基于代码覆盖的浏览器漏洞利用攻击检测方法

孟 辰
(同济大学电子与信息工程学院 上海 201804)

摘 要 根据漏洞利用攻击的概念验证,使用WinDbg逆向工程找出该类攻击的特征,并根据该特征编写检测代码。而后将检测代码封装至DLL中,并通过远程线程方式将DLL注入网页浏览器。被注入的DLL会以代码覆盖的方法拦截浏览器的API,使浏览器跳转到检测代码。根据浏览器打开网址时检测代码的返回值,来判断该网址是否包含利用该漏洞进行攻击的网页木马。通过将该技术部署于众多虚拟机中,批量检测网页,来向杀毒软件公司以及搜索引擎等提供高可信度的挂马网页黑名单。

关键词 漏洞利用攻击,概念验证,代码覆盖,网页木马,逆向工程,API拦截,DLL注入

Web Browser Vulnerability Exploitation Attack Test Technology Based on Code Overriding

MENG Chen
(School of Electronics and Information, Tongji University, Shanghai 201804, China)

Abstract Based on the proofs of concept for vulnerability exploitation attack, we can find the characteristic of the attack through reverse engineering using WinDbg and write test code according to the characteristic. We then encapsulate the test code into a DLL and inject the DLL into Web browser through remote thread. The injected DLL will hook browser's API by means of code overriding, so that the browser will jump to the test code. By visiting a website and acknowledging the return value of the injected code to judge whether the webpage contains trojan horse using the vulnerability. By deploying this technology into numerous virtual machines, we can analyze webpages in bulk, and then provide high-quality webpages blacklist to antivirus software companies and search engines.

Keywords Vulnerability exploitation attack, Proof of concept, Code overriding, Webpage trojan horse, Reverse engineering, API hooking, DLL injection

漏洞是指计算机系统在硬件、软件、协议的具体实现或系统安全策略上存在的缺陷,它使攻击者能够在未授权的情况下访问或破坏系统[1]。随着微软的Windows操作系统越来越庞大,越来越复杂,漏洞也随之越来越多。而这些漏洞一旦被别有用心的人利用,造成的损失往往是无法弥补的。操作系统的漏洞已经被越来越多的病毒编写者所利用,遭受的攻击也越来越频繁,某种意义上来说,操作系统的漏洞已经成为孕育计算机病毒的温床[2]。

随着网络的普及,人们在日常生活中越来越频繁地使用网络,黑客通过网页挂马的方式对用户进行攻击的现象频频发生。而黑客惯用的伎俩就是根据IE等浏览器的漏洞[6],通过缓冲区溢出等方式导致恶意代码运行并获取用户的系统控制权[4]。虽然大部分被发现的安全漏洞都已经被解决,但问题是新的安全漏洞正以令人担忧的速度出现,而不是在减少。而且即使漏洞的补丁已发布,也总会有部分用户由于没有及时安装补丁而暴露在危险当中。

为了保障用户上网安全,目前各大杀毒软件常用的网页木马检测技术包括特征值检测技术、校验和检测技术、启发式扫描[4]等。传统的基于病毒特征码的反病毒技术,对于已知病毒的检测和杀灭非常有效,但对于已知病毒变种或未知病毒,其病毒库缺乏相应特征码,导致查杀效果不佳[9]。不仅如此,目前的网页木马为了防止被检测出而被层层加密[3]。倘若杀毒软件无法及时识别被加密的木马,而导致木马进驻内存并解码后才被检测出,就会使检测失败、系统中毒。

为了克服现有的网页木马检测技术的局限性,提出的基于代码覆盖的浏览器漏洞利用攻击检测方法,取得了更好的检测性能。通过在IE中注入DLL,以代码覆盖的方法拦截浏览器的API,使进程跳转到根据漏洞利用代码的特征编写的检测代码,可以在木马发起攻击前迅速完成检测。由于是基于攻击方式的特征进行检测,该方法不会受网页木马加密的影响。而且由于该方法是基于漏洞利用攻击的概念验证,只要攻击所针对的漏洞不变,该检测方法不会由于攻击方式的变化而失效。更重要的是,由于检测方式极富针对性,使得检测更为准确,大大降低了网页木马的误报率。

1 漏洞CVE-2008-4844的概念验证

CVE-2008-4844是IE7中存在的漏洞:通过IE7打开包含恶意编写的XML的网页,可以利用该IE7内存越界的漏洞,使用JavaScript脚本操作Shell code去执行任意代码。该漏洞的利用代码从2008年的下半年开始流传,并于当年的10月份前后被私人买卖,11月份流入了黑市。2008年12月9日开始被挂在网页上,作为网页木马向用户发起攻击,影响范围不断扩大。

针对该漏洞,微软于2008年12月10日发布了安全通报

孟 辰(1988—),男,硕士生,主要研究方向为信息安全,E-mail:mengchen1988221@gmail.com。

961051，告知用户相关漏洞的存在，并给予了一些缓解措施。时隔不久，微软便于17日发布了安全公告MS08-078，通过为IE打补丁的方式避免用户受到影响。该补丁的严重级别被列为紧急，影响的浏览器包含从IE5.01直到IE8 beta2。由此可以看出该漏洞的覆盖面之广，涉及的用户数量之多。

该漏洞发生在“MSHTML.DLL”模块之中，若以特定的方式构造网页，可以使IE在数据绑定的过程中出现无效的指针引用。漏洞被利用的过程是：IE会在数据绑定被调用时，创建一个包含绑定数据的数组对象。当数组对象中的一个绑定数据被释放时，数组的长度却没有做到及时更新，这就导致了IE调用被释放的数据。网上流传的网页挂马利用了XML的SRC字符串对象，占用了被释放的数据的内存，而SRC字符串对象中包含有恶意函数指针，这就导致了恶意代码的执行。

安全网站Security Focus中提供了很多概念验证，其中包括了供技术人员研究的漏洞利用网页实例。在本文中，我们使用其中的一个网页实例，该实例能够触发并利用该漏洞，但不会下载或运行病毒程序，而只是在漏洞利用的结尾打开Windows自带的计算器程序。

2 注入检测代码的设计与实现

2.1 调试环境的部署

将Security Focus中提供的代码复制到html文件中，构成一个本地的网页。使用WinDbg启动IE7，在打开的IE7中浏览该本地网页，然后开始逆向工程调试。Security Focus提供的代码地址为http://www.securityfocus.com/bid/32721/exploit。

2.2 注入位置的确定

通过逆向工程，发现在内存地址为7ea81de0的地方，存在call dword ptr [ecx+84h]指令：

```
7ea81dd6 0f84dd000000 je mshtml! CXfer::TransferFromSrc+0x111 (7ea81eb9)
7ea81ddc 8b08 mov ecx,dword ptr [eax]
7ea81dde 57 push edi
7ea81ddf 50 push eax
7ea81de0 ff9184000000 call dword ptr [ecx+84h] ds:0023:8b087e88=????????
```

但是[ecx+84h]所指向的内存地址却为空：

```
0:001> dd ecx+84h
8b087e88???????? ???????? ???????? ????????
```

使用Windbg的函数调用查看功能得知，这段代码属于模块mshtml.dll的。通过Windbg查看已加载的模块可知，mshtml.dll加载在内存中的位置为7e830000至7eb9f000范围内：

```
ModLoad:7c800000 7c91c000 c:\windows\system32\kernel32.dll
ModLoad:7c920000 7c9b4000 c:\windows\system32\ntdll.dll
ModLoad:7e1e0000 7e7a9000 c:\windows\system32\ieframe.dll
ModLoad:7e830000 7eb9f000 c:\windows\system32\mshtml.dll
```

查看此时寄存器ecx的值为8b087e04：

```
eax=0a0a0a0a ebx=00000000 ecx=8b087e04 edx=7e9086ed esi=052ee7a0 edi=052cff70
eip=7ea81de0 esp=0184fc98 ebp=0184fcc0
```

可以看到[ecx+84h]地址并不在7e830000至7eb9f000范围内，也就是说该地址越过了mshtml.dll所加载的内存空间。由此可以判断程序在此时发生了异常行为。而之前已经得知[ecx+84h]所指向的内存地址却为空，也就是说，被调试的网页试图调用这个空地址，这就与概念验证中提到的“IE调用被释放的数据”相吻合。至此，我们更加确信该位置即为所要查找的漏洞攻击特征。记录下异常行为发生的内存地址，我们开始编写检测代码。

2.3 检测代码的设计

检测代码的设计思路是：通过代码覆盖的方式，用CPU的一条JUMP指令覆盖内存中7ea81de0位置的指令，这条JUMP指令用来跳转到检测函数的内存地址。通过该方式拦截浏览器的API函数，使得覆盖后的程序不会再去执行call dword ptr [ecx+84h]，而是跳转到检测代码，如图1所示。检测代码如果确认[ecx+84h]地址并不在7e830000至7eb9f000范围内，即call dword ptr [ecx+84h]试图跳转到一个已经被释放的内存位置时，我们就可以断定打开的网页包含漏洞利用代码。

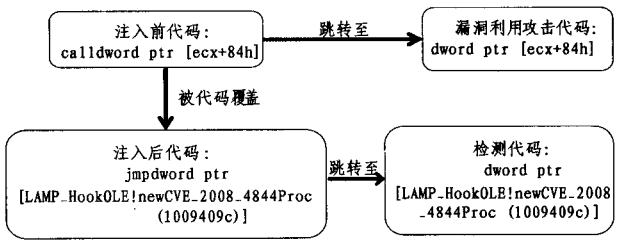

图1

注入前的代码：

```
7ea81dd6 0f84dd000000 je mshtml! CXfer::TransferFromSrc+0x111 (7ea81eb9)
7ea81ddc 8b08 mov ecx,dword ptr [eax]
7ea81dde 57 push edi
7ea81ddf 50 push eax
7ea81de0 ff9184000000 call dword ptr [ecx+84h] ds:0023:8b087e88=????????
```

注入后的代码：

```
7ea81dd6 0f84dd000000 je mshtml! CXfer::TransferFromSrc+0x111 (7ea81eb9)
7ea81ddc 8b08 mov ecx,dword ptr [eax]
7ea81dde 57 push edi
7ea81ddf 50 push eax
7ea81de0 ff259c400910 jmp dword ptr [LAMP_HookOLE!newCVE_2008_4844Proc (1009409c)] ds:0023:1009
```

注入后，原来的call指令被改写为JUMP指令，跳转到检测函数。

代码覆盖是通过如下方式实现的：

```
DWORD WINAPI GeneralHookProc (DWORD Address, char *szHookModule,DWORD HookNewFunc)
{
  DWORD dwResult=0;
  HMODULE hModule=NULL;
  hModule =::GetModuleHandleA(szHookModule);
  if (NULL == hModule)
  {
    hModule=LoadLibraryA(szHookModule);
    if (NULL == hModule) return dwResult;
  }
  DWORD dwOldPermission;
  dwResult=VirtualProtect((LPVOID)Address,
100,
            PAGE_EXECUTE_READWRITE,
```

```
            &dwOldPermission);
    *((char *)Address)=0xff;
    *((char *)Address+1)=0x25;
    *((DWORD *)((char *)Address+2))=HookNewFunc;
    return 0;
}
```

经先前调试得知检测代码注入的位置位于 mshtml. dll，在以上代码中，我们首先获取 mshtml. dll 在内存中的可写权限，修改后便可以在内存 7ea81de0 位置处实现代码的覆盖。在这里将其覆盖为：0xff25 HookNewFunc。0xff25 为 CPU 的 JUMP 指令，而 HookNewFunc 是检测函数的地址。通过这段代码，我们将原本的 call dword ptr [ecx+84h]指令覆盖为 jmp HookNewFunc，从而使浏览器进程跳转至检测函数。

检测函数：

```
__declspec(naked) int newCVE_2008_4844(void)
{
  __asm{
          pushad
          cmp [ecx+84h],0x7eb9f000
          ja Vuln
          cmp [ecx+84h],0x7e830000
          jb Vuln
          jmp real
    Vuln:
          lea eax,szV_newCVE_2008_4844
          push eax
          lea eax,szV_newCVE_2008_4844
          push eax
          call SendMessageToParent
          pop ecx
          pop ecx
    real:
        popad
        call dword ptr [ecx+84h]
        mov eax,dword ptr [esi+1Ch]
        jmp CVE_2008_4844_FuncPointer_H
    }
}
```

检测函数的检测方式不是直接查看[ecx+84h]地址所对应的内容是否为被释放的地址，而是判断[ecx+84h]是否在 0x7e830000 和 0x7eb9f000 范围之内。因为在 mshtml. dll 里面正常的函数调用通常都会发生在该范围内，若超出该范围，则可以断定发生了异常行为。在这个例子中，[ecx+84h]的值为 0x8b087e88，超出了范围。

在检测代码的 real 部分，我们做了栈的恢复处理。因为在漏洞利用代码中，位于 7ea81de0 的代码被执行了两次，第一次是合法的执行，第二次才是我们想要寻找的异常。为了使程序在第一次被检测后，不破坏原有栈的内容，我们需要进行一些恢复操作，包括：寄存器的恢复(pushad，popad)、被改写的指令的恢复。

在检测到漏洞利用后，检测代码会调用函数 SendMessageToParent，将检测信息发送给监听进程。

2.4 检测代码的注入

至此，我们已经完成了以代码覆盖的方法拦截浏览器的 API，并使得浏览器跳转到检测代码。接下来的工作就是将上面的代码封装至 DLL 中，并将 DLL 注入到 IE7 的进程中去。

注入代码到目标进程是实现拦截 API 很重要的一步。比较简单的方法是把要注入的代码写到 DLL 中，然后让目标进程加载这个 DLL[8]。向目标进程注入 DLL 的方法有很多，包括：使用注册表来注入 DLL，使用 Windows 挂钩来注入 DLL 以及使用远程线程来注入 DLL 等。我们这里采用的方式就是使用远程线程来注入 DLL。

从根本上说，DLL 注入技术要求目标进程的一个线程调用 LoadLibrary 来载入我们想要的 DLL。由于不能轻易地控制别人进程中的线程，因此这种方法要求在目标进程中创建一个新的线程。由于这个线程是我们自己创建的，因此可以对它执行的代码加以控制[7]。使用 Windows 的 CreateRemoteThread 函数在 IE7 进程中创建一个线程，该线程通过调用 LoadLibraryA 函数将先前封装的 DLL 加载至 IE7 的进程。实现的主要代码如下：

```
char strDllName[]="LAMP_HookOLE. dll";
bool InjectIntoIEProcess(DWORD Pid)
{
  HANDLE hIEProcess=NULL;
  hIEProcess=OpenProcess(
            PROCESS_CREATE_THREAD | PROCESS_VM_
OPERATION | PROCESS_VM_WRITE,
        FALSE,
        Pid);
  if (NULL == hIEProcess)
  {
    printf("Process Opened failure! \n");
    DWORD dwError=GetLastError();
    return hr;
  }
  LPVOID p_RemoteAddress=NULL;
  HANDLE p_InjectThread;
  DWORD ExitCode;
  p_RemoteAddress=VirtualAllocEx(hIEProcess,
                NULL,
                100,
                MEM_COMMIT,
                PAGE_READWRITE);
  if (! WriteProcessMemory(hIEProcess,
            p_RemoteAddress,
            (LPCVOID)strDllName,
            100,
            NULL))
      return hr;
FARPROC ptrLoadLib=GetProcAddress(GetModuleHandle
                    (TEXT("Kernel32. dll")),"Load-
                    LibraryA");
p_InjectThread =::CreateRemoteThread(hIEProcess,
                NULL,
                0,
                (LPTHREAD_START_ROUTINE)ptr-
                LoadLib,
                (LPVOID)p_RemoteAddress,
                0,
                NULL);
  return true;
}
```

(下转第 49 页)

ual[C]//Instruction set referenc. vol2,2006

[6] Dalton M,Kannan H,Kozyrakis C. Real-world buffer overflow protection for userspace & kernelspace[C]//SS'08:Proceedings of the 17th conference on Security symposium. UCSENIX Association,Berkeley,CA,USA,2008:395-410

[7] Shacham H. The geometry of innocent flesh on the bone:return-into-libc without function calls (on the x86)[C]//Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS). ACM,New York,NY,USA,2007:552-561

[8] Garfinkel T, Rosenblum M. A virtual machine introspection based architecture for intrusion detection[C]//Proc. Network and Distributed Systems Security Symposium. February 2003

[9] Grizzard J. Towards self-healing systems:re-establishing trust incompromised systems[D]. Georgia Institute of Technology, 2006

[10] Hund R,Holz T,Freiling F C. Return-oriented rootkits:Bypassing kernel code integrity protection mechanisms[C]//Proceedings of 18th USENIX Security Symposium. SanJose,CA,USA, 2009

[11] Seshadri A,Luk M,Qu N,et al. Secvisor:a tiny hypervisor to provide lifetime kernel code integrity for commodity oses[C]//Proceedings of twenty-first ACM SIGOPS symposium on Operating system sprinciples. ACM,NewYork,NY,USA,2007:335-350

[12] Krahmer S. X86-64 buffer overflow exploits and the borrowed code chunks exploitation technique. Phrack Magazine[OL]http://www.suse.de/krahmer/no-nx.pdf,2005

[13] Riley R,Jiang X,Xu D. Guest-transparent prevention of kernel rootkits with vmm-based memory shadowing[C]//RAID '08: Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection. Berlin,Heidelberg:Springer-Verlag,2008:1-20

[14] Microsoft: A detailed description of the data execution prevention (dep) feature in windows xp service pack2[OL]. http://support.microsoft.com/kb/875352

[15] Mueller U. Brainfuck:An eight-instruction turing-complete programming language [OL]. http://www.muppetlabs.com/breadbox/bf/?

[16] The x86 instruction set architecture[OL]. Http://www.ugrad.cs.ubc.ca/cs411/2009W2/downloads/x86.pdf

[17] Viro A. Linux kernel sendmsg() local buffer overflow vulnerability[OL]. http://www.securityfocus.com/bid/14785,2005

[18] Petroni N,Hicks M. Automated detection of persistent kernel control-flow attacks[C]//Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS). ACM, NewYork,NY,USA,2007:103-115

(上接第 43 页)

当 DLL 被成功注入 IE7 进程的地址空间后,DLL 的 DllMain 函数会收到 DLL_PROCESS_ATTACH 通知并且开始执行前面提到的代码覆盖以及跳转至检测代码。至此,基于代码覆盖的浏览器漏洞利用攻击检测的实现部分就已经完成。我们将整体的检测流程总结如下(见图 2)。

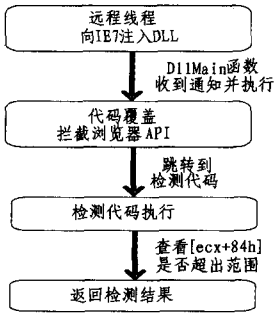

图 2

3 实验

该检测方法应用于以下诸多漏洞利用检测中(见表 1),取得了相当不错的效果。

表 1

CVE 编号	漏洞描述
CVE-2007-5601	RealPlayer ierpplug. dll ActiveX 控件播放列表名称栈溢出漏洞
CVE-2007-5017	利用 Yahoo! Messenger 8.1.0 ActiveX 控件漏洞挂马的病毒
CVE-2010-0249	"极光"变种漏洞病毒
CVE-2008-1309	RealPlayer ActiveX 控件属性的堆内存损坏漏洞
CVE-2007-4105	百度搜霸漏洞

结束语 本文所提出的基于代码覆盖的浏览器漏洞利用攻击检测方法,通过漏洞利用代码的概念验证,结合代码覆盖、逆向工程、API 拦截以及 DLL 注入等技术,最终实现了一套行之有效的检测方式。这种方法具有简洁、有效和误报率低等特点,可以通过将该技术部署于众多虚拟机中,批量检测网页,向杀毒软件公司以及搜索引擎等提供高可信度的挂马网页黑名单。

参 考 文 献

[1] 傅建明,彭国军,张焕国. 计算机病毒分析与对抗 [M]. 武汉:武汉大学出版社,2009:155-179

[2] 韩筱卿,王建锋,钟玮,等. 计算机病毒分析与防范大全 [M]. 北京:电子工业出版社,2006:168-179

[3] Szor P. 计算机病毒防范艺术 [M]. 段新海,杨波,王德强,译. 北京:机械工业出版社,2007:143-144

[4] 卓新建. 计算机病毒原理及防治 [M]. 北京:北京邮电大学出版社,2004:115-117

[5] 刘功申. 计算机病毒原理及其防范技术 [M]. 北京:清华大学出版社,2008:130-183

[6] Grimes R A. 恶意传播代码:Windows 病毒防护 [M]. 张志斌,贾旺盛,等译. 北京:机械工业出版社,2004:289-321

[7] Richter J. Windows 核心编程 [M]. 葛子昂,周靖,等译. 北京:清华大学出版社,2011,1:587-591

[8] 舒敬荣,朱安国,齐善明. HOOK API 时代码注入方法和函数重定向技术研究 [J]. 计算机应用与软件,2009,26(5):107-110

[9] 张瑜,李涛,吴丽华,等. 基于免疫的 Windows 未知病毒检测方法 [J]. 电子科技大学学报,2010(39):80-84