doi :10.3969/j.issn.1671-1122.2011.07.016

# 基于补丁引发新漏洞的防攻击方法研究

徐有福，文伟平，尹亮

（北京大学软件与微电子学院信息安全系，北京 102600）

摘 要：文章针对漏洞补丁的修补方法进行了深入研究，分析了漏洞补丁后可能引入新的安全隐患，提出了基于漏洞补丁引发新漏洞的漏洞挖掘方法，并进行了方法论证。最后以使用此方法发现的 Windows 操作系统未知漏洞案例验证了该方法的有效性。

关键词：安全漏洞；漏洞挖掘；漏洞补丁

中图分类号：TP393.08 文献标识码：A 文章编号：1671-1122（2011）07-0045-04

## Based on the Cause of the New Vulnerabilities Patch Attack Method Research

XU You-fu, WEN Wei-ping, YIN Liang

( 1. *Department of Information Security, SSM, Peking University, Beijing* 102600, *China*)

**Abstract:** Aiming at the vulnerabilities of the patch repair method is studied, the analysis after the patch may loopholes to introduce new security hidden danger, was put forward based on the cause of the new vulnerabilities loopholes patch holes mining method, the method and argument. Finally to use this method of the Windows operating system found that unknown vulnerabilities cases verify the validity of this method.

**Key words:** Security Vulnerabilities; digging vulnerabilities; vulnerability patches

## 0 引言

笔者在北京大学软件与微电子学院软件安全研究小组[1]的软件安全漏洞研究工作中，通过补丁比对分析提出一种基于漏洞补丁引发新漏洞的漏洞挖掘方法。相比现有的软件安全漏洞挖掘方法，该方法分析粒度更加全面、细致，能挖掘到补丁后可能存在的安全漏洞。此外，方法分析成果对发布补丁厂商有很大的参考价值。

## 1 基于漏洞补丁引发新漏洞的漏洞挖掘方法

作为软件安全漏洞的补救措施，软件开发商会定期或不定期地提供相应的修补程序，即软件安全补丁。补丁一般是对危险的漏洞进行应急响应的快速解决措施，对目标漏洞的修补不一定完善，此外，补丁本身由于没经过严格测试，也可能将新的漏洞引入原程序。

### 1.1 方法技术框架

通常情况下，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，因此，软件处理流程上不会有很大的变化。而这种漏洞修补方式可能存在如下安全隐患：

1）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未知漏洞；2）临时应急补丁的安全性有待考验，可能解决了当前安全漏洞，却带来了新的安全问题；3）补丁修改时，开发者只是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带来的影响；4）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。

通过对补丁的分析，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

收稿时间：2011-06-10

作者简介：徐有福（1985-），男，江西，硕士研究生，主要研究方向：网络安全；文伟平（1976-），男，湖南，副教授，主要研究方向：网络攻击与防范、恶意代码研究、信息系统逆向工程和可信计算技术等；尹亮（1986-），男，湖南，硕士研究生，主要研究方向：漏洞分析和漏洞挖掘。

Location，简称 B 点）。在 CG 图和 CFG 图 [2] 中，有可能存在循环。如果从初始节点开始，每条到达节点 B 的路径都要经过节点 P，则控制流程图中节点 P 是节点 B 的前必经节点。

软件漏洞的修补方式，按照 B 点和 P 点的相对位置关系，大致可以分为如下四种 ：

1）B 点和 P 点重合。对漏洞代码的修补，是直接在漏洞代码上修改的，例如替换不安全函数、直接修改触发漏洞点的逻辑条件等。这种情况较为简单，一般通过人工可以判断漏洞修补是否完善。2）B 点和 P 点位于同一函数中。漏洞代码和修补代码位于同一函数中，一般来说，从 P 点会顺序执行到 B 点，即 P 点是 B 点的前必经节点。可以通过对该函数的控制流进行分析，判断 P 点是否是 B 点的前必经节点。如果 P 点不是 B 点的前必经节点，也就是说存在其他路径从该函数的入口处到达 B 点，则说明该漏洞修补可能是有问题的。3）B 点和 P 点位于同一函数中。P 点是 B 点的前必经节点，如果 P 点的逻辑控制条件与其他函数相关，即其他函数能修改 P 点的逻辑控制条件，在触发其他相关函数后仍然可以触发 B 点，则漏洞修补不彻底。4）B 点和 P 点位于不同函数中。漏洞代码和修补代码位于不同函数中，这种修补方式，最有可能存在安全隐患。一个函数可能调用多个函数，也可能被多个函数调用，甚至被同一函数的多个语句反复调用 ；函数可以用函数名的方式被调用，也可以通过函数指针的形式被调用 ；对调用参数的约束和检查，可能是在被调函数中对形参的检查，也可能是在调用函数中对实参的检查，更多的情形是调用函数和被调函数中，都分别对实参和形参从不同方面进行一定的检查，当然还有情形就是调用函数和被调函数都没有对参数进行检查，这类情形往往会产生漏洞 [2]。

函数调用如此复杂，而补丁发布时间紧急，测试不充分，所以极容易造成漏洞修补不彻底。查找从程序入口点 Entry，绕过 P 点，到达 B 点的有效执行路径是研究该方法的关键。用静态分析的方法，会造成大量的误报和漏报，并会出现路径爆炸问题。而采用条件执行，可以在执行中收集路径上的约束信息，据此可以判断该路径是否是实际可执行的。从补丁信息入手，结合静态分析和条件执行，是寻找软件潜在漏洞的有效方法，会起到事半功倍的效果。图 1 是获取有效执行路径的流程，分为路径查找、条件执行和约束求解三个部分。

1）路径查找 ：已知 P 点和 B 点，通过静态分析函数调用图和各函数内的控制流图，可以得到一组从程序入口点 Entry 到 B 点的绕过 P 点的可能路径。2）条件执行 ：这里的条件执行，是基于路径的条件执行，是按需调度的，不追求路径的覆盖率，只是在指定路径上通过条件执行的方法，收集约束信息。3）约束求解 ：对条件执行收集的约束进行求解，找出符合约束的输入。如果无解，则表示与约束相对应的路径不可达，即程序实际执行时不会执行该路径 ；否则则相应的路径可达，并已求解出了触发该条路径的输入。

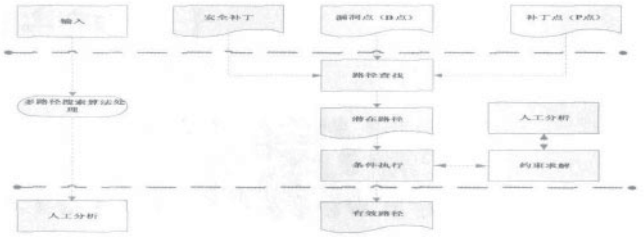


图1 获取有效执行路径流程示意图

以上三个部分，是有机结合的。路径查找会找到所有可能的执行路径 ；条件执行会尝试执行这些路径，收集路径上的约束信息，并根据需要及时进行约束求解，以判断当前路径是否是实际可执行的，及时调整条件执行的流程。目前条件执行的逻辑不仅限于是符号执行得到的结果。符号执行通过代码变量的逻辑抽象与控制流相结合得到条件约束，最后通过约束求解的方法，来判断代码内部是否存在安全漏洞。

### 1.2 方法验证

下面以 MS10-015 为例验证上述思路的正确性。MS10-015 修补的安全漏洞由谷歌的安全工程师 Tavis Ormandy 发现的，在 Windows 系列操作系统存在 17 年之久，几乎适应 Windows 全系列操作系统。该漏洞存在的主要原因是 ：在操作系统调用 Kitrap0D 进行内核异常处理的时候，会将调用前线程的 Context 结构保存在 _Ktrap_frame 结构中，而 Context 结构来自于 VdmTib->VdmContext，指向 VdmTib 结构体的指针是来自 ring3 级的 _Teb->VDM，是由用户态可以控制的，所以，在这个漏洞中伪造该 Teb 结构中的数据，就可以指定 Kitrap0D 的运行流程，使之运行到 Ki386BiosCallReturnAddress 函数中，而 Ki386BiosCallReturnAddress 函数会从 _Ktrap_frame 结构中提取数值，完成异常处理过程，跳到一个用户态地址，也就是当前进程里的权限提升代码的入口地址，然后回到当前进程里执行该系统调用从而达到本地提权的目的。经过逆向工程分析，补丁前后的 Kitrap0D 函数情况如表 1 所示 ：

表1 MS10-015 Kitrap0D函数补丁前后比较

| 补丁前 | 定义值 |
|---|---|
| mov eax, offset Ki386BiosCallReturnAddress<br>cmp eax, [edx]<br>jnz short loc_468176<br>mov eax, [edx+4]<br>cmp ax, 0Bh<br>jnz short loc_468176<br>jmp Ki386BiosCallReturnAddress | mov eax, offset Ki386BiosCallReturnAddress<br>cmp eax, [edx]<br>jnz short loc_4683CC<br>mov eax, [edx+4]<br>cmp ax, 0Bh<br>jnz short loc_4683CC<br>mov ebx, ds:0FFDFF01Ch // 取 KPCR<br>mov edi, [ebx+124h] //<br>cmp byte ptr [edi+51h], 1 // 取 VdmSafe 标志位<br>jnz short loc_4683CC //<br>mov byte ptr [edi+51h], 0<br>jmp Ki386BiosCallReturnAddres |

在补丁后版本中，触发 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 2 是补丁前后 _Kthread 数据结构的修改情况 ：

表2 MS10-015 _Ethread结构补丁前后比较

| 补丁前 | 定义值 |
|---|---|
| kd> dt nt!_kthread<br>… …<br>+0x051 Spare0 : [3] UChar | kd> dt nt!_kthread<br>… …<br>+0x051 VdmSafe : UChar<br>+0x052 Spare0 : [2] UChar |

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，于是通过对系统内核的动态跟踪调试，找到了满足要求的函数，该函数可以被普通用户直接调用。因此，尽管微软公司已经发布 MS10-015 的补丁，但这个补丁是不完善的。

## 2 漏洞挖掘实例

此漏洞挖掘实例为分析 MS10-011 发现的一个新的未知漏洞。

### 2.1 补丁分析

在分析 MS10-011 的漏洞补丁时，我们发现微软增加了两处比较判断操作，通常情况下我们会怀疑比较判定之后的跳转能否绕过的问题。具体关于 MS10-011 漏洞请参考文献 [4]。表 3 为 MS10-011 补丁新增的两处判定（这个补丁所做的事情不只这些，这里只描述关键的）。

表3 MS10-011补丁第一处新增判断

| 补丁前 | 定义值 |
|---|---|
| 无 | .text:75AA4780 and byte ptr [eax+39h], 0DFh<br>.text:75AA4784 mov eax, [ebp+var_1C]<br>.text:75AA4787 cmp [eax+58h], esi<br>.text:75AA478A jnz short loc_75AA4798 |

表 4 为另外一处增加的判断跳转：

表4 MS10-011补丁第二处新增判断

| 补丁前 | 定义值 |
|---|---|
| 无 | .text:75AA491B and byte ptr [eax+39h], 0DFh<br>.text:75AA491F mov eax, [ebp+var_1C]<br>.text:75AA4922 xor esi, esi<br>.text:75AA4924 cmp [eax+58h], esi |

经过分析发现表 4 这处代码是用于处理在 CsrLocate ThreadByClientId 函数返回值不为 0 的时候执行，此时进程 CSR_PROCESS 的线程计数不为 0，即进程中仍然存有线程。按照上节中的理论思路，我们进行深入研究，发现这处补丁修改的地方果然导致了新的漏洞，下面是漏洞挖掘详细过程。

### 2.2 漏洞描述

在通常情况下，当一个用户退出时，运行在该用户下的进程都将被终止。但是我们在研究过程中发现，能够在受限用户下运行特制的程序，在该受限用户退出的时候，所运行的特制的程序不会被终止，将继续在系统中运行。当管理员用户或者其他高权限用户登录时，该特制的程序能进行枚举窗口、记录键盘信息、进行截屏等一系列操作，因此能够造成权限提升。

本文所分析的操作系统为 Windows XP Professional SP3，已经更新截至 2010 年 9 月 29 日微软发布的所有补丁。该漏洞位于 CSRSrv.DLL 中，版本号为：5.1.2600.5915 (xpsp_sp3_gdr.091211-1412)。

### 2.3 漏洞原理分析

Windows LPC 通信机制：LPC（Local Inter-Process Communication）是一种高效的进程间通信机制，在 Windows 2000、XP、2003 操作系统中被使用。使用 LPC 通信的基本步骤如下：

1）服务端调用 NtCreatePort 函数创建一个连接端口；2）服务端通过 NtListenPort 函数监听所创建的连接端口，获取新的连接请求，必须始终有一个线程在这个端口上等待；3）客户端调用 NtConnectPort 或者 NtSecureConnectPort 连接到服务端在步骤 1 中所创建的连接端口；4）服务端分析连接请求，通过调用 NtAcceptConnectPort 和 NtCompleteConnectPort 确定接收客户端的连接，并创建一个对应的通信端口，客户端和服务端都将通过这个通信端口来通信；5）服务器启动一个循环，通过 NtReplyWaitReceivePort 来接收客户端的消息，处理消息后调用 NtReplyPort 回复客户端；6）客户端调用 NtRequestWaitReplyPort 发送一个新的请求到服务端，等待服务端处理。在此过程中，客户端线程将阻塞，直到收到服务端的回复。

### 2.4 漏洞利用过程

ApiPort 的连接：在 test.exe 进程创建的时候，将会连接 Csrss 建立的 ApiPort 端口，传递给 Csrss 的 LPC 消息类型为 LPC_CONNECTION_REQUEST (0xA)。在 CsrApiRequestThread 函数中处理该类型消息所调用的函数为 CsrApiHandleConnectionRequest，它的大概工作流程如下：

1）调用 CsrSrvAttachSharedSection 函数，在它内部调用 NtMapViewOfSection 将名为 CsrSrvSharedSection 的共享内存区映射到进程地址空间中。如果映射成功，则返回 0，否则则返回 NtMapViewOfSection 的出错状态；2）如果 CsrSrvAttachSharedSection 返回 0，在调用 NtAcceptConnectPort 时将创建一个通信端口，将该通信端口的句柄填充到 CSR_PROCESS 结构体的 HANDLE ClientPort 成员；3）在创建通信端口成功的情况下，调用 NtCompleteConnectPort 完成连接。并将 CSR_PROCESS.Flags[1] 第 6 比特置为 1。

ExitProcess 函数分析：在 Windows 系统中，当进程结束时，都将调用到 ExitProcess 函数，ExitProcess 函数位于 kernel32.dll 中（文件版本号：5.1.2600.5781 (xpsp_sp3_gdr.090321-1317)），它进一步调用位于地址 0x7C81CA6C 处的 _ExitProcess 函数，该函数在地址 0x7C81CAC3 处调用 CsrClientCallServer 函数。

CsrClientCallServer 函数是 ntdll.dll 的一个导出函数，内部将调用 NtRequestWaitReplyPort 函数向 ApiPort 端口发送 LPC 消息。在地址 0x7C81CAB6 处的 push 10003h 作为 CsrClientCallServer 函数的第三个参数，将填充到 LPC 消息偏移 0x1C 处，根据“2.3 Win32 子系统进程与 CSRSS 的通信”的分析，0x10003 对应了 BaseServerApiDispatchTable 分发表中的索引值为 3 的函数，即 BaseSrvExitProcess 函数。当 CsrClientCallServer 返回时，test.exe 对应的 CSR_PROCESS 结

构体中的引用计数为 1（CSR_PROCESS.ReferenceCount = 1），此时还没有从 CsrRootProcess 链表中摘除。

BaseSrvExitProcess、CsrDereferenceThread 函数：BaseSrvExitProcess 是在 CsrApiRequestThread 函数中调用的，BaseSrvExitProcess 返回后，进程 CSR_PROCESS 的引用计数为 2。接下来会调用 CsrDereferenceThread，使 test.exe 发送 LPC 消息的线程 CSR_THREAD 引用计数减 1，为 0，此时将调用 CsrThreadRefcountZero 函数，它将调用 CsrRemoveThread、CsrDeallocateProcess、CsrDereferenceProcess 三个函数。这三个函数的功能如下：

1）CsrRemoveThread 函数从 CsrThreadHashTable 数组中移除线程 CSR_THREAD 信息；从 CsrThreadHashTable 数组中移除线程信息，将线程所属进程的线程计数减 1，如果进程的线程计数为 0，并且 CSR_PROCESS.Flags[1] 第 6 比特为 0 则调用 CsrLockedDereferenceProcess 函数；2）CsrDeallocateProcess 释放线程 CSR_THREAD 占据的内存空间，该函数也可以用于释放 CSR_PROCESS 占据的内存空间；3）CsrDereferenceProcess 与 CsrLockedDereferenceProcess 函数功能相似，都是将进程 CSR_PROCESS 的引用计数减 1，如果为 0，则调用 CsrProcessRefcountZero 函数，并进一步调用 CsrRemoveProcess 函数从 CsrRootProcess 链表中摘除进程的 CSR_PROCESS 结构。

ApiPort 的关闭：接下来将执行 0x7C81CACC 处的 NtTerminateProcess 函数，test.exe 进程在这里被终止，它在创建的时候连接的 ApiPort 端口将被系统自动关闭，此时系统将会向 Csrss 发送一个类型为 LPC_PORT_CLOSED（0x5）的消息。在 CsrApiRequestThread 函数中处理该类型消息有两处代码，一处是从 0x75AA4769 开始的代码，如图 2 所示。

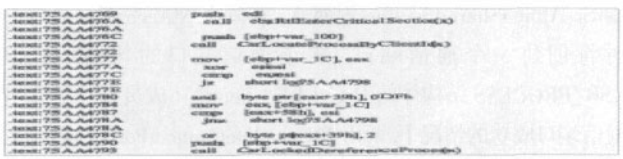

图 2 第一处处理 LPC_PORT_CLOSED 消息示意图

这一处的代码在系统关闭 ApiPort 端口时执行，此时进程 CSR_PROCESS 的线程计数已为 0，但引用计数为 1，进程中已经没有了活动线程，CsrLocateThreadByClientId 函数返回值为 0。接下来将调用 CsrLockedDereferenceProcess 函数，test.exe 对应的 CSR_PROCESS 的引用计数将减 1，变为 0，此时将会调用 CsrProcessRefcountZero 函数，进而调用 CsrRemoveProcess 函数将 test.exe 的 CSR_PROCESS 从 CsrRootProcess 链表中摘除。另一处是从地址 0x75AA4918 开始的代码，如图 3 所示。

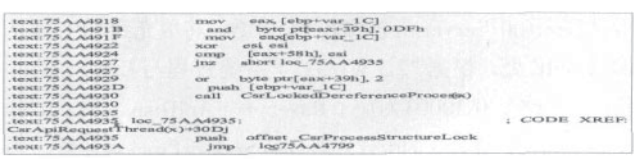

图3 第二处处理LPC_PORT_CLOSED消息示意图

这一处的代码在 CsrLocateThreadByClientId 函数返回值不为 0 的时候执行，此时进程 CSR_PROCESS 的线程计数不为 0，即进程中仍然存在有线程。

我们注意到在地址 0x75AA4780 和地址 0x75AA491B 处有一句代码：and byte ptr [eax+39h], 0DFh，这条语句的作用是将 CSR_PROCESS.Flags[1] 第 6 比特置为 0。漏洞利用思路，通过调试，并综合上文的分析，可以得出这样的结论：

1）任何用户的进程都可以调用 CsrClientCallServer 函数，设置第三个参数为 0x10003，可以调用 BaseSrvExitProcess 函数，在 BaseSrvExitProcess 函数返回后，进程 CSR_PROCESS 的引用计数为 2；2）紧接着在调用 CsrDereferenceThread 时，如果 CSR_PROCESS.Flags[1] 第 6 比特为 0，我们能调用 CsrLockedDereferenceProcess 函数、CsrDereferenceProcess 函数各一次，进程 CSR_PROCESS 的引用计数将减为 0；3）进程 CSR_PROCESS 的引用计数为 0 时，将调用 CsrRemoveProcess 函数从 CsrRootProcess 链表中摘除；4）关闭端口时，可以将 CSR_PROCESS.Flags[1] 第 6 比特置为 0。

我们只需要先执行第 4 步，然后再激活 BaseSrvExitProcess 函数，就可以实现进程 CSR_PROCESS 结构的摘除。具体的利用思路如下：

1）在进程正常运行的情况下，调用 NtSecureConnectPort 函数连接 ApiPort 端口，建立第二次连接；2）调用 CloseHandle 函数关闭第二次连接的 ApiPort 端口句柄，这样就能激活 CsrApiRequestThread 函数中 0x75AA4918 开始的代码，执行地址 0x75AA491B 处的代码：and byte ptr [eax+39h], 0DFh 后，进程的 CSR_PROCESS.Flags[1] 第 6 比特会置为 0；3）调用 CreateThread 新建一个线程，在新建的线程中调用 CsrClientCallServer 函数，第三个参数设为 0x10003，激活 BaseSrvExitProcess 函数，实现进程 CSR_PROCESS 结构的摘除；4）进入睡眠，等待管理员用户或者其他高权限用户登录，可以进行枚举窗口、记录键盘信息、截屏等一系列操作[3]。

## 3 结束语

本文提出的基于漏洞补丁，是对传统软件安全漏洞发掘方法的补充。与现有漏洞发掘方法相比，本文方法着眼于对厂商发布补丁的有效性及安全性分析，能发现更加隐蔽的安全漏洞，因此，对于发布补丁厂商有重大的参考价值。如何构建更为完整的漏洞发掘逻辑模型，实现模型静态分析和自动化的动态调试确定是软件发掘下一步研究方向之一。（责编 杨晨）

参考文献：
[1] 北大 BBS. 北京大学软件与微电子学院软件安全研究小组 [EB/OL]，http://www.pku-exploit.com/, 2010-11-14/2011-05-06.
[2] 程绍银，蒋凡，王嘉捷，张晓菲．一种自动生成软件缺陷输入的方法 [J]. 中国科学技术大学学报，2010,(06)：191-196.
[3] 谢新勤 . 基于访问控制列表的权限管理模型研究 [J]. 信息网络安全，2011，(05)：54-57.