

## 面向 RTF 的 OLE 对象漏洞分析研究

乐德广<sup>1,2,4</sup>, 章亮<sup>3</sup>, 龚声蓉<sup>1,2</sup>, 郑力新<sup>3</sup>, 吴少刚<sup>4</sup>

(1. 常熟理工学院计算机科学与工程学院, 江苏 常熟 215500;

2. 苏州大学计算机科学与技术学院, 江苏 苏州 215006;

3. 华侨大学工学院, 福建 泉州 362021;

4. 江苏中科梦兰电子科技有限公司, 江苏 常熟 215500)

**摘要:** 针对 RTF 文档在 OLE 对象解析过程中出现的安全漏洞问题, 提出了一种基于数据块解析及特征数据构造的 OLE 对象漏洞分析方法。利用逆向技术分析 OLE 对象漏洞的触发条件, 通过数据块解析定位 OLE 对象漏洞的触发点, 并基于特征数据构造检测 OLE 对象漏洞。实验表明, 该方法不但能正确检测出 RTF 的 OLE 对象漏洞, 而且能精准定位漏洞触发点, 为研究漏洞补丁提供有效依据。此外, 与现有方法相比, 该方法还具有更高的检测效果, 从而有效防御各种面向 RTF 文档的 OLE 对象漏洞利用攻击。

**关键词:** RTF 文件; 软件安全; OLE 对象漏洞; 漏洞分析

**中图分类号:** TP393

**文献标识码:** A

**doi:** 10.11959/j.issn.2096-109x.2016.00011

## Research on OLE object vulnerability analysis for RTF file

LE De-guang<sup>1,2,4</sup>, ZHANG Liang<sup>3</sup>, GONG Sheng-rong<sup>1,2</sup>, ZHENG Li-xin<sup>3</sup>, WU Shao-gang<sup>4</sup>

(1. School of Computer Science and Engineering, Changshu Institute of Technology, Changshu 215500, China;

2. School of Computer Science and Technology, Soochow University, Suzhou 215006, China;

3. College of Engineering, Huaqiao University, Quanzhou 362021, China;

4. Jiangsu Lemote Technology, Changshu 215500, China)

**Abstract:** In order to deal with the problem of OLE parsing vulnerability for RTF documents, a kind of vulnerability analysis method based on data block analysis and characterization data construction was proposed. The trigger conditions of OLE object vulnerability by reverse engineering technique were analyzed. The trigger point of vulnerability was located through data block analysis. The OLE object vulnerability was detected based on characterization data construction. Tests show that the proposed method not only detects the OLE object vulnerability correctly, but also locates the point of vulnerability accurately, which provides the effective support for the research on vulnerability patches. Besides, the detection effectiveness of the proposed method is higher than that of other methods, which can effectively defense the exploit attack of OLE object vulnerability for RTF documents.

**Key words:** RTF document, software security, OLE vulnerability, vulnerability analysis

收稿日期: 2015-10-25; 修回日期: 2015-11-19。通信作者: 乐德广, 306045675@qq.com

基金项目: 国家自然科学基金资助项目 (No.61202440, No.61170124, No.61402057); 福建省物联网云计算平台建设基金资助项目 (No.2013H2002)

**Foundation Items:** The National Natural Science Foundation of China (No.61202440, No.61170124, No.61402057), Fujian Internet of Things and Cloud Computing Program (No.2013H2002)

## 1 引言

随着计算机技术和软件技术的发展,各种办公文档和软件在人们的日常工作和生活中扮演着越来越重要的角色。然而,随着越来越多个人隐私和重要数据通过各种办公软件存储在电子文档中,针对办公软件的攻击事件不断发生,各种办公软件安全问题日趋严重<sup>[1]</sup>。产生办公软件安全问题的根本原因在于其存在文档格式解析和执行缺陷,即软件安全漏洞<sup>[2]</sup>。因此,办公软件安全漏洞问题成为信息安全的热门课题,国内外的安全研究机构和学者对办公软件安全漏洞进行了大量的分析研究<sup>[3~9]</sup>。在文献[3]中,杨丁宁针对微软的 MS Office 软件在解析 DOC、PPT 等文档时调用 ActiveX 控件所产生的漏洞,基于黑盒 Fuzzing 测试技术设计并实现了 ActiveX 控件漏洞挖掘工具 (ActiveX-Fuzzer),并通过该工具发现多个未公布的高危漏洞。刘奇旭在文献[4]中将静态分析和动态测试技术相结合,设计并实现了 Flash 跨站脚本漏洞挖掘工具 (FXD, flash XSS detector),并在对 Alexa Top100 的网站测试中发现可以导致 XSS 攻击的 Flash 文件。Rebert 在文献[5]中,根据文件格式结构化存储的特征,给出一种优化种子文件选择的方法,设计并实现一种基于 Fuzzing 的文件型漏洞智能挖掘与分析系统。在文献[6]中,史飞悦提出了一种动静态分析相结合的漏洞挖掘分析方法对微软 Office 漏洞进行挖掘分析。文献[7]通过对 RTF 文件的 pFragments 绘图属性进行深入研究,提出一种基于指令回溯及特征数据构造的漏洞分

析方法分析 MS Word 程序在 RTF 文件解析中产生的缓冲区溢出漏洞。该分析方法能有效检测出 MS Word 的 RTF 文件绘图属性解析漏洞。此外,Leathery 分析了在没有正确处理带有 Task Symbol 组件的 RTF 文件时造成堆内存错误而导致触发漏洞[8]。Li 分析了 MS Office 在解析文件时动态调用公共控件而引起的 OLE 安全漏洞<sup>[9]</sup>,该类漏洞不仅可以通过 DOC 或 PPT 文档触发,甚至通过 RTF 文件触发。因此,本文在研究 RTF 文件的 OLE 对象嵌入技术基础上,提出了一种基于数据块解析及特征数据构造的 OLE 对象漏洞逆向分析方法,并根据该方法给出了面向 RTF 文件的 OLE 对象漏洞分析流程。实验测试证明,本方法能有效检测出 RTF 文件的 OLE 对象解析漏洞。此外,与现有的同类检测方法相比,本文方法具有更高的检测效率。

## 2 研究背景

### 2.1 OLE 对象链接和嵌入技术

对象链接和嵌入 (OLE, object linking and embedding) 是一种面向对象的技术,该技术允许在程序之间链接和嵌入对象数据,从而建立复合文档<sup>[10]</sup>。微软 MS Office 97-2003 的各种文件格式 (DOC、RTF、XLS 和 PPT 等) 都采用 OLE 技术存储文件规范。通过该规范,一个文件不仅包含文本,而且可以包括图形、电子表格、甚至声音视频信息。这里以一个包含 OLE 结构的 RTF 文件为例,新建一个同时包含“Hello World”文本信息和 PNG 图片的 RTF 文件,提取出其中的 OLE 结构,通过 Offvis 分析其 OLE 结构<sup>[11]</sup>,如图 1 所示。

| Parsing Results        |                                         |            |            |                           |
|------------------------|-----------------------------------------|------------|------------|---------------------------|
| Name                   | Value                                   | Offset     | Size       | Type                      |
| OLESSRoot              |                                         | 0x00000000 | 0x0000e000 | OLESSDataFile             |
| OLESSHeader            |                                         | 0x00000000 | 0x00000200 | OLESSHEADER               |
| FAT[128]               |                                         | 0x0000da00 | 0x00000200 | List<SECTOR>              |
| MiniFAT[128]           |                                         | 0x0000e000 | 0x00000200 | List<SECTOR>              |
| DirectoryEntries[8]    |                                         | 0x0000dc00 | 0x00000400 | List<OLESSDirectoryEntry> |
| OLESSDirectoryEntry[0] | \Root Entry                             | 0x0000dc00 | 0x00000080 | OLESSDirectoryEntry       |
| OLESSDirectoryEntry[1] | \Root Entry\Data                        | 0x0000dc80 | 0x00000080 | OLESSDirectoryEntry       |
| OLESSDirectoryEntry[2] | \Root Entry\1 Table                     | 0x0000dd00 | 0x00000080 | OLESSDirectoryEntry       |
| OLESSDirectoryEntry[3] | \Root Entry\WordDocument                | 0x0000dd80 | 0x00000080 | OLESSDirectoryEntry       |
| OLESSDirectoryEntry[4] | \Root Entry\ SummaryInformation         | 0x0000de00 | 0x00000080 | OLESSDirectoryEntry       |
| OLESSDirectoryEntry[5] | \Root Entry\ DocumentSummaryInformation | 0x0000de80 | 0x00000080 | OLESSDirectoryEntry       |
| OLESSDirectoryEntry[6] | \Root Entry\ CompObj                    | 0x0000df00 | 0x00000080 | OLESSDirectoryEntry       |

图1 Offvis 分析 RTF 的 OLE 文档结构



0 0000000000000000E0000 为头部信息,D0CF11E0A1B11AE1 为数据流的起始标志信息,000000000000000000000000000000003E000300FFFF09000 为数据流信息。

### 2.3 RTF 的 OLE 对象安全性问题

根据 2.2 节知道 RTF 文件包含不同的 OLE 对象类型。由于在解析这些对象类型时需要调用公共控件或动态链接库去解析对象数据。此外, RTF 的 OLE 对象数据存储在对对象数据控制字中, 因此 \objdata 等对象数据控制字属于可存数据区域。在这些可存数据区域中, 可以构造溢出数据, 甚至恶意代码 (Shellcode 等)。当 OLE 对象在解析这些特殊构造的数据时, 将触发 OLE 对象漏洞, 并利用该漏洞执行恶意代码, 从而产生 RTF 文件攻击。因此, 在 RTF 文件中包含 OLE 对象引发安全漏洞, 称为 RTF 的 OLE 对象漏洞。例如, 当在 RTF 文件中嵌入 \objocx 对象类型的 OLE 控件 TreeView 或 ListView 时, 该控件在解析 \objdata 内部数据时会造成堆栈溢出漏洞, 利用该漏洞能进行远程代码执行攻击<sup>[13]</sup>。当 RTF 文件在解析嵌入 \objocx 对象类型的 OLE 控件 TabStrip 时会造成内存破坏漏洞<sup>[14]</sup>。在 RTF 文件中包含 \objemb 对象类型的 TIFF 数据时, 会由于 OGL.DLL 在处理 TIFF 文件格式数据时产生整数溢出导致堆溢出漏洞<sup>[15]</sup>。此外, 在 RTF 漏洞利用攻击过程中, 通常会利用 OLE 对象绕过 Windows 的安全防护机制<sup>[16]</sup>。例如, 在漏洞利用的时候经常需要绕过 Windows 的地址空间布局随机化 (ASLR, address space layout randomization) 机制, 这时可以在 RTF 文件 OLE 对象的 \objdata 数据区域中包含未开启 ASLR 机制的控件 (MSCOMCTL.OCX, MSGR3EN.DLL 等), 通过这种方式可以绕过 ASLR<sup>[17]</sup>。

由于 RTF 文件经常会伴随着嵌入的 OLE 对象漏洞进行攻击,因此本文重点分析 RTF 的 OLE 对象漏洞。目前,已有一些国外的信息安全专家对 RTF 的 OLE 对象安全性问题开展相关研究,如 Boldewin 设计和实现了一种用于分析和提取 RTF 中恶意嵌入 OLE 对象的工具 RTFscan<sup>[18]</sup>; Bonfa 提出了一种基于 OLE 数据结构分析的方法,并设计和实现了用于检查并解密 RTF 样本恶

意代码的 OLE 漏洞扫描工具 pyOLEScanner<sup>[19]</sup>。这些方法和工具虽然可以判断 RTF 文件中是否包含存在漏洞的 OLE 对象或 Shellcode，但是并不能准确定位漏洞的触发点。本文在此基础上，通过逆向分析技术分析 RTF 文件中的 OLE 对象嵌入机制，并提出一种基于数据块解析及特征数据构造的 OLE 对象漏洞分析方法。

### 3 面向 RTF 文件的 OLE 对象漏洞分析

### 3.1 RTF 的 OLE 对象类型逆向分析

通过一个包含 OLE 对象类型的 RTF 文件 (test.rtf) 对 OLE 对象嵌入机制进行逆向分析, 其关键数据信息如下

```
{\objectobjocx\xf37\objsetsize\objw1500\objh7
49{\*\objclass MSComctlLib.ListViewCtrl.2}
{\*\objdata010500000020000001B0000004D53
436F6D63746C4C69622E4C697374566965774374
726C2E320000000000000000000000E0000D0CF11
E0A1B11AE100000000000000000000000000000000
003E000300FEFF0900060000000000000000000000
00100000001000000000000000000010000002000000
1000000FEFFFFFFF000000000000000000000FFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF-
FFF.....52006F006F007400200045006E00740072
00790000000000000000000000000000000001EFCDA
B0000050000000000060000000800008005000080B
0303A030000000000000000000000000000016000500FF
FFFFFFFFFFFFFFF020000004BF0D1BD8B85D111
B16A00C0F0283628000000000062BCDFB9340DC
D.....2143341208000000746174696F6E00134E08
7DEB010006001C0000000000000000000000000000
0001560A000001EFCDAB00000500985D6501070
000000800008005000080000000000000000000000
000000000001FDEECBD010005009017190000000
800000049746D736400000002000000010000000C
000000436F626A640000000800000008000000E53
746C4C69622E4 .....}}
```

在以上 RTF 文件数据中，\object 控制字表示解析的是一个 OLE 对象类型；\objcx 控制字表示该对象类型为 OLE 控件类型；\objclass 控制字引用 MSComctlLib.ListViewCtrl.2 参数，该参数表

示 OLE 对象类型调用 MSComctlLib 中的 ListViewCtrl.2 控件; \objdata 控制字表示对象数据, 这些对象数据采用十六进制形式存储。将 \objdata 控制字对象数据的头部信息通过 Winhex 进行解析<sup>[20]</sup>, 如图 2 所示。

从图 2 可以看出, 根据 OLE 对象数据的头部组成, OLE 版本的值为 0x00000501; 格式 ID 的值为 0x00000002, 表示 OLE 对象嵌入在 RTF 文件中; 程序名长度的值为 0x0000001B, 表示程序名的长度为 27 字节; 程序名为 MSComctlLib.ListViewCtrl.2, 数据流大小的值为 0x00000E00, 表示数据流信息的大小是 3 584 字节。

下面通过 Windbg 对 MSComctlLib.ListViewCtrl.2 指定的 ListViewCtrl.2 控件进行逆向调试<sup>[21]</sup>, 发现 RTF 在解析该 ObjectStream 数据时, 是将数据读取到栈空间中, 如图 3(a)所示。例如, 读取 OLE 对象中 ObjectStream 的 01EFCDA B 0000050000000000006000000800008005000080B 0303A03 数据的关键代码, 如图 3(b)所示。

从图 3(b)可以看到在解析 ObjectStream 数据时, 通过 rep movs dword ptr es:[edi], dword ptr[esi] 指令, 将文档中数据读取至栈 (如图 3(a)所示) 中。这种串复制指令很容易造成堆栈溢出或内存破坏, 因此 RTF 在解析包含 OLE 对象的文档时可能存在 OLE 对象漏洞。

### 3.2 基于数据块解析和特征数据构造的 OLE 对象漏洞分析

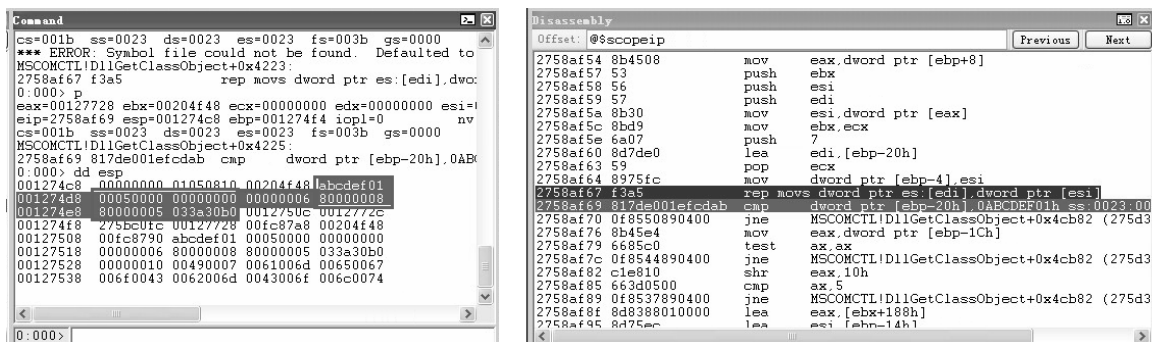
根据 3.1 节对 RTF 文件中的 OLE 对象类型分析, 知道当 RTF 中包含 OLE 对象, 并在解析 ObjectStream 时可能会产生 OLE 对象漏洞, 本小节在此基础上进行基于数据块解析的 OLE 对象漏洞分析。分析调试工具为 Windbg<sup>[21]</sup>, RTF 分析文档为 3.1 小节中的 test.rtf, RTF 文件解析程序为 MS Word 2010。

RTF 文件中的 OLE 对象数据保存在 ObjectStream 中, 而 ObjectStream 内部数据按照属性进行分类, 具有相同属性的数据属于同一个数据块, 例如 OCXNAME 数据块、Table 数据块、Contents 数据块等, 因此 ObjectStream 中一般存在多个数据块。RTF 文件解析程序通过类标识符 (CLSID, class identifier) 指定解析 ObjectStream 数据块时调用的控件类型。分析发现在函数 OLE32.ReadClassStg 中读取 OLE 对象中的 CLSID BDD1F04B-858B-11D1-B16A-00-C0F0283628, 该控件位于 MSCOMCTL.OCX 中。通过 Windbg 在解析该数据块入口的位置处下断点, 确定解析该数据块的部分关键代码如图 4 所示。

在图 4 中, 地址 0x276008F1 处调用的函数为 OLE32.CEExposedDocFile::openstream。该函数的作用是打开一个 Stream 流, 通过其中一个参数

| Offset   | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | A  | B  | C  | D  | E  | F  |                  |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------|
| 00000000 | 01 | 05 | 00 | 00 | 02 | 00 | 00 | 00 | 1B | 00 | 00 | 00 | 4D | 53 | 43 | 6F | .....MSCo        |
| 00000010 | 6D | 63 | 74 | 6C | 4C | 69 | 62 | 2E | 4C | 69 | 73 | 74 | 56 | 69 | 65 | 77 | mctlLib.ListView |
| 00000020 | 43 | 74 | 72 | 6C | 2E | 32 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | Ctrl.2.....      |
| 00000030 | 0E | 00 | 00 |    |    |    |    |    |    |    |    |    |    |    |    |    |                  |

图 2 OLE 对象数据的头部信息



(a) 栈空间 ObjectStream 数据

(b) 读取 ObjectStream 数据代码

图 3 RTF 解析 ObjectStream 关键代码

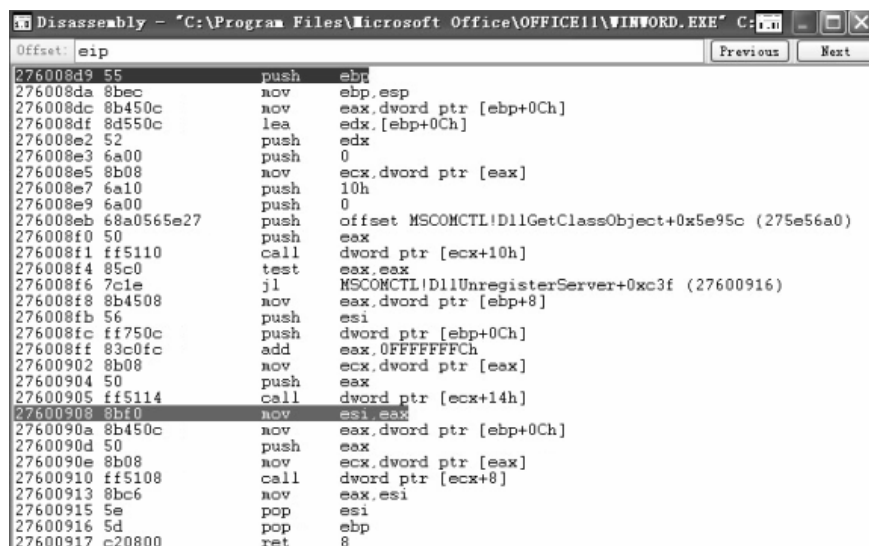


图4 解析OLE对象数据部分关键代码

[275E56A0]=Contents 知道该处打开的是一个内容为 Contents 的 Stream 流, 因此这里实际上要解析的是 OLE 对象中的 Contents 数据块。解析 Contents 数据块前 8 个字节的关键代码如图 5 所示。

在图 5 中，地址 0x2758AF04 处调用的函数为 OLE32.CExposedStream::Read，该函数的作用是读入 Stream 流，其中第 2 个参数为要读入数据返回的 buffer，第 3 个参数为要读入数据的大小。在这里 buffer 的地址为 0x001278E4，是一个栈空间的地址，读取的数据大小为 0x08 字节，因此通过 OLE32.CExposedStream::Read 将 Contents 数据块的前 8 个字节读入栈空间。读完后，在地址 0x2758AF08 和 0x2758AF18 处分别对读入的数据进行校验，校验是否为固定数据 0x12344321 和 0x08。如果校验成功，则继续读取 Contents 数据块余下的数据。如果校验失败，则退出该数据块

校验进程。Contents 数据块其他数据的读取方式也是将数据读取至栈区，再对数据进行校验，其读取校验过程如图 6 所示。

从图 6 可以看出, 在读取数据块中的数据时不是一次性读完, 而是读取其中一部分数据, 边读边校验。只有当校验通过时才会继续读取数据直至数据读取完毕, 否则剩余数据也不会再读取。根据图 6 的流程, 最终读取的数据如下

2143341208000000746174696f6e00134E087  
DEB010006001C0000000000000000000000000000060  
001560A000001EFCDAB00000500985D65010700  
000008000080050000800000000000000000000000  
000000000001FDEECBD0100050090171900000008  
00000049746D73640000000200000001000000**C0**  
**00000436F626A640000000800000008000000E537**  
**46C4C69622E4**

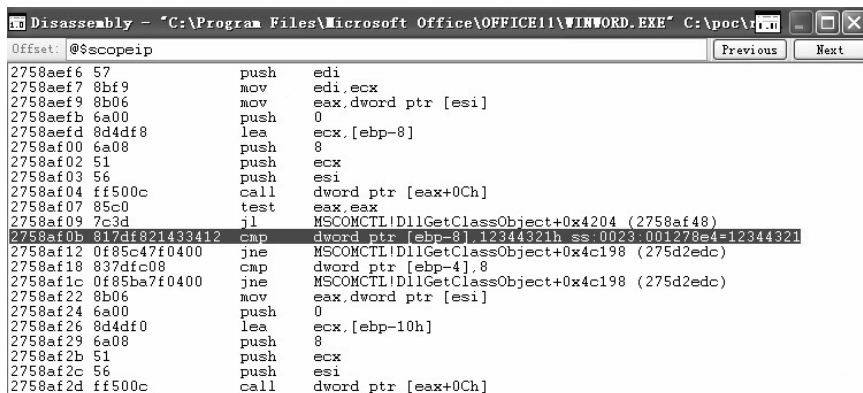


图5 解析 Contents 数据块部分关键代码

在以上数据中,当读取校验 Contents 数据块尾部中的 0C000000436F626A640000008000000E53746C4C69622E4 数据时,读入该部分数据的关键解析代码如图 7 所示。

从图 7 可以看出,在地址 0x275C89CA 处,通过 sub esp,14h 指令开辟 0x14 大小的空间用于存储 Contents 数据块尾部数据,并在地址 0x275C89DA 处第 1 次调用函数 MSCOMCTL.GetClassObject,该函数读入数据关键代码如图 8 所示。

从图 8 可以看出,在地址 0x275C8783 处,

通过 call dword ptr [eax+0Ch]调用 OLE32.CExposedStream::Read 函数,从 Contents 数据块尾部读取 1~4 字节的数据至栈区。随后在地址 0x275C878D 处,cmp dword ptr [ebp-4],edi 指令校验读入的数据是否和 0x0C 相等(参见图 7 地址 0x275c89d3 处的 push 0Ch)。如果不相等,则不读取数据;如果相等,则从数据块中读取 0x0C 字节数据至栈区,并且对读取数据中的前 4 个字节是否为 0x6A626F43 进行校验。如果校验不成功,则退出当前数据块的解析;如果检验成功,则判断所读取的 0x0C 字节数据中的后 4 个字节

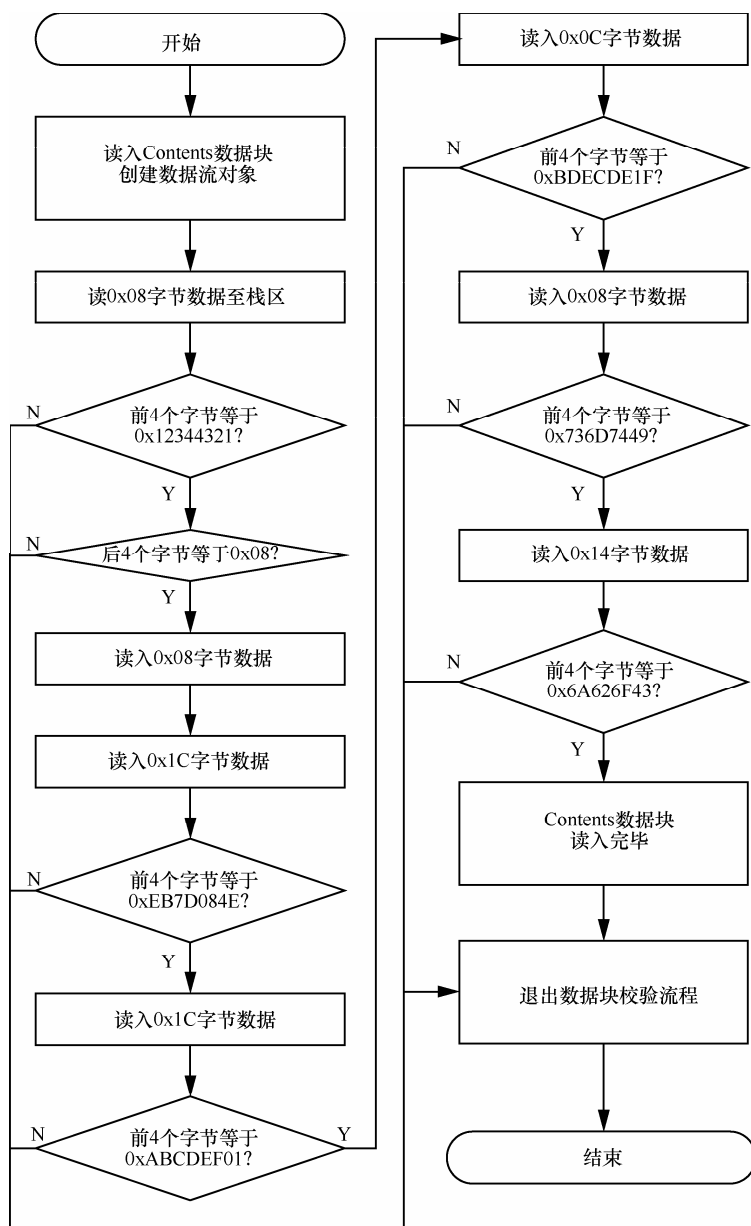


图6 Contents 数据块的读取校验流程

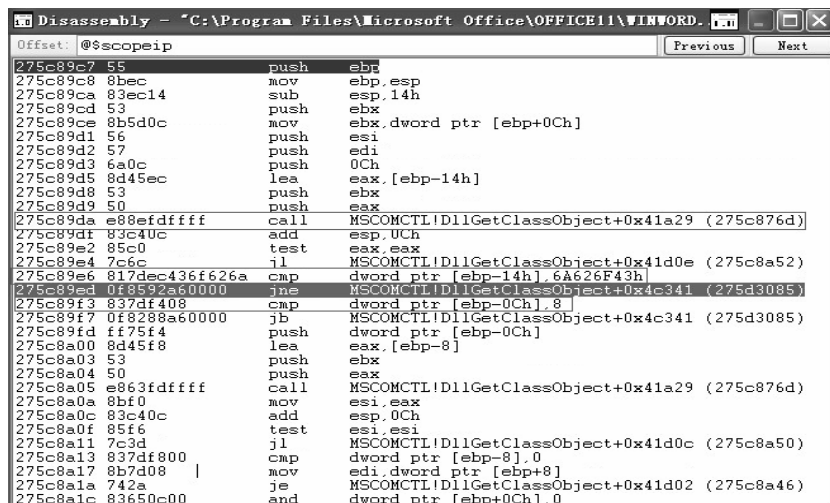


图7 解析 Contents 数据块尾部数据关键代码

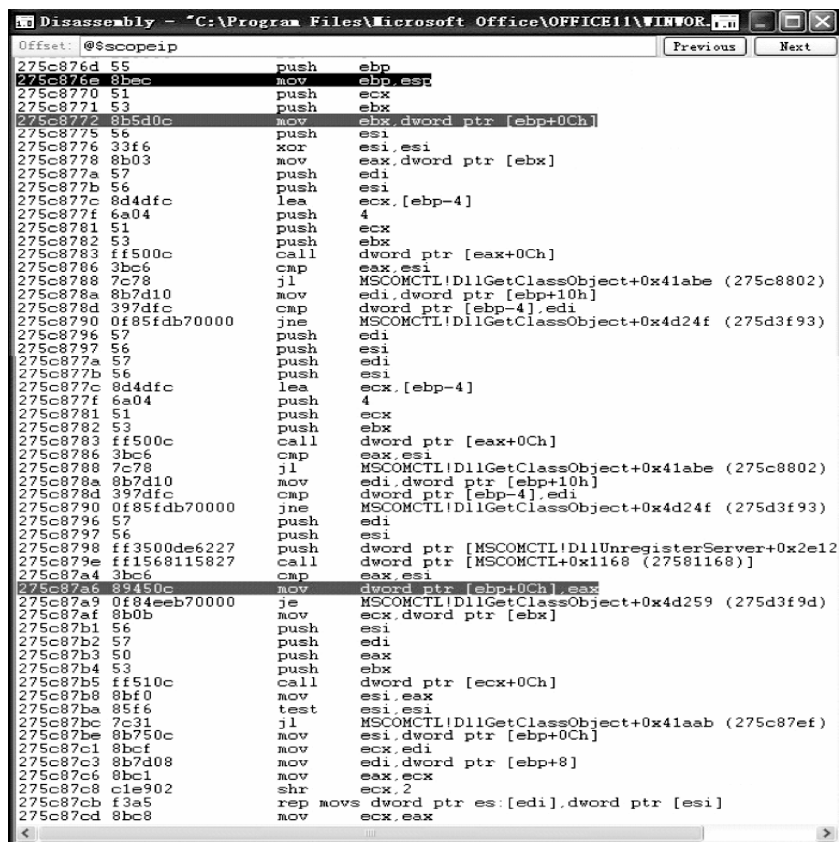


图 8 MSCOMCTL.GetClassObject 部分关键代码

是否大于或等于 0x08（参见图 7 地址 0x275c89f3 处的 `cmp dword ptr [ebp-0Ch], 8` 指令）。如果小于 0x08，则不读取数据，并退出当前数据块的解析；如果大于或等于 0x08，则继续调用函数 `MSCOMCTL.GetClassObject` 从数据块中取 4 个字节的数据至栈区。根据图 8 地址 0x275C878D 处

的 `cmp dword ptr [ebp-4], edi` 指令, 校验读入栈区的数据是否和 `[ebp-0Ch]` 相等 (参见图 7 地址 `0x275c89fd` 处的 `push dword ptr [ebp-0Ch]` 指令)。如果不相等, 则不读取数据, 并退出当前数据块的解析; 如果相等, 则从数据块中读取 `[ebp-0Ch]` 字节数据至栈区。

根据以上分析发现,程序刚开始只开辟了 0x14 大小的栈空间,第 1 次读取了 0x0C 字节数据至该区域。如果该区域中的最后 4 个字节大于或等于 0x08,那么程序会继续读取数据至栈区。如果在接下来调用 MSCOMCTL.GetClassObject 时,数据校验成功,就会读入超过 0x08 个字节数据至栈区,从而造成栈溢出。

### 3.3 RTF 的 OLE 对象漏洞分析流程

根据 3.2 节对 OLE 对象漏洞的分析,得到 RTF 中的 OLE 对象漏洞分析流程,如图 9 所示。

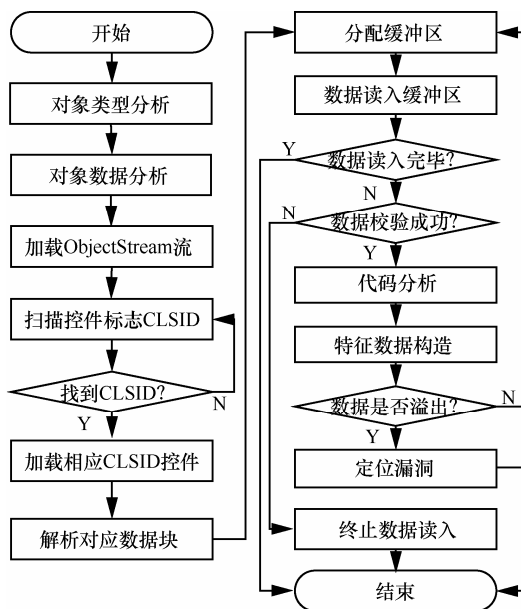


图9 RTF 的 OLE 对象漏洞分析流程

从图 9 可以看出,在解析 RTF 文件时,首先确定包含的 OLE 对象类型,并将对象数据读至缓冲区中;接着校验对象数据,并对读取校验成功数据的代码进行分析;最后根据代码执行流程,在数据块中相应位置构造特征数据,并判断是否会造成溢出。如果不溢出,则读取接下来的数据;如果溢出,则定位漏洞。

## 4 测试结果与分析

本节将对 3.3 节提出的 OLE 对象漏洞分析流程进行实验测试,实验环境为 Intel(R) Core(TM)i5-3230M CPU、4 GB 内存、OS 为 Microsoft Windows 7 SP1。首先,通过一个实例测试本方法的正确性;然后,针对不同 OLE 对象的漏洞进行测试,并与现有工具进行比较,证明本方法具有更高的检测

效率;最后,通过构造畸形测试用例来测试本方法的抗干扰性。

### 4.1 实例测试

根据 3.3 节的分析流程,OLE 对象数据是能否触发漏洞的关键,只有在 OLE 对象类型中构造合适的对象数据才能造成缓冲区溢出,触发漏洞。在 RTF 的 \objocx 对象类型中,首先通过 3.2 节对 \objdata 对象数据的 Contents 数据块尾部分析,发现由 Contents 数据块尾部的第 1 至第 4 字节所构成的 8 位 16 进制数等于 0x0C,且第 5 至第 8 字节的 8 位 16 进制等于 0x6A626F43 时,即 (4321)H=0x0C, (8765)H=0x6A626F43,才能将指定大小的数据复制到缓冲区中。该复制数据的大小由 Contents 数据块尾部的第 1 至第 4 字节构成的二进制数决定,即(4321)H。根据图 9 中的数据读入和数据校验判断条件构造 Contents 数据块尾部第 1 至第 8 字节的特征数据为 0C000000436F626A,这样可以使得第 1 次校验成功,而且会第 2 次调用 MSCOMCTL.GetClassObject (参见图 8)。

接着,根据图 9 中的数据溢出判断条件,通过特征数据构造方法,将 \objdata 对象数据的 Contents 数据块尾部第 9 至第 20 字节的数据构造成为 640000008800000088000000。最后,根据 Contents 数据块尾部第 9 至第 20 字节构造的数据在 Contents 数据块尾部的剩余数据中构造 0x88 大小的特征数据 aaaa…。这样,在第 2 次的 MSCOMCTL.GetClassObject 函数调用中,地址 0x275C879E 处调用 Kernel32.HeapAlloc 函数(参见图 8)开辟一段大小 0x88 字节堆空间,在地址 0x275C87B5 处调用函数 OLE32.CExposedStream::Read(参见图 8)继续从数据块中读取大小为 0x88 字节数据至分配的堆空间中。最后,在地址 0x275C87CB 处调用指令 rep movs dword ptr es:[edi],dword ptr ds:[esi](参见图 8)将这些数据读取至栈区,如图 10 所示。

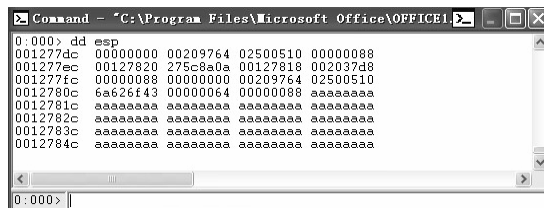


图10 栈溢出

从图 10 可以看出，在数据块中构造的数据 aaaa...成功读入栈区，并且造成溢出。通过这个实例证明了本文分析方法的正确性。

## 4.2 与其他工具比较测试

下面通过 Windbg 的脚本调试功能，创建一个模拟 Word 2010 解析 RTF 中 OLE 对象漏洞流程的脚本文件。然后，通过 Windbg 对不同的漏洞样本进行测试，并与现有的 RTFScan<sup>[18]</sup>和 pyOLEScanner<sup>[19]</sup>工具进行比较。这里选择 10 个 Word 漏洞进行测试，包括最常攻击利用的漏洞（CVE-2010-3333、CVE-2012-0158、CVE-2012-1856、CVE-2013-3906、CVE-2014-1761 和 CVE-2014-6357）以及 2015 年新发现的 OLE 对象漏洞（CVE-2015-1770、CVE-2015-2424、CVE-2015-1641 和 CVE-2015-2369）。为了更好地对比 3 种不同的检测方法，对每个 CVE 编号的漏洞通过构造不同特征数据、变换 Shellcode 及其加密方式，并添加额外正常 OLE 控件等规则生成 20 个测试样本（总共 200 个测试样本），表 4 记录了采用不同的方法检测到的样本数量。

表 4 本文方法与其他工具的测试结果对比

| CVE 编号        | RTFScan | pyOLEScanner | 本文方法 |
|---------------|---------|--------------|------|
| CVE-2010-3333 | 7       | 13           | 0    |
| CVE-2012-0158 | 8       | 11           | 18   |
| CVE-2012-1856 | 7       | 15           | 15   |
| CVE-2013-3906 | 0       | 0            | 8    |
| CVE-2014-1761 | 2       | 10           | 0    |
| CVE-2014-6357 | 2       | 8            | 9    |
| CVE-2015-1770 | 0       | 12           | 14   |
| CVE-2015-2424 | 0       | 0            | 12   |
| CVE-2015-2369 | 0       | 0            | 11   |
| CVE-2015-1641 | 2       | 13           | 17   |

从表 4 可以看出，对于 CVE-2012-0158、CVE-2012-1856、CVE-2013-3906、CVE-2014-6357、CVE-2015-1770、CVE-2015-2424、CVE-2015-2369 和 CVE-2015-1641，本文方法的检测效果明显优于其他两种方法，由于这 8 个漏洞都是典型的 OLE 对象漏洞，根据 3.3 节，通过找到控件标识符 CLSID，随后加载控件调用指定的控件解析对应数据块，而这 8 个漏洞的 CLSID 和控件都可以定位到，接着在读取数据至缓冲区时进行代码分析，并通过

样本中的特征数据测试定位到漏洞。对于 CVE-2010-3333 和 CVE-2014-1761，本文方法没有检测出异常样本，经分析发现这两个漏洞分别是 RTF 绘图属性缓冲区溢出漏洞和 RTF 数组溢出漏洞，所以本文方法检测不到该样本。此外，对本文方法未检测出的其他漏洞样本分析发现，主要是由于对象类型叠加或构造的特征数据未造成溢出，从而检测不到异常。pyOLEScanner 的检测效率优于 RTFScan，这是由于 pyOLEScanner 可以检测文档中是否还有宏和 ActiveX 等控件。例如，通过对 OLE 对象数据结构进行分析，发现在解析 CVE-2015-1770 漏洞样本数据时，加载了 ActiveX 控件 OSF.DLL，而 pyOLEScanner 存在 OSF.DLL 的解析模块，因此可以检测到该漏洞。RTFScan 相比其他两种方法，其检测效率最差，能够检测到的各个漏洞测试样本数量有限，主要原因是 RTFScan 靠检测文档中的 Shellcode，随着 Shellcode 以及加密技术的发展，文档中的 Shellcode 越来越难以通过特征匹配的方式定位，因此对于比较复杂的漏洞样本文件，RTFscan 几乎很难检测到。

## 4.3 抗干扰性测试

RTF 解析程序不但可以正确地解释基于文献 [12]语法规则生成的 RTF 文件，而且能够忽略它不知道或者未使用的控制字，并且能正确地略过被\*控制符标记的部分。因此，RTF 文件中往往存在没有完全符合语法规则的数据格式。在实际的漏洞样本中，为了躲避检测，经常会在样本中插入一些无意义的数据干扰漏洞检测。由于本文提出的方法需要对\object、\objocx、OLE 起始标志位以及 CLSID 进行分析，因此本测试采用网络上公开的 300 例测试用例对本文方法的抗干扰性进行测试<sup>[22]</sup>。在这 300 例测试用例中，分别对\object 和\objocx 采用空格等字符进行混淆，对 OLE 起始标志位和 CLSID 采用转义字符进行混淆，表 5 分别列出了不同字段未检测到的测试用例数目。

表 5 抗干扰测试结果

| 类型        | 未检测到测试用例 |
|-----------|----------|
| \object   | 35       |
| \objocx   | 24       |
| OLE 起始标志位 | 10       |
| CLSID     | 15       |

在表 5 中可以看出，在 300 例的测试样本中，总共有 84 例漏洞样本未检测出，其中\object 控制字 35 例，\objdata 控制字 24 例，OLE 起始标志位 10 例，CLSID 15 例。通过对测试样本的数据分析发现，\object 和\objocx 的混淆影响检测结果最严重。当样本中添加空格混淆，例如在样本中控制字可以表现为\object 等类似变形的字符串时，由于本文解析包含 OLE 对象的 RTF 文件时是通过查找标记为\object与\objocx 的字符串来确定包含的 OLE 对象类型，因此对检测效果影响较大。CLSID 和 OLE 起始标志位由数据串组成，可以采用 d\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0cf11e0a1b11ae1 或者“\bin”等转义字符进行混淆。通过对测试样本的数据分析发现，采用转义字符方式混淆的样本数量不多。因此，CLSID 和 OLE 起始标志位混淆对检测效果影响较小。

## 5 结束语

软件安全漏洞的分析已经成为了当今国内外信息安全研究的热点, 如何解决文档型办公软件的安全漏洞问题也成为安全研究人员的重点研究课题, 现有一些学者在分析 Flash、PDF、DOC 文件漏洞方面取得了很大的进展, 但对于 RTF 文件中的 OLE 对象漏洞这一问题, 尚未提出较好的方法。本文分析了 RTF 的 OLE 对象机制及其安全问题, 针对在解析 RTF 文件的 OLE 对象属性时会产生安全漏洞进行分析, 并给出了 OLE 对象漏洞一般分析方法, 最后通过实验测试证明了该方法的有效性。

通过测试发现本文的方法在针对混淆样本的检测中,漏检率较高,但其抗干扰性方面还需要进一步提高。此外,由于本文在数据块解析过程中采用逆向分析,而逆向分析的过程往往是通过手工或脚本方式实现,这样会降低分析的效率,因此在下一步的工作中,重点研究如何自动化实现逆向分析。

## 参考文献:

- [1] NAGARAJU S S, CRAIOVEANU C, FLORIO E, et al. Software vulnerability exploitation trends[R]. 2013.
- [2] 吴世忠, 郭涛, 董国伟, 等. 软件漏洞分析技术进展[J]. 清华大学学报(自然科学版), 2012, 52(10):1309-1319.

sis: a road map[J]. Journal of Tsinghua University (Science and Technology), 2012, 52(10): 1309-1319.

- [3] 杨丁宁, 肖晖, 张玉清, 等. 基于 fuzzing 的 ActiveX 控件漏洞挖掘技术研究[J]. 计算机研究与发展, 2012, 49(7): 1525-1532.  
YANG D N, XIAO H, ZHANG Y Q, et al. Vulnerability detection in ActiveX controls based on Fuzzing technology[J]. Journal of Computer Research and Development, 2012, 49(7): 1525-1532.
- [4] 刘奇旭, 温涛, 闻观行, 等. Flash 跨站脚本漏洞挖掘技术研究[J]. 计算机研究与发展, 2014, 51(7): 1624-1632.  
LIU Q X, WEN T, WEN G X, et al. Detection of XSS vulnerabilities in online flash[J]. Journal of Computer Research and Development, 2014, 51(7): 1624-1632.
- [5] REBERT A, CHA S K, AVGERINOS T, et al. Optimizing seed selection for Fuzzing[C]//The 23rd USENIX Security Symposium, San Diego. 2014: 861-875.
- [6] 史飞悦, 傅德胜. 缓冲区溢出漏洞挖掘分析及利用的研究[J]. 计算机科学, 2013, 40(11): 143-146.  
SHI F Y, FU D S. Research of buffer overflow vulnerability discovering analysis and exploiting[J]. Computer Science, 2013, 40(11): 143-146.
- [7] 乐德广, 章亮, 郑力新, 等. 面向 RTF 文件的 Word 漏洞分析[J]. 华侨大学学报 (自然科学版), 2015, 36(1): 17-22.  
LE D G, ZHANG L, ZHENG L X, et al. Research on word vulnerability analysis for the RTF file[J]. Journal of Huaqiao University (Natural Science), 2015, 36(1): 17-22.
- [8] LEATHERY J. Microsoft office zero day CVE-2015-2424 [EB/OL]. <http://www.isightpartners.com/2015/07/microsoft-office-zero-day-cve-2015-2424-leveraged-by-tsar-team/>.
- [9] LI H F, SUN B. Attacking interoperability: an OLE edition [EB/OL]. <https://www.blackhat.com/docs/us-15/materials/us-15-Li-Attacking-Interoperability-An-OLE-Edition.pdf>.
- [10] MICROSOFT. OLE concepts and requirements overview [EB/OL]. <https://support.microsoft.com/en-us/kb/86008>.
- [11] MICROSOFT. The microsoft office visualization tool (OffVis) [EB/OL]. <http://www.microsoft.com/en-us/download/details.aspx?id=2096>.
- [12] MICROSOFT. Rich text format (RTF) specification version 1.9.1 [S]. 2008.
- [13] MICROSOFT. Microsoft security bulletin MS12-060 [EB/OL]. <https://technet.microsoft.com/en-us/library/security/ms12-060>.
- [14] MICROSOFT. Microsoft security bulletin MS12-027 [EB/OL]. <https://technet.microsoft.com/en-us/library/security/ms12-027.aspx>.
- [15] MICROSOFT. Microsoft security bulletin MS13-096 [EB/OL]. <https://technet.microsoft.com/en-us/library/security/ms13-096.aspx>.
- [16] 王明华, 凌凌云, 冯登国. 基于异常控制流识别的漏洞利用攻击检测方法[J]. 通信学报, 2014, 35(9): 20-31.  
WANG M H, YING L Y, FENG D G. Exploit detection based on illegal control flow transfers identification[J]. Journal on Communications, 2014, 35(9): 20-31.
- [17] HUND R, WILLENS C, HOLZ T. Practical timing side channel attacks against kernel space ASLR[C]// IEEE Symposium on Security and Privacy (SP). Berkeley, c2013: 191-205.
- [18] BOLDEWIN F. OfficeMalScanner/RTFScan [EB/OL]. <http://www.reconstructor.org/>.
- [19] BONFA G E. PyOLEScanner [EB/OL]. <http://www.aldeid.com/wiki/PyOLEScanner>.

- [20] X-WAYS AG. WinHex hex editor [EB/OL]. <http://www.winhex.com/winhex/hex-editor.html>.
- [21] MICROSOFT. Debugging tools for windows [EB/OL]. <https://msdn.microsoft.com/en-us/library/ff551063.aspx>.
- [22] PARKOUR M. Malicious files for signature testing and research [EB/OL]. <http://contagiodump.blogspot.de/2013/03/16800-clean-and-11960-malicious-files.html>.



**龚声蓉** (1966-), 男, 江苏苏州人, 博士, 常熟理工学院计算机科学与工程学院教授, 主要研究方向为图像处理与信息安全等。

#### 作者简介:



**乐德广** (1975-), 男, 福建三明人, 博士, 常熟理工学院计算机科学与工程学院副教授, 主要研究方向为计算机网络安全与下一代互联网技术等。



**郑力新** (1967-), 男, 福建泉州人, 华侨大学教授, 主要研究方向为图像处理与信息安全等。



**章亮** (1990-), 男, 安徽合肥人, 华侨大学硕士生, 主要研究方向为信息安全与网络攻防等。



**吴少刚** (1973-), 男, 安徽宿松人, 博士, 主要研究方向为计算机系统安全、结构, 并行与分布式计算等。