基于符号执行的二进制代码漏洞发现

牛伟纳[1] 丁雪峰[2] 刘 智[1] 张小松[1]

(电子科技大学计算机科学与工程学院 成都 611731)[1] (四川大学信息管理中心 成都 610065)[2]

摘 要 软件漏洞是安全问题的根源之一,fuzzing(模糊测试)是目前漏洞发现的关键技术,但是它通过随机改变输入无法有效地构造出测试用例,也无法消除测试用例的冗余性。为了克服传统fuzzing测试的缺点、有效生成测试输入且无需分析输入格式,针对二进制程序设计并实现了基于符号执行的漏洞发现系统SEVE。将程序的输入符号化,利用动态插桩工具建立符号变量的传播关系;在分支语句处收集路径约束条件,最后用解析器求解之并将其作为新的测试用例。用mp3和pdf软件进行了实验,结果表明,该系统有效地提高了漏洞发现的效率与自动化程度。

关键词 漏洞,二进制程序,符号执行,插桩,路径约束

中图法分类号 TP311 **文献标识码** A

Vulnerability Finding Using Symbolic Execution on Binary Programs

NIU Wei-na[1] DING Xue-feng[2] LIU Zhi[1] ZHANG Xiao-song[1]

(Department of Computer Science & Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China)[1]

(Information Management Center, Sichuan University, Chengdu 610065, China)[2]

Abstract Software vulnerability is one main source of computer safety issues, and the key technology of vulnerabilities finding is fuzzing(fuzzy test) which is based on randomly changing the input, however, it cannot construct test cases effectively and eliminate the redundancy of test cases. In order to overcome the shortcomings of traditional fuzzing test, effectively generate test inputs and do not need to analyze the input format, we designed and implemented vulnerability found system(called SEVE) based on symbolic execution for binary program. SEVE makes the inputs symbolic and uses dynamic instrumentation tools to establish the propagation relationship of the symbolic variable, collect path constraints in branch statement, uses interpreter to solve these path constraints to obtain test cases. Experimental results which are based on mp3 and pdf software show that the system can improve the efficiency and the degree of automation of vulnerability discovery.

Keywords Vulnerability, Binary program, Symbolic execution, Instrumentation, Path constraint

1 引言

软件漏洞(又称软件脆弱性)[1]作为目前网络攻击的主要手段,给互联网和用户带来了严重威胁。蠕虫、DDOS等攻击的首要前提是渗透进入目标主机/服务器,而通常的办法是攻击软件漏洞,如Code Red利用了IIS服务器漏洞;SQL Slammer利用了SQL漏洞;Win32. conficker利用MS-067漏洞进行传播。只要能渗透进入目标机器并取得相应权限,黑客就能够进行各种形式的后续入侵攻击。因此软件安全是目前计算机安全的一个核心问题,但由于现代软件和网络的复杂性,发现或阻止软件漏洞十分困难。目前通常采用的防御手段是基于行为分析或特征码的入侵检测(如Snort)或入侵防御系统。行为分析通常部署在主机中,对安全行为进行采集并用数据挖掘[2]算法检测未知攻击,但缺点是误报率高且容易对目标机器产生极大性能消耗;特征码一般由杀毒软件或安全厂商采用人工提取,虽然较精确但提取耗时长,且特征码容易被变形攻击绕过。特征码从产生到分发到主机耗时较长,David Brumley经过观察从安全补丁产生到分发到所有合法IP地址的平均时间是24小时[3]。

另一方面,安全研究人员积极主动地发现软件存在的潜在漏洞并进行修补。目前软件漏洞发现技术主要分为静态和动态分析两类,它们又各自包含针对源码和二进制代码的分析。源码分析直接对C/C++等源代码进行扫描,但不适合COTS软件;二进制代码级别分析缺乏语义和类型信息,分析难度远大于源码分析。另一方面,静态分析提取源码或可执行程序的静态特征进行分析,如控制流图(CFG)、依赖图(Dependence graph)等,但缺乏类型及运行时信息且容易被攻击。针对上述问题,本文针对二进制程序,结合软件测试思想和动态插桩工具,提出并实现了基于符号执行的漏洞发现系统(Symbolic Execution-based Vulnerability Explorer, SEVE)。

到稿日期:2012-12-27 返修日期:2013-04-04 本文受四川省科技计划支撑项目(2012GZ0001),上海市科研计划项目(11511505300)资助。

牛伟纳(1990—),女,硕士生,主要研究方向为网络与软件安全,E-mail:niuweina1@126. com;丁雪峰(1974—),男,博士,高级工程师,主要研究方向为信息安全,E-mail:dingxf@scu. edu. cn(通信作者);刘 智(1985—),男,博士生,主要研究方向为网络与软件安全;张小松(1968—),男,博士,教授,主要研究方向为网络安全。

它首先符号化输入，利用插桩工具动态跟踪符号变量并收集路径约束；再分支判断语句建立路径约束并用求解器求解，将产生的解作为新测试用例。实验结果表明，相比传统 fuzzing，SEVE 能产生有效测试用例并发现软件漏洞。

2 传统漏洞发现技术

2.1 静态代码审计

2.1.1 源码审计

源码审计针对不安全语言如 C/C++，通过词法或语法分析检查源码中的缺陷。最简单的分析是对代码中危险库函数的使用(如 strcpy)，进一步检查循环和源码中的逻辑问题。但该方法误报率高，应用范围较小。

2.1.2 二进制代码审计

由于汇编语言的复杂性和多平台结构特性，分析难度较大。重点在于理解程序流程。

2.2 黑盒 fuzzing

黑盒 fuzzing[5] 又称为模糊测试，为目前软件漏洞发现的主流技术，属于半自动化动态测试技术。它通过人工分析测试输入，构造半有效的测试用例，然后对这些测试用例进行一定程度的变形产生新的用例。威斯康星麦迪逊分校的 Miller[6,7] 首先提出该技术，并对 Unix 和 Windwos 平台上的程序作了测试并发现了若干漏洞。SPIKE 为一款商业 fuzzing 工具，PEACH 为开源软件，它们主要针对网络协议和部分简单文件格式的漏洞发现。目前该技术的研究重点是设计高效算法产生新的用例。

模糊测试首先分析目标软件的输入格式，人工或采用工具构造输入中的字段和需要变形的部分。对于未知格式或协议，通过逆向工程分析难度较大，如 WORD 等复杂格式几乎无法通过逆向分析构造出，因此现在并未出现针对复杂格式的通用 fuzzing 工具。fuzzing 的缺陷在于盲目、无任何导向性地生成测试用例，绝大部分用例都无效，导致代码覆盖率极低。如 if($x==10$)，采用 fuzzing 触发执行路径的几率仅为 $1/2^{32}$，更不用说含有众多分支路径的程序。黑盒fuzzing的主要缺陷在于需要首先分析输入格式，并难以生成有效的测试用例遍历分支路径，且自动化程度低。

2.3 污点分析技术

污点分析[8] 类似编译器中的数据流分析，通过对不可信输入进行标记并跟踪其传播，若污点变量用作危险函数的参数调用则触发警告或停止运行。但它只能作为一种防御手段，并不能主动发现软件漏洞。

2.4 编译器检查

通过在编译器中加入安全检查机制检测漏洞。文献[9]检查关键字“canary”是否被覆盖。该方法由于不能用于二进制代码分析，因此应用范围较小。

3 符号执行算法

文献[10]首次在理论上提出符号执行(symbolic execution)[11]，它最初用于编译器、程序理解等领域。本文称传统输入(包含具体值)为具体输入，符号化后的输入称为符号输入，它们对应的程序执行为具体执行和符号执行，其关系如图 1 所示。Stanford 的 Dawson Engler 将符号执行应用于源码的 bug 发现[12]，并将其商业化为 Coverity 产品。符号执行的主要思想是将程序的输入符号化，并将这些符号变量作为输入进行执行，收集分支条件判断的谓词约束，最后用求解器得到新的测试输入。符号执行流程见算法 1。下面对算法 1 进行简单解释。

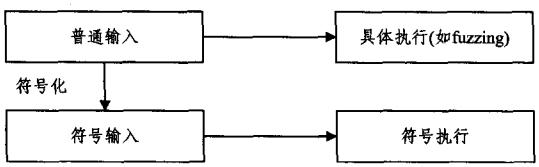

图 1 符号执行和普通执行关系

算法 1 利用符号执行生成新的测试用例

```
Input:Program P,input I,Path constraint(pc)=null
Output:Execution tree T,New test case set M
1. for each inst :=src op dest do
2. switch inst:
3.    case input_interface:
4.      mark_symbolic(I);
5.    end
6.    case src is symbolic
7.      update_symbolic();
8.    end
9.    case jmp(e)& e is symbolic related
10.      pc :=pc&e;
11.      result :=solve(pc);
12.        if result=null
13.          drop_branch();
14.        else
15.          continue;
16.        end
17.   end
18.   End
19. T :=Generate execution tree(pc);
20. M :=Traverse(T).
```

符号化(line 4)的本质是建立程序内存、寄存器字节的符号映射关系(以字节为单位)。形式化地，设定程序内存集合为 M，寄存器为 R，映射函数为 f，对应的符号变量集合为 M'、R'，则：

$$M \xrightarrow{f} M'$$

$$R \xrightarrow{f} R'$$

如对于汇编指令 mov [mem]，ebp，若 $f(ebp):=sv$，则符号化后的结果为 $f(mem):=sv$。目标程序对输入进行一系列解析并传递符号关系(line 7)，因此需要分析汇编指令(x86)语义并建立符号变量的传播关系。若汇编指令中有一个源操作数为符号变量，则目标操作数被符号化。在条件分支判断如 if(e)中，若谓词 e 为符号变量，则建立起该执行路径约束并利用求解器求解。

定义路径约束(Path Constraint，pc)为从程序入口点到执行该指令时满足该路径的所有条件分支的谓词集合。

求解器的原理是将路径约束转化为可满足性问题如 MiniSat 进行求解(line 11)。若无解，则该路径不可达(line13)，否则继续收集符号约束(line14－15)。如 f(input[0]):=a，f(input[0]):=b，pc 为 a=0x65&&b=0x66，可解得输入为‘ab’。由于死代码和求解器性能等限制，达到完全的覆盖率不太可能。可设定符号执行引擎停止条件(详见第 4.3 节)。在程序退出时，通过解析路径约束可构造出执行

树[1]，其中每个叶子节点对应一个测试用例相应的执行路径。遍历该树得到新的测试用例作为新的测试输入(line 19—20)。

4 系统设计与实现

本节详细介绍原型系统 SEVE 的模块功能、设计与实现。

4.1 符号化变量

该模块发现输入接口并符号化输入。本文针对的是 file-reading 型程序。它们通过文件读写 API 读取文件内容，因此可拦截 API 或通过插桩 API 得到输入的内存位置和读取字节数。插桩 API 效率更高且更准确，我们采用选取第二种方法。符号化的本质是建立内存、寄存器字节的映射关系。如何建立高效的映射关系关系到系统效率，不适当的数据结构与算法将使效率降低 10 倍以上。SEVE 用 C++实现，C++提供了 STL 数据结构 map。对内存和寄存器建立两个 map 数据结构，$map_m\langle addr, sv\rangle$ 和 $map_r\langle reg, sv\rangle$，即从内存地址和寄存器映射到符号变量。$sv := \{concrete, symbolic\}$。若该字节未被符号化，$sv=concrete$，$symbolic=$null；否则 $sv=symbolic$，concrete=null。SEVE 工作在二进制级，直接分析汇编指令，由于 x86 指令的复杂性，为了更容易处理汇编指令，我们定义以下符号变量表示(v_i 为符号变量)：

- $input(l)$：输入的第 l 个字节；
- v_1 op v_2：v_1 和 v_2 的算术或位运算，op 为运算符；
- $seqtag\langle v_0, \cdots, v_n\rangle (n=1$ 或 $3)$ 为顺序标签，其值为 $v_0 \cdots v_m$ 的字节值拼接，$n=1$ 表示字 word，$n=3$ 为双字 dword；
- $subtag\langle t, i\rangle (t=\{0, \cdots, 3\})$ 为子标签，表示 t 的第 i 个字节值。*seqtag* 和 *subtag* 对 casting 操作尤其有效。

4.2 符号变量的传播

本模块功能是利用插桩工具得到汇编指令并分析其语义信息，建立符号变量的传播关系，并更新映射关系。二进制级分析的挑战来源于汇编指令的复杂性和不易读性。精确的汇编指令语义分析和包含尽可能多的指令是构建有效系统的关键。我们对汇编指令进行形式化分析，将其分为 3 类：

(1)mov 型指令：mov，movz，movl etc；

(2)算术运算性指令：add，sub etc；

(3)跳转指令：call，jmp，jz，jnz，jg etc。

mov 型指令较为复杂，因为其操作数(即寻址方式)组合较多，如 mov R，R；mov [mem]，R；mov R，immd；mov [mem]，immd 等。R 为寄存器，mem 为内存，immd 为立即数。特别地，如 dest 为 eax(32 位)，但 dest 只有 16 位是符号变量时，仍然将 src 的低 16 位赋给 dest 的 16 位，它的高 16 位为非符号值。符号变量转播规则如下：

```
inst:mov dest,src
if(src. sv !=null){
  dest. sv. clear();
  dest. sv=src. sv;
}
else
  // no operation
```

算术运算指令和 mov 指令分析类似。动态插桩工具我们采用 Pin[13]，它由 Intel 开发，提供了类型丰富的 API 供操作。跳转指令中的 jmp 为无条件跳转，jz 表示为 0 时才跳转而 jnz 则表示不为 0 的时候跳转，jg 表示有符号大于的时候发生跳转，例如：

```
cmp eax,48h
jg label //jg:>时跳转
```

最终符号化的结果为〈…，t>0，…．〉，t=input(0)—48h，t>0|t<=0。

4.3 建立约束方程求解

为了能够遍历不同执行分析，我们对 if 判断的条件变量进行分析，并建立路径约束。汇编语句中通常的情况是 cmp src，test；jxx label。cmp 和 test 等指令属于隐含操作数的指令，其目的操作数为标志寄存器的某些位(如 ZF、CF)。这些指令执行后，标志寄存器的某些位会被符号化，因此条件跳转时能够根据符号变量建立约束方程。路径约束是满足达到当前执行路径的所有符号变量的约束集合。

利用解析器对当前路径约束进行求解，若有解，则此路径可达并继续执行，否则该路径不可达，停止该路径的探索。一种情况是约束其性能限制不能在有限时间内求得解，我们将某些符号变量用具体值代替(当然可能带来分析不精确问题)。当一定条件满足时，如程序正常退出或达到路径深度限制，利用路径约束可得到执行树。遍历此树(如深度优先)可得到每个叶子节点，它们对应每条执行路径和对应的测试用例。解析器采用 Microsoft Research 开发的 Z3[14]。STP Solver 是另一款功能较强的解析器，但由于只支持 Linux，因此我们选取 Z3 作为解析器。

5 实验结果与分析

实验环境为：Windows XP SP3，1GB 内存，CoreDuo 4200。为了与 fuzzing 对比，我们实现了一个基于 fuzzing 的文件格式漏洞分析系统 RFZ(Random FuZzer)。符号执行时间限定为 500 分钟。

第一组实验为 WMF 格式漏洞(CVE：CAN-2005-0611)[15]，它源于 Windows 的核心模块 gdi32. dll 处理 WMF 格式的方式。我们采用 Windows 自带的系统声音文件 grid_(cm). wmf 作为初始输入，大小为 2. 85kB。最终 SEVE 产生了 1388 个输入，其中第 38、39 个输入使程序崩溃(即发现可能的漏洞)。而产生第 38、39 个输入的时间为 56 分钟。

第二组实验为 mp3 格式漏洞(CVE-2006-0476)。选取操作系统自带文件 mqyrf. mp3 作为初始输入。在 500 分钟内共产生了 4202 个输入，其中第 110 个输入使程序崩溃(SEVE 在第 101 分钟时产生该输入)。

表 1 SEVE 实验结果

漏洞格式	测试用例个数(漏洞触发用例)	发现漏洞时间(分钟)	符号化(字节)	最大路径约束个数	发现漏洞时间(分钟)
WMF	1388(#38,39)	56	6103	38	56
mp3	4202(#110)	101	7902	51	101

表 2 fuzzing 实验结果

漏洞格式	测试用例个数	触发漏洞时间(分钟)
WMF	3902(#2910—3002)	381
mp3	4493(#2141—2205)	278

(下转第 138 页)

1) 执行树是隐式生成的

参 考 文 献

[1] Koblitz N. Elliptic curve cryptosystems [J]. Mathematics of computation, 1987(48): 203-209

[2] Miller V S. Use of elliptic curves in cryptography[C]// Proceedings of CRYPTO'85. LNCS 218, Springer, 1986: 417-426

[3] Lim C H, Lee P J. More flexible exponentation with precomputation Advances in Cryptology[C]// Crypto'94. Springer-Verlag, Berlin, 1994: 95-107

[4] Yang W C, Lin K M, Laih C S. A precomputation method for elliptic curve point multiplication[J]. Journal of the Chinese Institute of Electrical Engineering, 2002, 9(4): 339-344

[5] Lo G W. The study and implementation on elliptic curve digital signature schemes[D]. Taiwan, NCKU, 2000

[6] Fong K, Hankerson D, Lopez J, et al. Field inversion and point halving revisited [J]. IEEE Transactions on Computers, 2004, 53(8): 1047-1059

[7] Hankerson D, Menezes A, Vanstone S. Guide to Elliptic Curve Cryptography[J]. Computer Reviews, 2005, 46(1): 13-23

[8] Digital Signature Standard(DSS), FIPS 186-2[S]. USA: Federal Information Processing Standards Publication 186-2, National Institute of Standards and Technology, 2000

[9] 白羽，范恒英. 一种改进的椭圆曲线标量乘的快速算法[J]. 西南科技大学学报，2011，09

[10] Pang Shi-chun, Tong Shou-yu, Cong Fu-zong, et al. An Efficient Elliptic Curve Scalar Multiplication Algorithm Against Side Channel Attacks[C]// Proceedings of the 2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering (CMCE2010). Springer-Verlag, Berlin: 2010: 361-364

[11] 张建. GF(2^n)上椭圆曲线标量乘法快速算法的研究[D]. 呼和浩特：内蒙古大学，2012

[12] 周梦，周海波. 椭圆曲线快速点乘算法优化[J]. 计算机应用研究，2012，29(8)：3056-3058

[13] 李忠，彭代渊. 基于滑动窗口技术的快速标量乘法[J]. 计算机科学，2012，39(6A)：54-57

(上接第 121 页)

我们用 SEVE 产生若干个输入(实验设置为 5 个)后，马上将其运行以发现程序是否出错。表 1 的倒数两栏式符号执行性能评价，包括符号化字节数和最大路径约束个数。表 2 为使用 fuzzing 方法对上述两个漏洞的测试结果。RAZ 执行时间同样设定为 500 分钟。RAZ 分别产生了 3902 和 4493 个测试用例，其中针对第一个漏洞，第 2910－3002 号输入使程序崩溃；针对 mp3 漏洞，第 2141－2205 号输入使程序崩溃。

从表 1 及表 2 可以看出，SEVE 明显优于传统 fuzzing。首先，输入不依赖文件格式，即使对于 Word 等复杂格式也可分析，将 SEVE 稍作扩展，可用于网络型软件对协议漏洞进行分析，对于 SMB 等未公开协议尤其有效。其次，符号执行能够生成较有效的测试用例，在较早的时候发现软件 bug。

结束语 本文设计并实现了基于符号执行的二进制级漏洞发现系统，它有效克服了传统 fuzzing 测试的缺点，能有效生成测试输入，且无需分析输入格式。实验表明，我们的系统能在较短时间内产生测试输入并发现软件漏洞。后续工作主要在 3 个方面：1)对汇编指令进行精确语义分析并包含更多类型指令；2)求解器性能优化；3)执行树遍历算法研究。

参 考 文 献

[1] 王彤彤，韩文报，王航. 基于安全需求的软件漏洞分析模型[J]. 计算机科学，2007，34(9)：287-289

[2] 毛澄映，卢炎生，胡小华. 数据挖掘技术在软件工程中的应用综述[J]. 计算机科学，2009，36(5)：1-6

[3] Brumley D, Poosankam P, et al. Automatic patch-based exploit generation is possible: techniques and implications[C]// SP'08: Proceedings of the IEEE Security and Privacy Symposium. NJ: IEEE, 2008: 143-157

[4] 宋超臣，黄俊强，等. 计算机安全漏洞检测技术综述[J]. 技术研究，2012，01：77-79

[5] 吴志勇，王红川，等. Fuzzing 技术综述[J]. 计算机应用研究，2010，27(3)：829-832

[6] Miller B P, Fredriksenl. An empirical Study of the reliability of UNIX utilities [J]. Communications of the ACM, 1990, 33(12): 32-44

[7] Miller Barton P, Gergogy C, Fresrick M. An empirical study of the robustness of MacOS applications using random testing[C]// Proceedings of the 1st International Workshop on Random Testing. New York: ACM, 2006: 46-54

[8] 陈衍铃，赵静. 基于虚拟化技术的动态污点分析[J]. 计算机应用，2011，31(9)：2367-2372

[9] Cowbc, Pu C, et al. StackGuard : Automatic adaptive detection and prevention of buffer-overflow attacks[C]// Proceedings of the 7th conference on USENIX Security Symposium. Berkeley, 1998: 5-13

[10] King J C. Symbolic execution and program testing[C]// Communications of the ACM. 1976: 385-394

[11] 过辰楷，姬秀娟，许静. 基于分支混淆算法的符号执行技术 [J]. 计算机科学，2012，39(9)：115-119

[12] Cadar C, Ganesh V, et al. EXE: Automatically generating inputs of death[C]// CCS'06: Proceedings of the 13th ACM Conference on Computer and Communications Security. New York: ACM, 2006: 322-335

[13] Luk C-K, Robert C, et al. Pin: building customized program analysis tools with dynamic instrumentation[C]//PLDI05: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM, 2005: 190-200

[14] De Moura, Leonardo, Bjrner, et al. Z3: an efficient SMT Solver [C]//TACAS: Proceedings 14th International Conference. Berlin: Springer, 2008: 337-340

[15] 魏瑜豪，张玉清. 基于 fuzzing 的 MP3 播放软件漏洞发掘技术 [J]. 计算机工程，2007，33(24)：158-167