2019年第49届IEEE/IFIP国际可靠系统和网络会议（DSN）

检测“0天”漏洞：OSS秘密安全补丁的实证研究

王信达、孙坤、阿彻·巴切勒、苏轼·贾乔迪亚1121

1*美国弗吉尼亚州费尔法克斯乔治梅森大学信息科学与技术系安全信息系统中心*

2*诺斯罗普格鲁曼公司，美国华盛顿特区*

{}*贾乔迪亚ksun3 xwang44@gmu教育部, 阿切尔·巴切勒@ngc.com网站*

*摘要*-开放源码软件（OSS）中的安全补丁不仅为已识别的漏洞提供安全修复，而且还将易受攻击的代码公开给攻击者。因此，装甲攻击者可能会滥用此信息对未修补的OSS版本发起N天攻击。防止此类N天攻击的最佳做法是随时将软件升级到最新版本。然而，由于担心声誉和软件开发管理的方便性，软件供应商可能会选择在新版本中秘密修补其漏洞，而不向CVE报告，甚至不在其更改日志中提供任何明确的描述。当这些秘密修补的漏洞被装甲攻击者识别时，它们可以转化为强大的“0天”攻击，利用这些漏洞不仅可以危害同一软件的未修补版本，还可以危害类似类型的操作系统（例如。，由于代码克隆或类似的设计/实现逻辑，SSL库）可能包含相同的漏洞。因此，识别秘密安全补丁并将这些“0天”攻击的风险降低到至少“n天”攻击是至关重要的。在本文中，我们开发了一个防御系统并实现了一个工具集来自动识别开放源码软件中的秘密安全补丁。为了区分安全补丁和其他补丁，我们首先构建了一个安全补丁数据库，其中包含4700多个映射到CVE列表中记录的安全补丁。接下来，我们将使用机器学习方法识别一组有助于区分安全修补程序和非安全修补程序的功能。最后，我们使用代码克隆识别机制来发现类似类型的操作系统中类似的补丁或漏洞。实验结果表明，该方法具有良好的检测性能。对OpenSSL、LibreSSL和BoringSSL的案例研究发现了12个秘密安全补丁。

*索引项*-安全补丁，漏洞检测，开源软件

# 一、简介

|  |
| --- |
|  |
|  |  |

近年来，开源软件（OSS）的流行令人印象深刻。作为最大的托管服务提供商之一，GitHub宣布，2018年有3100万名开发人员在9600万个存储库中工作[6]。与此同时，OSS中的漏洞数量还在不断增长。Snyk的一份报告显示，从2016年到2017年，公开源代码漏洞的数量增加了53.8%[25]。一个原因是攻击者可以仔细分析OSS的源代码来发现未知的漏洞。更糟的是，保安

|  |
| --- |
| 这项工作得到了NSF资助CNS-1822094、IIP-1266147和ONR资助N00014-16-1-3214、N00014-16-1-3216和N00014-18-2893的部分支持。  978-1-7281-0057-9/19/$31.00©2019 IEEE  内政部10.1109/DSN.2019.00056 |

漏洞补丁正好指出了易受攻击的代码，它教会攻击者如何生成攻击未修补版本的漏洞。例如，就在apachestruts2（CVE2017-5638）中的远程代码执行漏洞被公开披露并修复的一天后，漏洞脚本就出现了。后来，由于其未修补的系统，Equifax遭到攻击，包括社会保险号码在内的数百万个人数据被曝光[1]。

尽管及时修补漏洞是抵御N天攻击的有效方法，但在现实世界中仍然存在一些挑战。在许多情况下，安全修补程序包含在大型软件修补程序或新版本以及其他类型的修补程序中，例如，错误修复和新功能。由于应用软件补丁或更新到新版本会增加服务系统的停机时间并引入额外的工作负载，管理员或用户倾向于推迟更新他们正在运行的软件，直到有一个稳定的版本可用，或者安全提示（如Common Vulnerabilities and Exposures，CVE）显示存在严重的安全漏洞补丁[13]。

但是，软件供应商可能会秘密修补其漏洞，而不创建CVE条目，甚至不在其更改日志中描述安全问题。一个原因是担心更改日志中过多的CVE条目或漏洞修复可能会损害其软件的质量信誉。此外，他们可能打算阻止相关CVE条目的发布，直到他们认为公开发布这些条目是安全的。但是，由于相关补丁或新版本已经可用，攻击者仍然可以直接或从两个版本之间的差异仔细分析补丁中的代码更改，然后生成滥用这些安全漏洞的漏洞。为了保护这一点，开发人员和用户需要一种方法来识别开放源码软件中是否存在秘密安全补丁，以便及时更新软件。此外，类似类型的软件可能包含相同的漏洞，因为代码克隆在开源软件中很常见，开发人员在解决困难的智力问题时往往会犯同样的错误[11]。在这种情况下，分析一个软件（例如OpenSSL）中的安全补丁可以帮助识别和修复具有类似功能的其他软件（例如LibreSSL）中的相应漏洞。我们将这些漏洞视为“0天”漏洞的一种。

本文提出了一种基于机器学习的机械控制系统-

anism帮助自动从发布的软件补丁中识别秘密安全补丁，以及两个版本的开源软件之间的差异。首先，由于安全修补程序上没有公开可用的数据集，因此我们创建了一个新的安全修补程序数据集，其中包含4702个安全修补程序，方法是对1999年至2018年CVE条目[4]中所有可用的引用链接进行爬网。由于安全补丁可能在不同的编程语言中有不同的模式，所以我们专注于C/C++语言，并选择了C/C++编写的1636个安全补丁。此外，我们还从GitHub存储库中随机获取1636个非安全补丁[6]。

为了识别安全补丁，我们面临的一个主要挑战是识别有效的特征来模拟安全补丁和非安全补丁之间的差异。在对收集到的大部分安全性和非安全性补丁进行人工分析的基础上，我们确定了一组61个特征，分别属于三类，即基本特征、句法特征和语义特征。接下来，我们开发了一种基于机器学习的方法，利用识别出的特征集来区分安全补丁和非安全补丁。为了提高检测精度，我们采用了一种投票算法，该算法集成了随机森林、贝叶斯网、随机梯度下降（SGD）、序贯最小优化（SMO）和Bagging等五种流行的分类算法。我们随机选择80%的数据集作为训练数据集，并使用剩余的20%作为测试数据集。实验结果表明，该模型具有良好的性能，真阳性率为79.6%，假阳性率为41.3%。

为了进一步评估系统的有效性，我们对OpenSSL、LibreSSL和BoringSSL三个开源SSL库进行了案例研究，发现了12个秘密安全补丁，其中秘密补丁到公开发布的最长延迟超过两年。

# 二。系统概述

图1显示了我们系统的概述，包括三个主要步骤。第一步是构建一个安全补丁数据集，用于（i）为机器学习模型提取有用的特征，（ii）在训练阶段训练基于机器学习的安全模型，以及（iii）在检测阶段评估安全模型的有效性。通过查询1999-2018年的所有CVE条目，我们从至少898个开源项目中抓取了4702个安全补丁。其中，重点研究了C/C++中的1636个安全补丁。此外，我们从GitHub存储库随机获取1636个非安全补丁。因此，我们的数据集包含1636个安全补丁和1636个非安全补丁。

第二步是为基于机器学习的安全模型导出一组基本的、语法的和语义的特征。一些特征是从以前的相关工作中收集的，另外一些特征是通过手动观察我们的安全补丁和非安全补丁数据库来识别的。

在第三步中，我们采用投票算法，将随机森林、贝叶斯网、随机梯度下降（SGD）、序贯最小优化（SMO）和Bagging等五种流行的分类算法集成在一起，建立了基于机器学习的模型。如果补丁是安全补丁或非安全补丁，我们将每个补丁的特征转化为一个向量和一个标签标记。为了评估系统性能，我们随机选择80%的数据集作为训练数据集，剩下的20%作为测试数据集。

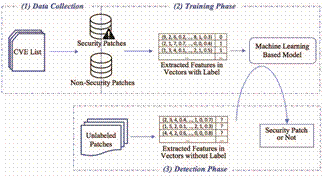
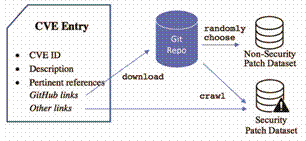


图1。系统概述

# 三、 修补程序数据库集合

由于没有公共可用的安全修补程序数据库，我们通过查询CVE条目[4]来构造一个数据库，其引用链接可能包含修补程序的URL。图2展示了数据库收集的整个过程。



## 图2。数据收集概述

为了收集安全补丁数据集，我们对1999年至2018年间所有CVE条目中包含补丁的所有相关参考URL进行爬网。为了收集非安全补丁数据集，我们下载出现在CVE列表中的开源存储库，并随机选择非安全提交作为非安全补丁数据集[1]。我们在下面描述数据收集的方法。

## A.安全补丁数据集

截至2018年11月4日，CVE列表由126491个CVE条目组成[4]。每个CVE条目都包括一个CVE ID、漏洞的简要描述以及报告、咨询和修补程序（如果有）的相关参考URL。根据我们的观察，参考url可以根据托管服务提供商的类型分为两类，即GitHub上托管的项目和其他网站上托管的项目。对于托管在GitHub上的项目，安全修补程序的引用URL的格式如下：https://github.com/owner/repo/commit/commit提交搞砸。承诺*搞砸*是哈希值，它是提交的唯一标识符。可以通过下载带有appendix.patch的提交URL来收集相应的补丁，即。，https://github.com/owner/repo/commit/commit提交哈希.补丁. 在GitHub上，一个提交对应于一个补丁。因此，每个提交都可以作为补丁下载。我们收集了4010个补丁，它们属于GitHub上托管的项目。

对于托管在其他网站上的项目，例如，它自己的网站，没有统一形式的参考url可以帮助判断它是否包含任何安全补丁。为了解决这个问题，我们对CVE条目中的所有引用url进行爬网，并使用特定的符号，即diff、@@、+++，以及作为补丁存在的指示器。一旦我们的爬虫识别出这样的标记，它就会删除网页中的HTML/CSS标签和其他不相关的内容（例如，标题、描述等），然后下载剩下的部分作为安全补丁。由于2010年以前的许多参考链接已不可用，因此我们仅以这种方式收集692个安全补丁。*--*

我们的模型旨在通过代码的语法和语义模式来识别安全补丁。由于不同的编程语言有不同的模式，我们专注于在开源生态系统中非常常见的C/C++项目补丁。我们的数据库包含了C++ C/C++项目中的1636个安全补丁。

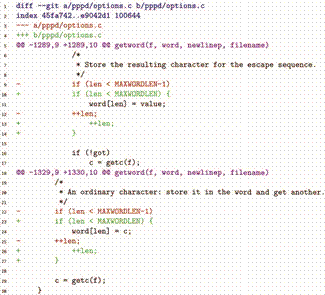


图3。CVE-2014-3158补丁样本

## B.非安全补丁数据集

为了训练我们的模型，我们还需要收集一个非安全补丁数据集。理论上，修补程序对应于漏洞修复、bug修复或功能更新。但是，由于版本控制原理不同，一些软件供应商可能会发布一个大补丁，其中包含多个补丁。此外，对于不在GitHub之类的控制版本系统中托管的项目，我们可能只会在原始和修改的源代码树之间生成一个统一的.diff文件，作为包含多个修补程序的大修补程序。为了避免这个问题，我们从出现在CVE列表中并且托管在GitHub上的项目中收集非安全补丁，这样每个提交就是一个补丁。在我们下载了所有898个相关的GitHub存储库（大约1T大小）之后，对于每个项目，我们使用gitlog命令获取所有提交及其哈希值。为了减少特定项目的偏差，我们随机选择1636个提交作为非安全补丁数据集。因为散列值是提交的唯一标识符，所以我们可以通过比较提交散列值来过滤掉已经包含在安全补丁数据集中的提交。因此，我们得到了一个大小相同的安全补丁和非安全补丁数据集。

# 四、 安全修补程序标识

根据前面的工作和我们的观察，我们在机器学习模型中收集了一组能够区分安全补丁和非安全补丁的特征。基于这些特征，我们的数据库中的每个补丁都可以表示为一个带有安全补丁或非安全补丁标签的向量。利用有监督的数据集训练机器学习模型后，当给定一个新的未标记补丁时，系统可以将其转换为相应的向量，然后识别它是否是一个安全补丁。

## A.特征提取

修补程序[7]包含新旧版本文件之间的差异。图3显示了CVE-2014-3158的补丁示例。补丁中显示的每个差异都以diff a/文件夹名/文件名b/文件夹名/文件名（例如，第1行）开头，并且每个差异可能包含多个更改块，这些更改块分别是连续删除和添加的标记为-和+的行。例如，第9行和第10行是一个变化的对象。表1展示了我们在这项工作中收集的特征。我们借用了Tian等人的工作[26]中的特征1-22，这表明这些特征在区分漏洞和缺陷修复补丁与新特征补丁方面是有效的。它们考虑文件、大块、条件语句、循环、行、字符和函数调用中的更改。Total是指这些基本程序特性的删除数和添加数之和，Net是添加数减去删除数。此外，我们的工作旨在区分安全补丁和非安全补丁，这需要更多的特性来表示安全补丁（即漏洞修复）和非安全补丁（即bug修复和新添加的特性）之间的区别。通过手动比较数据库中的安全修补程序和非安全修补程序，我们发现：与非安全修补程序相比，安全修补程序修改的代码更少。•

•安全修补程序更有可能对运算符和操作数进行修改。例如，在

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 表1特征列表   |  |  |  | | --- | --- | --- | | 不。 | 说明 | 类型 | | 1 | #更改的文件数 | 基本特征 | | 2 | #大块头 | | 3 - 6 | #删除/添加/总计/净行数 | | 7 - 10 | #删除/添加/总计/净字符数 | | 11 - 14 | #删除/添加/总计/净条件语句数 | 句法特征 | | 15 - 18 | #删除/添加/总计/净循环数 | | 19 - 22 | #已删除/已添加/总计/net函数调用数 | | 23 - 24 | #总/净修改函数 | | 25 - 28 | #删除/添加/总计/净算术运算符 | | 29 - 32 | #删除/添加/总计/净关系运算符的数目 | | 33 - 36 | #删除/添加/总计/净逻辑运算符数 | | 37 - 40 | #删除/添加/总计/净位运算符的数目 | | 41 - 44 | #删除/添加/总计/净内存运算符的数目 | | 45 - 48 | #删除/添加/总计/净变量 | | 49 - 51 | 大块内的平均/最小/最大Levenshtein距离（提取前） | | 52 - 54 | 大块内的平均/最小/最大Levenshtein距离（提取后） | | 55 | #同一个大块头（抽象之前） | | 56 | #同一个大块头（抽象之后） | | 57 - 58 | #和%受影响的文件 | 语义特征 | | 59 - 60 | #和%的受影响功能 | | 61 | 任何数据相关性更改（True或False） | |

由边界问题引起的漏洞，将>更改为>=或将n更改为n-1。

•安全修补程序更有可能将一段代码移动到另一个地方，而不进行其他更改。例如，将循环中的条件语句移到安全补丁的外部是很常见的。

•在安全修补程序中，相同或类似的更改可能会在不同的函数或文件中多次出现。

因此，我们总结出另外34个句法特征：

•#总/净修改函数。与以前的函数调用不同的是，在change-hunks中用函数名或指针来表示，修改的函数数表示有多少个函数包含change-hunks。这个数字有助于评估斑块的直接影响范围。例如，对于一个函数中包含3个变更块的修补程序，此数字总共计为3个，在网络中计为1个。

•#总的/净的/删除的/添加的基本运算符。我们统计每个补丁中出现的基本运算符的总数和净数，包括算术运算符、关系运算符、逻辑运算符和位运算符。此外，我们还分别计算了删除部分和添加部分中的数字。

•#总共/net/removed/added个内存操作符。我们计算对应的数量的C/C++内存相关的运营商，发生在每个补丁，例如，Maloc，CaloC，Realoc，免费，sieof，等AVE/MI/max LevsHeTin距离内亨克（抽象之前）。Levenshtein距离是相似性的度量[20]。在我们的工作中，Levenshtein distance within a change hunk是将移除的hunk转换为添加的hunk所需的删除、插入和替换的数量。由于在一个斑块中总是有许多大块，因此用它们之间的平均、最小和最大Levenshtein距离来表示斑块的这些特征。•

•平均/最小/最大Levenshtein距离（提取后）。为了进一步衡量每对被移除和添加的大块头的相似性，我们对代码进行了抽象。删除空格和注释后，我们用$替换所有标识符。然后，在这些抽象代码上计算相应的Levenshtein距离。

•#同样的大块头（在抽象之前）。我们认为每两个完全相同的变化的大块头是一对相同的大块头。

•#同样的大块头（抽象之后）。为了计算这对相似的大块头，我们首先去掉完全相同的大块头。然后，在对代码进行抽象之后，我们将每两个相同的抽象变更大块视为一对相似的大块。此外，我们还提出了5个语义特征：

•#和%受影响的文件。受影响的文件数是通过计算调用给定修补程序中已修改函数的文件数来计算的。百分比的计算方法是将受影响的文件数除以总文件数。

•#和%的受影响函数。我们从由Joern[28]生成的控制流图和数据依赖图相结合的代码属性图中计算受影响函数的数目和范围。我们将一个庞克的多个节点合并为一个节点，然后在函数级计算连接节点的个数，得到受影响函数的个数。百分比的计算方法是将受影响的函数数除以函数总数。

•数据相关性的任何更改。将一个单元中的多个节点组合为粗粒度节点后，如果连接的任何节点或连接边中的变量发生变化，则该值为真。否则，就是假的。

### 表二

收集的数据集的性能

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 训练数据集 | 测试数据集 | 总磷（%） | FP（%） | 精密度 | 召回 |
| 2618 | 654 | 266 (79.6%) | 132 (41.3%) | 66.8% | 79.6% |

## B.机器学习模型

在我们的机器学习模型中，我们使用了许多流行的分类算法，包括随机森林[8]、贝叶斯网[5]、随机梯度下降（SGD）[2]、序列最小优化（SMO）[21]和Bagging[3]。但是，每个分类器都不能很好地执行。为了提高性能，我们采用了一种融合上述五种分类器的投票算法来进行多数投票。

我们随机选取80%的安全和非安全补丁数据集，将每一个数据集转换成一个向量，分别对应于上述61个特征的值，其标签为“1”（即安全补丁）或“0”（即非安全补丁）作为输入训练数据。在检测阶段，我们将数据集中剩余的20%补丁转换成向量，然后应用我们的模型。如果向量被指定为“1”，则相应的修补程序被检测为安全修补程序。否则，相应的修补程序将被检测为非安全修补程序。

# 五、系统评价

为了评估我们系统的有效性，我们从三个方面进行了实验。首先，我们将我们的资料库分割成训练和测试资料集来评估我们模型的侦测精确度。其次，我们将我们的模型应用于20个较小版本的openssl1.0.1补丁上，并将我们的结果与以前的工作进行比较。第三，我们将实验扩展到几个流行的SSL库（即OpenSSL、LibreSSL和BoringSSL），以发现一些秘密的安全补丁。

## A.收集数据集的性能

我们随机选择80%的数据集作为训练数据集，剩余的20%作为测试数据集（334个阳性样本和320个阴性样本）。我们采用10倍交叉验证来选择最佳的参数配置。我们的实验是在3.1GHz Intel Core i7 CPU和16GB RAM的机器上进行的。训练阶段（包括2618个补丁）需要42秒，测试阶段（包括654个补丁）需要9秒。表二分别显示了我们测试结果的真阳性（TP）、假阳性（FP）、精确度和召回率。该模型的真阳性率为79.6%，假阳性率为41.3%。

## B.OpenSSL性能

并与其他安全补丁识别系统进行了有效性比较。据我们所知，西班牙[27]是唯一从事这类研究的国家。虽然西班牙专注于二进制级别的补丁分析，但它可以用来识别开放源代码补丁，而且它还对一个开放源代码项目OpenSSL进行了实验，以评估其准确性。为了与西班牙进行比较，我们将我们的模型应用于openssl1.0.1系列的20个次要版本之间的所有补丁（即，从openssl1.0.1到1.0.1s的所有提交）。

表三是西班牙与我们工作的比较。第二列和第三列显示了西班牙和我们手动确定为基本事实的安全和非安全补丁的数量。不同识别号的原因是西班牙只能识别一个函数中控制流变化的补丁，但我们的工作将其扩展到了函数间补丁。实际上，一个补丁可能涉及多个相互影响的函数之间的修改。西班牙可能将多个功能补丁的补丁视为多个补丁，而在我们的工作中，这种补丁被视为一个补丁。此外，我们的方法考虑了头文件的补丁，而西班牙没有。

我们可以识别的安全补丁的百分比比西班牙高8%。另一方面，我们的方法的假阳性率（例如190）高于西班牙（例如47）。我们认为，我们的方法显示出竞争的结果时，与以前的工作，如西班牙。我们的方法能够覆盖函数间补丁、头文件补丁和没有控制流更改的补丁，而西班牙不能。此外，我们的方法在NVD中对各种类型软件的大量漏洞表现出良好的性能和可扩展性，如表II所示。西班牙很难提供类似的性能结果，因为在获取各种二进制代码的基本事实方面需要付出巨大的努力。

## C.案例研究：SSL库

为了在野外识别秘密安全补丁，我们将实验扩展到三个开源SSL库，包括OpenSSL、LibreSSL和BoringSSL。首先，我们应用我们的工具集从GitHub提交的上述三个项目中识别安全补丁。一旦确定了安全修补程序，我们将使用代码克隆算法[23]、[24]来检测该漏洞是否已在其他项目中修补。

表IV总结了这三个SSL库中的12个秘密安全修补程序和修复信息。第一列显示一个项目中每个安全修补程序的CVE ID。每个项目的“修复日期”列从表示修复日期的每个漏洞的修补程序（提交）中的日期获取。灰色背景单元格表示每个漏洞的最早修复日期。破折号表示此漏洞不适用于此项目。每个项目的延迟日是其他项目的第一个固定日期与其固定日期之间的日期差，在此期间攻击者可以对这些类似类型的软件发起“0天”攻击。“尚未”是指项目包含此类漏洞，并且直到现在（2018年6月12日）尚未修复。我们通过手动检查CVE entry或其官方托管网站中的advisory得到第二列到最后一列，然后秘密日期可以计算为CVE ID所属项目的第一个修复日期和advisory发布日期之间的日期差，攻击者可以利用该日期差攻击未修补的版本。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 表III  与以前OPENSSL分析工作的比较   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 技术 | 基本事实 | |  | 检测结果 | |  | | #安全修补程序 | #非安全补丁 | #总磷 | #计划生育 | 精密度 | 召回 | | 西班牙[27] | 323 | 217 | 229 | 47 | 0.83 | 0.71 | | 这项工作 | 294 | 365 | 231 | 190 | 0.55 | 0.79 |   表四  三个SSL库中的秘密安全补丁   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | CVE编号 | OpenSSL | | 利伯雷塞尔 | | 钻孔SSL | | 公开披露日期 | 秘密日 | | 固定日期 | 滞后日 | 固定日期 | 滞后日 | 固定日期 | 滞后日 | | CVE-2018-5407（OpenSSL） | 04/19/2018 | - | 还没有 | 232+ | - | - | 11/15/2018 | 240 | | CVE-2018-0734（OpenSSL） | 10/23/2018 | - | 还没有 | 45+ | 还没有 | 45+ | 10/30/2018 | 7 | | CVE-2018-0732（OpenSSL） | 06/11/2018 | 974 | 06/13/2018 | 972 | 10/11/2015 | - | 06/12/2018 | 973 | | CVE-2018-0739（OpenSSL） | 03/22/2018 | - | 08/06/2018 | 137 | 03/27/2018 | 5 | 03/27/2018 | 5 | | CVE-2017-3731（OpenSSL） | 01/18/2017 | - | 02/01/2017 | 14 | - | - | 01/26/2017 | 8 | | CVE-2016-7053（OpenSSL） | 10/16/2016 | 849 | 07/11/2014 | 21 | 06/20/2014 | - | 11/10/2016 | 874 | | CVE-2016-7052（OpenSSL） | 08/22/2016 | - | - | - | 09/26/2016 | 35 | 09/26/2017 | 35 | | CVE-2016-6305（OpenSSL） | 09/10/2016 | - | 还没有 | 818+ | - | - | 09/22/2016 | 12 | | CVE-2016-6304（OpenSSL） | 09/09/2016 | - | 09/27/2016 | 18 | - | - | 09/22/2016 | 13 | | CVE-2016-6308（OpenSSL） | 09/10/2016 | - | 还没有 | 818+ | - | - | 09/22/2016 | 12 | | CVE-2018-8970（利伯斯尔） | 01/22/2015 | - | 03/22/2018 | 1134 | - | - | 03/24/2018 | 2 | | CVE-2018-12434（利比亚） | 06/19/2018 | 982 | 06/13/2018 | 976 | 10/11/2015 | - | 06/14/2018 | 977 | |

下面是我们观察到的几个有趣的现象。首先，虽然表IV中列出的每个漏洞至少存在于两个项目中，但只为其中一个项目创建了一个CVE ID，其他所有项目只是秘密地修补了这些漏洞。

其次，LibreSSL中的…/crypto/dsa/dsa ossl.c文件允许对ECDSA签名进行内存缓存侧通道攻击。NVD发布此漏洞的日期（CVE-2018-12434）为2018年6月14日，而包含相应补丁的新版本提前一天在LibreSSL的网站上发布。由于这个新版本只做了两次修改，攻击者不难分析并利用它在一天内攻击未修补的版本。更糟糕的是，OpenSSL包含同样的漏洞，一周后（2018年6月19日）它才在GitHub上发布了一个补丁，没有向CVE报告。30天后，一个包含该补丁的新版本在其网站上发布。

当我们试图申请CVE ID时，CVE网站要求我们联系相应的参与CVE编号机构（CNA），即OpenSSL软件基金会。但是，他们答复我们，此漏洞只能导致本地主机端通道攻击，因此不需要CVE。相反，LibreSSL没有参与的CNA，任何人都可以通过直接联系MITRE Corporation为LibreSSL申请CVE ID。我们可以看到，CNA机制为软件供应商提供了一个在不创建CVE ID的情况下秘密修补其漏洞的机会。在这种情况下，当比较OpenSSL和LibreSSL时，用户可能会得出一个有偏见的结论，即OpenSSL更安全，因为它最近的CVE数量较少。

第三，其他项目可能需要很长时间才能意识到那些秘密修补的漏洞并采取行动。CVE2018-0732、CVE-2016-7053、CVE-2016-6308和CVE2018-8970表明，第一次修复和另一个项目的CVE条目分配之间的日期差超过两年。由于这些软件供应商没有良好的渠道来共享新发现的漏洞，攻击者可能会长期滥用这些“0天”漏洞。

最后，软件版本控制过程应该更清楚、更准确地宣布安全修复。在CVE-2018-8970中*x509参数**设置主机*x509vpm.c文件中的函数不支持名称长度为零的特定特殊情况，这会导致主机名验证的无声省略。可以利用此漏洞对欺骗服务器发起中间人攻击，并通过特制的证书获取敏感信息。尽管为LibreSSL分配了一个CVE ID来解决这个问题，但LibreSSL在其更改日志中将其描述为一个bug修复，没有提及任何与安全相关的问题。然而，LibreSSL通常在其更改日志中将所有补丁显式地分类为安全修复、bug修复和新特性。在这种情况下，当用户只关注其更改日志时，它可能会绕过一个修补程序，因为没有漏洞修复程序，并且系统中可以容忍其他小错误。

# 六、 讨论和限制

从识别可疑漏洞到将其真正触发为真正的攻击，可能还有很长的路要走。据报道，并非所有CVE中报告的漏洞都有已知的触发方法[18]。此外，由于触发了许多没有CVE ID的漏洞，这表明并非所有安全补丁都有相应的CVE ID。我们从CVE列表中所有可用的补丁源中收集安全补丁数据库，这可能偏向于某些类型的严重漏洞。例如，我们提到OpenSSL拒绝为本地主机端通道漏洞分配CVE ID，因为他们认为该漏洞的可利用性很低。由于手动触发可疑漏洞并评估其严重性可能需要大量的工作和领域专家，因此我们考虑如何识别可利用的漏洞作为未来的工作。

我们的方法可能被攻击者利用。实际上，我们怀疑攻击者可能已经在某种程度上滥用了秘密补丁。我们的工作目标是促进软件厂商更加规范地维护自己的产品，增加软件厂商之间在信息共享方面的协作，最终消除这种“0天”攻击。

由于未知的秘密安全补丁，非安全补丁数据集可能仍然包含一些安全补丁，这可能会对我们的实验结果造成影响。我们手动检查从非安全补丁数据集中随机选择的1636个补丁中的536个，并根据我们的领域知识确定其中7%为安全补丁。与之前的工作[14][19]类似，我们认为这个比率是可以接受的，因为机器学习能够处理有噪声的数据集。将来，我们将通过从非安全修补程序数据集中删除这些安全修补程序来进一步清理数据集。

我们系统的当前设计使用GitHub上的提交作为补丁的最小单元。实际上，尽管GitHub是最流行的开源软件托管服务提供商之一，但并非所有的开源软件都托管在GitHub上。对于托管在其他网站上的开放源代码软件，只能从相邻版本的源代码的diff文件中获取修补程序。但是，diff文件可能包含许多属于多个修补程序的变更块。对于一个小的diff文件，我们可以简单地通过关键字匹配来分离它。然而，当这个文件很大时，例如，一个主版本引入了许多修改，很难将大块的内容分离成单独的补丁。我们把它作为我们今后的工作。

目前，我们的系统只支持在C/C++中编写的开源项目中的安全补丁。我们的系统可以通过修改特性来适应其他编程语言，例如语法分析相关特性11-56。在我们未来的工作中，我们计划将其扩展到用其他类型编程语言甚至多种编程语言编写的OSS项目。

# 七。相关工作

OSS漏洞检测已经成为一个活跃的研究领域。主要研究方向有两个：漏洞代码相似度比较和漏洞模式识别。对于易受攻击的代码相似性检测，传统的基于标记的技术去除所有空格和注释，并用特定字符替换变量和函数名，以检测仅对标识符、注释和空格进行少量修改的1型和2型代码克隆[22]。基于树的技术[9]，[29]主要是将程序转换为抽象语法树（AST），然后比较最长公共序列（LST）。基于图的技术[12]，[17]使用控制和数据依赖图将代码克隆检测为同构子图。对于脆弱性模式识别，提出了从脆弱代码中提取模式，然后搜索具有相同模式的代码的机器学习或深度学习方法。VulPecker[15]使用不同的功能集来检测不同类型的软件漏洞。VulDeepecker[16]训练了一个中立网络来检测由库/API调用引起的缓冲区溢出和资源管理错误。

一些工作试图创建安全补丁的数据库。Seulbae等人[10]从8个著名的Git存储库中收集数据，Zhen等人[15]从19个产品中构建了漏洞修补程序数据库（VPD）。然而，这些数据集的规模还不足以进行基于机器学习的研究，与开源项目上成千上万的CVE条目相比。尽管Li等人[14]基于NVD[4]中与Git相关的记录建立了一个大规模的安全补丁数据库，但他们的数据库尚未向公众开放。

某些秘密安全补丁在不同的研究中被临时报道。Zhen等人[16]在Qemu中披露了CVE-2016-9104之后，发现Xen默默地修补了这个漏洞。他们的研究结果还揭示了Seamonkey和Firefox（CVE-2015-4517）以及Libav和FFmeng（CVE-2014-2263）之间的秘密安全补丁。它促使我们对秘密安全补丁进行研究。

斑块分析已经引起了一些研究者的关注。Zame等人[30]对Mozilla Firefox中安全补丁和性能补丁的区别进行了案例研究。Perl等人[19]展示了漏洞贡献提交和其他提交之间的许多统计差异。但是，他们无法区分漏洞修复和非安全漏洞修复。Frank等人[14]首次对安全补丁和非安全漏洞修复进行了大规模的实证研究，并对安全补丁的基本特征进行了分析。Xu等人[27]提出了一种通过对执行跟踪进行语义分析来识别安全补丁的可扩展方法。但是，它不能处理跨功能的安全修补程序，并且不能很好地识别与安全修补程序类似的非安全修补程序。

# 八。结论

本文开发了一个基于机器学习的安全补丁识别系统。开发人员和用户可以使用我们的系统来帮助识别秘密的安全补丁，并决定是更新到新版本还是应用补丁。我们指出，一旦安全补丁被识别，它对应的漏洞应该在其他类似类型的软件中被检测到，如果被识别，这个补丁可以用来修补类似的漏洞。为了评估我们模型的有效性，我们建立了一个由CVE列表中的安全补丁组成的数据库。我们将其开源，以促进公众对提高OSS安全性的研究。我们发现了一组语法和语义代码特征来分析安全补丁。实验结果表明，本系统能够取得良好的检测性能。我们还将我们的系统应用于三个开源SSL库项目，并发现了12个秘密安全补丁。

## 参考文献

[1] “Equifax官方没有借口https://https://www.wired.com/story/equifax-break-no-forecy/。

[2] L.Bottou，“具有随机梯度下降的大规模机器学习”，COMPSTAT&apos;2010年论文集。斯普林格，2010年，第177-186页。

[3] L.Breiman，“装袋预测”，机器学习，第24卷，第2期，第123-140页，1996年。

[4] 常见漏洞和暴露（CVE），https://cve.mitre.org/cve/identifiers/index.html。

[5] N.Friedman、D.Geiger和M.Goldszmidt，“贝叶斯网络分类器”，机器学习，第29卷，第2-3期，第131-163页，1997年。

[6] Github，“2018年octoverse的状态”https://octoverse.github.com。[7] GNU困难，https://www.gnu.org/software/diffutils/。

[8] 胡锦松，“随机决策森林”，载《文献分析与认识》，1995年，第三届森林问题国际会议论文集，第一卷。IEEE，1995年，第278-282页。

[9] L.Jiang，G.Misherghi，Z.Su和S.Glondu，“Deckard:基于树的可伸缩和精确的代码克隆检测”，第29届国际软件工程会议论文集。IEEE计算机学会，2007，第96-105页。

[10] S.Kim、S.Woo、H.Lee和H.Oh，“Vuddy:易受攻击代码克隆发现的可扩展方法”，安全与隐私（SP），2017 IEEE研讨会。IEEE，2017年，第595-614页。

[11] J.C.Knight和N.G.Leveson，“多版本编程中独立性假设的实验评估”，IEEE软件工程学报，第1期，第96-109页，1986年。

[12] J.Krinke，“用程序依赖图识别相似代码”，逆向工程，2001年。诉讼程序。第八次妇女问题工作会议。IEEE，2001年，第301–309页。

[13] R.G.Kula、D.M.German、A.Ouni、T.Ishio和K.Inoue，“开发人员是否更新了他们的库依赖关系？“经验软件工程，第23卷，第1期，第384-4172018页。

[14] F.Li和V.Paxson，“安全补丁的大规模实证研究”，2017年ACM SIGSAC计算机和通信安全会议论文集。ACM，2017年，第2201-2215页。

[15] 李泽民、邹德文、徐国生、金海华、齐海华、胡俊杰，“Vulpecker:一个基于代码相似度分析的自动漏洞检测系统”，载于第32届计算机安全应用年会论文集。ACM，2016年，第201-213页。

[16] 李志明，邹德明，徐世旭，欧旭，金浩，王世旺，邓子忠，“Vuldeepecker:一个基于深度学习的漏洞检测系统”，arXiv预印本附件十四：1801.01681, 2018.

[17] 刘国强，陈国强，韩国强，余培生，「Gplag:程式相依图分析法侦测软体剽窃」，第十二届ACM SIGKDD国际知识发现与资料挖掘会议论文集。ACM，2006年，第872-881页。

[18] D.Mu、A.Cuevas、L.Yang、H.Hu、X.Xing、B.Mao和G.Wang，

“理解人群报告的安全漏洞的再现性”，第27期{}{}。USENIX，2018年，第919-936页。*乌塞尼克斯安全研讨会(乌塞尼克斯安全（18）*

[19] H.Perl、S.Dechand、M.Smith、D.Arp、F.Yamaguchi、K.Rieck、S.Fahl和Y.Acar，“Vccfinder：寻找开源项目中的潜在漏洞以协助代码审计”，第22届ACM SIGSAC计算机和通信安全会议论文集。ACM，2015年，第426-437页。

[20] 皮特塞和布莱克，算法和计算理论手册。CRC出版社，1999年。

[21]J.C.Platt，“12使用序列最小优化快速训练支持向量机”，《核方法进展》，第185–208页，1999年。

[22]C.K.Roy和J.R.Cordy，“软件克隆检测研究调查”，皇后计算学院TR，第541卷，第115期，第64-68页，2007年。

[23]SIM卡，https://dickgrune.com/Programs/similarity测试员/。

[24]猿猴，https://www.harukizaemon.com/simian/。

[25]Snyk，“2018年开放源码安全状况”https://snyk.io/stateofsecurity/。

[26]Y.Tian、J.Lawall和D.Lo，“识别linux错误修复补丁”，第34届国际软件工程会议论文集。IEEE出版社，2012年，第386-396页。

[27]Z.Xu，B.Chen，M.Chandramohan，Y.Liu，和F.Song，“西班牙：二进制文件的安全补丁分析，以理解疼痛和药丸”，第39届国际软件工程会议论文集。IEEE出版社，2017年，第462-472页。

[28]F.Yamaguchi，N.Golde，D.Arp和K.Rieck，“用代码属性图建模和发现漏洞”，安全和隐私（SP），2014年IEEE研讨会。IEEE，2014年，第590-604页。

[29]W.Yang，“识别两个程序之间的语法差异”，《软件：实践与经验》，第21卷，第7期，第739-7551991页。

[30]S.Zaman、B.Adams和A.E.Hassan，“安全与性能缺陷：firefox案例研究”，第八届挖掘软件存储库工作会议论文集。ACM，2011年，第93-102页。

[[1]](" \l "_ftnref1" \o ")数据集位于https://github.com/SecretPatch/Dataset