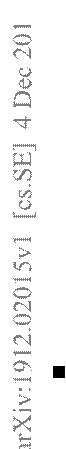
使用序列到序列

学习修复C

脆弱性

陈子民

皇家理工学院邮箱：zimin@kth.se

史蒂夫科姆鲁什

科罗拉多州立大学steveko@cs.colostate.edu

马丁蒙佩鲁斯

皇家理工学院马丁·蒙佩罗斯@csc.kth.se公司

***摘要***

**软件漏洞**是代码中的一个弱点，攻击者可以利用该弱点执行未经授权的操作。例如，一种常见的漏洞是缓冲区溢出，它允许攻击者覆盖缓冲区的边界并注入恶意代码。另一个例子是SQL注入，恶意SQL语句被插入到可执行查询中。网络犯罪每年给世界经济造成的损失高达数千亿美元[1]。

每个月，数千个这样的漏洞被报告到公共漏洞和暴露（CVE）数据库。它们中的每一个都被分配了一个唯一的标识符和类别。例如，CVE-2019-9208标识的条目是Wireshark中的一个漏洞，该漏洞会导致空指针异常，并将其归类为“空指针取消引用”。仅在2019年10月，就有1737个漏洞报告给国家漏洞数据库（NVD），这是CVE的主要数据库。在开发人员编写补丁之前，每个漏洞都代表一种威胁。

1

|  |
| --- |
| **CVE-2009-4004补丁**unsigned bank\_num=mcg\_cap&0xff，bank；  r=—埃因瓦尔；  -如果（！银行（数字）  +如果（！银行编号| |银行编号>=  KVM\u MAX\u MCE\u BANKS）输出；  if（mcg\u cap&（支持KVM\u MCE\u cap）  |0xff | 0xff0000））转到输出；  **CVE-2016-8658补丁**  （u8\*）设置（&S）->信标头[即偏移量]，  设置->灯塔头灯ie\U偏移量，  无线局域网（EID）；  -如果（！ssid（ie）  +如果（！ssid| | ssid|u ie->len>  IEEE80211\u MAX\u SSID\u LEN）返回-EINVAL；  memcpy（ssid）\_le.SSID公司，ssid\u ie->数据，ssid\u ie->len）； |

清单1:CVE-2009-4004和CVE2016-8658的补丁，开发人员在这两种情况下都添加了边界检查。

有许多可用的修补程序可以修复给定类型的漏洞。让我们考虑一下CVE类别“内存缓冲区边界内操作的不当限制”：CVE-2009-4004和CVE-2016-8658都是Linux内核中的漏洞，分别于2009年11月和2016年10月在NVD上发布。清单1显示了相应的修补程序。这两个漏洞都是通过在内存边界上添加检查来修复的。

考虑到数以百万计的提交在开源项目上是公开的，我们假设我们有足够的数据来构建一个系统，从过去的提交中学习，并使用它来预测新漏洞的修补程序。这是本文的贡献，我们提出了一种新颖而新颖的方法来学习如何自动生成安全漏洞补丁。

我们使用一种强大的机器学习技术，称为序列到序列学习（seq2seq）。Seq2seq学习令牌序列到另一个序列之间的映射。它被大量用于不同人类语言之间的自动翻译（比如Google translate），并推动了那里最先进的性能。对于机器翻译，seq2seq模型的训练数据由一组句子组成，例如英语句子和相应的法语翻译。为了训练seq2seq进行软件漏洞修复，我们需要这样一个语料库，但需要源代码。

在这项工作中，我们收集了一个数据集，其中包括2017年和2018年GitHub上2100万个bug修复提交。收集的数据集用于训练seq2seq模型。为了评估训练模型的有效性，我们从四个著名的开源项目中收集了真实世界的CVE漏洞：Linux内核、OpenSSL、systemd和Wireshark。我们的结果显示了使用seq2seq修复漏洞的前景，对于不同大小的漏洞函数，其修复精度分别为26.7%、13.7%和9.2%。

值得注意的是，我们的方法是完全通用的，而不是专注于一种特定的漏洞，如缓冲区溢出[2]。与相关工作相比，我们在CVE数据库中发现的大量真实漏洞上评估了我们的技术，而不是在合成代码[3]或手动收集的基准测试[4]上。据我们所知，我们的实验涉及有史以来最大的人工智能代码训练数据集。

总之，我们的贡献是：

•我们从

GitHub，我们共享最大的代码更改机器学习数据集（2100万个bug修复提交，91万个与C相关的提交）。

•我们成功地使用字节对编码来训练seq2seq模型，以解决源代码中词汇表的大小问题，在我们的数据集中达到600万个不同的单词。

•我们报告了使用机器学习修复现实世界漏洞的原始结果，使用CVE标识符的14/630漏洞可以以完全自动化的方式修复。

# 序列间学习

Sequence-to-Sequence（seq2seq）学习是一种现代机器学习框架，用于学习两个序列（通常是单词）之间的映射[5]。它广泛应用于自动翻译、文本摘要等与自然语言相关的任务中。seq2seq模型由两部分组成：编码器和解码器。编码器将输入序列=（x0，x1，…，xn）映射到中间连续表示=（h0，h1，…，hn）。然后，给定，解码器生成输出序列=（y0，y1，…，ym）。请注意，输入和输出序列的大小和可以不同。在训练数据集上优化seq2seq模型，使（Y | X）的条件概率最大化，等价于：（Y | X）=p（y0，y1，…，ym | x0，x1，…，xn）*十小时小时是的n米pp*

*米*

=Y（yi | H，y0，y1，…，yi−1）*p*

*我*=0

先前的工作已经表明，源代码和人类语言一样自然[6]，自然语言处理中使用的技术已经被证明可以很好地处理源代码，包括seq2seq学习[7]。在我们的工作中，我们使用了一个名为“transformer”的seq2seq模型[8]。transformer模型是seq2seq学习的最新架构。

# 源代码中的稀有词

在源代码中使用seq2seq模型的主要挑战之一是它几乎不能处理非常罕见的单词[9]。问题是稀有词，例如原始文字或特定域标识符，在收集的训练数据中太不常见甚至不存在，因此在解码时不能使用。实际上，在源代码中，罕见的变量和函数名比人类语言更为常见。处理罕见词问题的一个基本技巧是增加词汇量，比如说从10k增加到50k，但这只是一个解决方法：总会有一些罕见词在训练时没有足够的数据可用。

然而，一个罕见的词可能有频繁的子词。例如，单词underworld可能是一个罕见的单词，但子单词under和world是常见的单词。所以如果我们能用频繁的子词来表示我们的词汇，那么我们就可以用它生成任何单词。字节对编码（BPE）是学习最频繁子词的最先进技术[10]。BPE从基本字符开始作为词汇表（例如，拉丁字母表中的字母）。在每个时间步，最频繁的子词对被组合成一个新的子词，并被添加到词汇表中。它一直持续到我们达到预定义的词汇量。清单2展示了一个在C函数上应用BPE的示例。destroyKeyValuePair和freeValue等变量分别被视为4个子词和2个子词（destroy Key-Value Pair和free-Value），这是在其他代码段中可以找到的更常见的词。BPE已经成功地应用于机器翻译[10]和源代码建模[11]。在本文中，我们是第一个报告使用seq2seq和BPE来生成补丁的。

|  |
| --- |
| **C代码**  无效的destroyKeyValuePair（keyValuePair kvPair）{  kvPair->freeValue（kvPair->value）；kvPair->freeKey（kvPair->key）；free（kvPair）；  }  **应用BPE后**  \_void\u destroy Key-Value对（\u Key  值对（kv对）  \_自由值（kv对）；  \_千伏线对->自由密钥（\u千伏线对->  \_无钥匙（kv对） |

清单2：在C函数上应用所学BPE的示例。“”（U+2581）表示新词的开头。BPE学习了一些有用的子词，比如Key、Value和Pair。

# 数据收集和过滤

训练数据集。为了训练seq2seq模型，我们需要大量的bug和固定的源代码。我们通过挖掘GitHub开发平台来创建这样一个语料库。我们使用GH Archive[12]下载2017-01-01和2018-12-31之间发生的所有GitHub事件。这些事件可以由Github问题创建、打开的pull请求和其他开发活动触发。在我们的例子中，我们主要关注推送事件，当提交被推送到存储库分支时会触发推送事件。为了只收集bug修复提交，我们采用了基于关键字的启发式方法[13]：如果提交消息包含关键字（fix或solve或repair）和（bug或issue或problem或error或fault或漏洞），我们将其视为bug修复提交，并将其添加到我们的语料库中。我们总共分析了7.3亿个提交，并选择了2100万个确定为bug修复提交的提交。

在我们的实验中，我们把C代码作为自动修复的目标编程语言。因此，我们根据文件扩展名进一步过滤bug修复提交。我们删除了没有修复任何以“.c”结尾的文件的提交，这就留下了910000个错误的c提交。然后，对于每个提交，我们提取在提交中更改的函数对。我们学习的是函数级的变化，而不是文件级的变化，因为seq2seq需要长时间的输入和输出[14]。为了识别函数级更改，我们使用GNU编译器预处理器删除所有注释，并且只提取更改的函数。然后，我们使用Clang来解析和标记函数源代码。

最后，我们得到了1806879个函数级的变化，在删除重复项后减少到642399个。函数的大小各不相同，我们观察到有些C函数仍然太大，无法用seq2seq学习。因此，我们进一步将训练数据集划分为，和，其中更改前后的函数长度分别限制为200、100和50个令牌。在和中，我们分别有299976、146051和49340个功能级别更改。变更前的函数代码作为seq2seq模型的输入，变更后的函数作为训练的地面真值输出。*d*200*d*100*d*50*d*200*d*100*d*50

测试数据集。我们还收集了一个数据集来测试seq2seq修复实际漏洞的能力。我们使用Data7[15]从四个著名的项目中收集具有CVE标识符的已知漏洞：Linux kernel、OpenSSL、systemd和Wireshark。测试数据集中的每个示例都包含一个CVE编号和一个修复该漏洞的提交列表。接下来，我们从这些漏洞中提取函数级别的更改，并将其称为易受攻击的函数。我们认为一个漏洞是完全固定的，如果它的所有脆弱的功能是固定的。如果某个漏洞的至少一个脆弱功能已修复，则该漏洞部分已修复。测试集和仅包含易受攻击的函数，其中更改前后的令牌长度分别限制为200、100和50个令牌。例如，我们有1615个易受攻击的函数，代表630个漏洞。例如，我们有725个易受攻击的函数，代表288个漏洞。例如，我们有120个易受攻击的函数，分布在85个易受攻击的函数上。*t型*200*t型*100*t型*50 *t型*200*t型*100*t型*50

# 实验装置

训练数据集，随机分为训练数据和验证数据，98%为训练数据，2%为验证数据。通过超参数空间的网格搜索，选择验证精度最高的模型。我们在我们的测试数据集上评估结果模型，以及。我们在三个数据集上训练三个基线seq2seq模型，并将词汇表设置为前50k个最常见的标记。这些基线代表了最先进的seq2seq模型，没有特别注意解决这个罕见的词问题。*d*200*d*100*d*50 *t型*200*t型*100 *t型*50*d*200*d*100 *d*50

接下来，我们将探讨使用BPE处理稀有令牌的seq2seq模型。对于BPE配置，我们将子词词汇表的大小设置为1000、5000或10000，即词汇表是源代码标识符中最常见的1000、5000或10000个子词。在获得了最佳BPE子词汇表之后，我们训练seq2seq模型我们的训练数据集，并且。因此，除了我们的基线之外，我们还有九种不同的设置：三种不同令牌长度限制的叉积和BPE定义的三种不同词汇表。表1总结了总共12种不同的seq2seq模型。seq2seq模型在训练数据集和*d*200*d*100 *d*50

BPE配置：例如−d200指的是seq2seq模型，它的词汇表设置为前1k个最常见的子词。*业务流程*1000*d*200

在相应的测试数据集上评估所有12种不同设置的最佳模型：

*基线*−d200在上评估，−d100在上评估，等等。我们使用beam search预测易受攻击函数的修复，这意味着seq2seq模型生成每个易受攻击函数前50个最有可能的预测。当50个预测中的一个与地面真值人类修复相匹配时，脆弱函数被认为是固定的，正如前面的工作[7]中所做的那样，*t型*200*业务流程*1000*t型*100

[16].

我们使用OpenNMT tf[17]来训练transformer模型，使用SentencePiece[18]来学习训练数据上的BPE。

# 实验结果

经过训练的seq2seq模型能否为现实世界的漏洞生成补丁？主要结果见表1。第一列根据其BPE配置给出seq2seq模型的名称。第二列分别显示对、和的预测精度。第三列显示部分修复的漏洞数。第四列表示完全修复的漏洞数。*t型*50*t型*100 *t型*200

我们首先关注系统正确修补的具有漏洞的C函数的数量。总的来说，我们的模型能够修复现实世界中的漏洞，对于小于50个令牌的小型C函数中的漏洞，修复率高达32/120（26%）。seq2seq模型的性能随着输入和输出长度的增加而降低：模型上的值（四行的第一组）高于较大函数的值：更容易修复较短C函数中的漏洞。*d*50

回想一下，基线模型不使用BPE，表1表明它们的精确度接近于0。这表明具有固定词汇表的标准seq2seq模型不是处理罕见令牌问题的选项。为了进一步分析这一现象，我们分析了−d200生成的80750个预测（1615个脆弱函数，每个函数×50个预测）。我们发现80047/80346（99%）预测包含词汇表外标记，这进一步证实了源代码中稀有词的普遍性。现在，我们的结果显示bytepair编码（BPE）是解决这个问题的一个强大的解决方案：对于小函数（），固定C函数的数量从5跳到32，对于大函数（），从0跳到148。我们的数据表明，对于大型函数，使用大型子词词汇量仍有一定的改进空间（20000的子词词汇量可能会提高准确性）。*基线t型*200 *d*50*d*200

回想一下，单个漏洞通常会同时在多个函数中修复。表1的第三列和第四列关注的是固定漏洞的数量，而不是固定漏洞函数的数量。我们的seq2seq模型能够部分修复小功能中多达22/85的漏洞。完全修复漏洞要困难得多，因为必须修复漏洞中的所有函数：例如在CVE2011-1771中，修复超过95个易受攻击的函数。尽管有这个强大的需求，我们的方法仍然能够完全修复中的3/85漏洞、中的10/288漏洞和中的14/630漏洞。据我们所知，这是seq2seq learning关于修复一般漏洞的第一个结果报告。*t型*50*t型*100*t型*200

seq2seq模型的结果是基于地面真值人类定位计算的。在生产中，这种漏洞修复系统将在没有地面真相修复的情况下使用。基于可疑函数，我们将使用额外的检查来过滤seq2seq的输出：例如编译（删除不可编译的代码）和测试执行（删除导致测试失败的补丁）。以前的工作表明，使用这些自动过滤技术可以正确过滤出seq2seq生成的多达97%的补丁[7]。案例研究

|  |
| --- |
| int cifs\u close（结构索引节点\*索引节点，结构文件\*文件）  {  -   cifsFileInfo\u put（文件->私有\u数据）；  -   文件->私有数据=空；  +if（文件->私有数据！=空）{  +cifsFileInfo\u put（文件->私有\u数据）；  +文件->私有数据=空；  + }  /\*从->释放操作返回的代码总是被忽略\*/  返回0； |

清单3:CVE-2011-1771被seq2seq成功修复。

现在让我们来讨论一些有趣的案例。CVE2011-1771是一个来自Linux内核的漏洞，NVD对此漏洞的描述是：Linux kernel 2.6.39之前版本中fs/cifs/file.c中的cifs close函数允许本地用户在尝试打开cifs文件系统上的文件时通过设置O DIRECT标志来造成拒绝服务（空指针取消引用和错误）或可能产生其他未指明的影响。此漏洞的修补程序如清单3所示。修复包括为变量私有数据添加空检查。我们的seq2seq方法能够生成这个精确的补丁（12个模型中有4个可以这样做：-*业务流程*1000

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | 模型 | | 固定函数 | 脆弱性 | | | 部分固定 | 完全固定 | | *基线*  *业务流程*1000  *业务流程*5000  *业务流程*10000 | −d50级  −d50级  −d50级  −d50级 | 5/120 (4.2%) | 3/85 (3.5%) | 1/85 (1.2%) | | 26/120 (21.7%) | 17/85 (20%) | 3/85 (3.5%) | | 32/120 (26.7%) | 22/85 (25.9%) | 3/85 (3.5%) | | 28/120 (23.3%) | 18/85 (21.1%) | 3/85 (3.5%) | | *基线*  *业务流程*1000  *业务流程*5000  *业务流程*10000 | −d100型  −d100型  −d100型  −d100型 | 2/725 (0.3%) | 2/288 (0.7%) | 0/288 (0%) | | 68/725 (9.4%) | 40/288 (13.9%) | 6/288 (2.1%) | | 97/725 (13.4%) | 47/288 (16.3%) | 5/288 (1.7%) | | 99/725 (13.7%) | 45/288 (15.6%) | 10/288 (3.5%) | | *基线*−200美元  *业务流程*1000 − 200*d*  *业务流程*5000 − 200*d*  *业务流程*10000 − 200*d* | | 0/1615 (0%) | 0/630 (0%) | 0/630 (0%) | | 109/1615 (6.7%) | 45/630 (7.1%) | 9/630 (1.4%) | | 131/1615 (8.1%) | 52/630 (8.3%) | 9/630 (1.4%) | | 148/1615 (9.2%) | 55/630 (8.7%) | 14/630 (2.2%) |   表1：我们训练的seq2seq模型在测试数据集上的性能。第一列给出了seq2seq模型的类型。第二列分别显示、和的精度。第三列显示部分修复的漏洞数。第四列代表*t型*50*t型*100 *t型*200 |

完全修复的漏洞数。

*d*50, *业务流程*10000 −d级100, *业务流程*1000 −d级100 和*业务流程*1000 −d级200).

|  |
| --- |
| 静态int omninet\u open（struct tty\u struct\*tty，struct usb\u serial\u port\*port）  {  -   struct usb\u serial\*serial=端口->串行；  -   结构usb\串口\*wport；  -  -   wport=串行->端口[1]；  -   tty\u port\u tty\u set（&wport->port，tty）；  -   返回usb\串行\通用\打开（tty，端口）；  } |

清单4:CVE-2017-8925被seq2seq成功修复。

CVE-2017-8925是Linux内核的另一个漏洞，NVD的描述是：Linux内核4.10.4之前版本的drivers/usb/serial/omninet.c中的omninet open函数允许本地用户利用引用计数错误处理来造成拒绝服务（tty耗尽）。它被归类为“不适当的资源关闭或释放”。清单4所示的人工修复删除了不正确处理变量“port”和“tty”的语句。我们的所有seq2seq模型在和上都可以生成这个精确的补丁。通过适当的培训数据，我们生成漏洞修复的seq2seq方法能够预测与人类开发人员相同的补丁。*d*100 *d*200

# 结论

软件漏洞很常见，会造成很大的损害。在本文中，我们朝着自动修复安全漏洞迈出了一步。我们设计、实现并评估了一个新系统，该系统基于从软件存储库中对过去提交的序列到序列学习。我们从GitHub中挖掘了两年的提交历史，并使用字节对编码技术解决了源代码中的罕见字问题。我们最初的结果表明，现实世界中易受攻击的C函数可以以完全自动化的、数据驱动的方式修复。未来的工作需要提高自动漏洞工具在修复一般漏洞方面的性能，并探索将此类技术集成到软件开发过程中。

# 工具书类

[1] N.Loss，“估计网络犯罪的全球成本”，McAfee，战略与国际研究中心，2014年。

[2] F.Gao、L.Wang和X.Li，“Bovinspector:缓冲区溢出漏洞的自动检查和修复”，2016年第31届IEEE/ACM国际自动化软件工程会议（ASE），IEEE，2016，第786–791页。

[3] J.Harer，O.Ozdemir，T.Lazovich，C.Reale，R.Russell，L.Kim，et al.，“学习用生成性对抗网络修复软件漏洞”，神经信息处理系统进展，2018年，第7933–7943页。

[4] S.Ma、F.Thung、D.Lo、C.Sun和

R.H.Deng，“Vurle:通过学习示例自动检测和修复漏洞”，欧洲计算机安全研究研讨会，Springer，2017年，第229-246页。

[5] I.Sutskever、O.Vinyals和Q.Le，“神经网络序列间学习”，NIPS进展，2014年。

[6] A.Hindle、E.T.Barr、Z.Su、M.Gabel和P.Devanbu，“论软件的自然性”，第34届国际软件工程会议（ICSE），IEEE，2012，第837-847页。

[7] Z.Chen、S.J.Kommrusch、M.Tufano、L.-N.Pouchet、D.Poshyvanyk和M.Monperrus，“定序器：端到端程序修复的序列到序列学习”，IEEE软件工程学报，2019年。

[8] A.Vaswani、N.Shazeer、N.Parmar、J.Uszkoreit、L.Jones、A.N.Gomez、Ł。

Kaiser和I.Polosukhin，“注意力是你所需要的一切”，神经信息处理系统进展，2017年，第5998-6008页。

[9] V.J.Hellendorn和P.Devanbu，“深度神经网络是建模源代码的最佳选择吗？“2017年第11届软件工程基础联席会议记录，ACM，2017，第763-773页。

[10] R.Sennrich、B.Haddow和A.Birch，“具有子词单位的稀有词的神经机器翻译”，ArXiv预印本附件十四：1508.07909, 2015.

[11] R.-M.Karampatsis和C.Sutton，“也许深度神经网络是建模源代码的最佳选择”，ArXiv预印本附件十四：1903.05734, 2019.

[12] , http://www.gharchive.org/，访问日期：2019-11-27。*Gh档案*

[13] M.Martinez和M.Monperrus，“自动程序修复搜索空间推理的挖掘软件修复模型”，经验

*软件工程*，第20卷，第1期，第176-205页，2015年。

             内政部：10.1007/s10664-013-9282-8。

             [在线]。可用：

https://hal.inria.fr/hal-00903808/file/mining-software-repair-models.pd

[14] K.Cho、B.Van Merrie¨nboer、D.Bahdanau和Y.Bengio，“关于神经机器翻译的特性：编码器方法”，ArXiv预印本附件十四：1409.1259, 2014.

[15] M.Jimenez、M.Papadakis和Y.L.Traon，“利用vuldata7实现安全漏洞的连续分析”，载于2018年第18届IEEE源代码分析和操纵欺诈国际工作会议论文集，西班牙马德里，2018年9月23日至24日。

[16] M.Tufano、C.Watson、G.Bavota、M.Di Penta、M.White和D.Poshyvanyk，“通过神经机器翻译在野外学习缺陷修复补丁的实证研究”，ASE，2018年，第832-837页。

〔17〕G. Klein、Y. Kim、Y. Deng、J.SENELART和A. M. Rush，“OpenNM:神经机器翻译的开源工具包”，ARXIV-E-Primes，EPrims: 1701.02810。

[18] T.Kudo和J.Richardson，“Sentencepiece:用于神经文本处理的简单且独立于语言的子词标记器和去标记器”，ArXiv预印本附件十四：1808.06226, 2018.