计算机科学
COMPUTER SCIENCE

ROP 漏洞利用脚本的语义还原和自动化移植方法

施瑞恒，朱云聪，赵易如，赵磊

引用本文

施瑞恒，朱云聪，赵易如，赵磊. ROP 漏洞利用脚本的语义还原和自动化移植方法[J]. 计算机科学, 2022, 49(11): 49-54.

SHI Rui-heng, ZHU Yun-cong, ZHAO Yi-ru, ZHAO Lei. Semantic Restoration and Automatic Transplant for ROP Exploit Script[J]. Computer Science, 2022, 49(11): 49-54.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210900230

ROP 漏洞利用脚本的语义还原和自动化移植方法

施瑞恒　朱云聪　赵易如　赵　磊
空天信息安全与可信计算教育部重点实验室(武汉大学国家网络安全学院)　武汉 430072
(ruihengshi@whu.edu.cn)

摘　要　漏洞利用脚本在安全研究中有着极为重要的作用,安全研究人员需要研究漏洞利用脚本触发以及利用漏洞的方式,来对漏洞程序进行有效的防护。然而,从网络中获取的大量漏洞利用脚本的通用性和适配性都很差,局限于特定的操作系统及环境,会因运行环境的改变而失效。这个问题在基于 ROP 的漏洞利用脚本中尤为普遍,使得 ROP 漏洞利用脚本的移植利用分析变得非常困难,需要依赖于大量的人工辅助与专家经验。针对 ROP 漏洞利用脚本的移植利用难题,提出了 ROPTrans 系统,通过 ROP 漏洞利用脚本的语义识别,定位与运行环境相关的关键语义及其变量,随后自动化适配环境,生成目标环境下的 ROP 漏洞利用脚本,以实现 ROP 脚本的自动化移植。实验结果表明,ROPTrans 的成功率可以到达 80%,验证了该方法的有效性。

关键词:漏洞利用;控制流劫持;ROP;移植
中图法分类号　TP399

Semantic Restoration and Automatic Transplant for ROP Exploit Script

SHI Rui-heng,ZHU Yun-cong,ZHAO Yi-ru and ZHAO Lei
Key Laboratory of Aerospace Information Security and Trusted Computing,Ministry of Education,School of Cyber Science and Engineering,Wuhan University,Wuhan 430072,China

Abstract　Exploit script plays an important role in security research. Security researchers need to study how the exploit script trigger and exploit the vulnerability,so as to effectively protect the vulnerable program. However,many exploit scripts obtained from network have poor generality and adaptability. They are limited to specific operating system and execution environment,and the change of environment will lead to execution failure. This problem is particular common in exploit scripts based on return-orinted programming(ROP),makes the transplanting and exploit analysis of ROP scripts are difficult and rely on manual assistance and expert knowledge. To solve this problem,we propose ROPTrans system,which locates key semantics and its variables related to the running environment through analysing the semantic of ROP script,and then automatically generates ROP script adapted to the target environment,so as to achieve the target of transplanting ROP scripts automatically. Experimental results show that the success rate of ROPTrans can reach up to 80%,which verifies the effectiveness of our method.

Keywords　Exploit script,Control flow hijack,Return-orinted Programming,Transplanting

1　引言

漏洞利用脚本(译自英文单词 Exploit,以下简称 EXP 脚本)是利用软硬件中的某些漏洞来得到计算机控制权的程序。通常情况下,EXP 脚本至少包含两部分的内容:触发漏洞并能够劫持程序控制流或数据流的非法输入,以及执行恶意功能的机器指令(通常也被称为 Shellcode[1])。

EXP 脚本是漏洞存在性以及危害性的最直观体现。EXP 脚本通常对应于可以被利用的高危漏洞,因此,EXP 脚本的安全分析对网络空间安全防御有着重要意义。安全研究人员能够通过研究 EXP 脚本来定位漏洞根源,诊断漏洞条件,认知 EXP 脚本的攻击流程。这些漏洞及漏洞利用的技术细节可进一步支撑 IDS 防护规则的升级和程序热补丁的开发等。

安全研究人员通常从网络中获取 EXP 脚本并试图分析 EXP 脚本触发和利用漏洞的方式,经常遇到 EXP 脚本失效的问题,表现为从网络获取的 EXP 脚本无法执行其中嵌入的机器指令,仅能够造成漏洞程序的崩溃,甚至无法触发漏洞。因此,EXP 脚本分析工作中经常会出现移植利用问题。

EXP 脚本的移植利用问题在使用了 Return-oriented programming[2-3](ROP)漏洞利用方法的 EXP 脚本(简称 ROP 脚本)中尤为突出。由于当前主流的操作系统都部署了堆栈不可执行的防御方法,如 W⊕X,NX,DEP 等,这使得典型的

到稿日期:2021-09-27　返修日期:2022-03-26
基金项目:国家自然科学基金(62172305,U1836112);湖北省重点研发计划(2020BAA003)
This work was supported by the National Natural Science Foundation of China(62172305,U1836112) and Key-Area Research and Development Program of Hubei Province(2020BAA003).
通信作者:赵磊(leizhao@whu.edu.cn)

EXP 脚本普遍使用了 ROP 技术，因此 ROP 脚本在 EXP 脚本中占有重要的地位和较大的比例。在 ROP 脚本中，劫持控制流后直接执行的不再是 Shellcode，而是指向特定指令的指针或内存地址。ROP 脚本在漏洞程序中的相应内存位置（如堆栈）布置精心编排的指令内存地址和立即数（这些指针和立即数被称为 Gadgets[4]），串联起多个指向指令片段的指针，以绕过计算机的安全保护，实施代码重用攻击。

ROP 脚本出现移植问题的主要原因是 ROP 脚本中通常嵌入了与特定的操作系统环境密切相关的指令内存地址。当安全研究人员在自己搭建的环境中复现漏洞时，由于操作系统、共享库版本等因素的改变，导致漏洞程序的执行上下文环境发生了变化。这种变化破坏了 ROP 脚本中嵌入的指令内存地址与指令的映射关系，即 ROP 中的指针指向了其他未知的指令序列。因此，ROP 脚本无法准确定位目标指令，进而导致 ROP 脚本失效。

针对 EXP 脚本的移植利用，Bao 等[5]提出了 ShellSwap 系统。然而，该工作主要集中在 EXP 脚本中包含的 Shellcode 移植上。ShellSwap 通过替换现有 EXP 脚本的 Shellcode，实现其功能的拓展。但是替换 Shellcode 仅能够拓展 ROP 脚本的功能，无法解决 ROP 脚本的移植问题。

ROP 脚本的移植利用主要面临 3 个方面的难题：

（1）理清结构难。ROP 脚本输入给漏洞程序的载荷通常包括漏洞触发载荷、堆栈溢出载荷、Gadgets、shellcode 等，漏洞程序对其先后顺序、交互轮次有着严格的要求。如何理清载荷的组织结构，是移植 ROP 脚本首先需要解决的问题。

（2）分析语义难。ROP 脚本中 Gadgets 的表现形式是二进制串，其中包含指令内存地址、函数参数、字符串等内容，其形式抽象且成分复杂。如何从抽象复杂的二进制串中还原出 Gadgets 的功能和语义是移植 ROP 脚本的主要难题。

（3）获取约束难。漏洞程序通常对 ROP 脚本的载荷存在着一定的约束，如限制载荷中出现的字符、限制载荷的长度等。这种约束体现在漏洞程序的代码逻辑中，无法从 ROP 脚本中直接获取。如何获取漏洞程序对载荷的约束信息也是移植 ROP 脚本必须解决的问题。

本文提出了一个自动化的 ROP 脚本移植系统——ROPTrans。ROPTrans 识别并分析 ROP 脚本中与环境相关的语义及变量，使之自动化地适配当前的运行环境，生成目标环境下的 ROP 脚本，从而解决 ROP 脚本的移植问题。为了理清载荷的结构，ROPTrans 依据 ROP 脚本的抽象语法树，从输出载荷的函数处反向解析，从而分析出载荷的结构。此外，ROPTrans 使用基于自然语言处理的 Gadgets 语义还原方法，来解决 Gadgets 语义分析难题。该方法基于自然语言处理提取 ROP 脚本的语义特征，并结合专家知识，还原 Gadgets 的功能和语义。最后，为了获取漏洞程序对载荷的约束信息，ROPTrans 使用动态污点分析技术[6-7]，多次比对分析 ROP 脚本中的载荷和实际注入漏洞程序的载荷，从而获取该约束信息。

本文研究工作的主要贡献如下：

（1）针对 ROP 脚本的漏洞利用分析需求，分析并指出了当前普遍存在的 ROP 脚本移植利用问题。同时，通过实证实验，分析归纳了 ROP 脚本的结构和语义特性，指出了 ROP 移植利用的难点。

（2）提出了基于自然语言处理的 Gadgets 语义还原方法，该方法可以准确地还原 Gadgets 的语义信息，为 ROP 脚本的移植创造条件。

（3）设计了 ROPTrans 系统，它是可以自动化解决 ROP 脚本移植问题的端到端的系统，成功移植了 11 个 ROP 脚本中的 8 个。

2 研究思路与挑战

2.1 主要思路

EXP 脚本通常可以被划分为 3 个部分：漏洞触发、漏洞利用和 Shellcode。其中，漏洞触发部分用于到达程序的漏洞点；漏洞利用部分用于绕过计算机的安全保护，创造 Shellcode 的执行条件，最终由 Shellcode 实现 EXP 脚本的恶意功能。

Gadgets 位于 ROP 脚本的漏洞触发部分，通常用于绕过堆栈不可执行防御，为 Shellcode 的执行创造条件。因此，Gadgets 往往存在着特定的功能。如果我们能够分析出 Gadgets 的功能，就可以还原 Gadgets 的语义信息，并根据该语义信息生成适配当前运行环境上下文的 Gadgets，进而解决 ROP 脚本的移植问题。

我们对 80 个 ROP 脚本进行了统计分析，并从中发现了获得 Gadgets 语义信息的可能性。79 个 ROP 脚本都能够通过阅读 Gadgets 的注释来理解其功能。也就是说，绝大部分 ROP 脚本的注释中隐含着 Gadgets 的语义信息。因此，存在通过某种方式从注释中提取 Gadgets 的语义信息，进而解决 ROP 脚本移植问题的可能性。

2.2 研究挑战

虽然 ROP 脚本的注释中隐含着 Gadgets 的语义信息，但是注释本身存在的问题加大了获取语义信息的难度。据观察，Gadgets 的注释主要有以下 3 个方面的问题：

（1）注释不完备。在部分 ROP 脚本中，存在注释不完整、不规范的情况。注释不完整指注释中 Gadgets 对应的指令序列不完整，有省略或者缺失。注释不规范指对 Gadgets 的注释中仅有描述性的自然语言，而没有对应的指令序列。

（2）注释逻辑存在缺陷。在部分 ROP 脚本中，注释的语义逻辑本身就存在缺陷。即使漏洞程序按照 Gadgets 注释的指令序列执行，也无法实现预期的目标。

（3）注释存在其他隐式缺陷。隐式缺陷是静态分析时无法被发现的一种缺陷，表现为 Gadgets 注释的语义不符合漏洞程序的约束。

因此，要想从 ROP 脚本的注释中提取 Gadgets 的语义信息，需要解决注释本身存在的诸多问题。

为了帮助读者更好地理解 ROP 脚本的移植问题以及从注释中获取 Gadgets 语义信息的挑战性，我们给出了一个示例。如图 1 所示，这是一个缓冲区溢出漏洞的 ROP 脚本的 Gadgets 及其注释，并且为了方便阅读进行了一些简化。该 ROP 脚本在劫持程序的控制流后通过 Gadgets 调用 execve 函数执行/bin/sh 程序。可以看到，在该 ROP 脚本中，嵌入的

指令内存地址和特定的指令序列建立了映射关系。例如，在图 1 中第 6 行，地址 0x0807b744 和指令序列“pop eax；ret；”建立了映射关系。当我们在自己环境下编译运行该漏洞程序时，地址 0x0807b744 对应的指令序列为“mov DWORD PTR [eax＋0x1c]，0x6c20646；mov DWORD PTR [eax＋0x20]，0x20747369；…”。显然，该 ROP 脚本在劫持漏洞程序的控制流后，会去执行一个未知的指令序列，一旦进行了非法操作，就会立即造成漏洞程序崩溃。由此可见，ROP 脚本的移植问题会严重干扰安全研究人员调试与分析漏洞，是一个亟待解决的问题。

```
1  # rop execve ( bin/sh )
2  rop = "A"*1017 # junk
3  rop += pack('<I', 0x080e9101) # pop edx; pop ebx; pop
esi; pop edi; pop ebp; ret;
4  rop += pack('<I', 0x0811abe0) # @ .data
5  …… # padding
6  rop += pack('<I', 0x0807b744) # pop eax ; ret;
7  rop += '/bin'
8  rop += pack('<I', 0x0810ae08) # mov dword ptr [edx],
eax; pop ebx ;pop ebp ; ret;
9  …… # padding
10 ……
11 rop += pack('<I', 0x080b4970) # xor eax, eax; pop esi;
pop ebp; ret
12 …… # padding //向内存写入 “/bin//sh” //已使用112
字节Gadgets
13 rop += pack('<I', 0x0810ae08) # mov dword ptr [edx],
eax; pop ebx; pop ebp; ret;
14 ……# padding
15 ……
16 rop += pack('<I', 0x080b4970) # xor eax, eax; pop esi;
pop ebp; ret;
17 ……# padding
18 ……# inc eax 直到eax=11; ret; //设置各寄存器值 //已
使用240字节Gadgets
19 rop += pack('<I', 0x080c861f) # int 0x80; //调用execve
函数 //已使用244字节Gadgets
```

图 1　Gadgets 及其注释示例(EDB ID：44426)

Fig. 1　Example of Gadgets and its comments(EDB ID：44426)

为了证明从注释中获取 Gadgets 语义信息的挑战性，我们考虑一种直接的方法，即完全遵循 Gadgets 注释中提供的语义信息来生成适配当前运行环境上下文的 Gadgets。

如图 1 所示，该 ROP 脚本首先使用 112 字节的 Gadgets 向内存中写入字符串“/bin//sh”，随后使用 128 字节的 Gadgets 设置好各寄存器的值，最后使用 4 字节的 Gadgets 完成系统调用，总计 244 字节。因此当我们完全遵循 Gadgets 注释中的语义信息生成新的 Gadgets 时，其长度必然也是 244 字节。然而，经动态调试后，我们发现该漏洞程序最大可接收 49 字节 Gadgets，这就意味着新生成 Gadgets 的第 50 到 244 字节都是无效的。因此上述方法难以应对 ROP 脚本移植过程中的挑战。

出现这种情况的原因是上文提及的注释存在隐式缺陷。完全遵循注释的语义无法生成符合漏洞程序约束的 Gadgets，自然也就无法解决 ROP 脚本的移植问题。

除了隐式缺陷，注释的逻辑缺陷也很常见。我们随机抽取了 6 个 ROP 脚本，对其 Gadgets 的注释的逻辑进行分析后发现，3 个脚本的注释都存在逻辑缺陷。此外，ROP 脚本中的注释编写没有一个统一的标准，因此注释缺失、注释不规范的情况并不少见。

接下来我们将具体分析 ROPTrans 如何应对上述的诸多挑战。

3　ROPTrans 设计概述

3.1　ROPTrans 概述

ROPTrans 是一个可以自动化解决 ROP 脚本移植问题的端到端的系统。如图 2 所示，ROPTrans 将存在移植问题的 ROP 脚本和对应的漏洞程序作为输入，最终输出一个适配当前运行环境上下文的 ROP 脚本。图 2 中的关键步骤如下所示。

(1)Gadgets 定位：将原 ROP 脚本和漏洞程序作为 Gadgets 定位模块的输入，将动态污点分析的结果以及 ROP 脚本中载荷的结构和内容相结合，分析出 Gadgets 的位置信息。

(2)Gadgets 语义还原：将 ROP 脚本、禁忌规则和 Gadgets 位置作为输入，其中初始的禁忌规则为空的集合。该模块首先使用基于自然语言处理的 Gadgets 语义还原方法从注释中提取出能概况 Gadgets 语义的关键词，并根据关键词集合还原出 Gadgets 的语义；然后结合该语义信息和禁忌规则生成适配当前运行环境上下文的 Gadgets；最后根据 Gadgets 的位置信息，用新生成的 Gadgets 替换原 ROP 脚本中的 Gadgets，生成待检查的临时 ROP 脚本。

(3)动态检查：该模块将临时的 ROP 脚本和漏洞程序交互，如果漏洞利用成功，则将该临时的 ROP 脚本输出为最终的新 ROP 脚本。如果漏洞利用失败，则使用动态污点分析技术，得到漏洞程序对 Gadgets 的约束并向禁忌规则集合中添加相应规则，来约束 Gadgets 的生成。

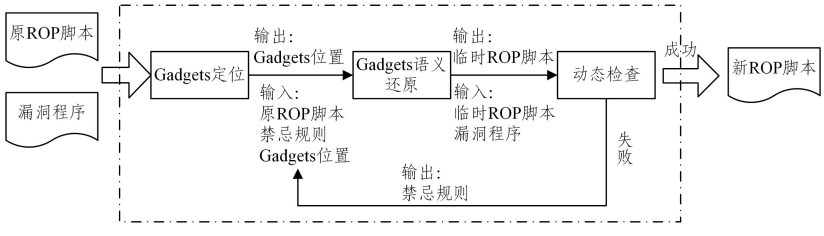

图 2　ROPTrans 系统架构

Fig. 2　ROPTrans system architecture

3.2　示例

本节将演示 ROPTrans 如何解决图 1 中 ROP 脚本的移植问题。

ROPTrans 首先将 ROP 脚本标记污点和漏洞程序进行交互，记录指令指针寄存器被 ROP 脚本中的载荷直接覆盖时寄存器和栈的状态。随后，ROPTrans 将 ROP 脚本完整的载荷与进入到漏洞程序有效区域的载荷进行比对，得知载荷的第 975 到 978 字节覆盖了指令指针寄存器。结合 ROP 脚本中对载荷结构的定义，ROPTrans 得知载荷的第 975 字节及以后的内容都是 Gadgets。至此，ROPTrans 成功获得了

Gadgets 的位置信息。

在 Gadgets 语义还原模块中，ROPTrans 根据注释中的关键词“int 0x80”“execve”“binsh”，还原出 Gadgets 的高层语义，通过系统调用，调用 execve 函数执行/bin/sh 程序。然后，ROPTrans 根据该语义信息生成 Gadgets，并根据 Gadgets 的位置信息，用新 Gadgets 替换原 Gadgets，生成临时的 ROP 脚本。

最后 ROPTrans 进行动态检查，将新生成的 ROP 脚本标记污点和漏洞程序交互，发现漏洞程序对 Gadgets 有如下约束：长度小于或等于 49 字节、不能出现\x00 和\x20。ROPTrans 将这 3 个约束添加到禁忌规则中，重新生成临时的 ROP 脚本。该 ROP 脚本通过了动态检查，作为最终的 ROP 脚本。至此，ROPTrans 成功解决了图 1 中 ROP 脚本的移植问题。

4 Gadgets 语义还原方法

本节将重点介绍基于自然语言处理的 Gadgets 语义还原方法，阐述该方法如何解决 Gadgets 注释中存在的逻辑缺陷、注释不完备问题，准确地还原出 Gadgets 的高层语义信息，并根据漏洞程序的约束灵活地生成适配当前运行环境上下文的 Gadgets。

如图 3 所示，基于自然语言处理的 Gadgets 语义还原方法首先根据 Gadgets 的注释生成解析树，并根据解析树中的关键词还原出 Gadgets 的语义信息，然后依据该语义信息匹配 Gadgets 模板并结合禁忌规则生成新的 Gadgets。

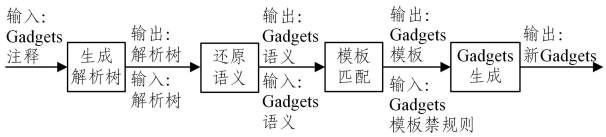

图 3 基于自然语言处理的 Gadgets 语义还原方法

Fig. 3 Gadgets semantic restoration based on nature language processing

生成解析树是基于自然语言处理的 Gadgets 语义还原方法的第一步。解析树是有序且有根节点的树，用于表示句子的语法结构[8]。解析树的根节点是树开始的节点，其内部节点被标记为非终止节点，代表相关的单词序列或短语的句法特征。解析树的叶子节点被标记为终止节点，代表句子中的每一个单词的词性[9]。在本文的工作中，ROPTrans 将每一行注释视为一个句子并生成相应的解析树。图 4 给出了 EDB ID 为 44426 的 ROP 脚本中的某行注释产生的解析树。树的根节点“PRN”表示这行注释的性质是附加说明类的短语；“NP”表示“# rop execve”和“bin/sh”是名词短语；“JJ”表示“# rop”是形容词；“NN”表示“execve”是名词；“NNP”表示“bin/sh”是专有名词短语[10]。从解析树中得到了注释中每一个词的词性，从而可以帮助 ROPTrans 筛选能够代表 Gadgets 语义的关键词。

还原语义的过程是基于自然语言处理的 Gadgets 语义还原方法的关键。首先根据解析树中的词性，排除掉注释中无关的单词，然后在其余单词中寻找能够代表 Gadgets 语义的关键词。通常来说，关键词可以是函数名、重要字符串和函数参数。ROPTrans 如果识别出某个单词是函数名或者是函数的参数名，就将该函数名视为关键词；如果某个单词和预定义的字符串相匹配，也会将该单词加入关键词集合中。如图 4 所示，从这段注释的解析树中可以识别出的关键词是“execve”和“bin/sh”。在对所有 Gadgets 注释的解析树进行关键词识别后，ROPTrans 得到一个关键词集合，集合中的关键词表示 Gadgets 的功能。

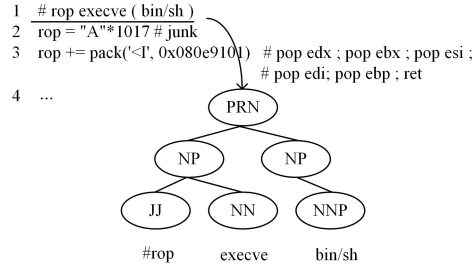

图 4 解析树生成示意图(EDB ID:44426)

Fig. 4 Schematic diagram of parse tree generation(EDB ID:44426)

随后，ROPTrans 根据关键词匹配预定义的 Gadgets 模板。如图 5 所示，模板中定义了关键词、寄存器状态、指令序列和触发器。模板中的关键词用于匹配注释中提取出的关键词；寄存器状态表示寄存器需要达到的目标状态；指令序列表示达到寄存器目标状态需要执行的指令序列；触发器用于在所有寄存器状态就绪后实现模板定义的目标。在 Gadgets 的模板中并没有定义具体的 Gadgets，而是给出了生成 Gadgets 的方向和目标，这赋予了生成 Gadgets 的灵活性。

图 5 Gadgets 模板示意图

Fig. 5 Schematic diagram of Gadgets template

最后，ROPTrans 结合禁忌规则和 Gadgets 模板生成适配当前运行环境上下文的 Gadgets。禁忌规则定义了 Gadgets 生成的约束，包括 Gadgets 长度约束、截断漏洞程序读取攻击载荷的坏字符等。新生成的 Gadgets 将用于替换原 ROP 脚本中的 Gadgets，最终生成临时的 ROP 脚本。

基于自然语言处理的 Gadgets 语义还原方法将自然语言处理技术和专家经验相结合，不仅能准确地还原 Gadgets 的高层语义信息，还有效地解决了 Gadgets 注释中存在的不完备、逻辑缺陷问题。此外，基于模板和禁忌规则的 Gadgets 生成在保证 Gadgets 功能完备的同时还具有很高的灵活性，能够生成满足漏洞程序约束且功能完备的 Gadgets。

5 动态污点分析

动态污点分析技术指对非信任来源的数据进行标记，追踪其在程序执行中的传播过程，以获取关键位置与输入数据关联信息的分析方法[11]。ROPTrans 系统将漏洞程序和 ROP 脚本作为动态污点分析器的输入，在指令指针寄存器被标记污点的数据覆盖时，记录此时栈和寄存器的状态。根据栈和寄存器的状态信息以及 ROP 脚本中载荷的结构和内容，分析出漏洞触发部分的载荷和 Gadgets 的位置信息。漏洞触发部分的载荷在新的 ROP 脚本中得以保留，而 Gadgets 的位置信息被用于替换旧的 Gadgets。

此外，动态污点分析技术也被用于 ROPTrans 系统的动态检查模块，以分析漏洞程序对 Gadgets 的约束。在生成第一个临时 ROP 脚本时，禁忌规则是一个空集合，因此生成的临时 ROP 脚本可能不符合漏洞程序的约束。ROPTrans 使用动态污点分析技术检查 ROP 脚本的载荷是否被完整地注入到漏洞程序的有效区域，如果发现载荷不完整的情况，则认为临时 ROP 脚本无效并分析漏洞程序约束。ROPTrans 将分析出的约束信息添加到禁忌规则中，重新生成临时的 ROP 脚本。如此迭代，直到 ROP 脚本能够完整地实现漏洞利用。

动态污点分析技术用于保留 ROP 脚本中漏洞触发部分的载荷、确定 Gadgets 的位置和分析漏洞程序约束，对 ROPTrans 系统有着重要作用。接下来，我们将通过实验来分析 ROPTrans 的性能，评估 Gadgets 定位的效果和基于自然语言处理的 Gadgets 语义还原方法的有效性。

6 实验分析

本节在 11 个 ROP 脚本中评估 ROPTrans 系统的有效性。实验结果表明，基于自然语言处理的 Gadgets 语义还原方法可以成功还原出大部分 ROP 脚本 Gadgets 的语义，说明 ROPTrans 可以解决大部分 ROP 脚本的移植问题。

6.1 数据集

本文评估 ROPTrans 系统有效性的数据集源于 EXPLOIT DATABASE。如表 1 所列，ROP 脚本利用的漏洞程序所在的平台包括 Linux 和 Windows；ROP 脚本绕过的系统安全机制包括 DEP/NX 和 ASLR；ROP 脚本涉及的漏洞类型主要是栈缓冲区溢出漏洞。

表 1 数据集信息

Table 1 Dataset information

EDB ID	漏洞类型	安全机制绕过	操作系统
13768	栈缓冲区溢出	DEP	Windows
13907	栈缓冲区溢出	DEP	Windows
17974	整数溢出	DEP	Windows
24943	栈缓冲区溢出	DEP	Windows
29920	栈缓冲区溢出	NX+ASLR	Linux
44073	栈缓冲区溢出	NX	Linux
44331	栈缓冲区溢出	NX	Linux
44426	栈缓冲区溢出	NX	Linux
45288	栈缓冲区溢出	NX	Linux
46997	栈缓冲区溢出	NX	Linux
48840	栈缓冲区溢出	DEP+ASLR	Windows

6.2 实验结果

如表 2 所列，ROPTrans 成功地移植了 11 个 ROP 脚本中的 8 个，接下来我们将全面分析 ROPTrans 的表现并探究其背后的原因。

表 2 实验结果

Table 2 Experiment results

EDB ID	Gadgets 注释存在的问题	Gadgets 定位结果	ROPTrans 移植结果
13768	逻辑缺陷	√	√
13907	逻辑缺陷	√	√
17974	—	√	√
24943	逻辑缺陷	√	√
29920	注释不完整	√	×
44073	注释不规范	×	×
44331	—	√	√
44426	隐式缺陷	√	√
45288	隐式缺陷	√	√
46997	注释不规范	√	√
48840	—	√	×

在成功移植的 8 个 ROP 脚本中，6 个 ROP 脚本的注释存在问题，其中 3 个存在逻辑缺陷，2 个存在隐式缺陷，1 个注释不规范。ROPTrans 使用基于自然语言处理的 Gadgets 语义还原方法，根据从 ROP 脚本注释中提取出的关键词，还原出 ROP 脚本中 Gadgets 的语义，屏蔽了 Gadgets 注释中存在的逻辑缺陷和注释不规范问题。此外，ROPTrans 使用动态检查模块分析漏洞程序的约束，克服了注释中存在的隐式缺陷，最终成功地移植了 ROP 脚本。

以 EDB ID 为 13907 的 ROP 脚本为例。如图 6 所示，该 ROP 脚本将 eax 寄存器的值设置为 0x0 后立即执行了“call eax”指令。该指令的执行会立刻造成漏洞程序崩溃。显然，该 ROP 脚本 Gadgets 的注释存在逻辑缺陷。ROPTrans 使用基于自然语言处理的 Gadgets 语义还原方法来克服这段注释中存在的逻辑缺陷。ROPTrans 根据第 10 行注释中的关键词“WriteProcessMemory”，还原出这段 Gadgets 的语义是调用 WriteProcessMemory 函数将 Shellcode 写入一块可执行的内存空间。通过这种方法 ROPTrans 屏蔽了 Gadgets 注释中存在的逻辑缺陷，成功地移植了该 ROP 脚本。

```
1  rop = "\x41" * 540 # Crash
2  rop += "\x09\x12\x0e\x07" # POP EDI # POP ESI #
POP EBP # XOR EAX,EAX # POP EBX # RETN
3  ..... # Junk
4  rop += "\x03\x85\x09\x07" # EAX CALL
5  ..... # Junk
6  .....#Omit trival content
7  rop += "\xcf\x22\x80\x7c" # dest address in
WriteProcessMemory()
8  rop += "\xcf\xc9\x0e\x07" # ADD EBX, EAX # XOR
AL,AL # RETN
9  rop += "\x5e\x89\x09\x07" # POP EAX # POP ESI #
RETN
10 rop += "\x13\x22\x80\x7c" # WriteProcessMemory
11 rop += "\xff\xff\xff\xff" # HProcess HANDLE (-1)
12 rop += "\x65\x08\x59\x78" # PUSHAD # RETN
```

图 6 Gadgets 及其注释示例(EDB ID:13907)

Fig. 6 Example of Gadgets and its comment(EDB ID:13907)

在 EDB ID 为 44426 的 ROP 脚本中，Gadgets 的注释存在隐式缺陷。在该 ROP 脚本中，Gadgets 的长度达到了 244 字节，而漏洞程序实际可以接收 Gadgets 的最大长度为 49 字节。ROPTrans 通过动态检查模块对漏洞程序进行约束分析，获得了漏洞程序对 Gadgets 长度的约束和坏字符的信息。ROPTrans 将漏洞程序对 Gadgets 的约束作用于新 Gadgets

的生成，最终成功解决了该 ROP 脚本的移植问题，克服了 Gadgets 注释中存在的隐式缺陷。

对于 ROPTrans 移植失败的 3 个 ROP 脚本，它们失败的原因并不相同。ROPTrans 在移植 EDB ID 为 29920 的 ROP 脚本时遭遇了失败，原因在于 ROPTrans 没有定义关键词“call * %gs:[0x10]”对应的模板。“call * %gs:[0x10]”实际上调用了 32 位 Linux 系统调用。ROPTrans 可以妥善地处理 Gadgets 模板缺失的问题，其只需要在配置文件中添加该关键词对应的 Gadgets 模板，就可以解决这类 ROP 脚本的移植问题。

EDB ID 为 44073 的 ROP 脚本将利用漏洞的过程分为两个阶段。第一阶段的 Gadgets 通过触发程序漏洞来泄露漏洞程序的信息；第二阶段的 Gadgets 用于向漏洞程序中写入需要执行的命令、字符串和 Gadgets，进行漏洞利用。在移植 EDB ID 为 44073 的 ROP 脚本过程中，ROPTrans 只能定位第一阶段载荷中 Gadgets 的位置，并且会将两个阶段的 Gadgets 注释视为一个整体，从中提取关键词。ROPTrans 在设计上没有考虑多次触发程序漏洞的 ROP 脚本，因此 ROPTrans 不仅无法准确地定位这类 ROP 脚本的 Gadgets，在还原 Gadgets 语义的过程中也会出现偏差。据统计，80 个从 EXPLOIT DATABASE 获取的 ROP 脚本中有 8 个需要多次触发的程序漏洞。

EDB ID 为 48840 的 ROP 脚本使用 Gadgets 向漏洞程序所在的操作系统添加了一个新的用户，并将该用户添加到管理员组中。ROPTrans 在移植该 ROP 脚本时遭遇失败，原因在于该 ROP 脚本 Gadgets 的功能复杂，难以根据注释中某几个关键词推断出 Gadgets 原本的功能。基于自然语言处理的 Gadgets 语义还原方法可以准确地还原出使用模式较为固定的 Gadgets 语义。但是对于功能复杂且难以概括的 Gadgets，现有的方法无法准确地还原出 Gadgets 的语义。我们对基于自然语言处理的 Gadgets 语义还原方法进行了评估，发现它能准确地还原出 80 个 ROP 脚本中的 67 个 ROP 脚本的 Gadgets 语义信息。

本节对 ROPTrans 和基于自然语言处理的 Gadgets 语义还原方法进行了较为全面的评估。基于自然语言处理的 Gadgets 语义还原方法可以准确地还原出大部分 ROP 脚本 Gadgets 的语义，ROPTrans 也可以解决大部分 ROP 脚本的移植问题。但是，它们都存在着一定的局限性。ROPTrans 系统从设计上没有考虑到多次触发程序漏洞的 ROP 脚本移植问题。面对功能复杂且难以概括的 Gadgets，基于自然语言处理的 Gadgets 语义还原方法也无法准确地还原出 Gadgets 的语义。这些问题我们将放到下一个阶段的工作中去解决。

结束语 本文指出了 ROP 脚本的移植问题，并解释了出现移植问题的原因。随后介绍了 ROPTrans 系统，它使用基于自然语言处理的 Gadgets 语义还原方法和动态污点分析技术，自动化解决 ROP 脚本的移植问题。最后，在数据集上全面地评估了 ROPTrans 的有效性，同时也指出了现有系统以及所提方法存在的局限性。

参 考 文 献

[1] ARCE I. The shellcode generation[J]. IEEE Security & Privacy, 2004, 2(5): 72-76.

[2] ROEMER R, BUCHANAN E, SHACHAM H, et al. Return-oriented programming: Systems, languages, and applications[J]. ACM Transactions on Information and System Security (TISSEC), 2012, 15(1): 1-34.

[3] VISHNYAKOV A V, NURMUKHAMETOV A R. Survey of Methods for Automated Code-Reuse Exploit Generation[J]. Programming and Computer Software, 2021, 47(4): 271-297.

[4] BUCHANAN E, ROEMER R, SHACHAM H, et al. When good instructions go bad: Generalizing return-oriented programming to RISC[C]// Proceedings of the 15th ACM Conference on Computer and Communications Security. 2008: 27-38.

[5] BAO T, WANG R, SHOSHITAISHVILI Y, et al. Your exploit is mine: Automatic Shellcode transplant for remote exploits[C]// 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017: 824-839.

[6] NEWSOME J, SONG D X. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software[C]// NDSS. 2005: 3-4.

[7] KANG M G, MCCAMANT S, POOSANKAM P, et al. Dta++: dynamic taint analysis with targeted control-flow propagation[C]// NDSS. 2011.

[8] CHARNIAK E. Tree-bank grammars[C]// Proceedings of the National Conference on Artificial Intelligence. 1996: 1031-1036.

[9] YOU W, ZONG P, CHEN K, et al. Semfuzz: Semantics-based automatic generation of proof-of-concept exploits[C]// Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 2139-2154.

[10] CHOWDHARY K R. Natural language processing[M]//Fundamentals of Artificial Intelligence. New Delhi: Springer, 2020: 603-649.

[11] LAM M S, MARTIN M, LIVSHITS B, et al. Securing web applications with static and dynamic information flow tracking[C]// Proceedings of the 2008 ACM SIGPLAN symposium on Partial Evaluation and Semantics-based Program Manipulation. 2008: 3-12.

SHI Rui-heng, born in 1997, postgraduate. His main research interests include automatic exploit generation and fuzzing.

ZHAO Lei, born in 1985, Ph.D, professor. His main research interests include software and system security, especially in security analysis of binary programs and automatic software vulnerability detection.

（责任编辑：杨雪敏）