基于长短期记忆神经网络的容器内进程异常行为检测

陈兴蜀[1,2],金逸灵[1,2],王玉龙[1,2],蒋 超[1,2],王启旭[1,2]
(1. 四川大学网络空间安全学院,四川成都 610065; 2. 四川大学网络空间安全研究院,四川成都 610065)

摘 要: 容器技术以其轻便、灵活和快速部署等特点提高了应用分发部署效率.然而,资源隔离性低和共享内核的特性却给容器和云平台引入了新的安全风险.本文提出了一种基于系统调用序列和长短期记忆(Long Short-Term Memory,LSTM)神经网络的容器内进程异常行为检测方案,通过无代理监控模式采集进程全生命周期的系统调用序列数据,并利用 LSTM 捕获序列的语义特征,同时采用局部窗口内累积偏差的方式,提出了两种异常判决方法.此外,为优化模型训练效率,设计了一种短序列样本同比去重算法.在公开数据集和复现的实际攻击场景下的实验结果表明,该方案能有效检出容器内进程的异常行为,且检测效果优于同类的其它方法.

关键词: 异常检测; 容器; 长短期记忆; 系统调用; 神经网络
中图分类号: TP309 **文献标识码:** A **文章编号:** 0372-2112 (2021)01-0149-08
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.12263/DZXB.20190220

Anomaly Detection of Processes Behavior in Container Based on LSTM Neural Network

CHEN Xing-shu[1,2],JIN Yi-ling[1,2],WANG Yu-long[1,2],JIANG Chao[1,2],WANG Qi-xu[1,2]
(1. *College of Cybersecurity,Sichuan University,Chengdu,Sichuan* 610065,*China*;
2. *Cybersecurity Research Institute,Sichuan University,Chengdu,Sichuan* 610065,*China*)

Abstract: Container technology improves the efficiency of application distribution and deployment with its features of lightness,flexibility and rapid deployment. However,the characteristics of low resource isolation and shared kernel introduce new security risks to containers and cloud platforms. This paper proposes an anomaly detection scheme of processes behavior in container based on system call sequences and long short-term memory (LSTM) neural network,the scheme collects the system call sequence data of the whole life cycle of processes through the agentless monitoring mode,and uses LSTM to capture the semantic features of sequences. At the same time,two methods of abnormal decision are proposed by means of cumulative deviation in local window. Furthermore,in order to optimize the training efficiency of the model,an algorithm for removing duplicate short sequence samples with the same ratio is designed. The experimental results on the public dataset and real attack scenarios show that the scheme can effectively detect the abnormal behavior of processes in container,and the detection performance is better than other similar methods.

Key words: anomaly detection;container;long short-term memory;system call;neural network

1 引言

容器技术作为一种新兴的虚拟化方式,提供了一个更加轻量的操作系统级别的虚拟主机环境.然而,在虚拟化平台广泛应用的同时,容器中的恶意进程却能利用容器或宿主机的漏洞实施攻击,进而威胁整个云平台的安全[1],故针对容器内进程进行安全监控和异常检测,对于维护云平台可用性和容器安全性等方面具有重要的现实意义.

作为应用程序与底层软硬件的中介,系统调用直接反映了进程访问系统资源的行为特征,进程的异常行为将在所执行的系统调用中有所体现[2,3].因此,系

收稿日期:2019-02-27;修回日期:2020-04-21;责任编辑:王天慧
基金项目:国家自然科学基金重点项目(No. U19A2081);国家自然科学基金青年科学基金(No. 61802270);国家"双创"示范基地之变革性技术国际研发转化平台(No. C700011);四川省重点研发资金(No. 2018G20100);国家自然科学基金联合基金(No. U19A2081)

统调用常被用作入侵检测的有效数据源[4]. 文献[5]利用机器学习方法验证了系统调用频率对异常识别的有效性,但忽略了数据的时序性. 文献[6]将 n-gram 提取的短序列及其频率映射为特征向量,训练了基于支持向量机的检测模型,取得了良好的检测效果,然而存在一个有意义的序列被错误拆分到不同子序列的可能性. 文献[7]与文献[8]将系统调用序列视为马尔科夫链,通过比较序列概率来评价进程行为的异常情况,但无法建模序列的长时间依赖关系,且在预测当前状态时仅考虑了部分历史信息. 文献[9]通过 strace 工具获取数据,并利用长短期记忆(Long Short-Term Memory, LSTM)神经网络捕捉长序列的语义信息,构建了一个语言分类模型,但需对 strace 采集的数据进行二次处理,还需根据序列长度建立不同的模型,灵活性较差.

上述方法都属于主机环境下针对特权进程的检测方案,此类方案需将检测范围限制在单个进程中以提高准确率,然而云计算环境下的容器和虚拟机与主机环境不同,其不仅存在着传统主机中的安全风险,还存在诸如逃逸攻击[1,10]等复杂的安全威胁,所以仅监控系统中的某个进程,往往无法保证检测的全面性. 文献[11]研究了虚拟机的 IOCTL 系统调用,并建立了基于隐马尔科夫的检测模型,虽能很好地发现异常虚拟机,但无法溯源至进程级别. 文献[12]存储了每个进程的所有系统调用及其直接后继,通过增加训练数据可以显著降低误报率,但从原始数据的层次建立正常轮廓的鲁棒性差,且会显著降低检测性能. 文献[13]结合最近邻、引导聚类和决策树等方法构建了一个集成分类器,依赖于虚拟机中代理程序采集到的数据进行检测,但忽略了代理程序的安全性和所采集数据的有效性. 文献[14]通过无代理的方式,在虚拟机监视器中分析系统调用和网络流量的联合频率信息,以检测虚拟机之间的恶意进程,保证了检测数据源的安全性,但误报率较高.

本文针对现阶段容器环境下入侵检测方案的不足,提出了一种基于系统调用序列和深度学习的进程异常行为检测方案. 主要贡献如下:

(1)针对代理程序会扩大攻击面和可移植性差的问题,提出了一种以无代理的方式,在主机用户层实时感知容器内进程的创建、运行和消亡等行为,从而动态捕捉容器内进程全生命周期的系统调用数据,该方法无需修改容器和宿主机操作系统,无需任何关于容器内系统结构和相应服务的先验知识.

(2)针对传统方法对序列数据的预测只考虑了较少且并不完全准确的历史信息的问题,提出了一种基于 LSTM 的深度学习检测模型,该模型能够利用循环神经网络的自连接特性,有效提取系统调用序列的语义特征用于预测,能更准确地描述局部系统调用之间的规律性,并且不用考虑序列长度.

(3)在建模阶段,本文根据系统调用频数信息,同比缩减训练样本数,加快了模型的训练速度,并利用已构建好的模型测试训练数据,自定义检测阶段的短序列概率阈值,减少了人工参与的工作量. 在检测阶段,本文基于局部窗口内的累积偏差,提出了两种异常判决方法,在提高检测率的同时有效降低了误报率.

(4)最后在 Docker 容器虚拟化平台上实现了 基于长短期记忆神经网络的容器内进程异常行为检测(Container Process Behavior Detecting based on LSTM, CPBDL)原型系统,并利用正常和异常程序样本对系统的功能和性能进行了测试. 结果表明,CPBDL 能有效检出容器内存在的异常进程,且检测效果优于同类的其他方法.

2 CPBDL 系统设计

图 1 给出了 CPBDL 的总体架构,四个模块都部署在待测容器外部的主机用户层,功能如下.

(1)进程信息获取模块:在容器外部采用无代理的方式,获取该容器内需被监控进程的信息,包括正在运行的所有进程和特定于管理该容器的运行时载体进程 docker-containerd-shim 的全局 PID 信息,传给数据采集模块.

(2)数据采集模块:基于 ptrace 监控进程的系统调用执行,实时感知该容器内进程的创建、运行和消亡等行为,动态追踪并透明采集进程全生命周期的系统调用数据,同步保存到调用日志.

(3)数据建模模块:基于调用日志中正常进程产生的系统调用序列,构建基于 LSTM 的异常检测模型,以此刻画正常进程行为轮廓.

(4)异常检测模块:加载异常检测模型,计算待测进程的异常程度,并将检测出的异常进程信息和具体的异常短序列输出到检测日志.

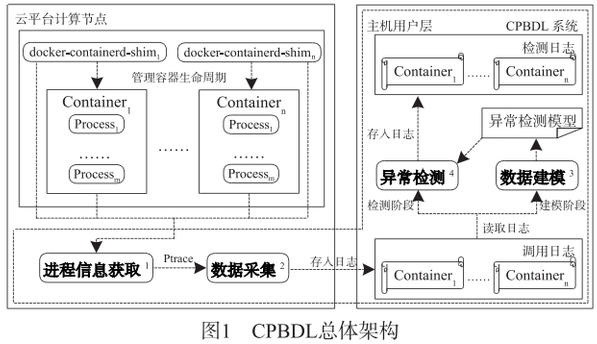


图1 CPBDL总体架构

3 CPBDL 系统实现

3.1 容器内进程的系统调用获取

为全面监控容器内运行进程的系统调用行为,本

文根据进程信息获取模块提供的被监控进程信息,实现的系统调用采集过程如图 2 所示. 其中,左侧为跟踪者执行流,右侧为被跟踪者执行流,两者之间的虚线分支表示跟踪者对应操作所触发的内核行为,内核执行返回后即可继续跟踪者自身的后续操作,实线连接表示内核通知跟踪者所用的信号,具体采集流程如下:

(1)数据采集模块作为跟踪者,通过 ptrace 系统调用,主动连接与待测容器相关的被跟踪进程,然后等待被跟踪者的陷入.

(2)当被跟踪者被内核暂停陷入到跟踪者的处理中时,跟踪者需为容器内正在运行的进程创建调用日志,但只需监控该容器的运行时载体进程的系统调用执行,无需采集其数据,然后设置自动连接到被跟踪者产生的子进程,同时激活系统调用的跟踪,此时内核恢复被跟踪者的执行,跟踪者则开始等待被跟踪者的再次陷入.

(3)恢复执行的被跟踪者在执行系统调用时会再次陷入到跟踪者的处理中,跟踪者可提取相应的陷入事件,如果是将要执行 execve 系统调用,则需判断该进程是否为载体进程的子进程 runc,若是,则表明该进程即将加载新程序,转变为容器内进程,此时才需为其创建调用日志,并开始采集其系统调用数据,否则直接进行数据采集工作,再让内核恢复被跟踪者的执行.

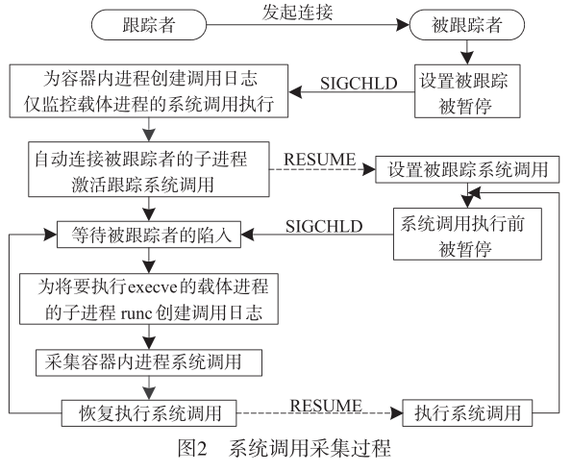


图2　系统调用采集过程

3.2　异常检测模型

进程正常运行时产生的系统调用序列具有相对稳定的局部规律性,而对该进程的某些攻击会破坏这种规律性,所以系统调用短序列能够在一定程度上反映进程正常和异常行为之间的差异[2]. 基于此思想,本文构建的异常检测模型框架如图 3 所示. 其中,涉及的几个定义如下:

定义 1　(系统调用序列) $Seq = <s_1, s_2, \cdots, s_L>$: $\forall k \in Z, 1 \leqslant k \leqslant L$ 有 $0 \leqslant s_k \leqslant N$, Seq 表示进程在运行过程中产生的系统调用数据的有序排列,L 为采集到的系统调用序列长度,N 为系统调用类型数,s_k 为第 k 个系统调用编号.

定义 2　(短序列) $ShortSeq_k = <s_k, s_{k+1}, \cdots, s_{k+T-1}>$: $\forall k, T \in Z, k+T-1 < L$, $ShortSeq_k$ 表示以 s_k 为起始的时间步长为 T 的系统调用短序列,短序列中的系统调用按序排列,若顺序不同,则为互异的短序列.

定义 3　(短序列概率) $ShortSeqProb_K = \prod_{i=k-M+1}^{k} Ps_{i+T}$: $\forall k, M \in Z, M \leqslant k \leqslant L\text{-}T$, $ShortSeqProb_k$ 表示模型根据 $<ShortSeq_{k-M+1}, ShortSeq_{k-M+2}, \cdots, ShortSeq_k>$ 预测的以 s_{k+T} 为结束的 M 窗口长度的目标输出系统调用短序列 $MShortSeq_k$ 的出现概率.

定义 4　(模型修正库) $ReviseLib = <s_i : ShortSeq_{i\text{-}T}, s_j : ShortSeq_{j\text{-}T}, \cdots, s_k : ShortSeq_{k\text{-}T}>$: $\forall i, j, k \in Z$ 有 $T < i, j, k \leqslant L$, $ReviseLib$ 表示模型针对训练样本预测错时,真实的系统调用和短序列的对应关系.

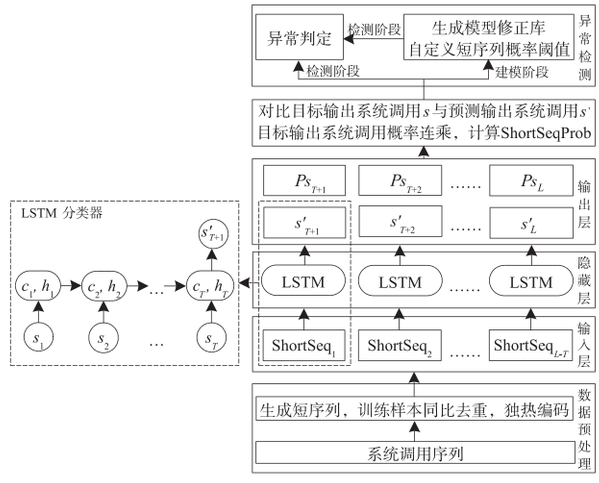


图3　异常检测模型框架

在建模阶段,首先,针对所有系统调用序列采用滑动窗口,以固定时间步长进行分割,生成短序列 *ShortSeq*,并基于每个 *ShortSeq* 与其对应的下一个系统调用 s 进行训练样本的同比去重处理,最后将训练样本进行独热编码以满足神经网络的输入层要求;然后,将经编码的 *ShortSeq* 输入到 LSTM 分类器中,其目标输出为该 *ShortSeq* 对应的经编码的下一个系统调用 s,实际输出为不同系统调用类型的概率分布向量(N 维),可据此取最大概率值对应的系统调用,即为预测输出的下一个系统调用 s',取目标输出 s 对应的概率值,即为分类器预测的目标输出系统调用概率 P_s. 分类器采用读取整个 *ShortSeq* 后产生单个特征输出的循环网络设计模式,从更抽象的层次提取 *ShortSeq* 的语义信息,利用预测输出的概率分布向量和经编码的目标输出向量定义

交叉熵损失函数,以此表征预测值和目标值之间的差距,应用 Adam 优化算法计算误差并不断更新网络权重,进而得到最终分类器;最后,针对长度为 L 的某个系统调用序列 Seq,以 T 作为时间步长,划分短序列,生成 L-T 个短序列 $<ShortSeq_1, ShortSeq_2, \cdots, ShortSeq_{L-T}>$ 与其对应的下一个系统调用 $<s_{T+1}, s_{T+2}, \cdots, s_L>$,将其编码后输入已构建好的 LSTM 分类器中,得到概率分布向量,据此获取短序列的预测输出系统调用 $<s'_{T+1}, s'_{T+2}, \cdots, s'_L>$ 和目标输出系统调用概率 $<P_{s_{T+1}}, P_{s_{T+1}}, \cdots, P_{s_L}>$. 针对 $<s'_{T+1}, s'_{T+2}, \cdots, s'_L>$,将其与 $<s_{T+1}, s_{T+2}, \cdots, s_L>$ 进行对比,以此生成模型修正库. 针对 $<P_{s_{T+1}}, P_{s_{T+1}}, \cdots, P_{s_L}>$,采用固定长度为 M 的滑动窗口进行按序分割,连乘窗口内的概率值,计算出所有 M 窗口长度的目标输出系统调用短序列 $MShortSeq$ 的出现概率 $ShortSeqProb$,并基于此自定义短序列概率阈值.

在检测阶段,读取待测容器调用日志,生成短序列,编码数据后输入建模阶段已构建好的 LSTM 分类器中,并将建模时自定义的短序列概率阈值和模型修正库应用于检测阶段,通过累积局部窗口内的异常短序列和异常值,进行异常判定.

3.2.1 短序列生成和训练样本同比去重

针对进程的系统调用序列,采用滑动窗口按序将其分割为固定长度为 T 的短序列和其对应的下一个系统调用,以此作为 LSTM 分类器训练时的输入和目标输出. 然而在训练阶段,为了构建尽可能完备的正常进程行为轮廓,需要采集进程多次运行产生的系统调用序列用于训练,这会导致生成大量重复短序列,增加了额外的训练时间. 针对该问题,本文实现了一个如图 4 所示的数据结构 Map,用于保存所有互异的短序列和其映射的不同系统调用,以及不同系统调用出现的频数,然后利用频数信息,同比去除重复样本,做到保留原始数据特征的同时节省训练所需时间开销.

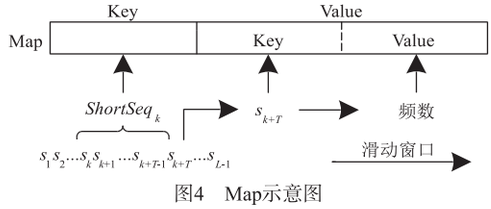


图4 Map示意图

同比去重并生成训练样本集的算法如算法 1.

算法 1 Algorithm GetSample( )

Input: *Seq*
Output: *Training Sample*
1: **for each** *s* **in** *Seq* **do**
2: *tmp_seq* ← *s*
3: **if** *len*(*tmp_seq*) > *T then* //序列长度
4: **if** *tmp_seq* **not in** *Sample* ***then***
5: *Sample* ← *tmp_seq*
6: Extract *ShortSeq* from *tmp_seq*
7: **if** *ShortSeq* **not in** Map **then**
8: Map←*ShortSeq*
9: **if** *s* **not in** Map[*ShortSeq*] **then**
10: Map[*ShortSeq*] ← *s*
11: Map[*ShortSeq*][*s*] ← 1
12: **else**
13: Map[*ShortSeq*][*s*] ++
14: Delete first value in *tmp_seq*
15: **end for**
16: **for each** *ShortSeq* **in** Map **do**
17: **if** *len*(Map[*ShortSeq*]) ! = 1 **then**
18: *gcd_v* ← *gcd*(Map[*ShortSeq*]) //最大公约数
19: **for each** *s* **in** Map[*ShortSeq*] **do**
20: *count* ← Map[*ShortSeq*][*s*] / *gcd_v* - 1
21: Combine *ShortSeq* and *s* to form a sample
22: Add *count* samples to Sample
23: **end for**
24: **end for**

3.2.2 容器内进程异常行为检测

在检测阶段,本文计算每个目标输出系统调用短序列的出现概率,概率越高表明该短序列越趋近于一个正常的进程系统调用序列. 然而,为了衡量短序列的异常程度,需要确定短序列概率阈值,多数方法[5,6,12,13]主要根据经验选定,或基于实际检测效果逐步修正,导致每个模型都需人工给定专属于该模型的概率阈值参数. 本文利用已构建好的模型测试训练数据,将最小的短序列概率作为模型的检测阈值 ε,针对训练样本中可能存在的进程行为不统一,导致预设阈值太低,进而产生异常漏报的问题,采用修正后模型预测的异常值统计方法进行二次检测,以提高检测率. 最后,基于异常局部性原理[3],累积局部窗口内的偏差,进行异常行为判定,并将容器内异常进程信息和相应的异常短序列输出到检测日志. 具体算法实现如算法 2.

算法 2 Algorithm Detection( )

Input: *Seq* of testing process
Output: Abnormal process and short sequences
1: Load normal model、*ReviseLib*、T、ε and M
2: Set W、θ_p、θ_v and θ_w
3: Input preprocessed testing *Seq* into model
4: Obtain *predictive output* and *target output probability*
5: Calculate *ShortSeqProb* of each *MShortSeq*
6: **for all** $ShortSeqProb_k$ **in** *ShortSeqProb* **do**
7: **if** $ShortSeqProb_k < \varepsilon$ **then**
8: Record abnormal $MShortSeq_k$
9: *count* ← Count abnormal *MShortSeq* in local W window

```
10:  if count > θ_p then
11:      abnormal_window_prob ++
12: end for
13: if abnormal_window_ prob > θ_w then
14:  Output abnormal process and short sequences
15: else
16:  for all s'_k in predictive output do
17:    if s'_k ≠ s_k then
18:      if s_k and ShortSeq_k-T not in ReviseLib then
19:        Record abnormal s_k
20:        count←Count abnormal s in local W window
21:        if count > θ_v then
22:          abnormal_window_value ++
23:  end for
24:  if abnormal_window_value > θ_w then
25:    Output abnormal process and short sequences
26:  else
27:    Output process is normal
```

其中,*ReviseLib* 为模型修正库,T 为时间步长,ε 为自定义的检测阈值,M 为短序列概率窗口长度,W 为局部窗口长度,θ_p和 θ_v分别为概率和异常值统计检测的局部窗口异常判决门限,θ_w为进程行为异常判决门限.

4　实验及结果分析

本节从功能及性能两方面对 CPBDL 进行评测. 本文将美国新墨西哥大学公开数据集和实时获取的容器内进程系统调用数据结合,构建容器环境下的测试数据集. 具体实验环境为:宿主机 CPU 型号为 Intel(R) Xeon(R) E5-2609 v2,主频为 2.50GHz,物理内存为 256GB,操作系统为64位CentOS7,内核版本为3.10.0,Docker版本为18.03.0-ce,深度学习框架为 Tensorflow 1.11.0.

4.1　模型时间步长确定

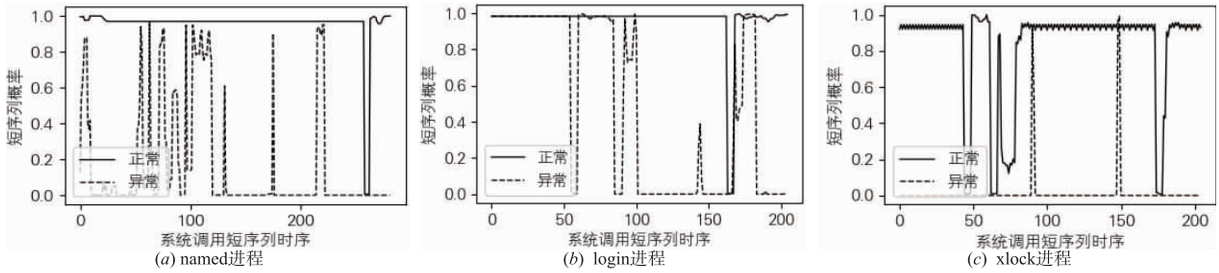
为了定量地评估不同时间步长对模型精度的影响,按照 8:2 的比例将构建的正常数据集划分为训练集和测试集,然后针对两个集合采用正确预测的系统调用占比作为度量标准. 在实验中设定 LSTM 分类器含 4 层隐藏层,每层节点数为 100,学习率为 0.005,批尺寸为 100,迭代次数为 100,由于此时还未涉及异常行为检测过程,因此无需确定剩余参数,得到的测试结果如表 1 所示. 由结果可知,时间步长取 15 时预测精度最高,达 91.24%,当时间步长小于 15 时,所用历史信息不足以确定下一个系统调用,而当时间步长大于 15 时,网络囊括了过多的无关信息,即下一个系统调用并不依赖与其相距太远的系统调用,导致模型精度下降. 因此,本文选择 15 作为时间步长来训练模型.

表 1　不同时间步长与模型精度的关系

时间步长 T	拟合精度/%	预测精度/%
5	89.39	88.82
10	89.51	89.25
15	**92.29**	**91.24**
20	89.15	86.92
25	88.80	85.88
30	76.32	69.15

4.2　进程系统调用行为轨迹

使用系统调用行为轨迹表示进程的系统调用短序列概率随时间变化的趋势. 图 5 给出了 named、login 和 xlock 进程的行为轨迹.



(*a*) named进程　(*b*) login进程　(*c*) xlock进程

图5　进程系统调用行为轨迹

在实验中,对于 named 和 login 进程,都是由短序列概率方式检测出的异常,而对于 xlock 进程,由于进程行为复杂或模型训练不够充分等实际应用中也可能存在的问题,导致自定义阈值太低,使得短序列概率方式没有检测出异常,但还是被二次异常值统计检测方法捕捉到了该进程的异常行为.

由结果可知,虽然存在因抽样不完全引起的部分匹配偏差,但正常进程的系统调用短序列概率相较于异常进程,更接近于训练模型的正常样本,多数短序列概率值接近于 1,而异常进程除部分在入侵发生前产生的正常行为外,大量数据均与正常行为轮廓不符,短序列概率值接近于 0,且异常行为呈现出局部区域内连续异常的趋势. 因此,本文采用的局部分析方法更有利于区分异常,并且可以避免在全局统计分析方法中,因进程在入侵发生前产生的大量正常行为掩盖入侵造成的异常这一问题.

4.3 功能实验

4.3.1 实际攻击检测

通过检测特定容器服务内的进程存在的异常行为说明系统在实际应用中的有效性. 针对 MySQL 容器,使用 mysqlslap 模拟 MySQL 服务器的客户端负载,以此生成 MySQL 容器服务的正常进程行为数据并构建训练模型,然后通过以下攻击实验生成异常进程行为数据.

(1)容器服务自身的异常行为. 使用 sqlmap 攻击 MySQL 容器内的数据库,导致在 MySQL 容器外部窃取到容器内数据库中的机密数据.

(2)容器内恶意程序对所在容器和宿主机的攻击. 通过在被感染容器中触发以下内核漏洞:CVE-2017-16995、CVE-2017-1000112 和 CVE-2017-1000253,导致攻击者由容器内普通用户提权为容器内特权用户,并可发起进一步攻击. 触发漏洞 CVE-2016-5195 使得攻击者实现容器逃逸,即由容器内普通用户获取到宿主机的执行权限,并可对云平台上的其他容器造成破坏.

表 2 给出了针对正常进程行为数据和上述攻击产生的异常进程行为数据的测试结果. 其中,异常短序列占比表示被判定为异常的短序列占总体短序列的比例. 由结果可知,异常进程行为数据产生了强烈的异常信号,异常短序列占比均在 93% 以上,而正常进程行为数据的异常短序列占比只有 1.31%. 由此可知,CPBDL 能够有效检出容器内存在的异常进程,发现云平台上潜在的安全风险.

表 2 实际攻击检测结果

实验描述	异常短序列占比/%
正常数据	1.31
Sqlmap 攻击	99.85
CVE-2017-16995	95.87
CVE-2017-1000112	93.27
CVE-2017-1000253	99.94
CVE-2016-5195	99.98

4.3.2 与其他方法的对比

本文使用正确检出的异常进程占比,即真正率评价检测率,使用正常进程被误检为异常进程的占比,即假正率评价误报率. 在实验中设定短序列概率窗口长度 M 为 6,局部窗口长度 W 为 20,通过调节窗口和进程异常判决门限,得到如表 3 所示的测试结果. 其中,由于数据集中存在多种进程,而模型针对不同进程的检测效果存在差异,因此取最差和最优结果作为表 3 中的评价结果.

表 3 与其他方法的对比

检测方法	系统环境	真正率/%	假正率/%
Bayes-like classifier based on Markov Model[15]	Host	60 ~ 99	0.56 ~ 3.40
OC-SVM Gaussian Kernel[6]	Host	80 ~ 93	5 ~ 10
Context-free grammar with ELM and SVM[5]	Host	81 ~ 88	2.1 ~ 3.9
Clustering combined with Bayesian classification[16]	Host	100	12.5
Kernel States Modeling[17]	Host	100	0 ~ 10
Key-value pair of system calls[12]	Cloud	0.82 ~ 100	0
Back propagation neural network with HMM[18]	Cloud	89 ~ 92	1.16 ~ 1.42
Introspection-Based Hybrid IDS[14]	Cloud	89 ~ 98	2.05 ~ 11.05
Ensemble-based classifiers[13]	Cloud	99.75	6.24
Modeling IOCTL with HMM[11]	Cloud	100	5.66
CPBDL proposed in this paper	**Cloud**	**89 ~ 100**	**0.29 ~ 0.55**

由结果可知,CPBDL 略优于表 3 中各检测方法,原因在于 CPBDL 有效结合神经网络,抽取了较长序列的语义信息,并采用局部分析的方式发现异常,可以更加稳定地表示进程的正常行为状态,使得对于异常行为具有较好的区分度. 其中,在相同检测率区间内,CPBDL 的误报率低于文献[5,6,13 ~ 15,18],而文献[11,16,17]的检测率存在优于 CPBDL 的情况,但在相同检测率时,误报率却高于 CPBDL,尽管文献[12]没有产生误报,但其检测区间较大,而 CPBDL 在误报率升高的同时,检测率变化幅度不大,且都保持在较高水平之上.

4.4 性能实验

在部署与未部署 CPBDL 的环境下,采用 UnixBench 微基准测试工具对被监控容器运行引入的性能损耗进行测试,结果如图 6 所示.

由结果可知,CPBDL 给容器内进程的运行引入了一定的性能开销,主要表现在文件传输、执行 execl 函数、进程创建和执行系统调用方面,原因在于 CPBDL 采

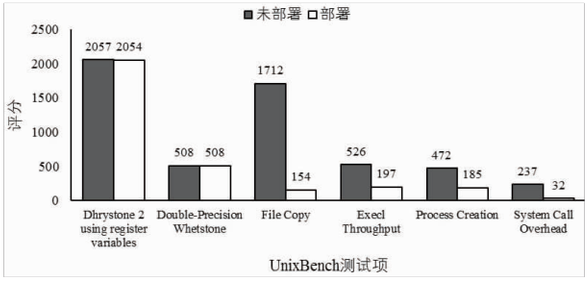


图6　UnixBench测试结果

用不侵入宿主机内核的用户层拦截方式采集进程的系统调用,并且为了在无代理监控期间,不漏掉任何进程的行为数据,需要实时感知容器内新建的所有进程,但考虑到无代理监控方式在保证容器服务透明性的同时,并未给容器引入额外的安全风险,以及做到了全面监控目标容器内所有进程的运行,系统所带来的性能损耗可以接受.

针对 named 进程,本文对同比去重训练样本的 CPBDL 和未做此操作的两种建模方式的训练短序列样本数、训练耗时和训练好的模型对测试集的预测精度进行了对比,结果如表 4 所示. 由结果可知,通过同比去重训练样本,CPBDL 在训练性能上有明显提高,且模型预测精度的降低可忽略不计.

表 4　同比去重训练样本对比结果

建模方式	训练短序列样本数	训练耗时/s	预测精度/%
CPBDL	**13208**	**357.73**	**99.79**
未同比去重	28552	558.09	99.83

5　结语

本文针对容器环境下入侵检测方案的不足,有效结合深度学习的优势,提出了一种基于系统调用序列和 LSTM 的容器内进程异常行为检测方案,对当前容器环境下的威胁防御具有重要的现实意义. 检测系统采用无代理的方式监控容器内所有进程全生命周期的系统调用行为,具有较好的可移植性和安全性,利用 LSTM 自动提取经同比去重的短序列样本的语义特征,并通过局部分析的方式发现异常,能更稳定地表示进程系统调用行为的规律性,使得在提升检测率的同时有效降低了误报率.

参考文献

[1] MATTETTI M,SHULMAN-PELEG A,ALLOUCHE Y,et al. Securing the infrastructure and the workloads of linux containers[A]. IEEE Conference on Communications and Network Security [C]. Florence, Italy: IEEE, 2015. 524 –526.

[2] FORREST S,HOFMERYR S A,SOMAYAJI A,et al. A sense of self for unix processes[A]. Proceedings of the 1996 IEEE Symposium on Security and Privacy[C]. Oakland:IEEE,1996. 120 –128.

[3] KOSORESOW A P,HOFMEYR S A. Intrusion detection via system call traces[J]. Software IEEE,1997,14(5):35 –42.

[4] MISHRA P,PILLI E S,VARADHARAJAN V,et al. Intrusion detection techniques in cloud environment: a survey [J]. Journal of Network and Computer Applications,2017, 77(C):18 –47.

[5] MASKE S A,PARVAT T J. Advanced anomaly intrusion detection technique for host based system using system call patterns[A]. International Conference on Inventive Computation Technologies[C]. Coimbatore,India:IEEE,2017. 1 –4.

[6] 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.

[7] XIAO X,WANG Z,LI Q,et al. Back-propagation neural network on markov chains from system call sequences: a new approach for detecting android malware with system call sequences[J]. Iet Information Security,2017,11(1):8 –15.

[8] 尹清波,张汝波,李雪耀,等. 基于动态马尔科夫模型的入侵检测技术研究[J]. 电子学报,2004,32(11):1785 –1788.
YIN Qing-bo,ZHANG Ru-bo,LI Xue-yao,et al. Research on technology of intrusion detection based on dynamic markov model[J]. Acta Electronica Sinica,2004,32(11): 1785 –1788. (in Chinese)

[9] Xiao X,Zhang S,MERCALDO F,et al. Android malware detection based on system call sequences and LSTM[J]. Multimedia Tools and Applications,2017,2:1 –21.

[10] 陈兴蜀,陈佳昕,赵丹丹,等. 基于虚拟机 IO 序列与 Markov 模型的异常行为检测[J]. 清华大学学报(自然科学版),2018,58(4):395 –401 +410.
CHEN Xing-shu,CHEN Jia-xin,ZHAO Dan-dan,et al. Anomaly detection based on io sequences in a virtual machine with the markov mode[J]. Journal Tsinghua University (Science and Technology) ,2018,58(4):395 – 401 +410. (in Chinese)

[11] ALARIFI S,WOLTHUSEN S. Anomaly detection for ephemeral cloud iaas virtual machines[A]. International Conference on Network and System Security[C]. Berlin Heidelberg:Springer,2014. 321 –335.

[12] GUPTA S,KUMAR P. An immediate system call se-

quence based approach for detecting malicious program executions in cloud environment[J]. Wireless Personal Communications,2015,81(1):1 – 21.

[13] TAHIR R,RAZA A,NAQVI M,et al. An anomaly detection fabric for clouds based on collaborative vm communities[A]. IEEE/ACM International Symposium on Cluster,Cloud and Grid Computing[C]. Madrid,Spain:IEEE,2017. 431 – 441.

[14] KASHYAP A,KUMAR G S,JANGIR S,et al. IHIDS:Introspection-based hybrid intrusion detection system in cloud environment[A]. International Conference on Advances in Computing, Communications and Informatics [C]. Udupi,India:IEEE,2017. 687 – 693.

[15] ELGRAINI M T,ASSEM N,RACHIDI T. Host intrusion detection for long stealthy system call sequences[A]. Information Science and Technology[C]. Fez, Morocco: IEEE,2012. 96 – 100.

[16] KOUCHAM O,RACHIDI T,ASSEM N . Host intrusion detection using system call argument-based clustering combined with bayesian classication[A]. Sai Intelligent Systems Conference[C]. London,UK:IEEE,2015. 1010 – 1016.

[17] MURTAZA S S,KHREICH W,HAMOULHADJ A,et al. A host-based anomaly detection approach by representing system calls as states of kernel modules[A]. IEEE International Symposium on Software Reliability Engineering[C]. Pasadena,CA,USA:IEEE,2014. 431 – 440.

[18] 黄杰. 云环境虚拟机安全关键技术研究与实现[D]. 四川成都:电子科技大学,2017.
HUANG Jie. Research and implementation of virtual machine security key technology on cloud environment[D]. Chengdu, Sichuan: University of Electronic Science and Technology of China,2017. (in Chinese)

作者简介

陈兴蜀 女,1968 年 8 月出生,贵州六枝人,四川大学教授、博士生导师,主要研究方向:云计算/大数据安全,威胁检测,开源情报.
E-mail:chenxsh@scu.edu.cn

金逸灵(通信作者) 女,1995 年 9 月出生,安徽安庆人,四川大学硕士研究生,主要研究方向:云计算,虚拟化安全.
E-mail:980262177@qq.com