基于注意力机制的恶意软件调用序列检测

张 岚 来 耀 叶晓俊
(清华大学软件学院 北京 100084)

摘 要 传统的机器学习方法通过构造特征来学习分类器,面对嵌入大量反检测功能的恶意软件不具有鲁棒性。攻击者通过打乱恶意软件代码或插入无关代码来逃避检测。针对互联网环境下恶意软件数目众多、混淆技术进步、人工构造特征成本高等问题,文中提出一种基于循环神经网络和注意力机制的恶意软件检测方法(G2ATT)。首先,在沙盒环境下运行软件获取其动态调用序列(API),并通过滑动窗口划分得到窗口子序列;其次,引入多示例学习和注意力机制来构建层次化特征抽取的深度神经网络,使用循环神经网络抽取 API 特征,结合两个注意力机制分别抽取窗口特征和序列特征,并使用序列特征检测恶意软件;最后,使用真实数据训练网络,以便使用得到的模型对未知恶意软件进行检测。基于真实数据集的实验结果表明,窗口特征抽取层和序列特征抽取层能够有效学习窗口内和窗口间的注意力权重,从而更好地描绘序列的特征,提升模型的查准率和查全率。G2ATT 对未知恶意软件检测的准确率达到 98.19%,查准率达到 98.78%,查全率达到 97.60%,AUC (Area Under the Curve of ROC)达到 99%,比基于 API 调用序列的 SVM、随机森林、朴素贝叶斯等方法的准确率提高了 10%以上。

关键词 恶意样本检测,深度学习,注意力机制,调用序列
中图法分类号 TP309.5 **文献标识码** A **DOI** 10.11896/jsjkx.181102171

Attention Mechanism Based Detection of Malware Call Sequences

ZHANG Lan LAI Yao YE Xiao-jun
(School of Software, Tsinghua University, Beijing 100084, China)

Abstract Typical machine learning approaches, which learn a classifier based on hand crafted features, are not sufficiently robust. Attackers can reorder the malware code or insert useless code to avoid detection. Aiming at the problems of the large number of malware, confusion technology progress and the cost of artificially constructed feature in the Internet environment, this paper proposed a different malware detection approach G2ATT based on API call sequence and attention mechanism in natural language process. First, dynamic API call sequences are extracted by using the sandbox environment and split them into several subsequences by using a sliding window. Then, the concept of multi-instance learning and attention mechanism are introduced to design the hierarchical feature extraction neural networks. Recurrent neural networks are used for API-level features. Two attention mechanism are combined to extract window-level features and sequence-level features. Then, those sequence-level features are used for malware detection. Ultimately, the model is trained and used to detect malware. The experimental results based on real dataset show that the window-level feature extraction layer learns effectively attention scores in the subsequences. In addition, the sequence-level feature extraction layer improves the performance of malware detection model on precision and recall by calculating attention scores across the subsequences. G2ATT achieves 98.19% on detection accuracy rate, 98.78% on precision rate, 97.60% on recall rate and 99% on AUC (Area Under the Curve of ROC), which improves by 10% compared with other machine learning approaches based on API call sequences on detection accuracy.

Keywords Malware detection, Deep learning, Attention mechanism, API

1 引言

目前,互联网服务已经进入了云计算、大数据和物联网时代,数据丢失和个人隐私泄露等安全风险使得安全防护要求不断提升。以恶意程序攻击检测为例,当今互联网安全公司每天截获的新增样本数以万计。传统的特征码匹配方法通过自动或手动选取可以代表某种恶意软件的二进制字节序列,进而使用字节序列匹配的方法检测恶意程序[1]。该方法能够在一定程度上提高检测的效率,但往往需要消耗很高的人力成本,且该方法需要不断更新特征库,才能对新出

到稿日期:2018-11-25 返修日期:2019-04-15 本文受国家重点研究计划项目(2019QY1402)资助。
张 岚(1994—),女,硕士生,主要研究方向为恶意软件检测、机器学习;来 耀(1994—),男,硕士生,主要研究方向为恶意软件检测、机器学习;叶晓俊(1964—),男,教授,CCF 会员,主要研究方向为数据库安全、数据库技术,E-mail:yexj@tsinghua.edu.cn(通信作者)。

现的恶意程序具有检测能力。

基于以上原因,人们开始设计基于机器学习的恶意程序自动检测方法,该方法通过使用训练模型可对恶意程序进行自动化检测。这种方法具有很强的泛化能力,可以学习到深层的软件行为,从而识别出恶意软件的变种或新出现的恶意软件。基于机器学习的方法主要包括基于专家定义的特征方法、基于软件原始字节序列的恶意软件检测方法、基于软件控制依赖图的恶意软件检测方法以及基于恶意程序运行的 API 序列特征的恶意软件检测方法等[2]。基于 API 序列特征的方法具有平台可移植性,且在互联网环境下程序一般是需要通过一系列的 API 调用来实现恶意行为,因而通过 API 序列特征提取进行恶意软件的检测具有广泛的应用前景。但该方法容易受到序列顺序变换、无关代码插入等代码混淆方法的攻击[3],使得基于机器学习的 API 序列行为检测方法受到挑战。

本文提出了层次化的循环神经网络模型,首先通过滑动窗口将恶意软件的 API 调用序列分解成若干窗口子序列,使每个子序列通过循环神经网络学习 API 特征;然后加入窗口层注意力机制学习窗口子序列内的特征;最终将学习到的每一个子序列特征通过序列层注意力机制组合成序列特征进行软件的分类。注意力机制在多个领域都有成功的应用,如图像领域、机器翻译领域等[4]。本文将窗口层和序列层两种注意力机制相组合,学习子序列内和子序列间的特征,并将注意力信息通过特征加权体现在恶意软件检测上。实验结果表明,本文的模型使得恶意软件检测的准确率达到 98.19%,比传统机器学习方法的准确率至少高出 10%,比直接使用循环神经网络的准确率高出 7%。

2　研究现状分析

恶意软件检测可分为静态方法和动态方法。静态方法是在不运行软件的情况下分析程序指令与结构来确定程序功能,例如通过分析 PE 文件头获取软件的导入函数等特征,或者通过反汇编工具对软件进行反汇编,获取其函数调用特征进行训练。Saxe 等从 Windows 可执行文件中提取上下文字节特征、PE 导入特征、字符串熵特征,将提取的特征输入三层神经网络模型进行训练,以实现恶意软件检测[5]。Arp 等从移动互联网应用程序中,通过反汇编提取函数调用、使用权限、连接网络地址等特征,通过支持向量机方法实现了恶意移动应用的检测[6]。Nataraj 等将原始的恶意软件二进制文件字节序列转化为二维灰度图的形式,并使用计算机视觉的方法来检测恶意软件[7]。静态方法的优势在于检测时不需执行软件,效率较高且相对安全,但由于大多数恶意软件都经过了混淆处理,因此使用静态方法分析混淆软件会比较困难。

动态方法是指在某种仿真环境中运行恶意程序以提取程序行为信息(例如 API 调用序列)来进行恶意软件的检测。与静态方法相比,动态方法可以抵御部分软件加壳和加密等阻止分析的混淆技术。Kolosnjaji 等提取动态运行过程中系统调用的 API 序列,并将其输入循环神经网络训练,通过序列分类技术来实现恶意软件的检测[8]。Xu 等借鉴文本序列距离的思想,定义 API 调用序列间的距离,通过计算已知的恶意软件与待检测的软件之间的距离来实现恶意软件的检测[9]。Tobiyama 等根据 API 调用序列,通过循环神经网络生成特征向量,并将其转化成特征图,再通过卷积神经网络进行恶意软件的判别[10]。

由于软件运行时产生的 API 调用序列可能过长,因此可以通过滑动窗口将原始长序列划分为若干子序列,进而将原始序列的检测问题转化为子序列的分类问题来解决。Rosenberg 等将 API 序列划分为不同的子序列,并分别通过循环神经网络进行检测,只要其中某个子序列被标记为恶意的,整个序列就可标识为恶意的[11]。多示例学习(MIL)为该方法提供了实践依据,将原始的样本看作一个包,并按一定规则将其划分为若干个示例。我们可以将该方法应用于恶意软件的检测,将每个序列看作包,将子序列看作多个示例。Rosenberg 的方法基于恶意软件的 API 调用序列必存在恶意子序列的假设,但此假设没有考虑恶意序列特征间隔较远的情况,因此攻击者可以采用调用序列顺序替换的方法来重新生成恶意软件样本,绕过判别模型。

3　基于注意力机制的恶意软件检测

3.1　模型描述

软件行为一般包括文件管理、线程调度、UI 操作等各种 API 调用序列,因此使用 API 调用序列就可描述软件的动态行为。业界一般使用沙箱执行待检测软件,解析得到动态 API 调用序列。本文对处理后的 API 序列进行建模,使用循环神经网络等深度学习方法对软件进行分类。

本文设计的恶意软件 API 行为检测框架(GRU with 2 Attention Mechanism,G2ATT)如图 1 所示。

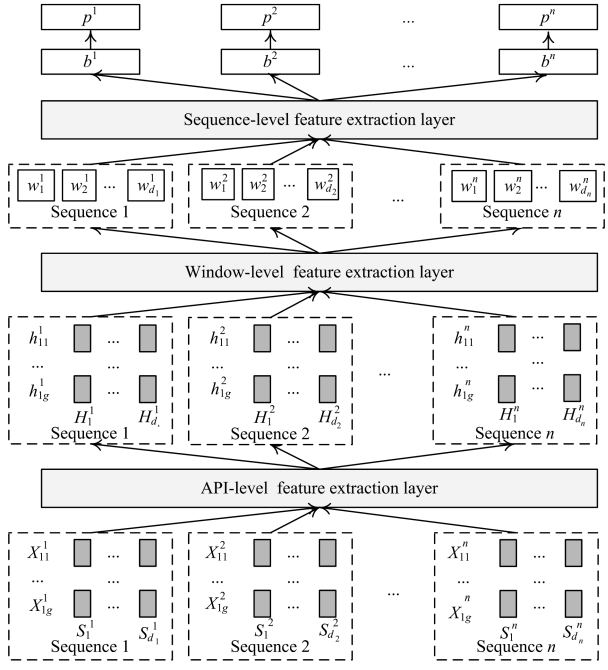

图 1　G2ATT 模型示意图

Fig. 1　Illustration of G2ATT

1)输入层:软件 API 调用序列被转换成 API 向量表示,通过滑动窗口将长 API 序列划分成多个窗口子序列的 API 向量。

2)API 特征学习层(API-level Feature Extraction Layer):输入窗口子序列的 API 向量,使用 GRU(Gated Recurrent Units)学习程序调用序列 API 特征。

3)窗口特征抽取层(Window-level Feature Extraction Layer):依据 API 特征,学习窗口子序列中 API 的重要性,从而抽取出窗口特征。即借鉴关系抽取中的注意力机制[12]学习窗口子序列中每个 API 特征的重要性,从而增大包含具有重要特征的窗口子序列的权重,对其加权后作为窗口特征。

4)序列特征抽取层(Sequence-level Feature Extraction Layer):基于所有窗口子序列特征学习窗口子序列的重要性,增加恶意窗口子序列的权重,降低良性窗口子序列的权重,从而得到序列的特征。序列特征抽取层计算所有窗口子序列间的注意力矩阵,学习序列整体的特征[13]。

5)输出层:基于序列特征向量对待检测序列进行分类。

G2ATT 的输入为软件 API 调用序列。通过对 API 序列建模,得到其窗口形式的向量表示。使用窗口特征抽取层得到软件调用序列每个窗口的局部 API 特征,再经过序列特征抽取层学习所有窗口整体的特征,并使用这个特征对软件分类。

对于 G2ATT 模型的输入 API 调用序列,通过输入层将其处理成多个窗口子序列,s_j^i 是第 i 个序列的第 j 个窗口子序列,x_{jk}^i 是 s_j^i 的第 k 个 API 的 one-hot 向量表示。图 1 中 d_i 是第 i 个序列的窗口子序列的个数,g 是滑动窗口大小。在 API 特征学习层,每个 API 都会学习到其对应的特征 $\boldsymbol{h}_{jk}^i$。同一窗口的 API 特征组合成第 i 个序列的第 j 个窗口的特征矩阵 $\boldsymbol{H}_j^i=\{\boldsymbol{h}_{j1}^i,\boldsymbol{h}_{j2}^i\cdots,\boldsymbol{h}_{jg}^i\}$。通过抽取窗口特征,最终得到该窗口的特征向量 $\boldsymbol{W}^i=\{w_1^i,w_2^i,\cdots,w_{d_i}^i\}$。序列特征抽取层通过注意力机制学习到序列各个窗口中的权重信息,得到序列特征 $\boldsymbol{b}^i$,并计算其属于各个类别的概率 $\boldsymbol{p}^i$。

3.2 输入层

3.2.1 API 序列的向量表示

恶意软件检测可以转化为序列分类问题[11]。软件 API 调用序列为 $\boldsymbol{S}^i=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_t\}$,其中 t 为此序列长度,$\boldsymbol{x}_k$ 是此序列第 k 个位置 API 的 one-hot 向量表示。每个调用序列会增加起始标志(START)和结束标志(END),即对于序列 i,其向量表示会转化为 $\boldsymbol{S}^i=\{\boldsymbol{START},\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_t,\boldsymbol{END}\}$。

每个 API 都被表示成 one-hot 向量 $\boldsymbol{x}_k$。假设一共存在 m 个 API 函数,所有 API 按顺序排列并从 1 到 m 编号,加上起始标志和结束标志,则每个 API 的 one-hot 向量特征向量都是 $m+2$ 维二进制向量。其中,起始标志向量为 $\boldsymbol{START}=\{1,0,\cdots,0\}$,结束标志向量为 $\boldsymbol{END}=\{0,\cdots,0,1\}$,编号是 k 的 API 表示为 $\boldsymbol{x}_k=\{0,x_1,x_2,\cdots,x_k,\cdots,x_m,0\}$,其中 $x_k=1$,其他维度位置的值都为 0。

3.2.2 子序列建模

分类器的输入是待检测软件的 API 调用序列。由于软件的调用序列可能很长,模型训练时可能会占用大量 GPU 资源,也会超出训练的时间限制[11]。因此,本文采用滑动窗口的方式对输入序列 $\boldsymbol{S}^i$ 进行处理。

限制窗口长度为 g,滑动步长为 l,则序列重复率为 $(g-l)/g$。长度小于 g 的窗口子序列用零向量填充。序列 $\boldsymbol{S}^i$ 经过滑动窗口划分后,得到 d 个窗口子序列,即 $\boldsymbol{S}^i=\{\boldsymbol{s}_1^i,\boldsymbol{s}_2^i,\cdots,\boldsymbol{s}_d^i\}$,其中第 j 个窗口子序列为 $\boldsymbol{s}_j^i=\{\boldsymbol{x}_{j1}^i,\boldsymbol{x}_{j2}^i,\cdots,\boldsymbol{x}_{jg}^i\}$,则 $\boldsymbol{s}_j^i\in\mathbb{R}^{g\times m}$,$\boldsymbol{S}^i\in\mathbb{R}^{d\times(g\times m)}$。

如果一个软件调用序列的任意窗口子序列被标注为恶意软件的序列,则此软件为恶意软件[11]。基于此假设,可采用多示例学习思路对待检测软件进行分类[11]:将每个 $\boldsymbol{S}^i$ 看作一个包(bag),每个包由一系列 $\boldsymbol{s}_j^i$ 组成。每个包有一个训练标记,如果标记为良性的,则包中所有 $\boldsymbol{s}_j^i$ 都是良性的;如果标记为恶意的,则包中至少有一个 $\boldsymbol{s}_j^i$ 是恶意的。因此只需要学习一个模型,分类包中的 $\boldsymbol{s}_j^i$,如果有一个被标记为恶意,则认为整个包为恶意的;如果全为良性,则认为此软件是良性软件[11]。

3.3 API 特征学习层

GRU 是一种特殊的循环神经网络(RNN)[14]。GRU 可以有效地学习软件 API 调用的行为特征,捕捉序列数据中过去数据对现在数据的影响,从而更好地描绘 API 及其上下文的特征。

输入 $\boldsymbol{S}^i$ 的第 j 个窗口子序列 $\boldsymbol{s}_j^i=\{\boldsymbol{x}_{j1}^i,\boldsymbol{x}_{j2}^i,\cdots,\boldsymbol{x}_{jg}^i\}$。通常,一个 GRU 单元包含更新门和重置门[14],更新门计算 API $\boldsymbol{x}_{jt}^i$ 需要保留的信息量,重置门确定其需要遗忘的信息。

对于输入 $\boldsymbol{S}^i$,窗口子序列 $\boldsymbol{s}_j^i$ 中的 API $\boldsymbol{x}_{jt}^i$ 的 API 特征为 $\boldsymbol{h}_{jt}^i$。

3.4 窗口特征抽取层

对于窗口子序列 $\boldsymbol{s}_j^i=\{\boldsymbol{x}_{j1}^i,\boldsymbol{x}_{j2}^i,\cdots,\boldsymbol{x}_{jg}^i\}$,API 特征学习层的输出矩阵 $\boldsymbol{H}_j^i=\{\boldsymbol{h}_{j1}^i,\boldsymbol{h}_{j2}^i,\cdots,\boldsymbol{h}_{jg}^i\}$,其中 $\boldsymbol{H}_j^i\in\mathbb{R}^{c\times g}$,$g$ 是窗口子序列的长度,c 是 GRU 每个单元隐藏层状态的维度。不同 API 的重要程度并不一样,窗口特征抽取层通过学习 API 特征的注意力分布,将 $\boldsymbol{s}_j^i$ 的 API 特征 $\boldsymbol{H}_j^i$ 转换成窗口子序列特征。本文基于注意力机制[12]学习 $\boldsymbol{s}_j^i$ 的窗口子序列特征 $\boldsymbol{w}_j^i$,通过计算输入特征的权重分布,抽取其中的关键信息:

$$\boldsymbol{M}_j^i=\boldsymbol{\mu}^{\mathrm{T}}\tanh(\boldsymbol{H}_j^i) \tag{1}$$

其中,$\boldsymbol{\mu}^{\mathrm{T}}\in\mathbb{R}^{1\times g}$ 是待学习权重 $\boldsymbol{\mu}$ 的转置,$\boldsymbol{M}_j^i\in\mathbb{R}^{l\times g}$。通过学习 API 的权重向量,合并窗口中 API 的特征。再通过归一化得到注意力分布 $\boldsymbol{\alpha}_j^i$:

$$\boldsymbol{\alpha}_j^i=\frac{\exp(\boldsymbol{M}_j^i)}{\sum_{k=1}^{g}\exp((\boldsymbol{M}_j^i)_k)} \tag{2}$$

其中,$\boldsymbol{\alpha}_j^i\in\mathbb{R}^{1\times g}$。注意力分布 $\boldsymbol{\alpha}_j^i$ 反映了序列中不同 API 的重要程度。基于注意力分布 $\boldsymbol{\alpha}_j^i$ 和 API 特征 $\boldsymbol{H}_j^i$,计算当前窗口子序列的上下文特征 $\tilde{\boldsymbol{w}}_j^l$:

$$\tilde{\boldsymbol{w}}_j^l=\boldsymbol{H}_j^i\boldsymbol{\alpha}_j^{i\,\mathrm{T}} \tag{3}$$

最终,窗口子序列特征 $\boldsymbol{w}_j^i$ 为:

$$\boldsymbol{w}_j^i=\tanh(\tilde{\boldsymbol{w}}_j^l) \tag{4}$$

其中,$\boldsymbol{w}_j^i\in\mathbb{R}^{c\times 1}$。

3.5 序列特征抽取层

对于 $\boldsymbol{S}^i$,通过滑动窗口得到多个 $\boldsymbol{s}_j^i$,经过 API 特征学习层和窗口特征抽取层得到序列特征 $\boldsymbol{W}^i=\{\boldsymbol{w}_1^i,\boldsymbol{w}_2^i,\cdots,\boldsymbol{w}_d^i\}$,$\boldsymbol{W}^i\in\mathbb{R}^{d\times c}$,$\boldsymbol{w}_j^i$ 表示 $\boldsymbol{s}_j^i$ 的窗口子序列特征。

恶意软件不一定只存在恶意活动,部分恶意软件为了伪装自己的恶意行为,会执行一些良性的行为,因此同一序列的窗口子序列的重要程度并不一样。本文采用注意力机制[13]来捕捉窗口子序列与原序列类别的信息,从而增加原序列特征中关键窗口子序列的权重。计算 $\boldsymbol{w}_j^i$ 与 $\boldsymbol{S}^i$ 类别的相关程度,

也就是各个窗口的注意力分布 $\boldsymbol{\alpha}_j^i$:

$$\boldsymbol{\alpha}_j^i=\frac{\exp(\boldsymbol{e}_j^i)}{\sum_{k=1}^{d}\exp((\boldsymbol{e}_j^i)_k)} \tag{5}$$

$$\boldsymbol{e}_j^i=\boldsymbol{w}_j^i\boldsymbol{A}\boldsymbol{r} \tag{6}$$

其中,$\boldsymbol{A}$ 为注意力对角矩阵,$\boldsymbol{r}$ 表示此窗口对应原类别的向量,因此 $\boldsymbol{e}_j^i$ 表示窗口与类别的相关性。序列 $\boldsymbol{S}^i$ 的特征 $\boldsymbol{b}^i$ 是 $\boldsymbol{S}^i$ 的所有窗口子序列的线性组合:

$$\boldsymbol{b}^i=\sum_{j=1}^{d}\boldsymbol{\alpha}_j^i\boldsymbol{w}_j^i \tag{7}$$

测试时,由于序列类别未知,不能像训练时一样使用序列对应类别的向量 $\boldsymbol{r}$,因此分别计算每个窗口子序列与良性和恶意两个类别的向量的注意力权重,再通过加权得到此序列的向量表示。

3.6　输出层

多个序列组合成矩阵 $\boldsymbol{S}=\{\boldsymbol{S}^1,\boldsymbol{S}^2,\cdots,\boldsymbol{S}^n\}$,经过序列特征抽取,得到序列特征 $\boldsymbol{B}=\{\boldsymbol{b}^1,\boldsymbol{b}^2,\cdots,\boldsymbol{b}^n\}$,$\boldsymbol{B}\in\mathbb{R}^{n\times c}$,$n$ 为序列个数。输出时,将 $\boldsymbol{b}^i$ 与每个类别向量的相似度作为其属于此类别的可能性 $\boldsymbol{o}$,即:

$$\boldsymbol{o}=\boldsymbol{B}\boldsymbol{R}^{\mathrm{T}}+\boldsymbol{d} \tag{8}$$

其中,$\boldsymbol{R}$ 是所有类别向量的矩阵,$\boldsymbol{d}\in\mathbb{R}^{n_r}$ 是偏置向量,$\boldsymbol{R}\in\mathbb{R}^{n_r\times c}$,$n_r$ 是类别总个数。对结果进行归一化,得到:

$$\Pr(r|\boldsymbol{B},\boldsymbol{\theta})=\frac{\exp(o_r)}{\sum_{k=1}^{n_r}\exp(o_k)} \tag{9}$$

其中,$\boldsymbol{\theta}$ 是网络中所有的参数,o_r 是输入序列属于第 r 个类别的可能性,$\Pr(r|\boldsymbol{B},\boldsymbol{\theta})$ 是 $\boldsymbol{B}$ 中每个序列属于类别 r 的概率。

3.7　目标函数

每次随机有放回地抽样训练集中的数据,使用自适应矩估计 Adam 算法(Adaptive Moment Estimation)来最小化目标函数[15]。本文实验时以一定概率丢弃(dropout)部分序列特征以防止过拟合[16]。训练时,目标函数使用交叉熵:

$$J(\boldsymbol{\theta})=\sum_{i=1}^{n}\log\Pr(r_i|\boldsymbol{b}_i,\boldsymbol{\theta}) \tag{10}$$

4　实验

本节通过一系列实验对 G2ATT 以及其他机器学习模型进行比较,分析窗口特征抽取层和序列特征抽取层的优势,并提出窗口长度和窗口子序列重复率参数的选择建议。

4.1　数据集

我们从 Malekal 恶意软件样本网站上以及北京大学提供的 CILPKU08 和 Henchiri-Dataset 数据库中收集了 6 946 个恶意软件。良性软件来源于 completely free software,softonic,Windows XP 系统文件等不同类型的常用应用程序,共 2 794 个。使用 VirusTotal 网站提供的 API 服务对爬取到的数据进行自动标注。VirusTotal 包含 60 多个不同安全厂商提供的恶意程序检测引擎。只有当所有检测引擎都判定这个样本为良性时才会将其标注为良性;如果多于 15 个检测引擎判定该样本为恶意,则认为它是恶意样本[17]。使用 Kaspersky 检测引擎的结果作为恶意软件的判断标签,并根据恶意软件的 md5 值进行恶意软件数据的去重处理。最终,恶意软件包含 Backdoor(后门程序)、Trojan-Downloader(恶意下载器)、Trojan-Ransom(勒索木马)、AdWare(广告软件)、Worm(蠕虫)等恶意软件家族。

我们将程序样本放入 Cuckoo 沙盒环境中运行,截取程序运行 2 min 以内的运行情况。实验使用的大部分恶意样本通过 VirtualBox 执行 Windows XP 的镜像,根据运行报告的 JSON 文件提取程序的 API 调用序列作为样本的动态特征。若使用其他高版本的 Windows 镜像,如 Windows 10,则需要注意其对程序运行的权限管理更为严格,因此使用 Cuckoo 沙盒环境运行软件时,需要在 Windows 10 安全设置上把 Cuckoo 的 agent.py 文件的运行权限提升至管理员权限,从而保证软件的正常运行,以获取其动态调用序列。

由于一些恶意软件使用反虚拟机技术,当其检测到运行环境为沙盒环境时,会自动执行非恶意的行为或直接结束程序,因此即使此样本被标注成恶意软件,其在沙盒环境下的行为也并不是恶意的。对此我们在实验中采用了 YARA 规则来忽略此类样本,并去除了过短的 API 调用序列的样本[11],只保留序列长度大于 15 的样本(检测到虚拟机并退出)[17]。恶意软件的调用序列存在大量冗余,为了保证数据信息量不变,并减少序列长度,需要去除连续相同的 API 函数[17]。通过以上方法过滤掉不合格的样本,考虑到两类样本的均衡性,最终选取了 2 497 个恶意和 2 497 个良性程序样本进行实验。

4.2　评估

为了使评估结果更准确,使用十折交叉验证来评估不同检测方法的效果。本文主要从以下几个角度来评估模型,其中,N_{TP} 表示恶意软件序列被正确分类为恶意软件的序列数,N_{FP} 表示良性软件序列被错误分类为恶意软件的序列数,N_{FN} 表示恶意软件被错误分类为良性软件的序列数,N_{TN} 表示良性软件被正确分类为良性软件的序列数。

1)查准率:模型预测为恶意软件的样本中实际为恶意软件的样本所占的比例。

2)查全率:实际为恶意软件的样本中模型正确预测为恶意软件的样本所占的比例。

3)准确率:全体样本中正确判断的样本所占的比例。

4)特征曲线(ROC)和曲线区域面积(AUC):综合度量查准率和查全率,评估模型性能。

4.3　参数设置

我们使用 Tensorflow 和 Scikit-Learn 库实现本文提出的模型以及其他模型,模型训练使用了 8 块 GTX 1080 Ti,表 1 为具体实验参数。

表 1　实验参数

Table 1　Parameters of experiment

指标	参数
嵌入层	输入维度:283 每个 batch 限制子序列个数:2000 窗口大小:600 序列重复率:50%
API 特征学习层	记忆单元维度 Dropout:0.5
序列特征抽取层	Dropout:0.5
其他参数	学习率:0.001 Max Epochs:15 梯度剪裁:1.0

4.4 实验 1 检测未知恶意软件

本实验实现的基于 API 序列的机器学习模型的性能对比如表 2 所列。每个模型都是在相同参数下训练得到。

表 2 各模型的性能对比

Table 2 Performance comparison of models

(单位:%)

实验编号	模型	*ACC*	*PR*	*RC*
1	Decision Tree	82.08	76.79	92.00
2	Random Forest	88.29	87.62	89.20
3	Naïve Bayes	70.37	67.83	77.60
4	SVM	87.28	83.13	93.60
5	RNN	91.09	98.12	83.80
6	GRU	91.69	97.27	85.80
7	GRU+Window-level ATTENTION	95.69	98.72	92.60
8	GRU+Sequence-level ATTENTION	94.19	96.82	91.40
9	RNN+2ATTENTION	96.99	97.18	96.80
10	LSTM+2ATTENTION	96.79	97.36	96.20
11	G2ATT	98.19	98.78	97.60

1)传统机器学习方法,包括决策树(Decision Tree)、随机森林(Random Forest)、朴素贝叶斯(Naïve Bayes)以及支持向量机(SVM)方法。实验时,均使用滑动窗口将原序列划分成窗口子序列进行训练和测试,窗口长度与序列重复率的设定与表 1 相同。截取后的窗口子序列转化成 OneHot 编码的特征向量,使用截断奇异值分解降维后,将每一段窗口子序列作为一条数据进行训练。测试时,同样遵循只要一个子序列为恶意则整段序列为恶意的原则。我们通过网格搜索的方法找到每种方法的最佳超参数。具体参数的设置如表 3 所列。随机森林方法设置的独立决策树数目为 10 个;朴素贝叶斯方法使用先验为伯努利分布的贝叶斯方法;支持向量机方法使用径向基函数核。

表 3 机器学习方法的参数

Table 3 Parameters of machine learning algorithm

方法	参数类型	参数范围	选取参数
决策树	单个决策树的最大深度	3~无限制	无限制
	叶子节点所需的最小样本数	1~60	1
	拆分内部节点所需的最小样本数	2~150	2
随机森林	决策树个数	10~70	70
	单个决策树最大深度	3~无限制	无限制
	叶子节点所需的最小样本数	1~60	1
	拆分内部节点所需的最小样本数	2~150	2
支持向量机	惩罚系数 C	0.01~100	10
朴素贝叶斯	平滑参数 α	0~100	0

2)RNN/GRU。API 特征学习层为 RNN/GRU,学习的每个窗口的特征通过全连接层,得到每个窗口属于某个类别的概率。训练时,每个序列的损失函数使用其所有窗口对应分类的均值和实际分类结果的交叉熵。预测结果遵循以下原则[9]:若有一个窗口子序列被判断为恶意,则判断所有序列为恶意序列。

3)GRU + Window-level ATTENTION。API 特征学习层为 GRU,学习的特征通过窗口特征抽取层得到每个窗口的特征,经过全连接层,得到每个窗口属于某个类别的概率。采用相同的方式计算损失函数和预测分类。

4)GRU+ Sequence-level ATTENTION。API 特征学习层为 GRU,学习的特征经过全连接层后,在序列特征抽取层学习各个窗口的注意力权重,得到序列属于某个类别的概率。损失函数的计算与 3.7 节目标函数的设计一致。

5)RNN/LSTM+2ATTENTION。API 特征学习层为 RNN/LSTM,组合窗口特征抽取层、序列特征抽取层和本文的输出层。损失函数的计算与 3.7 节目标函数的设计一致。

深度学习的优点:表 2 中实验 1-4 和实验 5-11 的结果对比表明,传统机器学习方法都是将序列中的 API 调用看作单个独立的特征,失去了 API 调用的先后顺序关系,即出现的 API 调用位置在特征空间上的距离相等;而基于循环神经网络的深度学习方法由于输入模型后保留了序列的先后顺序关系,因此效果优于传统方法。

GRU 的优点:实验 5 和实验 6 的结果表明,使用抽取窗口子序列特征时,使用 GRU 比使用基本的 RNN 效果好,这归结于 RNN 结构上的限制,即当序列过长时,之前输入的数据对后来的影响太小。实验 11 也说明使用 GRU 比实验 9 基本的 RNN 和实验 10 LSTM 作为 API 特征抽取层的效果要好。

窗口特征抽取层的优点:对比实验 6 和实验 7 可以看到,窗口特征抽取层可以抽取出窗口中的重要特征,而且这些特征能准确刻画出恶意软件的样本特征,因此其对查全率的提升比较明显。比较实验 8 和实验 11 也能发现,添加窗口特征抽取层后,序列的查全率提升效果明显,模型正确预测为恶意软件的样本增多。

序列特征抽取层的优点:实验 6 和实验 8 表明,序列特征抽取层能够学习到窗口间的重要性,从而学习到序列特征,但其注意力更倾向于增加潜在恶意子序列的权重,因此模型会出现第一类错误(拒绝真假设),查准率降低。而实验 7 与实验 11 也表明,在窗口特征抽取层的基础上添加序列特征抽取层,模型检测的恶意软件样本增加。

双注意力机制的优点:从实验 5-6 和实验 9-11 可以看出,联合两个特征抽取层后可以减少第一类错误的出现。本文的模型结合了两个注意力层,其不仅准确率高,还可以大幅度提高查全率,这对于发现未知恶意软件很重要。

4.5 实验 2 参数选择

输入层中,划分的窗口子序列与软件行为有关。图 2 分析了在相同参数设置(序列重复率 50%)下不同窗口长度对模型特征曲线和曲线区域面积的影响,其中横坐标是假阳性率(FPR),纵坐标是真阳性率(TPR),曲线构成面积 *AUC* 越大表明分类器的效果越好。从图 2 可以看出,窗口长度太小,将不能体现软件本身的特征,因此模型性能会受到影响。

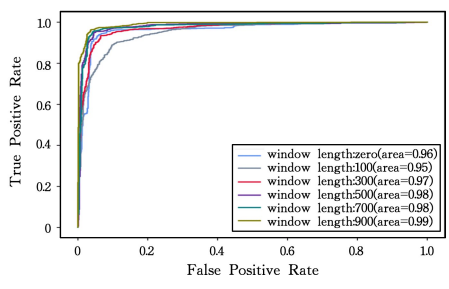

图 2 不同窗口长度下的模型性能

Fig. 2 Performance of G2ATT model with different window lengths

当窗口长度等于序列本身的长度，即不对软件的调用序列划分窗口时，模型会占用大量内存，训练时需要减小批处理的大小来保证训练的正常运行。但减小批处理的大小后，模型训练相同的数据集耗费的时间是使用本文预处理方法划分窗口的 14 倍以上，因此使用本文的预处理方法限制不同窗口长度对模型训练时间的影响很大。如果窗口过大，子序列长度小于窗口长度，会用零向量填充，此时会产生大量的冗余数据，并且网络中矩阵的维度过大也会增加计算量和训练的时间。

窗口子序列的重复率影响着计算难度和训练时间。图 3 分析了窗口长度为 300 时，在不同重复率的情况下模型的性能。重复率对模型性能有一定影响，但当重复率达到一定程度后，其对性能提升的收益减少。相对而言，序列重复率越大，步长越小，训练时间越长。

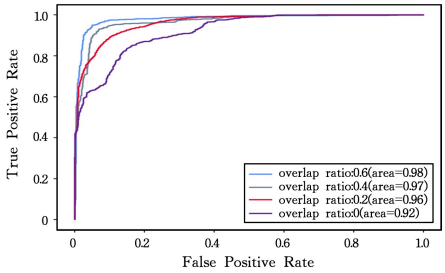

图 3　不同子序列重复率下的模型性能

Fig. 3　Performance of G2ATT model with different sequence overlap ratios

结束语　本文借鉴多示例学习和注意力机制的思想，提出了一种基于软件 API 调用序列的深度学习模型。在真实数据集上的实验结果表明，窗口特征和序列特征抽取层分别使用注意力机制学习窗口内和窗口间的重要性权重，能够更好地描绘序列的特征，本文模型的 *AUC* 达到 99%，准确率达到 98.19%。

本文提出的恶意软件检测方法 G2ATT 侧重于解决基于触发器的恶意软件，后续我们会将此方法在更大的数据集上进行实验验证。此外，现有方法抽取的恶意软件序列不包含某些需要执行特定操作才会触发恶意行为的软件，后续可以从分析环境入手来监控恶意软件的其他潜在恶意行为。本文方法也可以扩展到对恶意软件的分类问题上，通过学习不同类别的软件调用序列的特征，对潜在恶意软件进行分类。

参 考 文 献

[1] HU G, VENUGOPAL D. A malware signature extraction and detection method applied to mobile networks[C]// IEEE Internationl Conference on Performance, Computing, and Communications Conference, 2007(IPCCC 2007). IEEE, 2007: 19-26.

[2] ZHU P B. Research on malware detection using machine learning[D]. Beijing: Beijing University of Posts and Telecommani Cations, 2018. (in Chinese)
朱鹏博. 基于机器学习算法的恶意代码检测技术研究[D]. 北京：北京邮电大学，2018.

[3] WANG R, FENG D G, YANG Y, et al. Semantics-Based Malware Behavior Signature Extraction and Detection Method[J]. Journal of Software, 2012, 23(2): 378-393. (in Chinese)
王蕊，冯登国，杨轶，等. 基于语义的恶意代码行为特征提取及检测方法[J]. 软件学报，2012，23(2)：378-393.

[4] BAHDANAU D, CHO K, BENGIO Y. Neural machine translation by jointly learning to align and translate[J]. arXiv: 1409.0473, 2014.

[5] SAXE J, BERLIN K. Deep neural network based malware detection using two dimensional binary program features[C]// 2015 10th International Conference on Malicious and Unwanted Software (MALWARE). IEEE, 2015: 11-20.

[6] ARP D, SPREITZENBARTH M, HUBNER M, et al. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket[C]// Network and Distributed System Security Symposium. San Diego, CA, 2014, 14: 23-26.

[7] NATARAJ L, KARTHIKEYAN S, JACOB G, et al. Malware images: visualization and automatic classification[C]// Proceedings of the 8th International Symposium on Visualization for Cyber Security. ACM, 2011: 4.

[8] KOLOSNJAJI B, ZARRAS A, WEBSTER G, et al. Deep learning for classification of malware system call sequences[C]// Australasian Joint Conference on Artificial Intelligence. Cham: Springer, 2016: 137-149.

[9] XU J Y, SUNG A H, CHAVEZ P, et al. Polymorphic malicious executable scanner by API sequence analysis[C]// Fourth International Conference on Hybrid Intelligent Systems, 2004(HIS'04). IEEE, 2004: 378-383.

[10] TOBIYAMA S, YAMAGUCHI Y, SHIMADA H, et al. Malware detection with deep neural network using process behavior [C]// 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC). IEEE, 2016, 2: 577-582.

[11] ROSENBERG I, SHABTAI A, ROKACH L, et al. Generic Black-Box End-to-End Attack Against State of the Art API Call Based Malware Classifiers[C]// International Symposium on Research in Attacks, Intrusions, and Defenses. Cham: Springer, 2018: 490-510.

[12] ZHOU P, SHI W, TIAN J, et al. Attention-based bidirectional long short-term memory networks for relation classification [C]// Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. IEEE, 2016: 207-212

[13] LIN Y, SHEN S, LIU Z, et al. Neural relation extraction with selective attention over instances[C]// Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. IEEE, 2016: 2124-2133.

[14] CHO K, VAN MERRIENBOER B, GULCEHRE C, et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation[J]. arXiv: 1406.1078.

[15] DIEDERIK K, BA J. Adam: A method for stochastic optimization[J]. arXiv: 1412.6980, 2014.

[16] SRIVASTAVA N, HINTON G, KRIZHEVSKY A, et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting [J]. Journal of Machine Learning Research, 2014, 15(1): 1929-1958.

[17] PASCANU R, STOKES J W, SANOSSIAN H, et al. Malware classification with recurrent networks[C]// 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2015: 1916-1920.