软件安全与漏洞分析

1.2 软件安全基础 -- 计算机存储空间原理

Previously in Software Security

- 软件安全的技术和现实意义
- 软件安全所面对的主要威胁来源
- 软件安全威胁的发展趋势

计算机存储空间原理

- 本节主题 -- 1. 软件在计算机系统内的形态？
  - 如何存储？
  - 如何加载？
- 本节主题 -- 2. 软件是怎样执行的？
  - 如何寻址？
  - 如何利用共享代码？

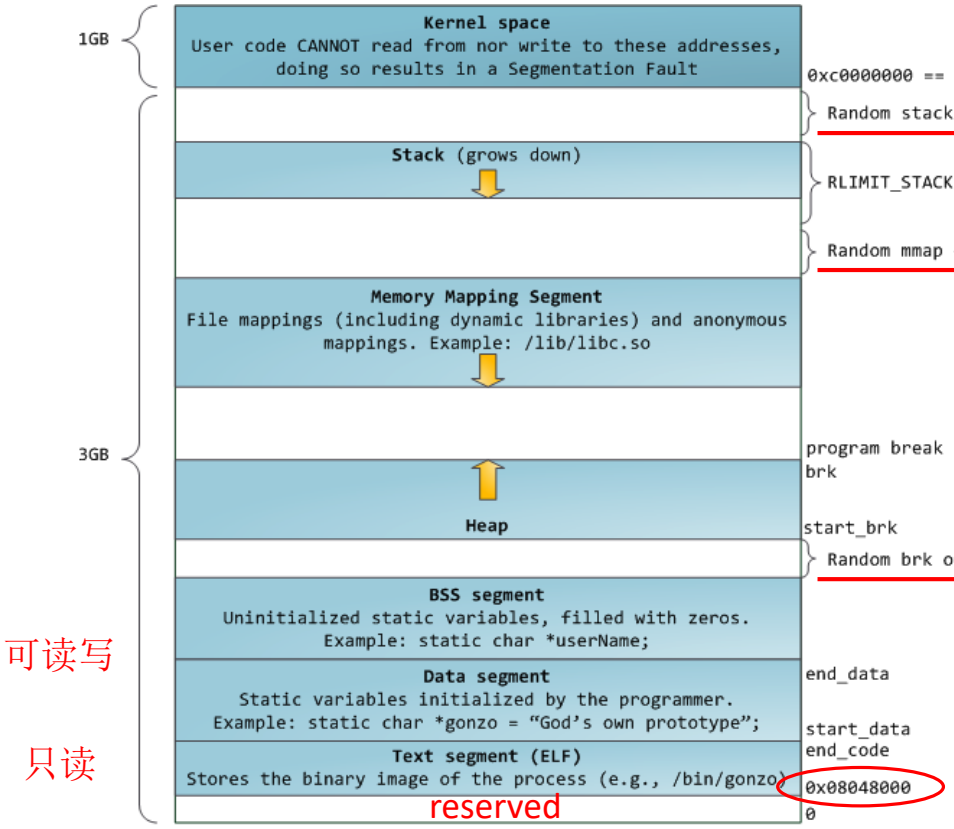
典型操作系统的内存布局

- 操作系统层面：以Linux和32位系统为例（why？）
  - 具有代表性
  - 衍生系统（Android）应用广泛
  - 大多数结论可以直接扩展至64位系统
- 底层：x86体系结构（why？）
  - 熟悉其汇编语言形式
  - 后续有关漏洞和攻击的实例中沿用

典型操作系统的内存布局

- 进程 -- 软件在操作系统中的基本形态
  - 具有独立的地址空间
  - 掌握着软件执行所需要的所有资源（或资源的位置）
- 32位操作系统的地址空间：最大4GB
  - 用户空间 --- 0-3GB
  - 内核空间 --- 3-4GB（独立于所有进程的用户空间）

典型操作系统的内存布局



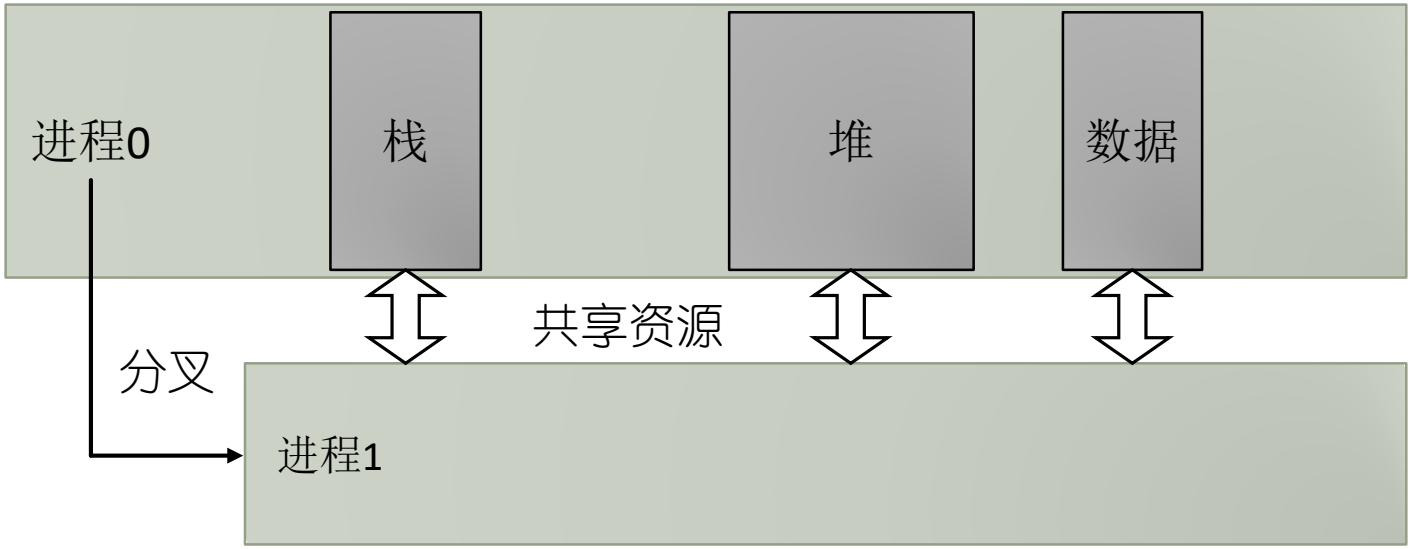
地址空间随机化 (ASLR)

- 分段的意义？
  - 缓存命中率高
  - 访问控制

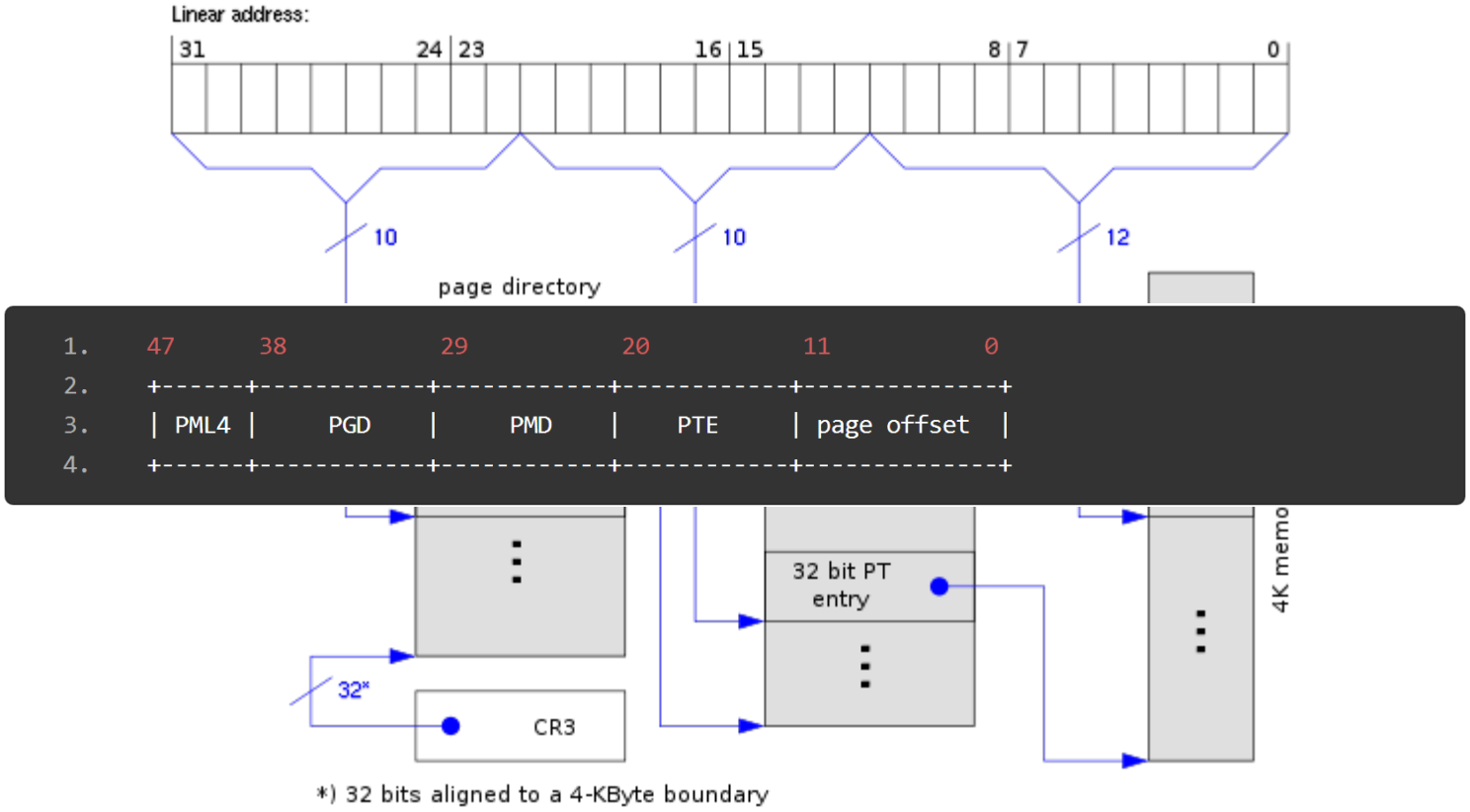
软件的加载

- 操作系统创建进程的步骤
  - 创建虚拟内存空间来容纳一个进程
  - 建立虚拟内存地址与可执行文件的映射关系表（根据文件头）
  - 初始化栈空间、堆空间
- 虚拟地址空间看上去并不够用？
  - “copy on write”（20%的程序执行了80%的时间）
  - 多级页表查询 + 缺页中断

软件的加载

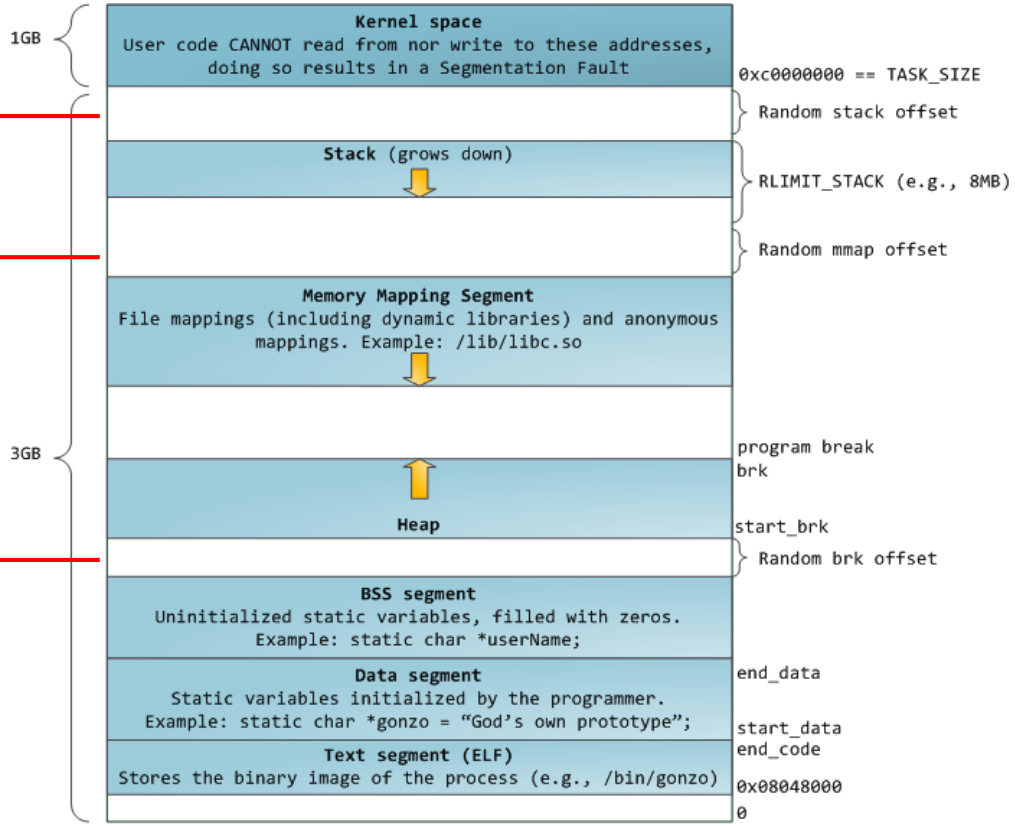


软件的寻址


Linear address:
31
24
23
16
15
8
7
0
10
10
12
page directory
47
38
29
20
11
0
PML4
PGD
PMD
PTE
page offset
32 bit PT entry
4K memo
32*
CR3
*) 32 bits aligned to a 4-KByte boundary

软件的寻址

最终会被内核判定为无效地址



可执行文件

- 另一个问题：共享代码
  - 不同的软件很可能会使用到同样的代码模块
  - 软件往往需要用到操作系统所提供的执行功能
- 怎么办：可链接的可执行文件
  - 将重复利用率很高的代码模块构造成共享库
  - 软件通过与共享库的链接使用这些公用代码

可执行文件

- 可用于代码共享的常见可执行格式：
  - **E**xecutable & **L**inkable **F**ormat（Linux系统）
  - **P**ortable **E**xecutable（Windows系列）
- 特点
  - 承担多个角色（程序本身，多种类型的共享库）
  - 地址空间可以被重定位（relocation）

可执行文件

```
There are 26 section header

Section Headers:
  [Nr] Name              Ty
  [ 0]                   NU
  [ 1] .interp           PR
  [ 2] .dynsym           DY
  [ 3] .dynstr           ST
  [ 4] .hash             HA
  [ 5] .rel.dyn          RE
  [ 6] .rel.plt          RE
  [ 7] .plt              PR
  [ 8] .text             PR
  [ 9] .note.android.ide PR
  [10] .ARM.extab        PR
  [11] .ARM.exidx        AR
  [12] .rodata           PR
  [13] .data.rel.ro.loca PR
  [14] .fini_array       FI
  [15] .init_array       IN
  [16] .preinit_array    PR
  [17] .data.rel.ro      PR
  [18] .dynamic          DY
  [19] .got              PROGBITS        0007fbb4 07ebb4 00044c 00
  [20] .data             PROGBITS        00080000 07f000 000744 00
  [21] .bss              NOBITS          00080748 07f744 008975 00
  [22] .comment          PROGBITS        00000000 07f744 000035 01
```

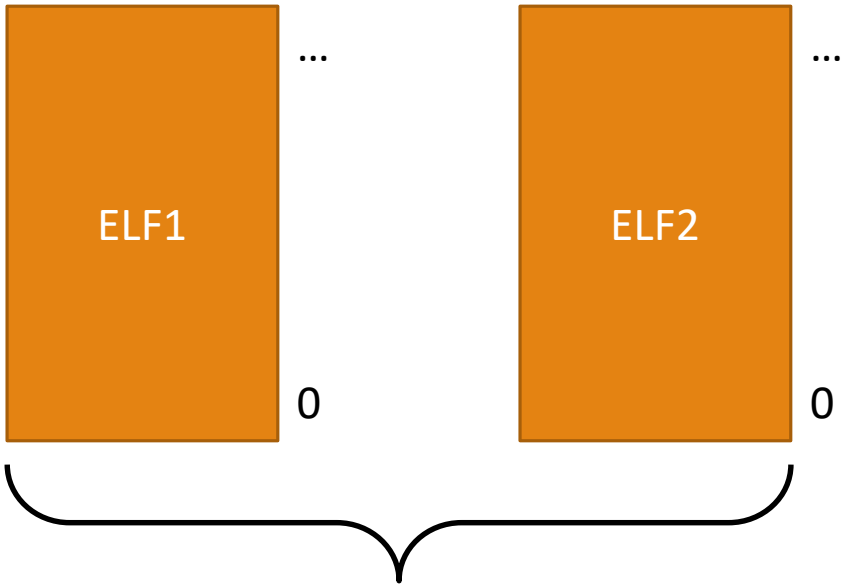
```
D:\>readelf.exe --segments android_server

Elf file type is DYN (Shared object file)
Entry point 0xbcf0
There are 8 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00000034 0x00000034 0x00100 0x00100 R   0x4
  INTERP         0x000134 0x00000134 0x00000134 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /system/bin/linker]
  LOAD           0x000000 0x00000000 0x00000000 0x7c9b8 0x7c9b8 R E 0x1000
  LOAD           0x07d028 0x0007e028 0x0007e028 0x0271c 0x0b095 RW  0x1000
  DYNAMIC        0x07eac0 0x0007fac0 0x0007fac0 0x000f0 0x000f0 RW  0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0
  EXIDX          0x0713ac 0x000713ac 0x000713ac 0x02b80 0x02b80 R   0x4
  GNU_RELRO      0x07d028 0x0007e028 0x0007e028 0x01fd8 0x01fd8 RW  0x8

Section to Segment mapping:
 Segment Sections...
  00
  01     .interp
  02     .interp .dynsym .dynstr .hash .rel.dyn .rel.plt .plt .text .note.android.ident .ARM.extab .ARM.exidx .rodata
  03     .data.rel.ro.local .fini_array .init_array .preinit_array .data.rel.ro .dynamic .got .data .bss
  04     .dynamic
  05
  06     .ARM.exidx
  07     .data.rel.ro.local .fini_array .init_array .preinit_array .data.rel.ro .dynamic .got
```

可执行文件

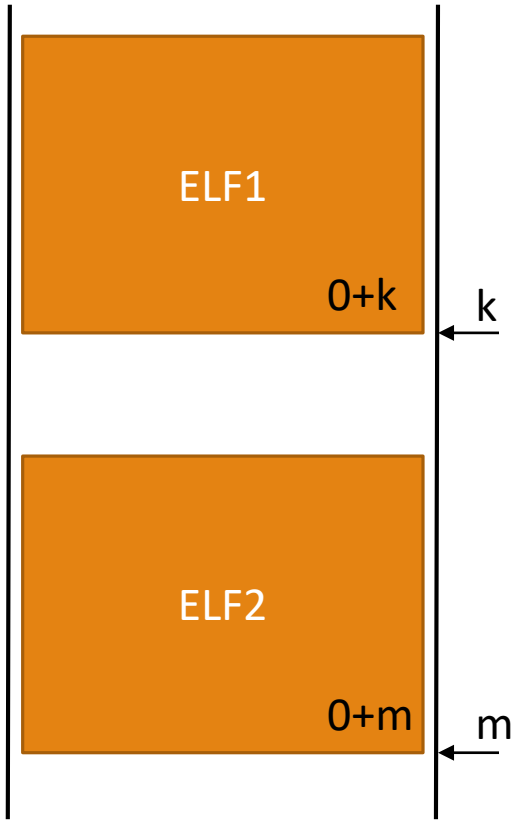
- ELF中与共享代码链接有关的重要section：
  - 符号表
  - 字符串表
  - 重定位表
  - 程序头部

```
choudan@ubuntu:~/coding/cpp$ readelf -r a.out

Relocation section '.rel.dyn' at offset 0x450 contains 2 entries:
 Offset     Info    Type                Sym.Value  Sym. Name
08049ff0  00000206 R_386_GLOB_DAT    00000000   __gmon_start__
0804a040  00000b05 R_386_COPY        0804a040   _ZSt4cout

Relocation section '.rel.plt' at offset 0x460 contains 8 entries:
 Offset     Info    Type                Sym.Value  Sym. Name
0804a000  00000107 R_386_JUMP_SLOT   00000000   __cxa_atexit
0804a004  00000207 R_386_JUMP_SLOT   00000000   __gmon_start__
0804a008  00000407 R_386_JUMP_SLOT   00000000   _ZNSt8ios_base4InitC1E
0804a00c  00000507 R_386_JUMP_SLOT   00000000   __libc_start_main
0804a010  00000a07 R_386_JUMP_SLOT   08048520   _ZNSt8ios_base4InitD1E
0804a014  00000607 R_386_JUMP_SLOT   00000000   _ZStlsISt11char_traits
0804a018  00000707 R_386_JUMP_SLOT   00000000   _ZNSolsEPFRSoS_E
0804a01c  00000907 R_386_JUMP_SLOT   08048550   _ZSt4endlIcSt11char_tr
```

重定位



地址一样，怎么共用……？



What’s next?

- 常见软件漏洞及其原理
  - 缓冲区溢出
  - 格式化字符串漏洞