2019年IEEE安全与隐私研讨会

使用安全属性生成漏洞补丁

黄震、李大卫、谭刚、特伦特·杰格∗††∗∗

∗宾夕法尼亚州立大学多伦多大学黄震@psu.edu公司, lie@eecg.toronto.edu, 网址：gtan@cse.psu.edu, 邮箱：tjaeger@cse.psu.edu†

# 摘要

安全漏洞是存在的最关键的软件缺陷之一。一旦确定，程序员的目标是尽快生成可防止该漏洞的修补程序，从而需要自动程序修复（APR）方法来自动生成修补程序。不幸的是，大多数当前的APR方法都有不足之处，因为它们近似于使用示例来防止该漏洞所需的属性。近似计算会导致修补程序无法全面修复漏洞，甚至可能引入新的bug。相反，我们提出了基于属性的APR，它使用人类指定的、独立于程序的和特定于漏洞的安全属性来派生安全漏洞的源代码补丁。与通过观察测试用例的执行来近似的属性不同，这样的安全属性是精确和完整的。主要的挑战在于将这些安全属性映射到可以实例化到现有程序的源代码补丁中。

为了应对这些挑战，我们提出了Senx，它在给定一组安全属性和触发漏洞的单个输入的情况下，检测漏洞输入所违反的安全属性，并生成相应的补丁来强制执行安全属性，从而消除漏洞。Senx用基于属性的APR解决了几个难题：它确定了为了检查安全属性而必须计算的程序表达式和变量，并确定了可以计算它们的程序范围，如果调用现有的程序代码会产生不必要的副作用，它会生成新的代码来有选择地计算所需的值，并且它使用一种新的访问范围分析技术来避免在循环中放置补丁，从而导致性能开销。我们的评估显示，Senx生成的补丁成功修复了11个应用程序中42个真实世界漏洞中的32个，包括用于操作图形/媒体文件的各种工具或库、编程语言解释器、关系数据库引擎、用于创建和管理二进制程序的编程工具集合，以及基本文件、shell和文本操作工具的集合。

# 一、简介

及时修复安全漏洞对于保护用户免受安全危害和防止供应商失去用户信心至关重要。最近的一项研究表明，创建软件补丁通常是修复安全漏洞的瓶颈[19]。结果，整个

|  |
| --- |
| 这项工作主要是在第一作者是多伦多大学电气与计算机工程系的博士候选人时完成的。  ©2019，黄震。获得IEEE许可。  内政部10.1109/SP.2019.00071 |

研究人员研究了自动程序修复（APR）[16]、[20]、[26]、[27]、[30]、[34]、[39]、[40]、[54]、[55]，它接受程序和漏洞，并自动提供修补程序来修复漏洞。

与人类开发人员生成的修补程序一样，APR工具生成的修补程序旨在修复缺陷或漏洞。大多数现有工具依赖于一组正/负示例输入来找到一个补丁，使程序在这些示例上正确运行。它们检查修补程序是否满足正示例输入，但会导致负示例输入错误[30]、[34]、[36]。然而，通常很难获得一组完整的示例输入，并且修补程序可能无法在其他输入上正确运行，或者修补程序打算修复的漏洞在给定其他输入的情况下仍然可以被利用。因此，它们可能会生成无法修复漏洞的修补程序，或者更糟的是，会引入新的bug。我们称这种传统方法为“基于实例”。

在本文中，我们提倡一种不同的方法，我们称之为“基于属性的”APR。基于属性的APR依赖于独立于程序、特定于漏洞、特定于人的安全属性。这种安全属性的一个例子可能是，程序不应访问超出缓冲区结尾的部分，以排除缓冲区溢出漏洞。这种基于属性的方法的优点是，一小部分安全属性可以指定一次并在大量程序上使用，而无需指定每个程序的任何特定内容，也无需收集一组全面的测试用例。此外，这些性质本身就是精确和完整的。与基于示例的APR相比，基于属性的APR生成适用于所有可能输入的修补程序；这对于安全性尤其重要，这要求我们不要为攻击者留下漏洞。

基于属性的APR不同于以前全面实施安全属性的工具。相反，APR针对特定漏洞强制执行每个属性，因此可以利用漏洞的上下文生成高效的自定义补丁。例如，SoftBound[32]会插入一个程序，在所有缓冲区上强制执行内存安全，这通常会导致高性能开销。相反，基于属性的APR将特定漏洞（例如缓冲区溢出漏洞）作为输入，并生成针对该漏洞的修补程序。

基于属性的APR的挑战归结为开发一种方法，以确保执行适当的安全属性。不幸的是，基于属性的APR在文献[24]中受到的关注有限，并且存在一些限制其适用性的突出挑战。

首先，由于强制执行的安全属性是特定于漏洞的，APR工具必须为给定的漏洞确定要强制执行的正确属性。在这项工作中，我们假设APR工具的输入只是触发漏洞的输入，可能会导致程序崩溃。因此，APR工具还必须正确识别漏洞和相应的安全属性，以便执行。

其次，我们的目标是生成可供开发人员采用的源代码补丁。因为安全属性是泛型的和独立于程序的，所以必须将它们映射到程序中的变量以生成源代码补丁。但是，有时并非所有与安全属性对应的程序构造都可以在同一程序范围内使用。例如，缓冲区的大小可以存储在仅在分配缓冲区的函数中可用的变量中，而不在访问缓冲区的函数中可用。这要求程序分析能够为这些安全属性生成等效表达式，并在程序中的某个点选择范围内的相交集。

第三，安全属性中的某些术语可能必须映射到不仅涉及程序变量和常量，而且还涉及函数调用的表达式上。例如，程序可能总是通过调用函数来计算对象的大小，因为大小是动态的。因为函数调用可能有副作用，所以补丁必须小心不要调用任何有副作用的函数。因此，APR工具需要检查函数是否有副作用，甚至可能需要在程序中生成新函数来计算所需的值而不引入不需要的副作用。

最后，许多漏洞取决于循环的迭代次数。天真的方法只需检查每个循环迭代的安全属性，从而导致性能开销。为了减少生成的修补程序对性能的影响，APR工具应该了解循环并生成一个修补程序，一旦超出循环就检查安全属性，以避免性能开销。

在本文中，我们提出了Senx，它解决了上述挑战，即使用特定于漏洞的安全属性自动生成安全漏洞的源代码补丁。虽然Senx在理论上可以为指定安全属性的漏洞生成补丁，但是我们演示了Senx对于三类重要的漏洞：缓冲区溢出、错误转换和整数溢出。我们发现Senx能够为超过76%的漏洞生成正确的补丁。Senx未能为其余部分生成修补程序的主要原因是，它无法在程序源代码中找到评估安全属性所需的所有变量都在作用域中的位置，这将需要更改函数原型以允许这些变量跨越这些作用域。

本文的主要贡献如下：

•我们描述了如何在Senx中指定安全属性，并演示了缓冲区溢出、错误转换和整数溢出漏洞的三个示例安全属性。

1               字符\*r e v（常量字符\*输入，字符\*输出）{

2               *//反转*

3               *//inp是输入*4//输出是一个输出缓冲区

5                           i f（i n p！=空）{

6                           int i，l e n=s t r l e n（i n p）；

7                           *//未能检查i f（len+1<=*

*大小\u of \u out）*

8                           对于（i=0；i<l e n；i++）9 o u t[i]=i n p[l e n−i]；

10                          o u t[i]=&apos;\0&apos;；

11                          退货；

12                          }其他

13                          return“###”；

14                          }

15

16              void main（int argc，char argv[]）{\*

17              int s i z e=a t o i（argv[1]）+1；

18              char o u t=（char）ma ll oc（s i z e）；19//补丁：if（s t r l e n（argv[2]）+1>s i z e）。\*\*

20              p r i n t f（“%s\n”，r e v（argv[2]，o u t））；

21              }

清单1：反转输入字符串的程序。它在函数rev中包含缓冲区溢出。

•我们描述了Senx的设计，这是一个基于属性的自动补丁生成系统，使用了新的程序分析技术：表达式翻译、循环克隆和访问范围分析。

•我们在KLEE符号执行引擎的基础上设计了Senx原型，并在11个流行应用程序的42个漏洞的语料库中对其进行了评估，包括PHP解释器、sqlite数据库引擎、用于创建和管理二进制程序的binutils实用程序以及用于操作图形/媒体文件的各种工具或库。Senx在其中32种情况下生成正确的补丁程序，其余情况下由于无法确定其他情况下的语义正确性而中止。评估表明，生成修补程序需要所有三种技术，而未能找到放置修补程序的公共功能范围是最常见的失败原因。

本文的结构如下。我们在第二节激励我们的工作。在第三节中，我们定义并描述了问题Senx地址。第四节和第五节分别描述了Senx的设计和实现。我们特别描述了Senx如何应对第四节D中的第二个挑战，第四节C中的第三个和第四个挑战。我们在第七节给出了评估结果，并在第八节讨论了相关工作。最后，我们在第九节得出结论。

# 二。动机

我们将讨论Senx在本节中要解决的最先进的自动补丁生成工具的局限性。我们使用清单1中的程序作为目标程序，它是从一个真实的缓冲区溢出漏洞CVE中采用的-

2012-0947在流行的媒体流处理库中[42]。

此程序反转输入字符串。它从命令行获取两个输入，一个字符串和一个指定

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | argv[1] | argv[2] | 输出 | 预期产量 |
| 第 | 1 | A | A | A |
| 第 | 2 | AB型 | 文学士 | 文学士 |
| 不 | 1 | 基础知识 | 中国男子篮球职业联赛 | ### |
| 不 | 2 | 基础知识 | 中国男子篮球职业联赛 | ### |

表一：清单1中程序的测试输入和输出。‘P’型测试输入为正测试输入，而‘N’型测试输入为负测试输入。

字符串的长度，并输出反转的字符串。如果发生错误，程序将输出“###”。为此，它根据传递给它的值动态地分配一个临时缓冲区，并将输入复制到缓冲区中。与真正的漏洞一样，输出缓冲区的分配和输入字符串的处理由两个不同的函数实现。此示例是处理音频/视频流的典型程序。

当输入整数的指定长度小于输入字符串的实际长度时，会发生缓冲区溢出。可以通过添加一个检查来修复缓冲区溢出，该检查强制字符串的实际长度小于要复制到的缓冲区的分配大小。因为缓冲区大小只在main中已知，所以应该在第19行添加检查，并将大小与strlen（argv[2]）进行比较。虽然人类开发人员可以很容易地将此类检查添加到修补漏洞中—实际上，在人类生成的漏洞修补程序中可以找到此类检查，但此代码示例基于[42]—这对当前的APR工具提出了挑战。我们将在下面更详细地描述这些挑战。

基于实例的方法。许多APR工具使用示例输入作为修复漏洞的基础[29]、[30]、[34]、[55]。例如，SemFix和Angelix收集路径约束以生成修复[30]、[34]。

这种方法导致两个问题。首先，生成的约束通常只捕获基于测试用例中使用的具体值的约束，而不捕获程序变量之间关系的约束。为了说明这一点，表I列出了使用这些工具所需的典型测试输入，我们认为这些工具可能会在我们的示例中使用这些输入

清单1。

考虑到这些测试用例，SemFix和Angelix会发现argv[1]值为1或2与阴性测试用例无关，因为它们在阳性和阴性测试用例中都具有这些值。因此，它将错误地推断strlen（argv[2]）<3需要添加到代码中才能正确。生成不正确的修补程序是因为测试用例套件没有包含strlen（argv[2]）>2的阳性测试用例。这说明了基于示例的系统的缺点，因为它们很容易成为测试套件中丢失案例的牺牲品，而这些案例很难完成。

基于属性的方法。AutoPaG[24]还使用一个安全属性（如谓词）来创建补丁。然而，AutoPaG只处理一种类型的漏洞，缓冲区溢出，因此它不需要识别漏洞的类型来强制执行适当的安全属性。如果漏洞不是缓冲区溢出，则它无法生成正确的修补程序。

此外，如果需要强制执行安全属性的位置与发生漏洞的位置不在同一函数中，则AutoPaG无法生成修补程序。在我们的例子中，缓冲区溢出发生在rev函数中，但是补丁必须放在main中。

最后，AutoPaG通过在运行时检测代码来强制其安全属性。在清单1中，AutoPaG将在第8行的for循环中插入并检查缓冲区大小，从而导致性能开销。相反，Senx通过分析循环边界象征性地提取循环访问的内存范围，允许它检查循环外部的安全属性。

# 三、 问题定义

我们首先定义什么是Senx补丁，Senx补丁提供什么保证，以及如何定义Senx特定于漏洞的安全属性。

## A.补丁

要生成修补程序，Senx需要能够触发目标漏洞的输入。通常，这是一种输入类型，可以从概念验证漏洞或fuzzer生成的输入中派生，fuzzer可以使程序崩溃。由此，Senx生成一个补丁，确保它支持的所有安全属性都保持不变，其中每个安全属性对应于特定类型的漏洞。Senx修补程序可以采用以下两种形式之一：A）检测安全属性是否不再有效，如果有效，则引发错误，将程序执行从漏洞所在的路径引开（我们称之为check anderror修补程序）；b）防止违反安全属性（我们称之为repair修补程序）。

## B.安全性能

每个安全属性对应一个漏洞类型，是一个抽象的布尔表达式，当映射到程序中的具体变量时，可以对其进行计算。我们描述了Senx目前支持的两种安全属性。

缓冲区溢出。当遍历缓冲区的一系列内存访问从缓冲区内的内存位置交叉到缓冲区外的内存位置时，会发生缓冲区溢出。相应的Senx安全属性定义了两个抽象值：内存访问和缓冲区。Senx使用术语buffer来指代任何有界内存区域，其中可能包括结构、对象或数组。内存访问可能对应于数组解引用或指针解引用，但必须发生在循环内部。此安全属性既包括内存访问超过缓冲区上限范围的情况，也包括内存访问低于下限范围的情况（有时称为缓冲区下溢）。

糟糕的演员阵容。此安全属性检查由基指针的偏移量导致的内存访问是否小于基指针所指向的缓冲区的上限。虽然这种脆弱性可能是由于各种原因造成的，但是

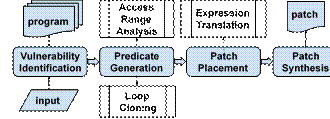


图1:Senx的工作流程：每个圆角矩形表示Senx的面片生成步骤，每个带竖线的矩形表示Senx的一个组件。

最常见的原因是程序员错误地将指针强制转换为与指针所指向的对象不兼容的类型。此安全属性可以防止简单结构和对象以及嵌套结构和对象的错误强制转换。

整数溢出。当给变量赋值大于或小于变量中可以表示的值时，就会发生整数溢出。如果溢出的结果用于分配缓冲区，则可能会出现整数溢出漏洞，从而导致缓冲区比预期的小得多。因此，相应的安全属性检查内存分配中使用的值是否不是整数溢出的结果。

作为一个原型，我们只从这三个漏洞类开始。尽管如此，它们具有足够的代表性，足以捕捉到相当比例的CVE漏洞。我们对2018年报告的CVE进行了非正式分析[15]。在我们分析时，已经报告了8507个漏洞，最流行的漏洞类别是拒绝服务、代码执行和溢出。通过随机检查这三个漏洞类别的100个CVE报告，我们发现25%的CVE漏洞属于缓冲区溢出、错误转换和整数溢出。我们相信Senx背后的原理可以扩展到其他漏洞类，我们打算在未来的工作中这样做。

# 四、 设计

## A.概述

Senx的目标是生成可以被开发人员轻松验证和采用的源代码补丁。Senx通过四个不同的步骤生成补丁：漏洞识别、谓词生成、补丁放置和补丁合成，如图1所示。

首先，在漏洞识别过程中，Senx获取一个程序和一个可以触发漏洞的输入，并输出违反的安全属性、程序中的漏洞点[4]和执行跟踪的源代码表达式。Senx利用concolic执行，使用触发输入的漏洞来生成输入的执行跟踪。然后，Senx检查执行跟踪，以确定违反了哪个安全属性以及违反了哪个安全属性的点，即漏洞点，该漏洞点也可以作为修补程序的候选位置。因为不是每个程序操作都会影响安全属性，所以Senx只检查可能影响安全属性的操作集的安全属性。例如，Senx只检查影响内存分配大小的整数是否溢出。执行跟踪被转换成表达式，用于合成源代码级补丁。由于我们的目标是修补源代码，这些表达式必须符合目标程序的编程语言的语法。

第二，在谓词生成中，Senx获取被侵犯的安全属性（也意味着漏洞的类型）和由漏洞识别生成的源代码表达式，并输出防止安全侵犯所需的谓词。谓词生成将上一步中标识的安全属性映射到程序源代码中变量和函数调用的具体表达式。因此，结果谓词表示适合于合成源代码级补丁的源代码表达式的安全属性。

第三，在补丁放置过程中，Senx使用漏洞识别中发现的漏洞点和谓词生成中产生的谓词来找到插入补丁的程序位置。这一步解决了查找谓词中所有必需变量都在作用域中的程序位置的问题。Senx使用表达式转换，如第IV-D节所述，将所有必要的变量转换为公共范围。对于检查和错误修补程序，Senx要求当前作用域具有一些要调用的错误处理代码。它使用Talos[19]来查找和选择错误处理代码。

最后，在补丁合成中，Senx获取补丁位置和实例化谓词来生成补丁代码。目前唯一可以生成修复补丁的漏洞类是整数溢出。在所有其他情况下，Senx生成一个检查和错误修补程序，检查修补程序谓词，并在谓词的计算结果为true时调用错误处理代码。

## B.漏洞识别

Senx在漏洞识别过程中使用的concolic执行引擎基于KLEE[7]。Senx使用vulnerabilitytriggering输入执行LLVM IR指令中的程序，直到违反安全属性为止。它将此点标记为漏洞点。此阶段的主要输出是脆弱点、违反的安全属性和一组表示已执行程序中变量的符号值的符号表达式。

我们修改了KLEE的llvmir执行引擎，以提取保留足够信息的表达式，从而可以轻松地转换回源代码。在大多数情况下，这涉及到将源代码符号名与LLVM一起存储

IR说明。

SEX还扩展了KLE，支持对诸如C/C++结构的复杂数据类型的支持，使得将符号表达式翻译成补丁生成的源代码语法成为可能。例如，结构的字段必须附加到其父对象，并且生成的语法会根据父对象是使用指针引用还是使用包含实际对象的变量引用而变化。数组和结构也可以嵌套，并且必须使用正确的语法来表示相对于顶级对象的嵌套级别。Senx有自己的IR来记录这些关系，因此可以将值转换回源代码表达式，该表达式包含父结构或数组的名称，而不仅仅是字段名或数组索引。这样，Senx就可以恢复foo→f.bar[10]等变量的完整表达式，如果没有这些扩展，Senx只知道该变量对应于数组中的第10个元素。

## C.谓词生成

谓词生成的目的是将前一阶段识别的违反安全属性的抽象布尔表达式映射到程序源代码中的具体表达式。Senx使用在脆弱性点违反安全属性的变量作为生成谓词的起点。

整数溢出属性的情况是最简单的。在这种情况下，当使用具有溢出的整数来确定缓冲区分配的大小时，将违反该属性。因此，Senx生成一个谓词，以防止整数变量溢出。造成这种脆弱性的原因有两种。在第一种情况下，程序将溢出运算的结果赋给大小大于运算大小的变量（例如，将溢出32位乘法的结果赋给64位变量）。在这种情况下，Senx通过修改操作的类型来生成修复补丁，以便不会发生溢出（例如，将32位乘法更改为64位乘法）。在第二种情况下，没有这种转让。因此Senx生成一个check和error谓词来检查结果是否小于预期值（例如，如果加法的结果小于它的任何一个输入）。

用于检测错误强制转换的谓词采用以下形式

（记忆）*访问>缓冲区上面的访问<缓冲区降低*

要生成这个谓词，Senx必须将∗映射到提供内存访问地址的源代码表达式，并开发捕获∗和∗的表达式。为了提取用于内存访问的表达式，Senx遍历嵌套的结构或类关系以生成一个表达式，该表达式包含生成正确的源代码表达式所需的任何父结构或对象。例如，要确定嵌套结构中某个字段的偏移量，Senx首先确定该字段与其直接父结构的偏移量，然后递归地累加每个嵌套级别的偏移量，直到可以确定与基结构的完全偏移量为止。*记忆接近缓冲器上面的缓冲器降低*

对于涉及使用基指针和偏移量进行内存访问的坏类型转换，缓冲区的下限由基指针给定。要为上限生成表达式，Senx需要找到分配缓冲区的点，以便提取缓冲区的大小。Senx首先尝试使用过程内路径敏感数据流分析将访问的缓冲区与分配点相匹配。如果这不能解析为唯一的分配点，Senx将使用从程序对漏洞触发输入的执行中派生的过程间调用历史来细化分析。一旦Senx找到分配点，它就从传递给分配函数的表达式（即malloc或某个变量）中导出大小。如果提取的表达式是正确的，并且内存分配被正确识别，那么这种计算是正确的，如果应用程序只使用Senx理解的标准内存分配函数，那么这种计算是正确的。我们的SENX原型目前只支持标准的LIB和C++内存分配功能。

最后，检测缓冲区溢出的谓词具有以下形式

（记忆）*接近上>缓冲区上面的接近下部<缓冲器降低*

提取缓冲区范围表达式的过程基于坏的cast情况，但有一些小的差异。因为缓冲区溢出的内存访问没有基指针，所以Senx通过记录同一条指令最后一次合法访问的缓冲区来推断非法内存访问可能要访问哪个缓冲区。然后通过找到缓冲区的分配点，类似地计算缓冲区的大小。在这种情况下，与其将sizeof（）参数传递给内存分配函数，不如将算术表达式传递给内存分配函数，以分配可变大小的数组。这种差异并不影响Senx，因为它的目标是提取表达式并将其注入源代码补丁，程序将在运行时对其进行编译和计算。

对于缓冲区溢出谓词，另一个要求是在发出一组顺序内存访问的循环中提取内存访问范围的表达式。这更复杂，因为循环可以执行的迭代次数可能不同。为了计算内存访问范围，Senx使用了两种互补的循环分析技术：循环克隆和访问范围分析。循环克隆和访问范围分析都将目标程序中的函数F和在缓冲区溢出中执行错误访问的指令inst作为输入，并返回inst的符号内存访问范围[A1，an]。然后可以将该符号访问范围与生成的修补程序中的分配范围进行比较，以确保安全。

访问范围分析。Senx依赖LLVM的内置循环规范化功能[8]来执行访问范围分析，该功能计算规范化循环的访问范围。循环规范化寻求将循环转换为标准形式，其中包含初始化循环迭代器变量的预头、检查是否结束循环的头和单个后缘。以这种方式提取单个循环的访问范围非常简单。主要的困难是扩展它来处理嵌套循环。

使用算法1中描述的算法对嵌套循环执行访问范围分析。它分析了循环

1                           字符（x，y）{\*

2                           返回（char）ma llo c（x y+1）；\*\*

3                           }

4

5                                                      int foo（字符）{\*

6                                                      +i f（（双）（c o l s+1）（s i z e/c o l s）+1>\*

7                                                      +行\*（c o l s+1）+1）

8                                                      +回路−1；

9                                                      char o u t p u t=f o o m a l o c（行，c o l s+1）；\*

10                                                   i f（！（上）

11                                                   回路−1；

12                                                   b a r（p，s i z e，c o l s，o u t p u t）；

13                                                   返回0；

14                                                   }

15

16                                                                            void b a r（字符src，int s i z e，int c o l s，字符d e s t）{\*\*

17                                                                            char\*p=d e s t；char\*q=s r c；

18                                                                            而（q<src+siz e）{

19                                                                            for（无符号j=0；j<cols；j++）

20                                                                            \*（p++）=\*（q++）；

21                                                                            \*（p++）=&apos;\n&apos;；

22                                                                            }

23                                                                            \*p=&apos;\0&apos;；

24                                                                            }

清单2：带有补丁（前缀为“+”）的缓冲区溢出CVE-2012-0947。

将内存访问指令inst封装在函数F中，从最里面的循环开始，迭代到最外面，累积循环归纳变量（包括inst使用的指针）的增量和减量。

我们使用清单2中的循环作为示例，说明如何将算法1应用于嵌套循环。在这种情况下，bar是函数F，inst是使用第42行的指针p进行的内存写入。对于每个循环，Sen通过调用helper函数find\u loop\u bounds来检索循环迭代器变量及其边界。Senx还获取循环的归纳变量列表及其值，这是通过调用另一个助手函数find\u loop\u updates，归纳变量在每次循环迭代中增加或减少的固定量。在我们的示例中，对于第19-20行中最里面的for循环，我们有=j=0，end=cols和j 7→1，p 7→1，q→7 1 In。*更新iter公司，首字母更新*

然后，算法1象征性地将对每个归纳变量的更新累积到引用的数据结构，该数据结构将每个归纳变量映射到表示对归纳变量的累积更新的表达式。例如，它将把j7→1、p7→1、q7→1存储到for最里面的for循环中。然后，它综合表达式来表示循环的总迭代次数。在算法的第16行，我们将得到=cols，它由（cols-0）/1简化而来。*acc公司acc公司计数*

有了总的迭代次数，它将每个归纳变量的累积更新乘以总的迭代次数。因此，在从第18行到第19行的循环之后，将有j7→colsp 7→colsq 7→cols*acc公司,,*

算法1。

完成后，它将继续分析下一个循环，如清单2所示，它是一个while循环，用于封闭内部for循环。因此，我们将使用算法1查找内存访问的访问范围。*仪器*

Input：：函数：中的内存访问指令*F级仪器F级*

输出：\访问的初始地址\访问的结束地址*acc公司最初的安装附件结束仪器*

1:程序分析\访问\范围

       2:：归纳变量的累计更新*. acc公司*

       3: ← ∅*acc公司*

       4:u2;←最里面的\_2;（仪器）*最里面的环环*

       5:最外层*最外层环环*

       6: ← ∅*访问*

       7:对于∈[innerst\ do]*我环，最外层环*

       8:←查找循环边界（F，l）*iter，首字母，结尾*

       9:←查找循环更新（l，已访问）*更新，已访问*

     10:象征性地添加导入更新*.*

     11:对于∈更新do*变量，upd*

     12:{var}←sym\u add（acc{var}，upd）*acc公司*

     13:结束

14:符号表示as的迭代次数*. 我计数*

     15:u1;←更新{iter}*升级版iter公司*

     16:←sym\_div（sym\_sub（end，initial），upd璝）*计数iter公司*

17:将归纳更新与*. 我*

     18:对于∈acc do*变量，upd*

     19:如果在最后一个循环（）中初始化了，那么*变量*

     20:{var}←sym\u mul（acc{var}，count）*acc公司*

     21:如果结束

     22:结束

     23:结束

     24:←获取指针（inst）*ptr公司*

     25:uu←循环\u头\u指令（最外层\uu）*第一仪器环*

     26:找到它的定义\_*. ptr公司第一仪器*

     27:u←达到定义（F，第一个）*acc公司最初的安装，ptr*

     28:←sym加（acc{p}）*acc公司结束首字母，根据*

29:返回\uuu30:结束程序*acc公司首字母，根据结束*



*iter公司*=q=src=src+size和p7→1 in在算法的第10行，j 7→cols，p7→cols+1，q 7→cols in和=size/cols在算法的第17行，最后j 7→colsp 7→（cols+1）\*（size/cols）q 7→size in。请注意，该算法不会将循环的迭代次数乘以j，因为j总是在最后分析的循环（最里面的for循环）中初始化。*，首字母，结束更新acc公司计数,,acc公司*

在分析了所有的循环之后，该算法得到了所使用的指针，并执行到达定义数据流分析，以找到到达最外层循环开始处的定义。例如，我们将使用=p和bar第16行的赋值p=dest作为p的到达定义。从这个到达定义中，它提取p的初始值，u2;=dest。最后，通过将初始值dest与p（cols+1）\*（size/cols）from的累积更新相加，再加上第22行通过指针p的最后一次写入，得到p的结束值u2;=dest+（cols+1）\*（size/cols）+1。因此它回来了*仪器ptr公司仪器ptr公司acc公司最初的acc公司结束acc公司*

[dest，dest+（cols+1）\*（size/cols）+1]作为表示访问范围[A1，An]的表达式。

我们的访问范围分析可以看作是一种基于模式的循环分析[14]，有几个不同之处。一方面，访问范围分析旨在将循环迭代次数导出为包含程序变量和/或常量的表达式，而基于模式的循环分析旨在将循环迭代次数导出为常量。另一方面，访问范围分析要求对循环进行规范化以适应基于模式的循环分析所需的模式，并依赖循环规范化来规范化循环。

循环克隆。访问范围分析不能应用于LLVM无法规范化的循环。对于这些循环，Senx使用循环克隆。作为一个例子，考虑清单3中的循环，其中Senx应用循环克隆来生成保留循环迭代次数的新代码，但删除了导致副作用的代码。然后可以使用新代码在生成的修补程序中安全地返回访问范围。

因为补丁必须插入到访问范围和分配范围都可用的函数中，所以循环克隆首先在分配缓冲区的函数和循环驻留的函数（即F）之间的调用上进行搜索。如果找不到这样的函数，Senx将无法生成修补程序。否则，它将指定找到的函数，然后将调用链中的每个函数从F克隆到返回访问范围的新代码。因此，每个fi要么是F的直接调用方，要么是F本身。*p我p我*

循环克隆需要满足两个要求：1）F必须计算访问范围并将访问范围传递给其调用者；2）F的任何直接或间接调用者必须将从其调用者处接收的访问范围向上传递给调用链上的下一个函数。使用以下步骤克隆每个FI。*我*

1） 循环克隆克隆Finto的整个代码*我*

克隆。*我*

2） 使用程序切片，它删除所有不需要的语句，以便计算访问范围或将访问范围传递给F。如果Fis F，它将保留inst的执行依赖的语句。如果fi是F的直接或间接调用方，则它保留对F的调用所依赖的语句。*p我我*

3） 它将F\u clone的返回类型更改为void，并删除F\u clone中的任何return语句。*我我*

4） 它将两个输出参数start和end添加到

克隆。如果是Fis F，它会在（嵌套的）循环之前插入语句，将错误访问中使用的指针或数组索引的初始值复制到start，并在循环之后插入语句，将此类指针或数组索引的结束值复制到end。如果Fis是F的调用者，它将更改call语句*我我我*

1                    int解码（const char in，char o u t）{\*\*

2                    国际一级；

3                    字符c；

4                    i=0；

5                    while（（c=（in++）！= &apos; \ 0 &apos; ) {\*

6                    If（c==&apos;\1&apos;）

7                    c=\*（in++）−1；

8                    输出[i++]=c；

9                    }

10                 返回i；

11                 }

12

13              字符自定义解码（const char data，int d a t a l e n）{\*\*

14              字符r e t=m al loc（d a t a l e n）；\*

15              i f（r e t&&！解码（d a t a+1，r e t））{

16              f r e（r e t）；

17              r e t=零；

18              }

19              返回时间；

20              }

清单3：一个复杂循环，包含一个复杂循环退出条件和对多个执行路径上的循环归纳变量的多个更新。

1                          +void d e c o d e \u c l o n e（const char in，char out，char\*\*

            \*\*s t a r t，字符\*\*结束）{

2                          字符c；

3                          +\*s t a r t=英寸；

4                          while（（c=（in++）！= &apos; \ 0 &apos; ) {\*

5                          If（c==&apos;\1&apos;）

6                          c=\*（in++）−1；

7                          }

8                          +\*结束=in；

9                          }

10

11                    字符自定义解码（const char data，int d a t a l e n）{\*\*

12                    字符r e t=m al loc（d a t a l e n）；\*

13                    +char\*s t a r t，\*结束；

14                    +d e c o d e l o n e（d a t a+1，r e t，&s t a r t，&end）；

15                    i f（r e t&&！解码（d a t a+1，r e t））{

16                    f r e（r e t）；

17                    r e t=零；

18                    }

19                    返回时间；

20                    }

清单4：一个克隆的切片循环，不再包含任何有副作用的语句并返回迭代次数。Senx添加或修改前缀为“+”的语句，以计算并返回循环迭代次数。

在调用参数列表中包含两个输出参数。

在克隆每个F之后，循环克隆会将对最后一个克隆的函数的调用插入到F中，从而返回start和end中的访问范围。随后将合成一个补丁以利用返回的访问范围。*我p*

要了解循环克隆是如何工作的，请考虑清单3中的示例，它提供了一个循环，该循环改编自PHP中的一个实际缓冲区溢出漏洞CVE-2007-1887[38]，PHP是一个脚本语言解释器。函数解码时发生缓冲区溢出。循环具有复杂的循环退出条件和对循环归纳变量in的多次更新，这些更新取决于in所指向的缓冲区的内容。循环克隆的结果如清单4所示。使用decode作为F调用循环克隆，第5行的错误访问作为inst。它首先发现udf\u decode函数位于要解码的调用链上，并且在该链中分配范围可用。因为udf\u decode直接调用decode，所以它只需要克隆decode。然后，它将函数decode克隆为decode\u clone，然后应用程序切片以第5行和变量c和in作为切片标准对\u clone进行解码。decode在第8行也有一个潜在的写缓冲区溢出，但是在这个例子中，我们着重于生成一个谓词来检查in是否可以超过它所指向的缓冲区的末尾。程序切片使用向后分析，并删除与第5行的c和in值无关的所有语句，包括第2、4和8行。程序切片后，它将decode\u clone的返回类型更改为void，并删除所有返回语句。在解码克隆的参数列表中增加了两个输出参数start和end。然后在第3行插入一条语句，将in的初始值复制到循环之前的start，在第8行插入一条语句，将in的结束值复制到循环之后的end。最后，它在函数udf\ U decode中插入一个在第14行对\ U clone进行解码的调用和一个在第13行声明start和end的语句。

循环克隆产生的新代码不能有任何副作用。如果不能产生这样一个无副作用的切片，Senx将中止补丁生成。

函数调用。在某些情况下，提取的表达式包含函数调用的结果。在这种情况下，Senx必须小心，不要在生成的谓词中调用有副作用的函数。

Senx将副作用定义为：1）对函数外部可见内存的可能更改，或2）对具有外部影响的系统调用的调用，或3）对具有任何副作用的函数的调用。我们把没有副作用的函数称为安全函数。首先，它认为对全局变量的任何写入都是一种副作用。为保守起见，它还将通过传递给函数的指针参数进行的任何写入视为副作用。其次，它使用了一个没有外部影响（如更改文件系统状态或输出到设备或网络）的常见API函数和系统调用的白名单。在白名单上调用函数或系统调用被认为没有副作用。

Senx使用流敏感、上下文不敏感的过程内静态分析来识别安全函数列表。在开始时，列表只包含白名单上的函数。Senx对目标程序的每个函数执行分析，并将没有副作用的每个函数添加到列表中。

对于每个函数，Senx都维护一组可能“不安全”的被调用函数。每当一个新函数F被添加到列表中时，对于调用F的每个函数，Senx都会从“不安全的”被调用函数集中删除F。当这样的函数集变为空时，Senx会将该函数视为“安全的”，并将该函数添加到列表中。

## D.表达式翻译

因为Senx生成的补丁是源代码补丁，所以补丁的谓词必须在单个函数范围内求值。然而，有时分配范围在一个功能范围中计算，而存储器访问范围在不同的功能范围中计算。因此，表示分配范围的表达式和表示内存访问范围的表达式在单个函数范围内并不都有效。要使表达式在单个函数作用域中都有效，一种可能的解决方案是将源函数作用域中有效的表达式作为调用参数发送到表达式无效的目标函数作用域。这种方法需要向目标函数添加新的函数参数，并在目标函数的每个调用位置添加相应的调用参数。我们决定不使用这种方法，因为它需要对从源函数到目标函数的调用链上的任何函数进行代码更改。此外，调用任何已更改函数的不相关函数也必须更改，这会导致非常具有侵入性的修补程序。

*表达式翻译*通过将源函数作用域中的表达式转换为目标函数作用域中的等效表达式来解决此问题。它不需要像前面提到的解决方案那样添加新的函数参数或调用参数。Senx使用表达式转换将缓冲区大小表达式和内存访问范围表达式转换为单个函数作用域，在该函数作用域中对谓词进行求值。我们称之为聚合谓词的过程。*出口fs公司出口fd公司*

在较高的层次上，表达式翻译可以看作是一种轻量级的函数摘要[17]。函数摘要建立函数的输入和输出之间的关系，表达式翻译建立函数的输入和函数的局部变量子集之间的关系。它的工作原理是利用调用者传递到函数中的参数与在被调用者的作用域中接受参数值的参数之间的等价性。使用这种等价性，表达式转换可以迭代地转换传递给调用图中的函数调用的表达式。形式上，表达式翻译可以将表达式、iff中的内存访问范围表达式和iff中的缓冲区大小表达式之间的比较沿着连接的边集E和程序调用图中的边集收敛，一种表达式，相当于或沿路径形成连续的边集，以便和可以沿这些边集转换为公共范围。*扩展fa公司出口fs公司fa公司fs公司扩展出口扩展出口*

请注意，程序所声明的变量在不同的函数（如C/C++）中的全局变量之间不需要进行转换，但使用这种变量并不十分普遍。我们把函数参数和这类变量统称为非局部变量。我们把只包含非局部变量的表达式称为非局部表达式。

Senx中表达式翻译的底层实现由两个函数组成。算法2中列出的函数translate\u se\u to\u scopes是表达式翻译的核心。它将特定表达式转换为调用堆栈中所有候选函数的作用域。*出口堆栈*

算法2将表达式转换为调用堆栈上每个函数的作用域。

Input：：调用堆栈由调用指令的有序列表组成*堆栈*

*出口*：要翻译的表达式：与之关联的指令*仪器出口*

输出：\在调用堆栈上每个调用函数的作用域中转换的*翻译出口出口*

1:过程将\u SE \u转换为\u作用域

2:转换为一个表达式，其中所有变量都是*. 出口功能*

       3:←获取函数（）*功能仪器*

       4:←使\u非局部\u expr（）*出口函数、指令、表达式*

       5:如果6=∅那么*出口*

       6:对于∈堆栈do*呼叫*

7:用中使用的相应参数替换中的每个参数变量*. 出口呼叫*

       8:←用参数替换参数*出口呼叫，出口*

       9:←获取函数（）*功能呼叫*

     10:u3;[func]←表达式*翻译出口*

     11:←使\u非局部\u expr（）*出口函数，调用，表达式*

     12:如果=∅那么*出口*

     13:休息

     14:如果结束

     15:结束

     16:如果结束

     17:返回\_*翻译出口*

18:结束程序



我们将用清单2中的代码说明它是如何工作的。为了简单起见，我们使用源代码行号来表示相应的指令。为了转换缓冲区溢出所涉及的缓冲区大小，Senx发现缓冲区是从与每个内存分配相关联的调用堆栈的第2行的malloc调用分配的，并调用translate\u se\u To\u scopes with=[line 9]，=“x\*y+1”，=line 2，=foo\u malloc。*堆栈出口仪器功能*

该函数首先将“x\*y+1”转换为一个定义，其中变量是foo\u malloc的所有参数，如果这种转换是可能的，我们称之为非局部定义。这个转换是由算法3中列出的make\_nonlocal\_expr函数完成的，它试图为中的每个变量找到一个非局部定义，然后用匹配的非局部定义替换每个变量。make \u nonlocal \u expr依赖于find \u nonlocal \u def \u for \u var，它递归地查找函数中局部变量的定义，最终根据函数参数、全局变量或函数调用的返回值为它们建立定义。请注意，非局部定义只能是算术表达式的形式，而不涉及任何函数。在这种情况下，结果也是“x\*y+1”，因为x和y都是foo\u malloc的参数。*出口出口*

算法3生成一个非局部表达式。

Input：：函数：中的指令：与*f仪器f表达式仪器*

输出：非本地化*非局部的出口出口*

|  |  |
| --- | --- |
| 1:程序生成非本地表达式  2:存储中每个变量的非局部定义*. 映射出口* | |
| 三： | *映射*← ∅ |
| 4: | 对于∈expr do*变量* |
| 5: | 如果¨is \u var \u nonlocal（），那么*f、 变量* |
| 6: | *定义*←查找非局部变量*f、 仪器，变量* |
| 7: | 如果=∅那么*定义* |
| 8: | *.* 我们找不到的非局部定义*变量* |
| 9: | 回流管∅ |
| 10: | 其他的 |
| 11: | *映射*[var]←定义 |
| 12: | 结束if |
| 13: | 结束if |
| 14: | 结束 |
| 15: | *.* 将每个变量的出现处替换为 |
| 非局部定义  16:u←替换\_vars（）*非局部的出口表达式，映射*  17:返回\_*非局部的出口*  18:结束程序 | |



然后，它从第9行开始迭代中的每个调用指令。对于每个调用指令，它用调用指令中使用的参数替换中的参数。对于第9行，它通过调用helper函数substitute \u parms \u with \u args，分别用行替换x，用cols+1替换y。因此，“x\*y+1”变为“rows\*（cols+1）+1”。因此，它将“rows\*（cols+1）+1”与函数foo相关联，并将关联存储在｜，因为第9行存在于函数foo中。之后，它尝试将“rows\*（cols+1）+1”转换为关于foo的非局部定义。此时，它将停止，因为行和列都被分配了函数extract\ int的调用返回值。否则，它将转到调用堆栈上的下一个函数，并继续向上转换调用堆栈。然而，在这种情况下，表达式转换还能够将内存访问范围表达式从bar的范围转换为foo的范围。因此，Senx将patch谓词放在foo中。如果表达式转换无法收敛表达式，Senx将中止补丁生成。*堆栈出口出口翻译*

# 五、实施

我们已经将Senx实现为KLEE LLVM执行引擎的扩展[7]。与KLEE一样，SENX也适用于编译成LLVM位代码的C/C++程序[48 ]。

我们重复使用KLEE的LLVM位码执行部分，如第IV-B节所述，来实现我们的表达式生成器，但不使用KLEE的任何约束集合或求解部分。为了简单和易于调试，我们将表达式表示为文本字符串。为了支持算术运算和简单的数学函数表达式，我们利用GiNaC，C++库设计，以支持代数表达式的符号操作[1 ]。

我们实现了一个单独的LLVM转换过程，用循环上的信息（如循环前头和头的标签）注释LLVM位码，这些信息随后被访问范围分析使用。这个过程依赖于LLVM对自然循环的规范化来规范化循环[8]。我们将LLVMSlicer[46]扩展为循环克隆。为了定位错误处理代码，我们使用了Talos[19]。

我们的内存分配记录器使用KLEE插入内存分配，并为每个内存分配存储调用堆栈。Senx扩展KLEE来检测整数溢出，并将现有的内存故障检测合并到KLEE中来触发补丁生成。对于别名分析，Senx利用DSA指针分析[22]。

SEX是用2543行C/C++源代码实现的，不包括用于识别错误处理代码的TALOS组件[19 ]。

# 六、 讨论

我们的Senx原型支持三种漏洞类型：缓冲区溢出、错误转换和整数溢出。我们相信，安全特性可以用于为许多其他脆弱性类型生成补丁。因为我们的目标是生成可供开发人员采用的源代码补丁，所以只要漏洞类型的安全属性可以具体化为目标程序源代码中的程序表达式，这种方法就可以应用于任何漏洞类型。

例如，可以使用关于哪些操作被视为检查以及哪些操作被视为使用的信息，为诸如检查时间到使用时间漏洞之类的时间漏洞编写安全属性。还可以为缺少的安全检查漏洞编写安全属性，其中包含有关哪些操作需要安全检查以及使用哪些API函数来执行这些安全检查的信息。虽然提供这样的信息可能需要开发人员付出一些努力，但只需要做一次。

我们注意到，对于某些漏洞，源代码中需要额外的工具。在我们对Senx的真实漏洞评估中，我们发现阻止Senx生成修补程序的最常见原因是无法找到一个公共程序作用域，在该作用域中，合成谓词所需的所有程序表达式都可用。对于这种情况，使用在函数作用域之间传递所需表达式的附加参数创建函数克隆的能力将使Senx也能够覆盖这些情况。我们计划在将来的工作中为Senx添加这样的功能。

# 七。评价

首先，我们评估了Senx在修复现实世界漏洞方面的有效性。其次，我们评估由Senx生成的补丁的质量。我们手动检查所有

|  |  |  |
| --- | --- | --- |
| 应用程序。 | 说明 | SLOC公司 |
| 自动追踪 | 将位图转换为矢量图形的工具 | 19,383 |
| 比努蒂尔斯 | 用于管理和创建二进制程序的编程工具的集合 | 2,394,750 |
| 立明 | 用于创建adobeflash文件的库 | 88,279 |
| 利伯蒂夫 | 用于操作TIFF图形文件的库 | 71,434 |
| 菲律宾比索 | PHP编程语言的官方解释程序 | 746,390 |
| 橄榄石 | 关系数据库引擎 | 189,747 |
| ytnef公司 | TNEF流读取器 | 15,512 |
| zziplib公司 | 用来阅读ZIP档案的图书馆 | 24,886 |
| 碧玉 | JPEG标准的编解码器 | 30,915 |
| 库存档 | 多格式存档和压缩库 | 158,017 |
| 杂烩 | 跟踪位图图形的工具 | 20,512 |
| 总计 | 不适用 | 3,817,268 |

表二：用于测试真实世界漏洞的应用程序。

生成正确的补丁，并将它们与开发人员创建的补丁进行比较。为了篇幅，我们只详细描述了其中的两个补丁。第三，我们将Senx与最先进的APR工具（包括Angelix和SemFix）进行比较。最后，我们使用一个更大的数据集来衡量循环克隆、访问范围分析和表达式翻译的适用性。

## A.实验装置

我们通过搜索在线漏洞数据库[10]、[15]、[33]、软件缺陷报告数据库、开发人员邮件组[5]、[37]、[41]和漏洞数据库[35]，选择Senx常用应用程序中的漏洞进行修补。我们关注的漏洞属于Senx目前可以处理的三种类型的漏洞之一。然后，我们选择满足以下三个标准的漏洞：1）触发漏洞的输入可用或可以根据可用信息创建，2）易受攻击的应用程序可以编译成LLVM位码并由KLEE正确执行，3）易受攻击的应用程序使用标准内存分配函数（如malloc）来分配内存，因为Senx目前依赖于此来推断对象的分配大小。Senx目前不支持使用自定义内存分配例程的应用程序。我们从安全研究人员的博客、错误报告、漏洞数据库、软件用户的邮件组或补丁提交的测试用例[2]、[3]、[6]、[35]、[44]、[56]中获取触发漏洞的输入或有关此类输入的信息。

从中，我们选择了42个真实世界中的缓冲区溢出、错误转换和整数溢出漏洞以及概念验证漏洞，以评估Senx在修补漏洞方面的有效性。这些漏洞来自表二所示的11个应用程序，其中包括8个媒体和归档工具和库、PHP、sqlite以及一组用于管理和创建二进制程序的编程工具。相关的漏洞包括19个缓冲区溢出、13个错误强制转换和10个整数溢出。

我们的所有实验都是直接在这些易受攻击的应用程序上进行的，这些应用程序位于一个具有四核3.40GHz Intel i7-3770 CPU、16GB RAM、3TB SATA硬盘和64位Ubuntu 14.04的桌面上。

*B.Senx在修补漏洞方面有多有效？*

对于应用程序的每个漏洞，我们在Senx下运行相应的程序，并使用漏洞触发输入。如果Senx生成了一个补丁，我们将手动检查补丁的正确性。如果Senx中止补丁生成，我们将检查是什么导致Senx中止。

我们的结果总结在表III中。“Type”列表示该漏洞是“B”缓冲区溢出、“C”错误转换还是“I”整数溢出。列“表达式”显示了Senx是否能够成功地构造合成补丁所需的所有表达式，因为某些代码构造可能包含超出Senx在其符号ISA中支持的理论的表达式。“循环分析”描述如果漏洞包含循环，是否使用循环克隆或访问范围分析（ARA）。“Patch Placement”列出了修补程序放置的类型：“trivity”表示修补程序放置在与漏洞相同的函数中，“Translated”表示修补程序必须转换为不同的函数。“数据访问”描述补丁谓词是否涉及复杂的数据访问，例如结构或数组索引中的字段。最后，“修补了？总结了Senx生成的修补程序是否修复了漏洞。Senx中止生成补丁的10个漏洞以红色突出显示。

在42个漏洞中，Senx生成了32个（76.2%）补丁，根据我们的三个标准，这些补丁都是正确的。在13个修补的缓冲区溢出中，循环分析大致分为循环克隆和访问范围分析（分别为6个和8个）。Senx选择不使用循环克隆主要有两个原因。首先，由于别名分析不精确，无法正确区分结构的不同字段，Senx使用的程序切片工具可能包含与将循环迭代计算成切片无关的指令。不幸的是，这些指令调用的函数可能有副作用，因此Senx不能使用切片。第二，在某些情况下，整个循环体的控制依赖于调用具有副作用的函数的结果。例如，CVE-2017-5225中涉及的循环仅在调用malloc成功时执行。因为malloc可以进行系统调用，Senx也不能克隆循环。

Senx必须将23.8%的补丁放在与存在漏洞的函数不同的函数中。这对于缓冲区溢出（31.6%的情况）尤其严重，它们必须将缓冲区分配与内存访问范围进行比较。这说明表达式转换对Senx的补丁生成能力有很大的贡献，特别是对于缓冲区溢出，缓冲区溢出构成了大部分内存损坏漏洞。48.5%的修补程序中也使用了Senx处理复杂数据访问的功能，这表明处理大量漏洞需要这种能力

Senx中止10个漏洞的补丁生成。这些中止的主要原因是Senx无法收敛到一个函数范围，在这个范围内，patch谓词中的所有符号变量都可用。还有一种情况（jasper-CVE-2017-5501），Senx无法找到合适的错误处理代码来合成补丁。在这些情况下，补丁程序需要对应用程序代码进行超出Senx能力范围的更重要的更改。在其他情况下，Senx检测到有多个补丁谓词的到达定义，但它没有执行输入。目前，Senx只接受单个漏洞触发输入执行的一个执行路径。在未来，我们计划通过允许Senx接受多个输入来覆盖其他到达定义存在的路径来处理这些情况。最后，由于循环克隆和访问范围分析都失败，Senx会因几个漏洞而中止。

*C.生产的贴片质量如何？*

对于Senx生成的每个补丁，我们手动检查补丁的正确性。为了确定修补程序是否正确，我们应用以下三个测试：a）将修补程序应用于目标程序，并验证漏洞触发输入不再触发漏洞；b）运行目标程序供应商提供的内置测试套件，以验证整个测试套件是否通过，c） 我们通过手动检查Senx生成的补丁是否以与官方补丁相同的方式影响目标程序的行为来检查与供应商发布的官方补丁的语义等价性（如果可用），并通过手动分析代码来检查语义正确性。我们认为只有当所有三个测试都通过时，修补程序才是正确的。我们的检查发现所有生成的补丁都是正确的。

在生成的32个补丁中，我们选择了2个补丁进行详细描述。

libtiff-CVE-2017-5225。这是libtiff中的堆缓冲区溢出，可通过巧尽心思构建的TIFF图像文件加以利用。溢出发生在函数cpContig2SeparateByRow中，该函数将TIFF图像解析为行，并根据每行的像素数和每像素的位数动态分配一个缓冲区来保存解析的图像。通过使用不一致的每像素位参数，攻击者可以导致libtiff分配小于像素数据大小的缓冲区，并导致缓冲区溢出。

当Senx通过使用特制的TIFF图像文件运行libtiff来捕获缓冲区溢出时，它首先标识缓冲区是使用变量scanlinesizein的值分配的，并且缓冲区的起始地址存储在变量inbuf中。因此它使用[inbuf，inbuf+scanlinesizein]来表示缓冲区范围。然后，Senx发现缓冲区溢出发生在3级嵌套循环中，用于访问缓冲区的指针依赖于循环诱导变量。Senx将该漏洞分类为缓冲区溢出。

循环克隆失败，因为循环片依赖于对\u TIFFmalloc的调用，后者随后调用malloc。因此，Senx应用了访问范围分析。访问范围分析检测到只有最外层和最内层的循环影响内存访问指针，并从提取的归纳变量计算表达式[inbuf，inbuf+spp\*imagewidth]来表示访问范围。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 应用程序。 | CVE公司# | 类型 | 表达 | 回路分析 | 贴片放置 | 数据存取 | 修补了吗？ | | 橄榄石 | CVE-2013-7443 | 我 | 确定的 | — | 失败 | — | 7 | |  | CVE-2017-13685 | 我 | 确定的 | — | 琐碎的 | 简单 | 3 | | zziplib公司 | CVE-2017-5976 | B类 | 确定的 | 克隆 | 翻译 | 复杂 | 3 | |  | CVE-2017-5974 | 我 | 确定的 | — | 翻译 | 复杂 | 3 | |  | CVE-2017-5975 | 我 | 确定的 | — | 翻译 | 复杂 | 3 | | 杂烩 | CVE-2013-7437 | C级 | 确定的 | — | 琐碎的 | 复杂 | 3 | | 立明 | CVE-2016-9264 | 我 | 确定的 | — | 琐碎的 | 简单 | 3 | | 利伯蒂夫 | CVE-2016-9273 | B类 | 不确定 | — | — | — | 7 | |  | CVE-2016-9532 | B类 | 确定的 | 克隆 | 琐碎的 | 复杂 | 3 | |  | CVE-2017-5225 | B类 | 确定的 | 阿拉 | 琐碎的 | 简单 | 3 | |  | CVE-2016-10272 | B类 | 确定的 | 阿拉 | 翻译 | 简单 | 3 | |  | CVE-2016-10092 | 我 | 确定的 | — | 翻译 | 简单 | 3 | |  | CVE-2016-5102 | 我 | 确定的 | — | 琐碎的 | 简单 | 3 | |  | CVE-2006-2025年 | C级 | 确定的 | — | 琐碎的 | 复杂 | 3 | | 库存档 | CVE-2016-5844 | C级 | 确定的 | — | 琐碎的 | 复杂 | 3 | | 碧玉 | CVE-2016-9387 | C级 | 确定的 | — | 琐碎的 | 复杂 | 3 | |  | CVE-2016-9557 | C级 | 确定的 | — | 琐碎的 | 复杂 | 3 | |  | CVE-2017-5501 | C级 | 确定的 | — | 失败/错误处理 | — | 7 | | ytnef公司 | CVE-2017-9471 | B类 | 确定的 | 克隆 | 琐碎的 | 简单 | 3 | |  | CVE-2017-9472 | B类 | 确定的 | 克隆 | 琐碎的 | 简单 | 3 | |  | CVE-2017-9474 | B类 | 确定的 | 失败 | — | — | 7 | | 菲律宾比索 | CVE-2011-1938 | B类 | 确定的 | 阿拉 | 翻译 | 简单 | 3 | |  | CVE-2014-3670 | B类 | 确定的 | 阿拉 | 翻译 | 复杂 | 3 | |  | CVE-2014-8626 | B类 | 确定的 | 克隆 | 琐碎的 | 简单 | 3 | | 比努蒂尔斯 | CVE-2017-15020 | B类 | 确定的 | 阿拉 | 翻译 | 简单 | 3 | |  | CVE-2017-9747 | B类 | 确定的 | 克隆 | 翻译 | 简单 | 3 | |  | CVE-2017-12799 | 我 | 确定的 | — | 琐碎的 | 简单 | 3 | |  | CVE-2017-6965 | 我 | 确定的 | — | 失败 | — | 7 | |  | CVE-2017-9752 | 我 | 确定的 | — | 翻译 | 简单 | 3 | |  | CVE-2017-14745 | C级 | 确定的 | — | 失败 | — | 7 | | 自动追踪 | CVE-2017-9151 | B类 | 不确定 | — | — | — | 7 | |  | CVE-2017-9153 | B类 | 不确定 | — | — | — | 7 | |  | CVE-2017-9156 | B类 | 确定的 | 阿拉 | 琐碎的 | 简单 | 3 | |  | CVE-2017-9157 | B类 | 确定的 | 阿拉 | 琐碎的 | 简单 | 3 | |  | CVE-2017-9168 | B类 | 确定的 | 失败 | — | — | 7 | |  | CVE-2017-9191 | B类 | 确定的 | 阿拉 | 失败 | — | 7 | |  | CVE-2017-9161 | C级 | 确定的 | — | 琐碎的 | 简单 | 3 | |  | CVE-2017-9183 | C级 | 确定的 | — | 琐碎的 | 复杂 | 3 | |  | CVE-2017-9197 | C级 | 确定的 | — | 琐碎的 | 复杂 | 3 | |  | CVE-2017-9198 | C级 | 确定的 | — | 琐碎的 | 复杂 | 3 | |  | CVE-2017-9199 | C级 | 确定的 | — | 琐碎的 | 复杂 | 3 | |  | CVE-2017-9200 | C级 | 确定的 | — | 琐碎的 | 复杂 | 3 |   表III:Senx生成的补丁 |

因为缓冲区范围和访问范围都从inbuf开始，所以Senx将patch谓词合成为spp\*imagewidth>scanlinesizein。然后，Senx发现cpContig2SeparateByRow包含错误处理代码，该代码的标签为bad，并生成如下修补程序。由于缓冲区分配和溢出发生在同一个函数中，Senx将补丁放在缓冲区分配之前。

     i f（spp imagewidth>s c a n l i n e s i z e i n）\*

         变坏了；

官方补丁调用相同的错误处理，并放置在与Senx补丁相同的位置。然而，官方补丁检查“（bps！= 8)”. 通过进一步的分析，我们发现这两个补丁是等价的，尽管人工生成的补丁依赖于libtiff格式的语义，而Senx的补丁直接检查循环不能超过分配的缓冲区的大小。

libarchive-CVE-2016-5844。libarchive中的ISO解析器中的此整数溢出可通过巧尽心思构建的ISO文件导致拒绝服务。当函数choose\u volume将块索引（32位整数）与常量相乘时，会发生溢出。这可能会超过由32位整数表示的最大值，并溢出为负数，然后用作文件偏移量。

Senx在使用特制的ISO文件运行libarchive的ISO解析器时检测到整数溢出。它将溢出值的表达式生成为2048和vd→location的乘积。此外，Senx检测到溢出值被分配给64位变量skipsize，因此将其分类为可修复整数溢出。Senx通过在相乘之前将32位值强制转换为64位值来修补此漏洞：

−s k i p s i z e=逻辑块大小vd−>l o c a t i o n；+s k i p s i z e=2048（i n t 6 4−t）vd−>l o c a t i o n；\*\*

官方补丁与Senx生成的补丁基本相同。唯一的区别是，官方补丁使用的是常量逻辑块大小，而不是乘法中的等价值2048。

## D.与其他工作的比较

为了说明Senx能否解决现有APR工具的局限性，我们评估了Senx对SemFix[34]和Angelix[30]的有效性。因为SemFix和Angelix需要为每个应用程序和漏洞付出相当大的努力，所以我们只能对一个应用程序autotrace的两个漏洞进行比较。我们将所有四个内置测试输入用于autotrace，将50个随机生成的输入用于示例，并为每个漏洞使用一个易受攻击的触发输入。

在这两种情况下，Semfix和Angelix都无法生成修补程序，这可能是因为它们无法定位要更改的现有程序构造以通过正测试输入和负测试输入，或者它们无法合成保护语句以防止触发漏洞。另一方面，Senx能够为这两个漏洞生成工作补丁。

## E.适用性

我们评估循环克隆、访问范围分析和表达式翻译在更大数据集上的适用性。为了生成这样的数据集，我们从GNU Coreutils中的11个程序中提取访问内存缓冲区的所有循环以及这些缓冲区的分配，而不管它们是否包含漏洞。然后我们将Senx的循环分析应用于所有的循环，发现循环克隆可以应用于88%的循环，访问范围分析可以应用于46%的循环。这与我们从漏洞得出的结果是一致的。为了篇幅，我们在附录中描述了这些实验的细节。

# 八。相关工作

## A.自动生成补丁

利用修复模式。通过观察常见的人类开发者生成的补丁，PAR使用固定模式生成补丁，例如改变方法参数、添加空校验器、调用具有相同参数的另一个方法、以及添加数组绑定检查器[20 ]。SEX与PAR在两个方面不同。首先，PAR是无法生成补丁时，正确的变量或方法合成一个补丁是不可访问的故障函数或方法。第二，PAR使用尝试和错误的方法，不仅尝试在给定的bug上对每个固定模式进行尝试，而且尝试在故障函数或方法中可访问的变量或方法来合成补丁。相反，Senx采用了一种引导性的方法来识别给定bug的类型，并选择相应的补丁模型为bug生成补丁，并根据补丁模型提供的语义信息系统地找到正确的变量来合成补丁。

LeakFix通过在正确的程序位置为泄漏的内存分配添加内存释放语句来修复内存泄漏错误[16]。通过将程序抽象为只包含与内存分配、释放和使用相关的程序语句的CFG，LeakFix将查找内存泄漏修复的问题转化为搜索CFG边缘，在该边缘可以添加内存释放语句来修复内存泄漏，而不会引入新的bug。

使用程序突变。GenProg是一项开创性的工作，它能诱导程序突变，即通过遗传编程生成补丁[55]。利用测试套件，它将重点放在为阴性测试用例而不是为阳性测试用例执行的程序代码上，并利用程序突变产生对程序的修改。作为对其程序变异算法的反馈，它考虑了修改后的程序通过的正测试用例和负测试用例的加权和。将程序突变的所有结果视为一个搜索空间，其继承者通过改变使用补丁而不是抽象语法树来表示修改和利用搜索空间并行性来提高可伸缩性[23]。

使用SMT解算器。SemFix[34]和Angelix[30]使用约束求解来找到所需的表达式，以替换程序中使用的不正确表达式。通过使用触发缺陷的输入和不触发缺陷的输入来象征性地执行目标程序，它们确定了目标程序必须满足的约束，以便正确处理这两种输入。然后，他们使用基于组件的程序合成（componentbased program synthesis）来合成一个补丁，该补丁将变量、常量和算术运算等组件结合起来，合成一个符号表达式，使目标程序满足所识别的约束。

从正确的代码中学习。Prophet从现有的正确补丁中学习[27]。它对从每个补丁的抽象树中提取的两个特征使用参数化对数线性概率模型：1）补丁修改原始程序的方式；2）补丁访问的值如何被原始程序和被补丁程序使用之间的关系。在概率模型中，它将为缺陷生成的候选补丁按其正确性概率进行排序。最后使用测试套件测试候选补丁的正确性。与其他生成和验证自动补丁生成技术一样，它的有效性取决于测试套件的质量。

## B.缓解安全漏洞

强化计划。防止漏洞被利用的一种方法是加强程序，使其对恶意输入更具鲁棒性。软件故障隔离（SFI）仪器在内存操作前进行检查，以确保它们不会损坏内存[18]、[31]、[52]、[57]。控制流完整性（CFI）学习程序的有效控制流传输，并验证控制流传输以防止执行攻击代码[13]、[51]、[58]。或者，一些技术修改关键内存区域的布局或权限，以检测或防止漏洞攻击[9]、[12]、[43]、[45]、[49]、[50]。

相比之下，Talos引入了快速响应安全解决方案（SWRR）的概念，它引导程序执行远离错误处理代码的漏洞，并将SWRR插入程序以防止恶意输入触发漏洞[19]。

虽然这些技术以额外的运行时开销或程序功能损失为代价来防止漏洞被利用，但Senx生成修补程序来修复漏洞，而不会产生这样的成本。

过滤输入。一些技术检测并简单过滤恶意输入[4]、[11]、[28]、[47]、[53]。其中，Bouncer结合静态分析和符号分析来推断攻击漏洞的约束，并生成一个输入过滤器来删除这些恶意输入[11]。Shields将漏洞建模为协议状态机，并在此基础上构建网络过滤器[53]。

虽然这些技术中的大多数是通过恶意输入的语义或语法来识别恶意输入，但其中一些技术侧重于漏洞的语义[4]，[28]。与这些技术类似，Senx还使用漏洞的语义来合成补丁。但是，Senx有一个不同的目标，即生成修补程序来修复漏洞。

整流输入。除了过滤输入，一些技术还可以纠正恶意输入，以防止它们触发漏洞。通过污点分析，SOAP通过观察良性输入的程序执行来学习对输入的约束。它从学习到的约束中识别出违反约束的输入，并尝试更改输入以使其满足约束。这样做，不仅使输入无害，而且允许正确处理校正输入中的所需数据[25]。

根据观察，嵌入在输入中的攻击代码通常容易受到任何轻微更改的影响，A2C使用一次性字典对输入进行编码，并且仅当程序执行超出可能存在漏洞的路径时才对其进行解码，以禁用嵌入的攻击代码[21]。

# 九、 结论

本文介绍了Senx系统的设计和实现，该系统利用人类指定的安全属性来生成缓冲区溢出、错误转换和整数溢出漏洞的修补程序。Senx使用了本文介绍的三种新的程序分析技术：循环克隆、访问范围分析和表达式翻译。此外，SENX使用一个表达式表示，便于将从符号执行中提取的表达式翻译成C/C++源代码。通过这些技术，Senx可以为42个真实世界漏洞中的76%正确生成补丁。Senx的主要限制是循环克隆的别名分析精度有限，并且无法收敛表达式以在程序中找到所有必需变量都在范围内的位置。

# 确认

这项研究部分得到了NSERC发现基金（RGPIN 2018-05931）和加拿大研究主席（950-228402）的支持。

# 参考文献

[1] 吉纳克不是中科院。http://www.ginac.de/, 2018. 查阅日期：2018年5月。

[2] http://blogs.gentoo.org/ago, 2018. 查阅日期：2018年5月。

[3] http://github.com/asarubbo/poc, 2018. 查阅日期：2018年5月。

[4] BRUMLEY，D.，NEWSOME，J.，SONG，D.，WANG，H.，和JHA，S。

        自动生成基于漏洞的特征码。在

*2006年IEEE安全与隐私研讨会论文集*（华盛顿特区，美国，2006），SP&apos;06，IEEE计算机学会，第2-

16.

[5] http://lists.gnu.org/archive/html/bug-coreutils/, 2018. 查阅日期：2018年5月。

[6] http://bugzilla.maptools.org/, 2018. 查阅日期：2018年5月。

[7] CADAR，C.，DUNBAR，D.和ENGLER，D.Klee：复杂系统程序的高覆盖率测试的自动生成。第八届USENIX操作系统设计与实现会议论文集（美国加州伯克利，2008），OSDI&apos;08，USENIX协会，第209-224页。

[8] http://llvm.org/docs/Passes.html#循环-simplify canonicalize naturalloops，2018年。查阅日期：2018年5月。

[9] CHEN，P.，XU，J.，HU，Z.，XING，X.，ZHU，M.，MAO，B.，和LIU，P.你看到的不是你得到的！用变色龙阻止及时的rop。2017年第47届IEEE/IFIP可靠系统和网络国际会议（DSN）（2017年6月），第451-462页。[10] http://cve.mitre.org, 2018. 查阅日期：2018年5月。

[11] COSTA，M.，CASTRO，M.，ZHOU，L.，ZHANG，L.，和PEINADO，M.Bouncer:通过阻止错误输入来保护软件。第二十一届ACM-SIGOPS操作系统原理研讨会论文集（美国纽约，2007），SOSP&apos;07，ACM，第117-130页。

[12] COWAN，C.，PU，C.，MAIER，D.，WALPOLE，J.，BAKKE，P.，BEATTIE，S.，GRIER，A.，WAGLE，P.，ZHANG，Q.，和HINTON，H.StackGuard:缓冲区溢出攻击的自动自适应检测和预防。进行中。第七届USENIX安全研讨会（1998年1月），第63-78页。

[13] CRISWELL，J.，dautenhan，N.，和ADVE，V.KCoFI：商品操作系统内核的完全控制流完整性。在

*2014年IEEE安全与隐私研讨会论文集*（华盛顿特区，美国，2014），SP&apos;14，IEEE计算机学会，第292-307页。

[14] 基于数据流的循环边界检测。最坏情况执行时间研讨会（2007年）。[15] http://www.cvedeails.com, 2018. 查阅日期：2018年5月。

[16] GAO，Q.，XIONG，Y.，MI，Y.，ZHANG，L.，YANG，W.，ZHOU，Z.，XIE，B.，和MEI，H.c程序的安全内存泄漏修复。第37届国际软件工程会议论文集第1卷（皮斯卡塔韦，新泽西州，美国，2015），ICSE’15，IEEE出版社，第459-470页。

[17] GOPAN，D.和REPS，T.《低层次图书馆分析和总结》。计算机辅助验证（柏林，海德堡，2007），W.Damm和H.Hermanns编辑，施普林格-柏林-海德堡，68-

81.

[18] HUANG，W.，HUANG，Z.，MIYANI，D.，和LIE，D.Lmp：硬件辅助的轻量级内存保护。第32届计算机安全应用年会论文集（美国纽约州纽约市，2016年），ACSAC&apos;16，ACM，第460-470页。

[19] HUANG，Z.，D&apos;ANGELO，M.，MIYANI，D.，和LIE，D.Talos:用安全解决方案中和漏洞以实现快速响应。2016年IEEE安全与隐私研讨会（SP）（2016年5月），第618-635页。

[20] KIM，D.，NAM，J.，SONG，J.，和KIM，S.自动补丁生成从人类编写的补丁中学习。《2013年软件工程国际会议论文集》（Piscataway，NJ，USA，2013），ICSE’13，IEEE出版社，第802–811页。

[21]KWON，Y.，SALTAFORMAGGIO，B.，KIM，I.L.，LEE，K.H.，ZHANG，X.，和XU，D.A2c:通过输入扰动的自毁漏洞执行。在NDSS&apos;17（2017）的会议记录中，互联网协会。

[22]LATTNER，C.，LENHARTH，A.，和ADVE，V.使堆克隆分析的上下文敏感点在现实世界中实用。第28届ACM SIGPLAN编程语言设计与实现会议论文集（美国纽约，2007），PLDI&apos;07，ACM，第278-289页。

[23]LE GOUES，C.，DEWEY-VOGT，M.，FORREST，S.，和WEIMER，W.《自动化程序修复的系统研究：修复105个错误中的55个，每个8美元》。2012年国际软件工程会议论文集（2012年6月），第3-13页。

[24]LIN，Z.，JIANG，X.，XU，D.，MAO，B.，和XIE，L.AutoPaG：使用源代码根本原因识别和修复实现软件补丁自动生成。第二届ACM信息、计算机和通信安全研讨会论文集（美国纽约，2007年），ASIACCS&apos;07，ACM，第329-340页。

[25]LONG，F.，GANESH，V.，CARBIN，M.，SIDIROGLOU，S.，和RINARD，M.自动输入校正。第34届软件工程国际会议论文集（Piscataway，NJ，USA，2012），ICSE&apos;12，IEEE出版社，第80-90页。

[26]LONG，F.和RINARD，M.《条件综合阶段程序修复》。在2015年第10届软件工程基础联席会议纪要（纽约，纽约，美国，2015）中，ESEC/FSE 2015，ACM，第166-178页。

[27]LONG，F.和RINARD，M.通过学习正确的代码自动生成补丁。第43届ACM SIGPLANSIGACT编程语言原理研讨会论文集（纽约，纽约，美国，2016），第16期，ACM，第298-312页。

[28]LONG，F.，SIDIROGLOU-DOUSKOS，S.，KIM，D.，和RINARD，M。

整数溢出错误的声音输入滤波器生成。第41届ACM SIGPLAN-SIGACT编程语言原理研讨会论文集（纽约，纽约，美国，2014），POPL&apos;14，ACM，第439-452页。

[29]MECHTAEV，S.，YI，J.和ROYCHOUDHURY，A.Directfix:寻找简单的程序修复。第37届软件工程国际会议论文集-第1卷（皮斯卡塔韦，新泽西州，美国，2015），ICSE’15，IEEE出版社，第448-458页。

[30]MECHTAEV，S.，YI，J.，和ROYCHOUDHURY，A.Angelix:通过符号分析的可伸缩多行程序补丁合成。第38届软件工程国际会议论文集（美国纽约州纽约市，2016），ICSE’16，ACM，第691-701页。

[31]MORRISETT，G.、TAN，G.、TASSAROTTI，J.、TRISTAN，J.-B.和GAN，E.RockSalt:x86更好、更快、更强的SFI。在

*2012年ACM SIGPLAN编程语言设计与实现会议论文集（PLDI）*（美国纽约州纽约市，2012年），PLDI&apos;12，ACM，第395-404页。

[32]NAGARAKATTE，S.，ZHAO，J.，MARTIN，M.M.K.和ZDANCEWIC，S.Softbound:C的高度兼容和完整的空间内存安全。ACM SIGPLAN编程语言设计和实现会议（PLDI）（2009），第245-258页。[33] http://nvd.nist.gov, 2018. 查阅日期：2018年5月。

[34]NGUYEN，H.D.T.，QI，D.，ROYCHOUDHURY，A.，和CHANDRA，S.Semfix:通过语义分析的程序修复。《2013年软件工程国际会议论文集》（Piscataway，NJ，USA，2013），ICSE’13，IEEE出版社，第772-781页。

[35] http://www.exploit-db.com, 2018. 查阅日期：2018年5月。

[36]PERKINS，J.H.，KIM，S.，LARSEN，S.，AMARASINGHE，S.，BACHRACH，J.，CARBIN，M.，PACHECO，C.，SHERWOOD，F.，SIDIROGLOU，S.，SULLIVAN，G.，WONG，W.-F.，ZIBIN，Y.，ERNST，

M.D.和RINARD，M.自动修补部署软件中的错误。ACM SIGOPS第22届操作系统原理研讨会论文集（2009年，美国纽约州纽约市），SOSP&apos;09，ACM，第87-102页。

[37] http://bugs.php.net, 2018. 查阅日期：2018年5月。

[38] http://www.php-security.org/MOPB/MOPB-41-2007.html, 2018. 查阅日期：2018年5月。

[39]戚，Y.，毛，X.，雷，Y.，戴，Z.，和王，C.自动程序修复随机搜索的强度。第36届软件工程国际会议论文集（美国纽约，2014），ICSE 2014，ACM，第254-265页。

[40]QI，Z.，LONG，F.，ACHOUR，S.，和RINARD，M.生成和验证补丁生成系统的补丁合理性和正确性分析。2015年软件测试与分析国际研讨会论文集（美国纽约州纽约市，2015年），ISSTA 2015，ACM，第24-36页。

[41] http://bugzilla.redhat.com, 2018. 查阅日期：2018年5月。

[42] http://www.openwall.com/lists/oss-security/2012/05/03/4, 2018.

[43]SHACHAM，H.，PAGE，M.，PFAFF，B.，GOH，E.-J.，MODADUGU，N.，和BONEH，D.地址空间随机化的有效性。第11届ACM计算机与通信安全会议论文集（2004年10月），第298-307页。

[44] http://www.mail-archive.com/sqlite-users@mailinglists.sqlite.org/, 2018.

[45] http://www.angelfire.com/sk/stackshield, 2018.

[46] http://github.com/jirislaby/LLVMSlicer, 2018.

[47]SÜSSKRAUT，M.和FETZER，C.《商用现货软件库的健壮性和安全强化》。在第37届IEEE/IFIP可靠系统和网络国际会议上，DSN 2007，2007年6月25-28日，英国爱丁堡，会议记录（2007），第61-71页。[48] http://llvm.org/, 2018.

[49] http://pax.grsecurity.net/, 2018.

[50]TIAN，D.，ZENG，Q.，WU，D.，LIU，P.，和HU，C.Kruiser:半同步非阻塞并发内核堆缓冲区溢出监视。在第19届年度网络和分布式系统安全研讨会上，NDSS 2012，圣地亚哥，加利福尼亚，美国，2012年2月5-8日（2012年）。

[51]TICE，C.、ROEDER，T.、COLLINGBOURNE，P.、CHECKOWAY，S.、ERLINGSSON，Ú.、LOZANO，L.和PIKE，G.在GCC和LLVM中加强前缘控制流完整性。第23届USENIX安全研讨会（USENIX安全14）（加利福尼亚州圣地亚哥，2014年8月），USENIX协会，第941-955页。

[52]WAHBE，R.，LUCCO，S.，ANDERSON，T.E.，和GRAHAM，S.L.高效的基于软件的故障隔离。ACM SIGOPS操作系统评论（1994），第27卷，第203-216页。

[53]WANG，H.J.，GUO，C.，SIMON，D.R.，和ZUGENMAIER，A.Shield：

漏洞驱动的网络过滤器，用于防止已知的漏洞攻击。2004年计算机通信应用、技术、体系结构和协议会议论文集（美国纽约州纽约市，2004年），SIGCOMM&apos;04，ACM，第193-204页。

[54]WEIMER，W.，FRY，Z.P.和FORREST，S.利用程序等价性进行自适应程序修复：模型和初步结果。第28届IEEE/ACM国际自动化软件工程会议论文集（Piscataway，NJ，USA，2013），ASE&apos;13，IEEE出版社，第356-366页。

[55]WEIMER，W.，NGUYEN，T.，LE GOUES，C.，和FORREST，S.使用遗传编程自动寻找补丁。第31届国际软件工程会议论文集（2009年，美国华盛顿特区），ICSE&apos;09，IEEE计算机学会，第364-374页。[56] http://sourceware.org/bugzilla/, 2018. 查阅日期：2018年5月。

[57]YEE，B.，SEHR，D.，DARDYK，G.，CHEN，J.B.，MUTH，R.，ORMANDY，T.，OKASAKA，S.，NARULA，N.，和FULLAGAR，N.《本机客户端：可移植的、不可信的x86本机代码的沙盒》。《安全与隐私》，2009年第30届IEEE研讨会（2009年），第79-93页。

[58]ZHANG，M.和SEKAR，R.《COTS二进制文件的控制流完整性》。作为第22届USENIX安全研讨会（USENIX Security 13）的一部分（华盛顿特区，2013年），USENIX，第337-352页。

# 附录

我们测量了表达式翻译能够将内存访问范围和缓冲区分配大小收敛到单个函数范围的频率，发现在85%的情况下它能够这样做。

我们使用表IV中所列的GNU Coreutils的11个程序来评估我们分析技术的适用性。Senx的访问范围分析中止的最常见原因是LLVM无法规范化循环。例如，解析字符串输入的循环迭代的次数取决于字符串的内容。这样的字符串不能通过访问范围分析进行符号分析。

要了解导致表达式翻译中止的原因，我们尝试将缓冲区大小和内存访问范围聚合起来，以便成功地分析并将结果制成表V中的表格。“访问范围”列列出了循环调用堆栈中表达式转换可以将内存访问范围转换为和“缓冲区范围”的函数的平均百分比将表达式转换可以将缓冲区分配大小转换为的缓冲区分配调用堆栈中函数的平均百分比制成表格。最后，“Converged”表示在所有循环中，有多少百分比的表达式可以找到一个

|  |  |  |  |
| --- | --- | --- | --- |
| 程序 | 类型 | SLOC公司 | LLVM位码 |
| 沙512sum | 数据校验和 | 581 | 135 KB |
| 公共关系 | 文本格式 | 1,723 | 194KB |
| 头 | 文本操作 | 761 | 109KB |
| 目录 | 目录列表 | 3,388 | 418KB |
| 外径 | 文件转储 | 1,368 | 237KB |
| 长征 | 目录列表 | 3,388 | 418KB |
| 比较基准64 | 数据编码 | 238 | 91KB |
| 厕所 | 文本处理 | 784 | 120 KB |
| 猫 | 文件连接 | 495 | 182KB |
| 分类 | 数据排序 | 3,251 | 433亿 |
| 打印F | 格式化和打印数据 | 694 | 198KB |
| 平均值 | 不适用 | 1,516 | 230 KB |

表四：适用性评估程序。

|  |  |  |  |
| --- | --- | --- | --- |
| 程序 | 访问范围 | 缓冲区范围 | 聚合 |
| 公共关系 | 100% | 10% | 100% |
| 头 | 100% | 25% | 100% |
| tr公司 | 86% | 36% | 100% |
| 外径 | 54% | 16% | 58% |
| 猫 | 100% | 33% | 100% |
| 目录 | 71% | 14% | 57% |
| 长征 | 42% | 33% | 34% |
| 比较基准64 | 100% | 33% | 100% |
| MD5总和 | 100% | 33% | 100% |
| 沙512sum | 97% | 80% | 97% |
| 分类 | 91% | 10% | 90% |
| 平均。 | 85% | 29% | 85% |

表五：表达式翻译的收敛性。

用于放置修补程序的通用函数范围。如我们所见，缓冲区分配大小似乎经常采用在调用堆栈中计算得相当接近分配点的参数，而这些值在调用链的高层不可用，因此限制了许多情况下可以收敛到的函数范围。