ASLR 绕过启示录

最近许多 APT 攻击使用了新的技术绕过地址随机化（ASLR），虽然地址随机化是当前操作系统最有效的防护机制之一，但还不够完善。在过去一年里，我们发现了以下几个有趣的绕过 ASLR 的技术：

0x1: **使用未开启 ASLR 的模块**

0x2: **修改 BSTR 长度/null 结束符**

0x3: **修改数组对象**

注：标准 BSTR 是一个有长度前缀和 null 结束符的 OLECHAR 数组。

下面详细讲解这些技术。

0x1:未开启地址随机化的模块

加载一个未开启地址随机化的模块是最容易也是最流行的绕过地址随机化保护的方法，在 IE 0day 中最常使用的两个未开启地址随机化的模块是：MSVCR71.DLL 和 HXDS.DLL。

JRE 1.6.x 包含了一个老版本的 C 运行库 MSVCR71.DLL，编译时没有加上/DYNAMICBASE 编译选项。默认情况下,在 win7+ie8 和 win7 +ie9 下，这个 DLL 会以固定地址加载到 IE 的进程中。

MS Office 2010/2007 中的 HXDS.DLL 编译时也没加上相关选项，该技术最先是在 http://www.greyhathacker.net/?p=585提出，也是当前在 IE 8/9 + win7 环境下使用最频繁的过地址随机化的方法，当浏览器加载一个带 try location.href = 'ms-help://' 语句的页面时，这个 DLL 就会被加载。

下面的0day 利用里面，至少使用了其中一种技术绕过地址随机化：

CVE-2013-3893, CVE2013-1347, CVE-2012-4969, CVE-2012-4792

使用条件：通过未开启 ASLR 模块来绕过 ASLR 的方法要求被攻击机器运行了较老的软件如 JRE 1.6 或者 Office 2007/2010 ，升级到最新版的 JAVA/OFFICE 就可以这种类型的攻击。

0x2:修改 BSTR 长度/null 结束符

这个技术是 Peter Vreugdenhil 在 2010 Pwn2Own IE 8 上利用的，它只适用于可以覆写内存的漏洞，例如缓冲区溢出，任意地址写 和 对可控指针所指向的内存内容执行增加或减少操作的漏洞。

任意地址写入 类型的漏洞不能直接控制 EIP，多数时候，这种类型的漏洞利用会覆写程序重要的数据如函数指针来达到执行代码的目的。对于攻击者来说，他们可以随意修改 BSTR 的长度，然后利用 BSTR 去控制超出边界的内存，这样就可以泄漏内存地址，精确找到可以构造 rop 的 dll 。一旦通过这种方式绕过了地址随机化，也可以使用同样的漏洞控制 EIP。

为数较少漏洞可以用来修改 BSTR 的长度。例如，一些漏洞只能增加或减少指针一两个字节，这种情况下，攻击者可以修改字符串的 null 结束符使该字符串和下一个对象连接起来，这样下一个对象就成为该字符串的一部分，而在这个对象中，一般可以找到和 DLL 基址相关的信息。

CVE-2013-0640

在 Adobe XFA 0day 利用中，使用了上述技术找到了 AcroForm.api 的基址，动态的构造了 rop 链绕过 ASLR 和 DEP ,在这个漏洞里，攻击者可以将可控指针指向的内容减一，然后从虚函数表中调用函数。

```
mov     ecx, [esi+44h]
test    ecx, ecx
jz      AcroForm!PlugInMain+0xa31a7
dec     dword ptr [ecx+4]
jnz     AcroForm!PlugInMain+0xa31a7
mov     eax, [ecx]
push    ebx
call    dword ptr [eax]
```

想像一下在 Dec 操作之前的内存布局，如下：

[string][null][non-null data][object]

dec 操作之后（在我测的时候，减少了两次），内存布局变成了：

[string][\xfe][non-null data][object]

更多细节请查看 immunityinc’s blog.

使用条件：

这个技术通常需要多次写入来泄漏需要的信息，攻击者需要非常精准的设计堆的布局，保证 length 字段被覆写而不是内存中其他的对象。从 IE 9 开始，微软使用了 Nozzle（http://research.microsoft.com/pubs/81085/usenixsec09b.pdf）来防止堆喷/风水，所以有时攻击者需要使用 VBArray 技术来精准布局堆。

修改数组对象

数组对象长度修改和 BSTR 长度修改相似：他们都需要“好用”的漏洞。从攻击者来看，一旦数组长度被修改，攻击者既可以任意读写内存或者控制程序流程，到达代码执行的目的。

下面是使用该技术的0day 利用。

CVE-2013-0634

这个漏洞是 Flash player 在处理 regex 时的堆溢出，攻击者覆写了 *Vector.<Number>* 对象的 length 属性，然后就可以读取更多的内容，获取 flash.ocx 的基址。

这个漏洞利用流程如下：

Step1:通过申请下面所示的对象来设置连续的内存布局：**错误!**

```
obj = new Vector.<Object>(16);
obj[0] = new RegExp(_loc_24, "");
obj[1] = new <Number>[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1];
obj[2] = new <Number>[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1];
obj[3] = new <Number>[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1];
obj[4] = new <Number>[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1];
....
```

Step2:如下所示释放掉上面申请的对象中序号为1的<Number>对象。

```
obj[1] = null;
```

Step3:申请一个 RegExp 对象，这次申请会重新使用刚才释放掉的位置。

```
boom = "(?i)()()(?-i)||||||||||||||||||||||||";
var trigger = new RegExp(boom, "");
```

然后，当触发漏洞后，会覆写掉 obj[2]里的 Vector.<Number>对象的长度属性，使其变大，攻击者就可以通过 obj[2]来读写一个大范围的内存空间来定位 flash.ocx 的基址，然后覆写掉虚表达到代码执行的目的。

CVE-2013-3163

这个漏洞是 IE *CBlockContainerBlock 对象 UAF，利用程序和* CVE-2013-0634很像，但是更复杂一些。

这个漏洞使用 OR 指令修改任意内存内容，大致如下：

```
or dword ptr [esi+8],20000h
```

利用流程如下：

首先，使用 Vector.<uint>对象填充堆。

```
this.s = new Vector.<Object>(0xc0c0);
while (loc1 < 0xc0c0)
{
        this.s[loc1] = new Vector.<uint>(0x1000 / 4 - 16);
        this.s[loc1][0] = 0xC0BFFF8;
        this.s[loc1][2] = 0xC0BFFF8;
        this.s[loc1][4] = 0xC0BFFF8;
        this.s[loc1][5] = 0xC0BFFF8;
        ++loc1;
}
```

在填充之后，这些对象以对齐的方式存储在一个固定的地址。

像是这样：

```
0:020> d 0c0c0000
0c0c0000  f0 03 00 00 00 20 d6 09-f8 ff 0b 0c 00 00 00 00  ..... ..........
0c0c0010  f8 ff 0b 0c 00 00 00 00-f8 ff 0b 0c f8 ff 0b 0c  ................
```

第一个 dword 0x03f0是 *Vector.<uint>* 对象的长度，黄色标记是我们填充的值。如果攻击者将 esi+8 指向 0x03f0,当 or 指令执行过后，变为0x0203f0.看到没，大多了!! 由于可控范围变大，可将紧接着的对象长度改为0x3F `this.s[i][0x1000 * j / 4 - 2] = 0x3FFFFFF0;`

```
0:014> d 0c0c1000
0c0c1000  f0 ff ff 3f 00 20 29 05-f8 ff 0b 0c 00 00 00 00  ...?. ).........
0c0c1010  f8 ff 0b 0c 00 00 00 00-f8 ff 0b 0c f8 ff 0b 0c  ................
```

然后，攻击者就可以控制整个 IE 进程的内存空间了，NB！地址随机化也就没用了，因为可以直接从内存中获取到 kernel32/NTDLL 的基址。通过从代码段动态寻找 stack pivot gadgets，从 IAT 定位到 ZwProtectVirtualMemory，就可以构造 rop 链修改内存属性绕过 DEP.

```
if ((this.s[me][(i - base) / 4] & 0xFFFF) != 0xC394)
{
	if ((this.s[me][(i - base) / 4] & 0xFFFF00) != 0xC39400)
	{
		if ((this.s[me][(i - base) / 4] & 0xFFFF0000) != 0xC3940000)
		{
			if ((this.s[me][(i - base) / 4] & 0xFF000000) == 0x94000000
&& (this.s[me][(i - base) / 4 + 1] & 0xff) == 0xC3)
			{
				xchgeaxesp = i + 3;
				break;
			}
		}
		else
		{
			xchgeaxesp = i + 2;
			break;
		}
	}
	else
	{
		xchgeaxesp = i + 1;
		break;
	}
}
```

通过精确的内存布局，攻击者又申请了一个包含 flash.Media.Sound()对象的 Vector.<object>，使用已经被修改过的 Vector.<uint>对象去寻找 sound 对象，然后覆写它的虚表指向 rop 链和 shellcode.

CVE-2013-1690

这是 FireFox 的 DocumentViewerImpl 对象的 UAF 漏洞，可以任意内存写入一个 word 值0x0001.

上面的代码中，所有以 ‘m’开头的变量都是从我们可以控制的对象中读取的。如果可以设置对象，

```
6916fa2f 5e              pop     esi
6916fa30 66899fe0000000  mov     word ptr [edi+0E0h],bx   ds:0023:11fa0023=0001
6916fa37 5b              pop     ebx
6916fa38 c9              leave
6916fa39 c20400          ret     4
```

在这个利用中，攻击者尝试使用 *ArrayBuffer* 来精确布局堆。下面的代码中，每一个 var2 中的 *ArrayBuffer* 原始大小为0xff004

```
var var1=0xB0;
var var2 = new Array(var1);
var var3 = new Array(var1);
var var4 = new Array(var1);
var var5=0xFF004;
for(var j=0;j<var1;j++)
{
        if( j<var1/8 || j==var1-1)
        {
                var tabb = new Array(0x1ED00);
                var4[j]=tabb;
                for(i=0;i<0x1ED00;i++)
                {
                        var4[j][i]=0x11559944;
                }
        }
        var2[j]= new ArrayBuffer(var5);
}
```

触发漏洞后，ArrayBuffer 长度增加为0x010ff004.也可通过在 JS 中比较 byteLength 来定位被修改了长度的 ArrayBuffer ，然后攻击者可通过这个 ArrayBuffer 来读写内存了。这次，攻击者选择从 SharedUserData(0x7ffe0300) 泄漏 NTDLL 的基址,然后手动硬编码偏移来构造 ROP.

CVE-2013-1493

这个漏洞是 JAVA CMM 的整数溢出漏洞，可以覆写数组长度。在漏洞利用中，数组长度可以到达 0x7fffffff，攻击者可以搜索 *securityManager* 对象然后设置它为 null 来绕过沙盒。

这个方法相较于覆写函数指针，处理地址随机化和数据执行保护以到达代码执行更加有效。

数组对象修改技术比其他的要好一些。因为 Flash ActionScript vector 技术，堆填充一点都没受影响。只要你有内存覆写漏洞，很轻松的就可以利用。

几种技术细节对比对比：

绕过 ASLR 技术	优点	使用限制	CVE	备注
修改数组对象	高级利用技巧 扩大数组长度后可以任意内存读写	需要有内存覆写漏洞 需要被攻击者安装 flash	CVE-2013-1690	FireFox uaf 可以覆写一个用户控制的指针修改 ArrayBuffer 长度到达任意内存读写
			CVE-2013-3163	IE UAF 。攻击者可以修改 Vector.<Uint>的长度，然后寻找构造 rop 的指令
			CVE-2013-1493	JAVA 整数溢出，覆写数组对象长度，到达任意内存读写
			CVE-2013-0633	FLASH PLAYER 正则表达式缓冲区溢出，修改 Vector.<Number>对象到达任意内存读写
修改 BSTR null 结束符	高级利用技巧 能泄漏部分内存绕过地址随机化	内存覆写漏洞 不同的 Adobe Reader 需要硬编码不同的 rop	CVE-2013-0640	Adobe Reader 未初始化内存使用 修改 BSTR null 结束符泄

		偏移		露内存
使用未开启地址随机化的模块	容易利用 不太需要技巧	特定版本的软件	CVE-2013-3893	IE UAF 从 hxds.dll 中构造 rop 绕过地址随机化和数据执行保护。 CVE-2012-4792 IE UAF JRE 1.6 和 OFFICE 2010 未开启地址随机化

总结

绕过地址随机化是现在漏洞利用最基本的需求，之前利用 MS OFFICE 未开启 ASLR dll 来绕过，但是微软在最新的操作系统和浏览器里面做了限制。之前的技术将淘汰也更容易被检测，攻击者需要更先进的技术，对于可以覆写内存的漏洞，结合 Vector.<uint> 和 Vector.<object>将更加可靠和灵活，从只能写一个字节到大范围读写内存将非常容易，而且适用于各种操作系统，应用程序，语言版本。

许多研究者公布了对于绕过地址随机化的研究，像 Dion Blazakis 的 JIT spray and tombkeeper 的 LdrHotPatchRoutine 技术。但是至今还未见使用，可能是因为这些技术是通用的对抗 ASLR 的方法，所以一经公布，很快就被修补了。

但是没有一种通用的办法修补特定的漏洞利用，希望将来有更多的0day 使用相同的或者更加高级的技巧，我们可能需要在 OSs 里面加入新的防护措施和安全产品来对抗0day 攻击。