

本科生毕业设计(论文)

基于强化学习的对抗性恶意程序生成方法研究

**Reinforcement Learning-Based Adversarial Malicious Sample Generation Method**

|  |  |
| --- | --- |
| 学 院： | 计算机学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 07112105 |
| 学生姓名： | 李函书 |
| 学 号： | 1120213139 |
| 指导教师： | 田东海 |

2025 年 月 日

原创性声明

本人郑重声明：所呈交的毕业设计（论文），是本人在指导老师的指导下独立进行研究所取得的成果。除文中已经注明引用的内容外，本文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。

特此申明。

本人签名： 日 期： 年 月 日

关于使用授权的声明

本人完全了解北京理工大学有关保管、使用毕业设计（论文）的规定，其中包括：①学校有权保管、并向有关部门送交本毕业设计（论文）的原件与复印件；②学校可以采用影印、缩印或其它复制手段复制并保存本毕业设计（论文）；③学校可允许本毕业设计（论文）被查阅或借阅；④学校可以学术交流为目的,复制赠送和交换本毕业设计（论文）；⑤学校可以公布本毕业设计（论文）的全部或部分内容。

本人签名： 日 期： 年 月 日

指导老师签名： 日 期： 年 月 日

基于强化学习的对抗性恶意程序生成方法研究

摘　要

近年来，随着勒索病毒和恶意PE可执行文件木马程序的流行，目前存在的一些反病毒软件的查杀引擎，例如360杀毒使用的基于机器学习的QVM引擎，对于恶意PE可执行文件的检测仍然有较多的漏报和误报问题，反病毒软件厂商为了解决漏报和误报问题，需要用户主动向反病毒软件的维护人员提交漏报样本和误报样本，但这无疑存在效率较低的问题。同时，部分反病毒软件过度依赖动态检测来对抗恶意软件，静态检测能力不足，难以做到在恶意软件刚出现但是没有被运行时隔离恶意软件。

为了提供静态对抗性恶意样本供反病毒软件训练，本文提出并实现了一种基于强化学习的Sarsa算法的静态对抗型样本生成框架，输入为PE可执行程序恶意样本集合，输出为能够规避部分反病毒软件静态检测的对抗性样本，其由三个模块组成，分别是样本筛选模块、恶意程序扫描模块、强化学习对抗性样本生成模块。样本筛选模块实现了对于收集到的样本的处理，删除不符合条件的样本，最终生成原始样本。恶意程序扫描模块实现了获取样本集处理前后的检出率，以及判定经过强化学习对抗性样本生成模块处理后的样本能否逃逸静态检测。强化学习对抗性样本生成模块由筛选后的样本和指定的行为集和参数生成能够逃逸静态检测的对抗性样本。在实验中，某个样本集经处理前后，Clam AV对其查杀率从31.3%下降到了20.7%，360杀毒对其的查杀率从99.1%下降到了4.2%。总而言之，使用本文中提出的方法生成的静态对抗样本集对反病毒软件进行训练，可以提高基于深度学习和机器学习的反病毒软件的对于恶意软件的检测能力，对恶意PE可执行文件检测的攻防对抗领域有一定的贡献。同时本模型不同于已存在的psp-mal模型，创新点在于使用了仍在更新的ClamAV反病毒软件，360杀毒等作为静态规避检测标准，而非已经过时的EMBER反病毒模型，保证了对于更流行的反病毒软件的规避。并且引入了基于文件大小变化的奖励修正，鼓励生成更小的对抗性样本。最后，操作集相比psp-mal的操作集新增了UPX加壳，图标资源增加，签名伪造等更多操作，增加了可迁移性。

关键词：强化学习；静态分析规避；PE恶意可执行文件

Abstract

In recent years, with the prevalence of ransomware and malicious Portable Executable programs, most antivirus engines—such as the QVM engine, which is based on machine learning and is used by 360 Anti-Virus—still exists tremendous false negatives and false positives in malicious PE executables detection. To solve these issues, antivirus software developers require users to actively submit samples which are malicious but not detected by antivirus software or are benign but software regard them as malware to their maintenance teams, which is inefficient. Additionally, some antivirus software excessively relies on dynamic detection to identify malware, which results in deficient static detection capabilities and has difficulty in quarantining newly emerged malware before it is executed by user.

To provide static adversarial samples for antivirus systems training whose purpose is detecting malware more accurately, this project raises and creates a static adversarial sample generation framework which is based on reinforcement learning Sarsa algorithm. The framework which consists of three modules: sample swifting module, malware scanning module, and reinforcement learning adversarial sample generation module, needs malicious PE executable samples as input, after processing, it outputs adversarial samples which can evade static detection of some antivirus software. The sample swifting module implements the processing of samples by removing samples which do not fit this experiment. The malware scanning module calculates detection rates of sample sets before and after processing, besides,it determinines whether samples which are processed by the reinforcement learning adversarial sample generation module can evade static detection. The reinforcement learning adversarial sample generation module generates adversarial samples which can evade static detection by using origin samples and specified behavior sets and parameters.The experiment shows the result that after processing a sample set, the detection rate by Clam AV dropped from 31.3% to 20.7%, while the detection rate of 360 Anti-Virus’s declined from 99.1% to 4.2%.Generally, training antivirus software with the static adversarial samples which are generated by the reinforcement model can enhance the malware detection capabilities of antivirus software and engines which are based on deep learning and machine learning. This experiment contributes to the defensive domain of malicious PE executable detection. Moreover, this model innovations make this model differ from the existing PSP-Mal framework.Firstly, it adopts actively updated antivirus engines such as ClamAV and 360 Anti-Virus as static evasion detection standard,which replaces the outdated EMBER antivirus model, ensures evasion effectiveness against prevalent anti-virus software. Secondly, a reward adjustment method which is modified by the change of file size is adopted to encourage the generation of smaller adversarial samples. Secondly, the operation set has been expanded,which is different from PSP-Mal's original configuration ,contains UPX packing, icon resource modification, and signature fabrication techniques, enhances the transferability.

Key Words: Reinforcement learning; Static analysis evasion; PE malicious executable

目　录

[摘　要 I](#_Toc198487968)

[Abstract 1](#_Toc198487969)

[第1章 引言 5](#_Toc198487970)

[1.1 研究背景和意义 5](#_Toc198487971)

[1.2 国内外对于对抗性样本生成研究现状 6](#_Toc198487972)

[1.3 创新点及主要工作贡献 7](#_Toc198487973)

[1.3.1 创新点 7](#_Toc198487974)

[1.3.2 主要工作贡献 7](#_Toc198487975)

[1.4 论文结构 8](#_Toc198487976)

[第2章 预备知识 9](#_Toc198487977)

[2.1 实验预备知识 9](#_Toc198487978)

[2.1.1 PE文件 9](#_Toc198487979)

[2.1.2 强化学习 10](#_Toc198487980)

[2.1.3 恶意软件种类 13](#_Toc198487981)

[2.2 本章小结 14](#_Toc198487982)

[第3章 系统模块的设计 15](#_Toc198487983)

[3.1 总体设计 15](#_Toc198487984)

[3.2 样本筛选模块的设计与实现 17](#_Toc198487985)

[3.2.1 设计思路 17](#_Toc198487986)

[3.2.2 具体实现 18](#_Toc198487987)

[3.3 恶意程序扫描模块设计与实现 20](#_Toc198487988)

[3.3.1 设计思路 20](#_Toc198487989)

[3.3.2 具体实现 21](#_Toc198487990)

[3.4 强化学习对抗性样本生成模块设计与实现 24](#_Toc198487991)

[3.4.1 设计思路 24](#_Toc198487992)

[3.4.2 具体实现 29](#_Toc198487993)

[3.5 本章小结 38](#_Toc198487994)

[第4章 实验设计与结果分析 39](#_Toc198487995)

[4.1 实验设计 39](#_Toc198487996)

[4.1.1 开发环境配置 39](#_Toc198487997)

[4.1.2 原始样本获取 39](#_Toc198487998)

[4.1.3 配置VMWare虚拟机 40](#_Toc198487999)

[4.1.4 结果评估标准 40](#_Toc198488000)

[4.2 实验结果 41](#_Toc198488001)

[4.2.1 实验结果的数据图示 42](#_Toc198488002)

[4.2.2 检出率及其变化的计算 45](#_Toc198488003)

[4.2.3 不同反病毒软件对于检出率变化的影响 49](#_Toc198488004)

[4.2.4 与psp-mal模型的对比 50](#_Toc198488005)

[4.2.5 静态对抗性样本对云查杀的影响 51](#_Toc198488006)

[4.2.6 静态对抗性样本对于动态检测技术的影响 56](#_Toc198488007)

[4.3 实验建议 57](#_Toc198488008)

[4.3.1 对于相关研究人员 57](#_Toc198488009)

[4.3.2 对于反病毒软件厂商 57](#_Toc198488010)

[4.4 本章小结 58](#_Toc198488011)

[结　论 60](#_Toc198488012)

[参考文献 61](#_Toc198488013)

[附　录 64](#_Toc198488014)

[致　谢 65](#_Toc198488015)

第1章 引言

1.1 研究背景和意义

恶意软件是经过精心设计，被用来攻击计算机系统或计算机网络并且造成损害的软件，危害不仅限于感染型病毒和蠕虫的自身复制耗尽系统资源，破坏操作系统、后门软件开放系统端口供黑客连接从而形成僵尸网络对服务器发起分布式拒绝服务攻击、木马伪装成正常程序，实则窃取用户的敏感数据和破坏系统，为黑客提供后门、勒索病毒加密文件要求用户支付赎金等。根据统计数据，仅2022年在全球大约发生了55亿次恶意软件攻击事件[1]。

在早期，恶意软件的代码较为简单，容易被反病毒软件检测到特征值从而处理清除。但是多年以来，恶意软件的复杂性不断发展，传统的基于黑名单哈希值的恶意软件检测技术难以应对当今恶意软件复杂的混淆策略，尽管这种方法速度很快，但是难以识别新一代的恶意软件以及一些0day恶意软件[2]。

目前，为了应对恶意软件带来的威胁，许多开源以及商业杀毒软件厂商不断升级病毒库，更替杀毒软件版本。目前针对恶意软件的识别主要分为静态分析、动态分析以及混合分析（同时结合了静态分析和动态分析）这三种。而本实验研究的强化学习模型，是生成针对反病毒软件的静态特征分析的PE可执行程序对抗性样本，目的是用于反病毒软件厂商的机器学习和深度学习反病毒引擎模型训练和逆向分析工作者们学习研究，本实验未研究动态特征分析对抗性样本生成。静态特征分析包括文件Hash（如MD5 SHA256）匹配，这在很多病毒样本分析网站，例如VirusTotal，VirusSCAN等被使用，如果上传的文件和病毒库里面存在的样本的Hash值匹配，则反病毒软件判断该文件是病毒。此外，静态分析还包括资源节分析，时间戳检查，数字签名检查，函数导入表检查，特征字符串匹配，DEBUG信息检查等。

在过去的十几年中，学术界出现了大量利用机器学习和强化学习的恶意软件查杀模型判断恶意软件的研究成果，甚至可以运用启发式杀毒规则，检测分析软件的代码行为，来判断从未出现过的新型恶意软件[3],[4],[5],[6],[7],[8]。

但不幸的是，某些基于静态特征分析的反病毒软件在某些情况下有着很高的误报率，可能因为机器学习模型自身的问题，误判一些正常的软件的行为，认为这些正常软件是恶意软件，例如QVM反病毒引擎误报Microsoft Visual Studio Complier、Clang等C/C++语言编译器，以及LLVM语法分析器等软件。此外，基于机器学习的反病毒软件也存在严重的漏洞，攻击者只需要对恶意软件进行修改，甚至有时只需要增加一个资源文件改变恶意软件的Hash值，就能绕过反病毒软件的检测，恶意软件的编写者的规避技术对于反病毒软件的识别带来了极大的挑战，这促使攻防对抗的双方不断采取更先进的措施。

1.2 国内外对于对抗性样本生成研究现状

在过往的PE对抗性样本生成实验中，许多已有的对抗性模型生成，有些采用值函数的强化学习算法，例如使用深度学习神经网络的DQN[9]。

有些使用遗传编程算法，基于适应度进行选择，交叉，编译[10]。

有些采用蒙特卡洛搜索树方法，将对抗样本转化为路径搜索问题[11]。

这些现有的方法已经展现了一定有效性，但时序差分算法，例如Sarsa或Q-learning的研究却很稀少，因此本实验的主要研究目标是使用时序差分算法Sarsa构建强化学习对抗性样本生成模型，来生成能够规避部分静态检测的PE可执行程序对抗样本，使部分样本逃逸反病毒软件的查杀。

基于强化学习的方法是通过构建动作（Action），状态（State），奖励（Reward），来使智能体与环境交互，通过多次操作获取经验，挑选一个相对更好的策略来修改已有的PE恶意程序来达到逃逸反病毒软件检测。

在2017年，基于强化学习生成静态对抗性样本绕过黑盒的反病毒软件的GYM-Malware框架被提出。Anderson等人定义了10种不会影响恶意软件和功能的扰动操作，例如利用签名漏洞来更改恶意软件的签名，修改恶意软件的调试（Debug）信息，修改可选首部检验码（checksum），修改现有的节的名称。这些操作与DQN强化学习算法结合和病毒检测软件交互以指导选择的扰动操作[9],[13]。

但遗憾的是，一部分使用强化学习模型生成的对抗性样本无法在虚拟机中正常执行，有一部分恶意软件经过某些操作后被破坏了。尽管某些操作修改的部分似乎与恶意软件的代码部分无关，但还是影响到了恶意程序的功能[8]。

但GYM-Malware模型仍然对后续的研究有着很大的作用，许多后续的研究基于其工作来进行。其中封装的一部分动作（Action），被拆成了函数用于其他的研究中，例如Mab-malware。

有一些研究[14]，关注了GYM-Malware中的一些造成恶意软件功能损坏无法正常执行的操作。Mab-Malware认为是Python的LIEF库导致的，并且对其进行修复，将使用LIEF库的一些对PE可执行程序的操作改为使用Python的Pefile库，以减少损坏的恶意程序数量。也有一些研究[15]，直接删除了可能导致恶意程序遭到破坏的操作，引入随机化操作来缩小动作空间，限制强化学习智能体可执行的动作次数，鼓励强化学习智能体寻找更优秀的Action集合。

而由Labaca-Castro等人开展的研究[16]，则考虑修正奖励函数，对奖励函数添加惩罚因子，为了进一步优化强化学习智能体的操作。对奖励函数的修正鼓励智能体尽可能用更少的步骤对恶意PE程序进行修改以逃逸反病毒软件的检测。

类似的，Gibert et al.等人使用空操作[17]，即插入大量NOP指令来修改恶意软件，表明使用插入无意义空操作的方法对于绕过MalConv等反病毒软件一样是有效的。

同时，目前也存在基于梯度的对抗性样本生成，例如FGSM、 Carlini和Wagner创建的C&W、以及deepfool模型[18],[19],[20]。

且存在一些研究能够在已知梯度信息的情况下，通过基于梯度的方法对恶意软件的字节或者外观表现形式进行修改来规避静态特征检测[21],[22],[23],[24],[25],[26]。

1.3 创新点及主要工作贡献

1.3.1 创新点

本实验采用的强化学习算法是基于时序查分算法的Sarsa算法。关注PE恶意程序的原因是Windows操作系统在个人电脑（PC），服务器等操作系统中占有72%左右的份额，因此超过70%的恶意软件将Windows操作系统作为攻击目标[27],[28]。相比已有的工作，本实验额外考虑了UPX加壳操作和sigthief造成的假数字签名，resourceHacker调用导致的图标类资源添加等行为，对静态检测的对抗性样本生成可能带来的影响。同时，使用了仍在更新的ClamAV反病毒软件，360杀毒等作为静态规避检测标准，而非已经过时的EMBER反病毒模型，保证了对于更流行的反病毒软件的规避。并且引入了基于文件大小变化的奖励修正，鼓励生成更小的对抗性样本。最后，生成的对抗性样本相比psp-mal，增加了可迁移性。

1.3.2 主要工作贡献

本文的主要贡献如下：

（1）研究了时序差分强化学习算法Sarsa对于对抗性样本的生成。

（2）能够生成高概率逃逸360杀毒云查杀的对抗性样本并且保证恶意程序的功能，生成的恶意程序样本即使开启了允许上传可疑文件，仍然需要经过多次扫描后上传分析才得以查杀。

（3）预测了未来可能出现的高危害性感染型病毒变种和结合了静态分析的挂马网站，以及部分木马下载器变种，并且建议反病毒软件厂商们加以防范，尽管动态分析技术仍然可以针对感染型病毒变种，但该类型病毒仍具有一定威胁性。这种感染型病毒变种会在感染新文件时对于新产生的病毒自带进行自动地静态检测规避处理，同时被感染的文件相较于传统的感染型病毒难以恢复。

（4）Action更加倾向于模块化，减少了模块之间的耦合程度，便于后续的研究人员从项目中直接提取函数，而无需修改大量内容，对于某些扰动函数只需要传入原恶意程序文件的绝对路径，和用户期望生成的文件的绝对路径，就能实现对抗型扰动操作。相比原有框架PSP-Mal[12]，本实验项目中的扰动函数与行为（Action）相互分离，并且额外加入日志系统，扰动函数自身不需要传递过多的参数。而原框架PSP-Mal对于恶意软件的操作相对混乱，难以单独从一个行为（Action中）抽象出修改函数，对恶意软件的扰动行为和智能体之间耦合度过高，导致修改较为困难。

1.4 论文结构

本文第一章为引言部分，说明了本课题的研究背景和意义，介绍恶意软件检测技术的研究现状，并且简要说明了工作的内容和创新点。第二章研究了本课题相关工作的PE文件结构及强化学习算法的相关理论性知识。第三章介绍实验的各个模块的功能的设计和实现。第四章是本文的实验部分以及运行环境配置的具体过程。第五章分析结果以及提出实验中预测的未来潜在威胁恶意软件以及对于相关方向人员的一些建议。在结论中总结了本文的成果和工作的创新点，提出了工作中存在的问题以及未来的可选修改方向。

第2章 预备知识

2.1 实验预备知识

2.1.1 PE文件

PE文件是Windows操作系统下的可执行文件形式，包括EXE（可执行文件），DLL（动态链接库），SYS（系统文件）等类型。

PE文件中包含PE文件头（IMAGE\_NT\_HEADERS），其中的COFF文件头（IMAGE\_FILE\_HEADER）和可选首部（IMAGE\_OPTIONAL\_HEADER）中的有些部分是我们需要修改的目标。

COFF文件头（IMAGE\_FILE\_HEADER）包含如下关键字段：

（1）Machine：目标CPU架构，指明了能运行这个程序的机器码，可以指明支持程序运行的机器架构是x86、x64、PowerPC、ARM等。

（2）NumberOfSections：指明所有节区的数量。

（3）TimeDateStamp：时间戳，指明了这个文件被编译生成的时间。

（4）SizeOfOptionalHeader：可选首部的大小。

（5）Characteristics：文件的类型，是动态链接库，还是可执行文件等类型。

可选首部（IMAGE\_OPTIONAL\_HEADER）包含如下字段，可选首部指明了当PE程序被载入内存后的一些情况：

（1）Magic：魔法位，包含PE信息，一定要和COFF文件头中的Machine对应，否则就会报错导致程序无法启动。

（2）AddressOfEntryPoint：程序入口点，即相对虚拟地址。

（3）ImageBase：加载机制。

（4）SectionAlignment：内存中节区的对齐粒度，不建议修改，否则程序可能无法启动。

（5）FileAlignment：文件中节区的对齐粒度，不建议修改，否则程序将无法启动。

（6）SizeOfImage：加载到内存后的总大小。

（7）SubSystem：子系统类型。

（8）DataDirectory：数据目录表，记录某些数据的位置及其大小。

PE文件中的节表（Section table）描述了每个节的属性，由多个IMAGE\_SECTION\_HEADER组成，每个条目对应一个节区，包括下列关键属性：

（1）Name：节区的名字。

（2）VirtualAddress：虚拟地址中的起始相对位置。

（3）SizeOfRawData：节区中数据的大小。

（4）PointerToRawData：节区偏移量。

（5）Characteristics：节区属性。

节区数据正常情况下应包含如下内容：

（1）.text：代码节，存放该PE程序执行的指令。

（2）.data：已经完成初始化的某些数据。

（3）.rdata：只读（Read Only）的数据。

（4）.rsrc：资源节，存放PE文件的图标等信息。

（5）.reloc：重定位信息，可以用于动态链接库的装载过程。

（6）.idata：import data，即导入函数信息。

数据目录表中包含的内容如下：

（1）Import Table：导入表，用于存放该PE文件依赖的动态链接库和调用的某些函数。

（2）Export Table：导出表，可以用于存放这个PE文件封装好的函数，多半运用于动态链接库封装函数供其余PE程序调用使用。

（3）Relocation Table：重定位表，修复地址偏移相关问题。

（4）TLS：线程存储表，与多线程程序有关，存储线程初始化数据。

（5）Debug Directory：存放该PE程序的调试信息。

2.1.2 强化学习

基本概念相关：

（1）智能体：在强化学习中决策，行动，学习。智能体是一个感知者，能感知并且理解当前的状态，智能体是一个决策者，能够知道在一个状态下应该采取什么行动，智能体是一个执行者，通过改变状态从而获取奖励。

（2）状态：描述了智能体与环境的相对状况。

（3）状态空间：所有状态的集合。

（4）动作：智能体在某一状态下能选择的操作。

（5）动作空间：所有动作的集合。

（6）状态转移：当执行一个动作时，智能体可能从一个状态转移到另一个状态的过程。

（7）策略：智能体在每一个状态下应该采取什么样的动作，允许分为确定性策略和随机性策略。

（8）奖励：作为人机交互的一个重要手段，可以设置合适的奖励来引导智能体按照我们的预期选择正确的决策，正数奖励表明我们鼓励智能体执行该行动，负数奖励表明我们不鼓励智能体执行该行动。

（9）回合/尝试：智能体执行一个策略与环境交互的过程中，智能体从开始状态到终止状态停止的过程被称为一个回合或尝试，一般用英文episode来表示。

（10）折扣因子：用于调整智能体对于近期奖励和远期奖励的重视程度，可以记作折扣因子为γ，γ在(0,1)的范围，且折扣因子的引入允许了无限长的轨迹。

（11）状态值：表达式如式（2-1）所示：

(2-1)

状态值说明智能体在一个状态之下，最终能获取到的回报。首先，需要了解基于时序差分策略的方法可被用于估计状态值。

时序差分方法的表达式若式（2-2）所示：

(2-2)

代表t时刻对于的估计，是t状态下对于状态的学习率。

在t时刻，只有当时正在被访问的状态的估计值会被更新。

在式（2-2）中：

（1）作为时序差分方法的目标。

（2）作为时序差分方法的误差。

（3）为在t时刻对于状态的学习率。

（4）为新的t+1时刻对于状态值估计值。

（5）为t时刻对于状态的状态值的估计值。

Sarsa（state action reward action state action）是基于时序查分方法的强化学习算法，但是该方法不是估计状态值而是估计动作值[29],[30]。

首先要引入动作值的概念：

对于一个状态-动作配对，动作值定义表达式如式（2-3）所示。

= (2-3)

动作值表示在一个状态采取一个动作之后获得回报的期望值。

这里将的估计值记作。

给定一个策略π，需要估计其动作值，可以从π的经验样本中，使用Sarsa算法来估计动作值，其表达式为式（2-4）：

(2-4)

式（2-4）中，学习率为

在t时刻，只有的动作值被更新，其它的动作值保持不变。

Sarsa算法主要是用于求解Bellman方程近似算法，近似算法的表达式如式（2-5）所示：

(2-5)

式（2-5）是一个基于动作值而不是状态值的Bellman方程。

图2-1中的伪代码描述了如何运用Sarsa算法学习最优策略的执行逻辑。

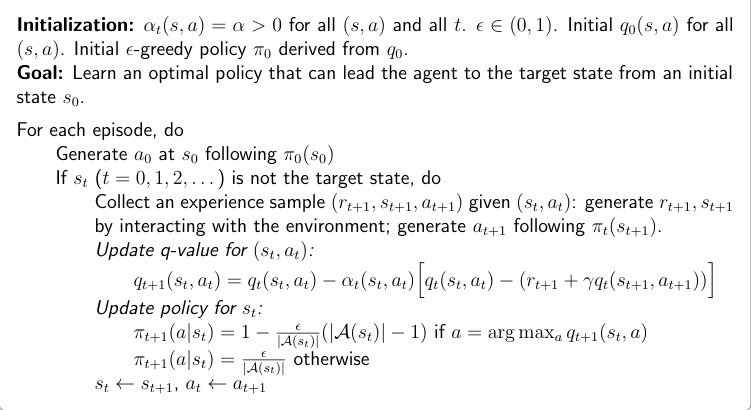


图2-1 Sarsa

目标：学习最优策略，使智能体能从给定状态到达目标状态。

对于图2-1中的Sarsa算法描述如下所示：

算法初始化：

对于所有数值和所有t数值，选取学习率因子 *= ，*贪婪因子，并且设定所有的初始值为，从获取初始贪婪策略。

算法目标：学习最优策略，使智能体能从给定的状态出发，到达目标终止状态。

对于每个回合：

如果当前状态是根据选取策略 a =

在时刻t，如果不是目标状态

收集经验样本(, , , )：在通过与环境交互生成,。

再根据生成

随后更新(, 的值

Sarsa算法也存在一些推广，例如Expected Sarsa算法、n-step Sarsa算法等，尽管本实验中并未涉及到这些推广算法，但是读者在构建强化学习对抗性样本生成模型上也可以尝试使用它们。

其中Expected Sarsa算法类似Sarsa算法，但是它们的时序差分方法目标上不同。具体实现流程上二者相似。

已知策略π，则Expected Sarsa算法对于该策略π的动作值估计公式如式（2-6）所示：

(2-6)

在式（2-6）中：

(2-7)

式（2-7）是在策略下的期望值。

不同于Sarsa算法的目标是求出，Expected Sarsa算法的目标是求出。

引入期望值会增加计算复杂度，但是会减小所求目标的方差，Expected Sarsa的涉及的随机变量是{}，相比Sarsa算法的{}，减少了一项，因此对于减少估计方差是有正面作用的[30],[31],[32]。

2.1.3 恶意软件种类

（1）病毒

计算机病毒是具有自我复制性的一段计算机命令或代码段，可以将自身插入到其他程序中，进而感染其他程序实行自我复制，这可能会造成用户的数据丢失以及计算机系统功能受损，被病毒感染的文件可以通过U盘，移动光盘等外部设备进行传播。计算机病毒大多数需要依附于受感染的程序被用户运行来传播，同时，计算机病毒具有高隐蔽性。

（2）木马

特洛伊木马是伪装成恶意程序的正常应用程序，实际上特洛伊木马被黑客植入了恶意功能。木马能够允许远程黑客对计算机实现入侵，窃取本机的重要数据和信息，同时，木马创立的远程连接也能够供黑客组建僵尸网络，发动DDOS攻击等行为。不同于病毒，特洛伊木马不会复制自身，但目前存在木马下载器变种，当木马下载器被运行时，可能会下载其他的病毒和木马程序到受感染的机器。

（3）后门程序

后门程序允许黑客绕过安全监测对计算机进行攻击，具有高隐蔽性。它的来源有多种。一部分是因为软件开发人员的失误问题，在软件的开发工程中，可能开发人员为了软件便于测试，会主动设计一些后门，例如测试使用的超级用户，然而，在正式版软件被发布前，这些预设的后门理论上应该被开发人员主动删除以避免软件存在的安全问题，但很可能软件开发人员会有遗漏，而这些被遗漏的后门很有可能被黑客利用向计算机发动攻击。另一部分后门是黑客在对某机器成功实现攻击后，为了以后能利用这台主机作为跳板对其他机器实施新的攻击，黑客会在被攻击的机器上主动安装后门程序。

2.2 本章小结

本章对论文研究的相关技术理论予以介绍，首先介绍了实验的目标-PE可执行文件的结构，之后介绍了强化学习的基本知识和基于时序查分方法的Sarsa算法，最后介绍了恶意软件的种类知识，包括病毒，特洛伊木马，后门程序等。

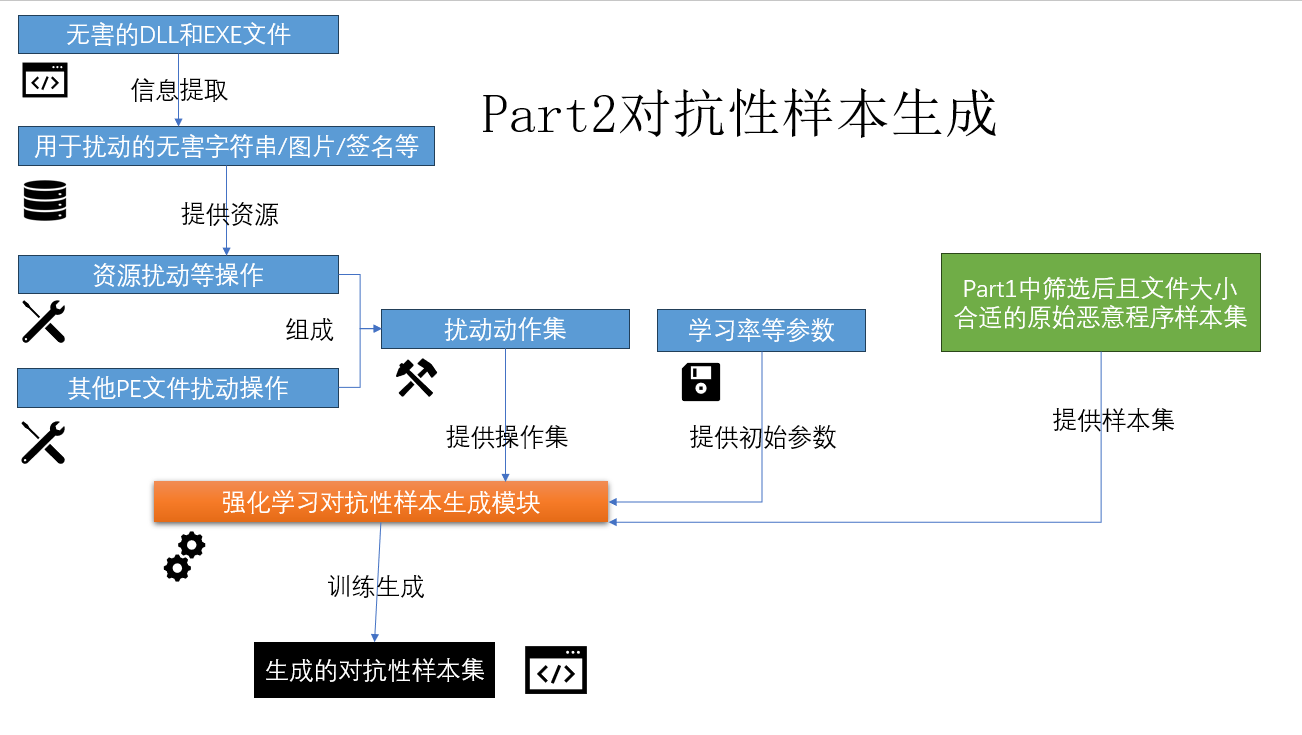
第3章 系统模块的设计

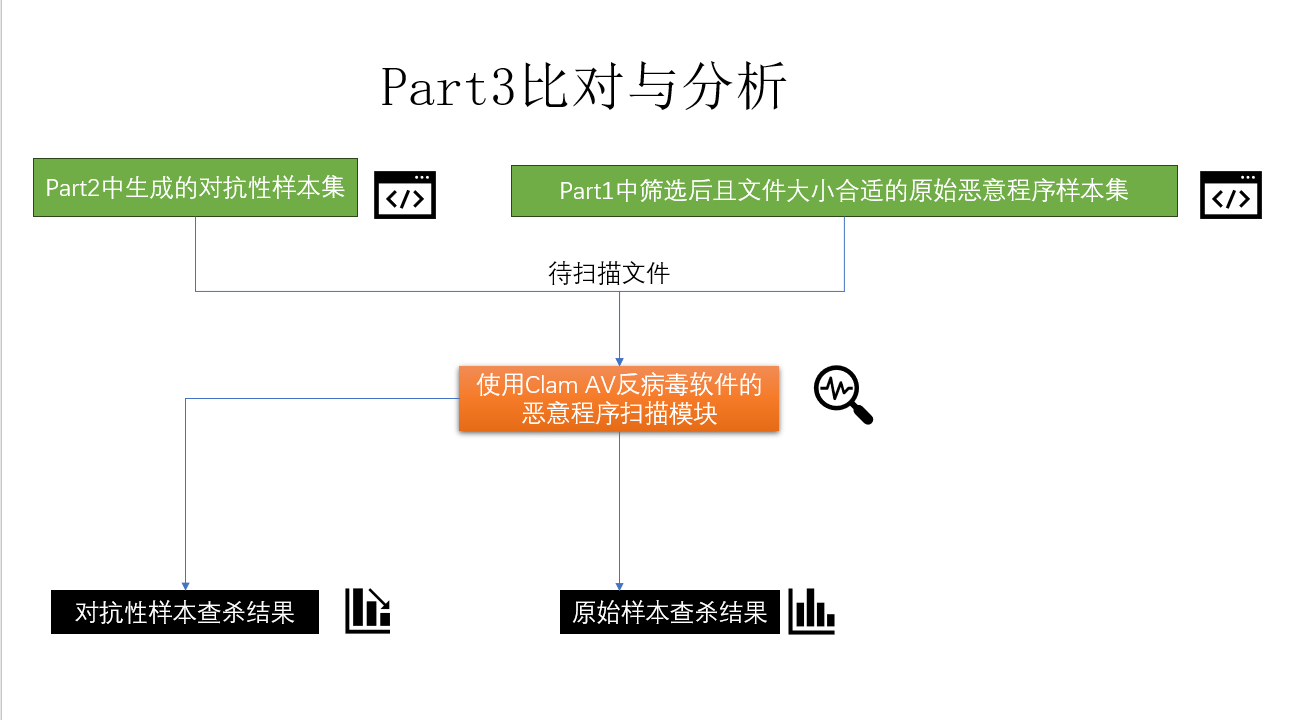
3.1 总体设计

本文涉及了一种基于强化学习Sarsa算法的静态对抗性样本生成系统，主要包含如下的几个模块：样本筛选模块，恶意程序扫描模块，强化学习静态对抗性样本生成模块。

其中，样本筛选模块的功能是，对于VirusShare网站上收集到的样本，其后缀名加上.exe，因为下载VirusShare原始样本时，该网站为了避免危害用户计算机，会默认去除恶意软件的后缀名。对于国内反病毒软件论坛上收集到的样本，需要去除非exe后缀的文件或文件大小较大的文件。强化学习对抗性样本生成模块只允许输入经由样本筛选模块后得到的筛选后样本。恶意程序扫描模块的功能是，使用ClamAV对收集到的原始恶意样本进行扫描，得到原始恶意样本检出率，以及对处理后的恶意样本进行扫描，得到处理后恶意样本检出率，以及供强化学习对抗性样本生成模块判定处理后的样本能否逃逸静态检测。强化学习对抗性样本生成模块是使用不同的action处理原始样本以生成对抗性样本，最终输出对抗性样本以及Qtable。总体设计图如图3-1，3-2，3-3所示，其中图3-1表明样本筛选模块在样本筛选部分（Part1）起到作用，最终获取到筛选后的恶意程序原始样本集，并保留其以备后续操作使用。图3-2表明强化学习对抗性样本生成模块在对抗性样本生成部分（Part2）起到作用，其所需的操作集中包含资源扰动操作和其它PE文件扰动操作，资源扰动操作的实现则需要从良性文件中提取资源。强化学习模型的初始参数由用户设置，待处理的样本集来源是样本筛选模块生成的筛选后的恶意程序原始样本集，强化学习对抗性样本生成模块会训练生成对抗性样本集以备后续操作使用。图3-3表明恶意程序扫描模块在比对与分析部分（Part3）中起到作用，它负责检验与分析生成的对抗性样本集与原始样本集的区别，通过扫描提供的样本集得到查杀结果。

图3-1 系统总体设计-1

图3-2 系统总体设计-2

图3-3 系统总体设计-3

3.2 样本筛选模块的设计与实现

3.2.1 设计思路

对于已收集到的样本（具体收集过程将在后文的实验中具体说明），需要注意的是，有些样本可能不是PE文件， 例如来源于火绒论坛上的一个样本包里面包含了恶意Windows Power Shell脚本和HTML恶意应用程序，恶意JavaScript脚本，恶意Visual Basic Script等非PE恶意程序。同时，因为本实验并没有对恶意动态链接库（DLL）和系统文件（SYS）进行对抗性样本生成，只是对恶意PE可执行文件（EXE）进行了对抗性样本生成，这是因为反病毒软件厂商提供的样本不包含动态链接库和系统文件，且检验动态链接库在对抗性样本生成后是否因为对抗性扰动导致损坏，相比检验PE可执行文件是否损坏更困难。所以，基于上述原因对于样本的筛选处理，理论上需要删除掉这些非PE可执行程序文件。

本实验设计样本筛选模块的原因是为了在实验中尽量避免有较大操作隐患的手动筛选，尽管手动筛选看起来很容易，但手动筛选很容易误操作，表现为用户可能意外执行恶意PE可执行程序导致可能遭受到恶意软件攻击。

筛选模块的输入应该是一个文件夹，其中包含待筛选的原始样本。对于来源于Virus Share的样本需要指定增加.exe后缀后的样本的输出位置，对于文件类型和大小的筛选可以直接使用Python的os模块删除不符合条件的样本文件，只保留实验需要用到的样本文件。

每次处理的文件夹其中建议包含400~600个未筛选的恶意软件，收集好每次处理后的样本，并将这些样本归为一个样本集，为的是以备实验中后续使用。

然而，这种筛选方式在后续实验中可能会存在一些问题，具体将在后文中予以详细解释。

3.2.2 具体实现

样本筛选模块的具体实现思路如图3-4。图3-4是实现对于来源于Virus Share的样本进行筛选，对传入的绝对路径下的所有文件添加.exe文件扩展名，并且忽略传入的绝对路径下的文件夹，将添加扩展名后的样本与指定的输出路径拼接成输出文件名，之后进行拷贝操作，将原样本文件内容复制到新文件中，进而生成筛选后的样本。实现对于来源于国内安全软件论坛的样本筛选和VirusShare处理后的样本的二次筛选是通过对于传入的绝对路径，忽略路径下的文件夹，只考虑所有文件，会删除非PE可执行文件，即文件扩展名不为.exe的文件和大于4096字节大小的文件，最终保留下来的文件为符合条件的样本。

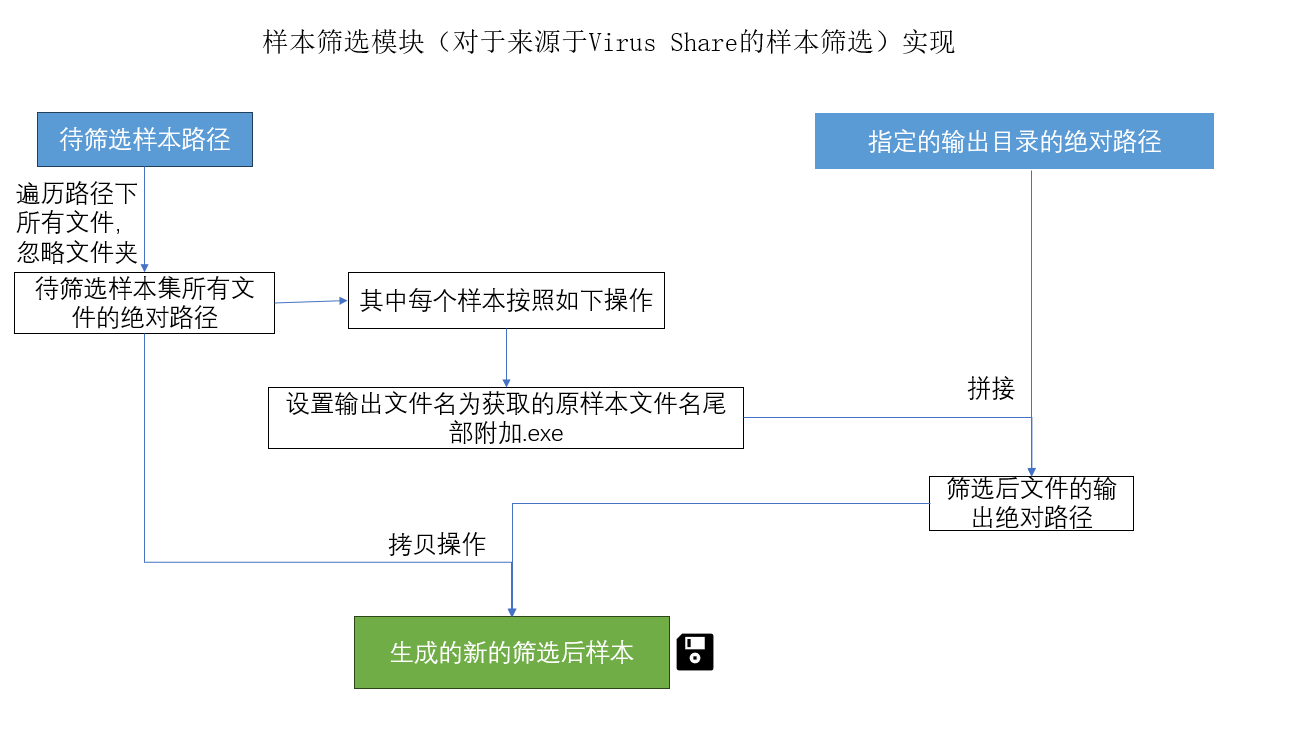


图3-4 样本筛选模块

实验中的已设计好的样本筛选模块如图3-5，图3-6，图3-5中对应的代码实现了对于来源于VirusShare的样本筛选，图3-6中对应的代码实现了对于来源于VirusShare的样本二次筛选以及对于来源于国内安全软件论坛的样本筛选：



图3-5样本筛选模块配置1

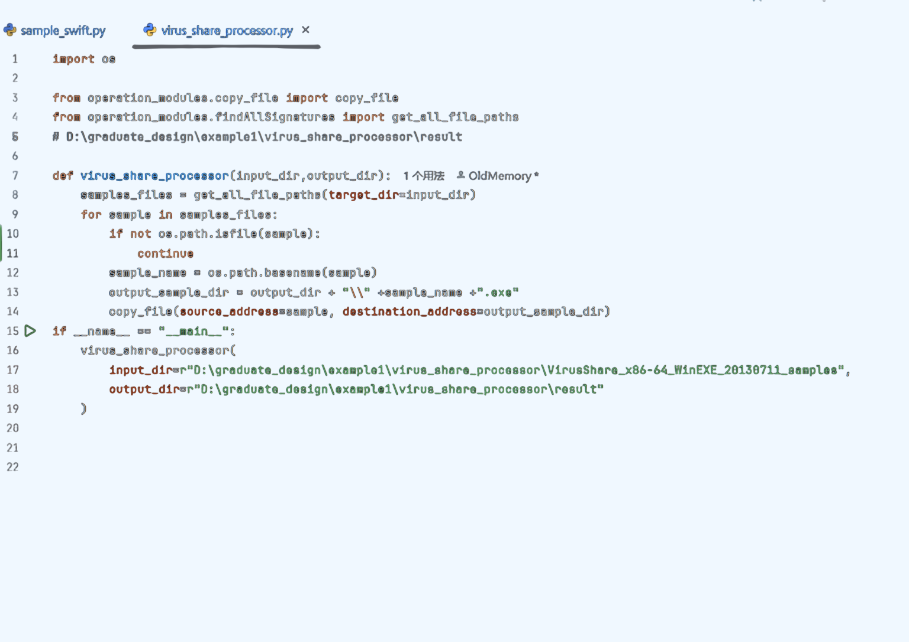


图3-6 样本筛选模块配置2

为了考虑样本筛选模块的完备性，需要配备异常处理应对可能出现的一些异常情况：

1. 传入的样本文件夹中包含子文件夹，筛选中需要忽略掉这些子文件夹。
2. 传入的样本无法被访问，此时应该抛出异常警告。

3.3 恶意程序扫描模块设计与实现

3.3.1 设计思路

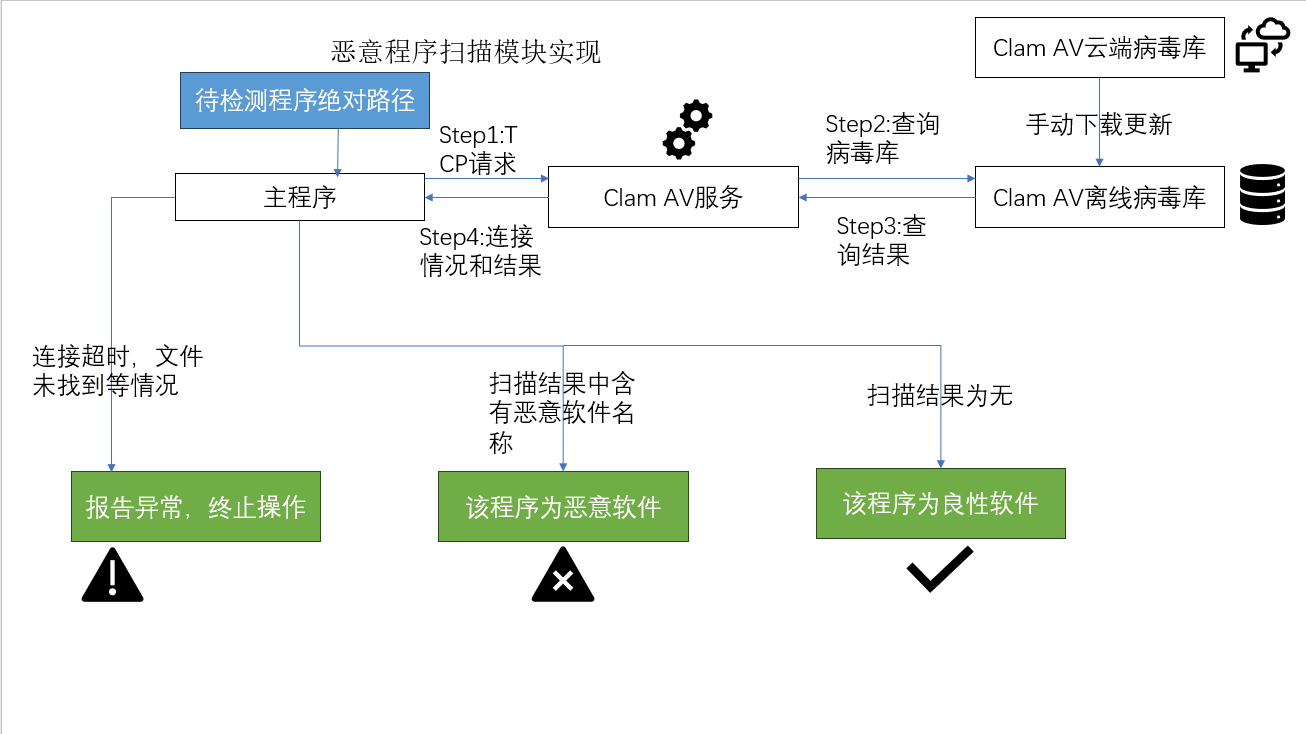
因为实验中需要判断样本的恶意性，所以需要搭建一个恶意软件检测器。在本实验中，选用了Python的pyclamed库的0.4.0版本搭建了一个基于ClamAV的恶意软件检测器。

恶意软件扫描模块的输入为待检测的PE可执行文件，输出结果为bool型，为true时，说明输入的PE可执行程序是恶意程序，为false时说明输入的PE可执行程序是良性程序。注意的是，检测器不应对输入的待检测的PE可执行文件做出任何修改。

然而，实验中恶意软件扫描模块可能会出现一些意想之外的情况，这将在后文会得到详细说明。

3.3.2 具体实现

恶意程序扫描模块的具体实现思路如图3-7，需要传入待检测程序的绝对路径。图3-7对应的过程为：在模块运行前，用户通过命令手动更新本地ClamAV病毒库使其同步云端病毒库。模块运行时，主程序向主机的ClamAV服务发送TCP请求，如果这一步能成功，则ClamAV服务会查询其本地病毒库，如果这一步不成功，出现了TCP连接的超时问题，则会返回TCP连接错误，提示用户需要修复ClamAV服务。随后，ClamAV服务会依赖于本地病毒库对待检测程序进行扫描，再将扫描的结果通过TCP连接传递给主程序，主程序根据ClamAV服务的返回结果判断待检测程序是否是恶意软件。

图3-7恶意程序扫描模块

为完成恶意程序扫描模块的视线，我们需要在主机配置ClamAV服务环境，首先需要在主机上安装ClamAV，并且修改安装目录下的clamd.conf文件，指定ClamAV服务的端口号。

为指定ClamAV服务的端口号，需要修改的是clamd.conf文件中TCP Socket的设置，因为是在本地配置ClamAV服务，clamd.conf中的TCPAddr，即TCP地址按照默认的localhost即可，无需修改。这一步需要注意端口号的冲突问题，最好事先检测主机已经使用的端口号，否则会影响恶意程序扫描模块的正常运行。

随后，需要修改freshclam.conf，这是因为ClamAV是基于本地病毒库来进行病毒扫描，需要配置病毒库地址和日志文件。

为了配置病毒库地址和日志文件，freshclam.conf中需要修改的项目是DataBaseDirectory和UpdateLogFile，分别对应ClamAV安装目录下的数据库目录和日志文件目录。

随后，打开cmd手动更新ClamAV病毒库，通过命令freshclam –config-file=./freshclam.conf实现对于ClamAV病毒库的更新操作，随后，ClamAV会从云端上下载病毒库。

更新数据库成功后。需要在Windows服务中重启ClamAV服务。这需要在服务的服务列表中选中ClamAV的ClamD服务左侧点击重启动该服务，或选中该服务右键，也可以达到重启该服务的目的。随后使用pyclamed库搭建恶意程序扫描模块，如图3-8所示，它的作用是检测传入的绝对路径指代的文件是否为恶意软件。需要输入待检测文件的绝对路径，随后会判断ClamAV服务是否处于活动状态，若不处于，则会抛出异常。随后执行文件扫描操作，判断文件是否为恶意软件，这是通过检测scanresult中的内容实现的。



图3-8恶意软件扫描模块配置

模块功能需要考虑模块运行的完备性，在恶意软件扫描模块中需要处理一些可能存在的异常情况：

1. 需判断在特定端口号下运行的ClamAV服务是否处于活动状态，如果不处于活动状态，就抛出一个网络连接异常。如果没有考虑这个可能存在的异常，将会使样本检测结果不准确。这个问题多半发生于用户主机使用了网络代理软件，网络代理软件按照自身规则会将主机对于本机环回地址127.0.0.1下对于Clam AV服务端口的访问分配到其他主机，将会造成模块无法访问到主机ClamAV服务，进而无法正常工作。
2. 需要注意参数file\_path指定的绝对路径对应的文件是否存在，如果文件不存在，也需要抛出异常。当文件存在时，才允许调用scan\_file接口对该绝对路径下的文件进行扫描。
3. 具体实现中，ClamAV返回的扫描结果是字符串数组，为空说明传入的文件是良性文件，若传入的文件为恶意软件，则可在返回扫描结果中获取具体的恶意软件名称，但是具体的恶意软件名称对于本实验没有作用，不需要传回它们，只需要检测字符串数组是否为空即可判断文件是良性文件还是恶意软件。

3.4 强化学习对抗性样本生成模块设计与实现

3.4.1 设计思路

首先，我们需要设计强化学习模型可能对PE可执行程序修改采取的action，应包括如下内容：

（1）使用sigthief，将一个从良性软件中提取出的数字签名附加在恶意软件上，通过修改ImportTable中的CertTableLOC，CertLOC，CertSize等内容实现，操作后的结果如图3-9所示，可以发现恶意程序被附加了一个可信任的CA证书，该证书由Xing Wang发行，从Xilinx的软件安装包上剥落而来。

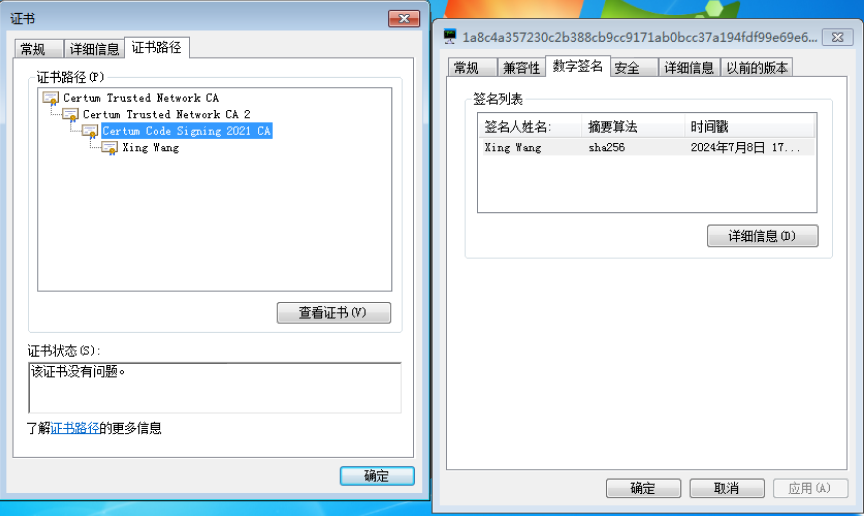


图3-9 签名伪造

（2）在PE程序的尾部添加无意义的字节:

添加的字节使用随机生成的字符串或一些良性动态链接库的.text段。

（3）修改校验和（checksum）:

需要指定修改后的checksum内容，将可选首部的Checksum字段修改为指定后的内容

（4）修改机器码:

修改PE可执行文件的Machine和Magic内容。但如果一旦修改后的机器码和软件运行平台的机器码不同，则程序会无法启动。然而，这会引起恶意软件损坏的问题，在后续被弃用。

（5）修改时间戳：

此功能本实验用pefile库和lief库分别实现了该功能，函数会随机生成一个过去一年之内的时间对应的时间戳，并且将生成的时间戳替换掉恶意软件原有的时间戳。

（6）UPX加壳操作：

使用The Ultimate Packer for executables加壳工具，是GitHub网站上的一个开源壳项目。读者可以访问<https://github.com/upx/upx>获取加壳工具。

同时，它也能做到压缩原PE可执行程序的大小。

在运行UPX加壳操作时需要注意UPX加壳程序的路径，防止因为路径问题导致抛出异常加壳失败。

（7）向节之间的空洞中添加无意义的字节：

添加的字节使用随机生成的字符串或一些良性动态链接库的.text段。

同时，也需要一个search\_cave()函数用于寻找节之间的空洞。

（8）修改导入表，增加PE可执行程序导入的函数：

从small\_dll\_imports.json中选取导入的函数来修改导入表，会添加引用的动态链接库或函数。

（9）在PE可执行文件后添加一个节，节中包含无意义的字符串数据：

此操作使用LIEF库实现，添加的字节使用随机生成的字符串或一些良性动态链接库的.text段。

（10）修改PE可执行程序首部校验和：

将PE可执行程序的首部校验和修改成0。

（11）删除PE可执行程序的调试信息：

可能因为没有debug信息，导致操作无意义，但不会抛出异常。

（12）重命名原PE可执行程序中的某个节，备选的重命名名称会在

.abitcsa .bbitcsb .cbitcsc .dbitcsd .ebitcse .fbitscf .gbitcsg .hbitcsh .ibitcsi .jbitcsj .kbitcsk中随机选择一个。

（13）向原PE可执行程序中增加资源：

增加的资源是多个ICO文件，是从备选的ICO文件中随机选取，然后截取随机大小，再调用resourcehacker将处理后的ICO文件加入到资源中。

这些action的传入参数应包含待修改的可执行文件，经过对抗性操作修改后生成对抗性样本，随后，这些对抗性样本需要被移动到原路径以完成修改。

对于对抗性样本生成的强化学习算法设计如下，其包含三个算法模块，主算法实现对抗性样本生成的功能，子算法2和子算法3实现训练，总奖励更新的功能。主算法用于强化学习对抗性样本生成模块运行。

主算法：

功能：对抗性样本生成函数

输入: 样本集目录sample\_directory,强化学习模型Sarsa\_model

输出: 生成的对抗性样本

算法伪代码：

1. **def** alogrithm\_generate\_controversy\_samples(sample\_directory,actions\_list,Sarsa\_model):
2. Sarsa\_model.init(actions\_list,gamma,alpha,epsilon) #init
3. evasion\_count = train(Sarsa\_model,sample\_directory)

主算法实现了对于强化学习模型的初始化和样本训练结束后统计成功逃逸的样本数量。

子算法1：

功能：完成主算法中的训练操作（train）

输入：样本集目录sample\_directory,强化学习模型Sarsa\_model

输出：成功逃逸的对抗性样本数量

算法伪代码：

1. **def** train(Sarsa\_model,sample\_directory):
2. episodes = get\_episodes\_for\_training(sample\_directory)   #get samples
3. evasion\_count = 0
4. **for** episode **in** episodes:
5. count = 0   #init
6. state = 0
7. total\_reward = 0
8. status\_done = False
9. reward = 0
10. action = choose\_action(state)   #choose action
11. do\_action(action,episode)
12. **while** **not** status\_done:
13. #total reward update
14. total\_reward=update\_total\_reward(episode,total\_reward)
15. **if** status\_done: #not final state
16. Sarsa\_model.update(state,action,reward,next\_state=None,next\_action=None)
17. **else**:   # final state
18. next\_action = choose\_action(next\_state)
19. Sarsa\_model.update(state,action,reward,next\_state,next\_action)
20. action = next\_action
21. do\_action(action,state,enable\_log)
22. **return** evasion\_count

子算法1实现了对于样本集中的每个样本，根据当前状态在行为表中选择操作并且执行，若当前状态为终止态，则更新强化学习模型并对下一个样本进行操作，否则继续选择操作并且予以执行。

子算法2：

功能：完成子算法1中的对total\_reward的更新操作

输入: 当前的样本episode,更新前的总奖励total\_reward

输出: 更新后的总奖励total\_reward

算法伪代码：

1. **def** update\_total\_reward(episode,total\_reward):
2. judgement = can\_evasion(episode)
3. **if** judgement == True :    # can evade detect
4. evasion\_count++
5. next\_state = None  # reach final state
6. reward = SUCCESS\_REWARD
7. status\_done = True
8. **else**:
9. count++
10. **if** count >= Max\_Count:  # operations amount run out
11. reward = FAIL\_REWARD
12. next\_state = None
13. status\_done = True   # reach final state
14. **else**:  # operations amount do not run out
15. next\_state = count
16. total\_reward+=reward   # update total reward
17. **return** total\_reward

子算法2实现了对于总奖励和状态的更新，当能成功逃逸恶意软件扫描模块的检测时，进入终止态并且获得奖励，若不能逃逸，进行判断，如果次数已经达到最大次数，则进入终止态并且获取奖励，否则更新状态。最后返回总奖励。

强化学习算法的总工作流程如下描述：

Step1：传入恶意样本集的目录和其它参数，先将恶意样本的绝对目录存储，以备后续Sarsa智能体模型使用，同时预加载Sarsa模型（Sarsa\_model）。

Step2：对于样本集中的每个样本episode，先初始化奖励。

Step3：根据状态当前选择一个行为，episode经由处理后，经由前文中封装的恶意程序扫描模块，检测是否成功逃逸反病毒软件的扫描。

Step4：如果成功，则直接进入终止态，获取奖励SUCCESS\_REWARD并且更新Sarsa模型的Qtable，否则进入下一个状态。

Step5：如果失败，但是次数没有达到Max\_Count，则更新Sarsa模型的Qtable，并且重复Step3的过程。如果失败且次数达到了Max\_Count，进入终止状态，并获取奖励FAIL\_REWARD（可以设置为负数，也可以设置为0）和更新Sarsa模型的Qtable。

这个算法在后续实验中也允许被略微修改，用于实现对强化学习模型采取某些行为的鼓励，例如可以修改为执行添加新的节的行为在完成后会有较小的奖励以鼓励智能体更多执行该行为。

3.4.2 具体实现

行为表中的部分action的具体实现思路如图3-10，图3-11，图3-12，图3-13，图3-14，图3-15，图3-16，图3-17所示。

图3-10描述了SigThief获取良性程序签名的过程，需要预备一些良性应用程序，首先SigTheif会检测与数字签名相关的ImportTable中的CertTableLoc，CertSize条目，如果不存在，则SigThief认为该良性文件不能获取到签名，直接跳过改文件，如果存在，则在CertLoc位置截取CertSize大小的字符，即为数字签名对应的字符串，并将其保存至一个.exe\_sig扩展名的文件，作为从良性文件上剥落的签名，以备后续为恶意软件添加数字签名。

图3-11描述了SigThief将从良性文件上剥落的签名（格式为.exe\_sig）附加在恶意软件上的过程，需要前一步生成的.exe\_sig文件作为签名内容，具体过程为修改输入的恶意软件中的CertTableLoc等内容，并将输入中剥落下的良性签名的内容追加在恶意软件末尾，随后将修改后的恶意软件输出至指定的绝对路径下。

图3-12描述了良性字符串的获取过程，需要预备一些良性文件（可以是PE可执行程序、动态链接库、系统文件）,此过程首先会判断良性文件的内容，如果不含.text节（实际上也可以指定为.data节或其他节），则直接跳过该良性文件，如果含有，则将.text节中全部内容保存在一个字符串中，以备后续过程使用。

图3-13描述了恶意软件尾部追加无意义字节的具体过程，首先会将待处理的恶意程序读取为二进制字符串，随后从获取到的良性字符串中随机选取一个，使用字符串拼接操作加在待处理的恶意程序二进制字符串尾部，最后，将拼接后的二进制字符串写入指定的目录下完成对恶意软件进行无意义字节的追加操作。

图3-14描述了恶意软件修改校验和的具体过程，首先会将待处理的恶意程序读取为二进制字符串，随后会修改可选首部的CheckSum字段为目标校验和，最后，将修改后的二进制字符串写入指定的目录下完成对恶意软件进行修改校验和的操作。

图3-15描述了恶意软件修改时间戳的具体过程，首先会将待处理的恶意程序读取为二进制字符串，随后会修改首部的TimeDateStamp字段为目标校验和，最后，将修改后的二进制字符串写入指定的目录下完成对恶意软件进行修改时间戳的操作。

图3-16描述了恶意软件尾部增加无意义资源节的具体过程，最开始完成的操作是决定新增加节的名称，需要获取恶意软件中所有节的名称，并且与所有预先设定好的备选节名称做差集，得到的结果为可用的备选节名称，在其中随机选取一个座位新增加的节的名称。新增加的节中的内容可从良性字符串集合中任选一个。在选取新增节的名称和新增节的内容后，使用lief库的add\_section操作即可对恶意软件进行新增节的操作。

图3-17描述了恶意软件添加新的图标资源的具体过程，首先需要预备一些带有图标的良性文件集合，使用resourcehacker从中提取ico图标资源。在操作进行前，需要对提取到的ico图标资源进行修改像素和截取操作生成待添加的备用图标资源，确保每次添加的图标资源都不同。最后，通过cmd调用resourcehack完成对恶意软件新增图标资源的操作，指定操作为addskip，待添加资源res为生成的备用图标资源中随机选取的一个备用资源图标。

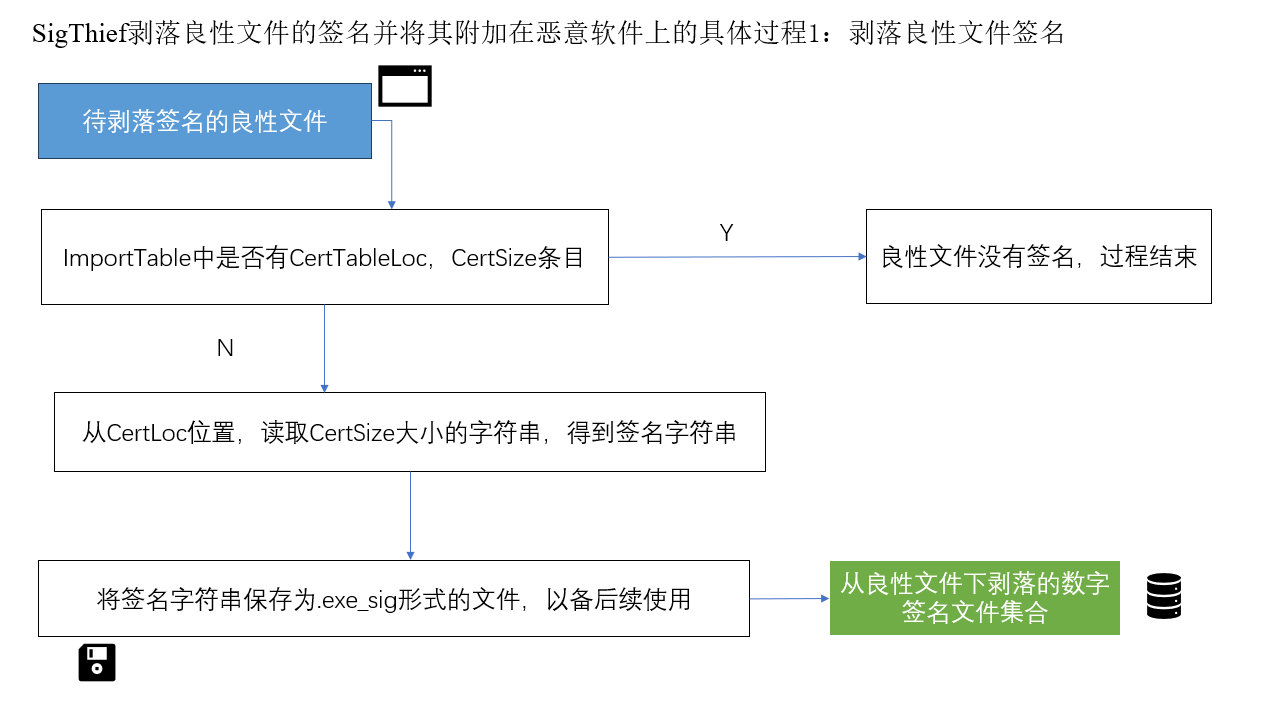


图3-10 剥落签名

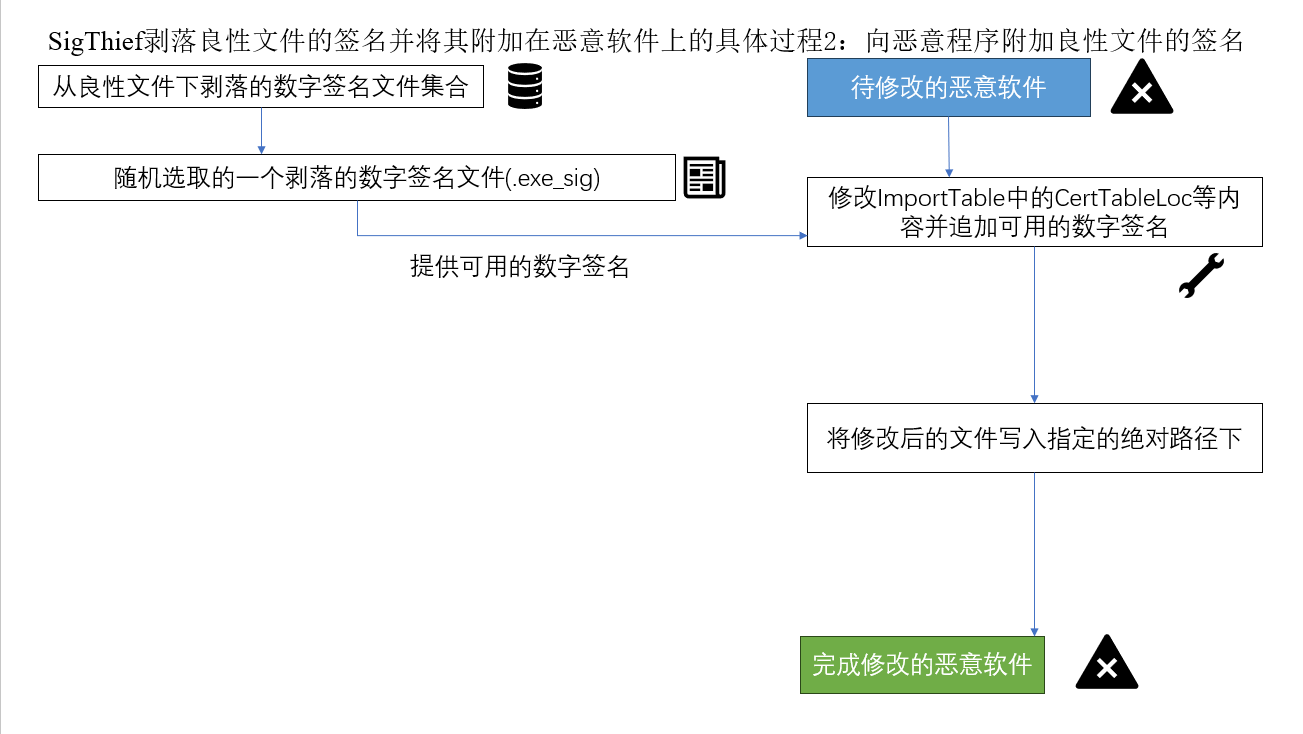


图3-11 附加签名

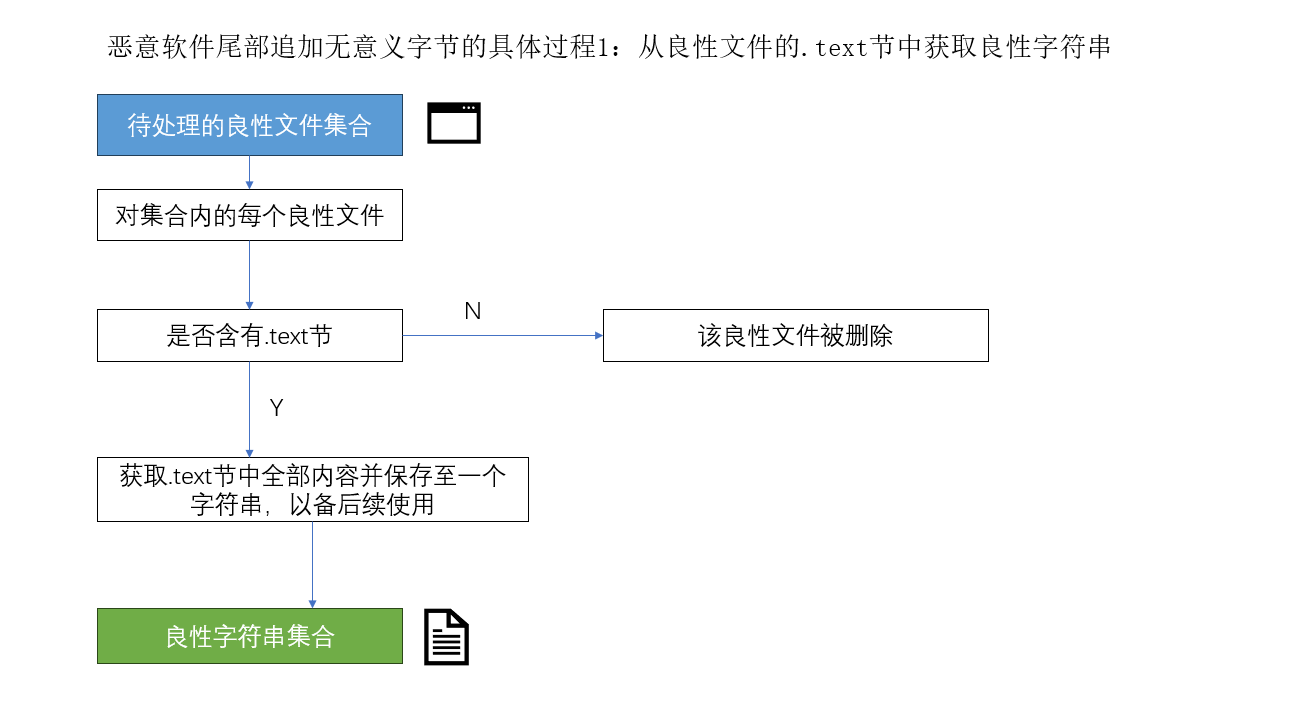


图3-12 获取良性字符串

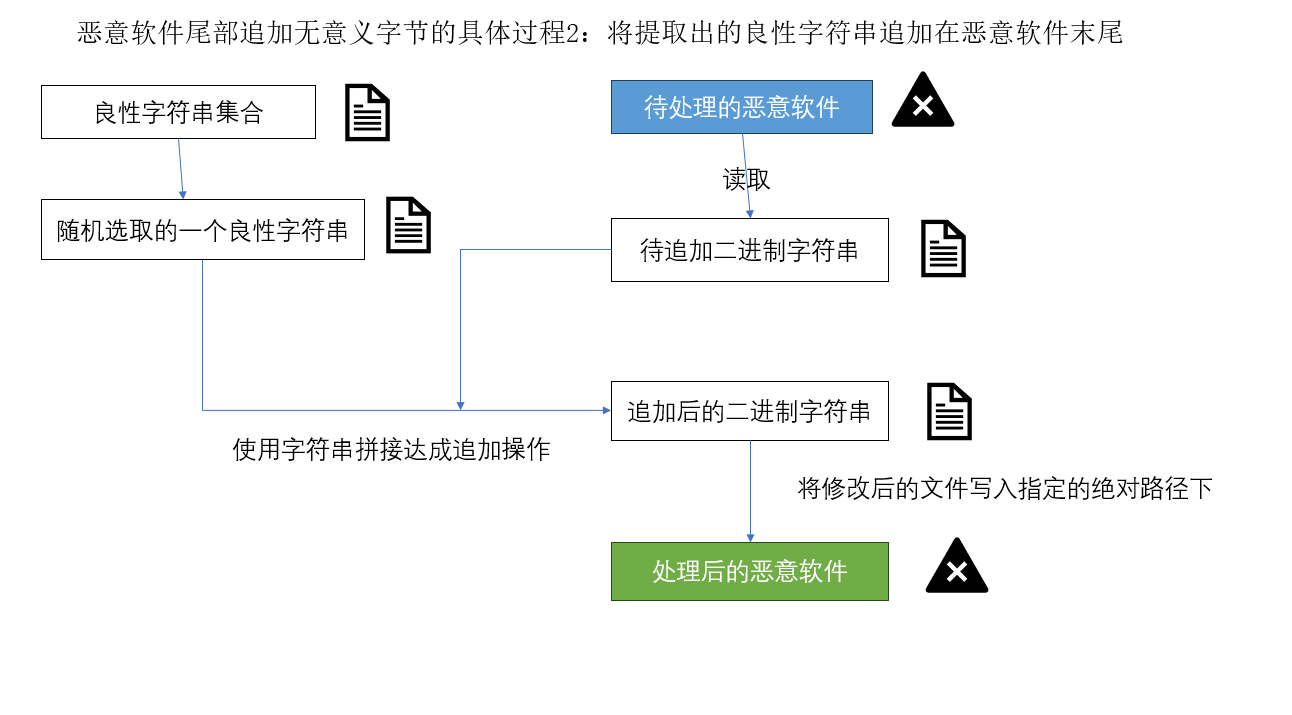


图3-13 尾部追加无意义字节

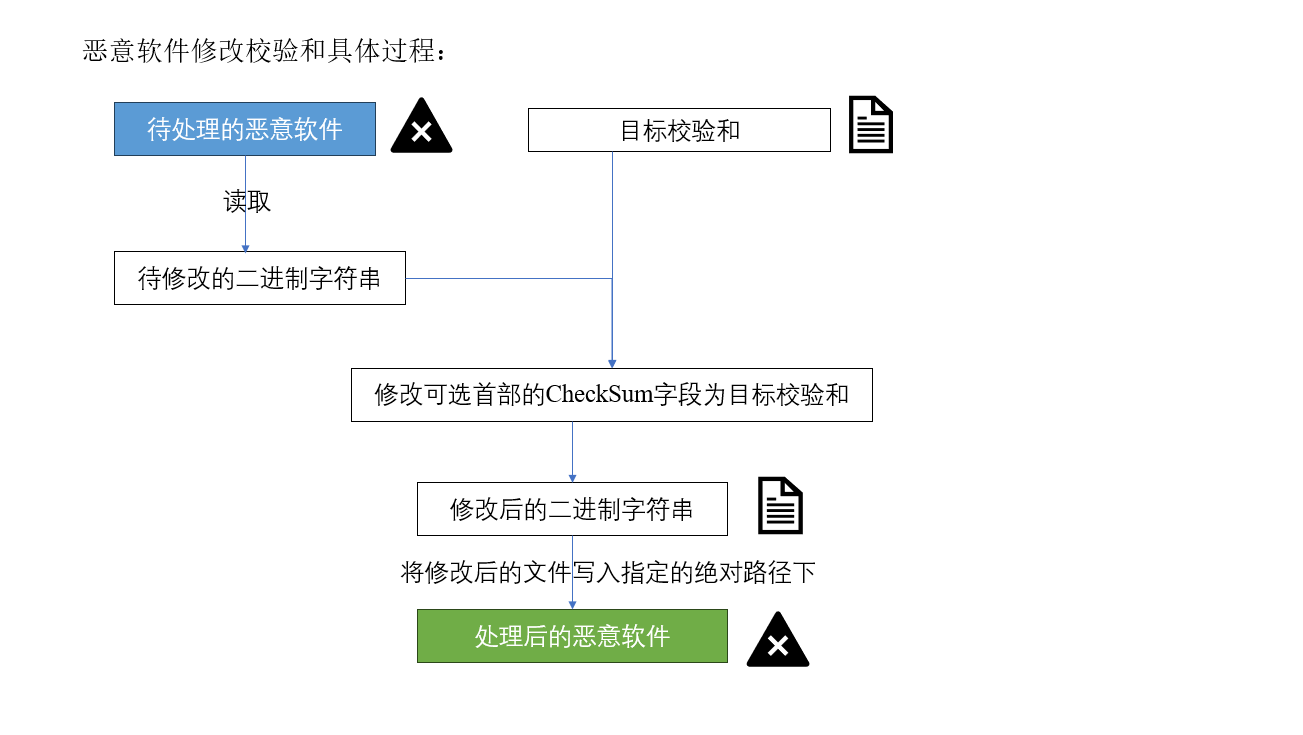


图3-14修改校验和

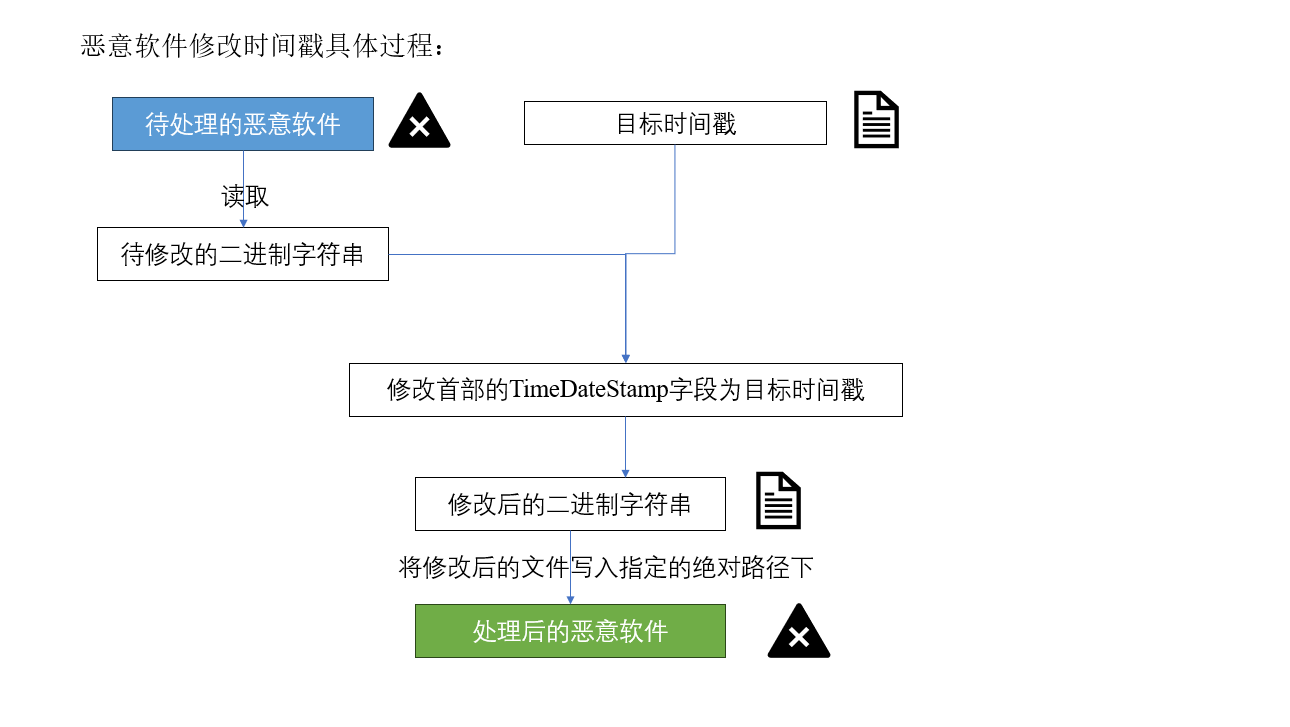


图3-15修改时间戳

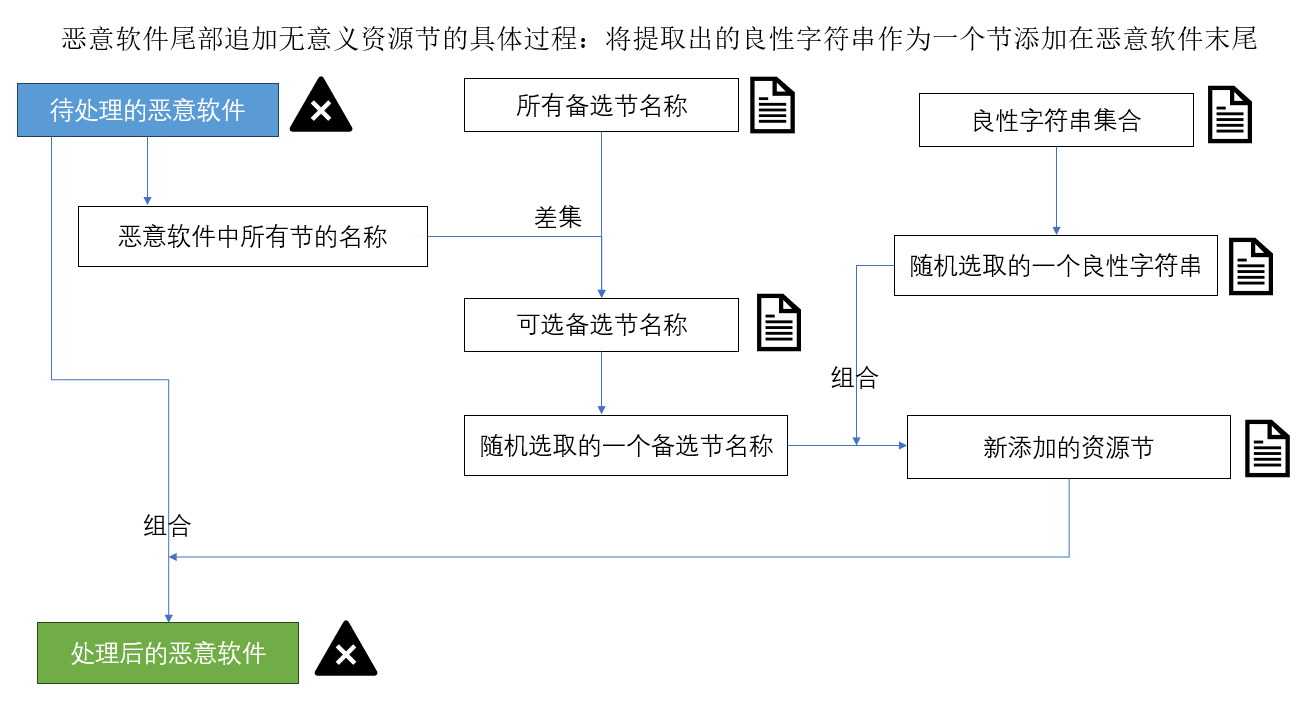


图3-16尾部追加无意义资源节

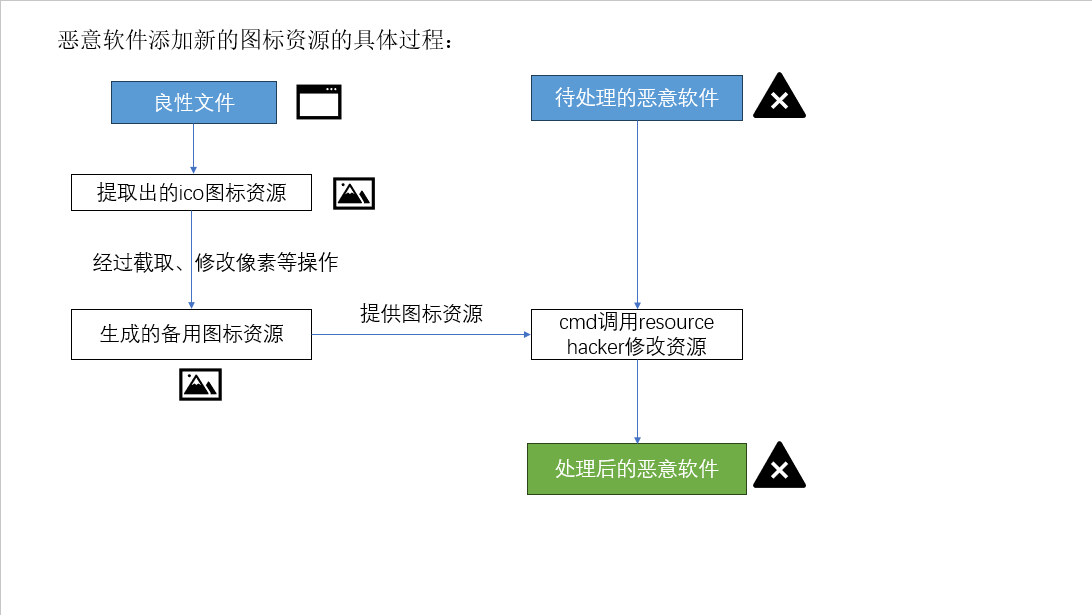


图3-17添加新的图标资源

图3-18中描述了强化学习对抗性样本生成模块的具体工作过程，对于传入样本集中的每个样本，首先设定最初奖励为0、总奖励为0、已进行操作数量为0、终态为否，再根据状态选择其执行的行为，在每次执行行为结束后，检测其是否能逃逸ClamAV的查杀，若能成功逃逸ClamAV查杀，则获取奖励后，进入终止态，并更新强化学习模型的Qtable和更新成功逃逸数量，随后处理下一个样本；若不能成功逃逸ClamAV查杀，则操作次数+1，此时再判断操作数量是否已经达到最大数量，如果已经达到了最大数量，则进入终止态，并且获取相应惩罚，并更新强化学习模型的Qtable，随后将处理下一个样本；如果没有达到最大数量，则更新强化学习模型的Qtable后，进入一个新的状态，并重复上述操作，直到传入样本集中的所有样本都被处理完为止，处理结束后，最终返回处理后能成功逃逸ClamAV查杀的样本数。同时为了提高图3-18中设计的强化学习对抗性样本生成模块的健壮性，我们需要考虑修正强化学习对抗性样本生成模块需要用到的行为表，先前强化学习对抗性样本生成模块关于行为表的设计是合理的，但缺乏健壮性，可能会存在一些隐患，导致某些行为被执行时可能会抛出异常进而导致整个程序崩溃。

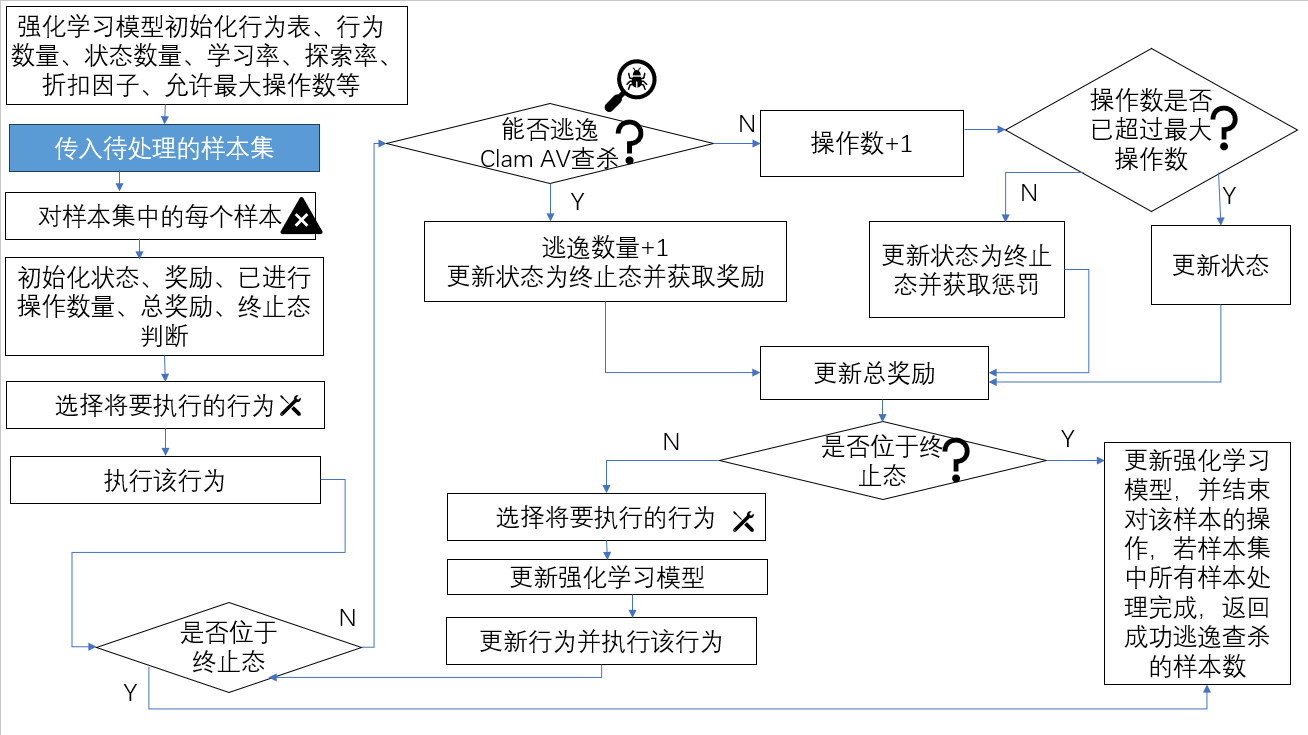


图3-18 强化学习对抗性样本生成模块

原因如下：

1. 样本筛选无法删掉一些伪装的PE可执行文件，尽管这些文件后缀名是.exe，但这些文件使用了结构化欺骗，实际上这些文件自身并不含有COFF文件头等内容，例如一个原本后缀是.sys的程序，后缀被篡改为.exe，如果action在执行时认为它们是PE可执行文件并且对其操作，就会因为PE格式解析异常报错，进而Python主程序将会抛出异常，没有被处理的异常会导致整个程序被迫终止。解决方案是通过为do\_action()函数设置异常处理的代码块来解决这个问题，即使遇到了不是正常的PE可执行文件，程序仍然因为不会抛出异常，导致强制终止整个程序。
2. 存在一些PE可执行文件恶意软件，自身大小非常大，大约64MB到数百MB，可能是这些文件本身已经经过了一些对抗性操作导致的，笔者在测试中发现，对抗性操作在处理它们时极其缓慢，有些甚至需要数分钟的时间，这无疑严重拖累了对抗性处理操作执行的速度。其次，这些较大的PE可执行程序读取和写入操作对磁盘的损害也是巨大的，极有可能在运行中发生系统蓝屏死机的问题，甚至可能会导致数据丢失。解决方案在样本筛选模块已提出，即样本中大于4096字节的PE可执行程序在筛选样本时需要被删除，以避免后续操作出现故障导致程序强制终止。
3. 对于（1）中的问题，在异常处理的基础上，应该额外为action配备日志系统来监控对于PE可执行文件的修改操作。同时记录修改前和修改后文件的SHA256哈希值和使用的操作名称在一个日志文件中。配备日志系统有利于进行对于单个action的黑盒测试时排查错误，以及根据文件的哈希值判断这些对抗性干扰操作是否有效。修改成功的一个必要条件是修改后文件的哈希值与修改前文件的哈希值不同。如图3-19所示，图3-19中表明是为PE文件尾部添加冗余良性字节的操作配备日志，其中使用了logging.basicConfig相关设定来指定日志格式，同时，日志系统最好设置为允许开关，否则可能会在使用较差的硬盘时导致IO时间浪费在日志输出上。

（4）部分action可能会造成原有恶意程序被破坏，例如修改机器码操作，实际测试证明它会损坏恶意程序，所以在操作集中不应该包含它。

（5）部分action如UPX加壳，调用resource hacker加资源，sigthief仿造签名等操作，是通过cmd调用已有应用程序实现，可能会存在进程同步异步问题，可以利用Python自身time库的sleep函数来解决。



图3-19 日志系统

随后，如图3-20所示，在运行前，需要调整Sarsa算法的初始参数中的学习率alpha，探索率gamma，折扣因子epsilon。

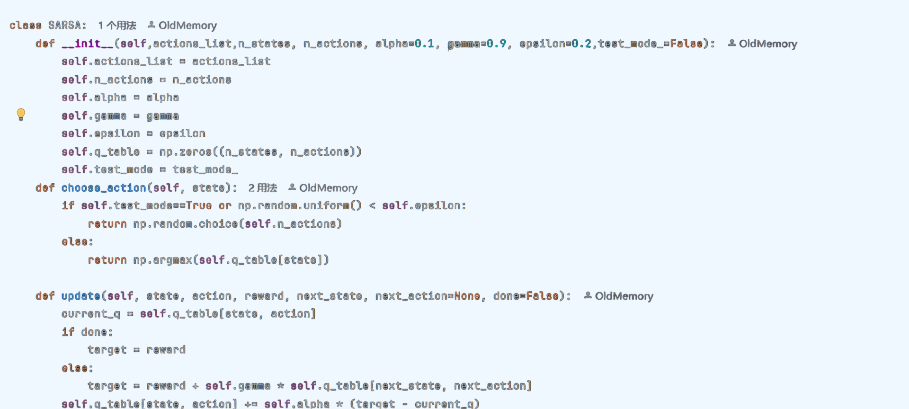


图3-20 初始参数

本实验在强化学习对抗性样本生成模块中实际构建中额外引入了惩罚因子，它与恶意PE可执行文件处理前后的大小有一定关系，式3-1定义了文件大小改变比：

（3-1）

式3-2定义惩罚因子:

（3-2）

因此，成功达到最终状态获取的奖励reward会被再次修正为reward=reward\*。

看似能尽量促进智能体选择尽量不过分扩大原恶意PE可执行文件的大小，然而实行运行时很小概率下会发生问题，这表现在使用UPX加壳后，过于小，导致惩罚因子变成负数，理论上本该因为达成了逃逸效果，被Clam AV检测为良性程序，应该获取的是正数的奖励，但实际上获取的是负值奖励，这与达成逃逸获取正数奖励的设定矛盾。

因此，最好修正惩罚因子，修正后的惩罚因子在式3-3中定义为。

（3-3）

这样可以避免使惩罚因子变为负数，从而影响智能体做出更优的决策的意外。

3.5 本章小结

本章首先介绍了实现强化学习的静态对抗性样本生成系统的设计与实现，将其分为三个模块，并描述需要各个模块的大致设计和需要实现的功能，介绍了对抗性样本生成的算法，恶意程序扫描模块，恶意程序样本筛选的方法。

第4章 实验设计与结果分析

4.1 实验设计

4.1.1 开发环境配置

实验使用如下开发环境：

（1）实体机操作系统及版本号：Windows10 22H2

（2）虚拟机软件，虚拟机操作系统：VMWare Workstation 16.1.2 build-17966106，Windows7 旗舰版

（3）集成开发环境：PyCharm 2024.3.5 (Professional Edition)

（4）Python虚拟环境：Anaconda 24.11.3 AnacondaNavigator2.6.5

（5）反病毒软件：主机安装Clam AV，建议手动更新病毒库到最新版本。虚拟机中安装火绒杀毒和360杀毒以备后续使用，本实验中不建议在主机上安装多个反病毒软件，因为可能会导致主机出现蓝屏崩溃问题，因此，其它用于后续使用的反病毒软件建议安装在虚拟机中。

（6）项目依赖位于项目根目录下的requirements.txt文件中，可以使用anaconda直接导入requirements.txt文件来创建Python虚拟环境。

（7）实体机存储：内存空间 32GB，磁盘空间 512GB

4.1.2 原始样本获取

首先，在开始实验前，需要提前收集足够量的恶意样本以供本实验使用，针对此问题的解决方案是从国外VirusShare的开放恶意样本库，以及部分国内安全软件论坛，例如火绒杀毒，360杀毒等官方论坛，收集并且下载恶意样本。本实验中大约使用2000个恶意样本，并将它们分为五组以备后续使用。如果是在Virus Share上收集，需要注意文件的类型，需要是PE可执行文件。

用于对抗性操作的无害程序，主要用于增加伪造的签名，无害段等对抗性操作，这些无害程序的获取方式，可以从国内一些软件公司的软件安装包获取，使用sigthief将签名剥落，关于sigthief的具体使用可以参考<https://github.com/secretsquirrel/SigThief> 的readme的相关说明，这些从良性程序剥落下的签名可以从一些官方途径下载的软件安装包可执行文件中剥落下来。用于增加无害段需要收集一些良性DLL文件，可以从Windows操作系统根目录下的一些DLL文件中获取。

4.1.3 配置VMWare虚拟机

需要创建实验用Windows7虚拟机并安装VMware Tools，并调整360杀毒相关设置关闭不需要的反病毒引擎，取消勾选系统修复引擎、behavioral脚本引擎，鲲鹏引擎，避免对实验结果造成干扰。同时，需要注意潜在的样本污染问题，对于360杀毒，需要关闭自动上传发现的可疑文件，取消勾选自动上传发现的可疑文件，否则可能导致有些样本被360杀毒上传到云查杀服务器数据库中，导致后续结果查杀率偏高的问题。

需要注意的是，安装360杀毒后需要重启虚拟机以保证360杀毒相应查杀服务开启，否则会导致实验结果出现偏差。

本实验不会测试360杀毒的鲲鹏引擎，是因为大多数反病毒软件，倾向于防御现有的威胁和威胁变种，而不是没有广泛传播的对抗性样本，鲲鹏引擎也如此，它多被360杀毒用于对抗目前广泛流行的恶意样本（例如银狐和某些流行的勒索病毒），而非对抗性样本或是离目前时间较久远的样本，而且鲲鹏引擎几乎没有机器学习的自学习能力，但QVM引擎具有机器学习能力，很明显，QVM引擎更适合用于本次实验的结果检验。这是因为有时需要反病毒软件对未知程序的判断更加精准以及判断速度更快，也就是尽量降低误报的可能性以及判断时间。为了减少误报率，很多反病毒软件厂商可能会使用白名单数字签名放行，或是使用文件哈希白名单来进行放行。因为例如Intel、AMD的硬件驱动程序，它们会加载驱动，如果这些硬件外设驱动程序因为释放了一些驱动文件被反病毒软件误报没有放行，将会导致某些外设驱动不能正常安装，对用户来说后果无疑是灾难性的，极有可能导致操作系统或是计算机崩溃。但这些硬件驱动程序跟某些病毒的行为很相似，某些病毒也会使用驱动来提升自己的操作权限，使自己具有更强的破坏力以及更好地针对和破坏反病毒软件，因此，对于这些驱动程序以及一些驱动安装工具，大多数反病毒软件会选择单独白名单规避此类问题。同时，360杀毒的behavioral脚本查杀引擎和系统修复引擎也需要关闭防止造成检出率偏高导致误差。

4.1.4 结果评估标准

首先，定义查杀率如下：

查杀率计算公式如式4-1：

(4-1)

对于某个恶意软件集合，某反病毒软件的查杀率被定义为经该反病毒软件扫描该恶意软件样本集合的检出恶意软件数量除以恶意软件集合中总恶意软件数量。

为了更好计算样本集的规避反病毒软件的查杀效果，使用该公式要求原始样本和生成的对抗性样本前后都使用同一款反病毒软件进行扫描，则查杀下降率定义如式（4-2）所示：

(4-2)

语言描述为：查杀下降率=（原始样本检出率-处理后样本检出率）/原始样本检出率。

Virus Total检出率的定义为式（4-3）所示：

(4-3)

对于某个恶意软件样本，VirusTotal检出率定义为检出引擎总数/Virus Total扫描引擎总数。

Virus Total检出率的下降率定义如式（4-4）所示:

(4-4)

对于某个恶意软件样本，VirusTotal检出率的下降率定义为（原始样本VirusTotal检出率-处理后样本VirusTotal检出率）/原始样本VirusTotal检出率。

4.2 实验结果

在本次实验中，一共使用了五组样本集，合计大约2000个样本来进行测试，测试结果如下所示。对于处理前的原始样本集和处理后的样本集，分别使用前文中配置好的恶意程序扫描模块和手动上传至虚拟机使用火绒杀毒和360杀毒扫描，并且随机选取一些原始样本和对应的处理后样本上传VirusTotal进行分析。值得注意的是，获取实验结果数据时，在虚拟机中使用360杀毒和火绒对样本扫描之前，需要关闭虚拟机中的反病毒软件，再在主机上使用XFTP等软件或VMwareTools拖拽向虚拟机中发送样本集，防止样本集发送过程中受到反病毒软件的干扰和潜在的数据污染问题，例如360杀毒会默认禁止存在风险的远程主机向本主机的FTP协议，在本次实验中对于虚拟机我们不需要这个防御功能。且在主机和虚拟机的FTP协议或VMWareTools拖拽完成文件传输后，虚拟机接收到文件后，360杀毒会自动扫描文件，这是不应该被允许的。因为会导致后续检出率偏高以及360杀毒等反病毒软件会直接隔离虚拟机接收到的样本集中的一些样本，对后续统计查杀数量带来了不便。

4.2.1 实验结果的数据图示

五组样本集处理前后的扫描结果图4-1、图4-2、图4-3、图4-4、图4-5、图4-6、图4-7所示，对其具体分析将在后续章节中进行。

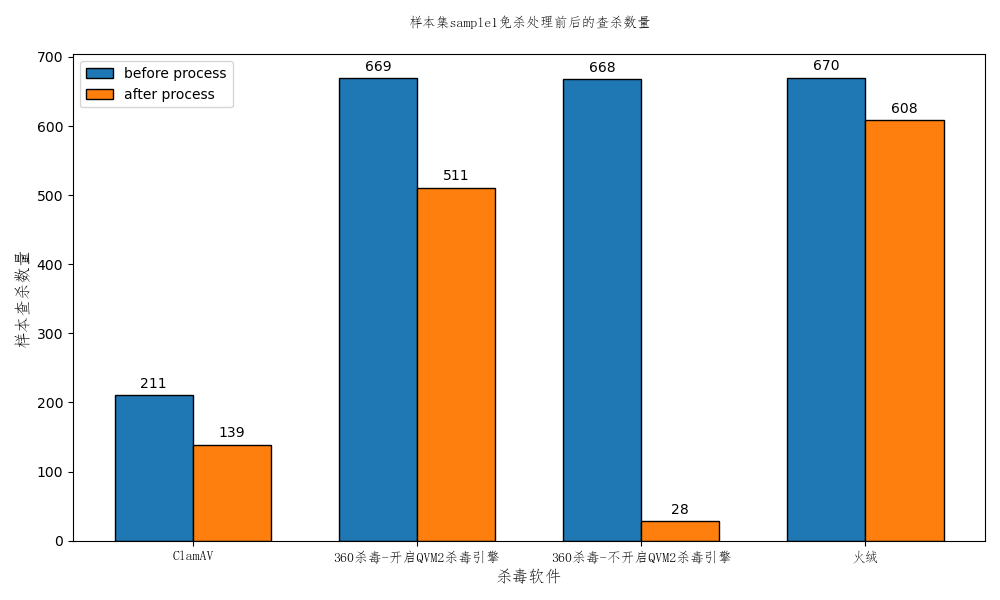


图4-1 sample1

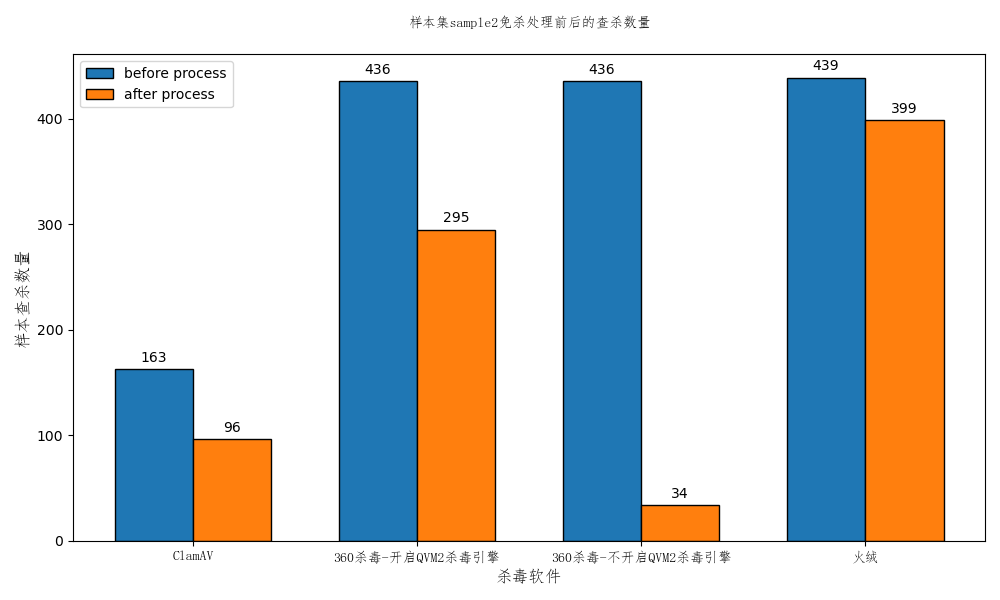


图4-2 sample2

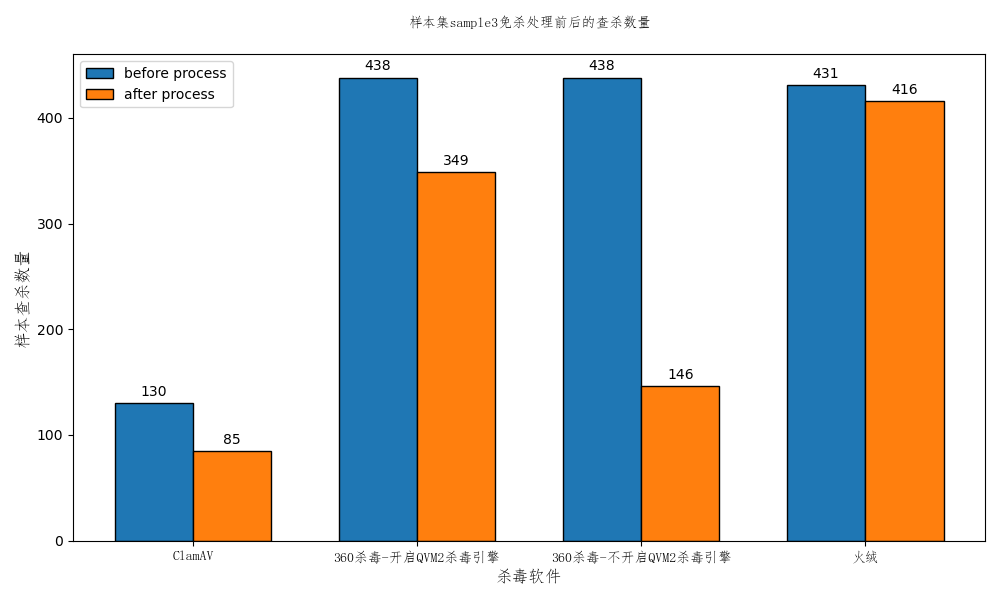


图4-3 sample3

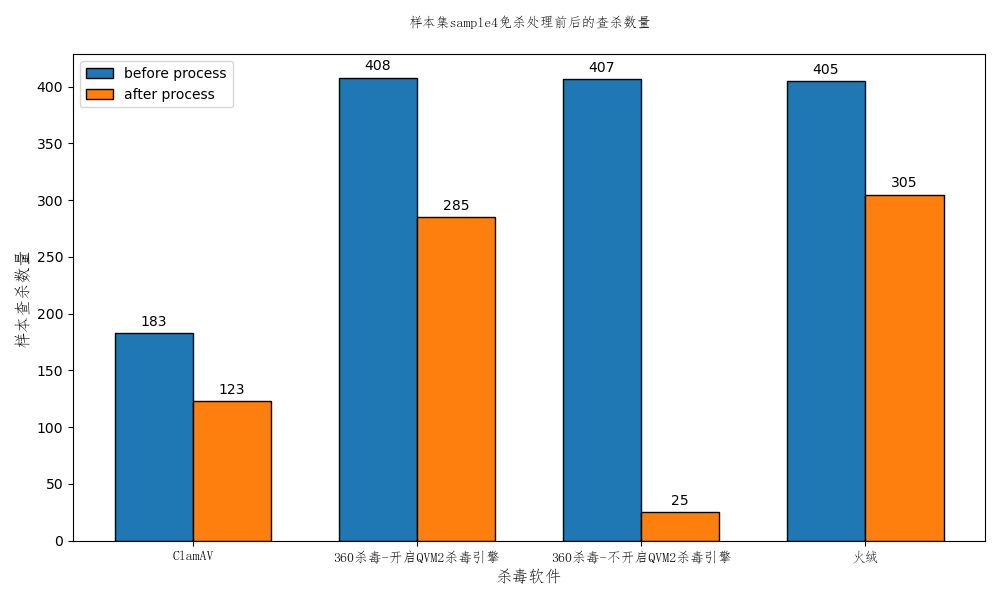


图4-4 sample4

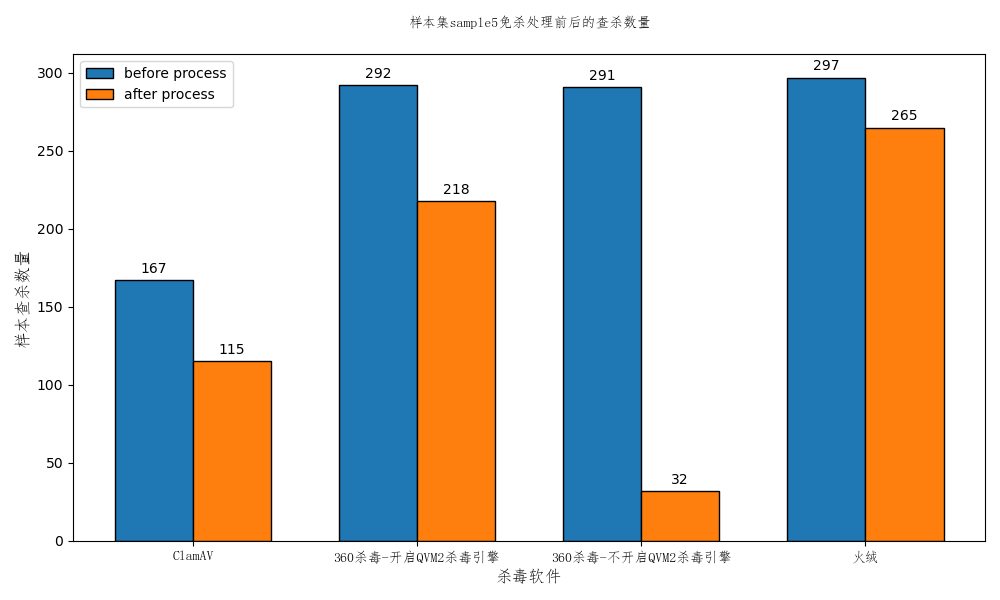


图4-5 sample5

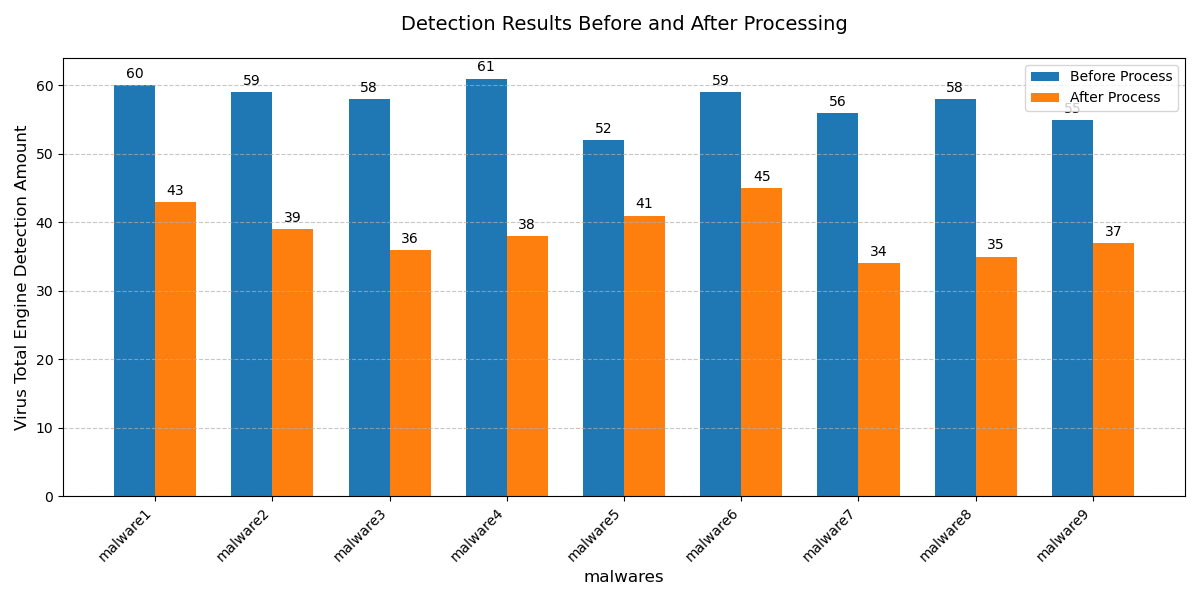


图4-6 Virus Total

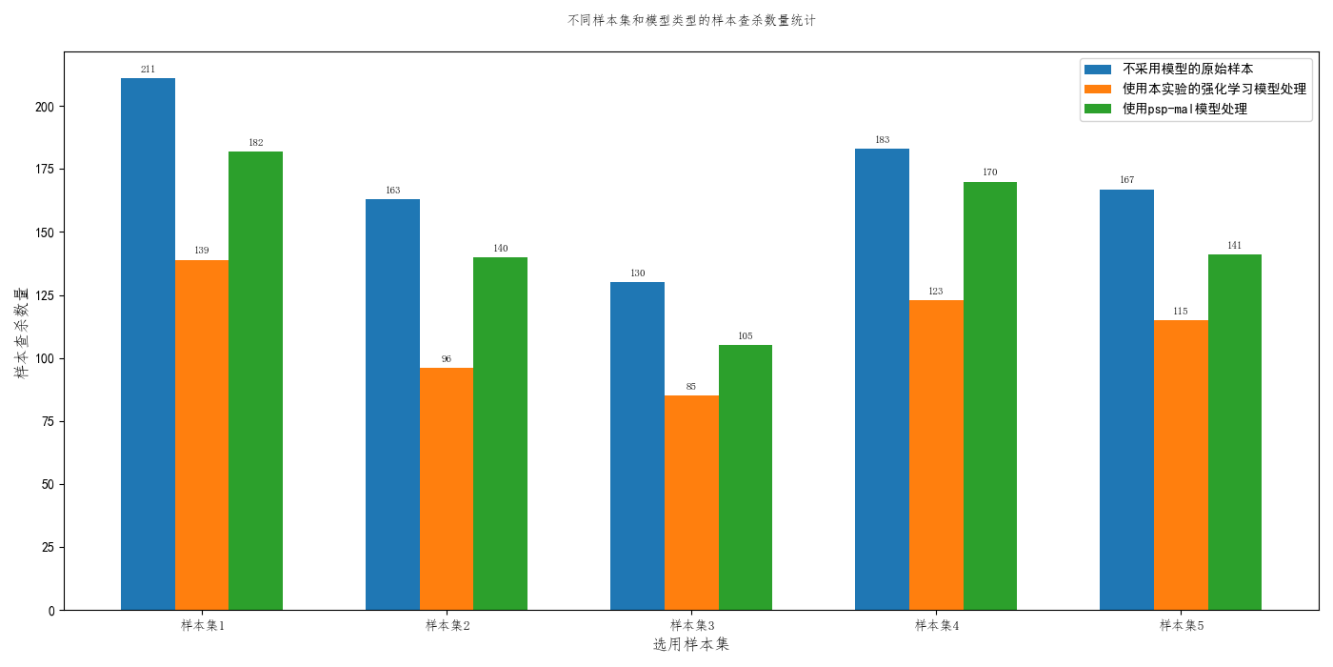


图4-7不同模型ClamAV检出对比

4.2.2 检出率及其变化的计算

首先，对于某组样本中的随机抽样的部分程序，笔者使用了Virus Total来进行上传扫描。

经过对图4-1，图4-2，图4-3，图4-4，图4-5中的数据进行计算，可以发现：

（1）对于样本集sample1：

Clam AV对本实验强化学习模型处理前后的样本检出率从31.3%下降到了20.7%；开启QVM杀毒引擎的360杀毒对本实验强化学习模型处理前后的样本检出率从99.3%下降到了75.8%；只开启360云查杀引擎，不开启QVM杀毒引擎的360杀毒对强化学习模型处理前后的样本检出率从99.1%下降到了4.2%；火绒杀毒对本实验强化学习模型处理前后的样本检出率从99.4%下降到了90.2%。

（2）对于样本集sample2：

Clam AV对本实验强化学习模型处理前后的样本检出率从37.4%下降到了22.0%；开启QVM杀毒引擎的360杀毒对本实验强化学习模型处理前后的样本检出率从100%下降到了67.7%；只开启360云查杀引擎，不开启QVM杀毒引擎的360杀毒对本实验强化学习模型处理前后的样本检出率从100%下降到了7.8%；火绒杀毒对本实验强化学习模型处理前后的样本检出率从100%下降到了91.5%。

（3）对于样本集sample3：

Clam AV对本实验强化学习模型处理前后的样本检出率从29.5%下降到了19.3%；开启QVM杀毒引擎的360杀毒对本实验强化学习模型处理前后的样本检出率从99.5%下降到了79.3%；只开启360云查杀引擎，不开启QVM杀毒引擎的360杀毒对本实验强化学习模型处理前后的样本检出率从99.5%下降到了33.2%；火绒杀毒对本实验强化学习模型处理前后的样本检出率从98.0%下降到了94.5%。

（4）对于样本集sample4：

Clam AV对本实验强化学习模型处理前后的样本检出率从44.9%下降到了30.1%；开启QVM杀毒引擎的360杀毒对本实验强化学习模型处理前后的样本检出率从100%下降到了69.9%；只开启360云查杀引擎，不开启QVM杀毒引擎的360杀毒对本实验强化学习模型处理前后的样本检出率从99.8%下降到了6.1%；火绒杀毒对本实验强化学习模型处理前后的样本检出率从99.3%下降到了74.8%。

（5）对于样本集sample5：

Clam AV对本实验强化学习模型处理前后的样本检出率从57.2%下降到了39.7%；开启QVM杀毒引擎的360杀毒对本实验强化学习模型处理前后的样本检出率从100%下降到了74.7%；只开启360云查杀引擎，不开启QVM杀毒引擎的360杀毒对本实验强化学习模型处理前后的样本检出率从99.7%下降到了11.0%；火绒杀毒对本实验强化学习模型处理前后的样本检出率从100.0%下降到了90.8%。

根据查杀下降率的定义，对于五个样本集合的查杀下降率计算结果如下表中所示：

表4-1 查杀下降率结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Sample1 | Sample2 | Sample3 | Sample4 | Sample5 |
| Clam AV | 34.1% | 41.1% | 34.6% | 32.8% | 31.1% |
| 360杀毒开启云查杀引擎和QVM引擎 | 23.6% | 32.3% | 20.3% | 30.1% | 25.3% |
| 360杀毒开启云查杀引擎不开启QVM引擎 | 95.8% | 92.2% | 66.7% | 93.9% | 89.0% |
| 火绒 | 9.3% | 9.1% | 3.5% | 24.7% | 10.8% |

计算平均值如表4-2中所示：

表4-2 查杀结果下降平均值

|  |  |
| --- | --- |
| Clam AV | 34.74% |
| 360杀毒开启云查杀引擎和QVM引擎 | 26.32% |
| 360杀毒开启云查杀引擎不开启QVM引擎 | 87.52% |
| 火绒 | 11.48% |

从结果中看出，对抗性样本对于不同反病毒软件的有效性，对360云查杀引擎最有效，对火绒的有效性最差。

前文中图4-6使用Virus Total扫描上传从某个样本集合内随机抽样的PE恶意程序，在云端总共使用了70余个反病毒软件进行分析。

处理前后检出率变化以及检出下降率变化如表4-3所示：

表4-3 处理前后的恶意软件在VirusTotal上检出率对比

|  |  |  |  |
| --- | --- | --- | --- |
|  | 处理前检出率 | 处理后检出率 | 检出下降率 |
| malware1 | 60/72=83.3% | 43/72=59.7% | 28.3% |
| malware2 | 59/73=80.8% | 39/72=54.2% | 33.0% |
| malware3 | 58/73=79.5% | 36/72=50% | 37.1% |
| malware4 | 61/73=83.6% | 38/71=53.5% | 36.0% |
| malware5 | 52/73=71.2% | 41/72=56.9% | 20.0% |
| malware6 | 59/72=81.9% | 45/73=61.6% | 24.8% |
| malware7 | 56/72=77.8% | 34/72=47.2% | 39.3% |
| malware8 | 58/72=80.6% | 35/72=48.6% | 39.7% |
| malware9 | 55/73=75.3% | 37/72=51.4% | 31.8% |

且经过统计，当最大允许处理次数设置为16时，对于样本集sample1的处理总共花费12463.1秒，对于单个样本的平均处理时间为18.49秒。

样本集1样本大小变化率的散点图如图4-8所示，大小变化率的定义为对于某个确定的样本，其处理后的大小除以处理前的大小，可以发现，有少数处理后的样本大小变的较大，甚至可能会达到1000倍以上，这可能是因为添加节所选取的良性DLL文件中的.text节过于庞大导致的。

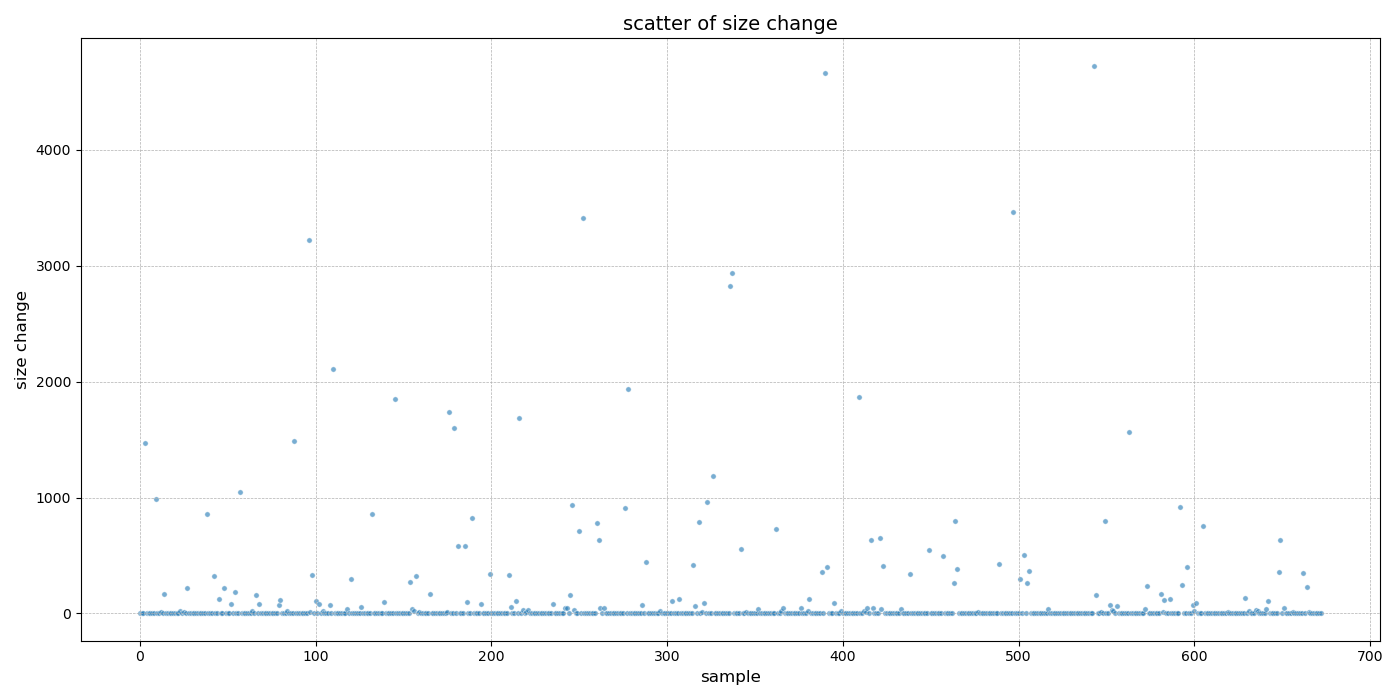


图4-8 scatter of sample1

4.2.3 不同反病毒软件对于检出率变化的影响

处理后的样本对于火绒杀毒来说，表4-1中数据计算出的逃逸概率低的原因是火绒有一定的反扰动能力，处理后的样本很多都会判断为：病毒类型-代码混淆器，火绒认为处理后的样本的某些特征是为了规避静态检测过程。说明有一部分反病毒软件的静态特征分析能力，能识别出对抗样本中的扰动。这也是为什么采用Sarsa算法不宜设置过大的state数值的原因，一方面是处理前后的PE程序反复读写，当程序大小因为新增节，新增无意义尾部内容，用无意义字节填充空洞等操作多次使用后变的过于庞大（在某次样本处理时，笔者观察到了一个PE程序处理前后，变为原大小大约6.6倍，从2.67 MB变成了17.6MB），如果不限制state值，程序中途会变得很大，不但导致后续的对抗性样本生成的干扰操作极其缓慢，因为要先读取原文件，然后再加以修改，最后写入磁盘中，Python对于文件的操作是很缓慢的。而且会导致根据查杀结果获取result的速度相当慢，因为Clam AV反病毒软件扫描一个样本的时间也和样本大小正相关，当样本很大的时候，判断查杀结果会相当的慢。另一方面，就是当无意义尾部内容过多，会被反病毒软件认为是疑似恶意程序刻意扰动，从而直接判定此程序为为了绕过反病毒软件检查的恶意程序。而从火绒杀毒对于样本的描述也可以看到这一点，火绒杀毒认为经过处理后的某些恶意程序是代码混淆器。同时，火绒查杀结果总数量会略高于样本数量，原因目前未知。

同样对于Clam AV，可知它的部分检测也能识别对抗样本中的扰动性操作，尽管经过处理后的恶意程序中有大量良性内容，但Clam AV仍然会查杀这些恶意程序。

由此可见，部分反病毒软件对于静态规避操作的对抗性扰动有一定的抵抗，会造成处理前后检出率变化较小。

但QVM引擎对于对抗性扰动能力偏弱，这是因为QVM引擎作为一个自学习的人工智能引擎，这个病毒检测引擎是基于机器学习的模型，通过检测文件的某些特征，来判断该文件是否是恶意程序，如果某程序被QVM引擎检测出是恶意程序，那么360杀毒在查杀结果中会报HEUR/QVM.xx.Malware.Gen字样。这里使用QVM引擎作为检测标准是为了研究论文中的强化学习对抗性样本生成模型产生的处理后静态检测规避对抗性样本在基于不同模型的反病毒软件之间的逃逸可迁移性，以及验证机器学习的病毒检测引擎对于静态检测规避对抗性样本的抗干扰能力。但遗憾的是，QVM引擎对于静态检测规避对抗性样本的抗干扰能力比较差，有时甚至仅仅增加了无意义的图标ICO资源，QVM引擎就认为样本是良性的。

4.2.4 与psp-mal模型的对比

如图4-7，我们可以发现本实验强化学习模型处理后的样本集，相比psp-mal模型，能更好规避ClamAV反病毒软件的查杀，原因可能是新增的伪造签名操作，增加无关资源操作，和UPX加壳等操作导致的。检出率的变化对比如表4-4所示，值得注意的，不同于本实验中使用了Sarsa强化学习算法，psp-mal模型使用的方法是沙普利先验，检验是否逃逸成功使用的是SOREL-20M和EMBER反病毒模型，遗憾的是，通过对图4-7的分析，我们可以知道psp-mal模型的可迁移性较差，尽管psp-mal模型生成的对抗性样本能逃脱EMBER反病毒模型，但是对于ClamAV的查杀来说，规避性能较差，甚至检出率变化不足20%。由此可见，本实验设计的创新点有利于对抗性样本的迁移，能达到对不同反病毒软件的静态检测逃逸效果。

表4-4检出率变化对比

|  |  |  |
| --- | --- | --- |
| 样本集\检出率变化 | 本实验中强化学习模型 | Psp-mal |
| Sample1 | 34.1% | 13.7% |
| Sample2 | 41.1% | 14.1% |
| Sample3 | 34.6% | 19.2% |
| Sample4 | 32.8% | 7.1% |
| Sample5 | 31.1% | 15.6% |

4.2.5 静态对抗性样本对云查杀的影响

相对于传统的对抗性样本需要恶意攻击者手动修改自身的恶意程序二进制代码，繁琐低效，而且可能因为修改失误导致恶意程序受损，产生对抗型样本的数量少，基于人工手动修改的对抗性样本难以对抗反病毒软件的云查杀，人工手动修改的变种终究只有少量。而本实验基于强化学习的Sarsa模型自动生成对抗型样本，对于单个恶意程序，能产生的对抗性样本数量较大，当变种数量足够多时，在有限的时间内，反病毒软件的云查杀系统未必能全部识别出。大量的对抗型样本对云查杀的压力也是较大的。

为了研究云查杀对于检出率的影响，需要在VMware创建的Windows7虚拟机中使用360杀毒进行测试静态检测规避对抗性样本的查杀情况且在360杀毒的设置中打开了自动上传发现的可疑文件。在结果中发现尽管部分处理后的样本逃逸了360杀毒的自定义查杀的检出，但是360杀毒仍然将这些逃逸样本归类为可疑文件。虽然这些逃逸样本不会报为高危风险项，但是360杀毒仍然会将这些样本上传到360云安全中心来进行检验，在可疑样本被360杀毒上传之后的一段时间，再次启动自定义查杀扫描剩余的逃逸样本，有部分样本被360杀毒检出判定为高危风险项。说明对于开启未知程序上报的云查杀（笔者后续得知360云安全中心是有动态行为检测的，并不完全只有静态检测）的安全软件来说，对抗性样本攻击虽然有一定危害，但大多数仍然最终会被检出是恶意程序。同时，360杀毒也具有动态检测功能，例如Behavioral脚本引擎、主动防御模块这些基于动态行为分析的检测，如果用户执行了恶意程序，360杀毒会拦截并且隔离恶意程序。当反病毒软件具有动态检测功能，则静态对抗性样本攻击的效果是有限的，但仍然不可以忽视。究其原因，是因为某些病毒具有反虚拟化策略、反沙箱、识别监控进程的功能[33],[34],[35],[36]。

云查杀技术对这些病毒来说未必有效，因为在被上传到云端沙箱（例如Cuckoo Sandbox）中之后，恶意软件不会表现出预期的恶意行为，从而阻碍了恶意软件行为分类和恶意行为的自动识别检测。过度依赖云查杀仍然可能会存在潜在的威胁，同时，云查杀需要联网访问云端恶意软件样本数据库的问题也是一大安全隐患，对抗性恶意程序可能在自己被运行时，因为反病毒软件因为对抗性恶意程序自身的扰动导致判断该恶意程序是未知安全性，判断后需要上传到云端服务器，来利用云查杀技术进行分析，而这时的恶意程序自身可能会通过修改网络设置导致计算机无法连接到互联网，或利用Windows防火墙规则，本地Hosts文件篡改反病毒软件请求服务器地址到本地环回地址127.0.0.1等方式，使反病毒软件无法连接到互联网，从而阻止了反病毒软件从云端服务器获取云端沙箱检测等传回来的信息，这就导致反病毒软件对于对抗性恶意程序无法进行精确地判断是否具有恶意性。更糟的情况是当云查杀技术面对未来可能出现的感染型病毒变种对抗性样本和蠕虫对抗性样本。

传统的感染型病毒，例如文件型病毒，只感染固定的目标文件[37]，例如PE可执行文件（.exe），动态链接库文件（.dll），html文件（.html/.htm），以及目前发现的一些宏病毒，只感染office文件，例如后缀为docx、xlsx、pptx、xls、doc、ppt的文件等。

其中的一部分传统感染型病毒，通过将自身的代码附加在正常程序后来感染正常程序。当程序被运行时，其中的病毒代码也随之运行。然而，传统的感染型病毒很容易被反病毒软件查杀，因为反病毒软件只需要进行全盘扫描，扫描文件系统中的所有文件来寻找已被病毒感染的文件，通过静态分析，检查其尾部内容是否有病毒库中黑名单的恶意代码字段，即可判断这个文件是否已经被感染型病毒所感染。而且反病毒软件对被感染的文件可以通过匹配相应二进制字符串，如通过正则表达式匹配，从而直接去除截断掉尾部的病毒代码进行文件修复。因为病毒代码和正常程序代码是物理分离的，所以反病毒软件的操作可以将受感染型病毒感染的文件恢复到原始文件状态，即对被感染的文件实行修复而不是直接删除或者隔离，可以使被感染的文件保留其原始正常文件的功能。

然而，通过本实验，可以说明，将来很可能存在某种感染型病毒变种对抗性样本。不同于传统的感染型病毒，它的危害相对较大，本身具有感染新文件后，自身也会突变不断制造新变种的新型攻击方式。这种病毒除了自身的恶意代码，也携带了对抗性扰动操作的相关模块，例如产生一个内部包含被pyinstaller打包的附带Python编译的PE可执行文件，用于产生静态检测对抗性扰动来针对基于机器学习和深度学习的反病毒软件以实现逃逸静态检测的查杀。该变种对抗性样本的部分行为可用程序流程图表述，如图4-9所示，图4-9为该变种对抗性样本的感染流程以及使被感染的文件规避静态检测流程，在病毒因为用户运行导致入侵未被病毒感染的系统后，该病毒会在磁盘中通过遍历文件系统的目录来寻找新的可被该病毒感染的目标文件，将自身代码附加在被感染程序后，并判断产生对抗性扰动操作的程序对应的Environment\_Variable\_α环境变量是否存在，若存在则调用产生对抗性扰动的可执行程序对被新感染的程序进行处理，否则释放可产生对抗性扰动的可执行程序，随后将其移动到安装有操作系统的磁盘的某个位置（可以为系统根目录）下，随后设置文件为隐藏属性，添加该可执行程序到系统的环境变量中，最后调用产生对抗性扰动的可执行程序对被新感染的程序进行静态免杀处理，其伪代码描述如下：

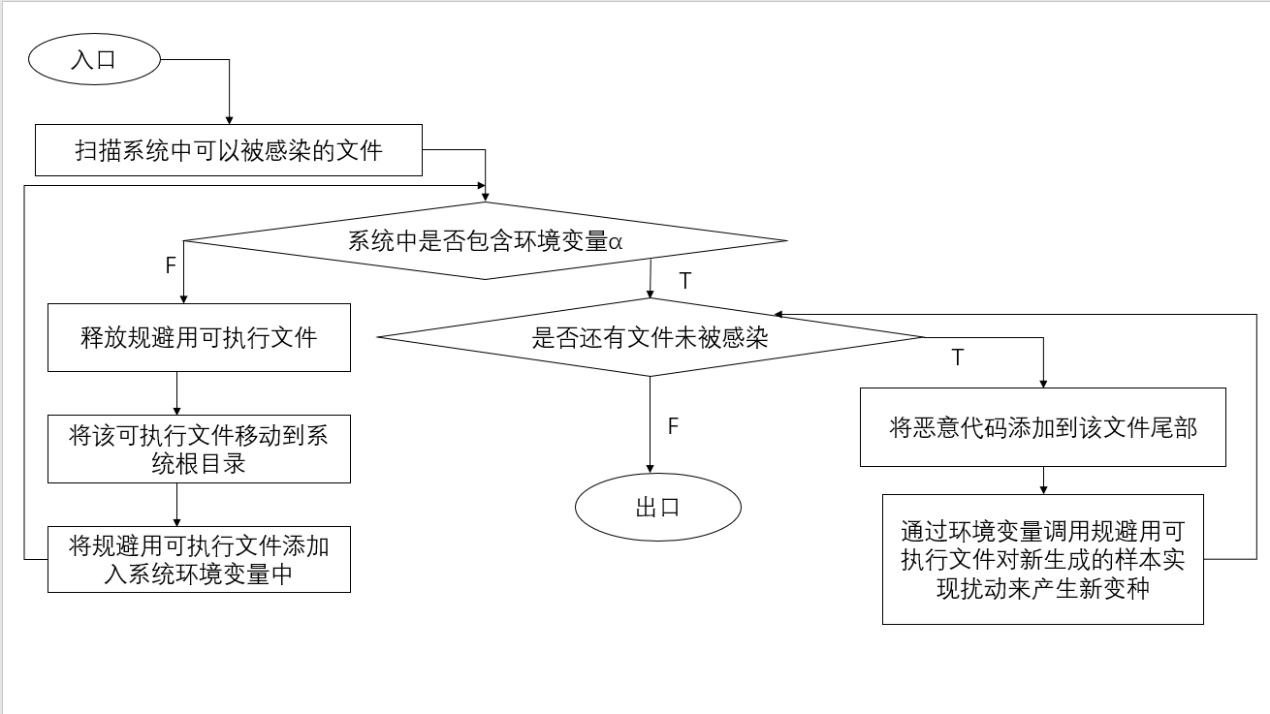


图4-9攻击过程

1. start(); //病毒被运行
2. scan\_all\_files\_in\_system\_to\_get\_can\_infected();
3. function1:
4. **if** System\_Environment\_Variables.contains(Environment\_Variable\_α){//检测到环境变量
5. **for** file in files\_virus\_can\_infected{//感染过程
6. append\_malicious\_code(file);
7. run\_evasion\_module(Environment\_Variable\_α);
8. }
9. }
10. **else**{//释放对抗性操作文件及环境变量准备
11. release\_executable\_file\_for\_evasion();
12. move\_evasion\_module\_file\_to\_system\_root\_folder();
13. add\_environment\_variable\_to\_system\_environment\_variables(Environment\_Variable\_α);
14. go to function1;
15. }

功能：感染型病毒变种入侵系统并感染可以感染的文件，在感染新文件时执行对抗性操作，被感染的新文件会受到静态规避操作的修改。

输入：无

输出：无

这种感染型病毒变种每次新感染的程序的尾部病毒代码将互不相同，因为受到了对抗性扰动可执行程序的修改。基于静态分析的反病毒软件很难查杀这种变种对抗性样本，很容易面临能杀毒，但是查杀不完全的问题，每次杀毒如果不能检测出感染型病毒变种对抗性样本感染的全部文件，遗留下来的受感染文件一旦被用户无意间通过启动，例如通过某些Windows应用程序的开机自启动，用户打开浏览器访问网页但是浏览器主程序已被感染，仍然会导致其他文件被感染。更危险的是，被感染的文件难以像应对传统文件型病毒的修复措施，即精准去掉尾部的病毒代码来恢复原始文件。因为需要处理对抗性扰动新增的节和无效字节，甚至有可能是导入表被篡改，节表被修改的行为，一旦去除有偏差，则会导致受感染文件无法正常运行。因此反病毒软件极有可能无法通过黑名单中的病毒代码来计算线性偏移量，进而直接截断固定长度的尾部从而修复直接受感染文件，只能选择删除，这对用户来说是危害巨大的，因为很多的Windows软件包含了大量的动态链接库（.dll）文件和PE可执行文件（.exe），一旦某些动态链接库被反病毒软件删除后，这些软件极有可能无法正常运行。即使反病毒软件拥有云查杀功能，频繁上传大量变种样本不但消耗了用户带宽资源，而且变种样本因为每个被感染的实例的尾部特征值唯一，难以在第一次扫描中就被反病毒软件检测出是恶意程序，只会被定性为不流行的可疑文件，这导致对感染型病毒的响应延迟，很有可能会面对云查杀查杀病毒并不彻底的问题。这将是一种可能存在的高级持续性威胁（APT）。

幸运的是，即使面对这种将来可能存在的感染型病毒变种对抗性样本，因为它只有不断产生静态规避变种的功能，而没有不断产生动态规避变种的功能，反病毒软件的动态行为检测仍能在病毒程序未执行感染操作时触发，识别出恶意程序并警告用户。同时，PE文件的动态规避目前尚未出现规模化的攻击方法，尽管已有研究证明可通过代码分片来攻击动态行为检测[38],[39],[40]。

同时，基于静态规避对抗性生成模型，未来也有可能会出现新的木马下载器变种，以及基于Web请求拦截器和对抗性生成模型挂马网站变种，这值得反病毒软件厂商警惕。许多网页框架，例如Java的SpringBoot CGI服务器，具有Interceptor接口用于拦截器操作，然而，这可以被黑客用于木马下载器来下载其他木马的静态规避变种，黑客可以使用SpringBoot的Interceptor接口确保每次访问挂马网站下载到的木马都是不同的。当木马下载器试图使用http的get请求访问黑客的挂马服务器来将木马下载到本地，这里假设为<https://www.malicous.net/trojan_file>，在黑客对挂马服务器业务逻辑处理进行修改后，当服务器后端接收到对于trojan\_file子网页的请求，首先会被拦截器拦截，拦截器含有的业务逻辑会先将原始木马文件生成对抗性样本，然后，拦截器的业务逻辑会删除掉上一次生成的对抗性样本，使用新生成的对抗性样本对其覆盖，当对抗性操作结束后，拦截器放行请求，相对应的trojan\_file\_servlet对http请求进行处理，返回301状态码进行重定向操作，定向到挂马服务器上已经经过对抗性模型处理后的对抗性样本，在这里假设为<https://www.malicous.net/trojan1_modified.exe> ，服务器返回静态对抗性木马trojan1\_modified.exe给木马下载器，随后，木马下载器将该文件保存到受害者的计算机上以备黑客后续入侵使用。尽管反病毒软件可能识别到了下载行为，但是下载的文件因为自身是静态分析对抗性样本，这会导致反病毒软件可能认为下载的文件是无恶意的文件。且木马下载器自身几乎不带有主动破坏计算机的恶意行为，只作为下载和启动其它木马的桥梁。此时，反病毒软件更多情况是认为木马下载器类似Motrix等的小型下载工具或者是正常程序的补丁更新，同时因为下载产生的数据流量不会过大，难以被判断为网络攻击。此时，相当于木马下载器完成了植入阶段的过程，在受害者的计算机上从黑客的挂马服务器上成功完成了下载木马程序。随后，将可能进入孵化期，上线期，攻击期[41]。

幸运的是，反病毒软件可以根据网站安全可信度，并将某个IP或某个域名及其子域名下的所有内容划分为黑名单，禁止用户和应用程序访问。但不容忽视的是，域名和IP安全性鉴定仍然需要时间。虽然开始时木马下载器变种在植入阶段选择下载能逃逸反病毒软件静态分析的对抗型木马，避免自身因为下载恶意程序被反病毒软件直接判断为恶意软件，由此逃逸反病毒软件的检测，但反病毒软件随后仍然可以通过判断高风险http请求拦截木马下载器的危险请求，相比之下，基于静态分析对抗性的木马下载器变种危害性较小且反制难度小于感染型病毒变种对抗性样本。但不可忽视的是，黑客可以通过上述的变种挂马服务器，导致每个被木马感染的计算机的木马都是具有不同的Hash值且产生变种木马的原始木马样本被经过修改的操作组合不同，仅仅对单个原始木马程序经过前文中10种行为（Action）和16次step就可能存在10^10以上量级的变种，原始木马程序庞大的变种可以达到干扰甚至摧毁云查杀威胁情报的共享机制。

因此，在特定的条件下，如过度依赖本地静态检测忽视动态上传上报可疑文件、过度依赖云查杀的动态检测沙箱和云端病毒库遭遇网络中断、反病毒软件无条件直接信任某些公司签发的安全证书，很大可能会导致静态规避攻击绕过杀毒软件的防御，构成实际威胁。

4.2.6 静态对抗性样本对于动态检测技术的影响

尽管本实验的方案无法绕过动态检测，但静态规避研究仍具有重要价值，可以为用户执行潜在的恶意软件提供预判防御。动态规避属于另一个研究方向，目前shellcode的逃逸检测上已经出现了动态规避，通过异或加密shellcode后封装在C/Python代码的数组中，在执行时解密即可实现动态规避。然而动态检测相比静态检测，对某些恶意软件的检测效果并不佳，因为有些恶意软件会在运行时试图破坏反病毒软件，例如关闭相应的防御服务，加载驱动让自己常驻内存来对抗反病毒软件等操作，恶意软件针对反病毒软件的破坏行为需要反病毒软件的一些自我保护机制来应对，例如采用Intel-VT虚拟化技术（目前已被360安全卫士采用），HVCI内存防护技术。但遗憾的是，反病毒软件使用Intel-VT虚拟化技术会对用户运行VMware workstation、Virtual Box等虚拟机软件的流畅造成影响。同时，反病毒软件仍然需要考虑到许多服务器和个人电脑使用的老旧计算机硬件和操作系统官方宣布已经过时不再支持维护的操作系统，例如Windows2000、Windows XP操作系统，以及过时的Windows server操作系统Windows Server 2003，这些CPU和操作系统，由于发行时间较早，当时没有考虑到后续2005年之后Intel和AMD硬件厂商推出的硬件虚拟化技术，因此反病毒软件通过虚拟化技术的自我保护在这些过时系统上无法成功实现。尽管这些过时不再被微软支持维护的Windows操作系统的使用者在逐渐减少，然而，对于恶意攻击者而言，攻击这些微软不再提供安全更新和漏洞修复的系统相对于攻击Windows10和Windows11这种仍然被长期支持的操作系统而言更容易，也是部分恶意攻击者更加青睐的目标，因为目前几乎没有第三方机构愿意为这些过时的操作系统提供安全更新服务，这些过时的操作系统一旦被恶意攻击者发现漏洞并且针对漏洞编写木马程序，后果相比仍然被支持提供安全更新的操作系统是灾难性的。不能按照常理认为更多用户选用被支持提供安全更新的系统，因此恶意攻击者更倾向于攻击用户数更多的操作系统，事实上，仍有大量计算机由于数据迁移困难，硬件不合格，软件和驱动的支持问题，没有淘汰掉过时的操作系统，而这些操作系统有可能是木马和勒索病毒利用Windows通用漏洞攻击下发生的重灾区，在过去几年前的永恒之蓝漏洞就被勒索病毒WannaCry利用445端口针对几乎所有Windows操作系统的攻击。

因此，尽管动态检测对于静态规避类对抗型恶意程序样本对于计算机的攻击确实具有一定的阻止作用，且在阻止对抗恶意程序的运行时的恶意行为起到了重大作用，但仍然可能因为恶意程序对反病毒软件的破坏导致反病毒软件的动态检测丧失功能。且因为防御生态的割裂性，杀毒软件的自我保护功能在某些过时操作系统能否有效仍然是不确定的。

4.3 实验建议

4.3.1 对于相关研究人员

本研究使用对抗样本对采用机器学习技术的病毒查杀引擎进行黑盒测试，但需要注意，对某些特征的修改和删除，是否是修改容易被规避的脆弱特征、容易损坏原有恶意文件恶意行为的特征，例如机器码，本研究中尽管代码中保留了修改机器码的函数，但最终并未将其封装为操作函数，而是将其弃用，因为修改机器码在测试中并未通过，测试时使用了一个输出HelloWorld的C语言机器码为X86-64 PE可执行程序，在修改机器码后，导致程序无法执行、以及一些容易导致程序被直接查杀的特征，例如UPX加壳，在测试中，只采取UPX加壳，QVM引擎会直接判定该程序是恶意程序，甚至被UPX加壳的文件只是毫无危害的HelloWorld C语言程序。而采用UPX加壳以及多种方式混合，QVM引擎认为该程序是恶意程序的概率会大幅下降。同时，证明修改后的样本对计算机仍然有危害是至关重要的，但需要注意测试的安全问题，因为几乎没有一种杀毒软件能100%的概率查杀静态处理后的样本，即使在测试中使用虚拟机，仍然可能会存在虚拟机逃逸问题，例如VMware WorkStation的漏洞CVE-2023-34048和CVE-2023-34056，测试前尽量对重要数据进行备份以及设置虚拟机还原点，防止运行恶意程序危害到实体机的意外发生。

4.3.2 对于反病毒软件厂商

本研究表明静态分析器容易被攻破，虽然目前没有出现前文中提及到的感染型病毒变种对抗性样本这种对静态分析针对性逃逸的流行性恶意软件，但规避静态分析的研究仍然具有价值，这也是为什么EMBER，MalConv等静态分析模型日益被关注的原因[15]，因为静态检测属于对恶意软件的第一道防线，保证能在用户执行恶意软件前就识别出恶意软件并且清除。

此外尽量不要过度依赖于本地病毒库的Hash值匹配病毒扫描策略，因为静态规避样本相比原样本Hash值不再相同，且达成静态检测规避的处理步骤不同，也会使不同的静态规避样本的Hash值不相同。同时，对某些异常PE可执行文件，例如尾部具有过长的字符串，可以考虑将它们判断为代码混淆器，疑似风险、不受欢迎文件提示给用户，并且上传到反病毒软件的云安全中心进行分析，而不是直接认为是无危害文件，尽管这样来说会有误报的可能，但是可以对抗一部分的规避静态分析的对抗性样本。例如ClamAV反病毒软件中允许开启的检测潜在不受欢迎程序（Potentially Unwanted Applications PUA）选项，虽然会误报一些用于计算机管理员监控计算机状态的网络嗅探、筛选工具和产生大量流量使用P2P网络的下载器程序，以及一些远程登录程序，JavaScript、ActiveX等类似语言编写的脚本、PC Hunter这种加载了驱动用于杀死进程的通用系统工具，但能检测出更多潜在恶意程序。此外，需要注意加强反病毒软件的自我保护功能和动态行为监测机制。同时，反病毒软件厂商也需要注意静态病毒库和动态病毒库的存放问题，本地可以尽量少存放静态病毒库，只存放一些近期流行的恶意程序样本，并且一段时间进行本地病毒库更新，这样可以为用户节省磁盘空间；而动态高风险动态行为检测库最好存放在本地，这样即使在网络环境不佳或者是离线状态，仍然能拦截病毒的运行，防止病毒危害计算机和造成用户损失。同时，云端病毒库可以对某些Hash值或者携带某些有效签名的程序的动态高风险行为设置白名单，例如一些签名注册工具。上述操作在网络正常的情况下，可以在经过本地高风险行为检测后，再进行云端查询判断程序是否是恶意程序，尽管会有离线模式下无法连接到云端病毒库导致误报某些非恶意软件的敏感操作的问题，但是可以很大程度上规避恶意程序运行造成的损失。在本实验中使用的Clam AV反病毒软件是基于本地离线病毒库，需要手动更新，大约有630MB大小。

4.4 本章小结

本章介绍了实验数据的具体来源以及实验环境配置，具体模块和具体功能的细节实现，采取了5组样本集进行实验并介绍对实验数据的评估指标，以及与psp-mal模型实现对比，同时说明对操作集的扩展在一定程度上可以提高样本免杀率。并且分析了静态对抗性样本对于云查杀和动态检验的一些影响，最后基于逃逸率结果提出了对于静态查杀对抗的研究人员以及反病毒软件厂商的一些建议，以及对于未来可能潜在摧毁云查杀机制的自我突变的对抗性样本的预测。

结　论

本实验中，笔者使用了Sarsa算法构建了一个强化学习的静态查杀规避处理框架，用于生成对一些黑盒的反病毒软件和反病毒引擎的对抗性扰动静态规避攻击样本，动作选择可以认为是多臂老虎机问题，在有限的尝试次数下探索较好的行为，尽量规避无效动作，降低修改总次数来实现规避概率最大化。实验表明，处理前后在Clam AV和火绒杀毒下检出率变化较小，这可能是因为这两款杀毒软件自带一定的抗扰动能力，能对抗本实验中对于恶意软件的针对静态检测的扰动处理，然而360杀毒的云查杀引擎在没有云上传功能下，对处理后的恶意软件样本查杀率很低。实验表明了，在采用相似检测机制的反病毒软件之间，该模型具有可迁移攻击特性，且能推测出QVM反病毒引擎的抗扰动能力较弱，说明部分基于机器学习的反病毒引擎仍然有待升级。本实验的对抗性样本生成强化学习框架在基于原有的GYM-Malware，PSP-Malware，Mab-Malware基础上，尝试了新的强化学习算法，并且降低了模块之间的耦合性，便于后续的研究人员利用已有的研究成果创建新的强化学习框架。

同时本实验中也存在着一些不足：

（1）使用的机器性能问题，为避免程序崩溃，Sarsa算法允许的最大步数上限不能调的很高，导致可能存在一些样本，这些样本再经过一些操作也能达到规避静态检测，但智能体在指定步数之内会判定扰动产生的对抗性样本无效。

（2）最后，本实验尚未考虑到Windows下其它恶意程序的对抗性样本生成，例如恶意shellcode，恶意JavaScript代码等。

参考文献

[1]SONICWALL. SonicWall cyber threat report 2023[R/OL]. (2023)[2024-03-15]. https://www.sonicwall.com/medialibrary/en/white-paper/2023-cyber-threat-report.pdf.

[2] GOPINATH M, SETHURAMAN S C. A comprehensive survey on deep learning based malware detection techniques[J]. Computers & Security, 2023, 124: 102996. DOI:10.1016/j.cose.2022.102996.

[3] ANDERSON H S, ROTH P. Ember: an open dataset for training static PE malware machine learning models[J/OL]. arXiv preprint arXiv:1804.04637, 2018.

[4] DAHL G E, STOKES J W, DENG L, et al. Large-scale malware classification using random projections and neural networks[C]//2013 IEEE International Conference on Acoustics, Speech and Signal Processing. Vancouver, BC, Canada: IEEE, 2013: 3422–3426.

[5] RAFF E, BARKER J, SYLVESTER J, et al. Malware detection by eating a whole exe[C]//Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence. New Orleans, LA, USA: AAAI, 2018.

[6] RIECK K, TRINIUS P, WILLEMS C, et al. Automatic analysis of malware behavior using machine learning[J]. Journal of Computer Security, 2011, 19(4): 639–668.

[7] 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). Fajardo, PR, USA: IEEE, 2015: 11–20.

[8] GENG J X, WANG J F, FANG Z Y, et al. A survey of strategy-driven evasion methods for PE malware: transformation, concealment, and attack[J]. Computers & Security, 2024, 137: 103595. DOI:10.1016/j.cose.2023.103595.

[9] ANDERSON H S, KHARKAR A, FILAR B, et al. Learning to evade static PE machine learning malware models via reinforcement learning[J/OL]. arXiv preprint arXiv:1801.08917, 2018.

[10] DEMETRIO L, BIGGIO B, LAGORIO G, et al. Functionality-preserving black-box optimization of adversarial Windows malware[J/OL]. arXiv preprint arXiv:2003.13526, 2020.

[11] QUIRING E, MAIER A, RIECK K. Misleading authorship attribution of source code using adversarial learning[C]//28th USENIX Security Symposium (USENIX Security 19). Santa Clara, CA, USA: USENIX Association, 2019: 479–496.

[12] ZHAN D Z, BAI W, LIU X, et al. PSP-Mal: Evading malware detection via prioritized experience-based reinforcement learning with Shapley prior[C]//Proceedings of the 39th Annual Computer Security Applications Conference. Orlando, FL, USA: ACM, 2023: 580–593.

[13] ANDERSON H S, KHARKAR A, FILAR B, et al. Evading machine learning malware detection[C]//Black Hat USA 2017. Las Vegas, NV, USA: 2017.

[14] SONG W, LI X, AFROZ S, et al. Mab-malware: a reinforcement learning framework for blackbox generation of adversarial malware[C]//Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security. Nagasaki, Japan: ACM, 2022: 990–1003.

[15] FANG Z, WANG J, LI B, et al. Evading anti-malware engines with deep reinforcement learning[J]. IEEE Access, 2019, 7: 48867–48879. DOI:10.1109/ACCESS.2019.2908035.

[16] LABACA-CASTRO R, FRANZ S, RODOSEK G D. Aimed-RL: exploring adversarial malware examples with reinforcement learning[C]//Machine Learning and Knowledge Discovery in Databases. Bilbao, Spain: Springer, 2021: 37–52.

[17] GIBERT D, FREDRIKSON M, MATEU C, et al. Enhancing the insertion of NOP instructions to obfuscate malware via deep reinforcement learning[J]. Computers & Security, 2022, 113: 102543. DOI:10.1016/j.cose.2021.102543.

[18] GOODFELLOW I J, SHLENS J, SZEGEDY C. Explaining and harnessing adversarial examples[C]//3rd International Conference on Learning Representations. San Diego, CA, USA: ICLR, 2015.

[19] CARLINI N, WAGNER D. Towards evaluating the robustness of neural networks[C]//2017 IEEE Symposium on Security and Privacy. San Jose, CA, USA: IEEE, 2017: 39–57.

[20] MOOSAVI-DEZFOOLI S M, FAWZI A, FROSSARD P. DeepFool: a simple and accurate method to fool deep neural networks[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, NV, USA: IEEE, 2016: 2574–2582.

[21] KOLOSNAJABI B, DEMONTIS A, BIGGIO B, et al. Adversarial malware binaries: evading deep learning for malware detection in executables[C]//2018 26th European Signal Processing Conference (EUSIPCO). Rome, Italy: IEEE, 2018: 533–537.

[22] AL-DUJAILI A, HUANG A, HEMBERG E, et al. Adversarial deep learning for robust detection of binary encoded malware[C]//2018 IEEE Security and Privacy Workshops (SPW). San Francisco, CA, USA: IEEE, 2018: 76–82.

[23] KREUK F, BARAK A, AVIV-REUVEN S, et al. Deceiving end-to-end deep learning malware detectors using adversarial examples[J/OL]. arXiv preprint arXiv:1902.03954, 2019.

[24] SUCIU O, COULL S E, JOHNS J. Exploring adversarial examples in malware detection[C]//2019 IEEE Security and Privacy Workshops (SPW). San Francisco, CA, USA: IEEE, 2019: 8–14.

[25] LIU X, ZHANG J, LIN Y, et al. ATMPA: attacking machine learning-based malware visualization detection methods via adversarial examples[C]//Proceedings of the International Symposium on Quality of Service. Hong Kong, China: ACM, 2019: 1–10.

[26] PARK D, KHAN H, YENER B. Generation & evaluation of adversarial examples for malware obfuscation[C]//2019 18th IEEE International Conference on Machine Learning and Applications (ICMLA). Boca Raton, FL, USA: IEEE, 2019: 1283–1290.

[27] STATISTA. Global market share held by operating systems for desktop PCs, from January 2013 to February 2024[EB/OL]. (2024)[2024-03-15]. https://www.statista.com/statistics/218089/global-market-share-of-windows-7/.

[28] AV-TEST INSTITUTE. AV-TEST statistics[EB/OL]. (2023)[2023-12-01]. https://portal.av-atlas.org/malware/statistics.

[29] RUMMERY G A, NIRANJAN M. Online Q-learning using connection systems[R]. Cambridge, UK: University of Cambridge, 1994.

[30] SUTTON R S, BARTO A G. Reinforcement learning: An introduction[M]. 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[31] VAN SEIJEN H, VAN HASSELT H, WHITESON S, et al. A theoretical and empirical analysis of Expected Sarsa[C]//2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning. Nashville, TN, USA: IEEE, 2009: 177–184.

[32] GANGER M, DURYEA E, HU W. Double Sarsa and double expected Sarsa with shallow and deep learning[J]. Journal of Data Analysis and Information Processing, 2016, 4(4): 159–176. DOI:10.4236/jdaip.2016.44013.

[33] YOKOYAMA A, ISHII K, TANABE R, et al. SandPrint: fingerprinting malware sandboxes to provide intelligence for sandbox evasion[C]//International Symposium on Research in Attacks, Intrusions, and Defenses. Paris, France: Springer, 2016: 165–187.

[34] RUDD E M, ROZSA A, GUNTHER M, et al. A survey of stealth malware: attacks, mitigation measures, and steps toward autonomous open world solutions[J]. IEEE Communications Surveys & Tutorials, 2017, 19(2): 1145–1172. DOI:10.1109/COMST.2017.2662561.

[35] BULAZEL A, YENER B. A survey on automated dynamic malware analysis evasion and counter-evasion: PC, mobile, and web[C]//Proceedings of the 1st Reversing and Offensive-Oriented Trends Symposium. New York, NY, USA: ACM, 2017: 2. DOI:10.1145/3152480.3152483.

[36] APOSTOLOPOULOS T, KATOS V, CHOO K K R, et al. Resurrecting anti-virtualization and anti-debugging: unhooking your hooks[J]. Future Generation Computer Systems, 2021, 116: 393–405. DOI:10.1016/j.future.2020.10.013.

[37] Anonymous. Parasitic malware: the resurgence of an old threat[J]. Network Security, 2008(3): 15–18. DOI:10.1016/S1353-4858(08)70024-2.

[38] PAPPAS V, POLYCHRONAKIS M, KEROMYTIS A D. Smashing the gadgets: hindering return-oriented programming using in-place code randomization[C]//2012 IEEE Symposium on Security and Privacy. San Francisco, CA, USA: IEEE, 2012: 601–615. DOI:10.1109/SP.2012.41.

[39] PAVITHRAN J, PATNAIK M, REBEIRO C. D-TIME: distributed threadless independent malware execution for runtime obfuscation[C]//13th USENIX Workshop on Offensive Technologies (WOOT 19). Santa Clara, CA, USA: USENIX Association, 2019.

[40] ISPOGLOU K K, PAYER M. malWASH: washing malware to evade dynamic analysis[C]//10th USENIX Workshop on Offensive Technologies (WOOT 16). Austin, TX, USA: USENIX Association, 2016.

[41] XIE J, LI S, YUN X C, et al. HSTF-Model: an HTTP-based Trojan detection model via the hierarchical spatio-temporal features of traffics[J]. Computers & Security, 2020, 96: 101923. DOI:10.1016/j.cose.2020.101923.

附　录

本实验中关于强化学习的内容，部分参考了赵世钰老师的《强化学习的数学原理》中文译本，其英文原本位于<https://github.com/MathFoundationRL/Book-Mathematical-Foundation-of-Reinforcement-Learning>，感兴趣的读者可以去学习其它的强化学习算法，例如值函数方法和策略梯度方法，以及演员-评论家方法，本文限于篇幅以及本实验中没有涉及到这些强化学习算法在此不予给出，读者可以参考该中文译本来进行对强化学习的进一步学习。

本实验的相关代码遵循MIT开源协议，笔者已经将代码上传到了GitHub上，项目地址是：<https://github.com/Oldmemory1/GraduateDesign/tree/master>

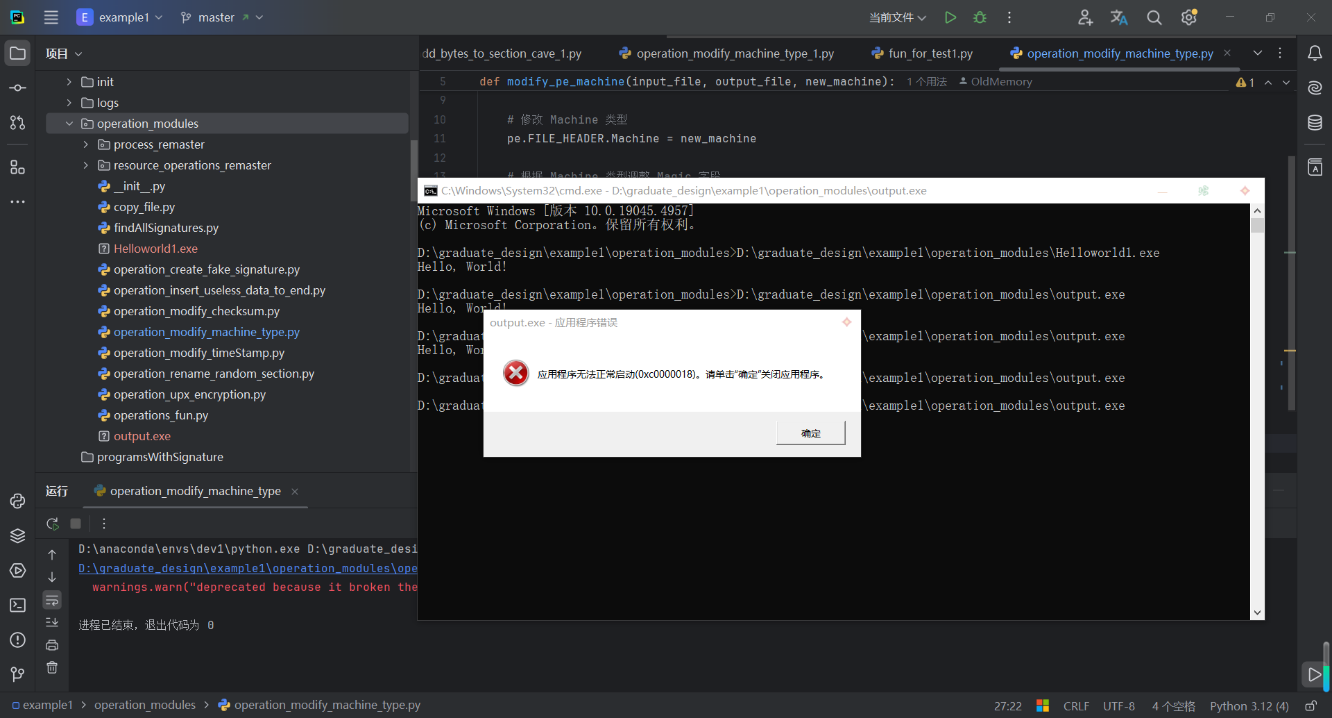
如图-附录-1所示,因为某些操作导致修改后的恶意程序损坏失效时，无法被执行。

图-附录-1 操作导致文件损坏

致　谢

值此论文完成之际，首先向笔者的指导老师-田东海表以感谢，在开题报告中的指导为本实验理清了思路，并每周予以指导给笔者提供可行的思路，且在本实验中首次强化学习对抗性样本生成框架，在样本即使处理后，反病毒软件对样本集的查杀率仍偏高的的问题上给予了方向指引，笔者在此方向上进行扩展研究，才得知有部分反病毒软件已经内置了对于对抗性样本干扰的抵抗，进而排查出原因。同时，指导老师为笔者提供了一些开源的强化学习对抗性样本生成模型的已有论文予以参考，从而笔者才能在前辈研究人员的基础上完成本实验。

随后，也向邓永琪学长表以感谢，帮助笔者在Ubuntu20.04下解决了numpy包的版本问题以及实验测试用样本的来源问题，从而使笔者能成功从PSP-Malware项目中提取并调试修改PE可执行文件的函数。也在此为前辈们的强化学习对抗性样本框架和pefile、lief库的作者表以感谢，如果没有他们的实验和Python库的开发，那么对PE可执行文件的修改将会相当困难，尽管某些前辈们的强化学习对抗性样本框架的部分函数难以被拆解提取，但对本实验而言，仍然提供了相当大的帮助。在本论文的编写中，由衷感激笔者的朋友们，雨涵，瑶光，默然清梦，白露清瑶，张修豪等人在精神上给予的鼓励（排名不分先后），以及张全新老师，成雨蓉老师，陆慧梅老师，黎有琦老师在笔者大三时对于本实验涉及到的汇编以及PE可执行程序知识的相关详细讲解。