# [译]如何在任意进程中修改内存保护属性

hellocaicaic**** / 2018-12-02 08:20:00 / 浏览数 2909 

---

最近，我们面临着一个非常特殊的任务：在任意进程中更改内存区域的保护标志。
这项任务看似微不足道，但是我们遇到了一些障碍，并在此过程中学到了新的东西，主要是关于Linux机制和内核开发。
以下是我们工作的简要概述，包括我们采取的三种方法以及每次寻求更好解决方案的原因。

## 引言

在现代操作系统中，每个进程都有自己的虚拟地址空间（从虚拟地址到物理地址的映射）。
此虚拟地址空间由内存页（某些固定大小的连续内存块）组成，每个页都有保护标志，用于确定此页面允许访问的类型（读取，写入和执行）。
这种机制依赖于体系结构页面表（有趣的事实是：在x64体系结构中，你不能使内存页只写，即使你特意从操作系统申请 - 它也总是可读的）。

在Windows中，您可以使用API函数VirtualProtect或VirtualProtectEx更改内存区域的保护标志。
后者使我们的任务变得非常简单：它的第一个参数hProcess是“要改变其内存保护的进程的句柄”。

另一方面，在Linux中，我们并不那么幸运：更改内存保护的API是系统调用mprotect或pkey_mprotect，并且两者始终在当前进程的地址空间上运行。
我们现在回顾一下在x64架构上的Linux中解决此任务的方法（我们假设是root权限）。

## 方法一：代码注入

好吧，如果mprotect总是作用于当前进程，我们需要让目标进程从它自己的上下文中调用它， 这称为代码注入。它可以通过许多不同的方式实现。
我们选择使用ptrace机制实现它，它允许一个进程“观察并控制另一个进程的执行”，包括更改目标进程的内存和寄存器的能力。
此机制用于调试器（如gdb）和跟踪实用程序（如strace）。 使用ptrace注入代码所需的步骤概述：

1. 使用ptrace附加到目标进程。 如果进程中有多个线程，那么停止所有其他线程也是明智之举。
2. 找到一个可执行的内存区域（通过检查 /proc/PID/maps），并在那里写操作码系统调用（hex：0f 05）。
3. 根据调用约定修改寄存器：首先，将rax更改为mprotect的系统调用号（即10）。
   然后，三个参数（它们分别是起始地址，长度和所需的保护标志）分别存储在rdi，rsi和rdx中。最后，将rip更改为步骤2中使用的地址。
4. 系统调用返回后恢复进程（ptrace允许您跟踪系统调用的进入和退出）。
5. 恢复被覆盖的内存和寄存器，从进程中分离并恢复正常执行。

这种方法是我们的第一个也是最直观的方法，并且在我们发现Linux中的另一种机制-seccomp 之前工作得很好。 seccomp
是Linux内核中的一个安全工具，允许进程将自己输入某种监牢里，除了read，write，_exit和sigreturn之外，它不能调用任何系统调用。
还可以选择指定任意系统调用及其参数以过滤它们。（译者注：seccomp 是 Linux 内核从2.6.23版本引入的一种简洁的沙箱机制

因此，如果进程启用了seccomp模式并且我们尝试在进程中调用 mprotect，那么内核将终止进程。因为不允许使用此系统调用。
我们希望能够对这些流程采取行动，因此寻求更好的解决方案仍在继续.....

## 方法二：模拟内核模块中的mprotect

seccomp 从进程的用户模式中防止了之前的解决办法，因此下一个方法肯定存在于内核模式中。
在Linux内核中，每个线程（用户线程和内核线程）都由名为task_struct的结构表示，并且当前线程（任务）可通过指针 current 访问。 内核中 mprotect
的内部实现就是使用指针current，所以我们首先想到的是，让我们将 mprotect
的代码复制粘贴到我们的内核模块，并用指向我们目标线程的task_struct的指针替换每次出现的 current

好吧，正如你可能已经猜到的那样，复制C代码并不是那么简单 - 大量使用我们无法访问的未导出的函数，变量和宏。
某些函数声明在头文件中导出，但内核不会导出它们的实际地址。 但是如果内核是用kallsyms支持编译的，那么这个特定的问题就可以解决，通过文件
/proc/kallsysm 导出它的所有内部符号。

尽管存在这些问题，我们仍尝试仅运行mprotect的实际内容，甚至仅用于学习目的。
因此，我们开始编写一个内核模块，它获取目标PID和参数以进行mprotect，并模仿其行为。 首先，我们需要获取所需的内存映射对象，它表示线程的地址空间：

```
/* Find the task by the pid */
   pid_struct = find_get_pid(params.pid);
   if (!pid_struct)
       return -ESRCH;

   task = get_pid_task(pid_struct, PIDTYPE_PID);
   if (!task) {
       ret = -ESRCH;
       goto out;
   }

   /* Get the mm of the task */
   mm = get_task_mm(task);
```

```
    if (!mm) {
        ret = -ESRCH;
        goto out;
    }

    …
    …

out:
    if (mm) mmput(mm);
    if (task) put_task_struct(task);
    if (pid_struct) put_pid(pid_struct);
```

现在我们有了内存映射对象，我们还需要深入挖掘。 Linux内核实现了一个抽象层来管理内存区域，每个区域由结构vm_area_struct表示。
为了找到正确的内存区域，我们使用函数find_vma，它通过所需的地址搜索内存映射。

vm_area_struct包含字段vm_flags，其以与体系结构无关的方式表示存储器区域的保护标志，以及vm_page_prot，其以体系结构相关的方式表示它。
单独更改这些字段不会真正影响页表（但会影响 /proc/PID/maps的输出，我们尝试过它！）。 你可以在这里读更多关于它的内容。

在阅读并深入研究内核代码之后，我们检测到真正改变内存区域保护所需的最基本工作：

1. 将字段vm_flags更改为所需的保护标志。
2. 调用函数vma_set_page_prot_func以根据vm_flags字段更新字段vm_page_prot。
3. 调用函数change_protection_func实际更新页表中的保护位。

这段代码有效，但它有很多问题 - 首先，我们只实现了mprotect的基本部分，但原始函数比我们做的要多得多（例如，通过保护标志分割和连接内存区域）。
其次，我们使用两个内核函数，这些函数不是由内核导出的（vma_set_page_prot_func和change_protection_func）。
我们可以使用kallsyms来调用它们，但是这很容易出现问题（将来可能会更改它们的名称，或者可能会改变内存区域的整个内部实现）。
我们想要一个更通用的解决方案，不考虑内部结构，因此继续寻求更好的解决方案...

## 方法三：使用目标进程的内存映射

这种方法与第一种方法非常相似 - 我们希望在目标进程的上下文中执行代码。
在这里，我们在自己的线程中执行代码，但是我们使用目标进程的“内存上下文”，这意味着：我们使用它的地址空间。

通过几个API函数可以在内核模式下更改地址空间，我们将使用use_mm。 正如文档明确指出的那样，“此例程仅用于从内核线程上下文中调用”。
这些是在内核中创建的线程，不需要任何用户地址空间，因此可以更改其地址空间（地址空间内的内核区域在每个任务中以相同的方式映射）。

在内核线程中运行代码的一种简单方法是内核的工作队列接口，它允许您使用特定例程和特定参数来安排工作。 我们的工作例程非常小 -
它获取所需进程的内存映射对象和mprotect的参数，并执行以下操作（do_mprotect_pkey是内核中实现mprotect和pkey_mprotect系统调用的内部函数）：

```
use_mm(suprotect_work->mm);
suprotect_work->ret_value = do_mprotect_pkey(suprotect_work->start,
                                             suprotect_work->len,
                                             suprotect_work->prot, -1);
unuse_mm(suprotect_work->mm);
```

当我们的内核模块在某个进程（通过一个特殊的IOCTL）获得更改保护的请求时，它首先找到所需的内存映射对象（正如我们在前面的方法中所解释的那样）然后只使用正确

这个解决方案仍有一个小问题 - 函数do_mprotect_pkey_func不由内核导出，需要使用kallsyms获取。
与前一个解决方案不同，这个内部函数不太容易发生变化，因为它与系统调用pkey_mprotect有关，而且我们不处理内部结构，因此我们可以称之为“小问题”。

我们希望您在这篇文章中找到一些有趣的信息和技巧。 如果您有兴趣，可以在我们的github中找到这个概念验证内核模块的源代码。

## 原文

https://perception-point.io/2018/11/20/linux-internals/


上一篇：优步漏洞悬赏：将self-xss变... 下一篇：3ve-网络虚假广告攻击事件分析
1. 0 条回复
   - 动动手指，沙发就是你的了！

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板