**Falco原理**

Falco是一款开源的行为监视器，旨在检测应用程序中的异常活动。 Falco由一系列规则组成，这些规则基于应用程序执行的系统调用来识别可疑行为。 Falco可以应用于容器环境、虚拟化环境、Linux物理主机环境

1. 文件完整性监控：

执行文件完整性监控的一种方法是定期扫描所有感兴趣的文件，计算其校验和，并与前一阶段的校验和进行比较。 这种方法面临的挑战是扫描所有文件的成本很高，因此通常每隔几个小时运行一次。 使用falco，我们可以只监视正在写入感兴趣文件的任何操作系统活动，并实时收到警报。

1. 网络监控：

基于网络的IDS系统已经有很长一段时间了。 但是，随着工作负载迁移到虚拟机和容器，将网络流量与应用程序活动协调起来变得越来越困难。 此外，基于网络的IDS与签名密切相关，因为您只能从网络流量中观察到一小部分系统行为。 使用falco，我们可以看到“从内部”的I / O，并且在应用程序和流量之间立即存在关联。

1. 集成的操作系统设施：

Linux有多个安全模块，特别包括SELinux和AppArmor。 这些是非常先进的访问控制系统，具有复杂的策略和概念。 因此，理解和配置它们是一件相当复杂的事情。 Falco的理解和配置要简单得多。 如果您过去使用复杂的安全产品时遇到困难，请不要气馁，试用falco！ （请注意，一个主要区别是falco目前只能检测到，而AppArmor和SELinux可以检测和/或强制执行，但这与配置的复杂性是正交的）。

Falco是一个长期运行的服务器代理。 在容器化环境中，它可以作为一个容器来监视主机本身和所有在其上运行的容器。 当然，它也可以以软件包的方式安装。

一旦部署完毕，falco开始监视配置文件中声明的每一个系统调用事件

Falco构建在一个故障排除的核心软件之上。 具体来说，它使用sysdig内核模块进行syscall拦截和sysdig用户库进行状态跟踪和事件解析。 目前，它们都在数十万台机器上部署和运行。 这意味着非常好的稳定性，即使是最初的“0.1.0”版本。 另一方面，性能也是一个大问题。 当部署在负载大的主机和具有大量策略集的场景下，CPU负载会因为Falco而变得很大。 这个问题将在以后的版本中改进

敏感的行为（例如上面的那些）被编译为规则部署在过滤器中。 每个规则都有一个关联的输出模板，用于指定发生匹配事件时要输出的信息。 请注意，falco不会对信息进行收集，警报，和修复。

比如，你当然不希望有一个shell运行在docker中，那么你可以通过添加策略来识别这种行为：

- rule: shell\_in\_container

desc: a shell running in a container

condition: container.id != host and proc.name = bash

output: “Shell running in container (user=%user.name container\_id=%container.id container\_name=%container.name shell=%proc.name parent=%proc.pname)”

priority: WARNING

rule：策略名

desc：描述

condition：规则 (container.id[是否容器] 、 proc.name[进程名])

output：输出行为信息

priority：安全等级

在这里值得强调的是，falco将会检测主机上的所有容器 - 只要运行falco容器，它就会立即监控你的所有容器以及后续启动的容器。

你感受过被SQL注入、Web服务器漏洞、代码执行攻击、PHP的利用等的痛苦吗？ 所有这些都很难用传统的基于网络的模式匹配来检测，并且需要您不断更新您的指纹库。

但是在一天结束时，大多数这些攻击通常会导致异常的服务器行为，例如创建进程或写入意外的文件。Falco只需几条简单的规则就可以可靠地检测到这些行为。

例如，您可以编写一个策略来检测进程何时启动以及分析进程属性如程序名与父进程名：

- rule: mysqld\_spawn\_process

desc: mysqld spawning a new process after startup. This shouldn't occur and is a follow on from some SQL injection attacks.

condition: spawn\_process and proc.name = mysqld and not proc\_is\_new

output: “mysqld spawned new process after startup (user=%user.name command=%proc.cmdline file=%fd.name)”

priority: WARNING

涉及两个宏：spawn\_process和proc\_is\_new

spawn\_process：

检查执行程序的execve系统调用

- macro: spawn\_process

condition: syscall.type = execve

proc\_is­\_new：

第二个使用自从一个过程开始的时间，阈值为5秒，以确定过程是否是“新的”

- macro: proc\_is\_new

condition: proc.duration <= 5000000000

如果一个像ls或ps这样的标准系统程序发出一个TCP连接，这显然是异常的 - 一个可能的解释是主机已经被插入rootkit了。 虽然您最好在发生Rootkit安装时（可能使用上述类型的规则）检测rootkit安装，但是深入防护并检测攻击发生后可能发生的行为仍然很重要。

这种情况，我们可以监控系统标准调用的connect行为，添加的策略如下：

- macro: open\_connection

condition: syscall.type=connect and evt.dir=< and fd.sockfamily =ip

- rule: system\_binaries\_network\_activity

desc: any network connection initiated by system binaries that are not expected to send or receive any network traffic

condition: open\_connection and proc.name in (ls, ps, mkdir, … )

output: Known system binary made network connection (user=%user.name command=%proc.cmdline connection=%fd.name)

priority: WARNING

我们将使用Falco监视pipe installer进程及其子项并标记可疑行为。 这包括两个步骤：

1. 我们不用bash运行安装脚本，而是运行一个与bash相同的新程序fbash（falco-safe bash），然后创建一个以自己为主导的session。
2. Falco会仔细检查fbash session中的进程所执行的系统调用，如果有人执行可疑的操作（如修改系统文件，启动服务器等），则发送通知。

Falco提供的保护有几个不同的层次：

1. 不管是否安装新软件，在任何情况下都不应该发生的行为。 Falco标记为WARNING级别
2. 有些行为只有在安装新软件时才会发生。 当没有执行软件安装时，这些行为被记录在WARNING级别，否则在INFO级别。
3. 有些行为在安装新软件时不应该发生。 这些行为记录在WARNING级别。

**Activities that should never occur**

# Only let rpm-related programs write to the rpm database

- rule: write\_rpm\_database

desc: an attempt to write to the rpm database by any non-rpm related program

condition: open\_write and not rpm\_mgmt\_programs and fd.directory=/var/lib/rpm

output: "Rpm database opened for writing by a non-rpm program (command=%proc.cmdline file=%fd.name)"

priority: WARNING

这个策略防止在pipe installer中的恶意代码操作/var/lib/rpm下的文件

### Activities that can occur during installation, but not generally

# General use of system management programs results in a warning

- rule: manages\_service

desc: an attempt by a program not in a pipe installer session to manage a system service (systemd/chkconfig)

condition: evt.type=execve and proc.name in (chkconfig, systemctl) and not proc.sname=fbash

output: "Service management program run by process outside a fbash session (command=%proc.cmdline)"

priority: WARNING

不使用proc.sname

# Within a fbash session, the severity is lowered to INFO

- rule: installer\_bash\_manages\_service

desc: an attempt by a program in a pipe installer session to manage a system service (systemd/chkconfig)

condition: evt.type=execve and proc.name in (chkconfig, systemctl) and proc.sname=fbash

output: "Service management program run by process in a fbash session (command=%proc.cmdline)"

priority: INFO

使用proc.sname

### Activities that should never occur during installation

- rule: installer\_bash\_starts\_network\_server

desc: an attempt by any program that is in a session led by fbash to start listening for network connections

condition: evt.type=listen and proc.sname=fbash

output: "Unexpected listen call by a process in a session led by fbash (command=%proc.cmdline)"

priority: WARNING

检测僵尸网络客户端脚本所做的主要事情（调用listen（）来启动侦听网络连接的服务器）。尽管期望服务充当服务器是合理的，但是在运行安装脚本时不会发生这种情况。 这就是为什么我们提出警告。

- rule: installer\_bash\_starts\_session

desc: an attempt by any program that is in a session led by fbash to start a new session

condition: evt.type=setsid and proc.sname=fbash

output: "Unexpected setsid call by a process in a session led by fbash (command=%proc.cmdline)"

priority: WARNING

第二个规则确保程序不能通过调用setsid（）来创建自己的session来逃避受监视的session。

Enforcement工具：根据策略改变进程的系统调用行为，或将进程杀死。例如Seccomp、seccomp-bpf、SELinux和AppArmor

Auditing工具：根据策略监控进程的系统调用行为（提供不同的警告等级），例如Falco、Auditd

## Basic Sandboxing: seccomp

sandbox的目标是通过减少进程可以执行的操作来减少潜在的攻击面。 Seccomp就是一个例子。 Seccomp是Linux内核中的一种机制，它允许进程单向转换到只能执行有限系统调用的受限状态。 如果一个进程尝试其他系统调用，它将通过SIGKILL信号被终止。 在最严格的模式下，seccomp可以防止除read（），write（），\_exit（）和sigreturn（）以外的所有系统调用。 这将允许程序进行初始化，然后进入限制模式，只能从已经打开的文件读取/写入数据。

## Sandboxing with Policies: seccomp-bpf

seccomp-bpf是seccomp的扩展，允许指定应用于每个系统调用的过滤器。 该过滤器是使用BPF编写的，后者起源于tcpdump，但实际上已成为Linux内核中的虚拟机实现。 加载到内核中的BPF程序以系统调用+参数开始，并导致过滤决定。 根据过滤器的结果，系统调用可以被允许，阻塞，或者可以终止进程。

seccomp-bpf的最广泛使用可能是由docker隔离容器化的应用程序。 Docker根据seccomp配置文件启动进程，禁用44个系统调用，阻止其使用。 禁用的系统调用的示例是mount（挂载文件系统），reboot（重新启动主机），和setns（更改命名空间以尝试转移容器）。

## Mandatory Access Control Systems: AppArmor and SELinux

从极端的思路看，如果您将策略添加到沙箱，结果是强制访问控制系统（如AppArmor和SELinux）。 这两个工具争取在系统范围内执行控制系统中每个程序可以执行的操作和资源的策略。 配置文件之外的活动可能会导致记录警告，系统调用失败或查杀进程。

与AppArmor不同的是，配置文件以流程为导向，SELinux策略要复杂得多，并分别应用于角色，动作和目标，整个中间件类型为不同的地方定义策略。

## System Auditing: Auditd

Auditd是一个访问监控系统，实际上被用作SELinux的记录器。 Auditd的配置由一系列在文件系统路径或系统调用中添加监视点的规则组成。 当访问相关文件和/或调用系统调用时，将记录该操作的详细信息。

所有这些工具都使用相同的方法 - 定义一个策略来指定程序可以/不能执行的操作，然后将该策略应用于计算机上的系统调用流。 Falco和其他工具的区别在于Falco运行在用户空间中，使用内核模块获取系统调用，其他工具在内核级执行系统调用过滤/监视。 这使得Falco可能受到攻击。 如果可以杀死/暂停 Falco进程，则可以禁用检测。 而其他工具在内核中替换一组加载的策略或BPF程序可能更困难。