基于 CVE-2012-0158 的软件漏洞分析与利用①

王大伟, 周 军, 梅红岩

(辽宁工业大学 电子与信息工程学院, 锦州 121001)

摘 要: 软件漏洞是引起计算机安全问题的重要根源之一. 以 CVE-2012-0158 漏洞为例, 探索了漏洞产生的原理及利用方式. 通过动态分析方法简要地描述该漏洞被触发时, 程序所执行的代码及函数调用情况, 从本质上解析了漏洞产生的原因及危害, 从而引起人们对安全开发、避免产生漏洞的重视. 给出了通过基于安全性的软件开发方式, 可以从根源上减少软件漏洞引起的计算机安全问题, 从而提升系统和软件的安全性能.

关键词: 漏洞分析; CVE-2012-0158; 动态分析; 缓冲区溢出; 安全开发

Analysis and Exploitation of Software Vulnerability Based on Cve-2012-0158

WANG Da-Wei, ZHOU Jun, MEI Hong-Yan

(School of Electronic and Information Engineering, Liaoning University of Technology, Jinzhou 121001, China)

Abstract: Software vulnerability is one of the important causes of computer security. Taking the CVE-2012-0158 as an example, the form prince and exploitation way of vulnerability is explored. Which codes and functions are called by the procedure when the vulnerability is triggered are briefly described through the dynamic analysis method and the causes and hazards of vulnerability are explained to arouse people's attention of taking safe development and avoiding vulnerability. Then safe development methods based on security are mentioned to reduce computer security problems caused by software vulnerabilities fundamentally, so as to improve the safety performance of the system and software.

Key words: software vulnerability; CVE-2012-0158; dynamic analysis; buffer overflow; security development

1 引言

网络安全事件频发, 计算机及网络安全问题对人们的生产、生活和社会的稳定与发展产生重要的影响. 软件漏洞是引起网络中计算机安全问题的重要因素之一. 现代社会对计算机软件的需求程度, 使得软件的规模越来越大、逻辑性越来越复杂. 而另一方面, 大多数的软件在发布的时候都不敢确保已彻底地消除了软件中的所有逻辑缺陷, 其中功能性逻辑缺陷称为 bug, 安全性逻辑缺陷称为漏洞(vulnerability)[1]. 软件漏洞虽然不会影响软件的正常使用, 但是若被恶意的攻击者利用, 将会使软件去执行不在程序设计范围内的额外代码, 从而给计算机用户带来极大的安全威胁.

软件漏洞尚无统一的定义. 经过广泛地研究和总结, 文献[2]中将软件生命周期中涉及到安全的设计错误、编码缺陷和运行故障[3]统称为软件漏洞. 常见的漏洞包括缓冲区溢出漏洞、XSS 漏洞、SQL 注入漏洞. 到目前为止曝光的漏洞相当多, 如仅 2013 年 3 月至 4 月的一个月时间内, 国家计算机网络入侵防范中心发布漏洞总条目为 475 条[4]. 为了能够合理的标识和公布漏洞, 世界上有两个权威机构致力于这方面的工作: 一个是 MITRE 公司从事的"公共漏洞列表"(Common Vulnerability Enumeration, CVE), 为每个漏洞建立统一的标识, 并进行审查[5], 方便漏洞研究的信息共享及数据交换; 另一个计算机应急响应组(CERT, Computer Emergency Response Team), 会第一时间跟进新漏洞的描述信息及 POC 的发布链接等.

软件漏洞是引发计算机及网络安全事件的重要因素. 对软件漏洞进行深入的分析与利用, 是提高网络安全性的前提条件. 本文首先给出了软件漏洞分析的一般过程, 并以 CVE-2012-0158 为例利用汇编技术注

① 基金项目:国家自然科学基金(61074014);辽宁省教育厅重点实验室项目(LS2010079)
收稿时间:2014-03-08;收到修改稿时间:2014-04-04

入 shellcode, 用反汇编技术分析漏洞实际产生的原因, 详细描述了漏洞分析中所采用的具体方法及过程. 为安全研究人员针对漏洞进行快速的补救封堵或者二次开发利用提供了有效的参考.

2 CVE-2012-0158漏洞描述

CVE-2012-0158 是 Microsoft Office 2003 SP3、2007 SP2 和 SP3, SQL Server 2000 SP4、2005 SP4 及 2008 SP2、SP3、R2 等程序中所暴露出来的漏洞, 即 “MSCOMCTL.OCX RCE 漏洞”. 该漏洞是由于程序通用控件中 MSCOMCTL.OCX 的 ListView、ListView2、TreeView 和 TreeView2 ActiveX 控件[6]对于范围判断所产生的逻辑疏忽而形成的缓冲区溢出问题. 其所造成的危害是破坏系统栈并使之溢出.

2.1 漏洞触发的环境

漏洞触发需要公布漏洞时的版本软件才能够完全实现情景再现. 以 CVE-2012-0158 为例, 选取的环境条件是: Windows XP Professional SP3、Microsoft Office Word 2007(12.0.4518.1014) 、 MSCOMCTL.OCX 6.1.97.82. MSCOMCTL.OCX 是 Windows 中的一个动态链接库.

2.2 系统栈平衡

内存加载进程后, 按照所提供的不同功能, 可分为: 代码区、数据区、堆区、栈区. 代码区用来存储进程的二进制代码, 处理器会从此区域取值并执行; 数据区用来存储进程的全局变量; 堆区用于进程动态地对内存进行分配和回收; 栈区用来动态地存储进程中函数之间的调用. 缓冲区溢出可以发生在栈区、堆区或存放静态变量的数据区, 本文仅以栈区的溢出为例进行分析. 系统栈由系统自动维护, 它用于高级语言中的函数调用. 用一个简单的例子展示函数调用在栈区的实现过程. 其代码如下:

```
int func_B (int B_arg){
    int B_var;
    B_var= pow(B_arg, 2);
    return B_var;}
int func_A(int A_arg1,int A_arg2){
    int A_var;
    A_var=func_B(A_arg)+A_arg2;
    return A_var;}
int main(int arg,char **argv,char **envp){
    int main_var;
    var_main=func_A(4,3);
    return main_var;}
```

这段代码经过编译器编译加载到内存后, 其运行过程中出现的调用过程以及参数、局部变量在栈区中的分布如图 1 所示.

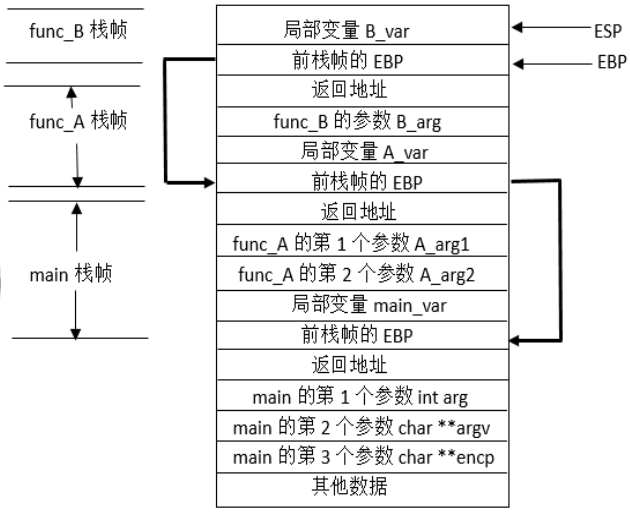

图 1　函数调用实现示意图

由图 1 可以看出, 当一个运行的函数调用其他函数时, 系统会自动将母函数的下一行地址作为返回地址压入栈, 结束完此栈帧. 开始的下一帧, 首先保存调用函数的局部变量, 然后是参数, 最后运行该函数. 按照函数调用的层次, 依次压栈, 当最后调用的函数运行完之后将弹出该函数的栈帧, 按照返回地址返回到母函数运行. 系统栈的这种体制是为了能够保证程序的正常运行, 每次都能正确的返回到母函数. 但是当局部变量中存在数组之类的缓冲区, 且存在缓冲区越界的缺陷时, 将有可能破坏掉 EBP、返回地址等数据, 从而破坏系统栈的平衡.

3 漏洞分析与利用

漏洞分析是在程序代码中迅速定位出漏洞, 准确分析其攻击原理的过程. 熟练的漏洞利用技术是进行漏洞分析的保障基础, 否则很可能将不可利用的 bug 判断成漏洞, 或者对漏洞等级评估错误. 一般情况下, 漏洞发现者提交漏洞信息时, 需要向安全专家提供一段能够重现漏洞的代码, 这段代码被称作 POC(Proof of Concept).

分析漏洞需要能够重现漏洞被触发的情形, 然后

才能通过跟踪和调试来分析它. 漏洞重现一般是找到漏洞的触发条件和步骤, 能稳定的复现漏洞, 通常需要一段 POC. 本文以 CVE-2012-0158 为例, 其 POC 就是一个能够释放计算器的 word 文件. 其触发时如图 2 所示.

图 2 触发 CVE-2012-0158 漏洞示意图

3.1 漏洞分析的一般方法

漏洞分析的主要目的就是能够找出造成漏洞的原因及位置. 一般进行漏洞分析的方法分为: 动态调试、静态调试、指令追踪、补丁比较. 动态调试一般是利用动态调试工具运行存在漏洞的软件或者进程, 单步运行其汇编指令, 了解程序的执行流程及堆栈数据, 从而快速定位漏洞原因. 静态调试是指利用逆向分析工具如 IDA 等对程序进行反汇编, 获得程序的“整体观”及反汇编代码, 通过对反汇编代码的理解, 分析所实现的功能, 查找存在的缺陷. 指令追踪是通过对比的分析技术, 分两次运行, 第一次正常运行程序, 记录下所有执行的指令序列, 第二次触发漏洞, 再次记下所有指令; 然后分析这两组指令, 重点调试和跟踪所执行指令不同的代码区, 最终定为到触发漏洞的函数. 补丁比较的研究人员就可以根据补丁前后的PE文件的反汇编代码得出存在漏洞的函数, 然后编写 POC.

本文主要以动态调试的方式介绍漏洞分析方法.

3.2 动态分析 CVE-2012-0158 漏洞

漏洞分析的主要目的就是能够找出造成漏洞的原因及位置. 一般进行漏洞分析的方法分为: 动态调试、静态调试、指令追踪、补丁比较. 动态调试一般是利用动态调试工具运行存在漏洞的软件或者进程, 单步运行其汇编指令, 了解程序的执行流程及堆栈数据, 从而快速定位漏洞原因. 静态调试是指利用逆向分析工具如 IDA 等对程序进行反汇编, 获得程序的“整体观”及反汇编代码, 通过对反汇编代码的理解, 分析所实现的功能, 查找存在的缺陷. 指令追踪是通过对比的分析技术, 分两次运行, 第一次正常运行程序, 记录下所有执行的指令序列, 第二次触发漏洞, 再次记下所有指令; 然后分析这两组指令, 重点调试和跟踪所执行指令不同的代码区, 最终定为到触发漏洞的函数. 补丁比较的研究人员就可以根据补丁前后的PE文件的反汇编代码得出存在漏洞的函数, 然后编写 POC.

动态调试的一般步骤为: 定位 shellcode[7]注入的范围区域; 定位 shellcode 的起始位置; 分析 shellcode 内容及函数调用情况. 在 CVE-2012-0158 漏洞中, 只要能定位出 POC 中注入 shellcode 的位置就能快速准确得定位到漏洞. 由该段 POC 在运行时弹出一个计算器可知, shellcode 中释放并执行了一个 EXE, 而可执行文件的调用一般与下列函数相关:

- WinExec: 运行指定的程序.
- GetFileSize: 判断文件长度.
- CreateFileW: 多功能的函数, 可打开或创建控制台、通信资源、目录、磁盘驱动器、文件等对象, 并返回可访问的句柄.
- WriteFile: 将数据写入一个文件. 该函数比 fwrite 函数要灵活的多.
- SetFilePointer: 设置当前的读写位置.
- CreateProcessW: 创建新的进程及其主线程, 该新进程运行指定的可执行文件.

本文利用 WinExec 与 WriteFile 进行调试举例.

3.2.1 定位 shellcode 注入的范围区域

首先运行 WINWORD.EXE 程序, 将其进程加载到 WinDBG 中. 由于该 POC 运行过程中要释放一个 cacl.exe 程序, 所以在 shellcode 中调用了 WinExec 函数[8], 对该函数下断点: kp kenerl32！WinExec.

通过 WINWORD 打开 POC 文件, 会运行到 WinExec 函数时停止, 此时由 ESP 指向的栈顶所存储的是返回地址. 查看 esp 所指向的地址以及 esp+4 的内容, 如图 3 所示. 分析 WinExec(lpCmdLine, uCmdShow) 的格式, 第 1 个参数为有明确路径的文件, 第 2 个参数是文件打开方式. 所以从栈顶到栈底存储的前三个元

素分别为: 返回地址、lpCmdLine、uCmdShow. 可知 esp+4 的地址 00121384 内中包含的内容是一个带有路径的文件, 即 WinExec 函数的第一个参数, 此文件就是 WinExec 调用的函数. 然后用反汇编命令分析返回地址, 由得出的汇编语言“mov eax, dword ptr[edi+5Dh]”, 可知此条返回地址没有执行“jmp xxxx”跳转回母函数, 即返回地址已被 shellcode 代码覆盖, 现在所在的内存地址 00122829 已经是 shellcode 的内部.

```
0:000> dd esp
001212f0  00122829 00121384 00000005 7c80ae30
00121300  7c810b07 7c80fdbd 7c810c1e 7c801812
00121310  7c801a28 7c835de2 7c810e17 7c809bd7
00121320  7c8623ad 7c80b55f 7c81cafa 00000470
00121330  72f73f3c 0012135c 77e5b9e8 00000400
00121340  00121804 72f73f3c 72f73f00 00121384
00121350  00121804 00150000 00212008 000005b0
00121360  000005b4 00214d76 00000000 0001c000
0:000> da 00121384
00121384  "C:\DOCUME~1\ADMINI~1\LOCALS~1\Te"
001213a4  "mp\paw.exe"
0:000> u 00122829
00122829 8b475c          mov     eax,dword ptr [edi+5Ch]
0012282c b9a6000000      mov     ecx,0A6h
00122831 01c1            add     ecx,eax
00122833 49              dec     ecx
00122834 c6017d          mov     byte ptr [ecx],7Dh
00122837 8b8780000000    mov     eax,dword ptr [edi+80h]
0012283d 31f6            xor     esi,esi
0012283f 8b7750          mov     esi,dword ptr [edi+50h]
```

图 3　查看栈顶的元素

3.2.2 定位 shellcode 的起始位置

溢出技术的核心思想就是覆盖, 所要做的工作就只有两个: 植入并定位精心构建的二进制代码, 实现执行流程的跳转[9]. 在确定 shellcode 的范围后, 转到反汇编视图窗口, 先追踪到 00122829, 由于 shellcode 只是内嵌到缓冲区中的一小段代码, 为了提升每次定位 shellcode 的准确率, 可通过在 shellcode 的前面添加多个字节的 NOP 指令实现跳过 NULL 字节[10], 所以在一般的 shellcode 中只要进入其范围, 发现有大量 nop 时, 便找到了 shellcode 的开始端, 如图 4 所示.

```
001226bd 90              nop
001226be 90              nop
001226bf 90              nop
001226c0 90              nop
001226c1 90              nop
001226c2 90              nop
001226c3 90              nop
001226c4 90              nop
001226c5 90              nop
001226c6 90              nop
001226c7 31c0            xor     eax,eax
001226c9 648b4030        mov     eax,dword ptr fs:[eax+30h]
001226cd 8b400c          mov     eax,dword ptr [eax+0Ch]
001226d0 8b401c          mov     eax,dword ptr [eax+1Ch]
001226d3 8b7008          mov     esi,dword ptr [eax+8]
001226d6 8b7820          mov     edi,dword ptr [eax+20h]
001226d9 8b00            mov     eax,dword ptr [eax]
001226db 66837f1800      cmp     word ptr [edi+18h],0
001226e0 75f1            jne     001226d3
001226e2 81ec00040000    sub     esp,400h
001226e8 89e7            mov     edi,esp
```

图 4　查找 sellcode 开始端

并由图 4 中的 001226e0 的汇编指令“sub esp 400h”可知, 十六进制的 400h 转变为十进制为 1024, 即在栈中分配 1024 字节大小的空间.

3.2.3 分析 shellcode 内容及函数调用情况

用 OllyDebug 载入 word 运行, 并下断点 WriteFile 后打开 POC 文件, 单步运行程序, 注意观察堆栈区, 若返回地址类似 00122xxx 时, 表示进入了 shellcode, 对堆栈区进行回溯查看.

在堆栈区中显示了返回地址为 0012281B, 在代码区查找该地址, 其汇编指令为“push dword ptr ds:[edi+1c]”为压栈传递参数数据, 所以上面的 push ebx 为分配 shellcode 的大小, push ecx 为调用的计算器的大小. 往上查看 ecx 的来源途径, 可在位置 001227F4、001227F7 发现汇编指令: “mov dword ptr ds: [edi+64], eax”、“mov eax, 1C000”, 可知 ecx 是硬编码写入的十六进制 1C000, 即十进制的 11468 字节. 查看释放出来的计算器 paw 属性可知其大小为 112KB, 即 114,688 字节. 单步运行之后, 调用 SetFilePointer 函数, 此处指定的 offset=2840, 表示这个计算器 exe 所在的文件位置是在攻击样本的 offset=0x2840 处.

查看函数调用, 在 OD 内存区中显示的便是该段 POC 所调用的 API. 其都是动态链接库 kernel32[11]中的函数, 分别为: GetProcAddress、GetFileSize、GlobalAlloc、SetFilePointer、ReadFile、WriteFile、CloseHandle、GetModuleFileNameA、WinExec、ExitProcess.

分析完一个漏洞的 POC 后, 就知道 shellcode 在程序中的注入方式以及调用的函数, 经过一步步分析, 发现函数 : MSCOMCTL!DllGetClassObject +0xb1b9 被调用了两次. 该函数只被分配了 20 个字节, 第一次调用使用了 12 字节空间, 在第二次调用之前有对参数大小的检查, 该大小与 8 比较, 应该当该大小大于 8 时直接结束, 但程序中为小于 8 时结束, 而当输入参数大于 8 时继续调用 MSCOMCTL!DllGetClassObject+0xb1b9, 使得发生了栈溢出.

3.3 漏洞利用

漏洞利用的目的是在目标系统上自动运行一些程序, 就如本文所涉及漏洞 CVE-2012-0158 的 POC 在运行 word 时自动调用计算器一样的 shellcode. shellcode 代码, 在发送到目标系统利用某个特定漏洞时, 被当作填充数据进入到程序中. 因此 shellcode 是溢出程序

的关键, 漏洞利用的核心就是 shellcode 编写, 且完全由编写代码的人来决定. 缓冲区溢出漏洞, 作为攻击中应用最广泛的漏洞类型之一, 其漏洞利用技术也不断被提出, 主要分为两个方面, 一个是漏洞的补救, 另一个是漏洞的攻击.

针对漏洞的补救, 主要是在发现漏洞时能够及时的将该漏洞进行修复或采取防御机制使其不可被触发. 如 Windows 立即针对 Microsoft Office Word 2007 Service Pack 2 版本发布了补丁 KB2598041, 并陆续地针对缓冲区溢出采用了多种防御机制[12], 如图 5 所示.

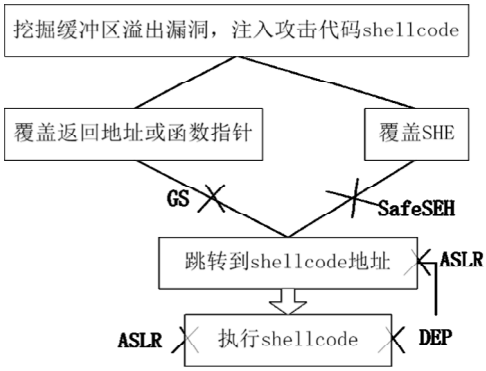

图 5 windows 的各种防御机制

其中缓冲区安全检查(GS)、安全的结构化异常处理 (SafeSEH) 和地址空间随机化分布 (ASLR)[13] 是 Windows 应用程序层阻止 shellcode 执行最关键的三种防护技术. 而恰好相反, 若想进行漏洞的攻击, 研究人员可以通过充分掌握操作系统内核的工作模式以及汇编语言等多种编程技术, 成功地绕过各种防御机制来实现自己的攻击目的.

5 结语

漏洞分析与利用是一项全面而又系统性的任务, 侧重于如何快速准确的定位漏洞位置、锁定其产生原因并对漏洞进行及时的补救或攻击. 本文通过对典型的缓冲区溢出漏洞 CVE-2012-0158 所进行的分析, 给出了由于一处简单的逻辑错误造成严重的计算机潜在安全问题. 从而说明了在算法设计或程序开发的过程中充分考虑安全性的重要性.

如果能在整个应用程序开发过程中考虑其可能存在的安全性问题, 并将安全方面的问题完全整合到开发过程中, 做到安全地编码, 便可以使应用程序更稳定, 降低漏洞的产生. 但如果不考虑安全性, 直到产品生命周期中的 QA 或用户验收阶段才考虑安全性, 很可能会导致返工, 延迟交付, 甚至交付之后由于漏洞引起严重的事故, 将会产生无法弥补的后果.

参考文献

1 王清主编.0 DAY 安全:软件漏洞分析技术(第 2 版).北京:电子工业出版社,2011.
2 吴世忠,郭涛,董国伟,王嘉捷.软件漏洞分析技术进展.清华大学学报(自然科学版),2012,10:1309–1319.
3 陆民燕.软件可靠性工程.北京:国防工业出版社,2011.
4 中国科学院研究生院国家计算机网络入侵防范中心.2013 年 4 月十大重要安全漏洞分析.信息网络安全,2013,(6):99–100.
5 单国栋,戴英侠,王航.计算机漏洞分类研究.计算机工程,2002,28(10):3–6.
6 中国科学院研究生院国家计算机网络入侵防范中心.2012 年 4 月十大重要安全漏洞分析.信息网络安全,2012,(6):98,100.
7 Anley C, Heasman J. The Shellcoder's Handbook - Discovering and Exploiting Security Holes(2nd ed.). New York, USA: Wiley Publishing Inc., 2007.
8 段钢编著.加密与解密(第 3 版).北京:电子工业出版社,2008.
9 王俊卿,陈高峰,连强.基于 MS Office 漏洞利用技术的研究.微计算机信息,2012,10:364–365,408.
10 吴伟民,郭朝伟,黄志伟,苏庆,陈秋伟. 基于 Windows 的结构化异常处理漏洞利用技术.计算机工程,2012,38(20):5–8.
11 Choi JC, Han YM, Cho SJ, et al. A static birthmark for MS windows applications using import address table. IEEE 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS). Taiwan, China. 2013. 129–134
12 邵林.软件缓冲区溢出漏洞自动化发掘系统[硕士学位论文].成都:电子科技大学,2009.
13 胡晗翰.Win32 下缓冲区溢出漏洞利用技术的分析和改进[硕士学位论文].武汉:华中科技大学,2008.