

Docker的安全性之我见

艾奇伟、Shawn



我是谁？

- 大鹰、e4gle
- Linux系统安全技术爱好者
- 绿色兵团UNIX版版主、西祠胡同UNIX版版主、chinaunix安全版版主（曾经）
- 各主流操作系统的MAC底层实现
- Kiwi.ai@qq.com



我是谁?

- citypw(Shawn Chang), GNU/Linux安全工程师, 自由软件狂热分子, Hardenedlinux社区发起人,
- citypw@gmail.com



HardenedLinux

大纲

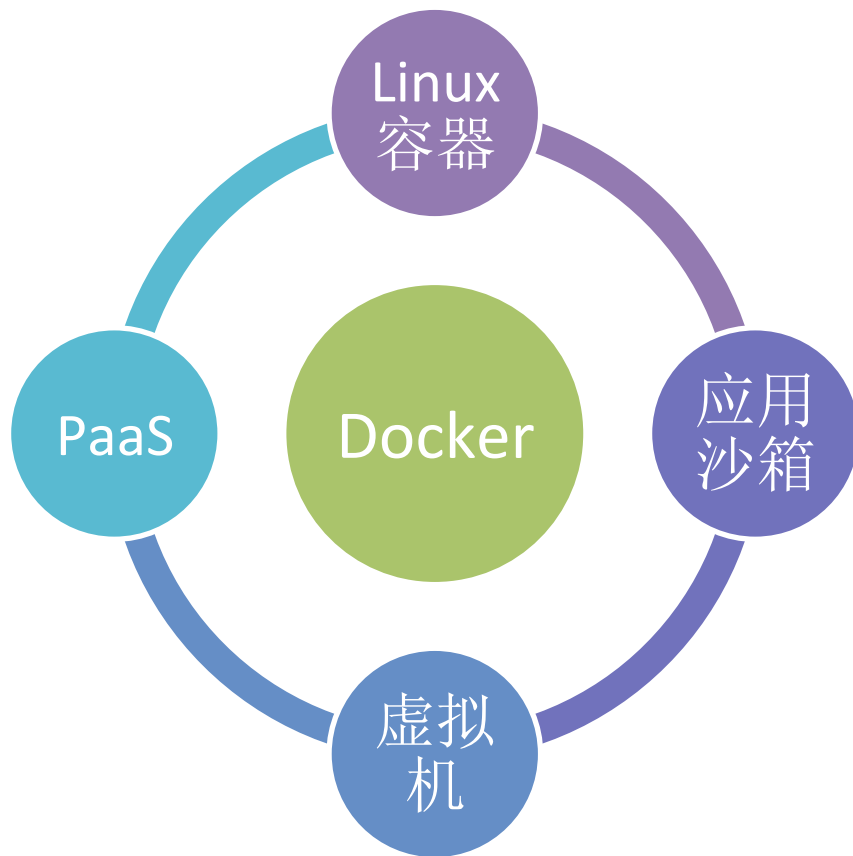
- Docker的标签
- 大家对安全性的担忧
- 安全问题解析
- 认清楚Docker
- 安全的使用Docker
- GRSEC以及防护



Docker的标签



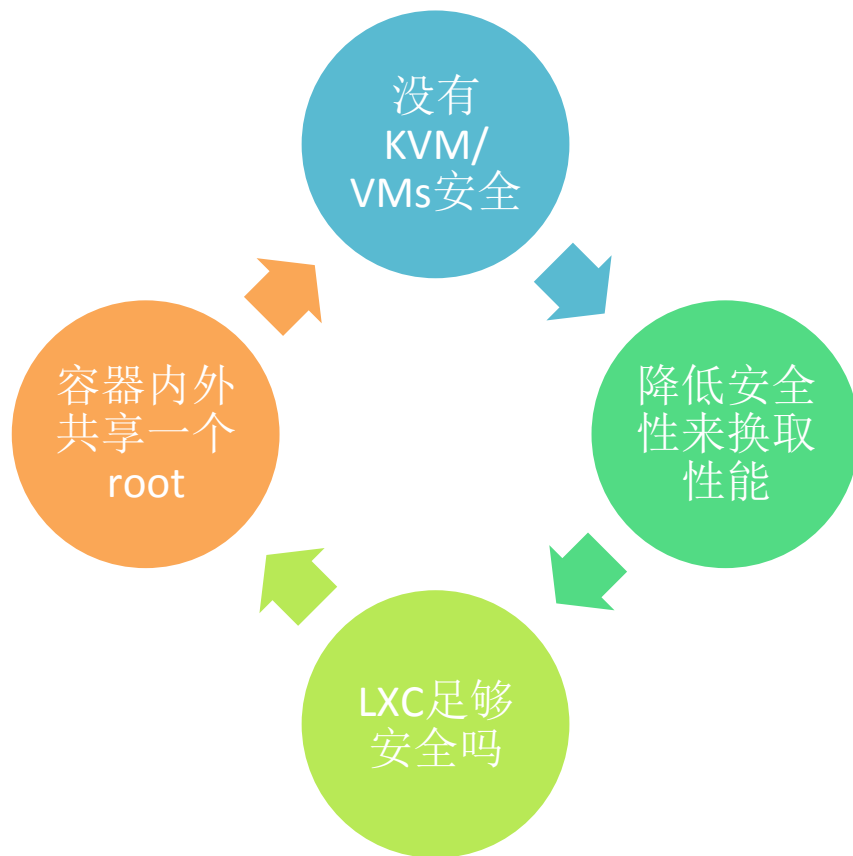
Docker的标签



大家对安全性的担忧



大家对安全性的担忧



安全问题解析



安全威胁解析

- 内核漏洞，**syscall**漏洞
- 容器中有些操作需要root权限
- NS并没有覆盖系统所有的资源
- 配置的问题



内核漏洞

- 容器与宿主机共用一个内核实例
- 在容器中可以直接调用内核提供的系统调用
- 当系统调用有漏洞时，从容器可以直接取得内核的权限



Syscall漏洞

<http://lwn.net/Articles/268783/>

vmsplice(): the making of a local root exploit

By **Jonathan Corbet**
February 12, 2008

As this is being written, distributors are working quickly to ship kernel updates fixing the local root vulnerabilities in the `vmsplice()` system call. Unlike a number of other recent vulnerabilities which have required special situations (such as the presence of specific hardware) to exploit, these vulnerabilities are trivially exploited and the code to do so is circulating on the net. Your editor found himself wondering how such a wide hole could find its way into the core kernel code, so he set himself the task of figuring out just what was going on - a task which took rather longer than he had expected.

The `splice()` system call, remember, is a mechanism for creating data flow plumbing within the kernel. It can be used to join two file descriptors; the kernel will then read data from one of those descriptors and write it to the other in the most efficient way possible. So one can write a trivial file copy program which opens the source and destination files, then splices the two together. The `vmsplice()` variant connects a file descriptor (which must be a pipe) to a region of user memory; it is in this system call that the problems came to be.

The first step in understanding this vulnerability is that, in fact, it is three separate bugs. When the word of this problem first came out, it was thought to only affect 2.6.23 and 2.6.24 kernels. Changes to the `vmsplice()` code had caused the omission of a couple of important permissions checks. In particular, if the application had requested that `vmsplice()` move the contents of a pipe into a range of memory, the kernel didn't check whether that application had the right to write to that memory. So the exploit could simply write a code snippet of its choice into a pipe, then ask the kernel to copy it into a piece of kernel memory. Think of it as a quick-and-easy rootkit installation mechanism.

If the application is, instead, splicing a memory range into a pipe, the kernel must, first, read in one or more `iovec` structures describing that memory range. The 2.6.23 `vmsplice()` changes omitted a check on whether the purported `iovec` structures were in readable memory. This looks more like an information disclosure vulnerability than anything else - though, as we will see, it can be hard to tell sometimes.



容器中有些操作需要root权限

- 对网络接口高级访问
- 直接访问设备
- 文件系统挂载



NS并没有覆盖系统所有的资源

- 在Linux中，不是所有资源都可以进行NS的隔离
- Docker只隔离了5类资源：Process， Network， Mount， Hostname， Shared Memory
- 还有很多重要的资源是开方的：SELinux， Cgroups， /sys， /proc/sys， /dev/sd* ...

配置的问题

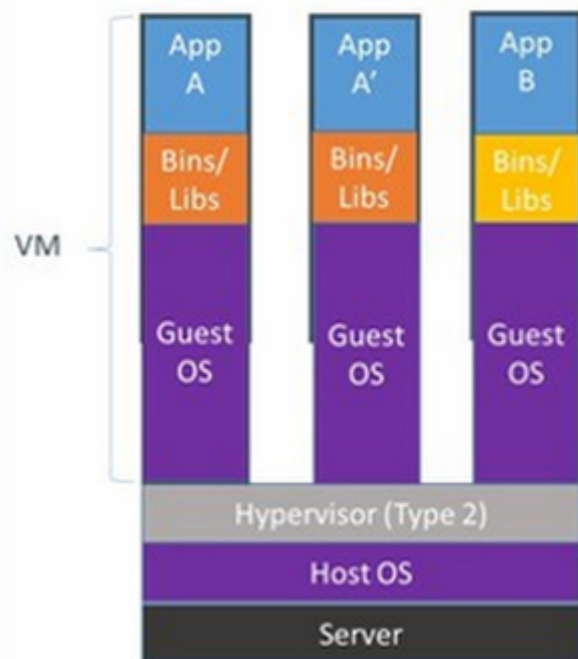
- 没有充分利用容器提供的隔离功能，向容器内部开放了太多或错误的资源

认清 Docker

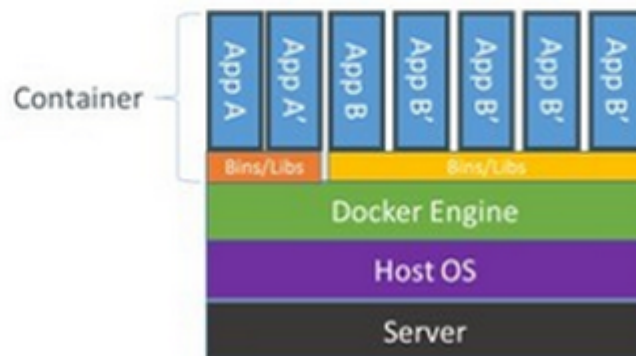


认清 Docker

Containers vs. VMs

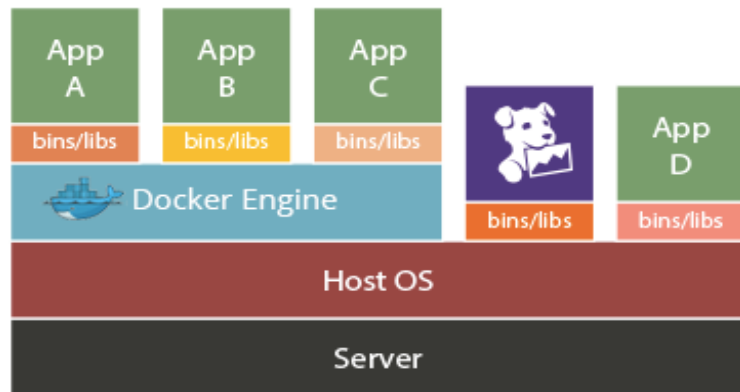


Containers are isolated, but share OS and, where appropriate, bins/libraries



认清 Docker

- 一个比喻: PaaS就像合租房
 - ▣ KVM/VMs: 一个独立的小隔间, 有门有锁
 - ▣ Docker: 只是从你的感觉 (声、光、触) 进行了隔离。给每人一个眼镜、耳机。。



Docker：应用容器

□对宿主机来说，docker实例，就是其内在的应用。在安全上，应该像对待其应用一样，来对待它。



不是虚拟机

- Docker具有很多虚拟机的特性，但它不是虚拟机
- 如果你需要的是虚拟机，不应该勉强Docker



Docker安全特性



Docker安全特性

- 文件系统保护
- Capability机制
- NS, Cgroups
- SELinux/AppArmor



文件系统保护

□ 文件系统只读

■ . /sys、 . /proc/sys、 . /proc/sysrq-trigger、 . /proc/irq、 . /proc/bus

□ 写时复制



Capability 机制

- Linux把原来和超级用户相关的高级权限划分成为不同的单元，称为Capability，这样就可以独立对特定的Capability进行使能或禁止
- 禁用容器中不需要的Capability

NS, Cgroups

- 命名空间：从可见性上进行资源隔离
- Cgroups：对容器所能使用资源进行配额限制

SELinux/AppArmor

- MAC能过强制访问控制来全面限制容器的
- sVirt: libvirt+SELinux
- MCS: 多类别安全

sVirt

□动态地标记镜像文件，并用正确的标记启动虚拟机，允许我们保护主机不受任何虚拟机的侵害，但不能阻止一个虚拟机攻击另一个虚拟机

MCS

□可以使用MCS隔离两个相同SELinux类型（svirt_t）的虚拟机，使libvirt分配一个不同的随机选择的MCS标记给每个虚拟机及其关联的虚拟镜像，libvirt保证它选择的MCS字段的唯一性，SELinux阻止以不同MCS字段运行的虚拟机互操作，这样就保证了虚拟机不会相互攻击。

安全的使用 Docker



安全的使用Docker

- 向容器提供应用所需要的最小权限
- 尽可能不用root
- 采用MAC
- 保持系统内核的安全：更新、打补丁
- 容器+虚拟机
- 日志审计



PaX/Grsecurity 以及防护



PaX/Grsecurity

- 第三方Linux内核补丁
- 提供内核本身的防御能力
- 可以和LSM实现 (AppArmor/SELinux/etc)混合使用
- 提供RBAC功能

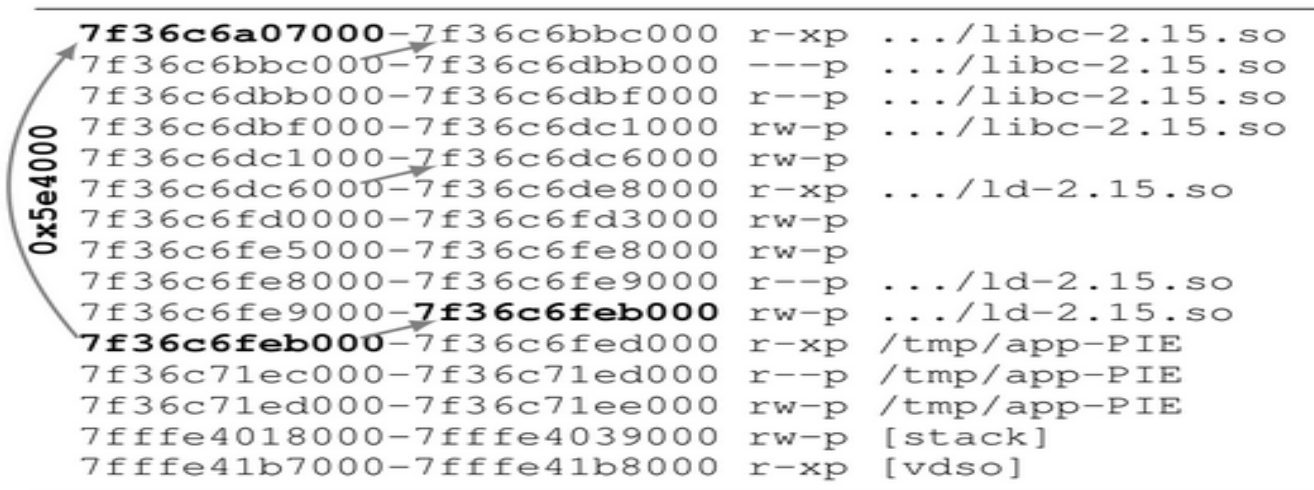


λ 内核加固方案对比

功能	SELinux	Apparmor	PaX/Grsecurity
强制访问控制(MAC)	支持	支持	支持
审计	支持	支持	支持
阻止用户态运行时代码生成	依赖规则	不支持	支持
自动阻止ptrace程序调试	依赖规则	不支持	支持
配合其他LSM一起使用	不支持	不支持	支持
降低内核本身的信息泄漏风险机制	不支持	不支持	支持
防御内核任意代码执行	不支持	不支持	支持
防御内核直接访问用户态内存空间	不支持	不支持	支持

案例：Offset2lib

- Linux内核ASLR的不恰当实现
- 导致glibc地址信息泄漏



A diagram showing a memory map with a curved arrow on the left indicating a range of 0x5e4000. The arrow starts at the address 7f36c6feb000 and points to 7f36c6a07000. The memory map itself is as follows:

7f36c6a07000 →7f36c6bbc000	r-xp	.../libc-2.15.so
7f36c6bbc000→7f36c6dbb000	---p	.../libc-2.15.so
7f36c6dbb000→7f36c6dbf000	r--p	.../libc-2.15.so
7f36c6dbf000→7f36c6dc1000	rw-p	.../libc-2.15.so
7f36c6dc1000→7f36c6dc6000	rw-p	
7f36c6dc6000→7f36c6de8000	r-xp	.../ld-2.15.so
7f36c6fd0000→7f36c6fd3000	rw-p	
7f36c6fe5000→7f36c6fe8000	rw-p	
7f36c6fe8000→7f36c6fe9000	r--p	.../ld-2.15.so
7f36c6fe9000→ 7f36c6feb000	rw-p	.../ld-2.15.so
7f36c6feb000 →7f36c6fed000	r-xp	/tmp/app-PIE
7f36c71ec000→7f36c71ed000	r--p	/tmp/app-PIE
7f36c71ed000→7f36c71ee000	rw-p	/tmp/app-PIE
7fffe4018000→7fffe4039000	rw-p	[stack]
7fffe41b7000→7fffe41b8000	r-xp	[vdso]

Memory map of PIE compiled apps.



案例：Offset2lib

后果很严重：极端情况下，远程利用可以“秒杀” NX(stack上不允许执行)，ASLR（地址随机化）+PIE(针对代码段的随机化)，SSP（压栈出栈检测是否被修改的canary）。

GNU/Linux过去13年的防御一夜之间被击垮。

PaX/Grsecurity:此种利用方式在2001年时已经防御，加固成功！

案例：Offset2lib

<http://cybersecurity.upv.es/attacks/offset2lib/offset2lib.html>

<http://www.solidot.org/story?sid=42174>

案例：Offset2lib

虽然是否出现公开漏洞利用的代码和漏洞利用成功的危害程度是重要的评估要素，，但并不代表说没有公开的漏洞利用就不存在风险，这里举一个 [BadIRET 漏洞](#) 的例子：

Linux 内核代码文件 arch/x86/kernel/entry_64.S 在 3.17.5 之前的版本都没有正确的处理跟 SS（堆栈区）段寄存器相关的错误，这可以让本地用户通过触发一个 IRET 指令从错误的地址空间去访问 GS 基地址来提权。这个编号为 CVE-2014-9322,漏洞于 2014 年 11 月 23 日被 Linux 内核社区修复，之后的几个礼拜里没有出现公开的利用代码甚至相关的讨论。当人们快要遗忘这个威胁的时候，Rafal Wojtczuk 于 2015 年 2 月公布了分析文章 Exploiting “BadIRET” vulnerability 证实了这个漏洞虽然利用极其困难，但并非不可能。Rafal 在 Fedora 20 64-bit GNU/Linux 发行版上完成了研究和测试工作，内核是 3.11.10-301，并且证明只有 PaX/Grsecurity 类似的 UDEREF 技术才能完全阻止此类漏洞利用。

另外一方面，Rafal 也谈到说这个如此严重的漏洞居然数月都没有公开讨论，但其实在 2014 年 12 月中旬[俄文的安全社区就已经进行了详细讨论并最终给出了 PoC](#)。这是一次针对非英文世界的公开威胁情报分析的重大失误的典型案列。既然俄文安全社区已经公布了 PoC 代码，那离稳定的漏洞利用就不远了，从这个案例可以看出，在做已知漏洞的风险评估时一定要考虑斯拉夫兵工厂的实力。

案例2：企业客户使用

CipherGateway的产品中使用了PaX/Grsecurity，和 Docker 以及运行在容器里的应用可以有效的配合：
<http://hardenedlinux.org/system-security/2015/09/06/hardening-es-in-docker-with-grsec.html>




Docker社区安全最佳实践的建议....

- 1, 使用PaX/Grsecurity内核
- 2, 配合LSM的MAC实现
- 3, 关闭不必要的权限以及合理使用CAPABILITY



谢谢！



Kiwi.e4gle 

北京 朝阳



扫一扫上面的二维码图案，加我微信

