

基于主机系统调用频率的容器入侵检测方法

季一木^{1,2,3,4}, 杨卫东^{1,3}, 李奎^{1,3}, 刘尚东^{1,2,3,4},
刘强^{1,3}, 邵思思^{1,3}, 尤帅^{1,3}, 黄乃娇^{1,3}

- (1. 南京邮电大学计算机学院, 江苏 南京 210023;
2. 国家高性能计算中心南京分中心, 江苏 南京 210023;
3. 南京邮电大学高性能计算与大数据处理研究所, 江苏 南京 210023;
4. 南京邮电大学高性能计算与智能处理工程研究中心, 江苏 南京 210023)

摘要: 容器技术由于其轻量级虚拟化的特点, 已成为云平台中广泛使用的虚拟化技术, 但它与宿主机共享内核, 安全性和隔离性较差, 易遭受泛洪、拒绝服务、逃逸攻击。为了有效检测容器是否遭受攻击, 提出了一种基于主机系统调用频率的入侵检测方法, 该方法利用不同攻击行为之间系统调用频率不同的特点, 收集容器运行时产生的系统调用, 结合滑动窗口和 TF-IDF 算法提取系统调用特征, 通过对比特征相似度进行分类。通过实验验证, 该方法的检测率可达 97%, 误报率低于 4%。

关键词: 主机系统调用; 入侵检测; Docker 容器; ADFA-LD 数据集

中图分类号: TP399

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2021073

Container intrusion detection method based on host system call frequency

Ji Yimu^{1,2,3,4}, YANG Weidong^{1,3}, LI Kui^{1,3}, LIU Shangdong^{1,2,3,4}, LIU Qiang^{1,3},
SHAO Sisi^{1,3}, YOU Shuai^{1,3}, HUANG Naijiao^{1,3}

1. School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210023, China
2. Nanjing Center of HPC, Nanjing 210023, China
3. Institute of High Performance Computing and Big Data Processing, Nanjing University of Posts and Telecommunications, Nanjing 210023, China

收稿日期: 2020-10-19; **修回日期:** 2021-01-11

通信作者: 刘尚东, lsd@njupt.edu.cn

基金项目: 国家自然科学基金 (62076139, 61902194); 江苏省 (高校) 自然科学基金 (BK20170900, 19KJB520046, 20KJA520001); 江苏省六大人才高峰项目 (JY02); 之江实验室开放课题 (2021KF0AB05); 南京邮电大学鼎山人才培养对象项目和南京邮电大学人才启动基金 (NY219132); 江苏省研究生创新计划项目 (KYCX19_0921, KYCX19_0906)

Foundation Items: The National Natural Science Foundation of China (62076139, 61902194), The Natural Science Foundation of Jiangsu Province (Higher Education Institutions) (BK20170900, 19KJB520046, 20KJA520001), Six talent peak projects in Jiangsu Province (JY02), Zhejiang Lab (2021KF0AB05), NUPT DingShan Scholar Project and NUPTSF(NY219132), Postgraduate Research & Practice Innovation Program of Jiangsu Province (KYCX19_0921, KYCX19_0906)

论文引用格式: 季一木, 杨卫东, 李奎, 等. 基于主机系统调用频率的容器入侵检测方法[J]. 网络与信息安全学报, 2021, 7(4): 18-29.

Ji Y M, YANG W D, LI K, et al. Container intrusion detection method based on host system call frequency[J]. Chinese Journal of Network and Information Security, 2021, 7(4): 18-29.

4. Research Center for High Performance Computing and Intelligent Processing Engineering,
Nanjing University of Posts and Telecommunications, Nanjing 210023, China

Abstract: Container technology has become a widely used virtualization technology in cloud platform due to its lightweight virtualization characteristics. However, it shares the kernel with the host, so it has poor security and isolation, and is vulnerable to flood, denial of service, and escape attacks. In order to effectively detect whether the container is attacked or not, an intrusion detection method based on host system call frequency was proposed. This method took advantage of the different frequency of system call between different attack behaviors, collected the system call generated when the container was running, extracted the system call features by combining the sliding window and TF-IDF algorithm, and classified by comparing the feature similarity. The experimental results show that the detection rate of this method can reach 97%, and the false alarm rate is less than 4%.

Keywords: host system call, intrusion detection, Docker container, ADFA-LD data set

1 引言

虚拟化技术已经成为一种大众所认可的服务器资源共享方式，它在提高系统资源利用率和灵活性的同时降低了系统管理成本。其中广为人知的便是虚拟机(VM, virtual machine)，可以利用虚拟机将一台服务器隔离成多个虚拟主机，每个虚拟主机上运行独立的应用，以便于管理。但是，经过一段时间的使用，人们逐渐发现虚拟主机自身占用了过多的资源，导致服务器的资源利用率低。因此出现了一种应用层面的新型虚拟化技术——容器，来帮助解决这个问题。

近几年来，容器技术取得了飞速的发展，成为虚拟化技术中“一颗快速升起的新星”。从目前发展趋势来看，容器技术的发展正在逐渐改变着传统虚拟机应用和软件开发与部署的方式。在众多容器技术中，Docker 是其中的佼佼者，与传统的虚拟化技术相比，由于 Docker 减少 Guest OS 这一层级，所以 Docker 容器技术有着更低的系统资源占用率，如图 1 所示。另外，在软件开发与部署方面，Docker 容器技术通过打包应用及其依赖包，将应用变成一种标准化的、可移植的、自管理的组件，能在任何主流系统中开发、调试和运行。正是因为 Docker 容器所具有的这些优点，越来越多的企业选择将 Docker 容器技术应用到其内部云平台架构中，但随着 Docker 容器技术的广泛应用，其所带来的未知风险和漏洞利用威胁也随之显现^[1]。

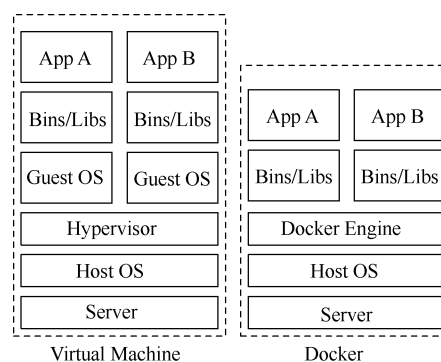


图1 VM与Docker架构
Figure 1 VM and Docker architecture

Docker容器的安全问题主要来源于不安全的镜像源以及 Docker 本身架构与安全机制的不完善^[2]。为了方便用户快速搭建环境，Docker 官方提供了 docker hub，允许用户上传自己创建的镜像，供其他用户下载使用，这在方便用户的同时也带来了安全问题，用户可能上传存在漏洞或后门程序的镜像，恶意用户可以利用这些漏洞和后门渗透到使用者的云平台^[3]。Docker 本身的模块化设计也引入了利用组件间通信进行拒绝服务(DoS)攻击的可攻击面^[4]。另外，Docker 本身的安全依赖于内核名称空间和控制组(Cgroup, control groups)，使恶意用户可以基于其共享宿主机内核的特性，利用内核漏洞发起针对容器的攻击^[5]。一旦恶意用户在容器内发起有效的攻击就可能使容器宿主机宕机，甚至从容器内逃逸至宿主机并获得宿主机控制权限，影响其他容器和整个系统的可靠运行^[6]。文献[7]实验对比了 VM 和各种容器技术下云平台的安全性，得到了与 VM

相比, 由于容器与主机操作系统紧密耦合并运行在主机操作系统之上, 攻击者可以通过破坏容器达到完全访问主机操作系统和资源的结论。因此, Docker 容器的脆弱性、攻击方法以及安全防御方案成为容器研究的热点。针对容器中攻击的特殊性、难以预测性和高破坏性, 可以通过对 Docker 容器资源和进程行为监控及时发现并阻止威胁行为, 将不可预料的攻击的破坏程度降到最低^[8]。与虚拟机不同, 容器与宿主机共享同一个内核。这使得可以通过监视主机中容器的系统调用进行入侵检测, 并且监控系统不需要具有关于容器性质的任何知识, 也不需要修改主机系统或被监控的容器。

本文在分析了多种攻击产生的系统调用和攻击特征之后, 设计并实现了一种基于主机系统调用的容器入侵检测方法。该方法适用于检测 Docker 容器中是否存在入侵行为, 并在检测到入侵后上报异常, 可以提升容器运行时的可靠性和安全性。该方法与常规基于主机的入侵检测方法相比, 由于容器通常用于运行某一特定应用, 具有较少的进程, 因此针对容器的入侵检测可以使用更简单的方法和更少的计算资源实现更高的入侵检测率。

2 相关工作

现有的容器入侵检测方法主要有基于主机的入侵检测和基于网络的入侵检测这两类^[9]。其中基于主机的入侵检测方法专注于容器内部, 监视容器内部全部或部分动态的行为以及整个容器的状态, 用作容器的分析器和监视器, 并不对外提供接口。基于主机的入侵检测方法的实现依赖于动态检查数据一致性这一特性, 即使是优秀的入侵者, 也不可避免地会留下他们入侵的痕迹, 通过对这些痕迹的处理和分析, 基于主机的入侵检测方法能够及时、准确地发现并上报入侵行为^[10]。基于网络的入侵检测方法是从网络上 TCP/IP 消息流中识别出潜在的攻击行为, 是对收集的漏洞信息、造成拒绝访问以及越权访问等危害计算机系统安全运行的行为进行检测的软件与硬件的组合。虽然该方法有配置简单、低性能需求和不影响正常业务运行的优点, 但它与基于主机的入侵检测方

法相比, 有误报率和漏报率高、没有主动防御能力和性能不足的缺点。

在基于主机的入侵检测方法中, Jian 等^[11]通过动态检查主机名称空间, 实现了容器异常进程的检测, 有效防止了逃逸行为, 但这种方法仅能检测通过容器对宿主机的攻击, 无法检测到容器内的攻击。为了更加有效地检测容器受到的攻击, 在基于主机的入侵检测方法中, 人们倾向于使用容器运行时产生的系统调用进行检测, 因为这样不仅能直接反映容器内进行的操作, 还能检测到更广的攻击范围。另外, 目前已有的基于主机系统调用的入侵检测方法中又可以分为基于系统调用序列的入侵检测方法和基于系统调用频率的入侵检测方法这两类。1996 年, Forrest 等^[12]提出了序列延时嵌入 (STIDE, sequence time-delay embedding) 技术, 使用大小为 k 的短序列系统调用来定义正常行为, 为了构建数据库, 在系统调用的轨迹上使用大小为 $k+1$ 的滑动窗口, 存储窗口内的系统调用序列。Warrender 等^[13]比较了基于 STIDE, RIPPER 和 HMM (马尔可夫模型) 的方法, 他们最终得出所有方法均表现良好, 但与 HMM 相比, STIDE 有更低的计算资源和存储空间的结论。虽然 STIDE 是一种简单有效的检测技术, 但由于它保存调用的顺序信息, 数据库的大小会随着系统调用数量线性增长。尽管后来 Forrest、Warrender 等对提出的 STIDE 技术进行了一些改进, 但依旧无法解决数据库随系统调用数量线性增长的问题^[13-14]。因此, 2005 年, Kang 等^[15]提出了系统调用包 (BoSC, bag of system calls) 技术, BoSC 是一种基于系统调用频率的入侵检测技术。它将 BoSC 定义为有序列表 $\langle c_1, c_2, \dots, c_n \rangle$, 其中 n 是不同系统调用的总数, c_i 是给定输入序列中系统调用 s_i 的出现次数。他们在两个公开可用的系统调用数据集 (新墨西哥大学 (UNM) 数据集和麻省理工学院林肯实验室数据集) 上结合多种机器学习方法实验证明, 与基于序列的方法 STIDE 相比, BoSC 具有更好的性能和准确性。此后, Abed 等^[16]将 BoSC 与滑动窗口技术结合, 提出了一种针对容器中应用程序的实时检测系统, 用于检测容器中应用程序的安全性, 该系统具有较高的检测率, 但对训练要求高, 需要尽可能多地训练到各种情况。文献^[17]

提出了一种实时入侵检测系统模型,用于检测容器中运行的恶意应用程序,使用 n -gram 系统调用,然后计算出该 n -gram 的出现概率,使用最大似然估计 (MLE) 和简单良好图灵 (SGT) 处理跟踪,并在 UNM 数据集上验证该方法,得到了 87%~97% 的准确度。

Khreich 等^[18]为了降低检测的误报率,提出了一种基于特征提取技术的异常检测系统,该系统结合了系统调用的频率和时间信息,基于单类支持向量机 (OC-SVM, one-class SVM) 检测器进行异常检测,并在实验中对比了各种优秀的入侵检测方法,证明了所提方法的优越性。虽然其与现有优秀的入侵检测方法相比具有较低的误报率和较高的检测率,但由于不同系统调用组合存在相同调用频率的特点,使完全基于频率的特征提取方式不可避免地产生误报和漏报,降低系统的准确率。因此,本文提出了一种新的基于主机系统调用频率的入侵检测方法,提取局部系统调用序列的频率特征,保证了局部系统调用序列中系统调用的有序性,提高了检测的准确率。

3 CIDM 检测方法

本文提出了一种基于主机系统调用频率的容器入侵检测方法 (CIDM, container intrusion detection method), 该方法的工作流程如图 2 所示。

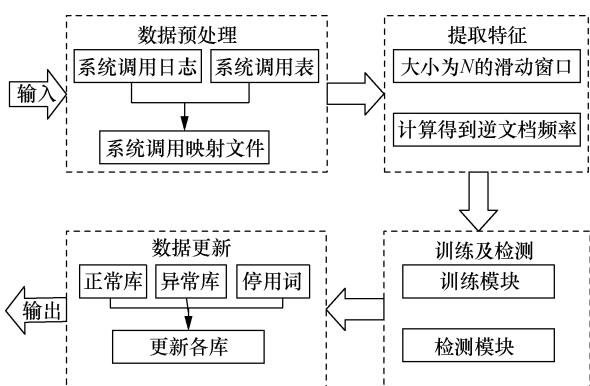


图2 CIDM 工作流程
Figure 2 CIDM workflow

容器上的操作都会产生相应的系统调用,攻击者的攻击也不例外。当容器工作时,会产生大量的系统调用信息,直接处理和存储这些系统调

用信息会占用很高的系统资源开销,因此,本文在收集到系统调用信息后,首先对收集到的数据进行预处理,去除参数等无用信息,仅保留系统调用名称并结合 Linux 系统中系统调用表文件 (此文件中存放着系统调用对应的编号,在 /usr/include/asm/unisd.h 路径下,不同内核版本名称和位置可能不同) 生成系统调用的映射文件。然后提取系统调用映射文件的词频特征,在提取特征的过程中首先使用滑动窗口技术,窗口大小设置为 N ,将每个窗口中系统调用的组合视为一个词,并保证窗口内系统调用的有序性,接着使用并改进现有的 TF-IDF (term frequency-inverse document frequency) 算法计算每个词的逆文档频率 (IDF)。

在得到所有词的逆文档频率后,根据系统调用日志是否具有分类信息进入训练模块或者检测模块。当系统调用日志具有分类信息时,系统进入训练模块,计算每个词的 TF-IDF 值,根据分类信息将其存入相应的文件中,得到正常系统调用库和异常系统调用库,并将两个库中都存在的元素存入停用词列表。当系统调用无分类信息时,系统进入检测模块,对接收到的系统调用进行处理,判断是否存在异常。

在训练或检测之后,对停用词列表中的元素结合正常系统调用库和异常系统调用库进行二次分类,解决某个词虽然在正常和异常系统调用库中都有出现过,但主要在某一类中出现,具备分类作用但被停用的问题,用来提升系统检测的准确性。

3.1 数据预处理

数据预处理是为了提取容器运行时产生的系统调用中的关键信息,主要处理通过 Linux strace 跟踪获得的日志文件,将其结合系统调用表生成后续检测使用的系统调用映射文件,减小系统存储资源和计算资源的开销,预处理前后数据对比如表 1 所示。数据预处理过程中两个重要的概念 (系统调用和系统调用表) 详细描述如下。

(1) 系统调用

操作系统的主要功能是为应用程序的运行创建良好的环境,为了达到这个目的,内核提供一系列具备预定功能的多内核函数,通过一组被称为系统调用 (system call) 的接口呈现给用户。系

表 1 预处理前后数据对比
Table 1 Data comparison before and after pretreatment

处理前	处理后
epoll_ctl(8,EPOLL_CTL_ADD,11,{EPOLLIN EPOLLRDHUP EPOLLET},{u32=1998754273,u64=1}})	214
recvfrom(3, "GET / RTTP/1.1rAmHost: 192.168.2. "...,1024, 0, NULL, NULL)=602	45
stat("/usr/share/nginx/html/index.html",{st_mode=S_IFREG 0644,st_size=612})=0	4
openat(AT_FDCWD, "/usr/share/nginx/html/index.html",O_RDONLY O_NONBLOCK)=12	257
fastat(12,{st_mode=S_IFREG 0644,st_size=612,...})=0	5
writetv(3,{{iov_base="HTTP/1.1 304Not Modilied\r\nServe"...},iov_len=181}, 1)=181	20
write(5,"192.168.2.176 - - [14/May/2020:0...]", 196)=196	1
close(12)	3
setsockopt(3,SQL_TCP, TCP_NODELAY, [11], 4)=0	54

统调用把应用程序的请求传给内核，调用相应的内核函数完成应用程序的请求，并将处理结果返回给应用程序，如果没有系统调用和内核函数，用户将无法编写大型应用程序，系统调用与用户程序、内核和硬件设备的调用关系如图 3 所示。Linux 系统调用，包含大部分常用系统调用和由系统调用派生出的函数。另外，不同版本和不同操作系统的系统调用函数数量和功能可能不同，但按照系统调用的功能，通常将其分为进程控制、文件管理、设备管理、信息维护、通信和保护这 6 类。

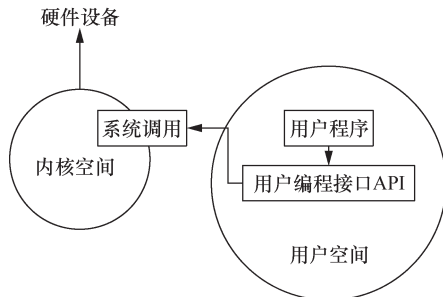


图 3 系统调用与用户程序、内核和硬件设备的调用关系
Figure 3 System calls relate to user program, Kernel and hardware devices

针对当前主流操作系统的研究分析可知，无论要实施何种危害程度的攻击行为，都需要通过用户空间执行系统调用而进入内核空间，以发起各种攻击^[19]。因此作为用户空间和内核空间之间通信接口的系统调用，既可以通过自身的优化来保障系统的安全，又可以通过对进程正常和异常运行时的执行轨迹建模，来刻画进程的正常和异常运行状态，从而建立执行轨迹的局部模式特征，并以此为基础，实现系统的入侵检测。

(2) 系统调用表

系统调用表是一张由指向实现各种系统调用的内核函数的函数指针组成的表，是 Linux 内核中的一个数据结构，该表可以通过系统调用编号来定位函数地址，完成系统调用。Linux 的所有系统调用都是预先定义好的，每一个系统调用都有唯一且不重复的编号，定义在内核服务函数中，如图 4 所示。

#define _NR_Linux	4000
#define _NR_syscall	(_NR_Linux + 0)
#define _NR_exit	(_NR_Linux + 1)
#define _NR_fork	(_NR_Linux + 2)
#define _NR_read	(_NR_Linux + 3)
#define _NR_write	(_NR_Linux + 4)
#define _NR_open	(_NR_Linux + 5)
#define _NR_close	(_NR_Linux + 6)
#define _NR_waitpid	(_NR_Linux + 7)
#define _NR_creat	(_NR_Linux + 8)
#define _NR_link	(_NR_Linux + 9)
#define _NR_unlink	(_NR_Linux + 10)
#define _NR_execve	(_NR_Linux + 11)
#define _NR_chdir	(_NR_Linux + 12)
#define _NR_time	(_NR_Linux + 13)
#define _NR_mknod	(_NR_Linux + 14)
#define _NR_chmod	(_NR_Linux + 15)
#define _NR_lchown	(_NR_Linux + 16)
#define _NR_break	(_NR_Linux + 17)
#define _NR_unused18	(_NR_Linux + 18)
#define _NR_lseek	(_NR_Linux + 19)

图 4 系统调用对应编号
Figure 4 System call corresponding number

3.2 特征提取

特征提取的目的是将处理后的系统调用映射文件特征化，得到并记录文件中每个词对应的 IDF 值以及所有词的词频 (TF, term frequency)，在训练和检测阶段通过对特征进行分类实现对系统调用映射文件的分类。本文在特征提取的过程中使用了滑动窗口技术和一种用于信息检索与数据挖掘的常用加权算法——TF-IDF 算法，并对

TF-IDF 算法进行了优化。

基于主机系统调用的入侵检测方法可以分为两类。一类是基于主机系统调用序列的检测方法，其思路是在训练阶段将固定长度且有序的系统调用信息作为特征存入数据库，当系统运行产生新的系统调用时，比对数据库中特征是否存在，不存在则判为异常。另一类是基于主机系统调用频率的检测方法，其思路是在训练时，统计特定步长中各个系统调用出现的频率并将其作为判断异常的特征存入数据库，在系统运行时，计算每个步长内系统调用的频率并与数据库中的数据匹配，匹配不到时声明异常。两者虽然都可以检测到系统的入侵行为，但都存在着检测准确度和效率不高的问题。基于主机系统调用序列的方法在检测时可以保证系统调用的有序性，保证了系统调用相同但顺序不同的序列不会匹配到数据库中的同一记录，这使该方法数据存储容量的大小随着序列长度的增加呈指数型增长，并且训练时不一定能列出容器的所有的操作，所以该方法不适合检测运行大程序的容器，只能针对某一特定功能的容器进行监测。相比基于主机系统调用序列的方法，基于系统调用频率的方法在增加序列长度时对存储空间的需求不会出现指数型增长，但其无法对具有相同系统调用和调用频率的序列进行准确分类，导致检测的准确度降低，由于正常和异常调用序列具有相同的系统调用和调用频率这种特殊情况并不常见，所以这种方法可用于容器的入侵检测，在非特殊情况下有着较高的检测准确度。

结合基于主机系统调用序列和基于主机系统调用频率的优缺点，本文设计了一种基于滑动窗口和 TF-IDF 的特征提取方法。在系统调用映射文件中使用大小为 N 的滑动窗口（通过实验证明 N 取 6 时有较好的结果），每个窗口内系统调用的组合视为一个词，保证了每个词内部系统调用的有序性，以表 1 处理后的数据为例，设滑动窗口大小为 6，如图 5 所示，可得到 4 个词。

```

214 45 4 257 5 20 | 3 54 → 214 45 4 257 5 20
214 | 45 4 257 5 20 | 3 54 → 45 4 257 5 20 1
214 45 | 4 257 5 20 | 3 54 → 4 257 5 20 1 3
214 45 4 | 257 5 20 | 3 54 → 257 5 20 1 3 54

```

图 5 滑动窗口大小为 6 时的词

Figure 5 Words when sliding window size is 6

对所有系统调用映射文件进行上述处理并记录每个词在多少文件中出现过，然后计算每个词的 IDF 值，IDF 的计算如式(1)所示。

$$IDF_i = \lg \frac{|N|}{1 + |\{j: t_i \in d_j, d_j \in N\}|} \quad (1)$$

其中， IDF_i 表示词 t_i 的逆文档频率， $|N|$ 表示文件总数， $1 + |\{j: t_i \in d_j, d_j \in N\}|$ 表示所有文档中包含词 t_i 的文件数，如果所有文件中不包含某词，就会导致分母为零，所以分母使用 $1 + |\{j: t_i \in d_j, d_j \in N\}|$ 。

IDF 的主要思想是如果包含词的文档数越少，IDF 越大，则说明这个词有很好的类别区分能力。但是，如果某类文档中包含词 c 的文档数为 m ，而其他类包含词 c 的文档总数为 k ，显然所有包含词 c 的文档数为 $s = k + m$ ，当 m 大的时候， s 也大，按照式(1)得到的 IDF 的值会小，就说明该词 c 的类别区分能力不强。但实际上，如果一个词在某一类文档中频繁出现，则该词条能够很好代表这个类的文本的特征，这样的词应当赋予它较高的权重，作为该类文本的特征词，这就是 IDF 的不足之处。针对原有 IDF 的不足，本文做了相应改进，在训练和数据更新时，对出现在多个类中的词进行二次计算。另外，在计算 IDF 时，也会记录各个文件中每个词的词频，词频指的是在给定文件中某个词在该文件中出现的频率，计算公式如下：

$$TF_{i,j} = \frac{t_{i,j}}{\sum_k t_{i,k}} \quad (2)$$

其中， $TF_{i,j}$ 表示词频， $t_{i,j}$ 是词 t_i 在文档 j 中出现的次数， $\sum_k t_{i,k}$ 表示所有文件中词 t_i 出现的总次数。

3.3 训练及检测

在得到每个词的逆文档频率和所有文件中每个词的词频后，进入训练及检测模块。通过判断训练标志选择进入训练模块或者检测模块。在训练模块，根据输入数据和类型信息生成相应的数据库。在检测模块，对比输入数据的特征和数据库中的特征，计算出相似度，判断得出相应类型，训练及检测流程如图 6 所示。

(1) 训练模块

训练开始时，根据特征提取阶段得到的 IDF

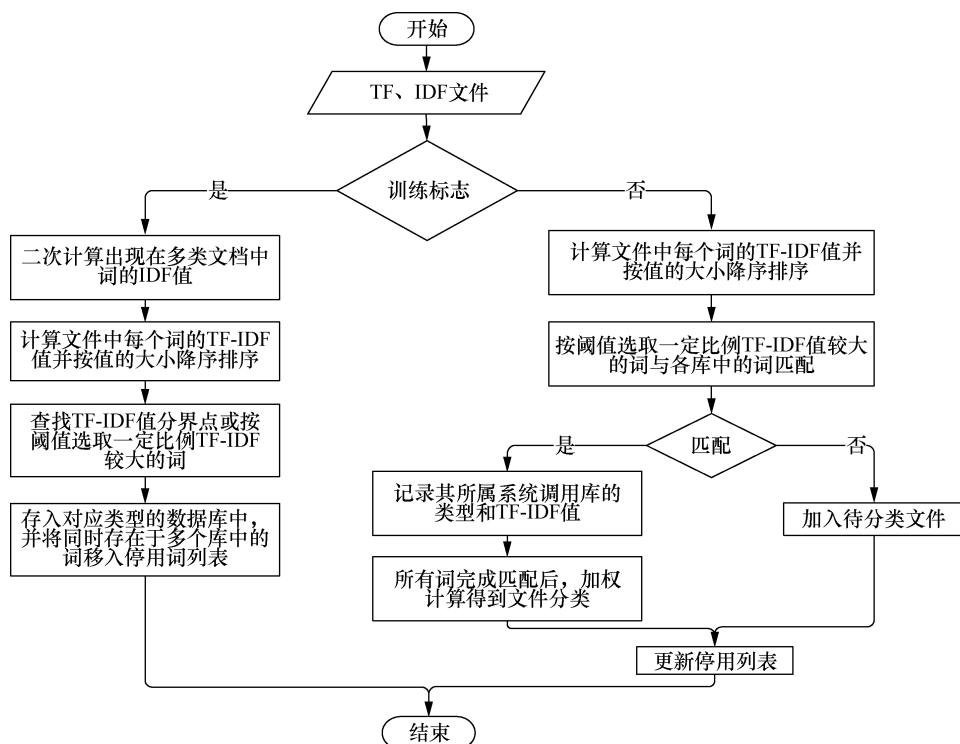


图 6 训练及检测流程

Figure 6 Training and testing process

值以及每个词在各类文件中出现次数和各类文件的总数对 IDF 值进行二次计算, 解决某个词在各类文件中出现但具有分类意义场景下 IDF 特征分类不准确的问题。IDF 值的二次计算是对在各类文件中出现的词进行 IDF 值的重新计算, 计算这个词在各类中的 IDF 值, 如果结果相近则不做更新, 如果在某类文件中 IDF 的值远高于其他类的 IDF 值, 则将该 IDF 值赋值给那个词。计算推导如下: 假设有两类文件, 文件总数分别为 m 和 n , 两类文件中都包含词 t , 每类文件中包含词 t 的文件数目为 a 和 b , 即如果代入式(1)有 $\lg(m/(1+a)) > 3 \cdot \lg(n/(1+b))$, 则

$$IDF_t = \lg(m/(1+a)) \quad (3)$$

然后, 计算出文件中每个词的 TF-IDF 值, 按照 TF-IDF 值的大小排序, 在排好序的 TF-IDF 中找到分界点, 结合文件类别将分界点之前的词存入对应的数据库中, 其中分界点的选择取决于 TF-IDF 是否存在前后差值比较大的点, 如果存在则以此为界, 将大于该点 TF-IDF 值的词存入库中, 否则按总词数的一定比例取对应的词存入库中, 并将训练

过程中同时存在于多个库中的词移入停用词列表, 记录各类文件中出现的次数, 在数据更新时使用。

(2) 检测模块

检测时, 同样计算出文件中每个词的 TF-IDF 值并按照 TF-IDF 值的大小排序, 然后按照排好的顺序将每个词分别与正常系统调用库和异常系统调用库进行匹配, 如果匹配, 记录其所属系统调用库的类型和 TF-IDF 值, 否则将其记录到待分类文件中。之后将词与停用词列表匹配, 如果停用词列表中包含该词, 则将该词也加入待分类文件中。检测时并不会对文件中所有词都进行上述操作, 首先将文件中的词按出现次数进行倒序排序, 然后按阈值选取一定比例词频高的词, 取其中仅在正常系统调用库或异常系统调用库中出现的词进行计算并分类。在计算过程中, 按照 TF-IDF 值在排序后的位置以及正常和异常系统调用库数据量的比例进行加权计算, 计算这些词在各类中的 TF-IDF 值总和, 然后根据总和值进行分类。并将待分类文件中的词根据得出的类型进行处理, 如果是停用列表中的词, 则将停用列表中此词对应类型的数值加一, 否则将该词加入

停用列表，并将对应类型的数值置为一，其他类型置为零，至此检测阶段结束。

3.4 数据更新

数据更新是为了提高检测的准确性和降低误报率，在每次检测后都对正常调用库、异常调用库以及停用词进行更新。更新不会对整个停用词列表进行操作，仅处理停用词列表中包含的待分类文件中的词，并在更新之后清空待分类文件中的内容，这样做可以减小计算量，加快执行速度，因为停用词列表随着时间的增长，内容会逐渐增大，并且每次检测只会改变部分内容，所以只针对改变的内容进行更新会减小计算量，缩短执行时间。

在更新过程中，计算停用词列表中那些发生了变化的词在各类中新的 IDF 值，并根据在各类中 IDF 值的大小，将词存入相对应的系统调用库中，其中 IDF 值的比较规则与 IDF 更新规则相同，以某词在正常系统调用库中 IDF 值远小于在异常系统调用库为例，则将此词存入正常系统调用库，并删除此词在异常系统调用库中的记录，在整个更新过程中不对停用词列表做删改操作。

4 实验

实验的目的是验证本文提出的检测方法的性能，以及阈值选取不同对检测结果的影响，本文选择了两个在基于主机系统调用入侵检测系统方

面具有代表性的数据集，对检测性能进行验证。

4.1 实验数据集

(1) ADFA-LD 数据集

ADFA-LD 数据集是澳大利亚国防学院 2012 年对外发布的一套主机级入侵检测系统的数据集合，被广泛用于网络安全、机器学习、数据挖掘和评估新提出的入侵检测系统或算法的性能，ADFA-LD 数据集文件结构如表 2 所示。

该数据集在制作时就充分考虑了渗透测试人员和黑客通常采用的方法。在 Linux 系统上，有序记录了各类操作和攻击产生的系统调用，并结合系统调用列表对数据进行了特征化和攻击类型标注，各种攻击类型结构和分布如表 3 所示。

(2) UNM 数据集

UNM 数据集是新墨西哥大学为了研究入侵检测系统而采集的数据集，数据集中主要包含了活跃进程下收集的系統调用。这些进程不仅包括守护进程和非守护进程的应用进程（这些应用程序在大小和复杂程度上分布广泛），而且包括可以引起潜在威胁的各种入侵进程，如缓冲区溢出、符号链接攻击、木马程序和特权程序等。

UNM 数据集中的每个文件都是由某个进程从执行开始到结束产生的系统调用组成的。由于程序复杂性的不同以及跟踪进程类型的不同，所以得到的数据集中文件大小可能存在较大差异。与

表 2 ADFA-LD 数据集文件结构
Table 2 File structure of ADFA-LD dataset

数据类型	数据量	系统调用数量	类型
Training	833	308 077	正常
Validation	4 373	2 122 085	正常
Attack	746	317 388	攻击
总计	5 952	2 747 550	

表 3 ADFA-LD 攻击类型结构和分布
Table 3 Structure and distribution of ADFA-LD attack types

攻击类型	攻击名称	载体	文件数
Hydra-FTP	密码爆破	Hydra-FTP	162
Hydra-SSH	密码爆破	Hydra-SSH	176
Adduser	添加新超级用户	客户端投毒	91
Java-Meterpreter	Java-based Meterpreter	TikiWiki 漏洞利用	124
Meterpreter	Linux Meterpreter payload	客户端投毒	75
Webshell	C100 Webshell	PHP 远程文件漏洞	118

ADFA-LD 数据集不同的是, UNM 数据集从多个位置或多个版本的程序中获取同一程序的系统调用数据, 每一个都可能是不同的数据集, 因此, 在一个位置或程序的某个版本收集的入侵不应该与来自不同集合的正常数据进行比较。与 ADFA-LD 相同的是, 每个收集的调用文件都与系统调用列表结合, 使用系统调用号代替系统调用名称, UNM 数据集数据格式如表 4 所示, 第一列数字是执行进程的 PID, 第二列数字是表示系统调用的系统调用号, 在一个文件中可能有多个交错的进程。为了保证数据输入格式的一致性, 本文将 UNM 数据集中每个日志文件按照 ADFA-LD 数据集中数据的格式对同一进程下的系统调用号按先后顺序进行组合, 生成检测阶段的输入文件。

表 4 UNM 数据集数据格式
Table 4 Data format of UNM dataset

PID	系统调用编号
551	4
551	2
551	66
551	66
551	4
551	138
551	66

4.2 评估指标

为了评估 CIDM 的检测性能, 引入检测率和误报率, 检测率计算如式(4)所示, 误报率计算如式(5)所示。

$$Dr = \frac{|i: d_i \in I|}{|I|} \cdot 100\% \quad (4)$$

$$Fpr = \frac{|i: d_i \notin I|}{|N|} \cdot 100\% \quad (5)$$

其中, Dr 表示检测率, Fpr 表示误报率, $|I|$ 表示包含攻击的文件数量, $|N|$ 表示不包含攻击的文件数量, $|i: d_i \in I|$ 表示检测到的攻击数量, $|i: d_i \notin I|$ 表示假警数量。实际上, 所有入侵检测方法的实现由于阈值取值的不同在检测率和误报率之间徘徊, 检测率高时误报率也会升高, 同样, 误报率降低时检测率也会降低。一般情况下, 所

有入侵检测方法都会在两者中根据用户需求取一个折中。在实际生产中, 误报通常不会引起大的危害, 因为操作本身是一个正常的操作, 不会引起系统的异常, 仅仅可能是浪费安全管理员的一些检查时间。而检测率不同, 检测率的高低直接影响了面对非法入侵时系统的安全性。

另外, 为了评估预测的准确率, 本文使用了分类问题中使用的评价指标: 精确率 (Precision)、召回率 (Recall)、准确率 (Accuracy) 和 F 值 (F -measure)。首先, 定义精确率、召回率、准确率和 F 值的相关参数。 $F_{s \rightarrow s}$ 代表安全的调用文件预测为安全的数量; $F_{s \rightarrow u}$ 代表安全的调用文件预测为不安全的数量; $F_{u \rightarrow u}$ 代表不安全的调用文件预测为不安全的数量; $F_{u \rightarrow s}$ 代表不安全的调用文件预测为安全的数量。

精确率: 分类为安全的安全文件的数量与所有分类为安全的文件数量的比。

$$P = \frac{F_{s \rightarrow s}}{F_{s \rightarrow s} + F_{u \rightarrow s}} \quad (6)$$

召回率: 分类为安全的安全文件的数量与所有安全文件数量的比。

$$R = \frac{F_{s \rightarrow s}}{F_{s \rightarrow s} + F_{s \rightarrow u}} \quad (7)$$

F 值: 精确率 P 和召回率 R 的调和平均值。

$$F = (2PR) / (P + R) \quad (8)$$

准确率: 所有预测准确的文件数量与总文件数量的比。

$$Acc = \frac{F_{s \rightarrow s} + F_{u \rightarrow u}}{F_{s \rightarrow s} + F_{u \rightarrow u} + F_{s \rightarrow u} + F_{u \rightarrow s}} \quad (9)$$

4.3 实验分析

本节主要从两方面来验证实验。一方面验证 CIDM 的可行性和有效性, 另一方面验证待分类文件中选取高频词比例的阈值对实验结果的影响。

(1) CIDM 检测方法可行性

CIDM 检测方法的可行性依赖于正常操作和入侵操作产生的系统调用是否具有不同的特征, 为此本文分别在 ADFA-LD 数据集和 UNM 数据集上进行了实验, 实验结果以 ADFA-LD 中 Training traces 和 Add user traces 为例, 如图 7 所示。

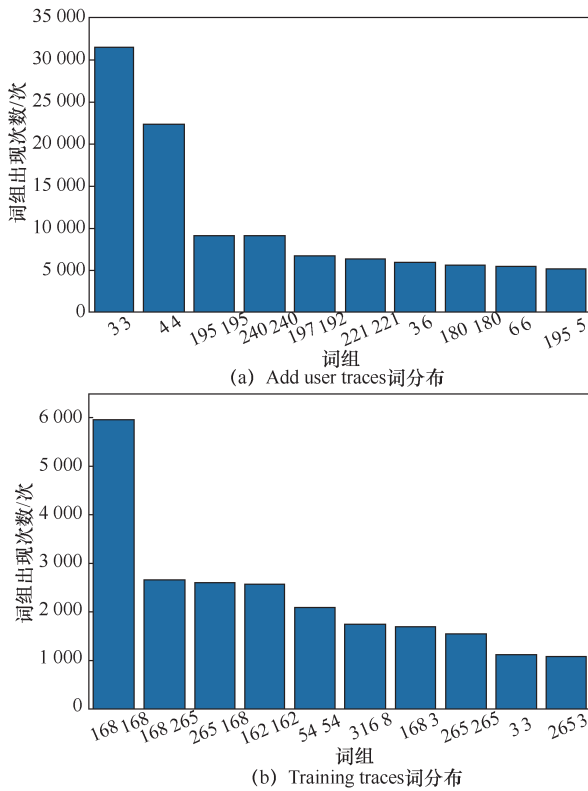


图7 Training traces 与 Add user traces 词频比较
Figure 7 Comparison of word frequency between training and add user traces

通过图7可见，以滑动窗口大小为2为例，“3 3”这个序列在正常系统调用序列中出现频次最高，而在以 Add user 入侵为例的系统调用中“168 168”这个序列出现频次最高。这在一定程度上表明两类数据集在滑动窗口为2这个层面上是线性可分的，本文对数据集中所有数据进行了上述实验，最终得出可以通过系统调用中的词频对系统产生的系统调用进行分类的结论。另外，通过实验，可以发现当滑动窗口大小为6时，各类攻击数据集之间以及攻击数据集与正常数据集之间，高频词的重叠率更低，即区分不同类型的能力更强，因此在实验中，选用大小为6的滑动窗口。

(2) 不同阈值对实验结果的影响

在评估指标中本文介绍了检测率和误报率是随着阈值的选取同步增长或同步减小的关系，因此通过实验得出所提检测方法在不同阈值时的检测率和误报率。在本文的检测方法中有两处使用了阈值，一处是特征提取阶段滑动窗口的大小，另一处是检测阶段对待分类文件进行分类时，选取多大比例的词作为分类时使用的数据。在 CIDM

检测方法可行性中，通过实验得到了词频可用于系统调用分类的结论，同时也得出了在窗口大小为6时，各类系统调用高频词的交集更少，更适合用于分类，因此在特征提取阶段选用大小为6的滑动窗口验证检测阶段待分类文件中选取高频词比例的阈值对实验结果的影响，在 ADFA-LD 数据集和 UNM 数据集上实验结果如图8所示。

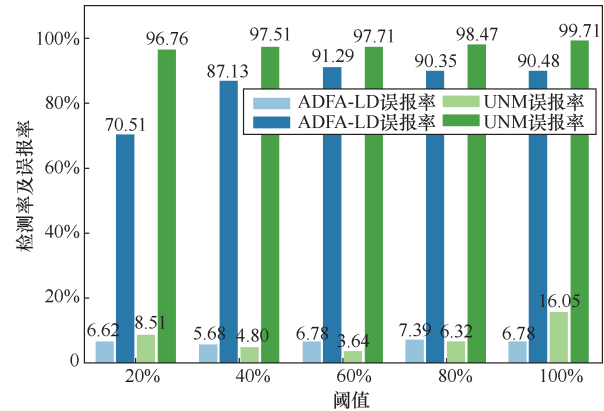


图8 检测率及误报率随不同阈值的变化
Figure 8 Change of detection rate and false alarm rate with different thresholds

通过实验可以发现，当阈值为待分类词的60%时，在两类数据集中有较高的检测率和较低的误报率，并且在 UNM 数据集中，检测率最高可达99.71%，误报率最低为3.64%。

另外，将本文提出的方法与基于传统机器学习算法实现的方法进行比较，比较在最高检测率时各方法的精确率、召回率、 F 值和准确率。在相关工作中，文献[19]比了各种特征提取算法与各种分类算法结合时在 ADFA-LD 数据集上的分类效果，最终得出使用 6-gram (N -gram, VN6) 提取特征向量结合一类高斯核支持向量机 (OC-SVM Gaussian Kernel) 或一类线性核支持向量机 (OC-SVM Linear Kernel) 时比基于 HMM 和 N -gram 模型的检测系统具有更高的检测精度结论。为了验证本文的方法，在 ADFA-LD 数据集上使用这3种方法进行实验，在实验中，本文将正常系统调用文件作为正样本，存在攻击行为的系统调用文件作为负样本。最终得到了在最高检测率时3种方法的精确率、召回率、 F 值和准确率，实验结果如表5所示。

通过3组数据可以看出，本文的方法在精确

表 5 不同方法检测性能对比
Table 5 Comparison of detection performance of different methods

方法	评价指标			
	精确率	召回率	F 值	准确率
CIDM	0.994	0.951	0.972	0.952
VN6+OC-SVM Linear Kernel	0.988	0.895	0.939	0.900
VN6+OC-SVM Gaussian Kernel	0.981	0.961	0.971	0.949

率值、 F 值和准确率上比 VN6+OC-SVM Linear Kernel 高出 0.006、0.033、0.052，比 VN6+OC-SVM Gaussian Kernel 高出 0.013、0.001、0.003。其次，本文方法虽然在召回率上比 VN6+OC-SVM Gaussian Kernel 低，但远远超过 VN6+OC-SVM Linear Kernel。通过对比还可看出，使用 OC-SVM Linear Kernel 和 OC-SVM Gaussian Kernel 会在不同指标上产生波动，如 OC-SVM Linear Kernel 在精确率上达到最好，却在其他指标中远远低于 OC-SVM Gaussian Kernel，相比之下，本文的方法具有良好的稳定性，能够在多个指标中取得最优或第二的性能。

5 结束语

本文提出了一种基于主机系统调用频率的容器入侵检测方法。针对容器运行时的安全问题，从容器运行时产生的系统调用中及时发现容器中的入侵行为，减少外部攻击带来的危害，保证了容器和容器宿主机主机的安全。在特征提取阶段，结合基于主机系统调用序列和基于主机系统调用频率的优点，采用滑动窗口和优化后 TF-IDF 算法结合的方式，提取正常操作和入侵操作产生系统调用的特征作为分类时的标准，对容器运行时产生的系统调用进行分类。通过实验结果可以得出，该方法在 ADFA-LD 数据集和 UNM 数据集上检测的准确率可达 99.71%。但是还有一些工作可以在后续继续展开，具体如下。

(1) 本文旨在实时检测运行的容器中是否存在入侵行为，并未在容器创建时对容器的安全性进行检测，在之后的工作中可以参考文献[20]的思路对创建容器镜像的 Dockerfile 进行检测，判断是否存在恶意代码，从而从镜像层面保证创建的容器是安全的。

(2) 本文提出的方法仅仅预测容器中操作是否异常，未对异常进行分类和规避，在之后的工

作中，可以对检测到的异常进行具体分类，如 DoS 攻击、容器逃逸等，另外借鉴文献[21]的思路，当检测到容器异常后，通过容器的运行时实时迁移来规避攻击。

(3) 目前使用的分类手段是根据特征值相似度进行分类，在之后的工作中可以结合机器学习的方法对提取的特征进行分类。

(4) 目前使用的 ADFA-LD 数据集存在数据不平衡的问题，在之后的工作中可以对数据集进行重采样。

参考文献:

- [1] GAO X, GU Z, KAYAALP M, et al. ContainerLeaks: Emerging security threats of information leakages in container clouds[C]//2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). 2017: 237-248.
- [2] COMBE T, MARTIN A, DI PIETRO R. To docker or not to docker: A security perspective[J]. IEEE Cloud Computing, 2016, 3(5): 54-62.
- [3] SHU R, GU X, ENCK W. A study of security vulnerabilities on docker hub[C]//Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy. 2017: 269-280.
- [4] 周天昱, 申文博, 杨男子, 等. Docker 组件间标准输入输出复制的 DoS 攻击分析[J]. 网络与信息安全学报, 2020, 6(6): 45-56.
- [5] ZHOU T Y, SHEN W B, YANG N Z, et al. Analysis of DoS attacks on Docker inter-component stdio copy[J]. Chinese Journal of Network and Information Security, 2020, 6(6): 45-56.
- [6] MARTIN A, RAPONI S, COMBE T, et al. Docker ecosystem-vulnerability analysis[J]. Computer Communications, 2018, 122: 30-43.
- [7] FLORA J, ANTUNES N. Studying the applicability of intrusion detection to multi-tenant container environments[C]//2019 15th European Dependable Computing Conference (EDCC). 2019: 133-136.
- [8] MANU A R, PATEL J K, AKHTAR S, et al. A study, analysis and deep dive on cloud PaaS security in terms of Docker container security[C]//2016 international conference on circuit, power and computing technologies (ICCPCT). 2016: 1-13.
- [9] YASRAB R. Mitigating docker security issues[J]. arXiv preprint arXiv:1804.05039, 2018.
- [10] ALARIFI S S, WOLTHUSEN S D. Detecting anomalies in IaaS environments through virtual machine host system call analysis[C]//2012 International Conference for Internet Technology and Secured Transactions. 2012: 211-218.
- [11] MASKE S A, PARVAT T J. Advanced anomaly intrusion detection technique for host based system using system call patterns[C]//2016 International Conference on Inventive Computation Technologies (ICICT). 2016, 2: 1-4.

- [11] JIAN Z, CHEN L. A defense method against docker escape attack[C]//Proceedings of the 2017 International Conference on Cryptography, Security and Privacy. 2017: 142-146.
- [12] FORREST S, HOFMEYR S A, SOMAYAJI A, et al. A sense of self for unix processes[C]//Proceedings 1996 IEEE Symposium on Security and Privacy. 1996: 120-128.
- [13] WARRENDER C, FORREST S, PEARLMUTTER B. Detecting intrusions using system calls: alternative data models[C]//Proceedings of the 1999 IEEE symposium on security and privacy (Cat. No. 99CB36344). 1999: 133-145.
- [14] HOFMEYR S A, FORREST S, SOMAYAJI A. Intrusion detection using sequences of system calls[J]. Journal of computer security, 1998, 6(3): 151-180.
- [15] KANG D K, FULLER D, HONAVAR V. Learning classifiers for misuse and anomaly detection using a bag of system calls representation[C]//Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop. 2005: 118-125.
- [16] ABED A S, CLANCY C, LEVY D S. Intrusion detection system for applications using linux containers[C]//International Workshop on Security and Trust Management. 2015: 123-135.
- [17] SRINIVASAN S, KUMAR A, MAHAJAN M, et al. Probabilistic real-time intrusion detection system for docker containers[C]//International Symposium on Security in Computing and Communication. 2018: 336-347.
- [18] KHREICH W, KHOSRAVIFAR B, HAMOU-LHADJ A, et al. An anomaly detection system based on variable N-gram features and one-class SVM[J]. Information and Software Technology, 2017, 91: 186-197.
- [19] 董玲, 张宏莉, 叶麟. 基于系统调用序列分析入侵检测的层次化模型[J]. 智能计算机与应用, 2014, 4(4): 13-16.
DONG L, ZHANG H L, YE L. A new study of the system call sequence analysis method[J]. Intelligent Computer and Applications, 2014, 4(4): 13-16.
- [20] 张东, 张尧, 刘刚, 等. 基于机器学习算法的主机恶意代码检测技术研究[J]. 网络与信息安全学报, 2017, 3(7): 25-32.
ZHANG D, ZHANG Y, LIU G, et al. Research on host malware detection using machine learning[J]. Chinese Journal of Network and Information Security, 2017, 3(7): 25-32.
- [21] AZAB M, MOKHTAR B M, ABED A S, et al. Smart moving target defense for linux container resiliency[C]//2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC). 2016: 122-130.

[作者简介]



季一木 (1978-), 男, 江苏南京人, 南京邮电大学计算机学院教授、博士生导师, 主要研究方向为 P2P 网络、云计算和大数据安全等。



杨卫东 (1994-), 男, 河南周口人, 南京邮电大学硕士生, 主要研究方向为容器安全。



李奎 (1988-), 男, 江苏南通人, 南京邮电大学博士生, 主要研究领域为高性能计算、大数据理论与技术。



刘尚东 (1979-), 男, 甘肃永靖人, 南京邮电大学讲师, 主要研究方向为网络行为分析、大数据处理等。



刘强 (1985-), 男, 安徽蚌埠人, 南京邮电大学讲师, 主要研究方向为图计算、大数据处理等。



邵思思 (1997-), 女, 江苏东海人, 南京邮电大学硕士生, 主要研究方向为边缘计算、云计算和大数据安全。



尤帅 (1995-), 男, 安徽芜湖人, 南京邮电大学硕士生, 主要研究领域为高性能计算、大数据和边缘计算。



黄乃娇 (1995-), 女, 河南信阳人, 南京邮电大学硕士生, 主要研究方向为机器学习和云计算安全。