SEMBeF：一种基于分片循环神经网络的敏感高效的恶意代码行为检测框架

詹 静[1,2,3], 范 雪[1], 刘一帆[1], 张 茜[1]

[1] 北京工业大学, 计算机学院, 北京 中国 100124
[2] 可信计算北京市重点实验室, 北京 中国 100124
[3] 信息安全等级保护关键技术国家工程实验室, 北京 中国 100124

摘要 词向量和循环神经网络(Recurrent Neural Network, RNN)能够识别语义和时序信息, 在自然语言识别方面中取得了巨大成功。同时, 代码运行时产生的 API 调用序列也反映了代码的真实意图, 因此我们将之应用于恶意代码识别中, 期望在取得较高正确率的同时减少人工提取和分析代码特征工作。然而仍然存在三个问题: 1)不少恶意代码故意通过随机混合调用敏感 API 和非敏感 API 破坏正常的上下文, 对这两种 API 同等对待可能产生漏报; 2)为尽可能全面收集代码行为, 代码运行期间产生的 API 序列长度较长, 这将导致 RNN 学习时间过长; 3)经典 RNN 常用的 softmax 分类函数泛化能力不强, 准确率有待提高。为了解决上述问题, 本文提出了一种基于分片 RNN(Sliced Recurrent Neural Network, SRNN)的敏感高效的恶意代码行为检测架构 SEMBeF。在 SEMBeF 中, 我们提出了一种安全敏感 API 权重增强的敏感词向量算法, 使得代码表示结果既包含上下文信息又包含安全敏感权重信息; 我们还提出了一种 SGRU-SVM 网络结构, 通过并行计算大幅降低了因代码 API 调用序列过长引起的训练时间过长的问题, 提高了检测正确率; 最后针对样本平衡和网络模型超参数选择问题进行了优化, 进一步提高了检测正确率。本文还实现了 SEMBeF 验证系统, 实验表明, 与其他基于经典词向量和 RNN 的深度学习方法以及常用的机器学习方法相比, SEMBeF 不仅检测正确率最高, 训练效率也得到了显著提升。其中, 检测正确率和训练时间分别为 99.40%和 210 分钟, 与传统 RNN 相比, 正确率提高了 0.48%, 训练时间下降了 96.6%。

关键词 恶意代码行为检测; API 序列; 敏感词向量模型; 分片循环神经网络(Sliced Recurrent Neural Network, SRNN)
中图法分类号 TP309; TP18 **DOI 号** 10.19363/J.cnki.cn10-1380/tn.2019.11.06

SEMBeF: A Sensitive and Efficient Malware Behavior Detection Framework based on Sliced Recurrent Neural Network

ZHAN Jing[1,2,3], FAN Xue[1], LIU Yifan[1], ZHANG Qian[1]

[1] School of Computer, Beijing University of Technology, Beijing 100124, China
[2]Beijing Key Laboratory of Trusted Computing, Beijing 100124, China
[3]National Engineering Laboratory of Key Technologies of Information Security Grade Protection, Beijing 100124, China

Abstract With word vector space model, Recurrent Neural Network (RNN) can identify semantic and temporal information, and has achieved great success in natural language recognition. Similarly, the sequence of API calls generated by the code runtime also reflects the real intention of the code. Therefore, we apply it to malicious code detection, expecting to achieve high accuracy while reducing the manual work of extraction and analysis of code features. However, there are still three problems: 1) many malicious codes intentionally destroy the normal context by randomly mixing sensitive APIs and non-sensitive APIs; 2) in order to collect code behavior as comprehensively as possible, the length of API sequence generated while code is running could be very long, which will lead to the long learning time; 3) softmax classification is commonly used with classical RNN, and there's still space for accuracy improvement. To solve the above problems, a Sensitive and Efficient Malware Behavior detection Framework (SEMBeF) based on Sliced Recurrent Neural Network (SRNN) is proposed in this paper. In SEMBeF, we propose a sensitive word vector space algorithm to enhance the weights of security-sensitive API, which makes the results of code representation contain both context information and security-sensitive weight information. We also propose a SGRU-SVM network structure, which greatly reduces the problem of long training

通讯作者: 詹静, 女, 讲师, 主要研究方向为网络安全, 可信计算, Email: zhanjing@bjut.edu.cn。
本论文工作得到国家重点研发计划项目(No.2016YFB0800204); 国防科研试验信息安全实验室对外开放项目(No. 2016XXAQ08); 国家高技术研究发展计划(No. 2015AA016002)资助。
收稿日期: 2017-12-13; 修改日期: 2018-03-07; 定稿日期: 2019-11-04

time caused by long API sequence of code and improves the detection accuracy. Finally, SGRU-SVM optimization is proposed to solve the problem of sample balance and hyper-parameter selection, which further improves the detection accuracy. This paper also implements the SEMBeF PoC (Proof of Cocept) system. Experiments show that compared with other deep learning methods based on classical word vector space model, machine learning methods and other common deep neural network models, SEMBeF system not only has the highest detection accuracy, but also improves the training efficiency significantly. The detection accuracy and training time of SEMBeF are 99.40% and 210 minutes, respectively. Compared with traditional GRU model, the accuracy increased by 0.48%, and the training time is decreased by 96.6%.

Key words malware behavior detection; API sequence; sensitive word vector space model; sliced recurrent neural network (SRNN)

1 前言

恶意代码变种的快速涌现给恶意代码检测带来了巨大压力。通常使用静态分析或动态分析的方式进行恶意代码检测。一方面, 静态分析不需要执行代码, 通过对样本二进制代码中的特征码检测[1]实现恶意代码检测, 具有高速、高效、低误报率的优点。该方法严重依赖已知恶意代码特征数据库, 而恶意代码特征数据库需要大量人工经验进行样本分析和规则提取, 否则容易出现误报漏报, 因此难以及时检测出新型恶意代码及不同变种; 另一方面, 动态分析则通过监控得到代码运行时的真实行为并进行分析, 由于相似的攻击意图通常对应一些频繁出现的操作序列组合, 因此动态分析能够较好地反映代码的真实意图, 从而识别未知恶意代码。

深度学习采用包括输入层、隐藏层和输出层的深度神经网络自动学习样本数据中的浅层和深层特征, 从而大量减少人力成本[2], 在图像识别、语音识别等领域得到了广泛应用。其他深度神经网络不同, Recurrent Neural Network (RNN)的输出是由当前的输入和当前输入之前的序列信息共同决定, 这表示 RNN 可以利用时序依赖关系分析任意长输入序列的特征[3], 而基于动态分析得到的代码 API 序列内正好存在时序依赖关系, 所以基于 RNN 对代表代码意图的 API 序列进行分析十分合适。结合现有的词向量模型和经典 RNN 能够基于上下文信息自动对样本行为进行表示, 学习和恶意性判断, 不再需要人工提取和分析代码的恶意特征。然而仍然存在三个问题: 1)不少恶意代码故意通过随机混合调用安全敏感 API 和非安全敏感 API)破坏正常的上下文, 对这两种 API 同等对待可能产生漏报; 2)为尽可能全面收集代码行为, 代码运行期间产生的 API 序列长度较长, 这将导致 RNN 学习时间过长; 3)经典 RNN 常用的 softmax 分类函数泛化能力不强, 有待提高。

为了解决上述问题, 本文提出了一种基于 Sliced Recurrent Neural Network (SRNN)的敏感高效的恶意代码行为检测架构 SEMBeF。其中包括: 一种安全敏感 API 权重增强的敏感词向量算法, 使得代码表示结果既包含上下文信息又包含安全敏感权重信息; 一种 SGRU-SVM 网络结构, 通过并行学习样本的长行为序列大幅降低学习时间, 并用 SVM 代替原有的 softmax 分类方法, 从而进一步提高了正确率。本文实现了 SEMBeF 的概念验证(PoC)系统, 实验表明, 与其他基于经典词向量和 RNN 的深度学习方法, 以及常用的机器学习方法相比, SEMBeF 不仅检测正确率最高, 训练效率也得到了显著提升。其中, 检测正确率和训练时间分别为 99.40%和 210 分钟, 与传统 RNN 相比, 正确率提高了 0.48%, 训练时间下降了 96.6%。

2 相关工作

传统的恶意代码检测技术通常采用静态分析或动态分析方法进行检测。静态分析需要大量人工经验生成检测所需的恶意代码特征数据库, 针对恶意代码静态分析方法过于依赖专家经验, 以及特征维度较高导致的高复杂度问题, 乔延臣[4]等人提出一种基于汇编指令与卷积神经网络的恶意代码分类方法, 通过逆行恶意代码可执行文件获取汇编代码, 并对其中的汇编指令使用 Word2vec 算法获取汇编指令词向量, 使用卷积神经网络训练分类模型, 实验证明该方法平均正确率达到 98.56%。但静态方法通常分析不够全面, 存在漏报现象, 并且绝大多数静态分析技术只能够识别已知的病毒和恶意代码, 也就是特征库内的病毒和恶意代码, 无法及时发现不在特征库内的未知恶意代码。基于动态分析的恶意代码检测方法则能够直接对代码行为进行分析, 识别其真实意图, 因此在识别未知恶意代码方面具有优势。因此, 本文主要关注基于动态分析的恶意代码检测方法进展。

CWSandbox[5]是一种轻量级恶意代码分析系统, 能记录和重定向沙箱内代码对系统文件, 注册表等的恶意修改行为, 在检测的同时不影响沙箱的系统环境。Cuckoo Sandbox[6]通过 API Hook 提取虚拟机内的代码行为, 能够快速恢复进行恶意代码检测的

虚拟机环境。Tian 等[7]提出通过 API 调用数量和特定 API 调用频率模式检测恶意代码的方法, 但没有考虑恶意代码可能插入大量冗余信息逃避检测。乔延臣等人[8]通过分析恶意代码的 API 调用习惯, 实现恶意代码同源性的自动确定, 具有较高的聚类准确率。上述方法都通过一些事先建立的具体规则对样本行为进行判断, 也存在可能无法检测未知样本的问题。

数据挖掘可以自动查找数据的模式特点, 使用所发现的模式预测未来数据, 或在不确定条件下做出决策, 因此成为了恶意代码识别的热点研究方向。Schultz 等[9]以 API 函数调用信息、PE 文件头的字符串序列等信息作为特征, 利用规则学习算法 PIPPER、朴素贝叶斯和多重朴素贝叶斯算法进行恶意代码分类检测, 正确率达到 97.11%。该方法的关键在于特征提取与特征选择, 以及分类方法选择。Abou-Assaleh 等[10]基于字节序列的 N-gram 特征采用 KNN 算法对 65 个可执行文件分类, 正确率为 98%。LAI[11]提取了可执行文件中可输出字符串, 用四种不同方法检测, 发现 SVM 算法正确率最高; Ding[12]计算了所有可执行文件中的每个 API 分布, 通过信息增益选择对这些特征筛选和分类器。还出现了大量基于决策树和随机森林, 支持向量机, 朴素贝叶斯, 或者多种分类器同时进行恶意代码检测的工作[13-14]。然而, 该方法在特征提取和特征选择阶段需要大量的人工干预和专家经验。

相对于传统的机器学习算法, 深度学习能够自动化识别特征, 在拥有较大规模样本的前提下, 能够提高检测效率、降低误报率, 目前成为了恶意代码检测的热点研究方向之一。常用的深度学习网络模型包括深度置信网络(Deep Belief Network, DBN), 卷积神经网络(Convolutional Neural Network, CNN), 循环神经网络(Recurrent Neural Networks, RNN) [15]等。

在应用深度神经网络进行恶意代码检测方面, CNN 通常用样本 Opcode 转换得到的灰度图作为输入特征, 但 CNN 结构并不适合处理样本内部操作的时序关系; RNN 则通常使用样本的 API 操作序列转换得到的词向量(如 Word2Vec[16])作为输入特征, 因此后者尽可能保留了代码行为时序特征。Kolosnjaji 等[17]构建卷积层和循环层的神经网络用于检测动态分析得到的恶意代码系统调用序列, 在包含 4753 个恶意样本的数据集上, 检测准确率为 89.3%。Pascanu 等[18]首次将 RNN 应用于恶意代码分类中, 提出一种基于动态分析的 2 层架构恶意代码检测系统, 第一层是 RNN, 用于学习 API 事件的特征表示; 第二层是逻辑回归分类器, 对学习到的特征进行分类, 但误报率(False Positive Rate, FPR)较高。Athiwaratkun 等人[19]在 Pascanu 的基础上, 利用 LSTM 进行恶意代码检测。Itaek 等[20]使用系统调用 API 作为循环神经 LSTM 网络的输入, 对恶意代码进行分类, 但对不同恶意代码种族的检测正确率差异较大。例如对 Lmir 家族的正确率较低小于 50%, 作者认为原因可能是因为该家族特征比较少, 以及来源于 VirusTotal 的分类标准可能不准确。Ye 等[21]通过 PE 文件解析器提取可执行文件调用的 API 序列, 并提出一个由多层限制玻尔兹曼机堆叠的 AutoEncoder 和一个关联存储器层组成的异构深度学习框架来检测未知恶意代码, 准确率达到 98.20%。方勇等人[22]设计了一种基于 LSTM 和随机森林的混合框架算法, 克服了传统识别模型在特征提取和识别效率上的问题, 能更好的应对海量钓鱼网站的实时检测, 实验结果表明, 混合框架算法的准确率达到 98.52%, 比单一的 LSTM 模型和随机森林算法分别提高了 11.30%和 2.90%。Yan 等[23]以 Opcode 和代码灰度图片为特征联合 LSTM 网络和 CNN 网络, 对恶意代码进行检测, 正确率达到 99.88%。Tobiyama 等[24]以检测到的代码操作为特征, 使用 RNN 对特征进行处理, 将其输出转化为图像作为 CNN 的输入进行训练, 得到的模型较为稳定。但上述基于组合深度神经网络的恶意代码检测方法涉及较多超参数, 优化和检测时间较长。

由于深度神经网络所需的训练和检测较长, 出现了一些优化模型研究, 其中, Yu 等[25]提出了一种切分循环神经网络(Slice Recurrent Neural Networks, SRNN), 将输入序列切分为几个等长子序列并行工作, 大幅减少了 RNN 的时间消耗。

3 恶意代码行为检测架构

由于代码运行时产生的 API 调用代表了其行为和真实意图, 我们针对 Windows 可执行代码建立了 SEMBeF 及恶意代码检测系统, 通过学习代码运行时产生的 Windows API 调用序列中的隐含知识, 检测未知代码是否具有恶意。虽然不同恶意代码的具体攻击意图和 API 调用实现方式不可能完全一致, 但我们认为从恶意代码的 API 调用序列从两个层次上还是相似的: 1)恶意代码一般都具有准备攻击、实际攻击和获取收益几个阶段, 从这个层次来说会出现相似的安全敏感 API 序列; 2)对于每个具体攻击意图, 恶意代码也会有较为相似的几类 API 调用序列, 而不同类别的 API 序列中的关键 API 从语义上说是相似的。为了准确获取代码 API 的语义和时序信息,

并且提高检测效率和准确率, 我们结合了改进的词向量和循环神经网络建立了 SEMBeF 及原型系统, 进行恶意代码检测。

SEMBeF 如图 1 所示, 包含两个阶段: 已知样本训练和未知样本检测。其中, 已知样本训练阶段包括 API 调用序列提取, 敏感词向量表示, 高效循环神经网络模型训练及优化, 最终得到词向量表及优化后的检测模型; 未知样本检测包括 API 调用序列提取和未知样本检测。

如图 1 所示, SEMBeF 架构包含四个模块: 样本处理模块, 高效循环神经网络模型训练模块, 模型优化模块和未知样本检测模块。

1) 样本处理模块。从恶意代码的执行到输入神经网络进行训练, 样本经过的处理依次为: (1)API 调用序列提取。本文使用 Cuckoo Sandbox 采集样本行为数据, 即 API 调用序列。将每个样本放入 VirtualBox 虚拟机中运行并通过 API hook 得到样本运行时调用的 API 序列, 包括代码进行的读写进程、文件和注册表, 进行网络通信等。为了尽可能得到完整的行为和时序信息设置每个样本的最大检测时间为 600s, 并且不对 API 序列进行去重处理。基于上述原因会导致 API 调用序列较长, 本文提取的 API 调用序列平均长度为 19683。对任意一个样本 x, 记为: $\text{Sample}_x=\text{File}_x\{\text{API1, API2, API3, } \cdots\}$。(2)敏感词向量表示。样本的 API 调用序列无法直接作为循环神经网络的输入, 需要事先进行编码, 该转换过程与自然语言文本使用词向量模型编码类似。但在传统词向量模型下, 语义相似度与中心词±窗口大小构成的上下文相关。由于恶意代码能够通过混合调用敏感 API 和非敏感 API 破坏正常的上下文, 因此本文提出了一种通过增加敏感 API 权重的改进词向量方法表示恶意代码行为。

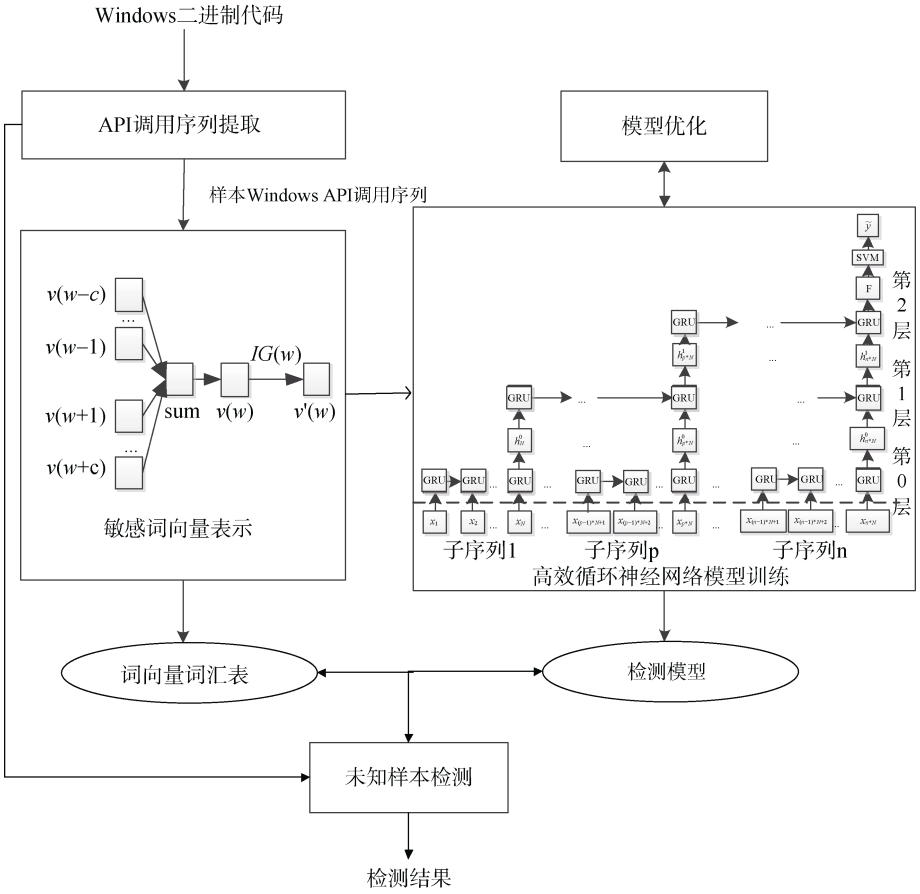


图 1 SEMBeF 架构

Figure 1 SEMBeF Framework

2) 高效循环神经网络模型训练模块。传统循环神经网络中的 RNN 循环单元具有长期依赖、长输入会造成训练时间过长以及输出分类函数泛化能力不够的问题, 因此本文提出了 SGRU-SVM 网络结构进行模型训练。

3) 模型优化模块。本文从两个方面进行优化: (1)对恶意代码和非恶意代码样本数量比例进行平衡处理, 一方面能减少占比较大的样本对训练的多余影

响, 另一方面如果将来需要添加更多具有代表性的样本, 无需再考虑恶意/非恶意样本数量比例的影响; (2)使用贝叶斯网络对高效循环神经网络模型进行超参数优化, 获取更好的模型。

4) 未知样本检测模块。在未知样本转换为 API 调用序列之后, 使用训练阶段生成的词向量词汇表和检测网络模型, 即可检测未知样本的结果。

4 恶意代码行为检测方法

4.1 基于敏感词向量的代码行为表示

自然语言中的单词组合成句子可以表达说话人的意图。类似的, Windows 代码通常通过调用 Windows API、访问特定资源达到代码的设计目的。恶意代码也是如此, 虽然不同恶意代码的攻击过程不尽相同, 但可以大致归纳为三个阶段: (1)攻击准备阶段。目的是隐藏自己。如, 先调用 Process32First, Process32Next 查找相关进程, 然后调用 OpenProcess 获取目标进程的句柄, 调用 VirtualAllocEx 获取写入其 DLL 路径的空间, 调用 WriteProcessMemory 在已分配的内存中写入路径, 最后调用 CreateRemoteThread 让恶意代码在另一个进程中运行; (2)实际攻击阶段。目的是收集攻击所需信息。如, 获取用户的键盘活动会调用 GetForegroundWindow 函数对打开的窗口进行记录, 然后调用 GetAsyncKeyState 函数来记录按键是否按下; (3)获取收益阶段。如, 通过网络通信回传时, 首先调用 WSACleanup 确保建立网络通信所必须的环境, 然后调用 socket 函数实现网络通信, connect 连接到服务器并调用 send, recv 函数来发送和接受网络数据。

但样本运行时产生的 API 调用序列无法直接作为循环神经网络的输入, 需要先进行编码表示, 将 API 数值化, 作为循环神经网络的输入向量。词向量将文本映射到向量空间, 并用向量来表示。目前常用的词向量模型 Word2Vec 是目前计算分布式词向量中效果最好的一种框架, 已经证明了具有相似上下文的词拥有相似的语义, 并且拥有相似语义的词向量之间的空间距离更短。然而, 词向量模型下的语义相似度依赖于中心词±窗口大小构成的上下文, 虽然一些敏感 API 组合在恶意代码中很常见, 但如果恶意代码混合调用敏感 API 和非敏感 API, 会导致敏感 API 组合不在窗口定义的上下文中, 一定程度上破坏了其语义。

综上所述, 恶意代码中的敏感 API 组合可以反映其真正意图, 敏感 API 的出现很可能表明该样本有恶意行为, 因此从检测的角度来说比其他 API 更为重要。但恶意代码可能通过破坏上下文的方式掩盖敏感 API 的语义信息, 如果能在尽可能保留 API 语义关系的同时, 适当增加敏感 API 的权重, 就能更好的识别恶意代码。

本文提出了一种融合敏感 API 权重信息和上下文语义信息的敏感词向量(Sensitive Word2Vec, S-Word2Vec), 用其表示恶意代码行为。对于一个 API 来说, 在检测过程中运用该 API 与不运用该 API 会有一定的变化, 这种前后信息量的变化的差值是 API 给检测带来的信息量, 即该 API 对于恶意代码检测越重要, 其得到的信息增益值也越大, 表示当前 API 对恶意代码检测越敏感。因此, 本文结合 Word2Vec 和信息增益(Information Gain, IG)。提出了敏感词向量 S-Word2Vec, 通过对 Word2Vec 词向量进行信息增益加权, 获得同时有语义信息和敏感信息的敏感词向量。S-Word2Vec 结构如图 2 所示。

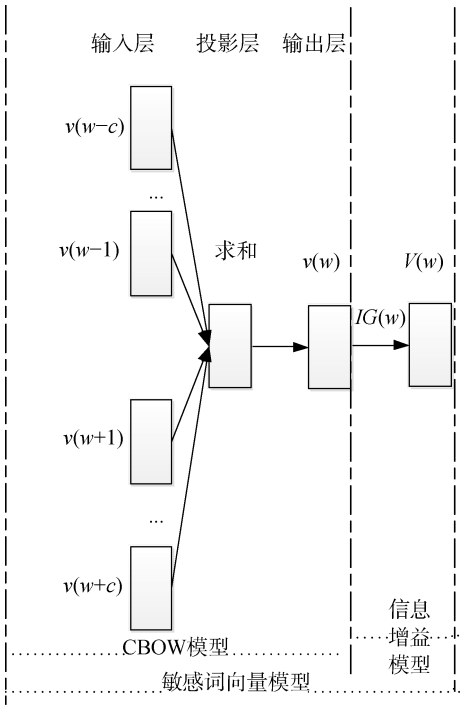


图 2 敏感词向量

Figure 2 Sensitive Word2Vec

S-Word2Vec 将 Word2Vec CBOW 词向量与 API 信息增益值进行求积运算, 从而得到最终的敏感词向量词汇表。包括两个阶段: 1)计算所有 API 的信息增益; 2)采用 CBOW 模型生成 Word2Vec 词向量, 加权获得所有 API 的 S-Word2Vec 值。

1) 计算所有 API 的信息增益。

假设样本集合中共有 n 个恶意样本, 形成恶意样本集合(BlackSamples), 有 m 个正常样本, 形成正常样本集合(WhiteSamples)。设 C 为样本集合的 API

词汇表, w 为词汇表 C 中任意 API。对于已知样本库, 假设含有 w 且为恶意样本的样本个数为 p, 含有 w 且为正常样本的样本个数为 q。则 API w 在恶意样本和正常样本集合中出现和不出现的次数如下表 1 所示。

表 1 API W 在恶意样本和正常样本集合中出现和不出现的次数

Table 1 Number of times API W appears and does not appear in malicious and normal code

	BlackSamples	WhiteSamples
w 出现	p_w	q_w
w 没有出现	$n-p_w$	$m-q_w$

设 $IG(w)$表示 w 所对应的信息增益值, IG_File 表示包含词汇表中所有 API 的信息增益值 $IG(w)$的文件, 计算信息增益的算法如表 2 所示。

表 2 信息增益算法

Table 2 Information Gain Algorithms

输入: BlackSamples, n, WhiteSamples, m, C
输出: IG_File

$H(X)=-\frac{m}{m+n}\log(\frac{m}{m+n})-\frac{n}{m+n}\log(\frac{n}{m+n})$
//计算信息熵
for each w in C :
p_w =CountBlackSamples(BlackSamples, w)
//统计词汇表 C 中 API w 出现在恶意样本中的次数
q_w =CountWhiteSamples (WhiteSamples, w)
//统计词汇表 C 中 API w 出现在正常样本中的次数
$H(X\mid w)=-\frac{p_w+q_w}{n+m}(\frac{p_w}{p_w+q_w}\log(\frac{p_w}{p_w+q_w})$
$+\frac{q_w}{p_w+q_w}\log(\frac{q_w}{p_w+q_w}))-\frac{(n-q_w)+(m-p_w)}{n+m}$
$(\frac{n-p_w}{(n-q_w)+(m-p_w)}\log(\frac{n-p_w}{(n-q_w)+(m-p_w)})+$
$\frac{m-q_w}{(n-q_w)+(m-p_w)}\log(\frac{m-q_w}{(n-q_w)+(m-p_w)}))$
//计算 API w 的条件熵
$IG(w)=H(X)-H(X\mid w)$
//计算 API w 的信息增益值
IG_File.write ($IG(w)$)
//将 API w 的信息增益值写入 IG_File

2) 计算所有 API 的 S-Word2Vec 值。

CBOW 用上下文即 $Context(w)=w-c,\cdots,w-1,\ w+1,\cdots,w+c$ 来预测中心词 w, 中心词 w 即 API。设 c 为上下文周围词的窗口大小, 条件概率 $p(w\mid Context(w))$ 即为当前中心词 w 在周围词窗口为 c 的上下文中出现的概率。CBOW 的优化目标为 $G=\prod_{w\in C}p(Context(w))$, 即求解词汇表 C 中所有 API 对应的条件概率的乘积最大值, 使得词汇表 C 中任意 API 的条件概率最大, CBOW 使用随机负采样和随机梯度上升的方法对 G 进行优化。

设 AllSamples 为所有样本集合, neg 为负采样个数, IG_File 为保存每个 API 信息增益值的文件, $v(w)$ 表示 CBOW 词向量, $V(w)$ 表示敏感词向量, S-Word2Vec 表示包含所有 API 敏感词向量的 S-Word2Vec 词向量词汇表。S-Word2Vec 词向量算法如表 3 所示。

表 3 S-Word2Vec 词向量算法

Table 3 S-Word2Vec Word Vector Algorithm

输入: AllSamples, C , neg , IG_File
输出: S-Word2Vec_File

for each sample in AllSamples:
for each w IN sample:
contextFile.write=(ReadVocabFromAllSamples(w))
//从每个 API 序列中按顺序读入每一个 API, 并将 API 和周围词 $(w,Context(w))$ 存入文件 contextFile
for each w in C :
$v(w)$ =InitWordVector(w)
//随机初始化 CBOW 模型词向量 $v(w)$
$NEG(w)$ =InitUnigramTable(w ,neg)
//随机负采样每个 API 的负样本集
for each line in contextFile:
SGD(w , $Context(w)$)
//从 contextFile 文件读取 $(w,Context(w))$, 并利用随机梯度上升法对词向量 $v(w)$ 进行更新
cbowDic.write($v(w)$)
//将更新之后的词向量存入字典 cbowDic
for each line in IG_File:
igDic=line.split()
//读取信息增益值, 并保存在字典 igDic
for each w in cbowDic:
$IG(w)$ =igDic(w)
$V(w)=v(w)$ * $IG(w)$
S-Word2Vec_File.write($V(w)$)
//将 Word2Vec 算法得到的词向量乘以相应的信息增益权值 $IG(w)$, 得到包含恶意检测敏感性信息和语义信息的敏感词向量, 并保存在文件 S-Word2Vec_File 中

通过上述 S-Word2Vec 模型, 本文得到了样本库中所有 API(265 个)对应 S-Word2Vec 词向量, 根据词向量词汇表即将代表所有样本行为的 API 序列转换为词向量序列, 供后续模型训练使用。

4.2 基于 SGRU-SVM 的高效准确恶意代码检测训练

本文结合切分循环神经网络 SRNN 和支持向量机(Support Vector Machine, SVM)提出了 SGRU-SVM

模型对恶意代码进行更为高效和准确的检测, 分为两个部分, SGRU 结构提供更为高效的恶意代码检测架构, SVM 用于提高检测准确性, SGRU-SVM 网络结构如图 3 所示:

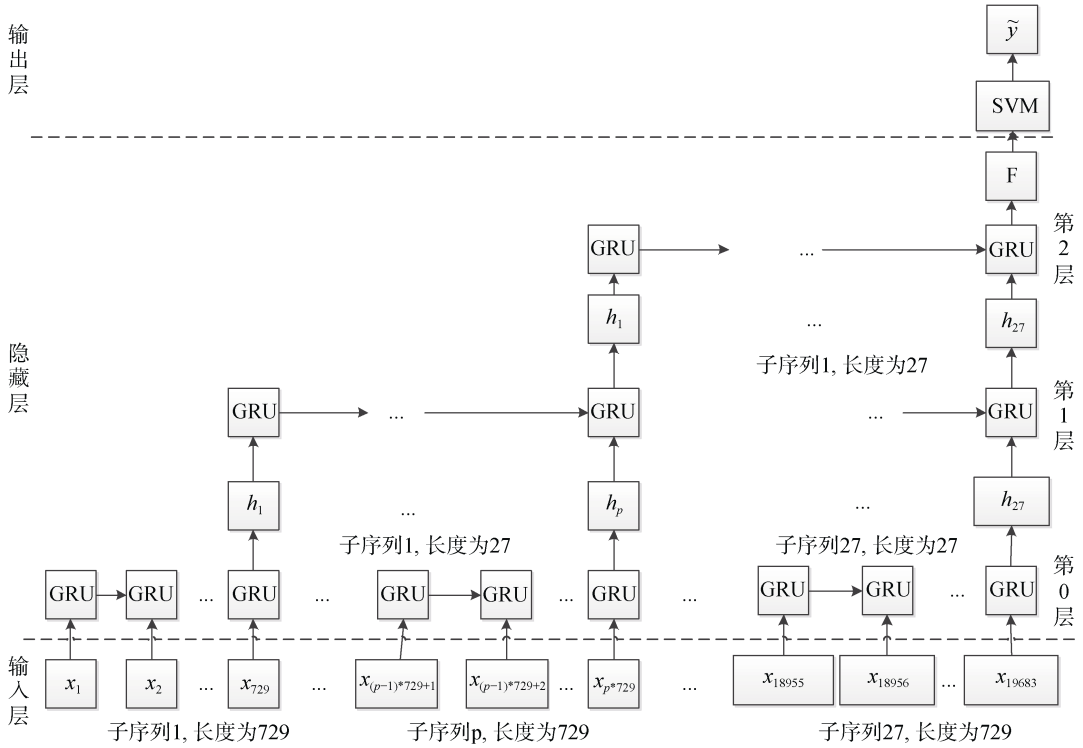


图 3 SGRU-SVM 网络结构

Figure 3 SGRU-SVM network structure

1) 基于 SGRU 模型的高效检测。

当前样本集合中代码运行期间产生的 API 序列平均长度多达 19000, 如果采用经典的 RNN 网络学习, 会出现长期依赖和训练时间过长两大问题。长短期记忆神经网络(Long Short-Term Memory Networks, LSTM)[26]通过增加忘记门、输入门、输出门和循环单元状态对 RNN 的循环单元进行了改进, 较好的解决了长期依赖问题。但是由于引入了 3 个门, 导致网络训练时间进一步增加。

为了解决应恶意代码 API 序列输入过长造成的检测训练时间过长的问题, 本文从具体循环单元和网络计算方式两个方面做了改进, 提出一种 SGRU 模型进行恶意代码检测。首先, 选择 GRU (GatedRecurrent Unit)循环单元代替 LSTM 循环单元。因为 GRU 将 LSTM 中的忘记门和输入门合成一个单一的更新门, 能够通过减少门数量减少训练时间; 第二, 选择 SRNN 并行计算结构代替经典 RNN 的串行计算结构。SRNN 将输入序列切分为几个相等长度的子序列, 循环单元在子序列内并行工作, 这样可以减少同层相邻循环单元之间的计算依赖时间。

本文基于 SRNN 结构建立了 SGRU 网络模型。假设输入序列为 $X=[x_1,x_2,\cdots,x_T]$, 其中 x 表示每一时刻的输入, T 表示序列的总长度, 通过对长度为 T 的输入序列 X 进行 k 次切割, 分为 n 个等长子序列, 则有: $T=n^{k+1}$。此时 GRU 单元在每个子序列内是并行工作的, 因此可以大幅减少计算等待时间。实际上, SGRU 只是一种特殊的 GRU 模型, 简要证明如下:

对于原始 GRU 模型, 每一时刻的输出都和当前时刻的输入和前一时刻的输出有关, 设 U 为输入层到隐藏层的权重, W 为隐藏层到隐藏层的权重, 即有: $h_t=f\left(U_{xt}+Wh_{t-1}+b\right)$, 其中 x 表示 t 时刻输入, h 表示隐层状态, 函数 f 表示激活函数。为了简化问题, 设激活函数为 $f(x)=x$, b=0, h_0=0, 递归运算后得到 GRU 最后的隐层状态如公式(1)所示。

$$\begin{aligned} h_T &= U_{x_T}+Wh_{T-1}=U_{x_T}+W(U_{X_{T-1}}+Wh_{T-2})=\ldots \\ &=U_{x_T}+WU_{x_{T-1}}+W^2U_{x_{T-2}}+\ldots+W^{T-1}U_{x_1} \end{aligned} \tag{1}$$

对于 SGRU 模型, 设 U 为输入层到隐藏层的权重, W 隐藏层到隐藏层的权重, SGRU 最终隐层状态 F 可表示为公式(2)[25]。

$$F = U_k U_{k-1}...U_0 x_T + U_k U_{k-1}...W_0 U_0 x_{T-1} + U_k U_{k-1}...W_0^2 U_0 x_{T-2} + ... + W_k^{n-1} U_k W_{k-1}^{n-1} U_{k-1}...W_0^{n-1} U_0 x_1 \quad (2)$$

设 I 为单位矩阵, 为简化问题, 设 $U_0=U$, $U_k=U_{k-1}=...=U_1=I$, $W_k = W^{n^k}$, 加上已知 $T=n^{k+1}$, 代入公式(2), 有:

$$F = IUx_T + IW^1 Ux_{T-1} + IW^2 Ux_{T-2} + ... + W^{(n^k+n^{k-1}+...n^0)} IUx_1 = Ux_T + WUx_{T-1} + W^2 Ux_{T-2} + ... + W^{T-1} Ux_1 = h_T$$

可知当激活函数 f 是线性函数, 且满足上述简化条件时, SGRU 的输出与 GRU 的输出相同, 即基于 SRNN 结构的 SGRU 是一种特殊的 GRU, 得证。

由于本文提取的 API 序列平均长度多达 19000, 我们选择切分 2 次, 得到 3 层隐藏层, 设置输入层每个子序列长度为 729, 即有: T=19683, k=2, n=27。本文的 SGRU 网络结构如图 3 所示, 隐藏层第 0 层子序列的数量为 27, 对 27 个子序列进行并行训练, 将得到的 27 个子序列训练结果作为隐藏层第 1 层网络的输入; 隐藏层第 1 层子序列数量为 27, 同样, 将得到的 27 个子序列训练结果作为隐藏层第 2 层网络的输入; 隐藏层第 2 层子序列数量为 1, 通过 3 层隐藏层得到最终隐层状态 F。在此计算过程中保留了子序列内部的时序关系, 以及子序列之间的时序信息; 隐藏层的输出即为整个 SGRU 网络的隐层状态。

在训练速度方面, 假设每个 GRU 处理数据的时间为 r, T 为输入序列长度, 则 GRU 网络训练所花费的时间为 $t_{\mathrm{GRU}} = T*r$, 而 SGRU 每一层训练所花费的时间都为 $n*r$, 因此 SGRU 的总训练时间为: $t_{\mathrm{SGRU}} = n*(k+1)$。加上已知 $T=n^{k+1}$, 则 SRNN 相比较于 RNN, 训练时间下降了 B, 如下所示公式(3)所示。

$$B = 1 - \frac{t_{\mathrm{SGRU}}}{t_{\mathrm{GRU}}} = 1 - \frac{n*(k+1)*r}{T*r} = 1 - \frac{n*(k+1)*r}{n^{k+1}*r} = \frac{n^k - k - 1}{n^k} \quad (3)$$

在本文 T=19683, k=2, n=27 的情况下, SGRU 与 GRU 相比, 理论训练时间下降了 1–1/234=99.6%, 具有显著提高。

2) 基于 SGRU-SVM 模型的准确检测。

在 SGRU 网络的分类器选择上, 我们采用泛化能力更强的支持向量机(Support Vector Machine, SVM)代替了 Softmax 函数进一步提高检测准确性。

SVM 和 Softmax 都能完成分类问题, 但是两者输出数值的含义和损失函数有很大不同。

首先, Softmax 函数的输出数值很容易受到正则化参数λ的影响。Softmax 函数的输出值计算方法为: $f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_j}}$, e^{z_j} 为属于该类的输出值, 并通过 e^x 函数将输出值控制在[0, 1]之间, e^{z_j} 表示属于其他类的输出值并将其控制在[0, 1]之间。Softmax 输出值大小代表属于该类别的概率。例如若网络模型的输出值为[1, –2, 0], 通过 Softmax 计算的输出值为[0.7, 0.04, 0.26], 其中第一类的概率最大。若增大 λ 的取值, 网络模型输出变为原来的一半, 即[0.5, –1, 0], 通过 Softmax 函数计算的输出值为[0.55, 0.12, 0.33], 使得概率的分散程度有了很大的变化。λ 越大, Softmax 输出值分散程度越小; 而 SVM 输出值顺序表示所属类别排序, 大小不代表任何意义, 没有这个问题。

第二, Softmax 使用交叉熵损失函数量化模型预测结果和实际结果之间的差距, 容易出现过拟合情况。

Softmax 使用 cross-entropy loss 损失函数, 计算方法为: $L_i = -\log\left(\frac{e^{f_{yi}}}{\sum_j e^j}\right)$。其中 L_i 表示损失值, $e^{f_{yi}}$ 表示属于该类的输出值并通过 e^x 函数将输出值控制在[0, 1]之间, e^j 表示属于其他类的输出值并将其控制在[0, 1]之间。Softmax 会为每个样本进行重新更新, 以便更加接近完美分类结果, 从而会导致过拟合的情况; 而 SVM 使用 hinge loss 损失函数, 计算方法为: $L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$。其中 L_i 表示损失值, $w_j^T x_i$ 表示属于该类所对应的输出值, $w_{y_i}^T x_i$ 表示其他类对应的输出值, Δ 一般情况下为 1。hinge loss 更加注重分类过程中正确分类和错误分类之间的距离 Δ, 只要大于设定的距离 Δ, 就忽略距离的大小, 忽略分类细节。

因此本文在上述 SGRU 结构的基础上增加了 SVM 分类, 得到 SGRU-SVM 网络模型。输入数据为样本 API 序列, 当样本 API 序列长度大于 19683, 对序列进行截断, 否则对序列进行填充。隐藏层共包含 3 层, 通过 3 层隐藏层得到最终的隐藏层状态 F, 输出层通过 SVM 对该样本进行检测, 得到检测结果 $\tilde{y}$。

本文采用 5 折交叉验证训练, 使用 Adam 算法对

参数进行更新, 当达到最大迭代次数时, 网络停止优化, 为了防止在训练过程中网络对训练集数据过拟合, 在网络层数之间添加随机控制神经元的闭合的 Dropout 层, 提高网络模型的泛化能力。设 epoch 为网络训练数据集次数, batch_size 为网络每次训练的样本数, learn_rate 为学习率, S-Word2Vec_File 为敏感词向量文件, AllSamples 为训练样本 API 序列以及标注, score 为 SGRU-SVM 网络在测试集的检测评价指标, SGRU-SVM 网络整体训练算法如表 4 所示。

表 4 SGRU-SVM 网络训练算法

Table 4 SGRU-SVM Network Training Algorithms

```
输入: epoch, batch_size, learn_rate, S-Word2Vec_File, AllSamples
输出: score
for sample in AllSamples:
for w IN sample:
X,Y=(ReadVocabFromAllSamples( w ))
//从每个 API 序列中按顺序读入每一个 API 以及标注, 将序列存储在列表 X 中, 将标注存储在列表 Y 中
for x in X:
x_padded_seqs=splitPadding(x)
//将存储在列表 X 中的 API 序列进行切分并保存在列表 x_padded_seqs
for line in S-Word2Vec_File:
embeddings=line.split()
//读取敏感词向量, 并保存在字典 embeddings
Elayer=Embedding(embeddings)
input=Input()
embed= Elayer (input)
gru_out=GRU(dropout=0.5)(embed)
preds=Dense(2)(gru_out)
model=Model(input,preds)
//构建网络模型 model
opt = Adam(lr= learn_rate)
model.compile(loss=[hinge_loss],optimizer=opt,metrics=['accuracy'])
//使用 Adam 算法对网络进行优化, 使用 hinge_loss 计算网络损失, 并使用正确率对网络进行评价
for indextr,indexte in range(5):
x=array(x_padded_seqs)[indextr]
x_test =array(x_padded_seqs)[indexte]
y=Y[indextr]
y_test= Y[indexte]
//将数据集分为训练集和测试集
model.fit(x,y,batch_size=batch_size,epochs=epoch)
//使用训练集进行网络训练
best_model= load_model(savebestmodel)
//保存正确率最高的模型
score=best_model.evaluate(x_test,y_test,batch_size)
//使用正确率最高的模型对测试集进行测试
//使用 5 折交叉验证对网络进行评价
print(score)
//返回网络模型在测试集上的评价指标
```

4.3 基于代价敏感及贝叶斯算法的模型优化

本文从样本分类平衡和超参数选择两个方面进行模型优化。

1) 基于代价敏感的样本分类平衡优化。

如果恶意代码和非恶意代码样本数量比例不均衡, 可能会使得训练结果偏向占比高的分类; 如果将来需要增加训练样本, 恶意/非恶意样本数量比例的变化可能反而导致模型准确性降低。本文采用代价敏感对样本进行学习, 设正常样本数量为 w, 恶意样本数量为 b, 正常样本分类错误代价 $C_w=b/(w+b)$; 错误样本分类错误 $C_b=w/(w+b)$。

本文恶意样本数量为 4129, 正常样本数量为 910, 计算得到的 C_w=0.82, C_b=0.18。基于 SEMBeF 架构对比输入样本分别进行代价敏感处理和不做代价敏感处理两组实验, 结果如表 5 所示。

由下表可知, 经过代价敏感处理的模型各项指标均达到了 99%以上, 比不采用代价敏感处理要高出 3%~4%, 可见采用样本代价敏感优化可以得到更准确的检测结果。

2) 使用贝叶斯网络进行超参数优化。

超参数是需要在网络训练之前提前设置的参数, 如网络训练数据集的次数 *epoch*, 网络每次训练的样本数 *batch_size*, 学习率 *learn_rate* 等。参数调节过程不仅耗时而且很难选择最优解。网格搜索和随机搜索是两种常用方法, 前者通过遍历所有参数取值组合来选择最优参数, 但搜索速度较慢; 后者对参数进行随机选择, 可能会错过最优参数组合。贝叶斯在优化过程中应用高斯模型, 在利用完整的历史信息的前提下进行后验概率分布的采集和更新, 具有迭代次数少、优化速度快、适用于求解优化目标函数表达式未知, 非凸, 多峰等复杂情况等优点。

因此, 本文采用贝叶斯优化方法对超参数进行优化, 基于正确率选择最佳模型。将 *epoch* 优化取值范围设置为[20, 25, 30], 将 *batch_size* 优化取值范围设置为[20, 30, 40], 将 *learn_rate* 优化取值范围设置为[0.0001, 0.001, 0.01], 设置 *max_evals*=15, 表示随机搜索的次数为 15 次。基于 SEMBeF 架构, 对不同超参数组合得到的恶意代码检测效果进行比较, 结果如表 6 所示:

根据正确率(最高为 98.01%), 本文选用了经过贝叶斯优化得到的第 4 组超参数, 分别为 epoch=25, batch_size=30, learn_rate=0.001。

5 实验与分析

5.1 基本设置

本文基于 SEMBeF 进行了系统实现, 软硬件环境配置如表 7 所示。

表 5 不采用/采用代价敏感优化方法检测指标对比

Table 5 Comparison of Detection Indicators with and without Cost-Sensitive Optimization Method

样本	acc	rec	pre	AUC	F1	训练时间(min)
无代价敏感样本	95.83%	95.59%	96.07%	94.57%	95.82%	216
代价敏感样本	99.40%	99.11%	99.60%	99.12%	99.29%	210

表 6 超参数设置结果对比

Table 6 Comparisons of Super-parameter Settings

组号	epoch	batch_size	learn_rate	acc
1	25	20	0.01	78.33%
2	30	30	0.0001	97.42%
3	30	40	0.001	97.68%
4	25	30	0.001	98.01%
5	30	40	0.0001	97.51%
6	30	40	0.01	85.88%
7	20	40	0.0001	97.42%
8	25	30	0.01	83.30%
9	20	30	0.001	97.42%
10	20	20	0.001	97.61%
11	25	20	0.0001	97.22%

表 7 SEMBeF 系统软硬件环境配置

Table 7 SEMBeF System Software and Hardware Environment Configuration

软硬件名称	型号
CPU	Intel® Xeon® CPU E5-2620 v3 @ 2.40GHz
GPU	NVIDIA-SMI 367.48
内存	128GB
操作系统	ubuntu 14.04.1 64bit
恶意代码分析操作系统	Windows XP

本文采用的恶意代码来源于 http://academictorrents.com/, 恶意代码的种类包括病毒, 蠕虫, 后门, 木马, 勒索病毒等, 全都经过了 VirusTotal 平台检测; 正常样本主要来自 Windows XP 操作系统自带的进程和相关应用软件, 以及来自 http://xiazai.zol. com.cn/网站的软件, 包括浏览器, 聊天软件, 工具类软件, 播放器等。为了躲避行为分析, 恶意代码可能调用冗余 API 隐藏意图。本文还尝试了为样本提取 5 分钟、10 分钟 API 进行检测两种情况, 发现两种情况下得到的恶意代码识别率都很高(>99%)且差异不大, 但样本产生的 API 序列长度增加了 39.2%, 模型检测时间也过长, 故在不影响检测效果的前提下, 设置代码行为提取时间为 5 分钟。但在真实环境中, 由于仍可能存在具有长延迟功能的恶意代码, 应进一步延长检测时间。

5.2 基本设置敏感词向量检测效果分析

本文提出的敏感词向量增加了对检测敏感信息, 本节通过对比 Word2Vec 词向量与 S-Word2Vec 词向量模型在 SGRU 网络检测中的得到的指标结果, 分析敏感词向量的效果。在 SGRU 模型中采用了五折交叉验证, *epoch* 设置为 25, *batch_size* 设置为 30, *learn_rate* 为 0.001, 得到的两组实验的五个指标: 正确率 acc, 召回率 rec, 精确率 pre, *ROC* 曲线面积 AUC, 精确率和召回率的调和均值 F1, 结果如图 4 所示:

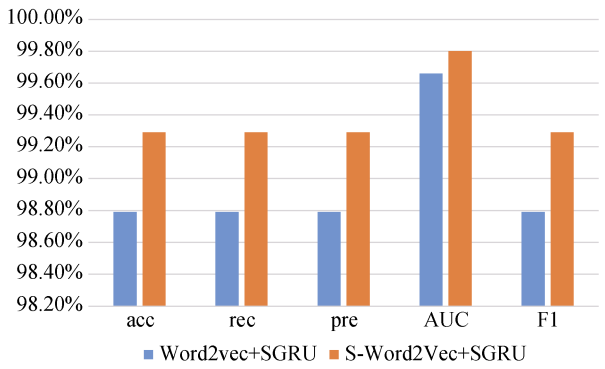


图 4 对比词向量实验结果

Figure 4 Comparisons of Word Vector Experiments

由图 4 可知, 对比 Word2Vec 词向量与 S-Word2Vec 词向量模型在 SGRU 网络检测中得到的指标结果, 使用 S-Word2Vec 方法得到的敏感词向量对于恶意代码检测正确率达到了 99.29%。其他评价指标均有提高, 其中, 精确率、召回率、F1 值均提高了约 0.5%, AUC 值提高了约 0.14%。这说明以 S-Word2Vec 词向量为输入能使网络模型更精确地检测恶意代码, 并且模型更稳定。因为与传统的 Word2Vec 方法相比, S-Word2Vec 方法得到的词向量不仅包含 API 语义相似度关系, 还包含了 API 对于恶意代码检测的重要性信息, 因此以改进词向量为输入能更好的检测恶意代码。

5.3 相关模型检测效果比较

本文实现的 SEMBeF 系统包括 S-Word2Vec 敏感词向量及 SGRU-SVM 模型(记为 S-Word2Vec+SGRU-SVM), 为了验证系统总体效果, 与传统 Word2Vec 词向量模型, GRU-softmax, SGRU-softmax 模型分别进行检测效果比较。比较的整体模型包括:

Word2Vec+GRU, S-Word2Vec+GRU, S-Word2Vec+SGRU 三类。超参数为 4.3 节选择结果, 即 *epoch*=25, *batch_size*=30, *learn_rate* =0.001, 得到的模型检验参数及时间开销如表 8 所示:

表 8 相关模型检测效果比较

Table 8 Comparisons of Relevant Model Detection Effectiveness

模型	acc	rec	pre	AUC	F1	训练时间(min)
Word2Vec+GRU	98.92%	98.92%	98.92%	99.50%	98.92%	6190
S-Word2Vec+GRU	99.27%	99.27%	99.27%	99.74%	99.27%	6240
S-Word2Vec+SGRU	99.29%	99.29%	99.28%	99.80%	99.29%	213
S-Word2Vec+SGRU-SVM (SEMBeF 系统)	99.40%	99.11%	99.60%	99.12%	99.29%	210

由上表可知, 基于传统的 Word2Vec+GRU 模型, 逐步采用敏感词向量模型 S-Word2Vec, SGRU, 以及 SVM, 其检测准确率分别为: 99.27%, 99.29%和 99.40%, 得到稳步提升, 其中, SEMBeF 系统取得了最优的 99.4%准确率; 系统训练时间方面, 与传统的 Word2Vec+GRU 模型相比, SEMBeF 检测时间大幅减少了 96.6%; 在其他模型评价指标方面, SEMBeF 系统的精确率、F1 值都为所有模型中的最优值, 召回率和 AUC 仅比 S-Word2Vec+SGRU 模型下降了 0.18%和 0.68%。

综上所述, 与其他循环神经网络相关模型相比, SEMBeF 系统不论是在模型检测效果、稳定性还是时间开销方面表现都接近最优。特别的, 我们提出的 SEMBeF 架构在准确性和稳定性较高的前提下, 比不使用 SGRU-SVM 的经典 GRU 的训练时间要减少 96.61%, 检测效率提升明显。

5.4 不同模型检测效果比较

本文还与机器学习分类算法和其他类型深度神经网络进行了恶意代码检测效果比较。比较算法和模型包括 K-近邻算法(KNN)、决策树(CF)、支持向量机(SVM)、朴素贝叶斯算法(NB)以及卷积神经网络 CNN, 实验结果如图 5 所示:

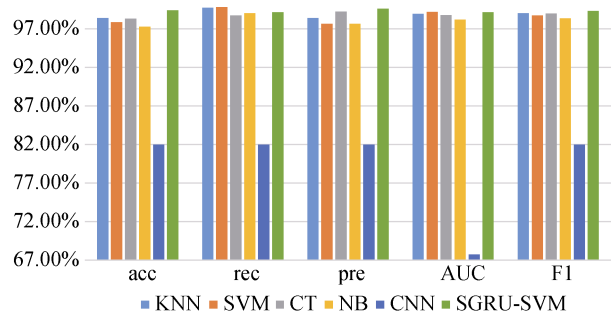


图 5 不同算法模型效果比较

Figure 5 Comparisons of Different Algorithmic Models

如图 5 所示, 在相同数据集下, SEMBeF 采用的 SGRU-SVM 模型在正确率(acc), 精确率(pre)以及 F1 指标上均达到最优, 分别为 99.40%, 99.60%, 99.29%, AUC 指标仅比 SVM 低 0.05%, 为 99.12%, 召回率(*rec*)指标比 SVM 模型低 0.7%, 比 KNN 模型低 0.57%, 为 99.11%。

综上所述, 相比于传统机器学习算法和卷积神经网络 CNN, SEMBeF 系统的准确率, 稳定性都更有优势。

5.5 SEMBeF 系统检测性能测试

SEMBeF 系统对未知样本进行检测所需时间主要包括两个部分。1)使用 Cuckoo 检测运行样本, 获取其 API 序列, 对于不同网络模型, 动态 API 序列提取时间是相同的, 因此无需比较; 2)基于敏感词向量词汇表及深度神经网络模型进行检测。本文分别基于 GRU、SGRU、SGRU-SVM 模型对未知样本进行检测, 平均消耗时间如表 9 所示。

表 9 检测时间比较

Table 9 Comparison of detection time

模型	时间(ms)
GRU	1047
SGRU	970
SGRU-SVM(SEMBeF 系统)	717

由于 SEMBeF 采用了并行计算神经网络架构 SGRU, 且超参数进行了专门优化, SEMBeF 系统的检测性能仍为最高, 所需时间平均为 717ms, 比 SGRU 模型减少了 26.08%, 比 RNN 模型减少了 31.52%。

6 下一步研究方向

本文基于循环神经网络进行了代码动态行为分析, 取得了较好的恶意代码检测效果, 但仍有一些改进空间。首先, 当前的 SEMBeF 系统仅基于样本调用的 API 函数名进行了分析, 但正常样本和恶意样本也可能调用相同的 API 函数, 只是传递的参数或者

参数值不同, 因此存在一定的误报可能, 在后续研究中, 可以综合考虑 API 函数、参数及参数取值特征进一步研究; 另外, 本文设置的动态行为提取时间为五分钟, 如果恶意代码在此过程中没有触发恶意行为, 则检测不到该恶意代码, 因此也存在一定的漏报可能。后续研究可在适当增加检测时间的同时, 结合静态分析和动态分析, 提高检测准确性。

7 结论

基于动态行为分析的恶意代码检测技术虽然能够发现代码的真实意图, 但是仍然需要大量的专家经验, 而循环神经网络在处理时序信息方面具有独特的优势, 能够减少人工分类成本, 因此本文提出了一种基于 SRNN 的恶意代码行为检测架构 SEMBeF 及系统, 能够高效准确的检测恶意代码。其中, 我们提出的安全敏感 API 权重增强的敏感词向量算法使代码表示结果既包含了上下文信息又包含安全敏感权重信息; 提出的 SGRU-SVM 网络结构通过并行计算大幅降低了训练时间, 提高了检测正确率; 最后针对样本平衡和网络模型超参数选择问题进行了优化, 进一步提高了检测正确率。实验表明, 与其他基于经典词向量、深度神经网络模型及常用的机器学习方法相比, SEMBeF 不仅检测正确率最高, 训练效率和模型稳定性也得到了显著提升。其中, 检测正确率和训练时间分别为 99.40%和 210 分钟, 与传统 RNN 相比, 正确率提高了 0.48%, 训练时间下降了 96.6%。

参考文献

[1] D. H. Lee, I. S. Song and K. J. Kim. “A study on malicious codes pattern analysis using visualization” 2011 nternational Conference on Information Science and Application(ICISA). IEEE. 2011: pp.1-5.

[2] Y. Bengio“Learning Deep Architectures for AI,”. Foundations & Trends® in Machine Learning, 2009, 2(1): pp.1-55.

[3] T. Mikolov, M. Karafiat and L. Burget. “Recurrent neural network based language model”. INTERSPEECH 2010,Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September. 2010: pp.1045-1048.

[4] Y. C. Qiao,Q. S. Jiang, and L. Zhan. “Malware Classification Method Based OH Word Vector ofAssembly Instruction and CNN”. Netinfo Security, 2019, 19(4): pp.20-28.
(乔延臣, 姜青山, 占亮, 等. “基于汇编指令词向量与卷积神经网络的恶意代码分类方法研究”. *信息网络安全*, 2019, 19(4): pp.20-28).

[5] C. Willems, T. Holz, and F. Freilin. “Toward Automated Dynamic Malware Analysis Using CWSandbox,” IEEE Security & Privacy, 2007, 5(2): pp.32-39.

[6] “Cuckoo sandbox”, https://cuckoosandbox.org/

[7] R. Tian, R. Islam, and L. Batten. “Differentiating malware from cleanware using behavioural analysis,” Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on. IEEE, 2010.

[8] Y. C. Qiao, X. C. Yun, and Y. Z. Zhang, et al. “An Automatic Malware Homology Identification Method Based on Calling Habits,” Acta Electronica Sinica, 44(10): pp.2410-2414(in chinese), 2016.
(乔延臣, 云晓春, 张永铮, 等.” 基于调用习惯的恶意代码自动化同源判定方法”, *电子学报*, 2016, 44(10): pp.2410-2414.)

[9] G. Matthew. Schultz, Eleazar Eskin and Erez Zadok. “Data Mining Methods for Detection of New Malicious Executables,” IEEE Computer Society, 2001: pp.38-49.

[10] T. Abou-Assaleh, N. Cercone, and V. Keselj. “N-garm-based detection of new malicious code,” Proceedings of the 28th Annual Internation Computer Software and Applications Conferences, 2004(COMPSAC 2004). IEEE, 2004, 2: pp.41-42.

[11] Y. Lai “A feature selection for malicious detection,” Ninth ACIS International Conference on Softmare Engineering,Artificial Intelligence,Networking, and Parallel/ Distributed Computing, 2008(SNPD’08). IEEE, 2008: pp.365-370.

[12] Y. Ding, X. Yuan, and K. Tang. “A fast malware detection algorithm based on objective-oriented association mining,” Computers & Security, 2013, 39(4): pp.315-324.

[13] D. Zhang, Y. Zhang, G. Liu and G. X. Song. “Research on host malcode detection using machine learning,” Chinese Journal of Network and Information Security, 3(07): pp.25-32(in chinese), 2017.
(张东, 张尧, 刘刚, 宋桂香.” 基于机器学习算法的主机恶意代码检测技术研究”, *网络与信息安全学报*, 2017, 3(07): pp.25-32.)

[14] W. F. Zhang, R. C. Liu and L. Xu. “Web Page Trojan Detection Method Based on Dynamic Behavior Analysis”, Journal of Software. v.29(05): pp.238-249(in chinese), 2018.
(张卫丰, 刘蕊成, 许蕾.” 基于动态行为分析的网页木马检测方法”, *软件学报*, 2018, v.29(05): pp.238-249.)

[15] A. Chinea. “Understanding the Principles of Recursive Neural Networks: A Generative Approach to Tackle Model Complexity,”. Lecture Notes in Computer Science, 2009, 5768: pp.952-963.

[16] T. Mikolov, T. Chen, and G. Corrado. “Efficient estimation of word representations in vector space,” In:Proceedings of the International Conference on Learning Representations. Scottsdale, USA: 2013.

[17] B. Kolosnjaji, A. Zarras, and G. Webster. “Deep Learning for Classification of Malware System Call Sequences,” Australasian Joint Conference on Artificial Intelligence. Springer, Cham, 2016.

[18] R. Pascanu, J. W. Stokes, and H. Sanossian. “Malware classification with recurrent networks,” IEEE International Conference on Acoustics. IEEE, 2015.

[19] B. Athiwaratkun and J. W. Stokes. “Malware classification with LSTM and GRU language models and a character-level CNN,” ICASSP 2017-2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2017.

[20] I. Kwon and E. G. Im. “Extracting the Representative API Call Patterns of Malware Families Using Recurrent Neural Network,” the International Conference. 2017: pp.202-207.

[21] Y. Ye, L. Chen and H. Hou. “DeepAM: a heterogeneous deep learning framework for intelligent malware detection,” Knowledge & Information Systems, 2017(4): pp.1-21.

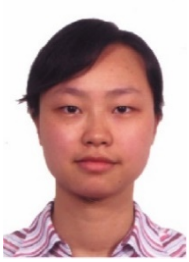
[22] Y. Fang, X. Long and C. Huang. “Research on classifying phishing URLs using hybrid architecture of LSTM and random forest”. Advanced Engineering Sciences, 2018, 50(5): pp.196–201.
(方勇,龙啸,黄诚,等.”基于 LSTM 与随机森林混合构架的钓鱼网站识别研究”. *工程科学与技术*, 2018, 50(5): pp.196–201.)

[23] Y. Jinpei,Q. Yong and R.Qifan. “Detecting Malware with an Ensemble Method Based on Deep Neural Network,” Security and Communication Networks. 2018.

[24] S. Tobiyama, Y. Yamaguchi and H. Shimada. “Malware Detection with Deep Neural Network Using Process Behavior,” Computer Software and Applications Conference. IEEE, 2016: pp.577-582.

[25] Z. Yu and G. Liu. “Sliced Recurrent Neural Networks,” The 27th International Conference on Computational Linguistics (COLING 2018), 2018.

[26] S. Hochreiter, Schmidhuber and Jürgen. “Long Short-Term Memory,” Neural Computation, 1997, 9(8): pp.1735-1780.



詹静 于 2009 年在武汉大学计算机科学与技术(信息安全)专业获得博士学位。现任北京工业大学讲师。研究领域为可信计算。研究兴趣包括: 大数据安全, 云计算安全。Email: zhanjing@bjut.edu.cn

范雪 于 2018 年在枣庄学院计算机科学与技术专业获得学士学位。现在北京工业大学计算机科学与技术(信息安全)专业攻读硕士学位。研究领域为大数据安全。研究兴趣包括: 大数据安全。Email: fanxue_1996@126.com

刘一帆 于 2018 年在山东师范大学计算机科学与技术专业获得学士学位。现在北京工业大学计算机技术(信息安全)专业攻读硕士学位。研究领域为工业控制安全。研究兴趣包括: 密码学, 云安全。Email: liuyifan_1998er@126.com

张茜 于2017年在长治学院网络工程专业获得学士学位。现在北京工业大学计算机技术(信息安全)专业攻读硕士学位。研究领域为信息安全, 工业控制安全。研究兴趣包括: 内存安全。Email: 1619042644@qq.com