

动态链接

内涵 — Dynamic Linking

使用原因 — 有效利用内存和磁盘资源 — 实现指令共享 — .so 中的指令只要在内存中出现一次 — 多个进程中不同虚拟空间映射的是同一块物理内存
— 更加方便地维护、升级程序

生成 跑的时候还要链接的可执行文件 — 内涵 — .o 和 .so 进行特殊的链接, 生成一个可执行文件 — 内涵 — 在解析目标文件引用的符号时 — 如果符号出现在共享对象的符号表中 — 这个符号被标记为动态链接符号
— 否则 — 这个按照静态链接的规则 — 重定位对应的符号引用

问题 — 绝对地址引用

装载时重定位 (Load Time Relocation)

内涵 — 在装载时重定位.so 文件中的符号 — 各个 .so 文件先确定各个段的地址, 再分配 .so 中符号的地址和可执行文件中符号的地址
特性 — 共享对象中的同一个指令, 因为其中有装载时重定位的符号, 在不同进程中实际上不同, 即共享文件在内存中的副本数量等于进程的数量 — 没法做到指令共享

解决问题 — 共享目标文件也有引用其它文件的符号 — 这些符号需要在程序运行到此之前, 确定其值或地址

内涵 — PIC, Position-independent Code

在编译的时候 — 原来需要被修改的地方, 即需要知道符号的虚拟地址的地方 — 代码段的函数的调用 — 不需要重定位
— 数据段 — static 数据 — 不需要重定位
— 指针变量 — 还是需要重定位 — 在装载的时候, .got 里的符号就被绑定了

调用函数 — 内涵 — 使用相对地址调用指令 — e8 <offset>

本质 — 得到下一条指令的绝对地址, 根据该绝对地址和数据存放位置的偏差, 引用

引用数据

内涵 — call 494 <_i686.get_pc_thunk.cx> — _i686.get_pc_thunk.cx: mov ecx (esp) ret
— add ecx 0x118c
— 将下一条指令的地址赋给ecx
— ecx + 0x118c 得到数据的地址

段名称 — 分成两个 — .got — 放变量地址
— .got.plt — 放函数地址

本质 — 一个指针数组

内涵 — GOT, Global Offset Table
— 指令访问变量、函数 b, 会访问 b@got — GOT 中存放着每个在外部的全局变量的地址
— 装载的时候会填充 GOT — 即装载的时候就完成地址绑定

位置 — 数据 segment — 放在 .data 后面

段名称 — .plt — 放的是代码

内涵 — PLT, Procedure Linkage Table

第一次访问符号的时候, 进行地址绑定 — 调用 ld.so 的 _dl_runtime_resolve 函数
— _dl_run_time_resolve 会把真正的地址写入 b@got

内涵 — 根据 b@got 和下一条指令的绝对地址的偏差, 来访问 b@got — 从而访问全局变量 b

立即绑定 — 内涵 — 直接用 got — 根据 func@got 和下一条指令的绝对地址的偏差, 来访问 func@got — 从而得到 func 的地址

延迟绑定 (Lazy Binding) — 内涵 — got fun@got 的值 初始是 fun@plt 的 push n 的地址
— 调用 想要 call fun, 所以 call fun@plt — 第一次 call fun, 会调用 fun@plt 的 push n 等 — _dl_run_time_resolve 会把 fun 真正的地址写入 fun@got
— 第二次 call fun, 就真的可以通过 call fun@plt 或者直接通过 call fun@got 来调用了

地址无关可执行文件 (PIE) — 内涵 — PIE, Position-Independent Executable

解决策略

地址无关代码

类型

内部

外部

全局偏移表 (GOT)

进程可链接表 (PLT)

引用数据

调用函数

问题及解决

装载, 启动进程 — 特殊步骤 — 但是最后并不是将控制权交给可执行文件的入口地址 — 而是启动动态链接器 (Dynamic Linker) ld.so — 将 ld.so 装载进入当前进程的虚拟地址空间

Id.so 进行链接工作 — 启动动态链接器 — 自举
— 装载所有需要的共享对象 — 将可执行文件的符号表和动态链接请的符号表合并成 — 全局符号表 (Global Symbol Table)
— 链接器根据 .dynamic 段查找可执行文件依赖的共享对象
— 重定向和初始化 — 此时动态链接器已经拥有了进程的符号表 — 据此进行重定位
— 如果某个文件有 .init 段, 就执行 .init

所有链接工作都完成 — 将控制权交给可执行文件的入口地址

共享库 (Shared Library) — 内涵 — 共享库就是普通的共享对象 — 共享对象的形式就非常合适构造库 — 广义上可以将二者视为同一概念