doi :10.3969/j.issn.1671-1122.2024.05.001

基于虚拟机自省的 Linux 恶意软件检测方案

文伟平，张世琛，王晗，时林
（北京大学软件与微电子学院，北京 100871）

摘　要：随着物联网和云计算技术的快速发展，Linux 恶意软件的数量和种类急剧增加，因此如何有效检测 Linux 恶意软件成为安全领域的重要研究方向之一。为了解决这一问题，文章提出一种基于虚拟机自省的 Linux 恶意软件检测方案。该方案利用虚拟机自省技术在沙箱外部安全获取内部运行状态，在实现全方位监控的同时，规避了恶意软件的反动态分析问题。与其他沙箱监控方案相比，文章所提方案增加了恶意软件在沙箱中的恶意行为表现的数量。针对特征之间的时序性，采用时序处理模型对沙箱获取的特征信息进行建模和训练，旨在判断 Linux 应用是否属于恶意软件。文章使用了 3 种神经网络，包括循环神经网络、长短期记忆网络和门控循环单元网络。实验结果表明，长短期记忆网络在该应用场景下检测效果更好，准确率达 98.02%，同时具有较高的召回率，将虚拟机自省技术与神经网络模型结合应用于恶意软件检测，既能在虚拟机外部监控虚拟机内部，又考虑特征之间的时序性。

关键词：恶意软件检测；虚拟机自省；深度神经网络；Linux 沙箱

中图分类号：TP309　**文献标志码：**A　**文章编号：**1671-1122（2024）05-0657-10

中文引用格式：文伟平, 张世琛, 王晗, 等 . 基于虚拟机自省的 Linux 恶意软件检测方案 [J]. 信息网络安全, 2024，24（5）：657-666.

英文引用格式：WEN Weiping, ZHANG Shichen, WANG Han, et al. Linux Malicious Application Detection Scheme Based on Virtual Machine Introspection[J]. Netinfo Security, 2024, 24(5): 657-666.

Linux Malicious Application Detection Scheme Based on Virtual Machine Introspection

WEN Weiping, ZHANG Shichen, WANG Han, SHI Lin
(*School of Software and Microelectronics*, *Peking University*, *Beijing* 100871, *China*)

Abstract: With the rapid development of the Internet of things and cloud computing technology, the number and type of Linux malware have increased dramatically. Therefore, how to effectively detect Linux malware has become one of the important research directions in the security field. To solve this problem, this paper proposed a Linux malicious application detection scheme based on virtual machine introspection. This scheme utilized the virtual

收稿日期：2023-08-15
基金项目：国家自然科学基金 [61872011]
作者简介：文伟平（1976—），男，湖南，教授，博士，主要研究方向为系统与网络安全、大数据与云安全、智能计算安全；张世琛（2000—），男，山东，硕士研究生，主要研究方向为恶意代码检测、漏洞挖掘；王晗（2000—），男，山东，硕士研究生，主要研究方向为软件与系统安全；时林（1998—），男，山东，硕士研究生，主要研究方向为漏洞挖掘、软件安全防护。
通信作者：文伟平 weipingwen@pku.edu.cn

machine introspection technology to securely obtain the internal running status outside the sandbox, realized all-round monitoring while avoiding the anti-dynamic analysis technology of malware at the same time. Compared to other sandbox monitoring methods, this scheme improved malware performance in the sandbox. In order to pay more attention to the timing between features, a timing processing model was used to model and train the feature information obtained by the sandbox, aiming to judge whether a Linux application was malicious. In this paper, three kinds of neural network were used, including recurrent neural network, long short-term memory network and gated recurrent unit network. The experimental results show that the long short-term memory network works better in this application scenario, with an accuracy rate of 98.02% and a higher recall rate. The innovation of this paper is that the combination of virtual machine introspection technology and neural network model is applied to malicious application detection, which can not only monitor the inside of the virtual machine outside the virtual machine, but also pay attention to the timing between features.

Key words: malicious application detection; virtual machine introspection; deep neural network; Linux sandbox

0 引言

随着互联网的高速发展，网络安全事件频繁发生，根据发布的《2020年中国互联网网络安全报告》[1]和《2020年我国互联网网络安全态势综述》[2]，CNCERT/CC捕获了34.8万个恶意软件家族共4000多万个恶意软件样本，其中新增恶意代码家族235个，全年恶意软件样本捕获数量整体呈上涨趋势，监测到的恶意软件传播次数为17.5亿余次。安全领域之前主要针对Windows的恶意软件进行研究，但随着物联网设备的增多和云平台的广泛应用，Linux恶意软件正迅速改变恶意软件的格局。根据Atlas VPN团队分析的数据[3]，Linux恶意软件数量逐年上升，2022年超过190万个。恶意软件数量趋势如图1所示。

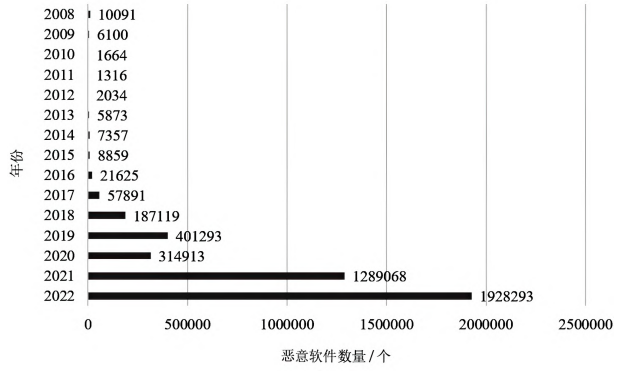

图1 恶意软件数量趋势

恶意软件分析主要包括静态分析[4]和动态分析[5]两种方法。静态分析从程序本身入手，对程序进行反汇编，根据逆向工程[6]获得的信息检测威胁。但由于恶意软件开发者采用加壳和加密措施保护二进制程序，这增加了静态分析的难度。为了弥补静态分析的不足，研究人员提出恶意软件动态分析技术，通过观察程序执行过程中的行为分析恶意软件。沙箱作为一个动态分析的优秀工具，对分析人员的要求低，具有可批量、自动化运行的优势。现有沙箱对Linux恶意软件的检测效果并不理想，随着恶意软件的威胁从计算机系统扩展到物联网，开发高效、精准的自动化分析和检测方法已成为安全领域的一个研究热点。

本文基于虚拟机自省（Virtual Machine Introspection，VMI）技术的Linux恶意软件检测，构建了Linux恶意软件分析沙箱，通过对比实验分析了不同神经网络模型在恶意软件检测场景下的表现，本文主要工作如下。

1）针对业界Linux恶意软件检测工具的不足，构建了一个基于虚拟机自省技术的沙箱，实现无感知、强隔离、全方位获取Linux恶意软件的动态行为信息[7]。

2）对于提取的行为信息不注重时序信息的问题，采用多种时序处理模型的神经网络算法进行建模，通过对比实验发现，长短期记忆网络（Long Short-Term Memory，LSTM）模型表现最佳。对LSTM网络的batch_size、epoch和学习率3个参数进行优化，提高了模型的准确率和召回率。

3）验证了将虚拟机自省技术与神经网络模型结合的有效性，并与业界其他主机侧的沙箱进行对比，证明本文提出的方案在准确率方面优于业界其他主机侧的沙箱。

1 相关技术

1.1 虚拟机自省

虚拟机自省是从虚拟机外部监控虚拟机内部状态的技术，其体系架构如图2所示，这项技术已经在恶意软件分析、入侵检测系统和内存取证等方面得到广泛应用[8]。

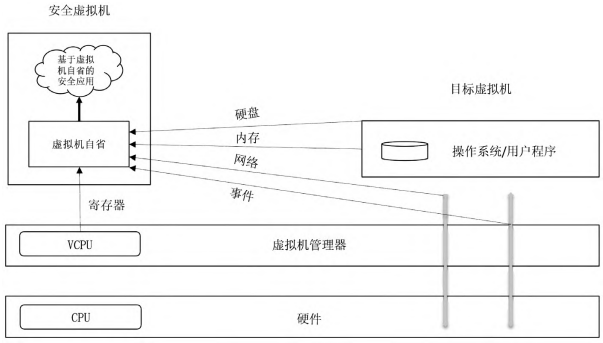

图 2 虚拟机自省技术体系架构

随着虚拟化技术的发展，产生了虚拟机监控程序，该程序也被称为虚拟机监视器（Virtual Machine Monitor，VMM），运行在物理层和操作系统层之间，在一台物理计算机上可以同时运行多个独立的操作系统，从而更有效地利用可用的存储空间和网络带宽等[9]。虚拟机监控程序除了将计算范式从多任务推进到多操作系统，还将系统监控从传统的虚拟机内监控推进到虚拟机之外，不需要在虚拟机内添加检测模块就可以监控虚拟机内部的操作。VMM本质上是一个软件层，更容易被修改、迁移和监视，被监控的操作系统在VMM提供的虚拟资源中运行，增强了灵活性和监控能力。通过在VMM层提取和重构被监控操作系统的状态，使监控系统能够从外部控制、隔离、干预、检查、保护和管理虚拟机。

1.2 LSTM

LSTM是循环神经网络（Reurrent Neural Network，RNN）的一种变体，由HOCHREITER[10]提出，旨在解决传统RNN在处理长序列任务时遇到的难题。LSTM通过引入记忆单元机制，能够有效捕捉和记忆长期依赖关系，因此在处理序列数据时表现出色。

传统RNN在处理长序列数据时常面临梯度消失或梯度爆炸的问题，这使得难以有效学习长期依赖关系。为了解决这个问题，LSTM引入输入门、遗忘门和输出门3个关键的门控机制以及记忆细胞。记忆细胞是LSTM的核心，负责存储和传递信息。遗忘门决定遗忘前一时刻记忆细胞中的哪些信息，输入门决定记住当前输入的哪些信息。通过调节遗忘门和输入门的开关状态，LSTM能够有选择性地忽略或存储信息，有效地传递长期依赖关系。LSTM隐藏单元结构如图3所示。

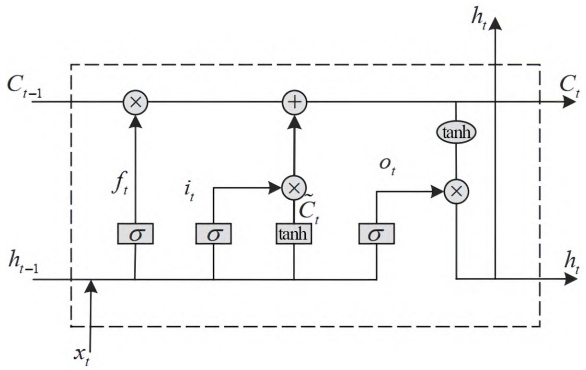

图 3 LSTM 隐藏单元结构

各组成部分的计算如公式（1）~公式（6）所示。

$$f_t = \sigma(W_f[\boldsymbol{x}_t, h_{t-1}] + b_f) \quad (1)$$

$$i_t = \sigma(W_i[\boldsymbol{x}_t, h_{t-1}] + b_i \quad (2)$$

$$o_t = \sigma(W_o[\boldsymbol{x}_t, h_{t-1}] + b_o \quad (3)$$

$$\tilde{C}_t = \tanh(W_C[\boldsymbol{x}_t, h_{t-1}] + b_C) \quad (4)$$

$$C_t = f_t C_{t-1} + i_t \tilde{C}_t \quad (5)$$

$$h_t = \tanh(C_t) o_t \quad (6)$$

其中，$\boldsymbol{x}_t$为t时刻输入LSTM的词向量，f_t、i_t和o_t分别为遗忘门、输入门和输出门，h_{t-1}和h_t分别为上一时刻和当前时刻的隐藏状态，C_{t-1}和C_t分别为上一时刻和当前时刻记忆单元状态。LSTM通过3个门控机制保留长序列的早期信息，并且舍弃不重要的信息，克服了传统RNN

的不足，显著提升了序列问题的检测性能[11]。

LSTM能够有效处理长期依赖关系，因此被广泛应用于处理序列相关的任务，如自然语言处理（包括语言建模、机器翻译等）、语音识别、时间序列预测等，其设计思想也为构建更复杂的RNN模型提供了参考，如门控循环单元（Gated Recurrent Unit，GRU）[12]。

2 系统详细设计

2.1 VMI Sandbox

目前，常用的开源沙箱、商用和在线文件分析平台针对Linux系统中ELF文件的恶意代码样本分析能力较弱，导致无法获取样本的全部恶意行为。针对恶意应用，本文构建了VMI Sandbox对Linux ELF文件进行分析。

2.1.1 VMI Sandbox 架构

VMI Sandbox架构如图4所示，Host在运行期间通过接收Guest发送的指令控制Target开关机、恢复快照等操作，流量监控模块负责抓包，并将流量包传回Guest。Guest是运行沙箱并进行监控的上层虚拟机，VMI监控模块负责处理Target机器产生的VM-exit事件、记录系统调用参数和返回值、检测其他异常情况等。控制模块主要负责控制VMI监控模块的开关并记录日志，同时与Host进行交互。Target是样本实际运行的虚拟机，其中运行了待分析的程序。

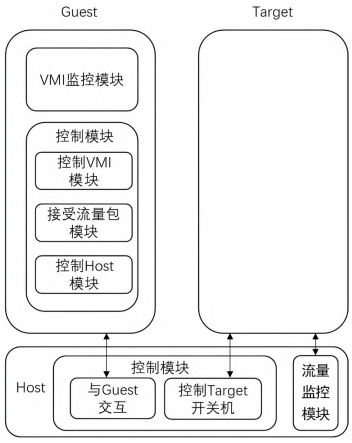

图 4 VMI Sandbox 架构

2.1.2 VMI Sandbox 通信流程

样本检测流程如图5所示，首先启动VMI监控程序，从样本目录中选择一个待测样本，从参数目录中查找样本相应的参数文件，启动Target机器，样本和具体参数利用VMI的文件注入Target机器中并执行，样本运行期间发生的系统调用信息会被Guest记录。样本运行结束后，将运行期间产生的流量包回传给Guest。

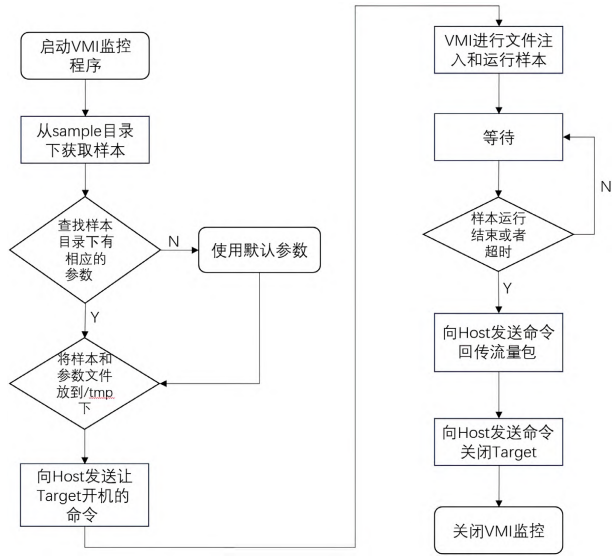

图 5 样本检测流程

2.1.3 VMI Sandbox 虚拟机自省部分的设计

为了实现监控的透明性和无感知性，本文采用基于高特权Hypervisor/VMM的自省方案HVMI[13]。HVMI是一种功能完备的开源虚拟机自省框架，其监控引擎在一个特殊的VM中运行。HVMI通过对目标系统关键信息进行提取解析来解决语义鸿沟问题，并提供内存监控、进程监控的能力。但HVMI引擎未提供对系统调用进行hook的能力，而系统调用是恶意代码体现恶意行为的关键信息。为了解决上述问题，本文对HVMI监控引擎进行改造，添加了关键系统调用对应内核函数的hook处理，并完成系统调用参数解析的工作。由于虚拟机自省程序工作在Hypervisor层，其对内核函数的hook对于上层操作系统是无感知的，运行在操作系统上的恶意代码无法发现内核函数被hook的情况。由此可见，基于虚拟机自省的沙箱能绕过恶意代码对运行环境的检测，激发更多的恶意行为。

本文基于LibVMI，通过编写LibVMI脚本在hypervisor

层面监控客户机的状态，并在客户机执行系统调用时获取客户机的寄存器和内存等信息，以获取系统调用对应的参数。由于VMI与操作系统的隔离性，运行在操作系统层面的恶意代码无法感知监控层，可以有效避开对抗行为。VMI内省流程如图6所示，具体包括：①VMI应用请求查看内核符号；②LibVMI通过内核符号文件找到符号的虚拟地址；③内核页目录映射到正确的页表；④页表映射到正确的页；⑤相应的数据页返回LibVMI库；⑥LibVMI返回VMI应用程序请求的数据，可能需要映射多个页面。

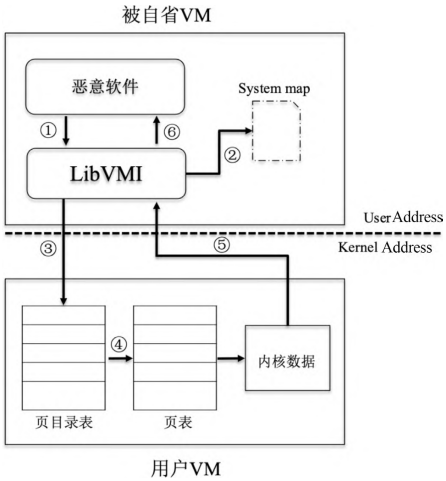

图 6 VMI 内省流程

1）系统调用特征提取

在x86_64中，MSR寄存器负责保存系统调用，进入系统调用时的MSR_SYSENTER_EIP是固定的，利用vmi_register_event注册一个内存访问event，当系统调用执行MSR_SYSENTER_EIP地址代码时，内存访问事件被触发，利用LibVMI读取寄存器的vmi_get_vcpureg对VM的寄存器信息进行读取，使用vmi_read_pa API读取内存信息，进而获取系统调用信息。系统调用序列监控逻辑示意图如图7所示。

2）进程追踪

在x86_64中，进程上下文切换时会更新CR3寄存器，每个进程对应的CR3寄存器的值唯一，因此可以通过vmi_register_event注册一个寄存器写event，监控CR3值的变化。当CR3值改变时触发event，调用vmi_

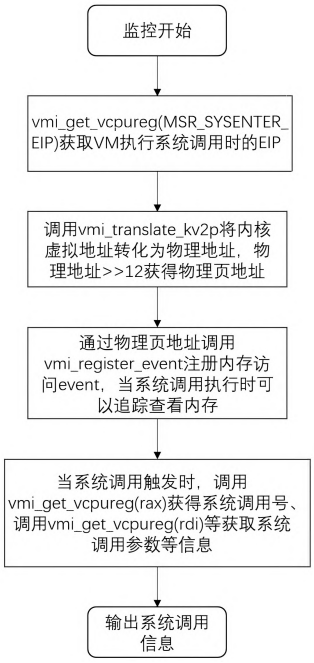

图 7 系统调用序列监控逻辑示意图

dtb_to_pid获得进程PID并记录下来。进程追踪模块监控逻辑示意图如图8所示。

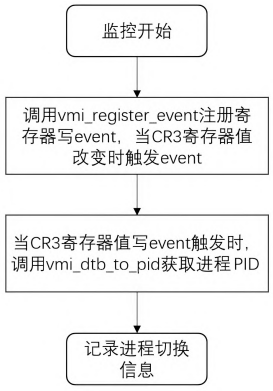

图 8 进程追踪模块监控逻辑示意图

2.2 Linux 恶意软件检测方案设计与实现

Linux恶意软件检测方案设计与实现包括系统的总体设计、功能实现以及检测系统的总体运行流程。

2.2.1 总体设计

基于虚拟机自省的恶意软件检测系统包含5个模块，分别是基于沙箱的行为获取模块、特征提取和预处理模块、神经网络训练模块、恶意应用检测模块和报告生成模块，其中基于沙箱的行为获取模块包括进

程追踪和系统调用两个部分[14,15]，用于获取样本的动态信息。特征提取和预处理模块根据沙箱提取的日志进行特征提取，生成样本集的特征矩阵。神经网络训练模块基于时序型神经网络模型进行训练和优化，生成最终模型。恶意应用检测模块使用最终模型对未知恶意样本进行分类，检测恶意样本。报告生成模块基于虚拟机自省技术沙箱的日志和时序型神经网络模型分类的结果生成检测报告。系统模块如图9所示。

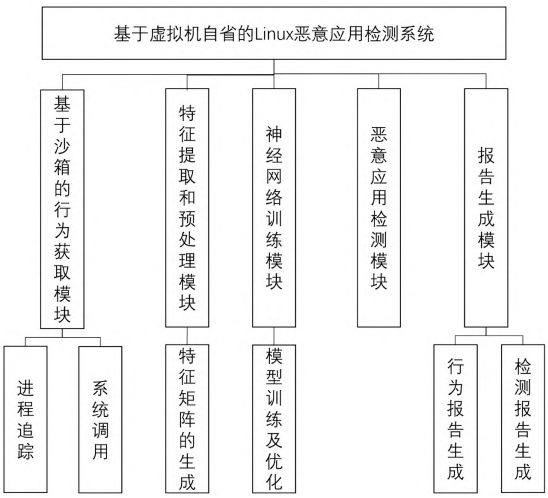

图 9 系统模块

2.2.2 系统功能实现

系统分层架构如图10所示，主要包括用户交互层、业务逻辑层、数据访问层、数据层和基础设施层。

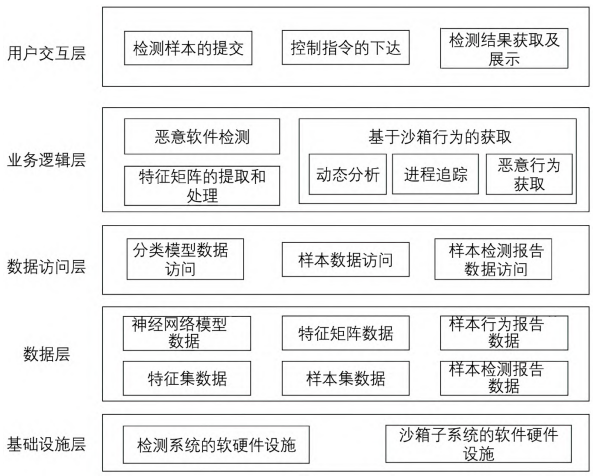

图 10 系统分层架构

1）用户交互层：用于用户和检测系统的交互，用户提交恶意样本，选择执行恶意软件检测流程，并获取和查看样本检测结果。

2）业务逻辑层：基于虚拟机自省的恶意软件检测系统的核心是业务逻辑，包括基于沙箱行为的获取功能、特征矩阵的提取和预处理功能以及恶意软件检测功能。其中，基于沙箱行为的获取功能是指进程追踪、系统调用的获取，以生成日志。特征矩阵的提取和预处理功能将生成的日志进行特征提取，转换为可以进行训练的特征矩阵。恶意软件检测功能是指使用训练好的神经网络进行未知样本的检测。

3）数据访问层：负责进行分类模型数据访问、样本数据访问和样本检测报告数据访问。其中，分类模型数据访问是指访问训练好的神经网络模型相关的数据，如神经网络参数。样本数据访问是指访问样本本身文件和相关特征矩阵。样本检测报告数据访问是指访问样本的行为日志和神经网络检测报告。

4）数据层：用于存储恶意软件在检测过程中生成的数据，包括神经网络模型数据、特征集数据、样本集数据、特征矩阵数据、样本行为报告数据和样本检测报告数据。

5）基础设施层：包括检测系统的软硬件设施和沙箱子系统的软硬件设施。

2.2.3 检测系统总体运行过程

系统检测流程如图11所示，主要包括样本输入、行为获取、特征处理、系统检测和结果输出5个阶段。

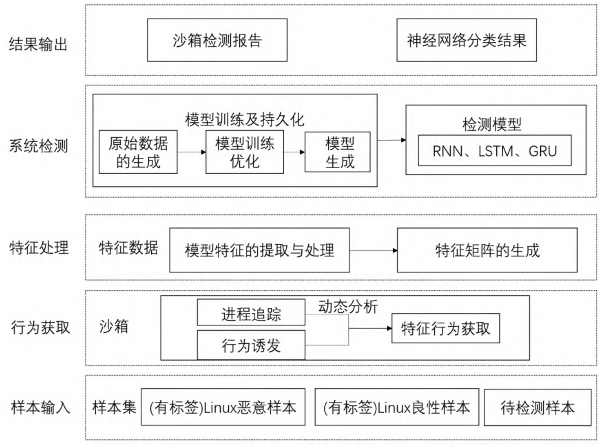

图 11 系统检测流程

1）样本输入：负责输入有标签的Linux恶意样本或者良性样本。

2）行为获取：通过进程追踪功能追踪沙箱中的恶意样本和良性样本，让样本在沙箱中尽可能多地表现恶意行为，形成样本的动态行为日志。

3）特征处理：对获取的动态行为日志进行特征提取，生成样本的特征矩阵，并将此特征矩阵作为后续神经网络模型所使用的数据集。

4）系统检测：该阶段负责训练用于分类的神经网络模型，并对输入的无标签样本进行恶意检测，包括模型训练及持久化、模型检测两部分。在模型训练及持久化过程中，根据特征处理阶段生成的数据集进行模型训练，寻找最优的参数以提高模型的准确率和召回率，生成神经网络模型。在模型检测过程中，使用持久化生成的神经网络模型根据恶意样本的特征矩阵对样本进行分类，检测出恶意软件。

5）结果输出：输出检测系统检测出的结果，包括沙箱提取的样本行为报告和神经网络模型分类后生成的分类报告。

3 方案测试及分析

3.1 实验环境

1）VMI Sandbox的实验环境

本文构建的VMI Sandbox包括3个端，分别为Host、Guest和Target，其中Guest和Target采用虚拟机嵌套的方式运行。Host使用主机系统的VMware创建，Guest和Target在Host系统中通过Qemu创建。具体的沙箱环境参数如表1所示。

表 1 沙箱环境参数

沙箱结构	参数名称	具体参数
Guest	操作系统	Linux Ubuntu18.04
	内存	4 GB
	CPU	4 核
	开源工具	HVMI
	开发语言	C、Python3.6.8
Target	操作系统	Linux Ubuntu18.04
	内存	4 GB
	CPU	2 核
Host	操作系统	Linux Ubuntu18.04
	内存	8 GB
	CPU	6 核
	软件版本	Qemu4.2.1、kvm

2）模型构建环境

模型构建环境设置为Linux操作系统，开发语言工具为Python，神经网络模型开发框架为Keras。模型构建环境参数如表2所示。

表 2 模型构建环境参数

实验环境	具体信息
操作系统	Ubuntu
CPU	Intel(R) Xeon(R) Platinum 8369 B CPU @ 2.90 GHz
GPU	A100(80 GB)
内存	128 GB
开发语言	Python 3.10.11
深度学习框架	Keras2.2.5

3.2 恶意行为检测实验

一般样本行为检测旨在测试沙箱对恶意样本一般行为的检测能力，本文选取商用沙箱中触发行为较多的典型样本与VMI沙箱日志进行对比测试，分析VMI沙箱的行为检测能力。传统沙箱难以检测ROOTKIT，主要原因在于传统沙箱和恶意软件处于相同的特权级别，即都在操作系统内核层面，一般通过hook系统调用或者利用系统提供的特殊机制进行检测。恶意软件可以直接与安全软件进行对抗，本文的VMI技术层次在客户机操作系统之下，可以检测并保护内核模块的修改，有效解决了恶意软件的对抗行为，同时能够直接定位到ROOTKIT恶意操作位置。ROOTKIT样本行为检测旨在测试沙箱对ROOTKIT的检测能力。

1）一般样本行为检测

为了进行一般样本行为检测，本文选取商用沙箱中触发行为较多的3个典型样本，样本的MD5分别是0E06FA98075F9B3F18B24F7172F67918、7beb45f0c5ee36d3747d9ab65eb1060e和fcbfb234b912c84e052a4a393c516c78，并与VMI Sandbox日志进行对比测试，分析VMI Sandbox的行为检测能力，检测结果分别如表3、表4和表5所示。

表 3 0E06FA98075F9B3F18B24F7172F67918 检测

恶意行为	系统指令污染	服务创建/自启动	敏感文件读取	文件创建	删除文件	文件权限更改	获取系统相关信息	Bash执行命令	休眠躲避
微步	—	√	√	√	√	—	√	—	—
阿里云	√	√	√	√	—	√	√	√	√
HVMI	√	√	√	√	√	√	√	√	√

表 4 7beb45f0c5ee36d3747d9ab65eb1060e 检测

恶意行为	服务创建 / 自启动	创建守护进程	获取系统相关信息	查看网络配置	修改进程命令行
微步	—	—	—	√	—
阿里云	√	√	√	—	√
HVMI	√	√	√	√	—

表 5 fcbfb234b912c84e052a4a393c516c78 检测

恶意行为	反调试	敏感文件读取	文件权限更改	获取系统相关信息	Bash 执行命令
微步	√	√	—	√	—
阿里云	—	√	√	—	√
HVMI	√	√	√	√	√

测试结果表明，本文构建的VMI Sandbox能够对恶意代码相关的多种恶意行为进行检测，对比微步沙箱，本文所提沙箱具有更强大的监控能力，能够针对隐藏行为进行检测。另外，本文所提沙箱与阿里云沙箱在一般恶意行为检测方面能力持平，可通过增加hook、细化检测力度以进一步提升检测能力。

2）ROOTKIT样本行为检测

ROOTKIT是一种特殊的恶意软件，主要功能包括实现各类隐藏、创建root权限后门、劫持或关闭安全程序、获得远程访问能力等，一般难以检测。

内核态ROOTKIT具有与操作系统相同的权限，在内核级别运行，通常作为设备驱动程序或可加载模块加载到目标设备中。内核态ROOTKIT的开发难度大，主要原因是源代码中的任何错误都会影响目标系统的稳定性，常用于APT定向攻击。常见的内核态ROOTKIT技术如表6所示。

表 6 常见的内核态 ROOTKIT 技术

内核态 ROOTKIT 技术	传统沙箱检测能力	VMI 沙箱检测能力
修改内核系统调用表，拦截系统调用	—	√
对内核函数进行 inline hook	—	√
更改 MSR 寄存器改变系统调用例程地址	—	√
更改 IDTR、GDTR 寄存器改变指向描述符的地址	—	√
更改 CR 寄存器改变页表翻译过程	—	√
更改内核页表内容	—	√
更改内核 idt 中断描述符表以改变中断、异常处理函数	—	√
基于 eBPF 机制更改内核关键函数执行流程	—	—

为了测试VMI Sandbox对ROOTKIT的检测能力，本文选取GitHub上开源部分中可正常编译的ROOTKIT进行测试，微步和阿里云沙箱的测试方式是将ko文件上传对应的检测接口，VMI Sandbox的测试方式是自动化加载ko文件来观察沙箱的结果输出，表7列举了VMI Sandbox的检测结果，可以看出VMI Sandbox能够较好地对恶意样本进行分析检测。

表 7 VMI Sandbox 与微步、阿里云沙箱的检测结果对比

ROOTKIT	微步	阿里云沙箱	VMI Sandbox
Kopycat	良性，无恶意行为	未检测到	良性，无恶意行为
Subversive	根据杀毒引擎判别为恶意，但无恶意行为	未检测到	良性，无恶意行为
Sutekh	良性，无恶意行为	未检测到	恶意，修改内核系统调用表
Zero	良性，无恶意行为	未检测到	良性，无恶意行为
Fops	良性，无恶意行为	未检测到	良性，无恶意行为
Xor128	良性，无恶意行为	未检测到	良性，无恶意行为
spy	根据杀毒引擎判别为恶意，但无恶意行为	未检测到	良性，无恶意行为
Diamorphine	根据杀毒引擎判别为恶意，但无恶意行为	未检测到	恶意，修改内核系统调用表
Puszek-Rootkit	良性，无恶意行为	未检测到	恶意，修改内核系统调用表
LilyOfTheValley	根据杀毒引擎判别为恶意，但无恶意行为	未检测到	良性，无恶意行为
Reptile	良性，无恶意行为	未检测到	恶意，修改了内核 inet_ioctl、load_elf_binary、vfs_read、next_tgid、vfs_statx、audit_alloc 符号对应的区域
module.c	根据杀毒引擎判别为恶意，但无恶意行为	未检测到	恶意，修改了内核 inet_ioctl、load_elf_binary、vfs_read、next_tgid、vfs_statx、audit_alloc 符号对应的区域
lkm2_sys_call_table	良性，无恶意行为	未检测到	良性，无恶意行为
lkm3_backdoor	良性，无恶意行为	未检测到	良性，无恶意行为
lkm4_hideFile	良性，无恶意行为	未检测到	恶意，修改了内核 ext4_dir_operations 符号对应的区域
lkm5_hide_process	良性，无恶意行为	未检测到	良性，无恶意行为
lkm6_hideMod	良性，无恶意行为	未检测到	恶意，修改了内核 modules_op、kernfs_dir_fops 符号对应的区域

在ROOTKIT检测方面，本文所提沙箱具有更强大的检测能力。根据测试结果，VMI Sandbox能够检测出GitHub上开源可正常编译的17个高star的ROOTKIT中的多个恶意行为，并且能够捕捉到细化行为，检出率为41%。而微步沙箱仅能根据杀毒引擎判断出5个为恶意样本，不能检出具体恶意行为，同时阿里云沙箱不支持此类检测。由此可见，VMI沙箱在ROOTKIT检

测方面能力较强。

3.3 模型检测实验及对比分析

3.3.1 LSTM 模型训练及优化

数据集中有9647个ELF文件，其中，7391个是恶意样本（来自VirusTotal和VirusShare），2256个是良性样本（来自Linux系统自带的ELF文件），将数据集按照4:1的比例分为训练集和测试集。

本文设置批量大小（batch_size）为128，训练轮数（epoch）为10。为了提高参数调优的效率，将初始参数设定为较小的值，评价指标如表8所示。

表 8 评价指标

评价指标	值
准确率	93.45%
精确率	88.74%
召回率	89.33%
*F*1 分数	0.8904

1）优化batch_size

batch_size是训练一次所使用的数据量大小，它对训练速度和时间有较大影响。较大的batch_size可能需要更多内存，因此会减慢训练速度。为了找到合适的batch_size取值，可以从较大值开始减半，并评估性能。

batch_size对准确率的影响如图12所示，初始的batch_size先设定为128，再逐步减半batch_size值，直到达到一个平衡点。经过实验表明，当batch_size为32时，准确率达到最大，在保证训练速度的同时满足内存需求。

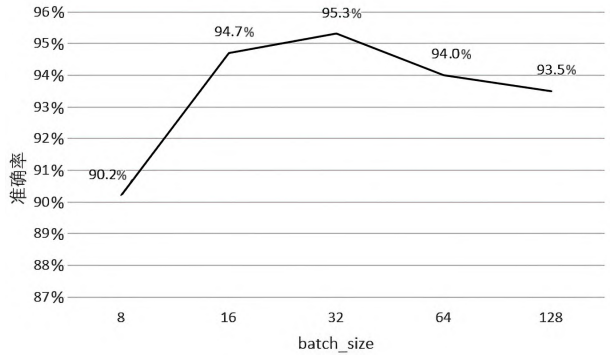

图 12 batch_size 对准确率的影响

2）优化epoch

epoch为训练轮数，初始状态epoch取值为10，此时的损失函数变化曲线如图13所示。通过观察图13可知，损失函数尚未收敛，说明模型仍在优化的过程中，因此应增大epoch值再次进行训练。

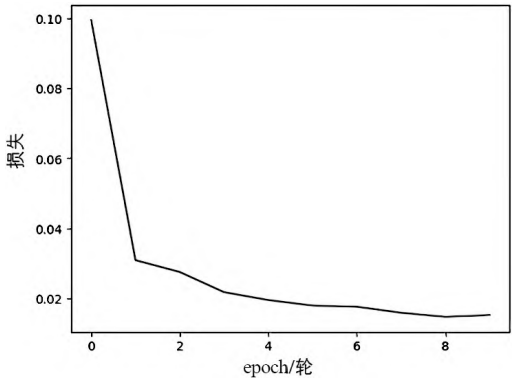

图 13 epoch 为 10 时的损失函数变化曲线

当epoch设定为50时，损失函数变化曲线如图14所示，模型随着训练轮数的增多逐渐收敛，当epoch为40时模型基本收敛，因此将epoch设定为40。

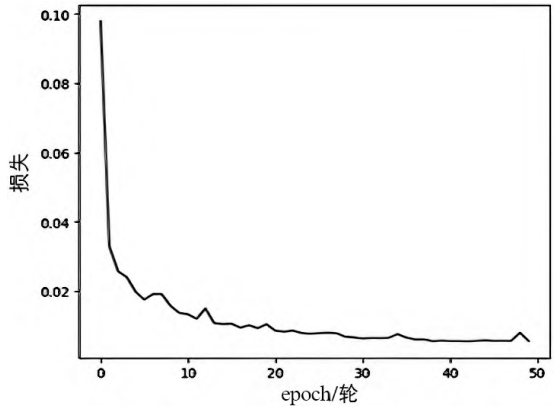

图 14 epoch 为 50 时的损失函数变化曲线

3）优化学习率

通常学习率的选取范围为[0.0001,0.1]，先选定[0.0001,0.001]、[0.001,0.01]和[0.01,0.1]3个区间。本文在batch_size为32、epoch为40的情况下，通过不断改变学习率进行训练，不同学习率的对比情况如表9所示。

表 9 不同学习率的对比情况

学习率	准确率	精确率	召回率	*F*1 分数
0.01%	96.23%	92.05%	95.21%	0.9360
0.1%	97.62%	94.04%	97.93%	0.9595
1%	98.02%	96.03%	97.32%	0.9667
10%	96.83%	92.72%	96.55%	0.9459

由表9可知，当学习率为1%时，模型的*F*1分数

达到最高，因此在batch_size为32、epoch为40、学习率为1%时取得模型最终结果，如表10所示。

表10 模型最终指标

评价指标	值
准确率	98.02%
精确率	96.03%
召回率	97.32%
*F*1分数	0.9667

3.3.2 不同检测模型对比与分析

本文方案的模型准确率为98.02%，*F*1分数为0.9667。为了选择最佳的时序处理模型，将本文方案与RNN模型、GRU模型和LSTM模型进行性能对比。实验结果如图15所示，LSTM模型在恶意软件检测方面的效果优于RNN模型和GRU网络模型。

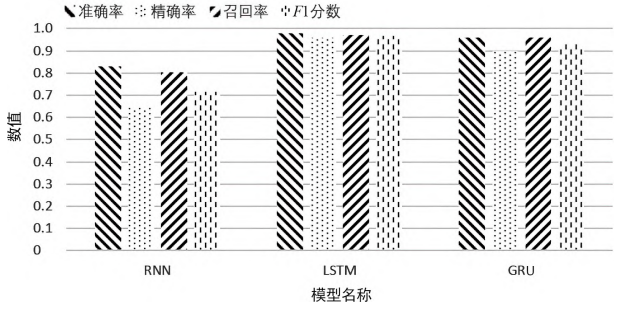

图15 不同时序处理模型的对比情况

4 结束语

本文对Linux恶意软件检测方案进行了深入研究，并针对恶意软件特征之间时序关系关注度较低和Linux恶意软件检测工具少等问题，提出一种基于虚拟机自省的Linux恶意软件检测方案。本文方案构建了一款用于分析Linux ELF文件的沙箱VMI Sandbox，通过对相关内核函数和系统调用函数进行hook操作，并对这些代码进行隐藏。

参考文献：

[1] CNCERT/CC. 2020 China Internet Network Security Report[EB/OL]. (2021-07-21)[2023-08-01]. http://www.cac.gov.cn/2021-07/21/c_1628454189500041.htm.
国家计算机网络应急技术处理协调中心. 2020年中国互联网网络安全报告[EB/OL]. (2021-07-21)[2023-08-01]. http://www.cac.gov.cn/2021-07/21/c_1628454189500041.htm.
[2] CNCERT/CC. Overview of China's Internet Network Security Situation in 2020[EB/OL]. (2021-05-26)[2023-08-01]. http://www.cac.gov.cn/2021-05/26/c_1623610314656045.htm.
国家计算机网络应急技术处理协调中心. 2020年我国互联网网络安全态势综述[EB/OL]. (2021-05-26)[2023-08-01]. http://www.cac.gov.cn/2021-05/26/c_1623610314656045.htm.
[3] FANG Zhan, LIU Jun, HUANG Ribian, et al. Research on Multi-Model Android Malicious Application Detection Based on Feature Fusion[C]//IEEE. 2021 4th International Conference on Robotics, Control and Automation Engineering (RCAE). New York: IEEE, 2021: 318-325.
[4] QIU Hongyuan, FERNANDO C. COLON O. Static Malware Detection with Segmented Sandboxing[EB/OL]. (2013-10-22)[2023-08-01]. https://ieeexplore.ieee.org/document/6703695.
[5] ALKHATEEB E M S. Dynamic Malware Detection Using API Similarity[EB/OL]. (2017-09-14)[2023-08-01]. https://ieeexplore.ieee.org/document/803 1489.
[6] KEDZIORA M, GAWIN P, SZCZEPANIK M, et al. Malware Detection Using Machine Learning Algorithms and Reverse Engineering of Android Java Code[EB/OL]. (2019-01-29)[2023-08-01]. https://www.researchgate.net/publication/331245836_Malware_Detection_Using_Machine_Learning_Algorithms_and_Reverse_Engineering_of_Android_Java_Code.
[7] BAUMAN E, AYOADE G, LIN Zhiqiang. A Survey on Hypervisor-Based Monitoring: Approaches, Applications, and Evolutions[J]. ACM Computing Surveys (CSUR), 2015, 48(1): 1-33.
[8] LI Baohui, XU Kefu, ZHANG Peng, et al. Research and Application Progress of Virtual Machine Introspection Technology[J]. Journal of Software, 2016, 27(6): 1384-1401.
[9] ROSENBLUM M, GARFINKEL T. Virtual Machine Monitors: Current Technology and Future Trends[J]. Computer, 2005, 38(5): 39-47.
[10] HOCHREITER S, SCHMIDHUBER J. Long Short-Term Memory[J]. Neural Computation, 1997, 9(8): 1735-1780.
[11] TAN Liuyan, RUAN Shuhua, YANG Min, et al. Educational Data Classification Based on Deep Learning[J]. Netinfo Security, 2023, 23(3): 96-102.
[12] DEY R, SALEM F M. Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks[EB/OL]. (2017-10-02)[2023-08-01]. https://ieeexplore.ieee.org/document/8053243.
[13] KUMAR R, CHARU S. An Importance of Using Virtualization Technology in Cloud Computing[J]. Global Journal of Computers & Technology, 2015, 1(2): 2623-2634.
[14] ZHANG Jixin, ZHANG Kehuan, QIN Zheng, et al. Sensitive System Calls Based Packed Malware Variants Detection Using Principal Component Initialized Multilayers Neural Networks[J]. Cybersecurity, 2018(10): 21-34.
[15] RABADI D, TEO S G. Advanced Windows Methods on Malware Detection and Classification[C]//ACM. The 36th Annual Computer Security Applications Conference (ACSAC'20). New York: ACM, 2020: 54-68.