，动态地将缓冲区大小调整到更大的尺寸，直至用完所有内存
- SafeStr
  - C++ std::string

守护页机制（Guard Page）

- 针对栈脆弱点，攻击者操纵栈不断增长，恶意增加栈向下覆盖其他地址空间内容
- 在栈中增加一个空白页，限制栈的最大的大小（在栈顶能向下增长的最大的地址之后）
- 该页的权限设置为不可读，不可写
- 若攻击者设法使栈恶意增长，越过守护页时，触发“segmentation fault”信号，导致程序运行出错，进入异常处理

溢出保护

- 改变栈的增长方向
- Top to Down □ Down to Top
- 溢出不破坏栈帧结构（返回地址）



- 不可执行保护（无直接硬件支持）
  - 基于页式管理实现
  - TLB划分成ITLB,DTLB
  - 重载页表U/S位

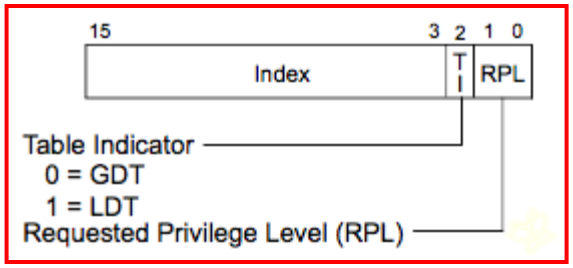
不可执行保护（无直接硬件支持）

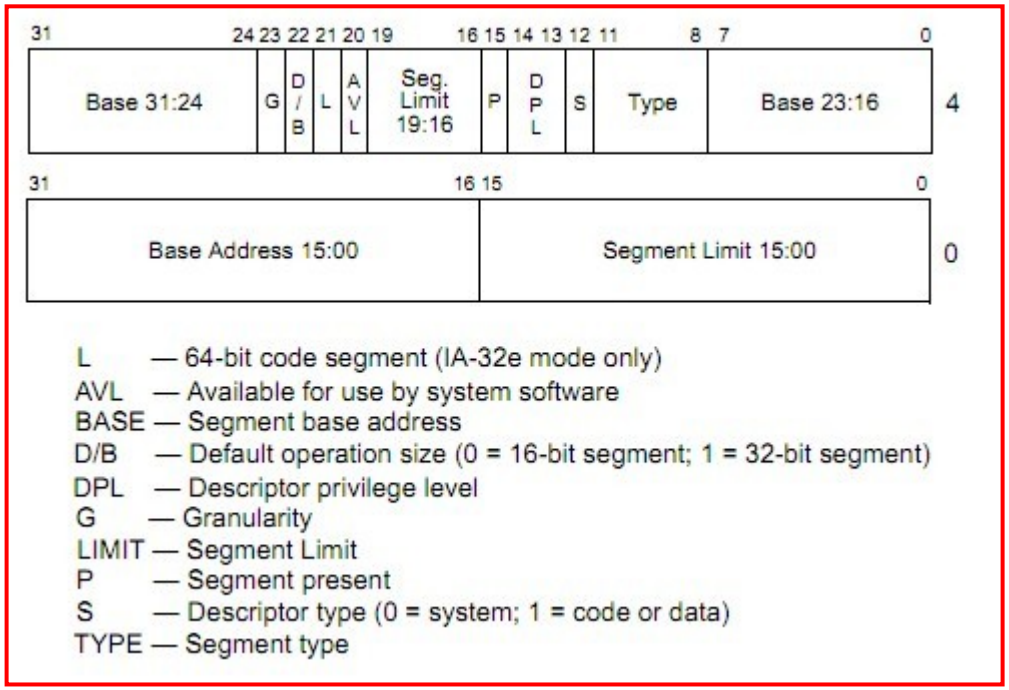
31–22	21–13	12	11–9	8	7	6	5	4	3	2	1	0	
Address of page directory[1]			Ignored					PCD	PWT	Ignored			CR3
Bits 31:22 of address of 2MB page frame	Reserved (must be 0)	Bits 39:32 of address[2]	PAT	Ignored	G	1	D	A	PCD	PWT	U/S	R/W	1	PDE: 4MB page
Address of page table			Ignored		0	Ign	A	PCD	PWT	U/S	R/W	1	PDE: page table
Ignored												0	PDE: not present
Address of 4KB page frame			Ignored	G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page
Ignored												0	PTE: not present

不可执行保护（无直接硬件支持）

- 访问规则
  - Supervisor mode (levels 0-2): user or supervisor pages allowed (u/s==*)
  - User mode (level 3): user only (u/s==1)
- 重载U/S位□ executable/non-executable status
  - U/S=1　executable page
  - U/S=0　non-executable page
- 受保护页的PDE和PTE的U/S=S(0)
  - 当访问到不可执行页时，产生页访问异常
  - 查看异常原因
  - 终止程序执行

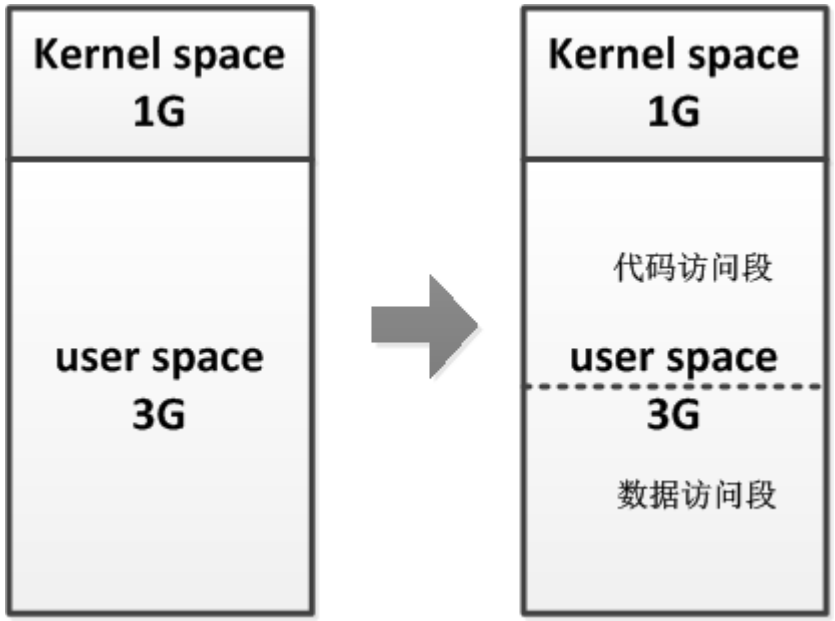
不可执行保护（无直接硬件支持）





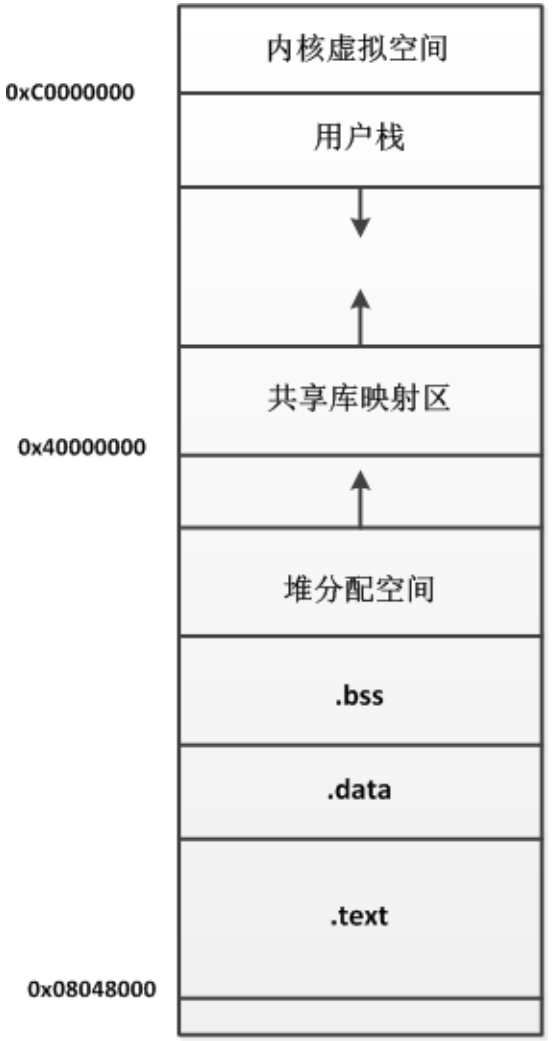
不可执行保护（无直接硬件支持）

- 基于段式管理实现
- 把用户空间3G划分出两段
- 其中一段为数据访问段，另一段为代码访问段



- 不可执行保护（无直接硬件支持）
  - 主要代表
    - Solar designer’ s nonexec kernel patch
    - Solaris/SPARC nonexec-stack protection
    - kNoX
    - RSX
    - Exec shield
    - PaX

- 虚拟空间固定分配弊端
  - 进程地址空间布局一致
  - 相同程序在同一平台的计算机中地址空间布局完全一致
  - 地址容易猜测，实施攻击难度低（ret2libc，ROP）



地址空间固定分配地址空间分布

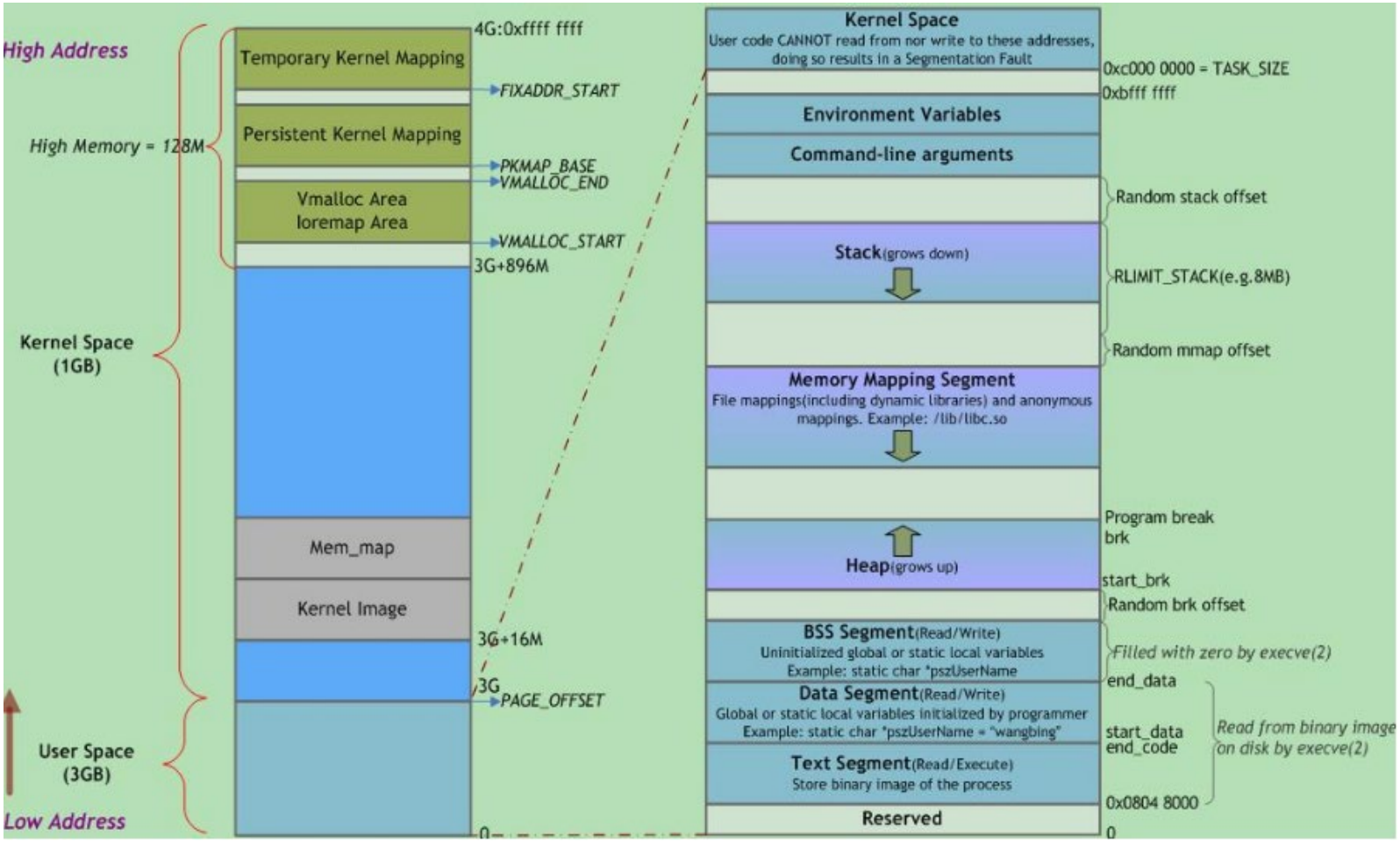
```
root@ubuntu:~# cat /proc/self/maps
08048000-08053000 r-xp 00000000 08:01 1054472    /bin/cat
08053000-08054000 r--p 0000a000 08:01 1054472    /bin/cat
08054000-08055000 rw-p 0000b000 08:01 1054472    /bin/cat
08055000-08076000 rw-p 00000000 00:00 0          [heap]
b7abc000-b7c21000 r--p 001c8000 08:01 922740     /usr/lib/locale/locale-archive
b7c21000-b7e21000 r--p 00000000 08:01 922740     /usr/lib/locale/locale-archive
b7e21000-b7e22000 rw-p 00000000 00:00 0
b7e22000-b7fc5000 r-xp 00000000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b7fc5000-b7fc7000 r--p 001a3000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b7fc7000-b7fc8000 rw-p 001a5000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b7fc8000-b7fcb000 rw-p 00000000 00:00 0
b7fda000-b7fdb000 r--p 005e0000 08:01 922740     /usr/lib/locale/locale-archive
b7fdb000-b7fdd000 rw-p 00000000 00:00 0
b7fdd000-b7fde000 r-xp 00000000 00:00 0          [vdso]
b7fde000-b7ffe000 r-xp 00000000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
b7ffe000-b7fff000 r--p 0001f000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
b7fff000-b8000000 rw-p 00020000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
bffdf000-c0000000 rw-p 00000000 00:00 0          [stack]
root@ubuntu:~#
```

第一次运行

```
root@ubuntu:~# cat /proc/self/maps
08048000-08053000 r-xp 00000000 08:01 1054472    /bin/cat
08053000-08054000 r--p 0000a000 08:01 1054472    /bin/cat
08054000-08055000 rw-p 0000b000 08:01 1054472    /bin/cat
08055000-08076000 rw-p 00000000 00:00 0          [heap]
b7abc000-b7c21000 r--p 001c8000 08:01 922740     /usr/lib/locale/locale-archive
b7c21000-b7e21000 r--p 00000000 08:01 922740     /usr/lib/locale/locale-archive
b7e21000-b7e22000 rw-p 00000000 00:00 0
b7e22000-b7fc5000 r-xp 00000000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b7fc5000-b7fc7000 r--p 001a3000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b7fc7000-b7fc8000 rw-p 001a5000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b7fc8000-b7fcb000 rw-p 00000000 00:00 0
b7fda000-b7fdb000 r--p 005e0000 08:01 922740     /usr/lib/locale/locale-archive
b7fdb000-b7fdd000 rw-p 00000000 00:00 0
b7fdd000-b7fde000 r-xp 00000000 00:00 0          [vdso]
b7fde000-b7ffe000 r-xp 00000000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
b7ffe000-b7fff000 r--p 0001f000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
b7fff000-b8000000 rw-p 00020000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
bffdf000-c0000000 rw-p 00000000 00:00 0          [stack]
root@ubuntu:~#
```

第二次运行

地址空间随机化分配（ASLR）



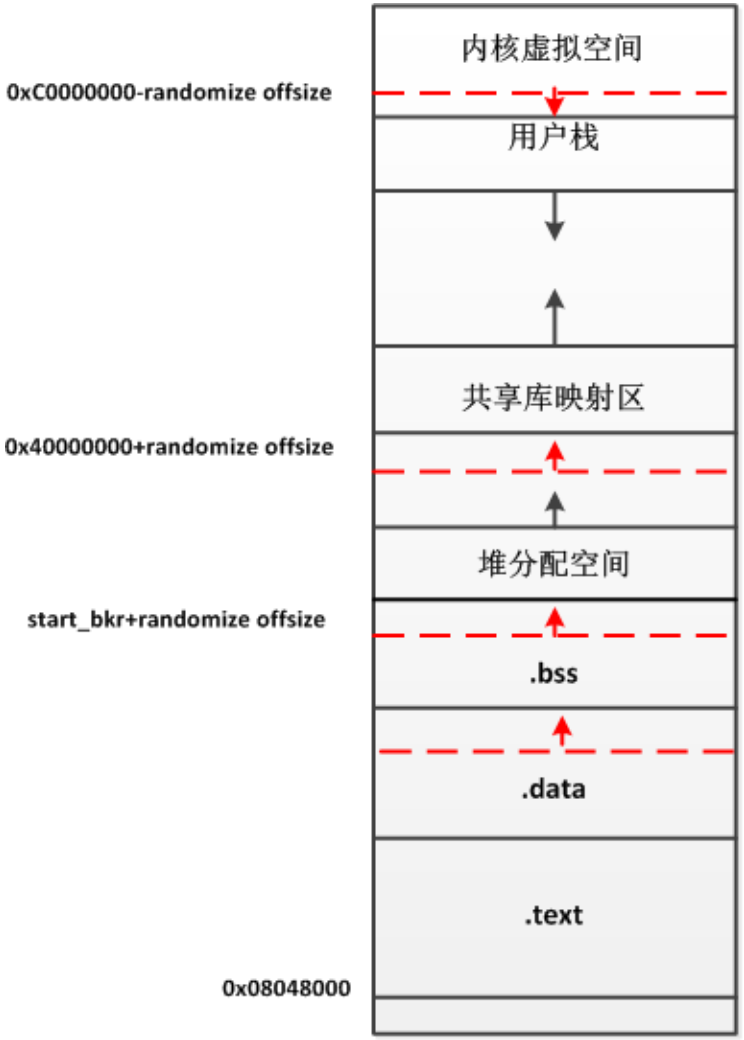
地址空间随机化分配

- ASLR——address space layout randomization
- 改变传统地址空间固定分配方式
- 每个地址区域起始地址=固定基值+/-随机偏移值
- 随机化关键因素
- 随机化策略
- 随机值的质量

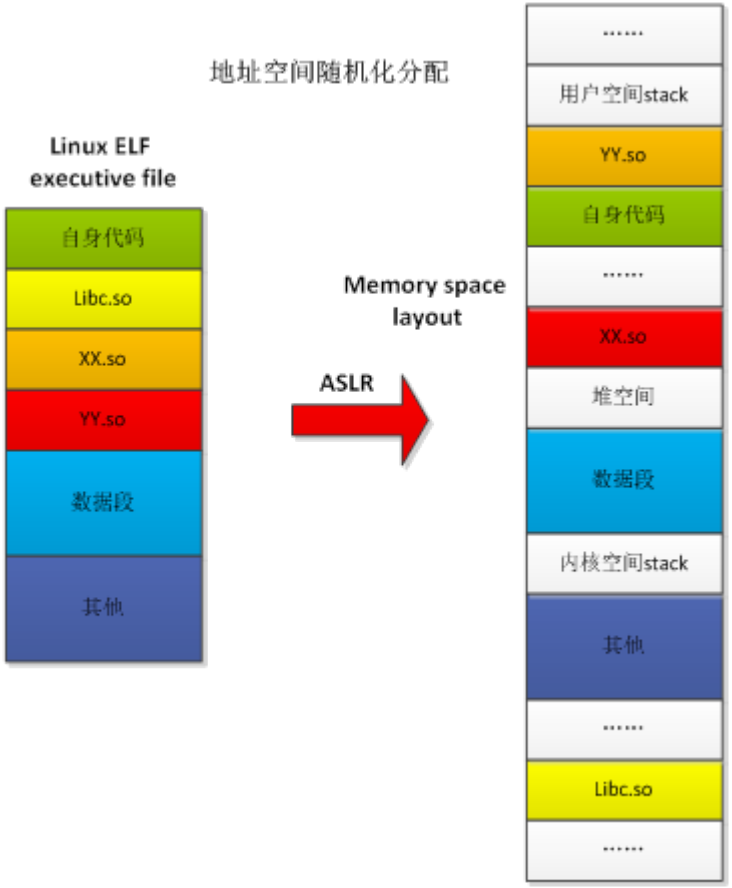
地址空间随机化分配地址空间分布

```
root@ubuntu:/home/abang# cat /proc/self/maps
08048000-08053000 r-xp 00000000 08:01 1054472    /bin/cat
08053000-08054000 r--p 0000a000 08:01 1054472    /bin/cat
08054000-08055000 rw-p 0000b000 08:01 1054472    /bin/cat
0925c000-0927d000 rw-p 00000000 00:00 0          [heap]
b7241000-b73a6000 r--p 001c8000 08:01 922740     /usr/lib/locale/locale-archive
b73a6000-b75a6000 r--p 00000000 08:01 922740     /usr/lib/locale/locale-archive
b75a6000-b75a7000 rw-p 00000000 00:00 0
b75a7000-b774a000 r-xp 00000000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b774a000-b774c000 r--p 001a3000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b774c000-b774d000 rw-p 001a5000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b774d000-b7750000 rw-p 00000000 00:00 0
b775f000-b7760000 r--p 005e0000 08:01 922740     /usr/lib/locale/locale-archive
b7760000-b7762000 rw-p 00000000 00:00 0
b7762000-b7763000 r-xp 00000000 00:00 0          [vdso]
b7763000-b7783000 r-xp 00000000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
b7783000-b7784000 r--p 0001f000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
b7784000-b7785000 rw-p 00020000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
bfac9000-bfaea000 rw-p 00000000 00:00 0          [stack]
root@ubuntu:/home/abang# echo 0 > /proc/sys/kernel/randomize_va_space
root@ubuntu:/home/abang# cat /proc/self/maps
08048000-08053000 r-xp 00000000 08:01 1054472    /bin/cat
08053000-08054000 r--p 0000a000 08:01 1054472    /bin/cat
08054000-08055000 rw-p 0000b000 08:01 1054472    /bin/cat
08055000-08076000 rw-p 00000000 00:00 0          [heap]
b7abc000-b7c21000 r--p 001c8000 08:01 922740     /usr/lib/locale/locale-archive
b7c21000-b7e21000 r--p 00000000 08:01 922740     /usr/lib/locale/locale-archive
b7e21000-b7e22000 rw-p 00000000 00:00 0
b7e22000-b7fc5000 r-xp 00000000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b7fc5000-b7fc7000 r--p 001a3000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b7fc7000-b7fc8000 rw-p 001a5000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b7fc8000-b7fcb000 rw-p 00000000 00:00 0
b7fda000-b7fdb000 r--p 005e0000 08:01 922740     /usr/lib/locale/locale-archive
b7fdb000-b7fdd000 rw-p 00000000 00:00 0
b7fdd000-b7fde000 r-xp 00000000 00:00 0          [vdso]
b7fde000-b7ffe000 r-xp 00000000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
b7ffe000-b7fff000 r--p 0001f000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
b7fff000-b8000000 rw-p 00020000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
bffdf000-c0000000 rw-p 00000000 00:00 0          [stack]
```

- 地址空间随机化分配
  - 随机分配方式一：相同属性的空间区域统一随机值错位
  - 各个空间区域相对位置保持不变
  - 优点：兼容性好
  - 缺点：随机化弱，容易被暴力破解
  - 代表：linux 内核



- 地址空间随机化分配
  - 随机分配方式二：所有的地址空间区域采用不同的随机值错位
  - 各个空间区域相对位置随机
  - 优点：随机化强，不容易被暴力破解
  - 缺点：兼容性不好
  - 代表：PaX 全随机化



- 地址空间随机化分配
  - 进程内核栈随机化
  - 每个进程有两个页面作为进程陷入内核态的栈，用于系统调用参数传递，上下文切换，中断、等
  - 代表：PaX

```
struct task_struct
{
 ...
void *stack; // 指向内核栈的指针
...
};

union thread_union
{
struct thread_info thread_info;
unsigned long
stack[THREAD_SIZE/sizeof(long)];
};
```

- 页面存在位保护
  - 页面存在位P=1的页面才可以访问
  - P=0,页面无效，页面访问异常，缺页处理

63	62：52	51：12	11	10	9	8	7	66	5	4	3	2	1	0
NX		PFN				G	PAT	D	A	PCD	PWT	U/S	R/W	P

- 读/写位保护
  - P=1,R/W=1，页面可读、写、执行
  - P=1,R/W=0，页面只读，可执行

63	62：52	51：12	11	10	9	8	7	66	5	4	3	2	1	0
NX		PFN				G	PAT	D	A	PCD	PWT	U/S	R/W	P

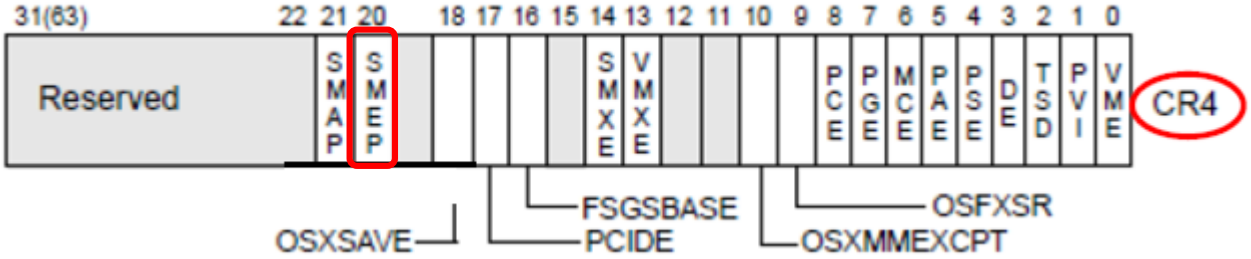
页面属于 read/write 权限，则必须要每一级页表项的 R/W 位都为 1。而属于 read-only 权限，只需要任何一级页表项的 R/W 位为 0

- WP（write protected）写保护
  - 处理器CR0.WP功能防止 supervisor-mode 改写只读（read-only）页面
  - CR0.WP = 0, supervisor-mode 可以对只读（read-only）页面进行写访问
  - CR0.WP = 1, supervisor-mode 不能对只读（read-only）页面进行写访问
  - 不论CR0.WP 为何值都不允许User-mode对只读（read-only）页面进行写访问

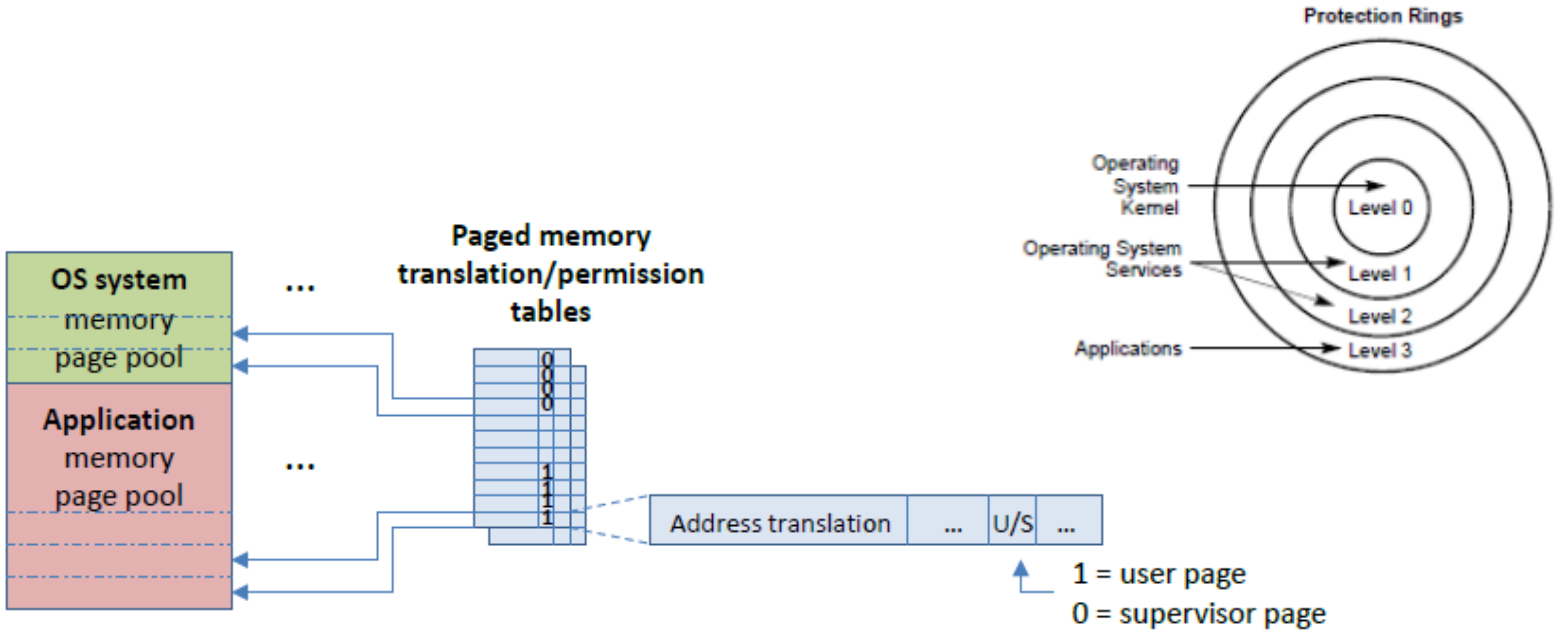
- 不可执行位
  - 增加页表执行位属性
  - NX，XD
  - 原理：取指令代码页时，如果页表的不可执行位置位时，产生页访问异常，终止进程

63	62：52	51：12	11	10	9	8	7	66	5	4	3	2	1	0
NX		PFN				G	PAT	D	A	PCD	PWT	U/S	R/W	P

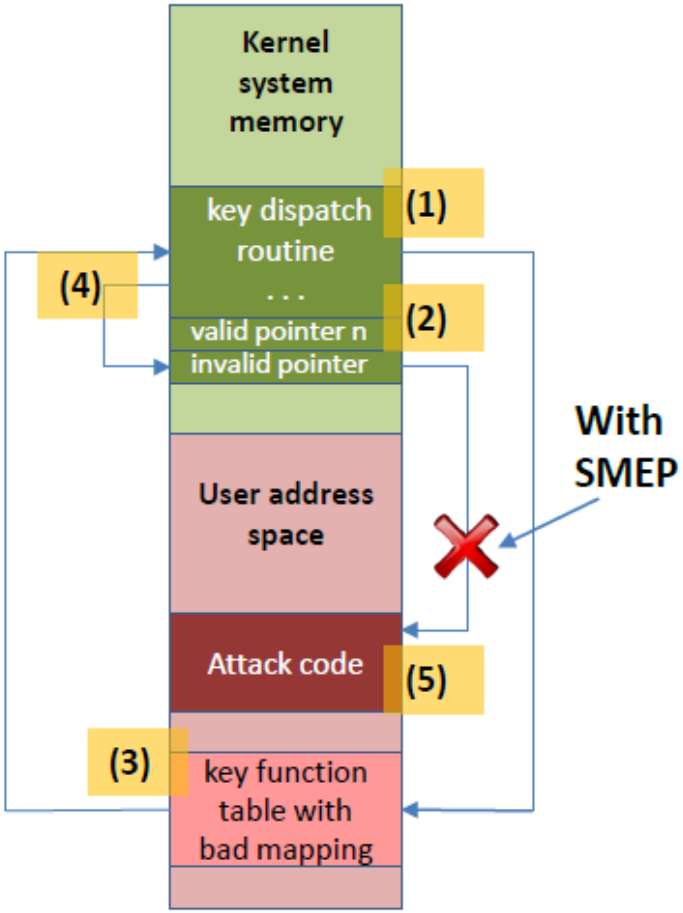
- 管理模式执行保护
  - SMEP——Supervisor Mode Execution Protection
  - 防止提权运行
  - 防止管理权限运行用户空间代码



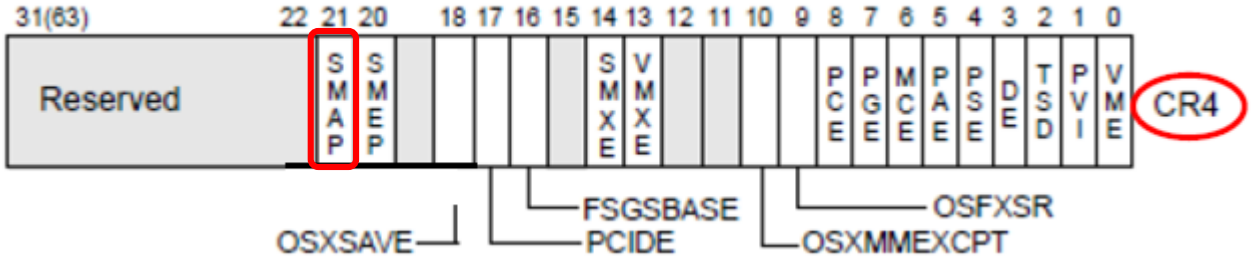
管理模式执行保护


Protection Rings
Operating System Kernel
Operating System Services
Applications
Level 0
Level 1
Level 2
Level 3
OS system memory page pool
Application memory page pool
Paged memory translation/permission tables
Address translation
U/S
1 = user page
0 = supervisor page

- 管理模式执行保护
  - **Historical access permission rules for code execution：**
    - Supervisor mode (levels 0-2) user or supervisor pages allowed (u/s==*)
    - User mode (level 3): user only (u/s==1)
  - **When SMEP is active:**
    - Supervisor mode: supervisor only (u/s==0)
    - User mode: user only (u/s==1)



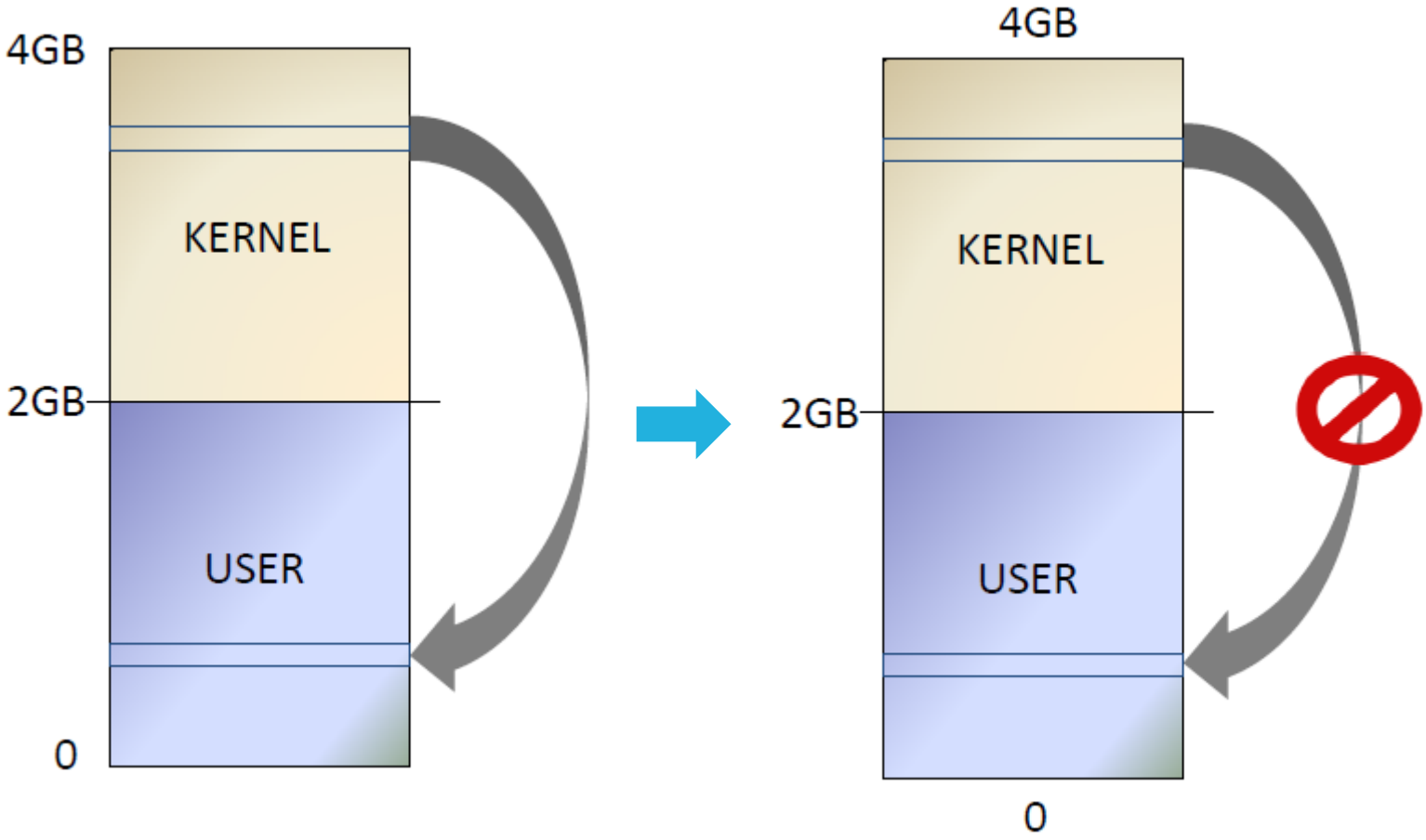
- 管理模式访问阻止
  - SMAP——supervisor mode access prevention
  - 防止提权页面访问
    - 阻止高特权运行模式下对用户页的访问
    - 当开启 SMAP 机制后，处理器运行在 supervisor 权限下（CPL < 3），将不能访问（包括 read 与 write）属于 user 权限的页面里的数据（U/S = 1）



- supervisor 与 user 页面
  - **在 32-bit paging 模式下，PDE 以及 PTE 的 U/S 位（bit 2）相 "AND" 后的结果。**
  - **在 PAE paging 模式下。**
    - 使用 4K 页面时：PDPTE，PDE 以及 PTE 三者的 U/S 位（bit 2）相 "AND" 后的结果。
    - 使用 2M 页面时：PDPTE 以及 PDE 两者的 U/S 位（bit 2）相 "AND" 后的结果。
  - **在 IA-32e paging 模式下。**
    - 使用 4K 页面时：PML4E, PDPTE，PDE 以及 PTE 四者的 U/S 位（bit 2）相 "AND" 后的结果。
    - 使用 2M 页面时：PML4E, PDPTE 以及 PDE 三者的 U/S 位（bit 2）相 "AND" 后的结果。
    - 使用 1G 页面时：PML4E 以及 PDPTE 两者的 U/S 位（bit 2）相 "AND" 后的结果。
  - **因此，页面属于 user 权限，则必须要每一级页表项的 U/S 位都为 1。而属于 supervisor 权限，只需要任何一级页表项的 U/S 位为 0 即可。**

- 无SMAP:
  - 在基于 CPL 权限的页级保护措施里，有二种访问权限： supervisor 与 user 访问权限:
  - 属于 supervisor 访问权限的只有在处理器处于 CPL < 3 权限时才允许访问
  - 属于 user 访问权限时， supervisor 与 user 都可以访问 user 页面数据

管理模式访问阻止



- SMAP 机制下的读访问
  - **当处理器运行在 CPL < 3 权限下（supervisor），尝试读访问 user 页面时，有下面的情况：**
    - 当 CR4.SMAP = 0 时，supervisor 允许读访问 user 页面。
    - 当 CR4.SMAP = 1 时，取决于 eflags.AC 标志位的值，将有下面的情形
      - eflags.AC = 0 （CLAC 指令）时，supervisor 不能读访问 user 页面，将产生 #PF 异常。
      - eflags.AC = 1 （ STAC 指令） 时，supervisor 允许读访问 user 页面
  - **将 eflags.AC 位清为 0，并且 CR4.SMAP = 1 时，才真正开启 SMAP 功能。**

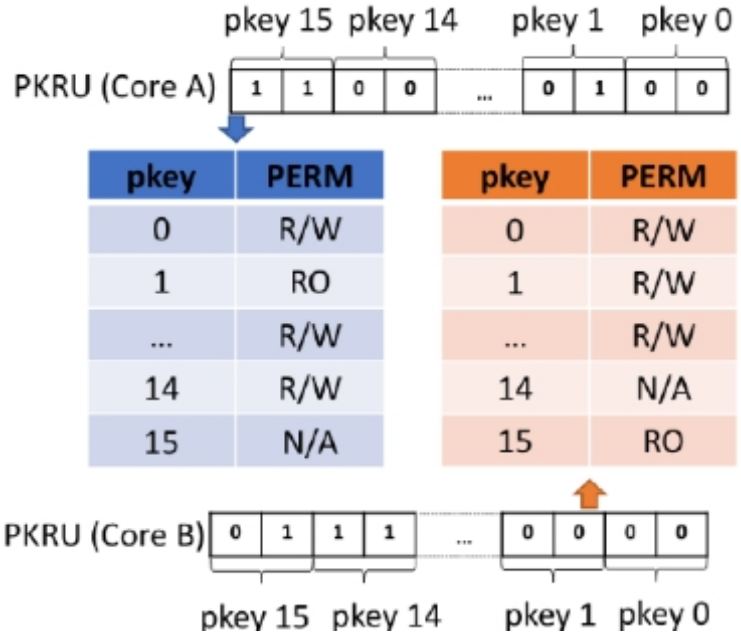
- SMAP 机制下的写访问
  - 当 CR0.WP = 0 并且 CR4.SMAP = 0 时，supervisor 允许写访问所有的 页面（包括 read-only 以及 read/write 页面）。
  - 当 CR0.WP = 1 并且 CR4.SMAP = 0 时，supervisor 允许写访问所有 read/write 页面，不能改写 read-only 页面。
  - 当 CR0.WP = 0 并且 CR4.SMAP = 1 时，取决于 eflags.AC 标志位有下面的情形：
    - AC = 0 时，supervisor 只能写访问 supervisor 的页面（包括 read-only 与 read/write 页面）。
    - AC = 1 时，supervisor 允许写访问所有的页面（包括 read-only 与 read/write 页面）。
  - 当 CR0.WP = 1 并且 CR4.SMAP = 1 时，取决于 eflags.AC 标志位有下面的情形：
    - AC = 0 时，supervisor 只能写访问 supervisor 的 read/write 页面。也就是：不能改写所有的 read-only 页面以及 user 的 read/write 页面。
    - AC = 1 时，supervisor 允许写访问所有的 read/write 页面，不能改写任何 read-only 页面。

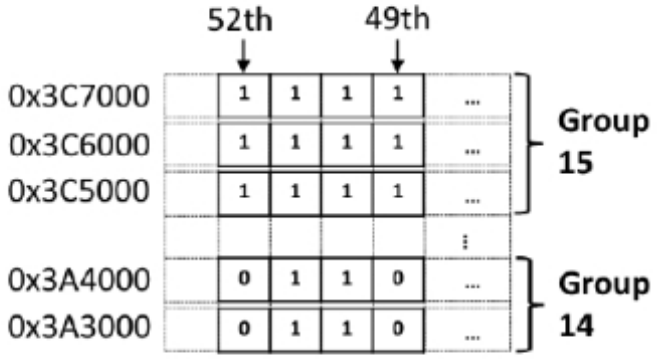
- 内存保护扩展
  - MPX——Memory Protection Extensions
  - ISA扩展，增加相应的边界寄存器和处理边界寄存器的指令
  - 在编译器、运行时库和操作系统支持下，为软件带来了增强的稳健性
  - 防止 user mode和supervisor mode缓冲区的越界访问
  - 防止缓冲区溢出

- 内存保护扩展
  - MPX——Memory Protection Extensions
    - bndmk： 在界限寄存器中创建 LowerBound (LB) 和 UpperBound (UB)
    - bndmov： 从内存中获取（上下）界限信息并将其放在界限寄存器中
  - **bndcl： 检查下界限**
  - **bndcu：检查上界限**

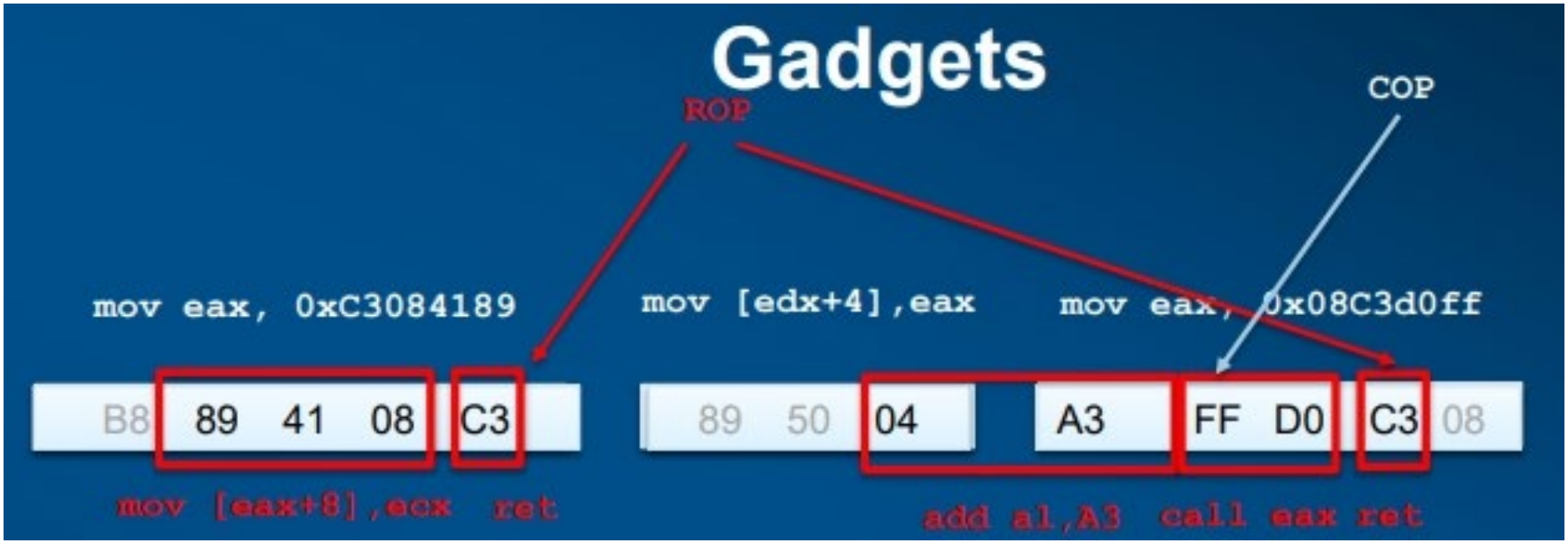
- 内存保护键
  - MPK——Memory Protection Keys
    - 一种用户空间基于页的内存权限管理机制
    - 提供更轻量的内存访问控制，可以指定一个内存区域的读，写，执行(用户空间可以直接设置)
    - 为频繁切换内存访问属性的应用提供方便，如加密应用
    - 页表条码预留4位用于’ Protection Key’,共16组
    - 可以把地址区域划分出16（PK）个区域，每个区域独立设置访问控制权限

- 每个处理器核拥有独立的32位PKRU（Protection Key rights for user pages）寄存器,包含 16 对 bit pair，对应 16 个 page group，每个 pair 包含两个 bit， 分别代表 R / W 权限。每个 pair 的取值对应三种可能的权限 (0, 0): read/write，(1, 0): read-only，(x, 1): no-access



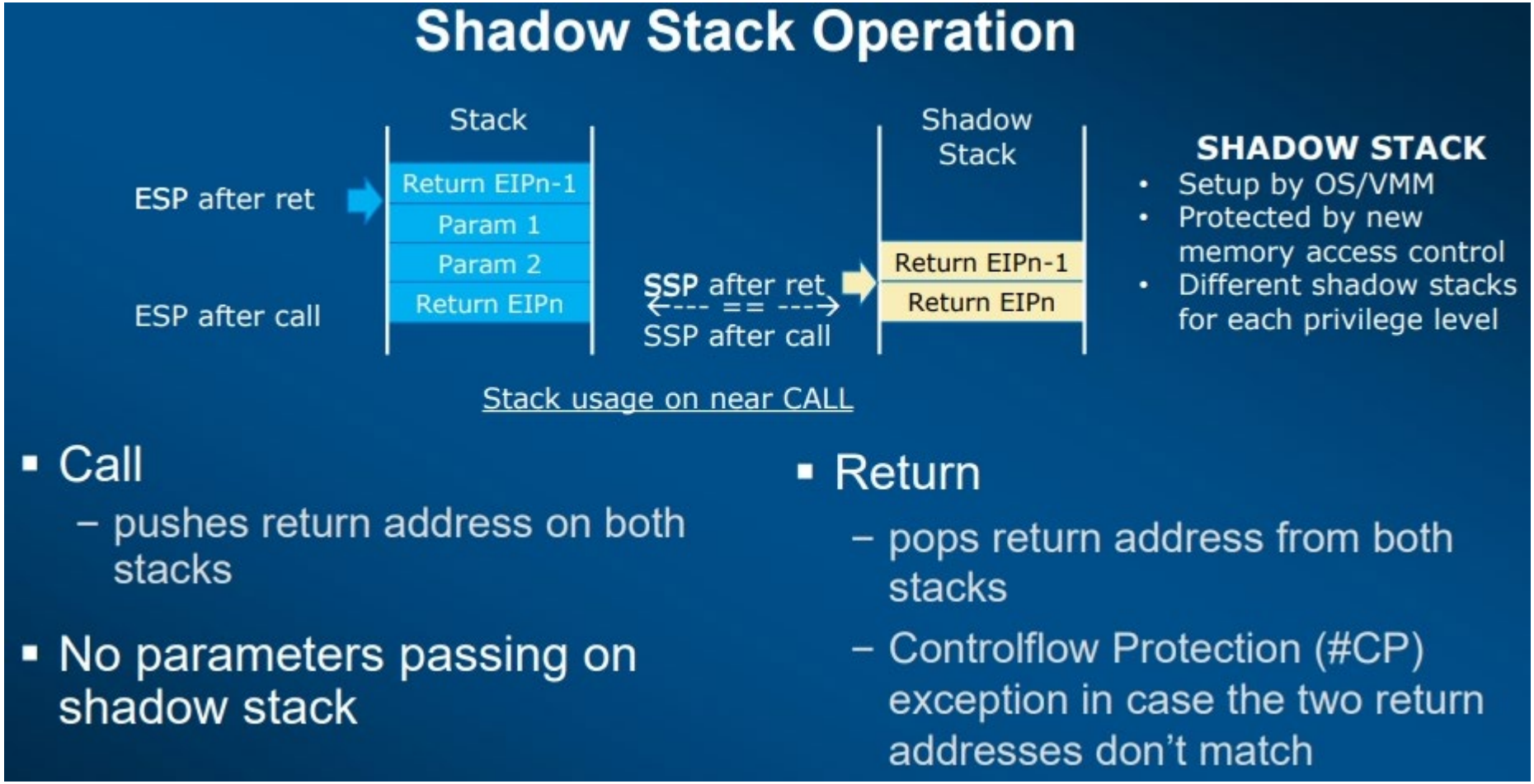


- 控制流增强技术
  - 以匹配'ret'特征的检测技术失效



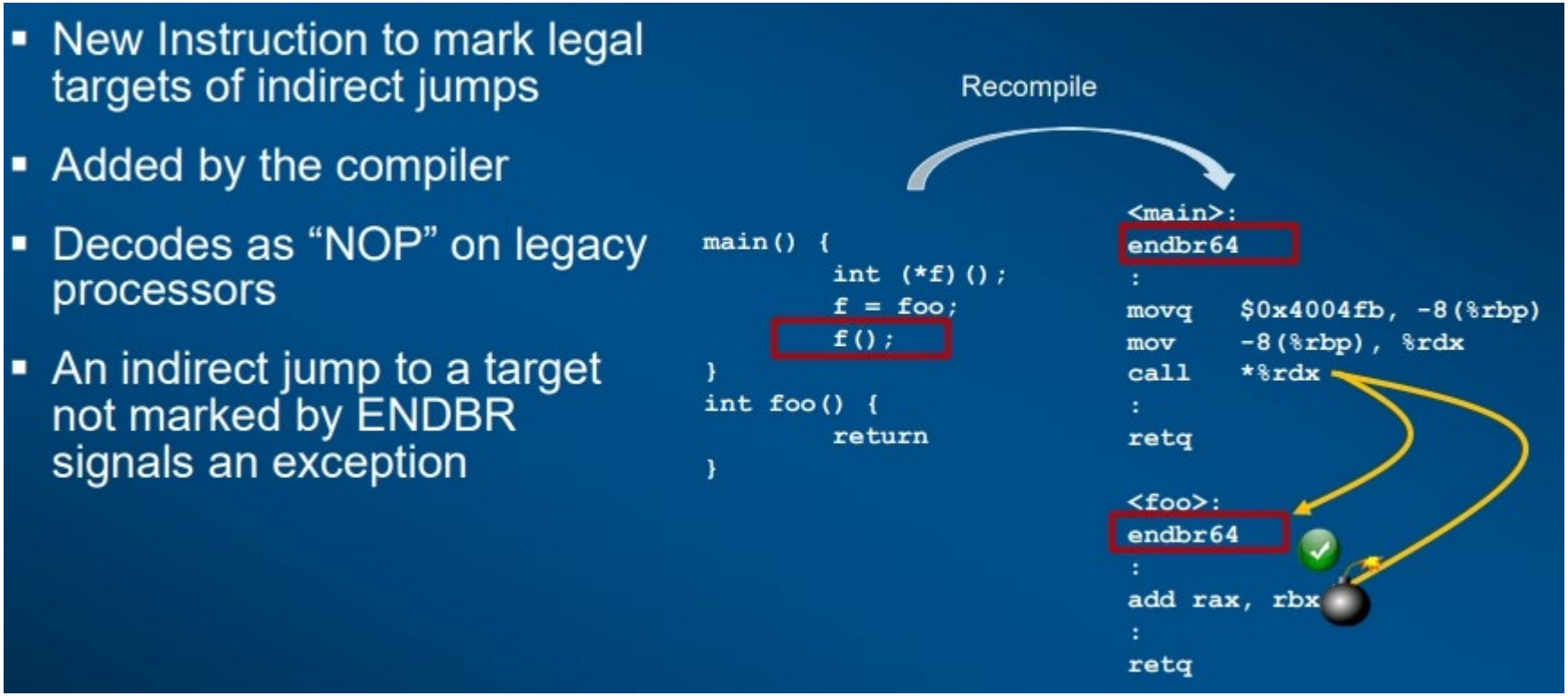
- 控制流增强技术
  - CET ——Control-flow Enforcement Technology
  - 提高防御ROP/JOP/COP控制流攻击的技术
  - 影子堆栈（Shadow Stack）
  - 返回地址保护来防范返回导向编程攻击

控制流增强技术

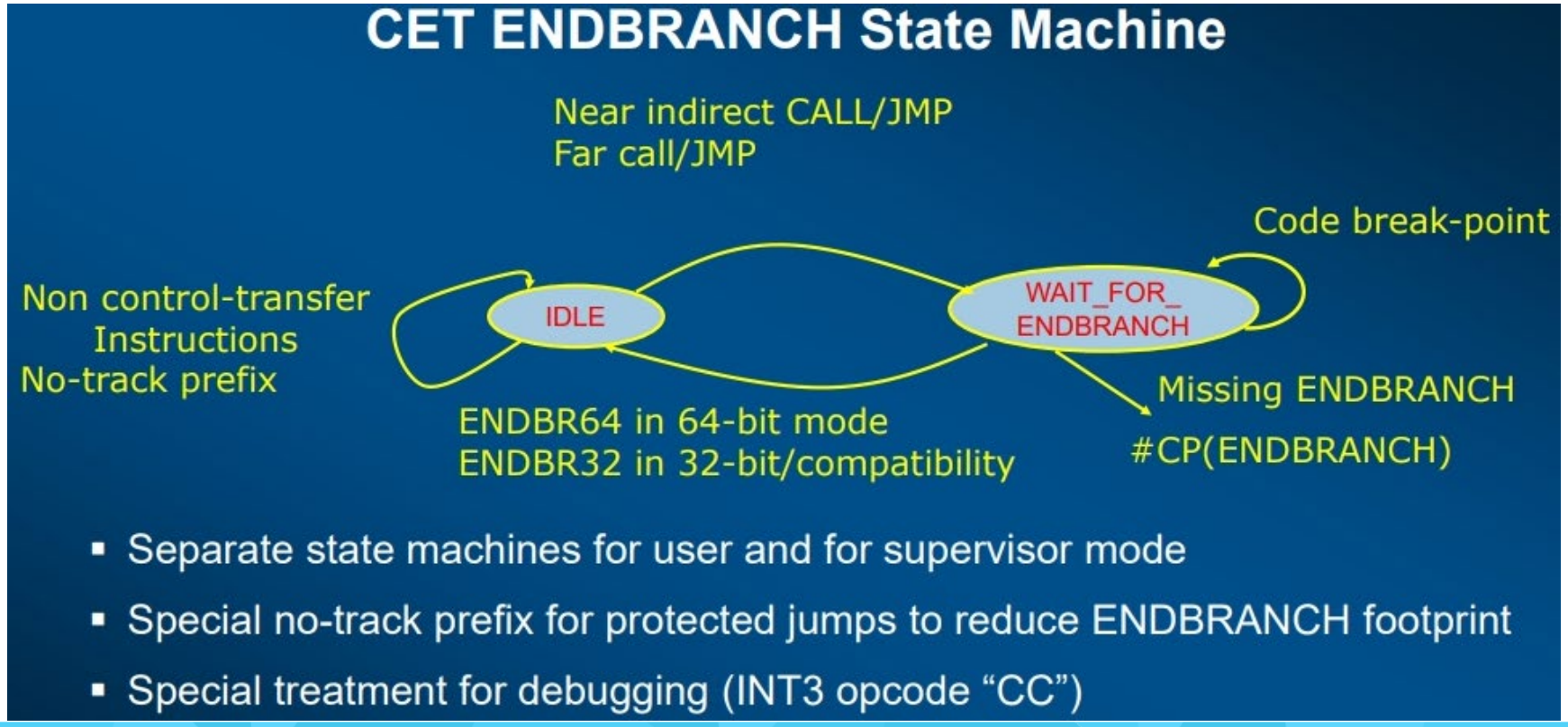

Shadow Stack Operation
Stack
Shadow Stack
ESP after ret
ESP after call
Return EIPn-1
Param 1
Param 2
Return EIPn
SSP after ret
SSP after call
Return EIPn-1
Return EIPn
SHADOW STACK
• Setup by OS/VMM
• Protected by new memory access control
• Different shadow stacks for each privilege level
Stack usage on near CALL
▪ Call
– pushes return address on both stacks
▪ No parameters passing on shadow stack
▪ Return
– pops return address from both stacks
– Controlflow Protection (#CP) exception in case the two return addresses don't match

- 控制流增强技术
  - CET ——Control-flow Enforcement Technology
  - 提高防御ROP/JOP/COP控制流攻击的技术
  - 间接分支跟踪（Indirect branch tracking）
  - 分支保护，以防止跳转/调用导向编程攻击

- 控制流增强技术-IBT(Indirect Branch Tracking)：ENDBRANCH
  - 通过编译器在合理的间接跳转（call/jmp）中用新的指令做标记，新指令包含 endbr32 和endbr64



- 控制流增强技术-IBT(Indirect Branch Tracking)：ENDBRANCH 状态机
  - CPU 在用户态和内核态分别设有一个 ENDBRANCH 状态机，状态机共有两个状态： IDLE 和WAIT_FOR_ENDBRANCH



- 随机化因子产生器
  - 由硬件协处理器生成随机化因子
  - 增强地址空间随机化，加密等

课后思考&动手

- 找一个处理器平台用户手册阅读
- 找一个开源安全解决方案，阅读文档，代码

内容概要

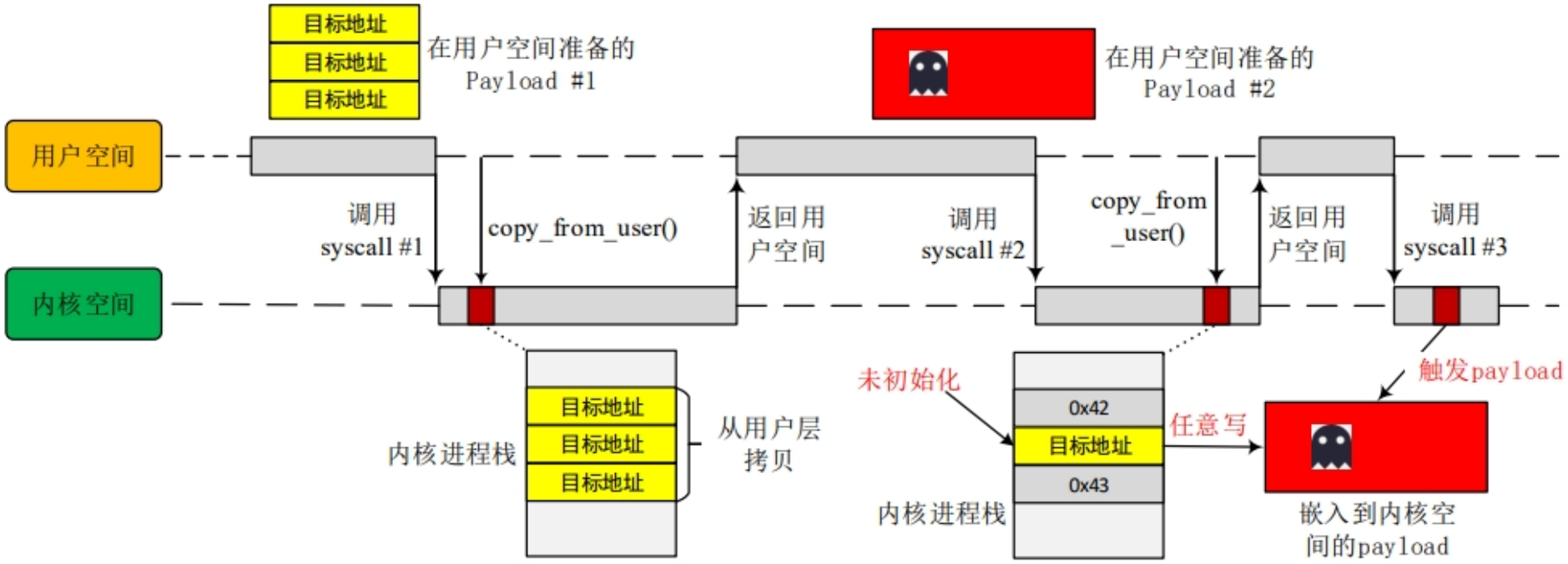
01 内存安全防护体系与防护技术

02 内存安全机制实现案例

内核栈格式化

- 攻击者通过多次系统调用，构造内核悬挂（空）指针，实现指向任意地址写

未初始化栈变量攻击



‘PAX_MEMORY_STACKLEAK’ 内核栈格式化

内核空间与用户空间数据拷贝安全审查

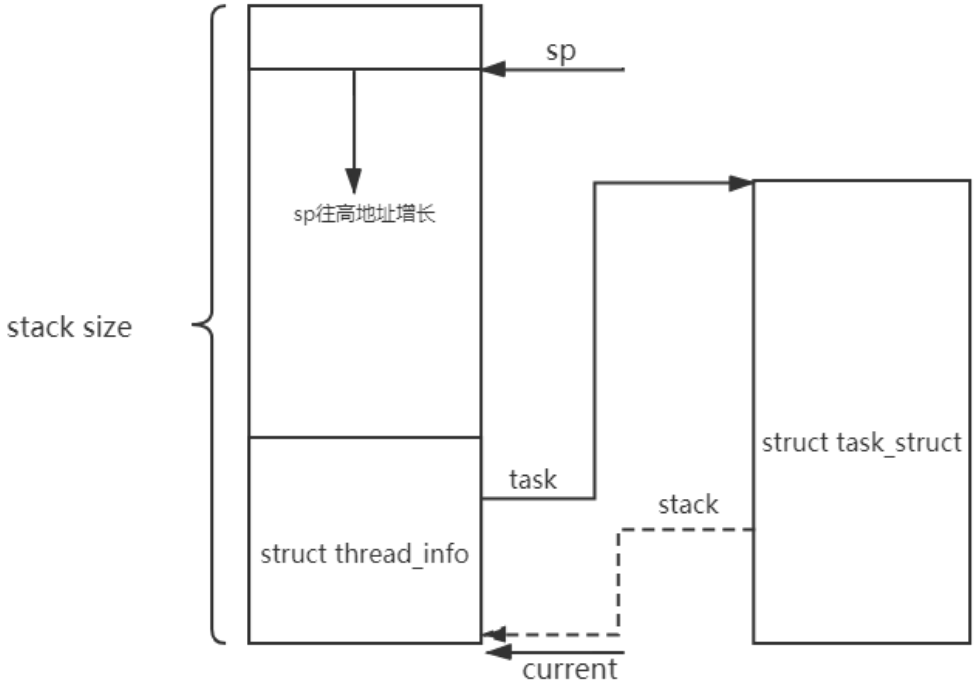
- 防止因内核与用户空间数据拷贝而引起的目的缓冲区溢出，堆溢出，添加了边界限制等相关检查和保护
  - **copy_to_user()**
  - **Copy_from_user()**
  - 主要针对堆溢出
  - 主要针对**SLAB**分配器进行了修改

‘PAX_USERCOPY’功能

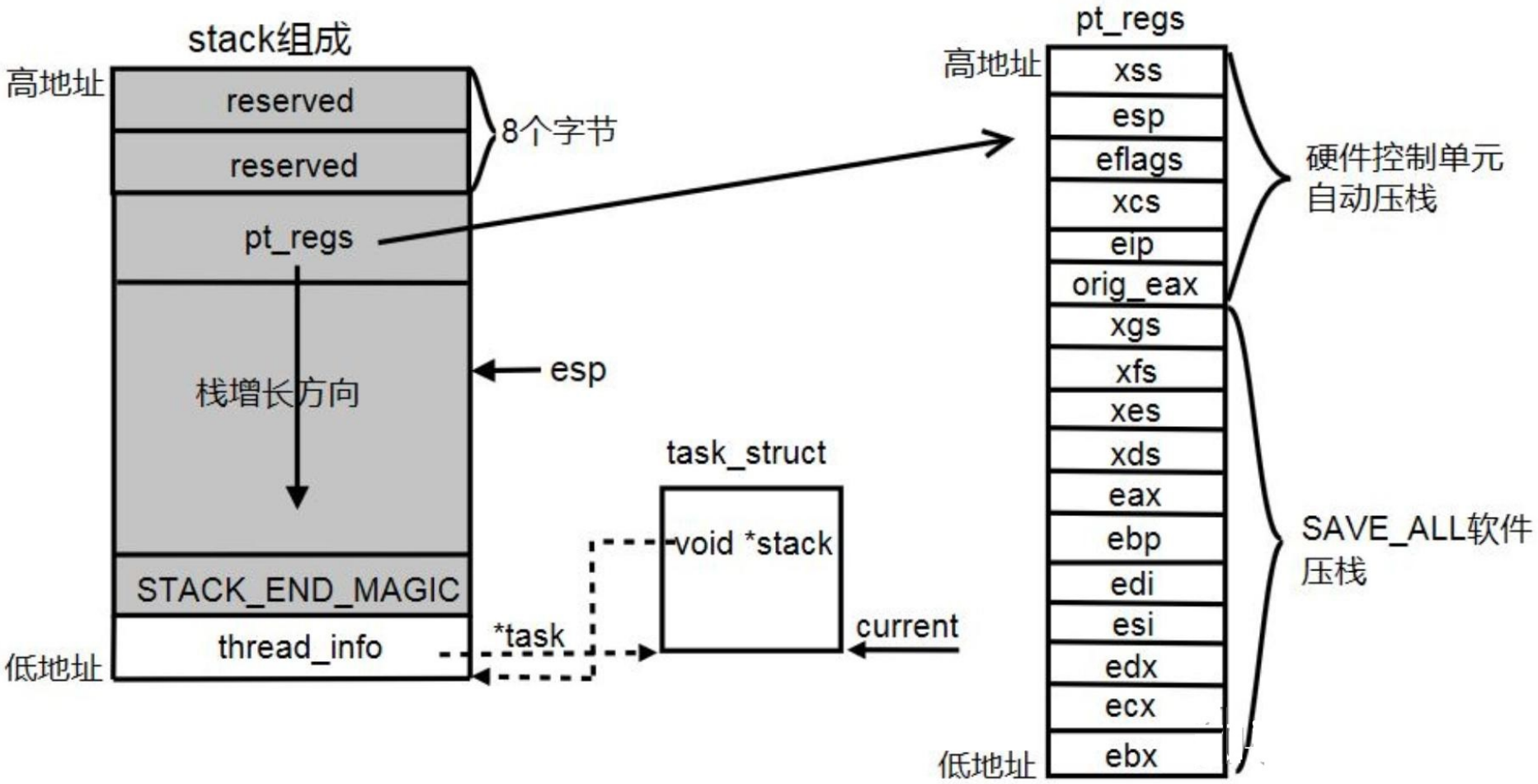
- 进程内核栈
  - 提供系统调用、中断等上下文的运行环境

linux进程内核栈定义：

```
union thread_union { struct
thread_info thread_info; unsigned
long
stack[THREAD_SIZE/sizeof(long)
]; };
```



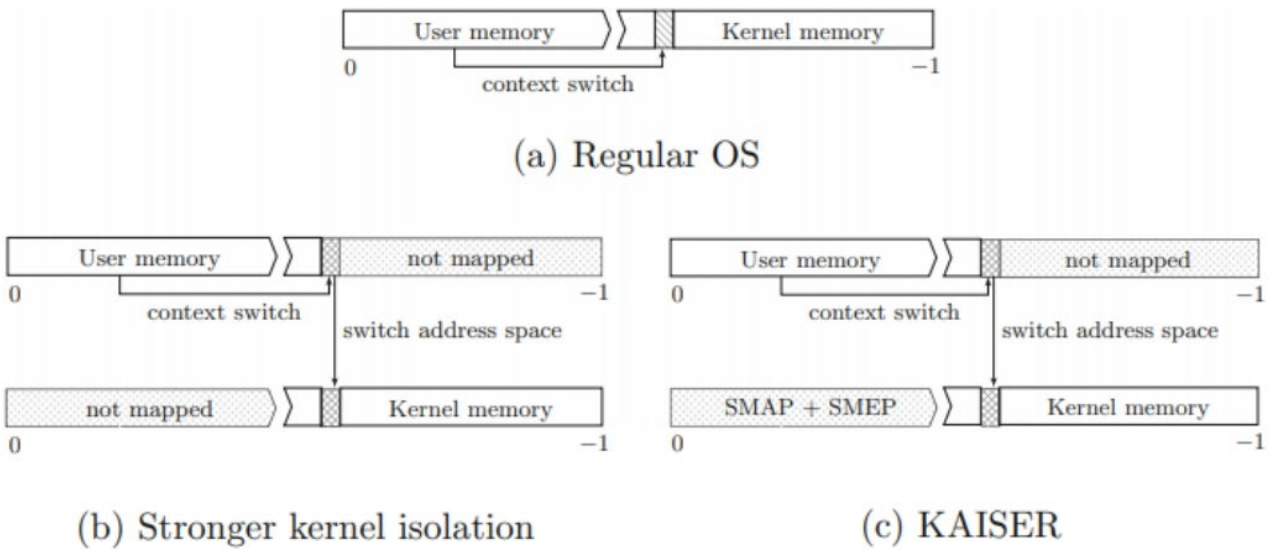
内核栈顶随机化


stack组成
高地址
reserved
reserved
pt_regs
栈增长方向
STACK_END_MAGIC
thread_info
低地址
8个字节
esp
*task
task_struct
void *stack
current
pt_regs
高地址
xss
esp
eflags
xcs
eip
orig_eax
xgs
xfs
xes
xds
eax
ebp
edi
esi
edx
ecx
ebx
低地址
硬件控制单元
自动压栈
SAVE_ALL软件
压栈

初始化后只读保护

- 内核核心数据结构初始化后只读保护
  - 系统调用表
  - 中断向量表
  - 内核模块描述符
  - ……

- 内核页表隔离
- **Kernel Page Table Isolation---KPTI**
  - 将原本在一套页表中用户层页表和内核页表分离，从而防止攻击者通过侧信道的方式来探测内核代码、数据的位置，增强内核的安全

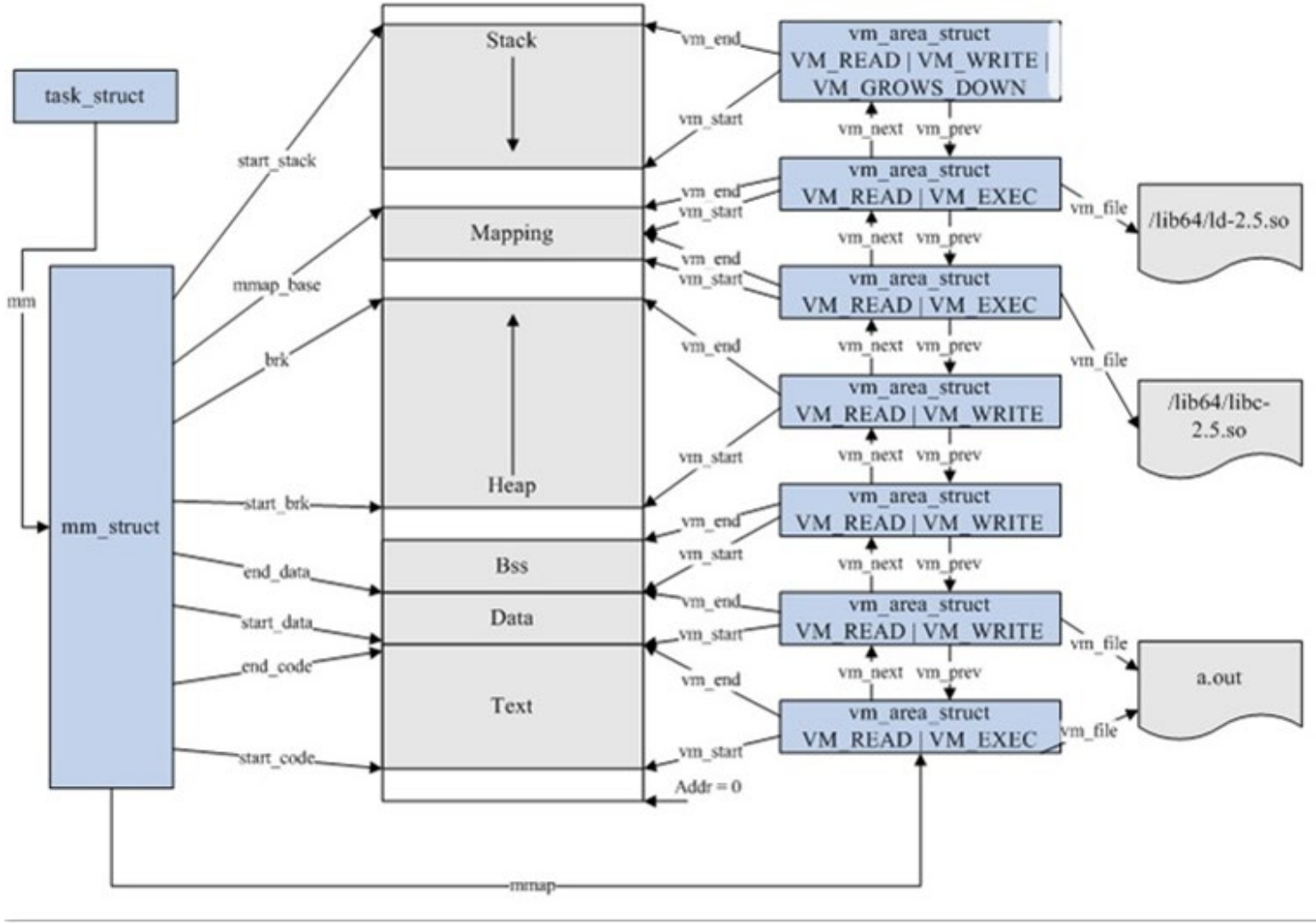


Linux系统地址空间随机化分配实现

- 随机化实现
  - 知识准备
  - 编译基本原理
  - ELF组织形式
  - 动态连接
  - Linux进程数据结构
  - 内存管理
  - Linux 进程地址空间组织
  - 平台体系结构
  - 可执行文件加载过程
  - 系统调用（exec,mmap……）
  - ……

Linux系统地址空间随机化实现架构

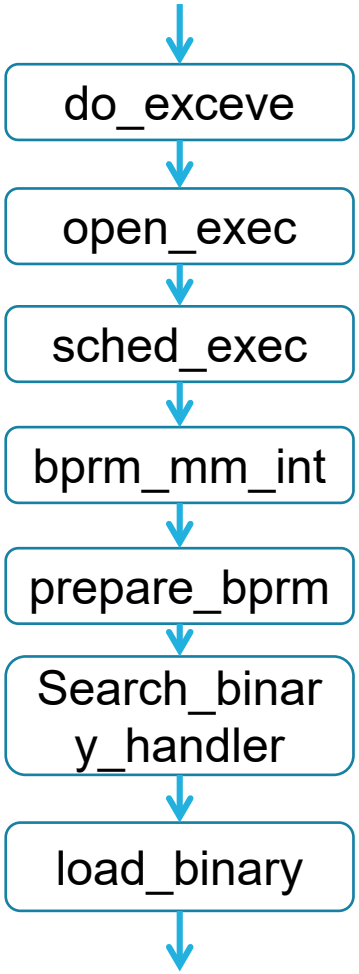
架构图



地址空间随机化分配运行效果

```
root@ubuntu:/home/abang# cat /proc/self/maps
08048000-08053000 r-xp 00000000 08:01 1054472    /bin/cat
08053000-08054000 r--p 0000a000 08:01 1054472    /bin/cat
08054000-08055000 rw-p 0000b000 08:01 1054472    /bin/cat
0925c000-0927d000 rw-p 00000000 00:00 0          [heap]
b7241000-b73a6000 r--p 001c8000 08:01 922740     /usr/lib/locale/locale-archive
b73a6000-b75a6000 r--p 00000000 08:01 922740     /usr/lib/locale/locale-archive
b75a6000-b75a7000 rw-p 00000000 00:00 0
b75a7000-b774a000 r-xp 00000000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b774a000-b774c000 r--p 001a3000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b774c000-b774d000 rw-p 001a5000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b774d000-b7750000 rw-p 00000000 00:00 0
b775f000-b7760000 r--p 005e0000 08:01 922740     /usr/lib/locale/locale-archive
b7760000-b7762000 rw-p 00000000 00:00 0
b7762000-b7763000 r-xp 00000000 00:00 0          [vdso]
b7763000-b7783000 r-xp 00000000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
b7783000-b7784000 r--p 0001f000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
b7784000-b7785000 rw-p 00020000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
bfac9000-bfaea000 rw-p 00000000 00:00 0          [stack]
root@ubuntu:/home/abang# echo 0 > /proc/sys/kernel/randomize_va_space
root@ubuntu:/home/abang# cat /proc/self/maps
08048000-08053000 r-xp 00000000 08:01 1054472    /bin/cat
08053000-08054000 r--p 0000a000 08:01 1054472    /bin/cat
08054000-08055000 rw-p 0000b000 08:01 1054472    /bin/cat
08055000-08076000 rw-p 00000000 00:00 0          [heap]
b7abc000-b7c21000 r--p 001c8000 08:01 922740     /usr/lib/locale/locale-archive
b7c21000-b7e21000 r--p 00000000 08:01 922740     /usr/lib/locale/locale-archive
b7e21000-b7e22000 rw-p 00000000 00:00 0
b7e22000-b7fc5000 r-xp 00000000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b7fc5000-b7fc7000 r--p 001a3000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b7fc7000-b7fc8000 rw-p 001a5000 08:01 660181     /lib/i386-linux-gnu/libc-2.15.so
b7fc8000-b7fcb000 rw-p 00000000 00:00 0
b7fda000-b7fdb000 r--p 005e0000 08:01 922740     /usr/lib/locale/locale-archive
b7fdb000-b7fdd000 rw-p 00000000 00:00 0
b7fdd000-b7fde000 r-xp 00000000 00:00 0          [vdso]
b7fde000-b7ffe000 r-xp 00000000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
b7ffe000-b7fff000 r--p 0001f000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
b7fff000-b8000000 rw-p 00020000 08:01 661146     /lib/i386-linux-gnu/ld-2.15.so
bffdf000-c0000000 rw-p 00000000 00:00 0          [stack]
```

可执行程序加载过程



读取可执行文件

选取最小负载CPU以执行程序

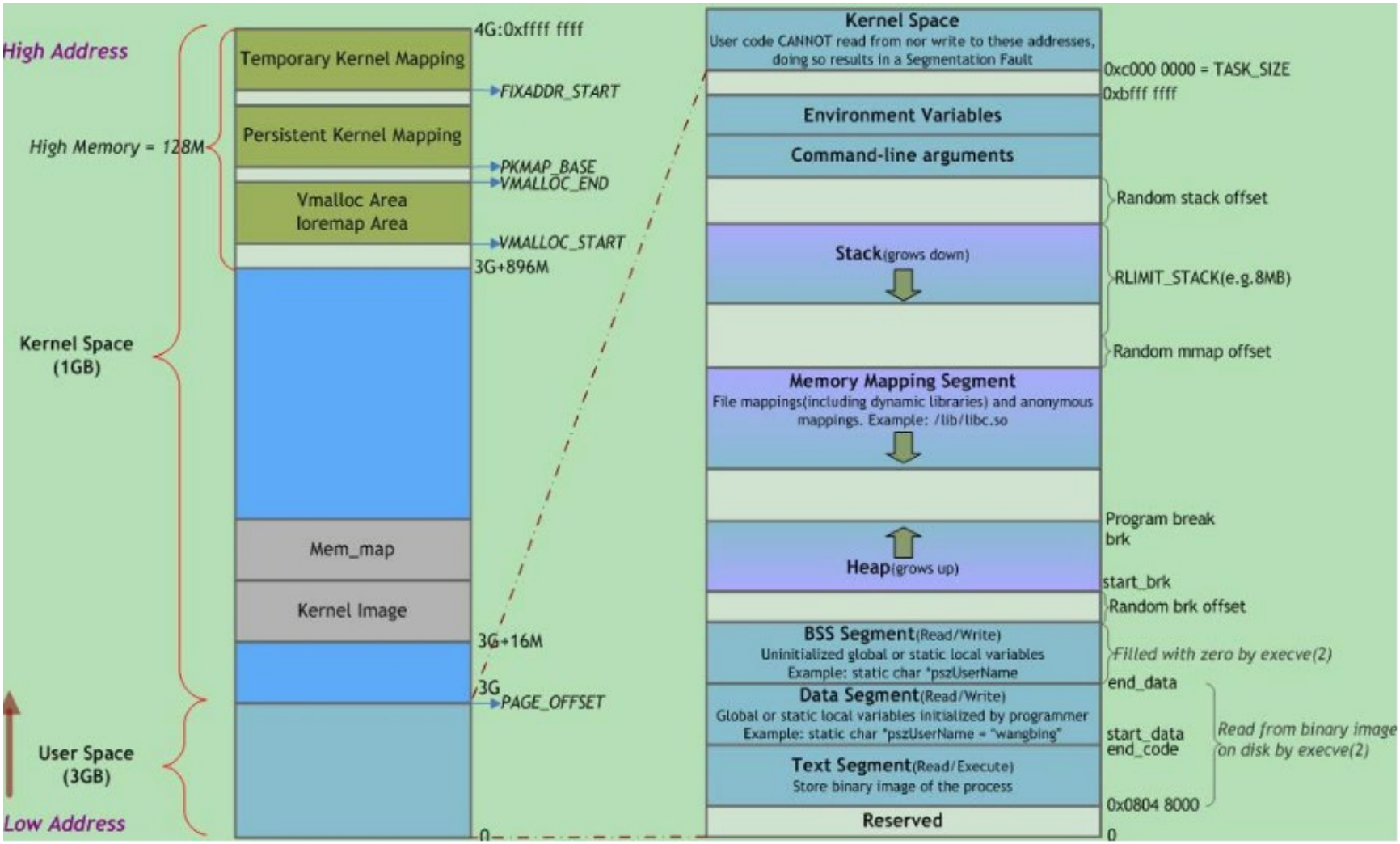
初始化内存管理

填充linux_binprm数据结构

Load_binary函数关键调用

- load_binary
  - 读取可执行文件头部，校验魔数，读取动态链接程序，读取可加载项，检查动态链接执行许可权……
  - arch_pick_mmap_layout
    - 决定进程线性区的布局，平台相关
  - setup_arg_pages
    - 为进程用户态分配一个新的线性区描述符
  - do_mmap
    - 创建一个新线性区对可执行文件正文段进行映射
  - do_brk
    - 创建一个新的匿名线性区来映射程序的bss段
  - start_thread
    - 处理内核态栈

进程空间随机化布局


High Address
High Memory = 128M
Kernel Space (1GB)
User Space (3GB)
Low Address
Temporary Kernel Mapping
Persistent Kernel Mapping
Vmalloc Area
Ioremap Area
Mem_map
Kernel Image
4G:0xffff ffff
FIXADDR_START
PKMAP_BASE
VMALLOC_END
VMALLOC_START
3G+896M
3G+16M
3G
PAGE_OFFSET
0
Kernel Space
User code CANNOT read from nor write to these addresses, doing so results in a Segmentation Fault
0xc000 0000 = TASK_SIZE
0xbfff ffff
Environment Variables
Command-line arguments
Random stack offset
Stack(grows down)
RLIMIT_STACK(e.g.8MB)
Random mmap offset
Memory Mapping Segment
File mappings(including dynamic libraries) and anonymous mappings. Example: /lib/libc.so
Program break
brk
Heap(grows up)
start_brk
Random brk offset
BSS Segment(Read/Write)
Uninitialized global or static local variables
Example: static char *pszUserName
Filled with zero by execve(2)
end_data
Data Segment(Read/Write)
Global or static local variables initialized by programmer
Example: static char *pszUserName = "wangbing"
start_data
end_code
Text Segment(Read/Execute)
Store binary image of the process
Read from binary image on disk by execve(2)
0x0804 8000
Reserved
0

PaX Linux内核增强补丁

- 支持不可执行
  - 页式管理
  - 段式管理
- 完全地址随机化映射
  - 每个系统调用的内核栈随机映射
  - 用户栈随机映射
  - ELF可执行映像随机映射
  - Brk()分配的heap随机映射
  - Mmap()管理的heap随机映射
  - 动态链接库随机映射

PaX Linux内核增强补丁

- 整数溢出保护
- 内核，用户空间数据拷贝保护
- 软件实现SMEP，SMAP（特定平台）
- 内核栈清零保护
- 内核页只读
  - Const结构只读
  - 系统调用表只读
  - 局部段描述符表（IDT）只读
  - 全局段描述符表（GDT）只读
- 数据页只读
- ……

Linux系统地址空间随机化分配开源实现

- 获取PaX
  - Docs: http://pax.grsecurity.net/docs/index.html
  - 补丁代码： http://pax.grsecurity.net/
- PaX缺点：不支持LKM,兼容性不好

课后思考&动手

- 找一个Linux系统，玩一下地址空间随机化分配等功能
- 给linux系统内核打PaX补丁，运行下，体验一下各个安全功能


Q&A