DOI:10.7544/issn1000-1239.20200492
59(1):197-208,2022

一种 Linux 安全漏洞修复补丁自动识别方法

周 鹏[1,2] 武延军[1,3] 赵 琛[1,3]
[1](中国科学院软件研究所 北京 100190)
[2](中国科学院大学 北京 100049)
[3](计算机科学国家重点实验室(中国科学院软件研究所) 北京 100190)
(zhoupengwork01@163.com)

Identify Linux Security Vulnerability Fix Patches Automatically

Zhou Peng[1,2], Wu Yanjun[1,3], and Zhao Chen[1,3]
[1](*Institute of Software*, *Chinese Academy of Sciences*, *Beijing* 100190)
[2](*University of Chinese Academy of Sciences*, *Beijing* 100049)
[3](*State Key Laboratory of Computer Science* (*Institute of Software*, *Chinese Academy of Sciences*), *Beijing* 100190)

Abstract It is critical to catch and apply the vulnerability fix patches in time to ensure the security of information system. However, it is found that open source software maintainers often silently fix security vulnerabilities. For example, 88% of maintainers delay informing users to fix vulnerabilities in the release notes of new software version, and only 9% of the bug fixes clearly give the corresponding CVE ID, and only 3% of the fixes will actively notify the security service provider in time. In many cases, security engineers can't directly distinguish vulnerability fixes, bug fixes, and feature patches from the code and log message of patches. As a result, vulnerability fixes can't be identified and applied by users timely. At the same time, it is costly for users to identify vulnerability fixes from a large number of patch submissions. Taking Linux as an example, this paper presents a method of identifying vulnerability patches automatically. This method defines features for the code and log message from patches, builds machine learning model, and trains to learn classifiers that can distinguish vulnerability patches. Experiments indicate that our approach is effective, which can get 91.3% precision, 92% accuracy, 87.53% recall rate, and reduce the false positive rate to 5.2%.

Key words identify vulnerability fix patches automatically; security vulnerability fixes; Linux kernel; machine learning; open-source software community

摘 要 及时获取并应用安全漏洞修复补丁对保障服务器用户的安全至关重要.但是,学者和机构研究发现开源软件维护者经常悄无声息地修复安全漏洞,比如维护者88%的情况在发布软件新版本时才在发行说明中告知用户修复了安全漏洞,并且只有9%的漏洞修复补丁明确给出对应的CVE(common vulnerabilities and exposures)标号,只有3%的修复会及时主动通知安全监控服务提供者.这导致在很多情况下,安全工程师不能通过补丁的代码和描述信息直接区分漏洞修复、Bug修复、功能性补丁.造成

收稿日期:2020-06-16;**修回日期**:2021-01-27
基金项目:国家重点研发计划项目(2018YFB0803600);中国科学院战略性先导科技专项(Y8XD373105);中国科学院前沿科学重点研究计划项目(ZDBS-LY-JSC038)
This work was supported by the National Key Research and Development Program of China (2018YFB0803600), the Strategic Priority Research Program of Chinese Academy of Sciences (Y8XD373105), and the Key Research Program of Frontier Sciences, CAS (ZDBS-LY-JSC038).

漏洞修复补丁不能被用户及时识别和应用,同时用户从大量的补丁提交中识别漏洞修复补丁代价很高.以代表性 Linux 内核为例,给出一种自动识别漏洞修复补丁的方法,该方法为补丁的代码和描述部分分别定义特征,构建机器学习模型,训练学习可区分安全漏洞补丁的分类器.实验表明,该方法可以取得91.3%的精确率、92%的准确率、87.53%的召回率,并将误报率降低到 5.2%,性能提升明显.

关键词 漏洞修复补丁自动识别;安全漏洞修复;Linux 内核;机器学习;开源软件社区

中图法分类号 TP311

随着软件系统复杂度的增加、开源软件日益广泛的应用对漏洞(vulnerability)披露的透明度和效率提升、各国政府和企业对软件安全愈加重视而增加投入,软件漏洞的披露数量逐年增加[1-2].同时软件正渗透到经济、社会生活的方方面面,Linux 内核、基础库(如 OpenSSL)被不同软件系统广泛共用,其安全漏洞被利用往往波及到广泛的软件系统和大量的用户[3-4].因此帮助用户及时应用安全漏洞补丁修复至关重要.

观察开源社区安全漏洞生命流程,总结科研工作者和企业安全漏洞阶段划分惯例[1,4-6],开源软件漏洞生命周期主要分为 4 个阶段:1)发现新漏洞;2)开发发布漏洞修复补丁;3)将漏洞修复补丁通知用户;4)用户应用漏洞修复补丁到软件系统.我们对这 4 个阶段进行分析发现,软件漏洞生命周期第 3 阶段很容易被忽视并存在安全提示信息不足,进而造成第 4 阶段存在明显的管理不足,影响安全补丁的及时应用从而给用户带来安全威胁.

第 1 阶段:该阶段已经引起学者们的广泛关注,比如漏洞检测、漏洞挖掘、漏洞分析、漏洞利用(是评估新漏洞的重要手段之一)一直是安全领域的研究热点,持续受到科研、企业、政府机构等密切关注[1,7-8].

第 2 阶段:该阶段反应快速,易获得优先处理.新安全漏洞一旦被发现确认,开源社区维护者(maintainer)会积极做出反应,快速开发、评审、处理漏洞修复补丁[5],例如文献[9]统计发现安全漏洞的修复反应速度比性能 Bug 快 2.8 倍,安全漏洞分配的开发者人数是性能 Bug 的 3.51 倍,是其他 Bug 的 4.7 倍;文献[1]跟踪分析的包括 Linux 内核、Ubuntu 等 600 多个开源项目和 Snyk 安全公司发布的开源软件安全状态工作报告[5],统计发现维护人员对待安全漏洞和安全修复补丁的响应速度、重视程度要明显高于其他 Bug.

第 3,4 阶段:第 2 阶段研究表明通过补丁注释特别是补丁的 commit message 明确告知补丁是修复安全漏洞、明确给出对应的 CVE ID(common vulnerabilities and exposures identifiers)[10],对软件的用户(安全软件工程师、操作系统发行版、软件集成商、运维工程师等)识别安全漏洞修复补丁,尽早应用到软件系统至关重要.但是,研究发现开源软件维护者经常悄无声息地修复安全漏洞,即漏洞修复补丁的代码部分和 commit message 中未给出安全描述和 CVE ID 信息是很常见的.比如研究报告[5]统计发现,在漏洞修复补丁发布时,维护者在文件中明确标注 CVE ID 的只占 9%,及时主动通知安全监控服务提供者只有 3%,88%的维护者在新版本发布的发布说明里才通知修复了安全漏洞,但是相对于安全补丁的发布,软件新版本发布要滞后很多,而且很多生产环境的基础服务,出于安全性、稳定性考虑,管理员更偏向于补丁升级而非升级整个软件版本,管理员不得不去识别具体的安全修复补丁;同时我们编写工具统计了收录到 NVD[11]数据库的 Linux 内核全部漏洞(2002—2020 年 5 月 23 日),Linux 内核一共有 4 064 个被 NVD 数据库收录确认的漏洞,然后我们的工具追踪 NVD 和 Linux 内核源码仓库,发现修复补丁被登记到 NVD 追踪的只有 1 633 个,即 NVD 已经确认收录的 Linux 内核漏洞,其中只有 40.2%的内核漏洞修复补丁登记关联到 NVD 数据库,从而可以 NVD 数据库反向识别;我们的工具直接追踪 Linux 内核源码仓库,发现其中只有 380 个漏洞修复补丁的提交信息给出了 CVE ID 等漏洞描述信息,仅占 Linux 内核 NVD 确认漏洞总数的 9.35%,占 NVD 反向可追踪数的 23.27%,因此 Linux 内核维护者在安全修复补丁文件中明确标注 CVE ID 的情况跟开源软件整体情况一样比例很低.由此用户(即便是专业的安全工程师)识别安全漏洞修复补丁,从而避免遗漏和尽早应用安全补丁面临挑战.

基于这些观察分析,开发安全修复补丁的智能化自动识别工具,实现对漏洞修复补丁的及时识别、及时通知、降低人工识别的工作量和遗漏,从而促进安全补丁的及时应用是本文的主要研究动机.

因此,本文以代表性的 Linux 内核源码社区为案例,设计实现了一种 Linux 安全漏洞补丁的自动化识别方法.该方法核心思想是为合并到 Linux 内核源码的补丁的代码和描述部分分别定义漏洞特征,构建机器学习模型,训练学习可识别安全漏洞补丁的分类器.该方法的实现主要包括 3 个步骤:1)如何持续收集和标注安全漏洞补丁和非安全补丁;2)跟其他的 Bug 修复、功能增强补丁相比,分析安全漏洞修复补丁的特点,对原始补丁文件进行特征定义和特征抽取;3)设计、实现机器学习模型,使用收集的数据集训练生成可识别漏洞修复补丁的分类器.

本文的主要贡献包括 3 个方面:

1) 给出了一种可以自动识别漏洞修复补丁的机器学习建模方法,并详细给出了如何做 Code 特征、Log 特征的定义与提取,以及联合 Code 和 Log 如何基于半监督的 Co-Training 方法建模.以代表性的 Linux 内核为应用实例,实验表明该方法将识别精确率提升到 91.3%,准确率到 92%,召回率到 87.53%,误报率降低到 5.2%,因此具有实用价值.并且该方法有很好的可扩展性,可以直接扩展支持其他开源项目.

2) 实现了一个可持续收集和标注安全漏洞修复补丁和 Bug 修复补丁的工具系统,并给出其设计与实现细节,该系统可以扩展支持其他以 Git,SVN 等版本控制工具管理的开源软件.

3) 本文研究也启示,不应盲目相信,或过度依赖 NVD、开源软件社区漏洞通告,因为其全面性、有效提示性和及时性存在不足.分析表明其通告质量存在漏、错、未更新等问题.这要求我们需结合人工和自动化识别工具去主动识别漏洞修复补丁,及时修复.

1　相关工作

近年来,使用机器学习技术对程序源代码进行分析和处理成为研究热点之一,其中分析软件漏洞、Bug 修复是重要的研究方面.下面介绍跟本文最相关的工作,以及我们的工作跟相关工作的区别.

文献[12]提出了使用补丁的说明信息(Log message)和 Bug 报告抽取特征自动识别安全漏洞补丁的研究方法,该类方法只使用补丁的自然语言相关的素材信息,不涉及代码;我们的方法同时从补丁的 Log message 和 Code 抽取特征,结合 2 方面的特征信息构建模型,因此在建模的特征工程和建模方法上都明显不同.

文献[13]提出了使用补丁的 Log message 和 Code 部分抽取特征识别漏洞修复补丁的方法,我们虽然都使用了补丁的 Log message 和 Code 部分,但是我们的特征工程和建模方法有明显区别,比如文献[13]将 Code 只做纯文本对待抽取关键字,然后跟 Log 自然语言部分一起合并;而我们的方法将 Code 部分按照编程语言进行分析,抽取其标识符、循环、代码修改等细粒度信息构建特征,并且我们将 Code 和 Log 部分的特征分开构建机器学习模型,因此建模方法也明显不同.

文献[14]提出了结合使用补丁的统计量(局部性、复杂度、控制循环)结合关键字抽取特征,然后用机器学习的方法对安全修复补丁进行更细粒度分类,如划分为 Buffer 错误(如栈溢出)、Injection 错误(如错误注入)、Numeric 错误(如整数溢出)等,我们都是借助补丁的统计特征建模,但我们的研究目标和构建模型的方法完全不同,从其实验效果可知文献[14]细粒度补丁类型分类有效但并不显著(其最佳准确率是 54.75%),因此并不能达到本文研究目标的可实用的程度.

文献[15]提出一种从 Bug 修复补丁中识别出漏洞修复的研究方法(漏洞修复补丁是 Bug 补丁的子类,所以我们在本文中称纯 Bug 修复补丁以做区分.),该方法使用 LLVM 编译器将补丁修改的文件源码预编译为 IR,然后使用手动建立的补丁模式(安全操作、关键变量、漏洞操作)对 IR 对照补丁模式进行预处理收集到模式数据,最后将收集的数据作为约束提供给手动定义的符号规则做约束求解,求解结果做出是否为安全补丁的判断.该方法为应用程序的语法信息提供了一种可借鉴的思路,但其每个步骤都需要人工建立模式、规则,导致人工负担较重,规则和模式的完备性需要验证,这限制了其应用规模.因此我们的研究方法和应用规模都有明显差异.

文献[1,9,16-17]从经验软件工程角度对不同补丁的统计特性进行了大量实证研究,研究表明功能补丁、Bug 修复补丁、安全漏洞修复补丁表现出统计上的显著差异,这些经验是本文对代码补丁做特征工程时进行特征定义与选取的主要依据.但本文工作跟这些研究的研究方法和目标都是不同的.

文献[18-19]等标注源码的研究方法,我们的研究目标和研究方法都差别很大.

2 设计与实现

2.1 补丁持续收集与标注方法

每个源代码补丁(简称补丁)由 Log message 和 Code 部分组成,前者是自然语言描述的补丁性质和行为,后者指定对源代码文件的修改(增、删、替换).开源软件是以补丁合并(merge)的方式发展演化,可分为 Bug、功能、漏洞修复 3 类补丁.从 Linux 内核社区直接获取补丁是很容易的;但是,因为补丁描述信息和补丁代码有很强的随意性,根据补丁信息对补丁进行直接自动化归类(标注)比较困难.而模型训练和更新需要足够的标注补丁,这就要求给出能够持续收集、标注补丁的方法,并设计开发自动化工具.

2.1.1 收集标注漏洞修复补丁的方法

因为直接根据补丁提交信息从 Linux 仓库收集安全漏洞补丁识别困难,不可靠.而 NVD 数据库里分配了 CVE ID 的漏洞是经过确认而具有可靠性.因此,我们采取了从 NVD 数据库解析信息,然后反向追踪 Linux 内核的 Git 源码仓库[20]的方法,其方法流程如图 1 所示:

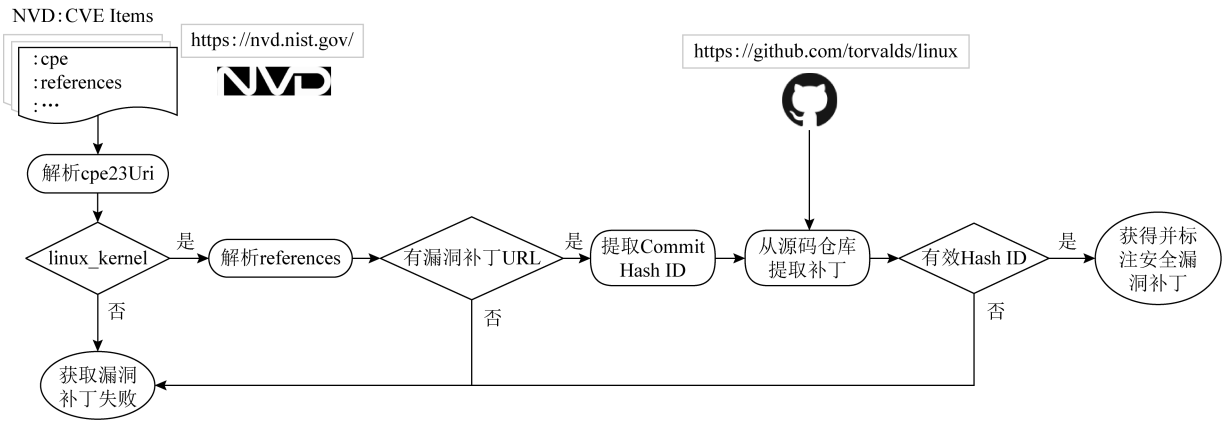

Fig. 1 Continuously collect and label vulnerability fix patches of the Linux kernel

图 1 持续收集、标注 Linux 内核漏洞修复补丁

1) 从 NVD 站点自动下载 NVD 安全漏洞数据,每天更新一次.

2) 数据以 JSON 格式存储,以名字为"CVE_Items"的 JSON 数组管理,数组的每一项是一个被确认的安全漏洞的信息,图 2 是一个 CVE Item 格式和记载字段的例子,这里只列出重要字段,省略大部分字段.

3) 通过解析 CPE[21]"cpe23Uri"字段判断该漏洞是否属于 linux_kernel.

4) 通过解析"references"字段的所有选项,匹配是否存在"git.kernel.org"或"github.com/torvalds/linux"的 url,如果有则表明可能提供了补丁链接."references"字段一般有多项,是描述该漏洞情况的来自于互联的相关说明链接.

5) 解析漏洞补丁 URL,提取其中的 Commit Hash ID 字段,Linux 内核是使用 git 管理的代码仓,每次 commit 补丁提交对应一个唯一的 Hash ID.

6) 通过 Hash ID 从 Linux 内核仓的主干分支(Linus Torvalds 维护)提取相应的补丁.

```
…
"CVE_Items":[{
  "cve":{
    "CVE_data_meta":{
      "ID":"CVE-2020-11609"
    }
    "references":{…
      "url":"https://git.kernel.org/cgit/linux/kernel/git/torvalds/
       linux.git/commit/? id=485b06aadb933190f4bc44e006076-
       bc27a23f205",
    }
  },
  "configurations":{
    "nodes":[{
      "cpe_match":[{"cpe23Uri":"cpe:2.3:o:linux:linux_
       kernel:*:*:*:*:*:*:*:*",}]
    }…]
  },
  "impact":{…
    "baseMetricV2"{"baseScore":4.9,…}
  },…
},{…}]…
```

Fig. 2 An example of CVE security vulnerability

图 2 一个 CVE 安全漏洞例子

7) 如果提取有效,将获得一个准确的安全漏洞

修复补丁,可以很确定地标注为漏洞补丁,加入标注数据集.注意由于失效、报告漏洞时错误等原因,存在提取的 Hash ID 是无效的情况,因此需要检测 Hash ID 是否有效.

通过以上方法流程,我们可以巧妙地为 Linux 内核自动收集、标注可靠的漏报补丁数据集,并可持续更新.显然该漏洞补丁标注方法很容易扩展支持其他开源项目.

2.1.2　收集标注功能和 Bug 修复补丁的方法

该部分工作是收集明确不是安全漏洞修复的补丁.其方法是收集可以明确是功能增强或纯 Bug 修复的补丁.这些补丁最后都统一标注为“nonvulns”.

收集明确的 Bug 修复补丁:首先通过“fix(fix 的派生词)”过滤 Log message 做初步筛查,然后剔除掉有安全关联敏感关键字的补丁[12],然后剔除掉在被明确做了安全标注的补丁,最后剔除掉有引用安全机构链接的补丁.

收集功能增强补丁:Log message 出现 Add support,Clean,Clean up,New feature 等关键字,同时不属于收集的安全漏洞修复、Bug 修复补丁.

2.1.3　收集未标注补丁

收集无法做出安全漏洞标注、Bug 修复标注、功能增强标注线索的补丁.这类补丁的分类标注为“unknowns”.

2.2　特征定义与特征提取

如图 3 所示提交到 Linux 内核的修复 CVE-2020-11494 安全漏洞的补丁,可以看出该补丁并未给出可直接引人注意的安全提示信息.该补丁包括 Log message 文字说明部分(行⑤～⑲所示)和代码修改描述部分(行㉑～㊷所示).前者是自然语言,后者是严格的代码规则语言,差别很大;因此需分别定义特征,分别给出特征提取方法.

2.2.1　补丁的代码部分

图 3 行㉑～㊷是代码样例.其遵循的是“合并格式”规范[22].其中行㉑～㉔给出被修改的一个源文件名字,接下来给出修改块(hunk),每个块以“@@..@@”开头给出修改涉及的行(“－”表示文件修改前版本,“＋”表示文件修改后版本,从行 148 开始连续 7 行),代码行涉及的上下文(默认是相邻的前后 3 行),“－”开头表示删除的行,“＋”开头表示增加的行.比如图 3 示例有 2 个修改块,分别是行㉕～㉝和行㉞～㊷.本文利用这些格式定义构建补丁的代码部分的特征.

```
① commit b9258a2cece4ec1f020715fe3554bc2e360f6264
② Author:Richard Palethorpe<rpalethorpe@suse.com>
③ Date:Wed Apr 1 12:06:39 2020+0200
④
⑤    slcan:Don't transmit uninitialized stack data in padding
⑥
⑦    struct can_frame contains some padding which is not explicitly
       zeroed in
⑧    slc_bump. This uninitialized data will then be transmitted if
       the stack
⑨    initialization hardening feature is not enabled(CONFIG_INIT_
       STACK_ALL).
⑩
⑪    This commit just zeroes the whole struct including the padding.
⑫
⑬    Signed-off-by:Richard Palethorpe<rpalethorpe@suse.com>
⑭    Fixes:a1044e36e457("can:add slcan driver for serial/USB-
       serial CAN
⑮ adapters")
⑯    Reviewed-by:Kees Cook<keescook@chromium.org>
⑰    Cc:linux-can@vger.kernel.org
⑱    …
⑲    Signed-off-by:David S. Miller<davem@davemloft.net>
⑳
㉑ diff --git a/drivers/net/can/slcan.c b/drivers/net/can/slcan.c
㉒ index 086dfb1b9d0b..91cdc0a2b1a7 100644
㉓ ---a/drivers/net/can/slcan.c
㉔ +++b/drivers/net/can/slcan.c
㉕ @@-148,7+148,7@@ static void slc_bump(struct slcan *sl)
㉖       u32 tmpid;
㉗       char *cmd=sl->rbuff;
㉘
㉙ -     cf.can_id=0;
㉚ +     memset(&cf, 0, sizeof(cf));
㉛
㉜       switch(*cmd){
㉝       case 'r':
㉞ @@-187,8+187,6@@static void slc_bump(struct slcan *sl)
㉟       else
㊱         return;
㊲
㊳ -     *(u64 *)(&cf.data)=0; /*clear payload */
㊴ -
㊵       /*RTR frames may have a dlc>0 but they never have any
           data bytes */
㊶       if(!(cf.can_id&CAN_RTR_FLAG)){
㊷         for(i=0;i<cf.can_dlc;i++){
```

Fig. 3　A patch example fixing vulnerability CVE-2020-11494

图 3　一个 CVE-2020-11494 漏洞修复补丁例子

经验软件工程有对不同补丁的统计特性进行了大量实证研究[1,9,16-17],这些研究表明功能补丁、Bug 修复补丁、安全漏洞补丁的代码修改,在很多方面(如文件数量、局部性、补丁复杂度)表现出统计上的差异;统计发现安全漏洞修复补丁表现出补丁更小、做更少的逻辑改动、偏向局部性(如文件局部性-改动涉及文件更少、函数局部性)等特征.本文综合相关经验研究发现的统计规律和 Linux 内核安全漏洞

经验,设计构建可定量计算的特征指标,并给出每个特征的具体提取方法,归纳总结如下.

$F01$:跨越的函数体个数,度量局部性.

$F02$:改动的文件数,度量局部性.

$F03$:修改块(hunk)数,局部性和复杂性.

$F04$:增加的行数,逻辑复杂性.

$F05$:删除的行数,逻辑复杂性.

$F06$,$F07$:增加/删除 IF 分支个数,逻辑复杂性.

$F08$,$F09$:增加/删除循环数(for,while,goto),逻辑复杂性.

$F10$,$F11$:增加/删除括号表达式,逻辑复杂性.

$F12$,$F13$:新增/删除文件数,局部性,分别通过匹配"---/dev/null""+++/dev/null"模式统计.

$F14$,$F15$:增加/删除 sizeof、取变量地址、指针运算操作.

$F16$ 至 $F19$:增加/删除函数调用数,逻辑复杂性.

$F20$,$F21$:增/删函数返回(return),复杂性.

$F22$ 至 $F25$:增加/删除的 continue,break 控制数.

$F26$,$F27$:增加/删除复制操作行数,复杂性.

$F28$,$F29$:增加/删除关系运算(>,!=,<,|,&& 等).

$F30$,$F31$:增加/删除宏数,如 #ifdef,#define,#undef,#endif,#elif.

$F32$:$|F04-F05|$,增加的减去删除的,取绝对值.

$F33$:$F04+F05$,增加的加上删除的.

$F34$ 至 $F54$:特征 $F06$ 至 $F11$、$F14$ 至 $F31$ 是以类似计算 $F32$ 和 $F33$ 的方式计算的特征.

2.2.2 补丁的 Log message 部分

补丁的 Log 部分(图 3 行⑤~⑲是 Log message 一个实例)一般对补丁的摘要、动机、改动、贡献者、参考链接等进行说明;是英语表达的自然语言文本.因此可以使用自然语言处理(NLP)技术进行特征提取.本文用词袋(BoW)[23]模型对补丁的 Log message 进行特征抽取.特征抽取流程如图 4 所示:

Fig. 4 Flow of feature extraction for commit log messages

图 4 补丁提交日志信息特征抽取流程

特征抽取流程的详细说明如下:

1) Log message 文本抽取.提取每个补丁的 Log message 部分,并以 commit Hash 字符串命名文件存储,这些文件按照 vulns,nonvulns,unknowns 这 3 个目录组织管理.

2) 数据预处理.原始 Log message 文本包含噪音干扰信息,需在特征抽取前做清理.包括:剔除作者行(Signed-off-by,Reviewed-by,Acked-by 等);统一小写(避免大小写增加词汇库);stemming 处理把派生词统一到词根减小词汇库,如 fixed,fixing,fixes 都替换为 fix,因为派生词的提示信息是一样的;停用词(stop words)处理,剔除掉英语语言中信息量低的高频词,如 his,hers,at,just,my,and 等;最后 Log message 文档的多行合并为一行,作为语料库的一个文档单元.

3) 构建语料库.以前面步骤预处理后在 vulns,nonvulns,unknowns 目录下管理的所有 Log 文档为文档单元构建 BoW 语料库,该语料库建设过程包括:收集所有的词汇;统计语料库全局空间词频(global term frequency),记为 TF_g;词频最高的前 50 的词汇构成词汇库($Vocabulary$);计算 50 个词在语料库全局空间的逆文档频率(inverse document frequency),记为 IDF.选取的 50 个词构成了每个 Log 文档的特征分量.

语料库中词频为

$$TF_{gi}=\frac{n_{gi}}{N_g}=\frac{n_{gi}}{\sum_j n_{gj}}, \tag{1}$$

其中 n_{gi} 是单词 i 在所有 Log 文档中出现次数,N_g 是语料库中所有单词出现次数的和.

IDF 的计算需要先计算 DF(词汇的文档频率):

$$DF_i=|D_i|, \tag{2}$$

其中 $D_i=\{d_j\,|\,t_i$ 是语料库中的单词,$t_i\in d_j\}$.

$$IDF_i=\ln\frac{|D|+1}{DF_i+1}+1, \tag{3}$$

其中,$D=\{d_1,d_2,\cdots,d_i,d_{i+1},\cdots,d_n\,|\,d_i$ 是 Log 文档,n 是文档总数$\}$.

4) 生成 Log message 特征.有了第 3 步统计学习得到的语料库的 $Vocabulary$,TF_g,IDF 信息,可以为每个 Log message 计算生成其 TF-IDF 特征表示(记为 $TFIDF$).任意文档 d_i 抽取的特征表示为

$$TFIDF_i = \| [TFIDF_{i1}, TFIDF_{i2}, \cdots, TFIDF_{ij}, \cdots, TFIDF_{iV}] \|_2, \quad (4)$$

其中 $V=|Vocabulary|=50$,特征权重分量 $TFIDF_{ij}$ 为

$$TFIDF_{ij} = TF_{ij} \times IDF_j, \quad (5)$$

其中 TF_{ij} 是词汇单词 j 在文档 d_i 中出现的次数.

流程的步骤 4)完成了对 Log message 文档的特征抽取,抽取 50 个特征,每个 Log 文档用 50 维的向量表示,向量权重分量是相应特征的 TF-IDF 值表示.

2.3　漏洞补丁分类器模型

收集的训练数据集包括标注数据集(vulns, nonvulns)和未标注数据集(unknowns),其中未标注数据数量远多于标注数据集,因此我们尝试采取半监督(semi-supervisied)的 Co-Training[24-25] 方法建模.将 vulns 视为正数据点 P,将 nonvulns(包括功能和单纯的 Bug 修复补丁)视为负数据点 N,将 unknowns 视为未标注数据点 U,P 和 N 构成了标注数据集 L.

算法 1 描述了训练可识别安全漏洞补丁分类器的流程,该算法执行过程中我们设置 $n_P=2$,$n_N=4$,$n_U=80$,$iter=30$;算法 2 是使用由算法 1 学习的分类器做预测的流程,因为分类器有 2 个构成,分别是基于补丁的 Code 特征训练和基于补丁的 Log 特征训练得到,该算法描述了在预测推理阶段如何整合这 2 个分类器构建 1 个统一安全漏洞补丁识别分类器.

注意:算法描述中的[begin:end]操作是从特征集中截取从 begin 索引开始到 end 结束的部分特征,end 未指定表示到序列结尾,begin 未指定表示从序列开头;算法中用到的子分类器模型是基于 SVM LinearSVC[26] 构造.

算法 1. 模型训练过程.

输入:***XL*** 为标注样本,***y*** 为补丁样本标注,***XU*** 为未标注样本,n_P 为每次迭代各分类器从 ***XU*** 中取出标注为正样本的个数,n_N 为取出标注为负的个数,n_U 为未标注样本缓冲池大小,$iter$ 为训练迭代次数;

输出:$clfCode$ 是用补丁的代码特征训练的分类器,$clfLog$ 是补丁的 Log 特征集训练的分类器.

① FUNCTION: $runTrain$ (***XL***, ***y***, ***XU***, n_P, n_N, n_U, $iter$) /* 将补丁的特征集分为 Code 和 Log 特征集 */

② ***XLCode*** = ***XL***[0:$CodeFeatureCount$],
XLLog = ***XL***[$CodeFeatureCount$:];
/* 使用 SVM LinearSVC 建模初始化分类器 */

③ $clfCode \leftarrow LinearSVC$, $clfLog \leftarrow LinearSVC$;

④ $c=0$; /* Co-Training 训练迭代次数计数器 */

⑤ ***UP*** ← 从 ***XU*** 随机取 nu 个元素构建缓冲池;

⑥ WHILE($c<iter$ 并且 ***XU*** 非空)/* 迭代训练使用补丁的 Code 特征集训练分类器 */

⑦ $clfCode \leftarrow Train(clfCode, \mathbf{XLCode}, \mathbf{y})$; /* 使用补丁的 Log 特征集训练分类器 */

⑧ $clfLog \leftarrow Train(clfLog, \mathbf{XLLog}, \mathbf{y})$; /* Code 分类器对池子样本做预测 */

⑨ $yPredByCode \leftarrow Infer(clfCode, \mathbf{UP}[0:CodeFeatureCount])$; /* Log 分类器对池子样本做预测 */

⑩ $yPredByLog \leftarrow Infer(clfLog, \mathbf{UP}[CodeFeatureCount:])$; /* 用 Code 推理标注 n_P 个正样本,n_N 个负样本 */

⑪ $(\mathbf{Pc}, \mathbf{Nc}) \leftarrow label(yPredByCode, n_P, n_N, \mathbf{UP})$; /* 用 Log 推理标注 n_P 个正样本,n_N 个负样本 */

⑫ $(\mathbf{Pl}, \mathbf{Nl}) \leftarrow label(yPredByLog, n_P, n_N, \mathbf{UP})$; /* 加入标注样本集,共 $2(n_P+n_N)$ 个 */

⑬ $\mathbf{XL} \leftarrow list(set(\mathbf{XL}) \cup set(\mathbf{Pc}) \cup set(\mathbf{Nc}) \cup set(\mathbf{Pl}) \cup set(\mathbf{Nl}))$;

⑭ $\mathbf{UP} \leftarrow list(set(\mathbf{UP}) - (set(\mathbf{XL}) \cup set(\mathbf{Pc}) \cup set(\mathbf{Nc}) \cup set(\mathbf{Pl}) \cup set(\mathbf{Nl})))$;

⑮ ***UP*** ← 从 ***XU*** 随机取 $2(n_P+n_N)$ 个并入 ***UP***;

⑯ ENDWHILE

⑰ RETURN $clfCode$, $clfLog$.

算法 2. 模型推理预测过程.

输入:$clfCode$ 是算法 1 生成的代码特征分类器、$clfLog$ 是算法 1 生成的 Log 特征分类器、***x*** 是数据点;

输出:1 预测为正(漏洞修复补丁),0 为负.

① FUNCTION: $vulInfer(\mathbf{x}, clfCode, clfLog)$

② $pred1 = pred2 = -1$;

③ $pred1 \leftarrow Infer(clfCode, \mathbf{x}[0:CodeFeatureCount])$;

④ $pred2 \leftarrow Infer(clfLog, \mathbf{x}[CodeFeatureCount:])$;

⑤ IF($pred1 == pred2$)

⑥ RETURN $pred1$;
⑦ ENDIF /* Code 和 Log 分类器预测一致 Code 预测为正、负的置信概率 */
⑧ $(pp1, pn1) \leftarrow InferProbability(clfCode, \boldsymbol{x}[0:CodeFeatureCount])$; /* Log 预测为正、负的置信概率 */
⑨ $(pp2, pn2) \leftarrow InferProbability(clfLog, \boldsymbol{x}[CodeFeatureCount:])$;
⑩ IF $(pp1+pp2>pn1+pn2)$
⑪ RETURN 1;
⑫ ENDIF /* 预测为正的概率大 */
⑬ IF $(pp1+pp2<pn1+pn2)$
⑭ RETURN 0;
⑮ ENDIF
/* 正、负置信度和相等,随机预测 */
⑯ RETURN 从集合{0,1}中随机取的值.

3 实验评估

在本节中,首先介绍数据集的情况;然后给出模型训练收敛性的实验测试评估,接着实验评估了模型的性能、跟代表性研究的对比;最后评估了安全漏洞补丁的报告情况分析.实验表明本文给出的漏洞修复补丁识别方法可行、效果明显,跟代表性研究对比其预测性能有明显提升.同时安全漏洞补丁的追踪情况现状分析表明,完全直接依托 NVD 和补丁的 Log 信息识别、管理,难以实现及时、全面地响应安全漏洞威胁;因此通过自动化识别等方法提升对漏洞修复补丁的识别准确率和推送效率很有必要.

3.1 度量标准

我们定义准确率、召回率和精确率公式为:

$$Accuracy=\frac{\#TruePositive+\#TrueNegative}{\#Positive+\#Negative}\times 100\%, \tag{6}$$

$$Recall=\frac{\#TruePositive}{\#TruePositive+\#FalseNegative}\times 100\%, \tag{7}$$

$$Precision=\frac{\#TruePositive}{\#TruePositive+\#FalsePositive}\times 100\%, \tag{8}$$

其中,$\#TruePositive$ 表示在测试时正数据点被正确识别为正的个数.$\#TrueNegative$ 表示负数据点被正确识别为负的个数.$\#FalseNegative$ 表示正数据点被错误识别为负的个数.$\#FalsePositive$ 表示负数据点被错误识别为正的个数.$\#Positive$ 表示正数据点的个数,等于 $\#TruePositive+\#FalseNegative$.$\#Negative$ 表示负数据点的个数,等于 $\#TrueNegative+\#FalsePositive$.

准确率、召回率和精确率这 3 个指标方便跟相关研究做对比.在工程实践中我们也关心误报率和漏报率这 2 个直观指标,前者影响补丁识别工作量,后者影响工具可靠性.因为漏报率是召回率的相反指标,所以我们只需定义误报率公式为

$$FPR=\frac{\#FalsePositive}{\#FalsePositive+\#TrueNegative}\times 100\%. \tag{9}$$

3.2 数据集

本文使用 2.1 节给出的方法收集和标注数据,收集的数据集整理为正数据、负数据和未标注数据.正数据是安全漏洞修复补丁,标注为 1;负数据是非安全漏洞修复补丁,标注为 0;未标注数据是分类未知的补丁,标记为 −1.表 1 给出了收集的 Linux 内核补丁数据集的构成与使用分配情况.覆盖了 2002—2020 年 5 月 23 日 Linux 内核补丁.

Table 1 Data Set Composition and Assignment

表 1 数据集构成与分配

数据构成	漏洞修复	Bug 修复	功能特征	未标注
训练	1 226	1 451	510	42 359
测试	407	483	169	0

3.3 模型训练收敛过程

实验参数设置:设置从无标注样本集选择标注的训练迭代次数(参数 $iter$)为 30,设置参数 $n_P=2$, $n_N=4$, $n_U=80$.模型的内部子分类器选型 SVM LinearSVC(只需完成正和负 2 个分类).正样本 1 633 个、负样本 2 613 个、未标注样本 42 359 个;标注样本的 3/4 用做训练数据,1/4 为测试数据.图 5 所示,当迭代训练到第 18 轮时模型可以趋于稳定收敛.

关于参数选取的说明:参数 n_P, n_N, n_U 的设置参考 Co-Training 原始文献[25]里的论述.Co-Training 模型对内部子模型的选择并没有特别的限制,本文考虑到识别漏洞修复补丁是二分类问题,因此使用 SVM LinearSVC 可以满足实验验证的需要;实际上选择或构建其他模型(如构建二分类的神经网络模型)代替这里的 SVM LinearSVC 作为参数传递给 Co-Training,充当内部子模型二分类器也是可行的.

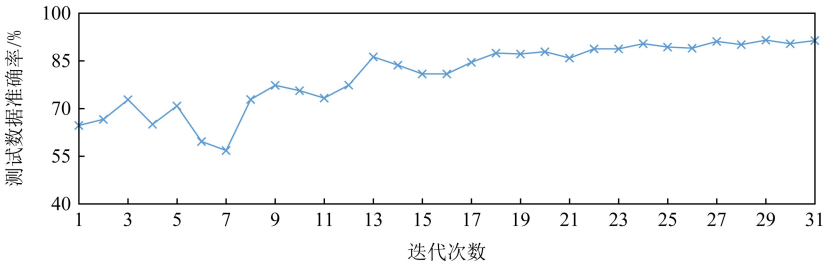

Fig. 5　Test accuracy versus number of iterations for a training experiment

图 5　一次模型训练过程实验期间测试准确率随迭代次数变化

3.4　模型的预测性能

本节通过实验测试模型的准确率、召回率、精确率和误报率,对模型的预测效果进行评估.跟代表性研究模型的性能指标对比表明,本文的方法模型能明显提高预测性能.

表 2 是实验测试的主要性能指标,其精确率、召回率和准确率指标表明本文建模方法对安全漏洞修复补丁识别效果显著.同时误报率 5.2%是可接受的.

Table 2　The Test Results of Model Performance

表 2　模型的性能指标实验结果　%

Precision	*Recall*	*Accuracy*	*FPR*
91.3	87.5	92.0	5.2

表 3 是跟主流模型的性能对比.文献[12]建模只使用了补丁的 Log message 信息;文献[13]同时使用了 Log message 信息和补丁的 Code 信息,但是对 Code 信息只做纯文本,因此特征抽取比较粗糙;我们的方法同时使用 Log message 和 Code 信息,并且将 Code 部分按照代码对待,抽取其标识符、结构、语言构造等程序信息,因此特征提取更充分.性能指标对比表明我们的方法可以明显提升模型的预测性能.

Table 3　Performance Comparison with State-of-the-Art

表 3　跟代表性主流模型的性能对比　%

模型	*Precision*	*Recall*	特征工程
Zhou[12]	70	71	Log message
Sabetta A[13]	80	43	Log message+Code 文本
本文	91.3	87.5	Log message+Code 语义

3.5　对 unlabled 补丁的自动检测

实验目的是检验模型从未标注补丁中发现漏洞修复补丁的能力.表 4 给出了自动检测报告的 13 个补丁的情况,其中前 11 个是漏洞修复补丁,后 2 个是被工具误报的纯 Bug 修复补丁.该实验表明本文方法可以有效检测漏洞修复补丁.

Table 4　Automatic Detection of Vulnerability Fix Patches

表 4　自动检测漏洞修复补丁

Commit Hash	漏洞类型	模块	补丁大小
682630f00a21	double free of buffer	rdma	11
15bfd21fbc5d	out of bound access	blkdev	12
3619dec5103d	out of bound access	dh	14
a3a374bf1889	wrong bound check	xfs	11
945d015ee0c3	use after free	af_packet	31
3d0641015bf7	double free	rdma	61
4579a1ba692a	derefence uninitialized	vector_kern	26
2026d35741f2	truncate the data type	compiler	9
0769b27739ee	double free	mwifiex	20
8e59c7f23410	null pointer dereferences	power	25
4f9de4df901f	memory leak(flood)	qed_ll2	27
fad2d4ef6366	'if test' bug fix	drbd_req	11
964d978433a4	'logic err' bug fix	cacheinfo	9

实验方法是从 unlabled 数据集中随机抽取 1 500 个交由训练的模型做检测,然后对检测结果报告为正(即漏洞修复补丁)的补丁做人工检查.众所周知,Bug 具有漏洞的特点并不表示很快可以找到该漏洞的利用方法,从而确定性证实或者证伪该漏洞.即该实验的严格验证比较困难.因此人工检查是按照补丁是否修复典型的漏洞特征进行界定.工具报告了 13 个安全漏洞修复补丁,经过手动检查 11 个被确认,2 个是误报(纯 Bug 修复补丁).

3.6　Linux 安全漏洞补丁报告概况

我们的工具自动收集了从 2002 年起到 2020 年 5 月 23 日止,收录于 NVD 数据库的 Linux 内核全部登记漏洞.这些漏洞状态的汇总统计如表 5 所示:

Table 5　Statistics of Linux Vulnerability NVD Status

表 5　Linux 漏洞 NVD 登记状态分析

#CVEs	#Patches	#Patched CVEs	#Fault CVEs	#Lost Commits	#Patches With CVE
4 096	1 633	1 497	51	55	380

登记信息的完整性:一共有 4 064 个被 NVD 数据库收录确认的漏洞,只有 380 个漏洞修复补丁的 Log message 给出了 CVE ID 信息,仅占内核确认漏洞的 9.35%,占 NVD 反向可追踪数的 23.27%;漏洞修复补丁在 NVD 的 CVE 信息中被记录追踪的只有 1633 个(关联 1497 个 CVE),即只有 40.2%的漏洞修复补丁在 NVD 数据库端有引用、只有 1 497 个 CVE 的登记信息是完整的.因此用户如果通过 Linux 社区补丁提交的 Log message 信息收集漏洞修复补丁,只能及时识别 9.35%.

登记信息的正确性:我们的工具分析发现,NVD 数据库汇总有 51 个内核 CVE 登记的补丁链接已经失效,涉及 Linux 内核的 55 个失效补丁,因此错误率为 51/1 497=3.4%.

因此无论从源码仓库端或 NVD 数据库端收集、识别安全漏洞修复补丁都存在全面性、有效提示性和及时性的不足,这表明完全依靠 NVD 和开源软件社区漏洞通告做安全维护是不可靠的.同时这也表明智能自动化的漏洞修复补丁识别研究具有重要价值.

3.7 讨 论

1) 局限性.本文实验表明,从补丁的统计量构建特征,将补丁区分为安全漏洞修复和非安全漏洞修复准确率很高.但如果进一步将安全漏洞修复补丁做细粒度分类(比如区分为缓冲区溢出、整数溢出、错误注入、访问控制错误等)则面临挑战.同样,如果某个补丁违背了实证软件工程的一般统计规律,本文的研究方法很难区分这类异常补丁,考虑到在规范的大型开源项目比如 Linux,Xen,OpenStack,实践中出现可能性很少,所以总体而言这种少量异常补丁的可能存在不影响本文方法的应用价值.如果一个社区的补丁管理严重不规范,比如功能混淆、缺乏一般的统计规律,对本文方法将是很大的挑战,这不仅对工具是挑战,对自然人手动识别也会带来困难.我们认为结合补丁的语法语义解析构建特征,有可能对这类补丁的识别带来帮助,这是我们下一步工作重点考虑的建模因素.

2) 模型的可扩展性.本文给出的识别漏洞修复补丁的建模方法是通用的,即可以直接扩展支持其他开源项目.虽然,因为一方面不同开源项目存在编程语言、功能代码粒度、项目功能、编程规范等差异,另一方面当前 AI 技术尚存在泛化能力瓶颈(即一般地,一个训练好的 AI 模型只能解决一个专门的问题,难以像人类智能做到触类旁通、举一反三),因此用 Linux 数据集训练的模型并不能直接应用于其他开源项目的检测;但是可以用同样的建模方法,必要时对特征工程的参数做适当调整,为其他开源项目训练预测模型.另外,如果想复用在一个项目上的训练经验,迁移学习是值得探讨的.

4 系统部署

在实际应用中,我们提供了 2 套部署场景.

场景 1:识别上游 Linux 内核最新合并的补丁是否是漏洞修复补丁.脚本程序周期性地将 Linux 官方分支拉取(Pull)到团队的内部内核仓库,脚本检测到更新则通过 git show 获取新补丁,然后传递给我们的模型识别,模型将识别结果邮件推送给安全工程师.

场景 2:内核开发团队补丁评审(review).工程师创建 PR(pull request)请求新补丁评审,机器人评审员(reviewer)对补丁进行安全识别,如果识别为安全漏洞修复补丁,则在评审中要求在 Log message 里增加安全提示性说明,并邮件通知安全工程师确认.

5 总 结

本文提出了一种漏洞修复补丁自动化识别的机器学习建模方法.帮助用户和安全工程师更及时、全面地识别安全漏洞修复补丁.实验结果表明,本文实现的漏洞补丁识别模型可以明显提高识别性能,取得 91.3%的精确率、87.53%的召回率,降低误报率,因此,本文的方法是可行且高效的.下一步,有必要对以下工作进行探讨:

1) 语法特征.为补丁构建特征工程时将补丁语法结构作为特征定义、提取的要素.这对于识别复合补丁(可能存在的一个补丁同时涉及漏洞修复、增加新功能、或普通 Bug 修复)是否涉及安全漏洞修复,以及进一步提高模型预测性能可能是有帮助的;

2) 跨开源项目的迁移学习.出于提升模型的训练效率、预测性能或泛化能力考虑,以及解决一些开源项目可标注数据稀缺问题,应用迁移学习的方法将一个项目的训练学习经验在另一个开源项目的训练中进行迁移复用,是下一步很值得探讨的工作.

作者贡献声明:周鹏负责论文观点的提炼与细化,完成论文撰写,负责试验设计、试验实施、专家意

见修改回复等;武延军负责研究方向指引、论文框架、论文评审与详细修改,参与试验设计、专家意见讨论评审等;赵琛负责论文指导、论文评审与修改,参与试验框架设计、对专家意见的改进方法进行改进,开发试验环境保障等.

参 考 文 献

[1] Li F, Paxson V. A large-scale empirical study of security patches [C] //Proc of the 2017 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2017: 2201-2215

[2] Zhao Shangru, Li Xuejun, Fang Yue, et al. A survey on automated exploit generation [J]. Journal of Computer Research and Development, 2019, 56(10): 2097-2111 (in Chinese)
(赵尚儒, 李学俊, 方越, 等. 安全漏洞自动利用综述[J]. 计算机研究与发展, 2019, 56(10): 2097-2111)

[3] Ghafoor I, Jattala I, Durrani S, et al. Analysis of OpenSSL heartbleed vulnerability for embedded systems [C] //Proc of the 17th IEEE Int Multi Topic Conf. Piscataway, NJ: IEEE, 2014: 314-319

[4] Nappa A, Johnson R, Bilge L, et al. The attack of the clones: A study of the impact of shared code on vulnerability patching [C] //Proc of the IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2015: 692-708

[5] Snyk Ltd. The state of open source security [OL]. 2017 [2020-05-20]. https://snyk.io/wp-content/uploads/The-State-of-Open-Source-2017.pdf

[6] The Linux Kernel Organization. Linux kernel [DB/OL]. 2020 [2020-05-20]. https://www.kernel.org/

[7] Liu Jian, Su Purui, Yang Min, et al. Software and cyber security—A survey [J]. Journal of Software, 2018, 29(1): 42-68 (in Chinese)
(刘剑, 苏璞睿, 杨珉, 等. 软件与网络安全研究综述[J]. 软件学报, 2018, 29(1): 42-68)

[8] Li Zan, Bian Pan, Shi Wenchang, et al. Approach of leveraging patches to discover unknown vulnerabilities [J]. Journal of Software, 2018, 29(5): 1199-1212 (in Chinese)
(李赞, 边攀, 石文昌, 等. 一种利用补丁的未知漏洞发现方法[J]. 软件学报, 2018, 29(5): 1199-1212)

[9] Zaman S, Adams B, Hassan A E. Security versus performance bugs: A case study on firefox [C] //Proc of the 8th Working Conf on Mining Software Repositories. New York: ACM, 2011: 93-102

[10] MITRE Corporation. CVE: Common vulnerabilities and exposures [DB/OL]. 2020 [2020-05-20]. https://cve.mitre.org/cve/request_id.html

[11] US National Institute of Standards and Technology. NVD: National vulnerability database [DB/OL]. 2020 [2020-05-20]. https://nvd.nist.gov/

[12] Zhou Yaqin, Sharma A. Automated identification of security issues from commit messages and bug reports [C] //Proc of the 11th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2017: 914-919

[13] Sabetta A, Bezzi M. A practical approach to the automatic classification of security-relevant commits [C] //Proc of the IEEE Int Conf on Software Maintenance and Evolution (ICSME). Piscataway, NJ: IEEE, 2018: 579-582

[14] Wang Xianda, Wang Shu, Sun Kun, et al. A machine learning approach to classify security patches into vulnerability types [C] //Proc of the IEEE Conf on Communications and Network Security (CNS). Piscataway, NJ: IEEE, 2020: 1-9

[15] Wu Qiushi, He Yang, McCamant S, et al. Precisely characterizing security impact in a flood of patches via symbolic rule comparison [C/OL] //Proc of the 27th Annual Network and Distributed System Security Symp. 2020 [2020-11-20]. https://www.ndss-symposium.org/wp-content/uploads/2020/02/24419-paper.pdf

[16] Soto M, Thung F, Wong C P, et al. A deeper look into bug fixes: Patterns, replacements, deletions, and additions [C] //Proc of the 13th IEEE/ACM Working Conf on Mining Software Repositories (MSR). Piscataway, NJ: IEEE, 2016: 512-515

[17] Tian Y, Lawall J, Lo D. Identifying Linux bug fixing patches [C] //Proc of the 34th Int Conf on Software Engineering (ICSE). Piscataway, NJ: IEEE, 2012: 386-396

[18] Nguyen A T, Hilton M, Codoban M, et al. API code recommendation using statistical learning from fine-grained changes [C] //Proc of the 24th ACM SIGSOFT Int Symp on Foundations of Software Engineering. New York: ACM, 2016: 511-522

[19] Polosukhin I, Skidanov A. Neural program search: Solving programming tasks from description and examples [J]. arXivpreprint, arXiv:1802.04335, 2018

[20] The Linux Kernel Organization. Linux kernel [DB/OL]. 2020 [2020-05-20]. https://github.com/torvalds/linux

[21] US National Institute of Standards and Technology. CPE: Common platform enumeration [DB/OL]. 2020 [2020-05-20]. https://nvd.nist.gov/products/cpe

[22] Free Software Foundation. Diff-utils unified format [OL]. 2020 [2020-05-20]. https://www.gnu.org/software/diffutils/manual/html_node/Detailed-Unified.html#Detailed-Unified

[23] Wallach H M. Topic modeling: Beyond bag-of-words [C] //Proc of the 23rd Int Conf on Machine Learning. New York: ACM, 2006: 977-984

[24] Li Xiaoli, Liu Bing. Learning to classify texts using positive and unlabeled data [C/OL] //Proc of the 18th Int Joint Conf on Artificial Intelligence. 2003 [2020-05-20]. https://www.ijcai.org/Proceedings/03/Papers/087.pdf

[25] Blum A, Mitchell T. Combining labeled and unlabeled data with co-training [C] //Proc of the 11th Annual Conf on Computational Learning Theory. New York: ACM, 1998: 92-100

[26] Scikit-learn Developers. LinearSVC [OL]. 2020 [2020-05-20]. https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC

Zhou Peng, born in 1984. PhD. His main research interests include intelligent software, operating system, virtualization, system security.

周 鹏, 1984 年生. 博士. 主要研究方向为智能软件、操作系统、虚拟化、系统安全.

Wu Yanjun, born in 1979. PhD, professor, PhD supervisor. His main research interests include operating system and system security.

武延军, 1979 年生. 博士, 研究员, 博士生导师. 主要研究方向为操作系统、系统安全.

Zhao Chen, born in 1967. PhD, professor, PhD supervisor. His main research interests include compiling, auto-testing, operating system, and networking software.

赵 琛, 1967 年生. 博士, 研究员, 博士生导师. 主要研究方向为编译、自动化测试、操作系统、网络软件.