Hooking linux内核函数（一）：寻找完美解决方案

Edvison / 2018-10-21 21:54:23 / 浏览数 2164 技术文章 技术文章 顶(0) 踩(0)

本文是《Hooking Linux Kernel Functions, Part 1: Looking for the Perfect Solution》的翻译文章。

前言

我们最近参与了一个Linux系统安全相关项目，需要hooking几个重要的Linux内核函数调用，例如打开文件和启动进程，并利用它来启用系统活动监控并抢先阻止可疑进程。

最后，我们发明了一种有效的方法，用于通过名称来hook内核中的任何函数，并在ftrace（Linux内核跟踪功能）的帮助下围绕其调用执行代码。
在这个由三部分组成的系列文章的第一部分中，描述了在提出新解决方案之前我们尝试hookLinux内核函数的四种方法。 并详细介绍了每种方法的主要优缺点。

四种可能的解决方法

可以尝试这几种方法来拦截关键的Linux内核函数：

- 使用Linux安全API
- 修改系统调用表
- 使用kprobes
- 拼接
- 使用ftrace处理程序、

下面，我们将详细讨论每个内核选项。

使用Linux安全API

起初，我们认为使用Linux安全API的hook函数是最佳选择，因为这个接口就是为此而设计的。
内核代码的关键点包含安全函数调用，这些调用可能导致安全模块安装的回调。该模块可以研究特定操作的上下文，并决定是允许还是禁止它。

不幸的是，Linux Security API有两个主要限制：

- 安全模块无法动态加载，因此我们需要重新编译内核，因为这些模块是其中的一部分。
- 除了个别例外，系统不能有多个安全模块。

虽然内核开发人员对系统是否可以包含多个安全模块有不同的看法，但是模块无法动态加载是可以肯定的事实。
为了确保系统从一开始就保持安全，安全模块必须是内核的一部分。

因此，为了使用Linux安全API，我们需要构建一个定制的Linux内核，并与AppArmor或SELinux集成额外的模块，后者在流行的发行版中使用。然而，这个选项并不适合我

修改系统调用表

由于监控主要用于由用户应用程序执行的操作，所以我们可以在系统调用级别上实现它。 所有Linux系统调用处理程序都存储在sys_call_table表中。
更改此表中的值会导致更改系统的行为。 因此，我们可以通过保存旧的处理程序值并将自己的处理程序添加到表中来hook任何系统调用。

这种方法也有一些优点和缺点。 更改系统调用表中的值的主要优点如下：

- 完全控制所有系统调用，作为用户应用程序的唯一内核接口。 因此，你不会错过用户进程执行的任何重要操作。
- 轻微的性能开销。 在更新系统调用表时有一次性投资。 另外两个开销是不可避免的监视有效负载和我们需要调用原始系统调用处理程序的额外函数调用。
- 次要内核要求。 理论上，这种方法几乎可以用于任何系统，因为您不需要特定的内核功能来修改系统调用表。

不过，这种方法也有几个缺点：
技术实施较复杂。 虽然替换表中的值并不困难，但还有一些额外的任务需要某些条件和一些不明显的解决方案：

- 查找系统调用表
- 绕过表的内存区域的内核写保护
- 确保更换过程的安全性能
  解决这些问题意味着开发人员必须浪费更多时间来实现，支持和理解该过程。

有些处理程序无法替换。在4.16版本之前的Linux内核中，x86_64体系结构的系统调用处理有一些额外的优化。其中一些优化要求在汇编中实现系统调用处理程序。这些类型

只hook系统调用。由于此方法允许你替换系统调用处理程序，因此它极大地限制了入口点。
所有的附加检查都可以在系统调用之前或之后立即执行，我们只有系统调用参数及其返回值。
因此，有时我们可能需要仔细检查进程的访问权限和系统调用参数的有效性。 此外，在某些情况下，需要复制两次用户进程内存会产生额外的开销费用。
例如，当参数通过指针传递时，会有两个副本：一个是你自己创建的副本，另一个是由原始处理程序创建的。
有时，系统调用还提供低粒度的事件，因此还可能需要应用其他过滤器来消除噪音。

首先，我们尝试更改系统调用表，以便我们可以覆盖尽可能多的系统，我们甚至成功实现了这种方法。
但是x86_64体系结构有几个特定的功能，还有一些我们不知道的钩子调用限制。 确保支持与启动特定新进程相关的系统调用 - clone()和execve() - 对我们来说至关重要。 这就是我们继续寻找其他解决方案的原因。

使用Kprobes

我们剩下的选择之一是使用Kprobes - 一种专为Linux内核跟踪和调试而设计的特定API。
Kprobes允许你为任何内核指令以及函数入口和函数返回处理程序安装预处理程序和后处理程序。 处理程序可以访问寄存器并可以更改它们。
这样，我们就有机会监控工作流程并改变它。

使用Kprobes跟踪Linux内核函数的主要好处如下：

- 一个成熟的API。 Kprobes自2002年以来一直在不断改进。该实例程序具有良好的文档化接口，并且已经发现和处理了大多数缺陷。
- 跟踪内核中任何一点的可能性。 Kprobes通过嵌入在可执行内核代码中的断点（int3指令）实现。
  因此，只要知道其位置，就可以在任何函数的任何部分中设置跟踪点。
  另外，你可以通过切换堆栈上的返回地址并跟踪任何函数的返回来实现kretprobes（除了那些根本不返回控制的函数）。

然而，Kprobes也有其缺点：
技术复杂性。 Kprobes只是在内核中的特定位置设置断点的工具。
要获取函数参数或局部变量值，你需要知道堆栈的确切位置以及它们所在的寄存器，并手动将它们移出。
此外，要阻止函数调用，还需要手动修改进程的状态，这样就可以让它认为它已经从函数返回了控制权。

Jprobes已被弃用。Jprobes是一个专门的kprobes版本，旨在使执行Linux内核跟踪变得更容易。
Jprobes可以从寄存器或堆栈中提取函数参数并调用你的处理程序，但处理程序和跟踪函数应该具有相同的签名。
唯一的问题是jprobes已被弃用，并已从最新的内核中删除。

开销太大。 即使这是一次性程序，定位断点也是非常昂贵的。虽然断点不影响其余功能，但它们的处理也相对昂贵。
幸运的是，通过使用为x86_64架构实现的跳转优化，可以显着降低使用kprobes的成本。 但是，kprobes的成本超过了修改系统调用表的成本。

Kretprobes的局限性。kretprobes功能是通过替换堆栈上的返回地址来实现的。为了在处理结束后回到原始地址，kretprobes需要在某个地方保留原始地址。
地址存储在固定大小的缓冲区中。 如果缓冲区过载，例如当系统执行跟踪函数的同时调用太多时，kretprobes将跳过某些操作。

禁用抢占。Kprobes基于处理器寄存器的中断和故障。因此，为了执行同步，所有处理程序都需要以禁用的抢占方式执行。因此，处理程序有几个限制:你不能在其中等待，

不过，如果只需要跟踪函数内部的特定指令，那么kprobes肯定是有用的。

拼接

还有一种配置内核函数hooking的经典方法：将函数开头的指令替换为通向处理程序的无条件跳转。
原始指令被移动到不同的位置，并在跳回到截取的函数之前执行。 因此，只需两次跳转，就可以将代码拼接成一个函数。

此方法的工作方式与kprobes跳转优化的方式相同。使用拼接，也可以获得与使用kprobes相同的结果，但开销更低，并且可以完全控制流程。

使用拼接的优点非常明显：

- 内核的最小需求。拼接不需要内核中的任何特定选项，可以在任何函数的开头实现。 你需要的只是函数的地址。
- 最低成本。跟踪代码只需执行两次无条件跳转即可将控制权移交给处理程序并收回控制权。这些跳转很容易为处理器预测，而且相当便宜。

然而，这种方法有一个主要缺点 - 技术复杂性。 更换函数中的机器代码并不容易。 为了使用拼接，你需要完成以下几项操作：

- 同步挂钩安装和删除（如果在更换指令期间调用了该函数）
- 使用可执行代码绕过内存区域的写保护
- 替换指令后使CPU缓存失效
- 拆卸已替换的指令，以便将它们作为一个整体进行复制
- 检查替换后的函数是否没有跳转
- 检查替换后的函数是否可以移动到其他位置

当然，你也可以使用livepatch框架并查看kprobes的一些提示，但最终的解决方案仍然过于复杂。 此解决方案的每个新实现都会包含太多的睡眠问题。

如果你已准备好处理隐藏在代码中的这些恶魔，那么拼接可能是一种非常有用的方法来hook Linux内核函数。
但由于我们不喜欢这个选项，我们将其作为替补方案，以防我们找不到更好的选择。

有第五种方法吗？

当我们研究这个主题时，我们注意到了Linux ftrace，一个可用于跟踪Linux内核函数调用的框架。
虽然使用ftrace执行Linux内核跟踪是常见的做法，但此框架也可以用作jprobes的替代方案。 事实证明，ftrace比jprobes更适合跟踪函数调用的需求。

Ftrace允许我们通过名称hook关键Linux内核函数，并且可以在不重建内核的情况下安装钩子。 在本系列的下一部分中，我们将更多地讨论ftrace。
什么是ftrace？ ftrace如何运作？ 我们将回答这些问题，并为你提供详细的ftrace示例，以便你更好地了解该过程。 我们还会告诉你主要的优缺点。

等待本系列的第二部分了解有关这种不寻常方法的更多细节。

结论

有许多方法可以尝试在Linux内核中hook关键函数。 我们已经描述了完成这项任务的四种最常用的方法，并解释了每种方法的优点和缺点。
在我们的三部分系列的下一部分中，我们将告诉你更多关于我们的专家团队最终提出的解决方案 - 使用ftrace hook Linux内核函数。

有什么问题吗？ 您可以在此处了解有关我们Linux内核开发经验的更多信息。

上一篇：WebAssembly黑暗的一面（下） 下一篇：Hooking linux内核函数...
1. 0 条回复
   - 动动手指，沙发就是你的了！