模糊测试中基于神经网络的敏感区域预测算法研究

张羿辰[1,2], 赵磊[1,2], 金银山[1,2]

[1]武汉大学国家网络安全学院 武汉 中国 430072

[2]武汉大学空天信息安全与可信计算教育部重点实验室 武汉 中国 430072

摘要 软件漏洞是造成计算机安全问题的根本原因, 模糊测试技术由于其易扩展, 高效的特性, 是目前主流的漏洞检测技术之一。然而以往的模糊测试技术存在着对识别高结构样本失效以及盲变异效率低下的问题。针对这些问题, 本文提出了基于神经网络的敏感区域预测的模糊测试方法。该方法以输入文件的某些区域的极小改变会引起程序行为较大改变的现象为出发点, 引入了敏感区域概念, 并引入了能够学习总结数据特征的神经网络方法检测敏感区域。在检测敏感区域的基础上, 本文引入了增量学习策略, 进行了变异策略的优化, 使检测效率以及检测深度有更多提升。为了验证提出方法的有效性, 本研究在三种热门格式文件 PNG、TIFF、XML 的处理软件上进行了实验, 在模糊测试覆盖率上取得了 8%~20%的提升,从而验证了本文方法的有效性和可行性。

关键词 软件漏洞; 模糊测试; 敏感区域;神经网络

中图法分类号 TP181/TP309 **DOI 号** 10.19363/J.cnki.cn10-1380/tn.2020.01.02

Sensitive Region Prediction based on Neural Network in Fuzzy Test Algorithm Research

ZHANG Yichen[1,2], ZHAO Lei[1,2], JIN Yinshan[1,2]

[1]School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China

[2]Key Laboratory of Aerospace Information Security and Trusted Computing Ministry of Education, Wuhan University, Wuhan 430072 China

Abstract Software vulnerabilities are the root cause of computer security problems. Due to highly efficiency and easy expansion, fuzzy test becomes the most widely used vulnerabilities detection technology. However, previous fuzzy test technology can not deal with highly-structed problems, and has low efficiency on blinding mutation. For these problems, this paper proposes a fuzzy test algorithm based on sensitive region prediction via Neural Network. This method takes the phenomenon of small changing of some regions causing great change on software behavior as starting point. We use the conception of sensitive region and use Neural Network which has great performance on learning data features to detect these regions. After sensitive region detection, this paper uses enhanced learning strategy and optimizes the mutation strategy which improve efficiency and depth of detection. In order to verify the validity of the proposed method, this study was conducted on the programs dealing with three widely used format file as PNG, TIFF and XML, and itshows 8%~20% improvement on fuzzing test coverage which verifies the validity and feasibility of proposed method.

Key words software vulnerability; fuzzy test; sensitive region; neural network

1 引言

在计算机安全领域中, 软件安全漏洞问题一直以来都是一个重要的关注点。软件漏洞造成威胁的严重程度各不相同, 这取决于开发复杂性和攻击面等因素[1]。在过去的二十年中, 出现了许多软件漏洞事件, 这些软件漏洞对公司和个人都造成了重大的损害。为了强调这一问题的重要性, 本文列举了近年来的一些事件。一个广为人知的事例就是浏览器插件中的漏洞已经威胁到数百万互联网用户的安全与隐私, 例如 AdobeFlash 播放器[2]和 java[3]安全漏洞事件。开源软件中的漏洞还威胁到全球数千家公司及

通讯作者: 赵磊, 博士, 副教授, Email: leizhao@whu.edu.cn。

本课题得到国家自然科学基金(No.61672394)资助。

收稿日期: 2019-09-05; 修改日期: 2019-11-26; 定稿日期: 2019-12-09

其客户的安全。例如: HeartBleed[4]、ShellShock[5]和Apache Commons[6]。

考虑到漏洞所造成的严重危害, 学术界和工业界针对软件和信息系统的漏洞发现技术已经投入了大量的工作, 提出了许多漏洞检测技术, 例如动态技术、静态技术、符号执行和模糊测试[7]技术。

与其他技术相比, 模糊测试技术对目标的需求知识较少, 可以很容易地扩展到大型应用程序中, 已经成为最流行的漏洞检测解决方案。但是模糊测试技术仍然存在着以下几个关于测试用例方面的挑战: 1)如何获取初始种子, 常用的收集种子输入的方法包括使用标准基准、从 Internet 爬行和使用现有的 POC 样本; 2)如何生成测试用例, 即在更短的时间内覆盖更多的代码, 现有的技术有 Rawat 等人[8]提出了一种应用程序感知的灰盒模糊器 Vuzzer, Junjie Wang 等人[9]提出了一种数据驱动的种子生成解决方案 Skyfire 以及基于机器学习技术的 learn&fuzz[10]; 3)如何挑选种子, 良好的种子选择策略可以显著提高模糊效率, 帮助更快地发现更多的 bug[8,11-12], Böhme 等人[11]提出了综合静态和动态分析的 Aflfast, 还有基于定向选择策略的 Qtep[12]和 Aflgo[13]。

针对测试用例生成, 目前常用的方法是基于变异。模糊器在模糊过程中利用现有的种子输入语料库, 通过变异所提供的种子来生成输入。AFL 是目前非常经典的变异模糊器, BuzzFuzz[14]应用动态污染分析自动定位原始种子输入中影响脆弱点(即程序可能包含错误的点)使用的值的区域, SAGE[15-16]通过最大化每次符号执行运行生成的新测试输入的数量来执行模糊变异, Driller[17]以一种互补的方式将模糊和协同执行结合起来, 以发现深层缺陷。

这些基于变异的模糊方法如可以有效地模糊处理非结构化或简单结构化输入(例如图像和多媒体)的程序。然而对于处理高度结构化输入(例如 XSL 和 JavaScript)的程序, 它们会表现的很差。这是因为大多数基于变异的模糊化的错误输入在程序执行的早期阶段因未能通过语法分析而被丢弃, 使得模糊器在处理语法正确性时浪费了大量时空资源, 而在正确的漏洞点收益微小, 这严重限制了它们查找深层错误的能力。即对于有严格语法结构的程序而言, 采用这类变异方法所生成的种子往往是低质量的, 因此, 如何针对这些程序生成高质量测试用例, 是本文所关注的方面。

针对如何生成高质量测试用例的问题, 本文通过观察发现, 输入的每一个区域对于变异的增益都不是等价的, 即存在某些极小的变化会引起应用程序的行为和输出有较大的改变的区域, 这一区域被称为“敏感区域”。敏感区域相比非敏感区域更易发现更深层的错误, 识别价值相比而言更大, 可以有更大概率生成高质量的测试用例。

本文首先将敏感区域与高质量测试用例关联, 对输入敏感区域进行预测, 利用神经网络擅长学习数据的内在联系并能够根据一类数据总结出其内在的抽象特征这一优势, 提出了基于神经网络的敏感区域预测算法。需要指出的是, Mohit 等人[18]也通过神经网络对初始种子进行敏感区域的预测, 然而其方法缺少对模糊测试变异过程的适应能力。这是由于模糊测试初期的训练集较小而导致的神经网络预测能力弱, 随着模糊测试进程的加深, 初始神经网络的预测效果将不断下降。同时, 使用预测能力弱的神经网络会导致后续变异样本种类少、变异路径单一的问题。针对上述问题, 本文在敏感区域的神经网络基础上, 引入了增量学习方法[19], 通过模糊测试变异过程中不断获得的高质量敏感区域变异样本对神经网络进行调整, 使得模型不仅能识别旧有样本的敏感区域, 同时也对新增样本产生适应性, 进而提高神经网络的预测能力。针对敏感区域的变异效率问题, 本文研究设计了针对敏感区域感知的变异算法, 通过优化变异方法加快模糊测试进程。为了验证方法的有效性, 本文选取了主流的格式文件 PNG、TIFF、XML 以及读取程序 Pngfix、Tiff2pdf、Xmllint 进行实验。

2 背景与动机

程序在处理复杂格式的输入时, 对输入不同区域的更改常常做出不同的反应。输入中一些非敏感区域发生较大的改变并不能对程序的执行造成任何影响。相反另外一些敏感区域较小的改变则会对程序执行造成较大的影响, 甚至不能正常的解析输入。简单来说就是输入中 20%的字段影响程序 80%的程序行为。

以 PNG 为例, 对 PNG 图片的不同部分进行更改或者破坏可能会对最终呈现的图像产生不同的影响, 图 1 中的每个图片(原始图像(a)除外)是经对原始 PNG 文件的某一部分进行随机模糊处理后, 经图片解析器解码后的图片。这组图像说明了经过变异模糊处理图片的不同部分, 可以对最终解码图像产生显著不同的影响。特别是对一些部分对解码后的输出图像的质量(甚至能够解析出图片)至关重要, 相反另外一些部分模糊修改后基本上完全保留了解码后的图像。

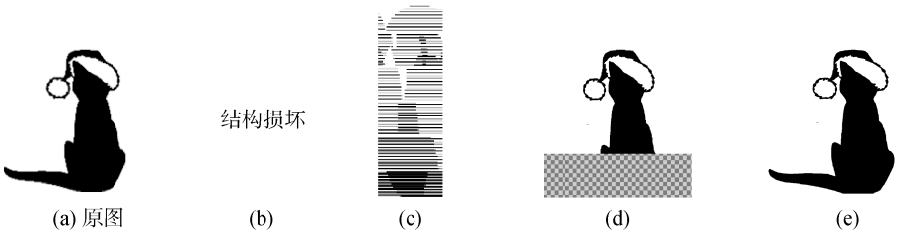

图 1　对图片不同部分进行模糊

Figure 1　Fuzzing different parts of picture

图 1(b)显示了修改 PNG 文件标志位后的文件, 可以看出由于 PNG 文件标志被破坏, 导致图片解码器无法解析文件。同样对 PNG 文件头数据块的破坏会导致输出的严重失真;输出具有不同的尺寸, 内容与解码器在原始图像上的输出没有相似之处(图 1(c))。这是因为文件头数据块储存着图片的宽度高度、颜色类型、色深等基本信息。调色板中大量引用部分也很重要, 因为这些部分的更改可以明显扭曲已解码图像的大部分(图 1(d))。

相比之下图像数据块对于图片的解析质量就影响较小。图 1(e)显示了模糊图像数据块区域的效果, 这些模糊发生在图像的末尾。在这种情况下, 图像底部会有少量的局部失真。通常, 给定图像数据块的关键性取决于它在图像文件中的位置。由于图像编码的压缩算法(每个图像数据块都包含着一系列压缩算法), 每个编码的像素都会影响所有后续编码的像素。因此, 在文件早期出现的图像数据块比较关键, 而在文件后期出现的图像数据块则可以很小有影响。

许多应用程序处理的都是复杂, 高度结构化的输入。从 PNG 例子可以看出, 高结构化的输入文件的某些区域极小的变化会引起应用程序的行为和输出有较大的改变。相反输入文件的一部分区域发生较大的改变, 也只会引起应用程序行为和输出极小的变化, 甚至没有产生任何影响。所以识别这些对应用程序行为和输出影响较大的位置, 并倾斜更多资源在这些位置进行充分的变异探索, 可以发现更深层的错误。而针对那些对应用程序行为和输出基本不能造成影响的位置, 花费较小的资源去变异探索这些收益较小的位置, 可以节省大量的时间和资源。即, 识别关键变异区域, 是改进模糊测试的时间复杂度以及探索深度的重要手段。

3　系统设计与实现

针对识别敏感区域问题, 本文提出基于敏感区域的模糊测试系统, 如下图 2 所示。首先使用过往模糊测试中的 *interesting* 种子及其覆盖率进行对神经网络模型进行训练, 神经网络模型学习输入文件中的敏感区域。然后模糊测试阶段, 对于测试程序种子队列, 首先经过种子挑选算法挑选出最好的种子, 把该种子转化为字节流特征数据放入神经网络去预测该种子的敏感区域, 然后对该种子的敏感区域施加变异算法。变异后的文件放入中程序执行, 对程序执行产生增益的种子又被放入种子队列以供后续的模糊测试变异探索。另一方面这部分产生变异增益的变异文件又可以放入模型进行增量学习, 不断地完善模型预测算法, 适应后续的模糊测试进程。

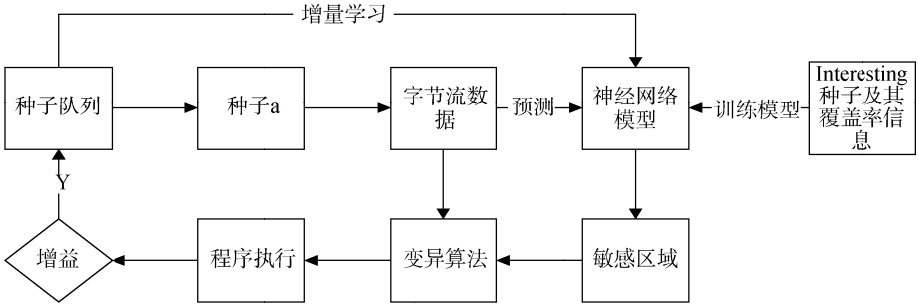

图 2　基于敏感区域预测的模糊测试系统

Figure 2　Fuzzy test system based on sensitive region prediction

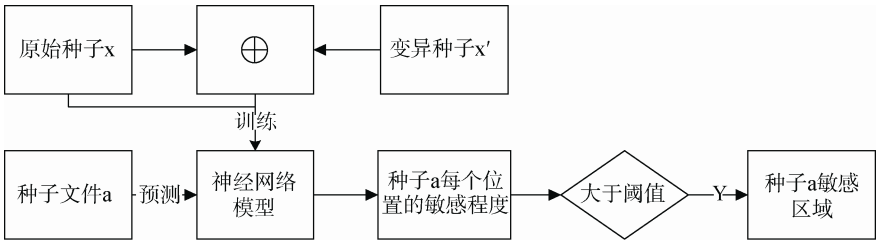

图 3　基于神经网络的敏感区域预测算法

Figure 3　Sensitive region prediction algorithm based on Neural Network

3.1　敏感区域预测

基于神经网络的敏感区域预测算法流程如图 3 所示，⊕表示异或操作。神经网络接收两个参数，一个是原始种子x，另一个是原始种子x和变异种子x′的差异(即变异种子x′改变了哪些位置)。因为神经网络训练使用的是对程序执行产生增益的数据，所以变异种子x′改变了的位置即是原始种子 x 的敏感位置。接收到变异种子与原始种子差异后，根据变异字节及其上下文等信息对于单字节构造特征向量，并将其输入到神经网络模型中训练。神经网络模型训练完成之后，可以直接对种子 a 进行预测。因为神经网络模型输出值是 0 到 1 之间的实数，表示种子 a 中每个字节的敏感程度，数值越大表示该字节对程序执行越敏感。对每个位置进行判断，如果大于预设的阈值，则判定该位置属于敏感区域。效果如图 4 所示，阴影部分则是敏感区域。

89	50	4E	47	0D	0A	1A	0A
00	00	00	10	00	00	00	10
53	00	00	00	04	67	41	4D
E9	00	00	00	20	63	48	52
84	00	00	FA	00	00	00	80
60	00	00	3A	98	00	00	17
9B	50	4C	54	45	00	00	00
66	62	B2	59	60	AF	58	74

00	00	00	0D	49	48	44	52
08	03	00	00	00	28	2D	0F
41	00	00	AF	C8	37	05	8A
4D	00	00	7A	26	00	00	80
E8	00	00	75	30	00	00	EA
70	9C	BA	51	3C	00	00	01
74	C7	68	73	C6	67	72	C5
C7	68	71	C3	65	6F	89	07

图 4　Png 文件敏感区域实例

Figure 4　A Sensitive region sample of png

种子文件以字节流形式输入，为此本文考虑以字节流的特征向量作为神经网络模型输入，如式(1)所示。

$$\begin{aligned} &\mathrm{Fea_k}:\{Byte_{k-1}, Byte_k, Byte_{k+1}\} \\ &Byte_k \in [0x00, 0xff] \\ &k \in [1, \mathrm{N}] \end{aligned} \tag{1}$$

同时考虑到种子文件可变长度的性质，本文把敏感区域预测函数定义为一个函数簇(如式(2)所示)。为了表述简单，本文把该函数定为f，函数f接收一个可变长度为k字节的输入文件，根据该输入文件每一个位置获得增益可能性的大小，输出该文件数据的敏感区域，“1”表示该位置是重要的敏感区域，反之“0”表示该位置是非敏感区域。在程序执行之前否决掉被判定为无效变异的文件，避免执行几乎不能产生增益的无效变异文件。当然敏感区域位置的粒度也可以选在比特位水平上，此时的函数f如式(3)所示。

$$f_{\mathrm{k}}:\{Fea_1 \dots Fea_{\mathrm{k}}\} \to [0,1]^{\mathrm{k}} \tag{2}$$

$$f_{\mathrm{k}}:\{Fea_i\}^{8k} \to [0,1]^{8k} | i \in [1,8k] \tag{3}$$

$$\sum_k [(\mathrm{x} \oplus \mathrm{x}') \wedge f(\mathrm{x})] > \alpha \tag{4}$$

变异文件是否被判定为有效变异，如式(4)所示。x表示输入的种子，x′表示种子x变异后的文件，⊕表示异或操作，∧表示与操作，$f(\mathrm{x})$即预测文件敏感区域的函数f，α表示阈值变量。x⊕x′表示种子文件x与该种子x变异文件x′的不同。$(\mathrm{x} \oplus \mathrm{x}') \wedge f(\mathrm{x})$表示变异文件变异的敏感区域位置。变异文件变异了敏感区域位置的个数大于阈值α则表示该变异为有用变异。在程序执行阶段，有用变异会被放入被测程序去执行，检测程序执行时代码覆盖率，执行路径等状态。

3.2　模型训练数据获取

为了学习到预测敏感区域的函数f，本文方法需要种子文件x及其变异位图b和该种子的变异文件x′及其变异位图b′。变异位图例如代码覆盖率，基本块覆盖率等数据在大多数灰盒模糊器中很容易得到，并不需要额外的指导信息去生成。黑盒模糊器也可以很容易的生成针对目标程序的相关变异信息b, b′。虽然很明显知道变异文件没有导致变异位图产生增益改变说明变异发生在无用位置，但是并没有直接的方法通过<输入文件，变异增益>键值对获得敏感区域。本文的想法是创建一个监督学习的训练数据集，如公式(5)所示。

$$\begin{aligned} (\mathrm{X}, \mathrm{Y}) = \{&(\mathrm{x}, (\mathrm{x} \oplus \mathrm{x}')) | \\ &s(\mathrm{b}, \mathrm{b}') > \gamma\} \end{aligned} \tag{5}$$

$s(\mathrm{b}, \mathrm{b}')$表示变异文件的变异位图b′与原种子文件的变异位图b的增益。变异了敏感区域的变异文件会比变异非敏感区域的得分高。γ是预定义的阈值。变异

得分大于该阈值的数据才会被纳入训练集。这个训练集的目标是给一个种子文件学习其原文件与变异文件的差异, 反过来模型又可以预测该文件的敏感区域, 以供后面的模糊测试变异。

上述方法的优点是它可以剔除掉$s(b, b')$得分低于敏感区域变异文件的非敏感区域变异文件。从$s(b, b')$得分较高的数据集(X, Y)中可以学到较好的敏感区域。一个种子会有较多的$x \oplus x'$值, 为了最小化这种一对多关系的相对损失, 模型学习给定种子x与其$x \oplus x'$的期望值, 即$E(x \oplus x')|x$。

由于执行了目标程序新的代码块、代码覆盖率的改变、导致程序崩溃等都是有增益的, 因此需要选择合适的函数量化变异增益函数$s(b, b')$。本文根据原种子x的代码变异位图b和变异文件x'的变异位图b'定义增益函数$s(b, b')$, 公式(6)所示。

$$s(b, b') = \sum_{1 \leqslant i \leqslant |b|} [b_i < b'_i] \tag{6}$$

如何获得初始种子, 最常见的基于覆盖的模糊器采用了基于突变的测试用例生成策略, 这在很大程度上依赖于初始种子输入的质量。良好的初始种子投入量可以显著提高模糊的效率和效果。具体来说: (1)提供格式良好的种子输入可以节省构建一个种子所消耗的大量 CPU 时间; (2)良好的初始输入可以满足复杂文件格式的要求, 这在突变阶段是很难达到的; (3)基于格式良好的种子输入的突变更可能生成能够达到更深层次和更高层次的测试用例; (4)良好的种子输入可以在多次测试中重复使用。

收集种子输入的常用方法包括使用标准基准、从互联网上爬行和使用现有的 POC 样本。开放源码的应用程序通常是使用标准基准, 可以免费使用该基准测试项目, 所提供的基准是根据应用程序的特性和功能构造的, 这自然构造了一组好的种子输入。考虑到目标应用程序输入的多样性, 从互联网上爬取是最直观的方法, 可以轻松下载具有特定格式的文件。此外, 对于一些常用的文件格式, 网络上有许多开放的测试项目提供免费的测试数据集。本文使用网络爬取的数据集样本。针对每一种目标程序, 选择 60 个种子文件使用 AFL 进行 24 小时的变异, 产生增益的文件会被用来训练神经网络模型。流程如图 5 所示。具体而言本文γ选择 0, 即覆盖新的代码块的变异文件会被选择训练神经网络模型。

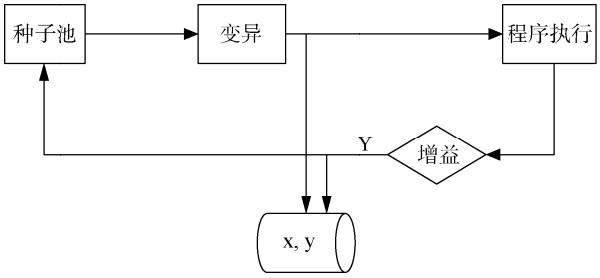

图 5 模型训练数据集生成

Figure 5 Model training data generation

3.3 敏感区域增量学习策略

为了解决后续样本带来的敏感区域识别能力问题, 需要引入使神经网络模型不断更新学习的能力。本文采用基于敏感区域的增量学习策略, 对模型进行自适应的改进。考虑对于当前所训练的m个神经网络模型, 对于当前样本的每一个字节的敏感预测概率y如式(7)所示:

$$y = \sum_i^m w_i f(x) \tag{7}$$

此时对于变异后的样本, 放入程序执行后, 可以由式(4)给定当前变异样本的变异有效性标签y, 此时可以计算出当前的网络模型对于该变异后的样本关于每一个字节的敏感区域预测结果结果。此时对当前样本的误差率e_m如式(8)所示:

$$e_m = \sum_i^m w_i I(H(x) \neq y) \tag{8}$$

利用e_m, 可以对m个神经网络模型参数进行更新, 目的是强化对当前样本敏感区域预测误差低的神经网络模型权重, 同时保留对于旧样本预测误差低的神经网络模型, 更新公式如式()所示:

$$w_i = w_i \times \begin{cases} \frac{e_m}{1-e_m}, H(x) = y \\ 1, \ H(x) \neq y \end{cases} \tag{9}$$

敏感区域增量学习流程如图 6 所示。该方法的好处是当得到新的有效变异样本, 通过调整神经网络模型之间的权重, 可以更好地适应新样本的预测概率结果, 同时保留原有的神经网络模型, 使得原有样本关于敏感区域的记忆不会丢失。

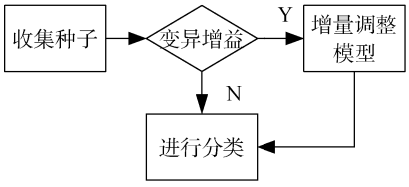

图 6 敏感区域增量学习流程

Figure 6 Enhanced learning process of sensitive region

3.4 敏感区域感知的变异策略

基于神经网络的敏感区域预测算法可以学习文件中对目标程序执行和输出影响较大的敏感位置。专注于敏感区域进行变异不仅可以显著提高模糊测试效率, 还能够发现更多的程序错误。变异算法流程

如图 7 所示。

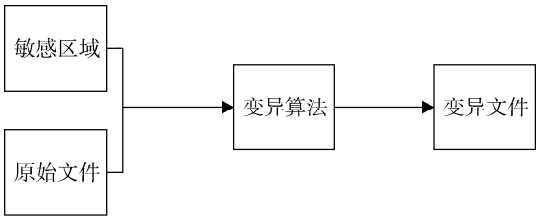

图 7　变异算法流程
Figure 7　Mutation algorithm process

对原始文件和该文件的敏感区域进行预设的变异算法进行变异。变异后的文件放入程序执行, 监测程序的非预期行为。好的变异策略可以使得模糊测试探索更大的变异空间, 可以使得覆盖更多的代码块以及更大的代码覆盖率。本文参考了 AFL 的变异策略将变异探索分为确定性部分和随机部分。变异方法如下:

1) 位反转: 在比特位水平上变异, bit 位 0 翻转成 1,1 翻转成 0。在翻转长度上设置为 1、2、4、8、16、32 比特位进行翻转, 步长分别选择 1、1、1、8、8、8;

2) 加减运算: 在字节水平上, 对每个字节进行加减运算, 字节长度选择 1、2、4;

3) interesting 字段替换: 在字节水平上, 对种子文件以 1、2、4 个字节为单位分别替换预定义的 interesting 字段;

4) 字典替换: 在字节水平上, 把种子文件上的字节内容替换成用户自定义的字典内容;

5) 随机字节翻转: 在字节水平上, 随机选择一个字节、两个字节、四个字节进行翻转操作;

6) 随机字节替换: 在字节水平上, 随机选择某些字节以一定几率替换为随机字节或者种子中的随机字节;

7) 随机字节重写: 在字节水平上, 随机选择字节进行重写操作;

8) 随机插入: 在字节水平上, 随机选择位置以一定几率插入随机字节或者种子文件中的随机字节;

9) 随机删除: 在字节水平上, 随机选择位置随机删除某些字节。

同时, 针对变异过程中带来的样本迭代效率问题, 本文设计了加速变异策略: 同时对多个敏感区域进行(比如 1%、2%、3%、5%等的敏感区域)位翻转、字节翻转、加减运算、*interesting* 字段替换、字典替换、重写和随机替换重写等变异方法。加速变异策略如算法 1 所示。

算法 1　加速变异算法

输入: $Byte, MutaPlan$
输出: $MutaByte$

$FOR\ l \in [1, length\ of\ SensiArea]$:
　　$IF\ isSensiArea(Byte_l)$:
　　　　$plan = rand(MutatePlan)$
　　　　$len = SensiAreaLen(Byte_l)$
　　　　$Mutate(plan, Byte_l)$
　　　　$MutaByte = Byte$
$l = l + len$

加速变异策略可以加快模糊测试探索进程, 以实现对敏感区域更深的探索。同时考虑对于非变异敏感点存在变异增益的可能性, 对于非变异敏感点, 以 10%的概率进行变异。

4　实验

4.1　实验数据来源

为了验证方法的有效性, 本文选取了几种热门的格式文件 PNG、TIFF 以及 XML。并选取了使用这些文件的程序 Pngfix-1.6.36, Tiff2pdf-3.6.1 以及 Xmllint-2.4.16。

4.2　实验评测标准

在计算机科学中, 测试覆盖率是一种度量方法, 用于描述当特定的测试用例运行时, 目标程序源代码的执行程度。测试覆盖率的测试用例在测试期间执行了更多的源代码, 与测试覆盖率低的测试用例相比, 它包含的未检测到的软件漏洞的几率更小。有许多不同的度量标准可以用来计算测试覆盖率, 最基本比如基本块覆盖率 Basic Block Coverage(BBC)、分支覆盖率 Branch Coverage(BC)、函数覆盖率 Function Coverage(FC)等。在程序分析中, 程序由基本块组成, 基本块是只含有一个入口点和一个出口点的代码片段, 基本块中的指令顺序执行, 并且只执行一次。在代码覆盖率的度量中, 最常用的方法是以基本块作为最佳粒度。其主要原因是:基本块是程序执行的最小相干单位。基本块覆盖率通过覆盖的基本块数量与总基本块数量相除得到。

$$\begin{aligned} \mathrm{BBC} &= \frac{\mathrm{Num_{CoverBB}}}{\mathrm{Num_{TotalBB}}} \\ \mathrm{BC} &= \frac{\mathrm{Num_{CoverBranch}}}{\mathrm{Num_{TotalBranch}}} \\ \mathrm{FC} &= \frac{\mathrm{Num_{CoverFunc}}}{\mathrm{Num_{TotalFunc}}} \end{aligned} \tag{10}$$

分支覆盖率(BC)是一个关键代码覆盖率度量标准, 用于帮助开发团队确定其代码库的总体质量。分

支覆盖率建立在序列点覆盖率的基础上, 分支覆盖率是通过计算代码覆盖分支与总分支关系的一种重要的代码覆盖率度量指标。函数覆盖率(FC)是一种粗粒度的测试标准, 它表示程序有多少比例功能被测试用例执行到。对于分析哪些功能没被执行到特别有用。因此, 本论文拟选取基本块覆盖率、分支覆盖率和函数覆盖率作为实验评测标准, 计算方法如式 4-1 所示。被测程序相关信息如表 1 所示, 列出了输入格式, 被测程序, 版本号及其基本块数量、分支数量和函数数量。

表 1 被测程序及相关信息

Table 1 Program set and related information

输入格式	程序	版本	基本块数量	分支数量	函数数量
Png	Pngfix	1.6.36	39924	86509	388
Tiff	Tiff2pdf	3.6.1	2972	4996	116
Xml	Xmllint	2.4.16	4570	9084	207

表 2 CNN 网络模型参数

Table 2 CNN network parameters

Layer(type)	Output Shape	Param#
convId_1 (ConvID)	(None,1024,128)	512
convId_2 (ConvID)	(None,1024,128)	49280
max_poolingId_1 (MaxPooling)	(None,512,128)	0
convId_3 (ConvID)	(None,512,256)	98560
convId_4 (ConvID)	(None,512,256)	196864
max_poolingId_2 (MaxPooling)	(None,256,256)	0
convId_5 (ConvID)	(None,256,256)	196864
convId_6 (ConvID)	(None,256,256)	196864
max_poolingId_3 (MaxPooling)	(None,128,256)	0
convId_7 (ConvID)	(None,128,112)	86128
convId_8 (ConvID)	(None,128,112)	37744
max_poolingId_4 (MaxPooling)	(None,42,112)	0
Flatten_1(Flatten)	(None,4704)	0
dense_1(Dense)	(None,112)	526960
dense_2(Dense)	(None,112)	12656
dense_3(Dense)	(None,1024)	115712
Totalparams: 1518144 Trainable params: 1518144 Non-trainable params : 0		

4.3 神经网络选取与参数设置

卷积神经网络(CNN: Convolutional Neural Network)最大的优势体现其在特征提取方面。CNN 的特征提取层(卷积层和池化层)是直接从训练数据进行特征提取, 避免了人为的显式特征提取而造成信息丢失的影响, 隐式的从训练数据集中学习。卷积神经网络以其权值共享、局部连接的特殊结构, 不仅在图像领域取得了巨大成功, 而且对于序列数据方面也存在巨大的潜力。卷积神经网络(一维卷积神经网络 Conv1D)是本文拟采用的神经网络模型, 去学习输入数据中的敏感区域。卷积神经网络模型结构如表 2 所示。数据长度最大定义为 1024, 特征提取部分具体设置为每两个卷积操作后接一个最大池化提取数据特征。分类部分设置为 3 个全连接层。最后一层全连接层用于输出每个位置的重要程度。

不同于卷积神经网络提取特征的方式, LSTM 能够处理特征数据之间的联系, 比如在机器翻译领域的依赖问题(当前场景依赖于上下文语境)。在输入数据的结构问题上同样存在着依赖问题。数据文件中后面的解析路径一定程度上依赖于文件前面的数据结构。比如当前的 if 分支依赖于之前的 if 条件判断。针对该问题, 本文设置了基于循环神经网络的敏感区域预测。循环神经网络模型结构及参数如表 3 所示。数据长度最大定义为 1024, 循环神经网络三个 LSTM 层用于提取输入文件数据的特征, 经 3 个全连接层最后输出每个位置的重要程度。网络中使用 Batch Normalization 有如下几个优点: 1)用于加快训练速度, 可以使用较大的学习率来训练网络; 2)提高网络的泛化能力; 3)可以打乱样本的训练数据。

表 3 LSTM 网络模型结构及参数

Table 3 LSTM network model parameters

Layer(type)	Output Shape	Param#
Masking_1(Masking)	(None,1024,1)	0
batch_normalization_1 (BatchNormalization)	(None,1024,1)	4
lstm_1(LSTM)	(None,1024,256)	264192
Batch_normalization_2 (BatchNormalization)	(None,1024,256)	1024
lstm_2(LSTM)	(None,1024,256)	525312
batch_normalization_3 (BatchNormalization)	(None,1024,256)	1024
lstm_3(LSTM)	(None,256)	525312
batch_normalization_4 (BatchNormalization)	(None,256)	1024
dense_1(Dense)	(None,128)	32896
batch_normalization_5 (BatchNormalization)	(None,128)	512
dense_2(Dense)	(None,128)	16512
Batch_normalization_6 (BatchNormalization)	(None,128)	512
dense_3(Dense)	(None,1024)	132096
Totalparams: 1500520 Trainableparams: 1498370 Non-trainableparams: 2050		

4.4 对比试验设置

为了验证本论文方法的效果，设置了基于敏感区域的模糊测试和 AFL 对比实验，如图 8 对比实验流程。首先使用 AFL 对原始种子进行变异，随机挑选经 AFL 变异生成的 *m* 个 *interesting* 种子。把该 *m* 个 *interesting* 种子当成对比实验的原始种子。一方面该 *m* 个 *interesting* 种子重新使用 AFL 进行 24 小时的模糊测试，最后测试该 *m* 个种子达到的覆盖率 *a*(基本块覆盖率、分支覆盖率和函数覆盖率)，以此与本文中的方法进行比较。另一方面同样把该 *m* 个种子作为基于敏感区域模糊测试的原始种子，使用深度学习模型预测该种子的敏感区域，对种子的敏感区域进行变异，经程序执行获取增益的文件又会被放入种子池，以供后续变异。最后，同样经过 24 小时的模糊测试最后计算其达到的覆盖率 *b*(基本块覆盖率、分支覆盖率和函数覆盖率)。

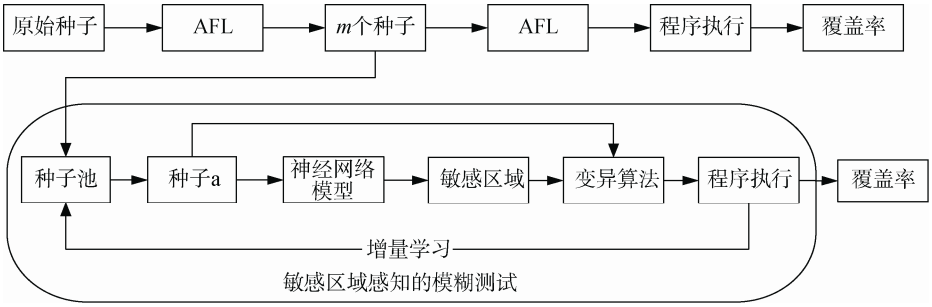

图 8 对比实验流程

Figure 8 Comparative experiment process

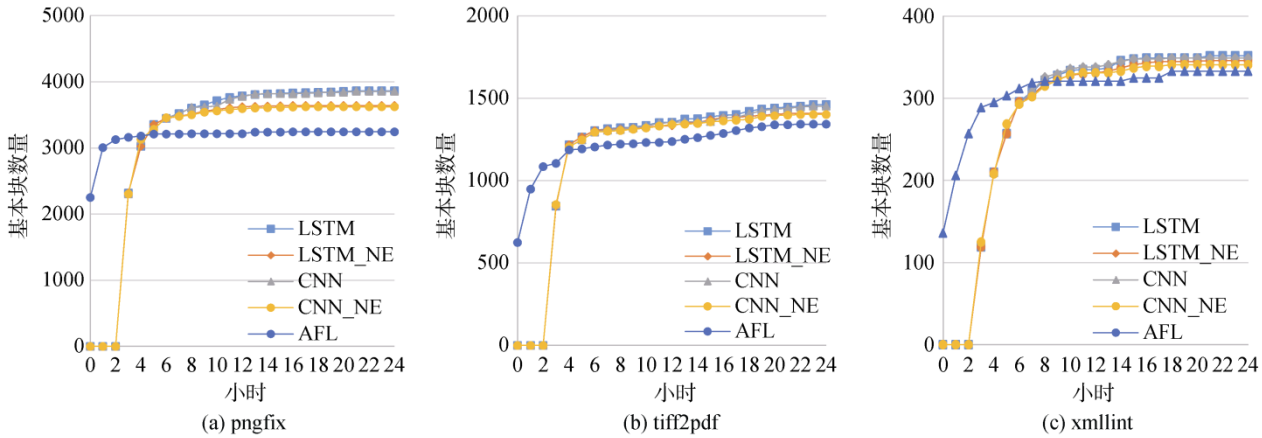

图 9 发现被测程序 pngfix, tiff2pdf, xmllint 基本块数量随时间变化

Figure 9 Number of Basic Block changing with time in pngfix, tiff2pdf, xmllint

4.5 实验结果对比与分析

为了测试基于神经网络的敏感区域变异的模糊测试方法效果，本文使用 tiff2pdf、pngfix 和 xmllint 三种程序评估结果。在网络上个爬取了 61 个 Tiff 格式、Png 格式和 xml 格式的文件，其中每种格式的 60 个文件用于 AFL 提前生成模型训练数据，剩下的一个种子文件用于本文方法与 AFL 的实验对比。本文选择重要度大于 0.05 的位置设置为敏感区域。

需要指出的是，由于 Mohit 等人[18]的方法并未开源，为了形成与其方法的对比，本文实现了在本文方法中去除增量学习以及变异优化策略后的神经网络预测模型以进行后续的对比实验。

图 9 中分别是本文中方法和 AFL 在被测程序 pngfix、tiff2pdf 和 xmllint 上发现代码块数量随时间变化。如图可以看到本文的方法在三种被测程序上 24 小时实验结束时发现基本块数量多于 AFL。具体以图 9(a)为例，前两个小时本文中的方法 pngfix 基本块发现数量为 0，这是因为本文为了对比试验的科学性，把预测敏感区域的模型训练时间计算到了对比试验中。基于卷积神经网络(CNN)和基于循环神经网络(LSTM)的方法发现基本块数量差异较小，两条线基本重合。在 4 小时时，本文中的方法发现基本块的数量接近 AFL，并在 5 小时时发现基本块数量多于 AFL。在 24 小时最终试验后，LSTM 和 CNN 发现基本块数量均多于 AFL。与未加入增量学习及变异优化策略的 LSTM_NE 和 CNN_NE 相比，加入增量

学习及变异优化策略的LSTM与CNN在发现的基本块数量上也有所提升。

基本块覆盖率、分支覆盖率、函数覆盖率及其相对于 AFL 以及未加入增量学习及编译优化策略的 LSTM 与 CNN 的提升比例结果如表 4 所示。针对被测程序 pngfix, CNN 和 LSTM 相较于 AFL 提升比例最大, 在基本块覆盖率、分支覆盖率和函数覆盖率方面提升分别可以达到 19%左右、大于 16.7%和 12.5%。与未加入增量学习的方法相比, 提升分别可以达到 6.9%、6.5%以及 3.5%。在被测程序 Xmllint 上, 本文的方法相对提升较少, 分析其原因是基于变异的模糊测试用例生成方法依赖于当前种子的质量, 在质量不好的种子上进行变异很难达到较高的代码覆盖率。

表 4 实验结果对比

Table 4 Comparison of experimental results

程序	方法	基本块覆盖率	提升比例/%	分支覆盖率	提升比例/%	函数覆盖率	提升比例/%
Pngfix	AFL	0.0813	-	0.0239	-	0.369	-
	CNN_NE	0.0910	11.9	0.0271	13.4	0.402	9.0
	CNN	**0.0966**	**18.8**	**0.0289**	**20.9**	**0.415**	**12.5**
	LSTM_NE	0.0912	12.2	0.0261	9.2	0.402	9.0
	LSTM	**0.0968**	**19.1**	**0.0279**	**16.7**	**0.415**	**12.5**
Tiff2pdf	AFL	0.4523	-	0.254	-	0.776	-
	CNN_NE	0.4721	4.4	0.283	11.4	0.789	1.7
	CNN	**0.4892**	**8.1**	**0.293**	**14.0**	**0.793**	**2.2**
	LSTM_NE	0.4741	4.8	0.274	7.9	0.789	1.7
	LSTM	**0.4926**	**8.9**	**0.286**	**11.3**	**0.793**	**2.2**
Xmllint	AFL	0.0729	-	0.0141	-	0.150	-
	CNN_NE	0.0746	2.3	0.0153	4.1	0.153	2.0
	CNN	**0.0764**	**4.8**	**0.0157**	**8.5**	**0.155**	**3.2**
	LSTM_NE	0.0757	3.8	0.0148	4.9	0.153	2.0
	LSTM	**0.0770**	**5.7**	**0.0149**	**5.5**	**0.155**	**3.2**

5 结论

软件漏洞是造成计算机安全问题的根本原因,而模糊测试是目前最有效的漏洞发现解决方法之一。传统的基于突变的模糊测试测试用例生成方法存在生成测试用例质量不高, 变异效率低下的问题, 针对这些问题本文提出了针对敏感区域突变的模糊测试用例生成方法, 并提出了一种优化的变异策略。在 tiff2pdf、pngfix 和 xmllint 模糊测试实验中, 本文中的方法在代码块覆盖率、分支覆盖率和函数覆盖率方面均优于基于随机变异的 AFL。并且在较短的时间发现了更多的代码路径。在接下来的工作中, 本文将着力于提高敏感区域定位模型的引入与优化, 以更好地指导变异。

参考文献

[1] Nayak K, Marino D, Efstathopoulos P, et al. Some Vulnerabilities are Different than others[M]. ResearchinAttacks, Intrusionsand-Defenses. Cham: Springer International Publishing, 2014: 426-446.

[2] “Adobe Flash and Microsoft Windows Vulnerabilities.”, https://www.us-cert.gov/ncas/alerts/TA15-195A, 2015.

[3] “Oracle Java Contains Multiple Vulnerabilities”, https://www.us-cert.gov/ ncas/alerts/TA13-064A, 2013.

[4] “The Heartbleed Bug.”, http://heartbleed.com/, 2014.

[5] “ShellShock: All you need to know about the Bash Bug vulnerability.”, Symantec Security Response,http://www.symantec.com/connect/blogs/shellshock-all-you-need-know-about-bash-bug-vulnerability,2014.

[6] BREEN, Stephen. What Do WebLogic, WebSphere, JBoss, Jenkins, OpenNMS, and Your Application Have in Common. This Vulnerability, 2015.

[7] Valentin J.M. Manes, HyungSeok Han, Choongwoo Han, et al. Fuzzing: Art, science, and engineering. arXiv preprint arXiv:1812.00140, 2018

[8] Sanjay Rawat, Vivek Jain, Ashish Kumar, et al. VUzzer: Application-aware Evolutionary Fuzzing[J]. *NDSS*.2017, 55(17):1-14.

[9] Wang J J, Chen B H, Wei L, et al. Skyfire: Data-Driven Seed Generation for Fuzzing[C].*2017 IEEE Symposium on Security and Privacy (SP)*, May 22-26, 2017. San Jose, CA, USA. Piscataway, NJ: IEEE, 2017: 579-594.

[10] GodefroidP, PelegH, SinghR. Learn&Fuzz: Machine Learning for Input Fuzzing[C]. *2017 32nd IEEE/ACM International Conference*

on Automated Software Engineering (ASE),October 30-November 3, 2017. Urbana, IL. Piscataway, NJ: IEEE, 2017: 50-59.

[11] Böhme M, Pham V T, Nguyen M D, et al. Directed GreyboxFuzzing[C]. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS '17*, October 30-November 3, 2017.Dallas, Texas, USA. New York, USA: ACM Press, 2017: 2329-2344.

[12] Wang S, Nam J, Tan L. QTEP: Quality-aware Test Case Prioritization[C]. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017*, August 21-September 8, 2017. Paderborn, Germany. New York, USA: ACM Press, 2017: 523-534.

[13] Theofilos Petsios, Jason Zhao, Angelos D. Keromytis,et al. Slowfuzz: Automated domain-independent detection of algorithmic complexity vulnerabilities[C].*Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM*, 2017:2155-2168.

[14] Wang T L, Wei T, Gu G F, et al. TaintScope: A Checksum-Aware Directed Fuzzing Tool for Automatic Software Vulnerability Detection[C]. *2010 IEEE Symposium on Security and Privacy*, May 16-19, 2010. Oakland, CA, USA. Piscataway, NJ: IEEE, 2010: 497-512.

[15] GodefroidP, Levin M, Molnar D. SAGE: WhiteboxFuzzingfor Security Testing[J].*Communications of the ACM*,2012, 55(3): 40-44.

[16] Godefroid, Patrice, Michael Y. Levin, David A. Molnar. Automated Whitebox Fuzz Testing[J]. *NDSS*. 2008, 34（8）：151-166.

[17] STEPHENS N, GROSEN J, SALLS C, et al. Driller: Augmenting Fuzzing Through Selective Symbolic Execution[J]. *NDSS*.2016, 33（16）:1-16.

[18] RAJPAL, Mohit; BLUM, et al. Not all bytes are equal: Neural byte sieve for fuzzing.*arXiv preprint arXiv:1711.04596*, 2017.

[19] Polikar R, Upda L, Upda S S, et al. Learn++: An Incremental Learning Algorithm for Supervised Neural Networks[J]. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 2001, 31(4): 497-508.

张羿辰于 2017 年在西南交通大学通信工程专业获得学士学位。现在武汉大学网络空间安全专业攻读硕士学位。研究领域为软件安全、网络安全。研究兴趣包括: 软件安全、网络安全、机器学习。
Email: eddenzhang@outlook.com

金银山于 2017 年在山东工商学院计算机科学与技术专业获得学士学位。现在武汉大学网络空间安全专业攻读硕士学位。研究领域为软件安全、网络安全。研究兴趣包括: 软件安全、网络安全、机器学习。
Email: jys_whu@outlook.com

赵磊于 2012 年在武汉大学信息安全专业获得博士学位。现任武汉大学国家网络安全学院副教授、博士生导师。研究领域为软件安全、网络安全、机器学习安全等。
Email: leizhao@whu.edu.cn