DOI:10.16511/j.cnki.qhdxxb.2011.10.007

ISSN 1000-0054 清华大学学报(自然科学版) 2011年 第51卷 第10期 7/25
CN 11-2223/N J Tsinghua Univ (Sci & Tech), 2011, Vol.51, No.10 1264-1268

参考安全补丁比对的软件安全漏洞挖掘方法

文伟平[1], 张普含[2], 徐有福[1], 尹 亮[1]

(1. 北京大学 软件与微电子学院, 信息安全系, 北京 102600; 2. 中国信息安全测评中心, 北京 100085)

摘 要: Windows 操作系统作为目前全球使用最广泛的桌面操作系统, 一旦其漏洞被利用将造成严重后果, 所以对 Windows 操作系统的漏洞发掘意义重大。当前对软件安全漏洞的发现更多的是依靠安全研究人员的经验和运气, 缺乏系统且有效的方法指导。为了找到一种能够快速发现 Windows 一类漏洞的方法, 本文从安全补丁的修补方法入手, 分析了漏洞补丁后可能引入新的安全隐患的 4 种模式, 提出了一种参考安全补丁比对的软件安全漏洞挖掘方法, 并对方法进行了形式化描述。利用此方法较好的解决了半自动化挖掘 Windows 系统漏洞问题。最后以 Windows 操作系统未知漏洞案例验证了该方法的有效性。

关键词: 漏洞发现; 补丁比对; 形式化描述

中图分类号: TP 393 文献标志码: A

文章编号: 1000-0054(2011)10-1264-05

Software security vulnerability exploitation method based on a security patch

WEN Weiping[1], ZHANG Puhan[2], XU Youfu[1], YIN Liang[1]

(1. Department of Information Security, School of Software and Microelectronics, Peking University, Beijing 102600, China;
2. China Information Technology Security Evaluation Center, Beijing 100085, China)

Abstract: Windows is the world's most widely used desktop operating system, so security vulnerabilities in windows have an enormous impact on system security and exploiting vulnerabilities in the Windows operating system has great significance. At present, the discovery of software security vulnerabilities depends mainly on the experience and luck of security researchers since they lack systematic and effective methods to find vulnerabilities. To more quickly find a class of vulnerabilities, this paper focuses on patch vulnerability with four types of security threat modes introduced patches. Then this paper describes a software security vulnerability exploitation method based on patch comparison. This method can be used to solve the semi-automatically find Windows vulnerabilities with patches. This method found unknown Windows operating system vulnerabilities to verify its effectiveness.

Key words: vulnerability exploitation; vulnerability patches comparison; formal description

随着互联网应用的飞速发展, 互联网及由互联网支撑的网络空间的安全问题、安全环境日趋复杂。在这种开放复杂的网络环境下, 安全漏洞所带来的危害更加严重。无论从国家层面的网络安全战略还是社会层面的信息安全防护, 安全漏洞已成为信息对抗双方博弈的核心问题之一。然而, 针对具体安全漏洞, 安全研究者往往进行大量的重复工作, 研究效率和效果上也有相当的局限性。因此, 快速、高效和准确挖掘软件中隐蔽的安全漏洞成为最近相关研究的热点和难点。

基于补丁比对的漏洞分析技术较早得到了应用, 其理论模型由 Halvar Flake 第一次提出了结构化比对[1] 的补丁比对算法, 同年 Tobb Sabin 提出图形化比对算法作为补充[2]。2008 年 1 月 David Brumley 等人在 IEEE 上发表论文[3], 肯定了补丁比对技术在现实环境中的应用价值。2006 年 11 月 eEye 发布了 eEye Binary Diffing Suite(EBDS)[4]。这些工具的出现和改进, 使得补丁比对技术在漏洞分析过程中得到了越来越广泛的应用。在国内, 开展补丁比对理论和技术研究的相关单位逐步增多。解放军信息工程大学在串行结构化比对算法[5] 基础上, 提出了一种基于全局地址空间编程模型实现的并行结构化比对算法[6]。北京大学软件与微电子学院软件安全研究小组[7] 在开源工具 eEye Binary Diffing Suite(EBDS) 基础上已经基本实现基于结构化比对的补丁漏洞分析工具, 能够实现函数级结构化比对和基本块级结构化比对功能, 与以往补丁漏洞分析工具相比, 速度更快、定位更加准确。

补丁比对提高了定位二进制文件安全漏洞的效

收稿日期: 2011-08-15
基金项目: 国家自然科学基金资助项目 (61170282)
作者简介: 文伟平(1976—), 男(汉), 北京, 副教授。
E-mail: weipingwen@ss.pku.edu.cn

率，作为一种软件安全漏洞分析技术被广泛使用。但补丁实施之后带来新的安全隐患一直未被业界所重视。

针对上述情况，本文通过补丁比对分析提出一种参考安全补丁比对的软件安全漏洞挖掘方法。相比现有的软件安全漏洞挖掘方法，该方法分析粒度更细，能挖掘到补丁后更加隐蔽的安全漏洞。该方法对发布补丁厂商和软件安全研究者有重要的参考价值。

1　参考安全补丁比对的软件安全漏洞挖掘方法

软件安全漏洞一旦被发掘到并且公开后，软件厂商会定期或不定期的提供相应的安全漏洞补丁。由于补丁本身可能未经过严格的安全性测试，有可能在原程序中引入新的安全漏洞。

1.1　原因分析

通常情况下，由于系统函数调用关系复杂，系统或应用软件的关键数据区域可被不同进程或线程修改，软件厂商在修补安全漏洞的时候，希望能通过做最小的改动来解决当前遇到的安全问题。通过对众多安全补丁的补丁比对分析发现，软件厂商对漏洞代码的修改及代码运行流程基本不会有太大的变化。而这种漏洞修补方式可能存在如下安全隐患：

1）软件厂商修补漏洞缺乏全局考虑，通常注重对漏洞点的修补。在复杂系统中，软件模块复用情况较多，与本漏洞相同或相似特征属性的漏洞在系统中可能还会存在，而此时由于安全补丁暴露了一种漏洞特征属性，分析人员可以利用这种漏洞特征属性来挖掘其他未知漏洞；

2）通过补丁比对发现，软件厂商对漏洞代码进行修改时，往往只考虑当前漏洞的上下文环境，而未必考虑到整个系统或者第三方代码对全局变量或逻辑条件带来的影响，本文列举的案例证明了这一点；

3）软件厂商进行补丁开发，一般修改漏洞点对应或相关的源代码。但是从源代码的角度进行修改，未必能考虑到真实逆向分析环境中出现的各类复杂情况。

1.2　方法原理

通过对安全补丁的分析，可以找出补丁所修补的代码位置（patch location，简称 P 点）以及实际出现问题的代码位置（bug location，简称 B 点）。在实际环境中，B 点一般是一个漏洞点，但 P 点可能是一个补丁点或者多个补丁点的集合。如果从代码执行开始，每条到达节点 B 的路径都要经过节点 P，则控制流图中节点 P 是节点 B 的必经节点。根据 B 点和 P 点的相对位置关系，大致可以分为如下4种情况：

1）B 点和 P 点重合。直接修改漏洞代码，例如替换漏洞代码所在的基本块或不安全函数、直接修改触发漏洞点的逻辑条件等。

2）B 点和 P 点位于同一函数中。如果 P 点不是 B 点的必经节点，存在其他路径绕过 P 点到达 B 点，则说明该漏洞修补可能存在安全隐患。

3）B 点和 P 点集合中的某个补丁点 P_n 位于同一基本块中。P_n 是 B 点的必经节点，如果 P_n 的逻辑控制条件与系统中可能调用到的其他函数相关，即其他函数可能修改 P_n 的逻辑控制条件，在触发其他相关函数后仍然可以触发 B 点，则漏洞修补存在安全隐患。

4）B 点和 P 点位于不同基本块中，且 B 点和 P 点分布在不同函数中。漏洞代码和修补代码，位于不同函数中，这种安全漏洞修补方式，最有可能存在安全隐患。由于系统函数调用关系相当复杂，如果对每个函数调用参数的约束和检查不到位，污点数据的动态传递很可能重新触发漏洞代码而导致新的安全隐患。

1.3　形式化描述

情况一　B 点和 P 点重合的情况，经过研究和以往案例的总结，推测该类漏洞修复方式一般不会引入新的安全隐患。

情况二　使用形式化语言描述满足触发漏洞的程序执行路径如下：

1）CG（call graph）为程序中函数调用图，CG图可以表述为一个三元组有向图 $G=(F, E, \text{entry})$。其中，F 表述函数集合；E 表述函数调用集合，即一个函数 F_i 调用另一个函数 F_j 的有向边 $\langle F_i, F_j \rangle$；entry 表示入口函数集合 $\text{entry} \subseteq F$；

2）CFG（control flow graph）为函数控制流图，CFG 表述为一个四元组有向图 $G=(N, E, \text{entry}, \text{exit})$。其中，$N$ 表示一个函数中各个节点集合；E 表示从一个节点 N_i 转移到另一个节点 N_j 的有向边 $\langle N_i, N_j \rangle$；entry 表示函数入口节点集合 $\text{entry} \subseteq N$；exit 表示函数出口节点集合 $\text{exit} \subseteq N$；

3）如果从节点开始，所有到达节点 a 的路径都要经过节点 b，则称节点 b 为节点 a 的必经节点；

4) 设 B 点所在的函数为 F_b，P 点所在的函数为 F_p。则查找存在问题的路径为：从入口函数 F_{entry} 开始到达 F_b，并且绕过 F_p 的路径。从 F_{entry} 绕过 F_p 到达 F_b 的路径分为两种情况，CG 图查找路径和 CFG 图查找路径；

5) CG 图查找路径：$P=(F_0, F_1, F_2, \cdots, F_{n-1}, F_n)$，$F_0=F_{entry}$，其中存在 $F_n=F_b$，且任意 $0\leqslant j\leqslant n$ 满足 $F_j\neq F_p$；

6) CFG 图查找路径：第 5 步，满足第 5 步的条件，函数 F_j 调用 F_{j+1}，则从函数 F_j 到开始调用函数 F_{j+1} 所执行的路径为 $P_{\langle F_j, F_{j+1}\rangle}=(N_0, N_1, N_2, \cdots, N_{n-1}, N_n)$，$N_0=N_{entry}$，$N_n$ 是调用函数 F_{j+1} 的基本模块。

最终找到的 P 和 $P_{\langle F_j, F_{j+1}\rangle}$ 都是满足触发漏洞的代码执行路径。

情况三　F_b 和 F_{pn} 位于同一基本块 N_n。$P=(F_0, F_1, F_2, \cdots, F_{n-1}, F_n)$，$F_0=F_{entry}$，$0\leqslant j< i\leqslant n$，其中存在 $F_j=F_{pn}$，$F_i=F_b$，存在路径 $P=(F_0, F_1, F_2, \cdots, F_{j-1}, F_j)$ 影响 F_{pn} 的逻辑运行条件。触发漏洞的有效执行路径为：$P=(F_0, F_1, F_2, \cdots, F_{j-1}, F_j, \cdots, F_{n-1}, F_n)$，$0\leqslant j\leqslant n$，$j\neq i$。路径 P 满足触发漏洞的代码执行路径。

情况四　F_b 位于基本块 N_i，F_p 位于基本块 N_j 中，$i\neq j$，若存在路径 $P_{\langle F_k, F_{k+1}\rangle}=(N_0, N_1, N_2, \cdots, N_i, \cdots, N_j, \cdots, N_{n-1}, N_n)$，$N_0=N_{entry}$，$N_i$ 先于 N_j 运行，路径 $P_{\langle F_k, F_{k+1}\rangle}$ 满足触发漏洞的代码执行路径。

2　案例验证

本案例中触发漏洞的有效执行路径搜索大部分通过手工完成，实现复杂环境下有效执行路径的自动化搜索，缓解路径爆炸也是软件漏洞发掘迫切需要解决的难题之一。本案例实验环境为 Windows XP Professional SP3，补丁包更新至 2011 年 2 月 8 日。

2.1　案例分析

以 MS10-015[8] 为例验证上述方法的有效性。漏洞存在的主要原因可以查看文[8]。经过对漏洞触发代码的逆向工程分析，安全补丁在触发漏洞点函数即 Ki386BiosCallReturnAddress 函数之前增加了逻辑检测条件：

```
cmp  byte ptr [edi+51h], 1
```

而此时的 edi 是当前线程的_Ethread 结构起始地址(也是_Kthread 的起始地址)，edi+51h 为_Ethread 结构中某数值。表 1 是补丁前后_Kthread 数据结构的修改情况(说明：[X] 表示该字段占 X 字节)：

表 1　MS10-015_Ethread 结构补丁前后比较

补丁前	补丁后
kd> dt nt!_kthread	kd> dt nt!_kthread
……	……
+0x051 Spare0: [3]	+0x051 VdmSafe: [1] +0x052 Spare0: [2]

可见这一位就是微软专门为针对这个补丁而增设的标志位 VdmSafe。漏洞点 B 及补丁点 P 已经定位。如果要触发 B 点 Ki386BiosCallReturnAddress 函数，可以将 VdmSafe 置为 1，那么原来的攻击代码仍然能运行成功。因此，尽管微软公司已经发布 MS10-015 的补丁，但这个补丁可能是不完善的，是属于列举的第 3 种情况。

2.2　方法运用

通过对 MS10-015 漏洞触发原理分析，如果要触发存在漏洞的函数 F_b 即 Ki386BiosCallReturnAddress 函数，必需将逻辑运行条件 VdmSafe 置为 1，那么原来的攻击代码仍然能运行成功。实际上通过驱动在 ring0 级别下修改_Ethread 的结构亦能实现攻击目标，但与本地权限提升的思路不符，所以要从 ring3 调用系统本身的内核函数，实现同样的效果。

通过对系统内核驱动使用到函数进行调用关系分析及动态跟踪调试，找到了一个满足要求的函数，也是微内核中的唯一一个可以被普通用户直接调用的函数：Ki386SetupAndExitToV86Code 函数，这个函数在补丁后也被修改了。因此，本案例属于存在多个补丁点的情况，其修改情况如图 1 所示。

这个函数会将 VdmSafe 置位。于是设想：如果能调用到这个函数，把当前线程的 VdmSafe 位修改为 1，然后再进行攻击代码的加载，应该也会成功。

通过研究，该函数可以被其他驱动调用到，这里调用它的驱动是 XXX.sys，而且 XXX!VpInt10CallBios 可以被 XXX!xxxPortQueryServices 函数调用，而 xxxPortQueryServices 为导出函数，是个在一般用户权限下能够调用的函数，函数调用关系如图 2 所示。

```
;__stdcall Ki386SetupAndExitToV86Code(x)
_Ki386SetupAndExitToV86Code@4 proc near
push   ebp
push   ebx
push   esi
push   edi
……
mov   dword ptr[eax+78h], 23h
mov   dword ptr[eax+74h], 11FFEh
mov   [eax+70h], edi
mov   dword ptr[eax+4Ch], 0FFFFFFFFh
mov   dword ptr[eax+48h], 0FFFFFFFFh
mov   edi, [ebx+124h]  //
取得当前线程_Ethread 结构起始地址
mov   byte ptr[edi+51h], 1 //修改 VdmSafe 标
      志的语句
add   eax, 8Ch
cli
……
_Ki386SetupAndExitToV86Code@4 endp
```

图 1　MS10-15 安全补丁修改的关键代码

```
kd> kb
ChildEBP RetAddr  Args to Child
f80ee38c 80596adc 00013000 f80ee6dc f80ee714
nt! Ki386SeupAndExitToV86Code
f80ee3e4 f8150f48 00000010 f80ee410 e14e7008
nt! Ke386CallBios+0x17e
f80ee6f8 f89ef454 81bbeaa8 f80ee714 81bbe848
XXX! VpInt10CallBios+0x102
f80ee760 f89ed165 81bbeaa8 f80ee82c
f815b320yyy! InitializeModeTable+0xd8
f80ee76c f815b320 81bbeaa8 820ecaf0 81fa0210
yyy! VgaInitialize+0x23
f80ee82c 804ef003 81bbe790 820ecae0 820ecae0
XXX! pVideoPortDispatch+0xcc4
f80ee83c 805786f2 81bbe778 81be535c f80ee9e4
nt! IopfCallDriver+0x31
……
```

图 2　关键函数的函数调用关系

因此，该补丁可能存在新的安全隐患。补丁后攻击代码实现攻击流程如图 3 所示。

2.3　其他案例

除上述案例外，北京大学软件与微电子学院软件安全研究小组[7] 发现的 MS11-010[9] 进一步证明该方法的有效性。该漏洞是在分析 MS10-011[10] 安全补丁的基础上，采用本方法发现的一个新的安全漏洞。MS10-011 补丁为本文列举的第 4 种情况，*B* 点和 *P* 点位于不同的函数中，补丁增加了运行

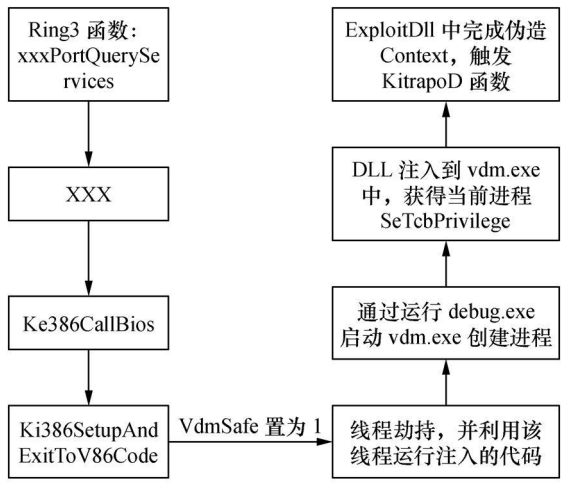

图 3　补丁后的攻击代码的实现流程

至 *B* 点的逻辑条件，而这个逻辑条件是普通用户构造某些函数进行特定序列调用可以进行修改的，从而逻辑条件被非法的利用导致新的安全隐患。其漏洞挖掘过程的技术细节可以参看相关技术分析报告[7]。

3　结　论

当前对软件安全漏洞的挖掘缺乏系统且有效的方法指导。本文提出的一种参考安全补丁比对的软件安全漏洞挖掘方法，通过分析软件补丁修改过程中可能存在的不足，并可能导致新的安全漏洞，是对传统软件安全漏洞发掘方法的补充，能发现更加隐蔽的、因安全补丁引入的安全漏洞。并且利用此方法在实际的工作中发现了多个未知漏洞。因此，该方法对于发布补丁厂商和软件安全研究工作者有重要的参考价值。

参考文献　(References)

[1] Flake H. Structural comparison of executable objects [C] // Proceedings of the IEEE Conference on Detection of Intrusions, Malware, and Vulnerability Assessment. Dortmund, Germany: SIG SIDAR, 2004.

[2] Sabin T. Comparing binaries with graph isomorphisms [Z/OL]. (2011-06-25), http://razor.bindview.com/publish/papers/comparing binaries.html, 2004.

[3] Brumley D, Poosankam P, Song D, Jiang Z. Automatic patch based exploit generation is possible: Techniques and implications, security and privacy [C] // Proceedings of the IEEE Symposium on Security and Privacy. California, USA: IEEE Computer Society, 2008: 143 - 157.

[4] eEye Security. eEye binary diffing suite (EBDS) [Z/OL]. (2011-06-25), http://research.eeye.com/html/tools/RT20060801-1.html. Version 1.0.5.

[5] Brumley D, Caballero J, Liang Z, Newsome J, Song D. Towards automatic discovery of deviations in binary implementations with applications to error detection and fingerprint generation [C]// Proceedings of the USENIX Security Symposium. Boston, USA: USENIX, 2007.

[6] 罗谦, 舒辉, 曾颖. 二进制文件结构化比较的并行算法实现[J]. 计算机计应用, 2007, **27**(5): 1260 – 1263.
LUO Qian, SHU Hui, ZENG Yin. Parallel algorithm for structural comparison of executable objects [J]. *Journal of Computer Applications*, 2007, **27**(5): 1260 – 1263. (in Chinese)

[7] Software Security Research Group, School of Software and Microelectronics, Peking University. [Z/OL]. (2011-06-25), http://www.pku-exploit.com/.

[8] Microsoft Corporation [Z/OL]. (2011-06-25), http://technet.microsoft.com/en-us/security/bulletin/ms10-015.

[9] Microsoft Corporation [Z/OL]. (2011-06-25), http://technet.microsoft.com/en-us/security/bulletin/ms11-010.

[10] Microsoft Corporation [Z/OL]. (2011-06-25), http://technet.microsoft.com/en-us/security/bulletin/ms10-011.

(上接第 1263 页)

[6] Masood A, Bhatti R, Ghafoor A, et al. Scalable and effective test generation for role-based access control systems [J]. *IEEE Transactions on Software Engineering*, 2009, **35**(5): 654 – 658.

[7] Chupilko M. Constructing test sequences for hardware designs with parallel starting operations using implicit FSM models [C]// Proceedings of the IEEE East-West Design and Test Symposium. St. Petersburg, Russia: IEEE Computer Society, 2010: 487 – 490.

[8] Marczynski R, Thornton M, Szygenda S A. Test vector generation and classification using FSM traversals [C]// Proceedings of the 2004 IEEE International Symposium on Circuits and Systems. Vancouver, Canada: Institute of Electrical and Electronics Engineers Inc, 2004: 309 – 312.

[9] Tretmans J. Conformance testing with labelled transition systems: Implementation relations and test generation [J]. *Computer Networks and ISDN Systems*, 1996, **29**(1): 49 – 79.

[10] 蒋凡, 宁华中. 基于标号变迁系统的测试集自动生成[J]. 计算机研究与发展, 2001, **38**(2): 1435 – 1445.
JIANG Fan, NING Huazhong. Automatic test suit generation based on labelled transition system [J]. *Journal of Computer Research and Development*, 2001, **38**(2): 1435 – 1445. (in Chinese)

[11] Malik Q A, Lilius J, Laibinis L. Model-based testing using scenarios and event-B refinements [J]. *Lecture Notes in Computer Science: Methods, Models and Tools for Fault Tolerance*, 2009, **5454**: 177 – 195.

[12] 张敏, 冯登国, 陈驰. 基于安全策略模型的安全功能测试用例生成方法[J]. 计算机研究与发展, 2009, **46**(10): 1686 – 1692.
ZHANG Min, FENG Dengguo, CHEN Chi. A security function test suite generation method based on security policy model [J]. *Journal of Computer Research and Development*, 2009, **46**(10): 1686 – 1692. (in Chinese)