doi 10.3969/j.issn.1671-1122.2011.07.010

IE 浏览器防攻击关键技术分析

毛宁祥[1]，文伟平[1]，傅军[2]

（1. 北京大学 软件与微电子学院，北京 102600；2. 厦门市信息技术服务中心，福建厦门 361012）

摘 要：软件漏洞带来的危害性日益增强。为了增加攻击者攻击的难度，Windows 操作系统逐渐从操作系统层面上提供对 DEP(数据执行保护) 和 ASLR（地址空间随机化）等安全机制的支持，其他应用软件可方便地应用这些保护机制。IE 浏览器也不例外。不过由于各方面的原因，IE 浏览器上的保护机制存在着各种各样的绕过方式。文章着重分析了其中的 DEP 和 ASLR 保护机制的原理及其绕过方式，并通过实例演示了堆扩散攻击和 ROP 编程。

关键词：IE 浏览器 ; 堆扩散 ; DEP; ASLR

中图分类号：TP393.08 文献标识码：A 文章编号：1671-1122（2011）07-0026-04

IE Browser the Attack Key Technology Research

MAO Ning-xiang[1], WEN Wei-ping[1], FU Jun[2]

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

2. *Xiamen Information Technology Service Center, Xiamen Fujian,* 361012,*China*)

Abstract: The harm caused by software vulnerabilities is increasing. To increase the difficulty of the attack launched by attacker, Windows gradually supports DEP and ASLR and other security mechanisms at the system level. Other software can apply these mechanisms easily. So IE browser can also apply these mechanisms to increase its security. However, there are still many methods which can bypass these mechanisms in IE browser. This paper provides details about DEP and ASLR protection mechanisms and also gives examples which demonstrate how Heap Spray and ROP attack happen.

Key words: Internet Explorer; Heap Spray; DEP; ASLR

0 引言

为了便于第三方开发者开发更多的互联网应用，浏览器逐渐加入了各种客户端开发技术和组件技术。IE7 及其之后的浏览器在安全方面作出了大量技术改进。DEP、ASLR、沙盒技术、插件过滤等技术的逐渐应用极大的提升了浏览器及运行在其上的互联网应用的安全性，给攻击者造成了很大的障碍。本文主要介绍了 DEP、ASLR 安全保护机制的原理，并对其绕过方式给出了实例分析。

1 相关工作

文献 [1] 首先全面的总结了 Windows Vista SP1 及之前版本的 Windows 系统下的各种安全机制的原理和实现细节。接着探讨了这些安全机制在原理和实现细节上的局限性，总结 / 提出了一些绕过这些安全机制的理论。最后通过实际案例验证了前面提到的部分理论，并探讨了在现实环境中产生稳定可用攻击代码的技巧。文献 [2，3] 比较详细的介绍了指针推断技术（Pointer Inference）和由堆扩散技术（Heap Spray）发展而来的 JIT 扩散（JIT Spray）技术。文献 [4-6] 分别探讨了绕过 DEP 保护机制的几种方式，本文对这些绕过方式进行了总结。文献 [7] 详细地讲述了 ROP 的构造过程及 DEP 的绕过方式。文献 [8] 总结了几种较新的绕过 Windows 浏览器内存保护的方法。最后，本文参考了文献 [9-11] 中对 MS10-002 漏洞原理的部分讲述。

收稿时间 2011-06-10

作者简介 毛宁祥（1986-），男，湖南，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，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；傅军（1978-），男，湖南，工程师，主要研究方向：办公自动化、数据安全等。

2 IE 浏览器中的相关安全机制

2.1 DEP

DEP（数据执行保护）会阻止攻击者在非代码段执行其攻击代码。传统的攻击过程需要攻击者首先把攻击代码存放在系统的数据页面（如堆栈页面上），然后通过各种方式控制程序流，使程序流运行到攻击代码位置，最终执行攻击代码。当 DEP 启用后，由于保存攻击代码的数据页面已被 CPU 标记为不可执行，那么当程序流到达此位置时，就会触发系统异常而导致程序中止。

完整的 DEP 功能需要 CPU 的支持。对支持 DEP 保护机制的 CPU，CPU 的内存管理单元会使用指向虚拟内存页面的 PTE 上的一位来标记其页面的可执行属性，即 NX 属性。对于不支持 DEP 保护机制的 CPU，即软件 DEP，DEP 提供的功能等同于 SafeSEH，此时提供缓冲区保护，但不能阻止代码在数据页上执行。

2.2 ASLR

ASLR，也就是地址空间布局随机化，会阻止攻击者调用 / 跳转到固定位置执行代码。传统的攻击过程往往需要攻击者硬编码系统中 jmp/call esp 类指令的位置作为溢出返回地址，当程序流执行到返回地址时，会跳转到 jmp/call esp 类指令，接着又会返回到栈上来执行攻击代码。ASLR 使得系统不再有固定的程序位置，这就使得攻击者难以控制程序的执行流程。硬编码系统中特殊指令的地址（如 jmp/call esp 类指令地址）、栈地址等常常会使攻击者的攻击代码不再有效。

在 Windows7 下，系统会随机化 PE 映像文件（EXE 或者 DLL）的加载基地址、进程的栈基址、进程的堆地址、PEB/TEB 地址。文献 [1] 中给出的加载映像基地址的未公开函数 MiSelectImageBase 部分伪代码如图 1 所示，该函数位于 ntoskrnl.exe 中，从中可以看出加载映像基地址的随机化方法。当程序启动后，将可执行文件映射到内存时，系统将在原来映像基址的基础上加上偏移 Delta，该 Delta 由取自 RDTSC counter 的 8 位随机数变换而来并保证变化后的所有可能的值的概率相同。

```
if(Int_header->Characteristics& IMAGE_FILE_DLL) == 0)
{
RelocateExe:
    unsigned int Delta=((RDTSC>>4)% 0xFE)+1)* 0x10000;
    dwImageSize=image size rounded up to 64KB
    dwImageEnd=dwImageBase+dwImageSize;
    if (dwImageBase>=MmHighestUserAddress||
      dwImageSize>=MmHighestUserAddress||
      dwImageEnd<=dwImageBase||
      dwImageEnd>=MmHighestUserAddress)
        return 0;
    if (arg.0->dwOffset1+Delta==0)
        return dwImageBase;
    if (dwImageBase+Delta){
        dwNewBase=dwImageBase.Delta;
    }
    else {
        dwNewBase=dwImageBase+Delt
        if (dwNewBase<dwImageBase|| dwNewBase+ImageSize>MmHighestUserAddress) {
                dwNewBase+ImageSize<dwImageBase+ImageSize)
                return 0;
    }
    return dwNewBase;
}
```

图1 MiSelectImageBase部分伪代码

对于需要启用 ASLR 保护的特定应用程序，在使用 VS 系列的编译器编译时需要加入 /DYNAMICBASE 编译选项，该选项同样作用于生成的 PE 映像文件的文件头（_IMAGE_OPTIONAL_HEADER），在其中加入 IMAGE_DLL_CHARACTERISTICS_DYNAMIC_BASE 标志。程序运行后，系统会检查该标志来判断该应用程序是否兼容 ASLR。

3 IE 浏览器中安全机制的突破方式

3.1 堆扩散

堆是 Windows 系统中一种特殊的内存存放机制，一般用于存放动态数量和大小的数据变量，以及占用内存空间超过栈大小的大尺寸对象等。Windows 系统在每个程序开始执行时都预先给它分配一个堆，用于存放进程初始数据，程序在运行时还可以根据需要申请别的堆。由于堆内存空间是有限的，因此当程序重复分配大量的堆内存空间后，会在接下来的某个时刻重复利用先前已释放的堆，这样这个被释放的堆就可以被攻击者所控制，存放攻击者需要的攻击数据，如果程序接下来再次访问这个被释放的堆，就会使得堆中的字节序列被执行。另一方面，反复申请同样大小的堆内存块后，会使得返回的堆地址呈现出一定的规律性，这就会造成内存地址信息泄漏。堆扩散常常需要经历分配大量堆空间 - - - 重用堆空间——访问已释放堆地址三个步骤。首先，分配大量堆空间后才会在接下来有机会重用之前已释放的堆空间，只有在重用已释放的堆并存放攻击者的攻击代码后，才会在接下来程序访问已释放的堆地址时执行攻击者的攻击代码。堆扩散攻击通常需要消耗大量的堆内存，这也是堆扩散最初的由来。为了让堆扩散攻击有效，程序必须事先尽可能多的消耗掉可用堆，这样才会使得接下来的堆分配会重复利用先前释放的堆。此外，堆扩散最终的目的是让攻击数据执行，因此，实践中需要重复申请堆，并将攻击数据填充到堆中，通过重复填充一块又一块内存空间的手段来提高命中率，确保溢出后让攻击数据得以执行。

在介绍堆扩散攻击实例之前，我们先看看从堆扩散技术发展而来的 JIT 扩散技术（JIT Spray）的原理。

文献 [2，3] 比较详细地介绍了 JIT 扩散技术。JIT 扩散技术类似于堆扩散，攻击者同样在堆中分配很多页面，不过此时分配的页面都具有可执行属性。将攻击代码所用到的字节序列喷射到这些页面后，再通过迫使 JIT（Just-In-Time）编译器解释这些字节序列，从而导致攻击代码执行。

我们以 2010 年的极光（Aurora）漏洞（MS10-002）来演示堆扩散攻击的步骤和有效性。我们重点演示堆扩散的攻击过程和漏洞的验证过程，因此对漏洞原理的静态分析没有给出详细步骤，只给出分析结果。下面的测试是在 XP SP2 上的 IE6 上完成的，IE 版本为 6.0.2900.2180。下面是最初的测试代码。

图2 极光漏洞POC

将图 2 中的代码保存到以 HTML 结尾的文件中。在浏览器中运行后双击，会发现 IE 浏览器会异常退出；如果用 WinDbg 附加到 IE 进程后再打开此 HTML 文件，则会出现异常，此时可用 u 命令显示异常位置函数的反汇编代码以及用 kb 命令查看栈回溯情况，如图 3 所示。

图3 访问违例异常（Aurora测试）

从栈回溯中的 mshtml!CWindow::FireTimeOut 等函数可以看出，异常是在定时器例程中触发的，因此可以肯定此时的栈回溯并没有被破坏。从此处的反汇编代码及 ecx 所指位置中的数据可以看出，代表 CElement 对象的指针 ecx 所在内容已经被覆盖，因此随后调用虚函数表 +34h 偏移处的函数指针也会指向错误的位置，从而导致访问违例异常。

此处发生异常由 CElement 对象指针指向错误的位置引起，我们需要了解 CElement 对象指针是怎么来的。查看 CEventObj::GenericGetElement 的反汇编代码可以知道，该函数会首先调用 CEventObj::GetParam 函数将 EVENTPARAM 指针参数保存到局部变量 ebp+var_8 中。随后调用 CElement::GetDocPtr，也就是最终发生异常所调用的函数，此时的 CElement 对象指针 ecx 指向局部变量 ebp+var_8 保存的数据，因此如果 ebp+var_8 所保存的数据所代表的位置中的数据已经被覆盖，那么此时的 CElement::GetDocPtr 调用就会发生异常。现在再来看看 CEventObj::GetParam 是怎么获得 EVENTPARAM 指针的，反汇编代码如图 4 所示。从这段代码可以看出，CEventObj::GetParam 会将 CEventObj 对象偏移 +18h 处的 EVENTPARAM 对象的地址作为返回值。因此最终导致出现访问违例的 CElement 对象指针保存在 CEventObj 对象 +18h 偏移处的 EVENTPARAM 对象中。此时我们需要知道 CEventObj 是什么时候被创建的。查看调用关系可知，CEventObj 需要在 CEventObj::Create 函数中创建，而该函数又被 CDocument::createEventObject 创建，此函数是跟 JavaScript 代码中的 document.createEventObject 对应的。CDocument::createEventObject 所做的主要工作是创建 CEventObj 对象并拷贝传入的 CEventObj 对象中的 EVENTPARAM 参数。进一步逆向 CImgElement 可知，CImgElement 会在对象 +10h 偏移处保存有所附属的 CTreeNode 指针，而 CTreeNode 对象也会保存有 CImgElement 对象指针，而 CEventObj 中的 EVENTPARAM 参数正保存有 CTreeNode 指针。进一步分析得到的漏洞原因是这样的：CDocument::createEventObject 中创建的 CEventObject 复制传入对象的 EVENTPARAM 参数时由于忘记增加所指向的 CTreeNode 对象的引用计数，随后 CTreeNode 被垃圾回收器所回收，而后引用 CTreeNode 对象及其所保存的 CImgElement 指针，不再是有效的对象（指针）。

图4 CEventObj::GetParam反汇编代码

3.2 绕过DEP

由于传统的攻击方式最终都需要在数据页面上执行代码，因此随着 DEP 的加入，传统的缓冲区溢出攻击便不再有效。尽管如此，ret2lib 以及发展而来的 ROP 攻击可以在一定范围内有效攻击 DEP。ROP 攻击所用到的 ROP 序列是一小段指令序列，并且以 ret/jmp/call 类指令结尾，这类短指令的一个重要特点就是执行完后会返回到栈上继续取下一个地址，然后跳到下一个 ROP 序列继续执行。另外，由于 ROP 序列属于代码段，因此是可执行的，不会触发 DEP 保护异常。ROP 攻击以栈为轴心，每执行一段 ROP 序列后，又会返回到栈上，就这样，通过反复执行不同的 ROP 序列，改变栈上的参数设置，构造一个伪造的栈，随后调用关闭 DEP 保护的函数，关掉进程的 DEP 保护，使得攻击代码可以在栈上运行。

文献 [7] 中提到，Window下绕过 / 改变 DEP 设置的方法主要有如下 4 类：执行命令，比如 WinExec 函数，这是一种典型的 ret-to-libc；在运行 shellcode 之前改变当前进程的 DEP 设置，比如 SetProcessDEPPolicy 和 NtSetInformationProcess 函数，允许你改变当前进程的 DEP 策略，因此你能从栈上执行你的 shellcode；将包含你的 shellcode 的页面（例如栈）标记为可执行，VirtualProtect 函数可以改变给定内存页的访问保护级别；将数据拷贝到可执行区域，然后跳到那里。可以先用 VirtualAlloc 或者 HeapCreate/HeapAlloc 分配一段可执行属性的内存区域，然后将栈上的攻击代码拷贝到此位置，也可以直接用 WriteProcessMemory 拷贝攻击代码到指定的可执行内存位置。

需要注意的是，这些函数并非总是有效。比如在 Windows7 上，如果进程设置了永久 DEP 标志，调用 SetProcessDEPPolicy 就会失败。

ROP 攻击的重点就在于找到用于设置其栈上参数的 ROP 序列。接下来我们以 ms07-17 漏洞为例，演示 ROP 序列的构造过程，以及演示绕过 ROP 机制的方法，使用的是 WinExec 函数。测试系统是 Windows SP2，IE 浏览器版本为 6.0.2900.2180。

user32.dll 中的 LoadAniIcon 读取光标文件时，会先根据

不同的块标志跳转到不同的代码中进行处理。对于 anih 块，读取第一个 anih 块时，会对其长度是否为 24h 进行严格的检查，但之后则不再检查长度而直接将块的数据复制到栈中，从而导致缓冲区溢出。

经过对 LoadAniIcon 的进一步分析，保存 anih 块的局部变量距离返回地址的偏移为 +50h。因此构造超过 +50h 长度 anih 块即可覆盖返回地址。

我们以系统自带的 sizens.ani 光标文件 (%SystemRoot%\Cursors 目录)，演示如何增加新的 anih 块。

要增加一个新的 anih 块，就要先添加 anih 四个字符的块标志，接着 4 个字节表示后面块数据的长度，最后就是 AniHeader 的内容。在此，我们取数据长度的值为 54h，因此最后的 4 个字节会覆盖返回地址。另外注意，添加了新的 anih 块后，要修改 ani 文件偏移 +4 处的长度（我们这里修改为 386h）。接下来，我们用 WinDbg 附加到 IE 进程看程序返回地址是否为 0xcccccccc，注意观察 esp 寄存器所指位置。

对于传统的缓冲区溢出，此后攻击者会将覆盖的返回地址替换为系统中 jmp esp 的地址，然后再填充 14h 字节的数据，最后跟上 shellcode。不过如果系统开启 DEP，程序就会发生异常，如果我们打开了 DEP 保护，并且将 anih 块修改为相关数据（其中 77f11678 是系统中一个 jmp esp 的地址）。用 WinDbg 运行程序会看到相关结果，此时程序无法继续运行下去。

3.3 绕过ASLR

绕过 ASLR 有如下几种方法 ：

1) 暴力方式。系统 DLL 只对基址的 8 比特进行了随机化，攻击者有 1/28 的几率得到正确地址 (PEB 基址的随机化只用到了 4 比特)。当溢出只发生在一个线程中且不会造成整个进程的崩溃时，可以用暴力猜测的方法不断尝试不同的跳转地址，此时从外部可能无法获得攻击的任何迹象。

2) 信息泄漏。这里的信息泄漏主要指的是内存地址的泄漏。分析 ASLR 机制可知，程序中的相对地址是不变的。因此如果能够读出程序中某个变量的内存地址，就可通过固定的偏移寻找到该模块的基地址，然后就可以获得模块中的其他地址。

前面阐述的堆扩散攻击就是信息泄漏的一个典型例子。当我们可以分配大量的堆空间时，返回的堆地址就会呈现出一定的规律性，此时我们就可以使用该地址作为跳转地址。

文献 [2] 中阐述的 ActionScript 脚本中的技术也是信息泄漏的典型实例。Pointer Inference 技术通过和浏览器插件的“ 正常 ” 交互来获得插件脚本内部对象的内存地址。ActionScript 脚本是一种动态类型语言，其变量的类型不是在运行前就已确定，而是将变量的类型和值同时存储，同时提供运行时函数来检查和比较对象类型。其保存对象引用（指针）和普通整数都用整数保存，只是整数的低位用来保存类型，其他位用来保存值。因此可以将整数和攻击所用的 shellcode 字符序列一同插入 ActionScript 字典对象中，插入的整数和 shellcode 字符序列的位置保持一定的关系，通过这种关系可以得到 shellcode 字符序列的低位地址。

3) 寻找不兼容 ASLR 保护机制的模块。寻找不兼容 ASLR 保护机制的模块。由于这些模块不兼容 ASLR 保护机制，因此该模块将被加载到固定位置，从而可以从这些模块中寻找固定的跳板地址。这也是实际攻击中常用到的方法。相对于其他方法，这种方法通常简单并且有效，因为当前仍然存在很多浏览器第三方插件不兼容 ASLR 保护机制。对于上一节的 MS07-17 漏洞，如果需要绕过 ASLR 保护机制，则只需要寻找不兼容 ASLR 保护机制的模块，从这些模块中寻找可用跳板地址。如果需要关闭 DEP，则可以像上一节中一样构造 ROP 序列关闭 IE 进程 DEP，只是 ROP 序列中用到的地址只能来自这些不兼容 ASLR 保护机制的模块。

4 结束语

本文首先分析了 IE 浏览器的两种主要安全保护方式 DEP 与 ASLR 的原理，接着围绕这两种保护方式分析了其绕过的方法，并以 MS10-002（极光漏洞）、MS07-017（光标漏洞）为例演示了堆扩散、ROP 攻击方法。

未来的研究工作包括以下几个方面的内容 ：1) 分析新的安全漏洞在各种安全机制启用情况下的可能攻击方式 ；2) 研究如何改进现有的安全机制。（责编 杨晨）

参考文献:
[1] Sotirov A, Dowd M. Bypassing browser memory protections. In Blackhat,USA, 2008.
[2] Dion Blazakis. INTERPRETER EXPLOITATION: POINTER INFERENCE AND JIT SPRAYING. http://www.semantiscope.com/research/BHDC2010/BHDC-2010-Paper.pdf.
[3] Alexey Sintsov. Writing JIT-Spray Shellcode for fun and profit. http://dsecrg.com/files/pub/pdf/Writing%20JIT-Spray%20Shellcode%20for%20fun%20and%20profit.pdf.
[4] Bernardo Damele A. G. DEP bypass with SetProcessDEPPolicy(). http://bernardodamele.blogspot.com/2009/12/dep-bypass-with-setprocessdeppolicy.html.
[5] skape, Skywing. Bypassing Windows Hardware-enforced Data Execution Prevention. http://uninformed.org/index.cgi?v=2&a=4.
[6] Spencer Pratt. Exploitation With WriteProcessMemory. http://packetstormsecurity.org/papers/general/Windows-DEP-WPM.txt, 2004.
[7] corelanc0d3r 著 . dragonltx 译 . Exploit 编写系列教程第十篇 - 用 ROP 束缚 DEP- 酷比魔方 . In bbs.pediy.com,2010.
[8] Chen XiaoBo, Xie Jun. 绕过 windows 7 浏览器内存保护 . Xcon 2010.
[9] Dino Dai Zovi. Operation Aurora Exploit Analysis.
[10] Geoff Chappell. Operation Aurora http://www.geoffchappell.com/viewer.htm?doc=notes/security/aurora/index.htm.
[11] 轩辕小聪 .[原创 + 整理]IE 极光漏洞的原理探秘 http://bbs.pediy.com/showthread.php?t=105899.