基于优化遗传算法的恶意软件漏洞检测方法

陈宇翔[1]
CHEN Yuxiang

摘 要 恶意软件的攻击技术不断进化，呈现出多态性与变形性的显著特征。这些恶意软件能够灵活地对自身代码结构进行调整，致使每次发起攻击时的形态都各不相同，让人难以预测。传统检测方法依赖静态漏洞评分标准，难以准确评估恶意软件变化攻击下的漏洞威胁，导致检测漏洞数量与实际不符，准确性低。为此，文章提出一种基于优化遗传算法的恶意软件漏洞检测方法。该方法将链路报文进行转化，并与端口报文相结合，通过考虑信息流传输过程中产生的叠加流量，获取恶意软件最终信息流。采用运行码频率提取方法，从信息流中筛选出高频运行码特征，快速定位与恶意软件行为最相关的关键特征。将这些关键特征作为优化遗传算法的初始种群，计算漏洞存在时的最优个体值，并与攻击成功的经验阈值比较，评估漏洞威胁程度。通过实时检测恶意软件攻击的风险态势，及时获取漏洞情况。实验结果表明，在特定时间点（5、11、20、26 s），软件响应频率显著突变，且检测出的漏洞数量与实际完全吻合。这表明此方法能有效检测漏洞，准确获取恶意软件漏洞数量，显著提升检测准确性。

关键词 遗传算法；优化算法；恶意软件；漏洞检测；高频运行码特征

doi：10.3969/j.issn.1672-9528.2025.03.022

0 引言

随着信息技术的飞速发展和互联网技术的广泛应用，恶意软件已成为网络安全领域面临的一大挑战。恶意软件利用软件漏洞进行传播、窃取数据、破坏系统等恶意活动，给个人、企业乃至社会都带来了严重的安全风险。随着恶意软件技术的不断演进，其复杂性和隐蔽性不断提升，使得防御工作越发困难。传统的恶意软件检测方法，如特征码匹配和行为模式分析，在面对不断变异和进化的恶意软件时，已逐渐显露出其局限性。恶意软件开发者通过修改代码、混淆技术或利用新的漏洞等手段，使得恶意软件能够绕过传统的检测机制，继续在网络中传播和危害。为了有效应对这一挑战，准确且高效的漏洞检测方法显得尤为重要。这类方法能够在恶意软件发动攻击之前，及时发现并修复系统中的漏洞，从而显著提升系统的安全防护能力。这不仅能够有力保护个人隐私和企业的商业机密，更能确保国家关键信息基础设施的安全稳定运行。因此，深入研究恶意软件漏洞检测方法，对于提升网络安全防护水平、保障信息系统的持续稳定运行具有深远的意义。

目前，已有多种针对恶意软件漏洞检测的方法被提出并应用于实践中，例如，文献 [1] 提出基于学习的源代码漏洞检测方法，该方法首先识别并定义了漏洞的特征，随后将这些特征输入到机器学习算法中进行学习，以识别源代码中的漏洞。然而，由于该方法所定义的漏洞特征局限于已知的攻击模式，当新型恶意软件出现且其攻击特征与已定义特征不完全匹配时，检测方法可能无法准确识别这些新攻击，从而导致检测漏洞数量与实际不符，准确性下降。另一方面，文献 [2] 提出了一种在确定约束条件下的机器学习框架漏洞检测方法，即 ConFL。该方法能够生成通过合法性校验的有效输入数据，这些数据能够触发框架内部更深层次的代码逻辑，从而增加检测到潜在漏洞的机会。然而，ConFL 方法依赖于从框架源代码中自动提取的约束来生成有效输入数据，在面对新型或未知漏洞的攻击时，生成的输入数据可能无法全面覆盖所有可能的攻击场景，导致一些实际存在的漏洞被遗漏，进而降低了检测的准确性。

针对上述问题，本文提出了一种基于优化遗传算法的恶意软件漏洞检测方法，并通过实际实验进行了深入分析与验证。

1 恶意软件漏洞检测

当恶意软件实施多阶段攻击时，会逐步调整其行为模式以适应不同的攻击阶段，并试图躲避检测。在攻击的初期阶段，尽管恶意行为的频率相对较低，但随着攻击的逐步推进，系统区域或网络地址中的信息流动规模和方向会发生显著变化。通过信息流检测手段，可以持续追踪这些动态变化，从

1. 江西财经大学现代经济管理学院 江西九江 332020

而及时发现恶意软件的攻击行为。与此同时，对相关信息流的特征进行提取，并应用优化遗传算法进行针对性地检测，能够进一步提升检测的准确性和效率。

1.1 获取恶意软件的最终信息流

在网络环境下，恶意软件利用信息流机制，通过特定节点或端口未经授权地传输数据，形成系统安全的重大威胁，这些隐匿于复杂网络环境中的漏洞难以轻易发现。为了更有效地获取恶意软件的信息流漏洞，本文提出了一种新方法，该方法将链路报文与端口报文相结合，并全面考虑叠加流量的影响。这种方法能够深入捕获恶意软件在网络传输过程中的更深层次信息流。这些信息流为恶意软件漏洞的检测提供了有力的支持。

在网络环境下，可以参照树结构来具体分析恶意软件中的危险状况，通过解析该结构来获取精确的漏洞信息。树结构能够清晰地展示各节点端口间信息流的传播路径，其中信息的起点位于树结构的根节点[3]。为了获取精确的漏洞信息，首先将恶意软件中的信息流转化为一维列矩阵表示 $\boldsymbol{N}$，具体形式为：

$$\boldsymbol{S}_{N\times1}=[s_1,s_2,\cdots,s_N]^{\mathrm{T}} \tag{1}$$

式中：s_N 为节点在单位时间内发送的信息流数量。根据信息流的具体数量，结合发送时间可以计算出信息流的发送频率[4]。

在实际操作中，恶意软件的信息流漏洞需要经过所有节点进行运作，才能最终完成节点信息流量的统一。在这一过程中，上级节点转发时可能引发的叠加流量是一个不可忽视的因素。为了准确计算叠加流量，采用了多拓扑生成算法，该叠加流量的计算方式为：

$$Q(\alpha)=xS_{N\times1}Q(a+tQ_1) \tag{2}$$

式中：Q_1 为信息流漏洞的数量；a 为节点数量；x 为经过节点信息流的节点数目；t 为各节点信息流漏洞产生的时间[5]。

当恶意软件的各个节点发送消息时，整个 $\boldsymbol{S}_{N\times1}$ 矩阵中的所有端口都会接收到相应的报文消息。根据报文处理机制，选取最优节点设定为主节点，并将其导入到建立的信息流模型中。通过这一模型，能够构建各节点端口的恶意软件漏洞信息数据集。若恶意软件存在多个漏洞，则信息流模型 $\boldsymbol{R}_{p\times n}$ 会相应地扩展，以反映所有漏洞的信息。其中，p 表示恶意软件存在漏洞的数量。为了将恶意软件的漏洞与链路和端口相对应，设定了端口的数量 k，并计算链路数量。链路数量的计算公式为：

$$d=\frac{kQ(\alpha)}{2} \tag{3}$$

式中：d 为链路的数量。通过链路数量进行转化链路报文，将端口报文与链路报文进行结合，再加上信息流传输过程中产生的叠加流量，可以获取软件最终的恶意软件信息流，计算公式为：

$$F_{ij}=\frac{dn_{ij}}{\sum_{k=0}^{m}n_{kj}} \tag{4}$$

式中：n_{ij} 表示节点 i 在恶意软件信息中出现的次数；$\sum_{k=0}^{m}n_{kj}$ 表示节点 j 中的恶意软件信息总数。

1.2 提取恶意信息流特征

在获取恶意软件信息流后，恶意软件会通过多种途径（如网络窃取用户数据和USB设备接收外部控制指令）进行活动。这些活动会导致信息流特征受到多种来源的干扰，呈现多样化。特别是当恶意软件通过 USB 设备接收控制指令时，信息流中会混入来自 USB 设备的信息流，进一步增加了特征的复杂性。因此，为从复杂的信息流中准确识别出恶意代码，需要对信息流特征进行提取[6]。为此，本文使用 *N-L* 方法从加密文件中提取特征图，生成矩阵数据。通过计算候选词概率、特征权重和信息熵，选择特征权重最小的候选词作为特征对象。利用运行码频率提取方法，从恶意信息流中筛选出高频运行码特征，这些特征通常与恶意软件的核心功能和攻击行为紧密相关。通过关注这些高频特征，可以更有效地识别恶意软件的存在和潜在威胁[7]。

在恶意软件信息流特征的提取中，首先在训练集上统计信息流中的 L 类候选词数量。针对其中一个候选词 a 进行分析，计算其在数据集中出现的概率为：

$$P_A=\frac{a}{nF_{ij}} \tag{5}$$

式中：a 为在数据集中出现的频次；n 为所有候选词的数量。接着，计算候选词 a 在具体类别 L 中出现的概率为：

$$P_B=\frac{l_a}{nl} \tag{6}$$

式中：l_a 为候选词出现在类别中的次数；l 为样本的类别数量。然后，计算候选词 a 的特征权重：

$$w(a)=\frac{|P_A-P_B|}{P_A} \tag{7}$$

式中：w 为权重特征。通过上述公式计算得出 $w(a)$ 的特征权重值。为了评估样本集合的精准度，设定了信息熵的概念。在决策树中，根据样本集中的第 L 类数据所占的比例，选择特征词 a 的最大特征权重来对数据集进行划分，获取特征词的信息增益[8]。通过计算每个类别中所有候选词的特征权重，并对这些数据进行加和，可以得到最终的特征权重总值。然后，针对特征权重具体数据进行排序，选择特征权重最小的候选词作为特征对象。利用这个特征对象对信息流进行特征提取，得到的信息流特征可以表示为：

$$e=-\sum l_k \log_2 l_k w(a) \tag{8}$$

式中：e 为信息流特征，e 值越小，表明样本数据精准度越高，特征越准确。

在采用运行码频率提取方法进行特征提取时，根据运行码的频率选择出高频运行码作为最终目标。这些高频运行码通常与恶意软件的核心功能和攻击行为紧密相关，因此能够获得较好的提取效果。通过关注这些高频特征，可以更有效地识别恶意软件的存在和潜在威胁。

1.3 基于优化遗传算法的漏洞检测

恶意软件通过改变函数调用顺序、插入冗余代码、修改逻辑结构等多种方式进行变异，这些变异手段极其复杂且多变。这种变异使得传统的检测方法难以准确识别并应对。遗传算法作为一种优化搜索算法，在漏洞检测领域展现出了一定的潜力，但面对恶意软件的多样变异，其快速适应和策略调整能力仍有待提升。为此，本文提出了一套基于优化遗传算法的漏洞检测方法，该方法将高频运行码特征作为关键输入，这些特征源自恶意软件信息流，与核心功能和攻击行为紧密相关，有助于精确搜索漏洞。在基于路径和分支覆盖的基础上，生成遗传算法的适应度函数，并对遗传算子进行更新。最后，将遗传算法与随机森林等机器学习算法相结合，构成一种整合式的漏洞检测算法，实现对恶意软件的快速准确检测 [9]。具体检测过程如下：

（1）对问题软件的信息流进行随机森林聚类分析，根据信息流特征将数据点划分为不同的簇。然后，从每个簇中选择代表性的个体作为遗传算法的初始种群。这样可以确保初始种群在整个问题空间中具有更广泛的分布，避免初始种群过于集中在某些局部区域。

（2）初始化种群中的每个染色体，以特征数量作为目标函数，将评估过的信息流特征子集保存在数据库中，防止数据丢失。

（3）采用非支配排序方法对种群进行排序，并通过遗传算子生成后代种群。其公式为：

$$\delta = o\cdot \mathrm{ranh}(lk) \tag{9}$$

式中：o 为染色体；ranh 为自定义函数。

（4）在选择过程中，优先考虑拥挤度较低的染色体，经过随机变异，使得父种群与新生的种群进行汇总，从而生成一个新的染色体。在此过程中，对新的染色体进行自主排序，选出排名前一半的染色体构成下一代种群。这样通过平衡算法的搜索能力，选择排序中的首位染色体作为最优个体，计算公式为：

$$k=\frac{t(D+1)}{g} \tag{10}$$

式中：D 为漏洞标识；g 为恶意软件攻击类型。

（5）定义攻击主要影响因素，并给出攻击成功的经验阈值 λ，其公式为：

$$\lambda=\frac{1}{y}\sum_{j=1} w+\frac{p}{wd} \tag{11}$$

式中：y 为攻击成功概率；p 为关联度。

（6）将软件漏洞存在时的最优个体值与攻击成功的经验阈值进行比较，以此确定漏洞对软件安全的威胁程度 [10]。如果最优个体值 k 小于攻击成功的经验阈值 λ，则表明软件无漏洞；反之，如果最优个体值 k 大于阈值 λ，则需进一步分析恶意软件攻击 Δt 的风险态势，确定漏洞检测结果。具体的检测结果可以表示为：

$$H=\frac{E\Delta t}{\lambda e}+\frac{k\delta}{E} \tag{12}$$

式中：E 为关联分析值。通过这一流程，可以对软件进行实时的漏洞检测，实现持续的安全监控和更新，以有效应对不断变化的威胁环境。

2 实验测试与分析

2.1 搭建实验环境

为了验证本文所提检测方法的有效性，在研究过程中结合网络环境，配置了实验平台环境，具体配置如表 1 所示。

表 1 实验环境

配置	参数
处理器	R7
显卡	RTX 1020
内存	8 GB
硬盘	SSD
Python 集成开发环境	PyCharm
Python 版本	3.1.3

在实验中，采用了在漏洞检测领域广泛使用的 SAT 数据集，该数据集包含了各类软件漏洞代码及其相应的补丁信息。文中选用了 C++ 测试用例样本组成的测试套件，这些样本能够代表通用的源代码类型。从数据集中挑选了 1 200 个文件，以此构建测试所用数据集。在此过程中，随机选择了部分数据集用于训练模型，剩余则作为测试集使用。为了确保实验的严谨性，还在数据集中额外添加了 5 个漏洞样本。

2.2 模型结构

在训练阶段，为了优化模型性能，设置了多个关键参数。采用交叉熵为损失函数，以精准衡量模型输出与实际标签之间的差距；同时，选用高效的 Adam 优化器来更新模型参数。设定训练轮次确保模型充分学习。此外，还设定了初始学习率，并在每经过 5 轮训练后，对学习率进行逐步衰减，以促进模型更好地拟合训练集数据，其参数具体如表 2 所示。在训练过程中，为了提升训练速度和效果，对已有数据集进行降采样处理，并融入了新数据。通过迭代训练，不断读取已有模型的数据信息，并基于新数据对数据集进行扩充。经过

多次迭代训练后，保存模型并输出结果。这一训练过程能够提升模型的检测能力和准确性。

表 2 模型训练超参数

项目	参数
损失函数	交叉熵损失函数
优化器	Adam
训练轮次	150
Batch size 大小	120
训练平台	RTX 1020
初始学习率	0.001

2.3 验证指标

在漏洞检测环境下，扰动是对目标系统（如软件、网络等）的输入、配置或者状态进行微小改变的操作，因此实验中采用扰动率作为衡量这种改变程度的指标进行实验验证，计算公式为：

$$D=\frac{m_l}{o_l} \tag{13}$$

式中：m_l 表示对软件进行调用序列修改的函数个数；O_l 表示修改前调用序列的函数个数。

2.4 结果分析

在测试阶段，设定了 30 s 的测试时间窗口，以评估本文方法在限定时间内能够完整检测出的漏洞数量。为了直观展现测试效果，绘制了软件运行过程中的扰动率曲线，并将本文方法与文献 [3] 方法、文献 [4] 方法以及实际值进行了对比分析，对比结果如图 1 所示。

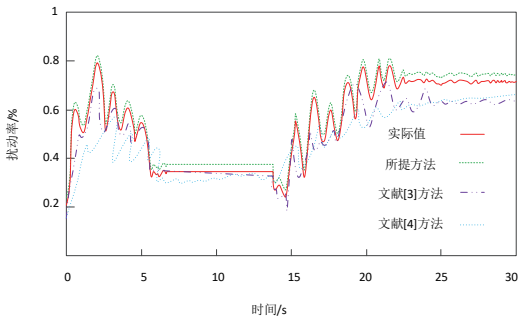

图 1 软件运行中的响应曲线

通过观察图 1 中的扰动率曲线，可以发现，在 0~0.3 s 的时间范围内，3 种方法的扰动率曲线呈现出一定的变化趋势。在对比这 3 种方法的扰动率曲线时，可以清晰地看出，本文所提出的方法的扰动率曲线在绝大多数时间点上与实际值的曲线高度吻合，这充分表明该方法在识别恶意软件及其漏洞方面具有更高的准确性。相比之下，文献 [3] 方法和文献 [4] 方法的扰动率曲线与实际值之间存在较为明显的偏差，这进一步说明了这两种方法在识别恶意软件及其漏洞时的能力相对有限。本文所提出的方法通过利用遗传算法的全局搜索能力和优化策略，结合从信息流中提取的关键特征，能够更准确地评估漏洞的威胁程度，使得该方法在对比中展现出显著优势。

3 结语

本文深入信息安全领域的漏洞检测问题，提出了一种基于优化遗传算法的恶意软件漏洞检测新方法。该方法深入解析信息流传输中的叠加流量，精确捕获恶意软件的最终信息流，并借助运行码频率提取技术，有效识别出与恶意行为密切相关的高频特征。随后，将这些关键特征用作遗传算法的初始种群，通过算法优化计算出漏洞存在时的最优解，并与实际攻击成功的经验阈值进行对比，实现对漏洞威胁程度的精准评估。实验结果显示，该方法在多个关键时间点均能准确识别软件响应频率的显著变化，且检测出的漏洞数量与实际完全相符，显著提升了检测的准确性与响应速度。本文不仅为恶意软件漏洞检测带来了新的技术视角，也为加强网络安全防护能力提供了重要的创新支撑。

参考文献：

[1] 苏小红，郑伟宁，蒋远，等．基于学习的源代码漏洞检测研究与进展 [J]. 计算机学报，2024,47(2):337-374.
[2] 刘昭，邹权臣，于恬，等．一种约束制导的机器学习框架漏洞检测方法 [J]. 计算机学报，2024,47(5):1120-1137.
[3] 董伟良，刘哲，刘逵，等．智能合约漏洞检测技术综述 [J]. 软件学报，2024,35(1):38-62.
[4] 胡雨涛，王溯远，吴月明，等．基于图神经网络的切片级漏洞检测及解释方法 [J]. 软件学报，2023,34(6):2543-2561.
[5] 嵇友晴，卢跃，潘世文，等．基于动态切片与预训练模型的代码漏洞检测 [J]. 小型微型计算机系统，2024,45(6):1529-1536.
[6] 何清林，王丽宏，陈艳姣，等．一种自动实时的物联网在野漏洞攻击检测方法 [J]. 北京航空航天大学学报，2024, 50(7): 2195-2205.
[7] 张卓，刘业鹏，薛建新，等．基于数据流传播路径学习的智能合约时间戳漏洞检测 [J]. 软件学报，2024, 35(5): 2325-2339.
[8] 杨宏宇，杨海云，张良，等．基于特征依赖图的源代码漏洞检测方法 [J]. 通信学报，2023,44(1):103-117.
[9] 徐寅森，李红艳，张子栋．基于机器学习的传感网核心节点漏洞检测仿真 [J]. 计算机仿真，2024,41(3):410-414.
[10] 潘超，吕翘楚，肖巍．基于启发式遗传算法的即时通信网络漏洞检测 [J]. 计算机仿真，2023, 40(8): 191-195.

【作者简介】

陈宇翔（1989—），男，江西南昌人，硕士，助教，研究方向：软件工程、计算机应用技术。

（收稿日期：2024-10-29）