doi:10.11959/j.issn.1000-436x.2017104

RTF 数组溢出漏洞挖掘技术研究

乐德广[1,2,3]，龚声蓉[1]，吴少刚[3]，徐锋[3]，刘文生[4]

(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. 中科梦兰电子科技有限公司，江苏 常熟 215500；4. 泉州市公安局公共信息网络安全监察支队，福建 泉州 362000)

摘 要：在虚函数执行中，由于错误操作 C++对象的虚函数表而引起数组溢出漏洞。通过攻击虚函数造成系统崩溃，甚至导致攻击者可直接控制程序执行，严重威胁用户安全。为尽早发现并修复此类安全漏洞，对该安全漏洞的挖掘技术进行深入研究，结合 MS Word 解析 RTF 文件和虚函数调用之间的联系，发现 MS Word 在解析异常的 RTF 文件时存在数组溢出漏洞，并进一步提出基于文件结构解析的 Fuzzing 测试方法来挖掘 RTF 数组溢出漏洞。在此基础上，设计了 RTF 数组溢出漏洞挖掘工具（RAVD，RTF array vulnerability detector）。通过 RAVD 对 RTF 文件进行测试，能够正确挖掘出数组溢出漏洞。实际的模糊测试表明，设计的工具相比传统的漏洞挖掘工具具有更高的挖掘效率。

关键词：RTF 文件；漏洞挖掘；Fuzzing 测试；数组溢出

中图分类号：TP393 **文献标识码**：A

Research on RTF array overflow vulnerability detection

LE De-guang[1,2,3], GONG Sheng-rong[1], WU Shao-gang[3], XU Feng[3], LIU Wen-sheng[4]

(1. School of Computer Science & Engineering, Changshu Institute of Technology, Changshu 215500, China;
2. School of Computer Science and Technology, Soochow University, Suzhou 215006, China;
3. Lemote Electronic Technology Co., Ltd., Changshu 215500, China;
4. Public Information Network Safety Supervision Division, Quanzhou Municipal Public Security Bureau, Quanzhou, 362000, China)

Abstract: When the virtual function was executed, it could cause array overflow vulnerability due to error operation of the virtual function table of C++ object. By attacking the virtual function, it could cause the system crash, or even the attacker to control the execution of program directly was allowed, which threatened user's security seriously. In order to find and fix this potential security vulnerability as soon as possible, the technology for detecting such security vulnerability was studied. Based on the analysis of the virtual function call during the MS Word parsing RTF files, the array overflow vulnerability generated by MS Word parsing abnormal RTF files, and a new RTF array overflow vulnerability detection method based on the file structure analytical Fuzzing was proposed. Besides, an RTF array overflow vulnerability detection tool (RAVD, RTF array vulnerability detector) was designed. The test results show RAVD can detect RTF array overflow vulnerabilities correctly. Moreover, the Fuzzing results show RAVD has higher efficiency in comparison with traditional file Fuzzing tools.

Key words: RTF document, vulnerability detection, Fuzzing test, array overflow

1 引言

计算机软件技术的飞速发展和广泛应用，大大方便了人们日常的社会和经济生活。与此同时，软件的安全问题也日益突出，软件漏洞是安全问题的根源之一[1]。近年来，利用软件漏洞尤其是文件型

收稿日期：2016-11-08；**修回日期**：2017-04-05

基金项目：国家自然科学基金资助项目（No.61202440，No.61402057）；江苏省产学研前瞻性联合研究基金资助项目（No.BY2016050-01）；江苏省科技计划基金资助项目（No.BK20160411）

Foundation Items: The National Natural Science Foundation of China (No.61202440, No.61402057), The Production and Research Prospective Joint Research Project of Jiangsu Province (No.BY2016050-01), The Jiangsu Provincial Natural Science Foundation of China (No.BK20160411)

软件漏洞在未授权情况下对计算机系统进行访问或破坏的现象日益严重。各种文件型软件漏洞引发的信息窃取、资源被控和系统崩溃等问题给人们造成了严重的经济损失，并对国民经济和社会稳定产生重大威胁。因此，对软件漏洞，尤其是文件型软件漏洞的研究成为近年来的热点[2,3]。

文件型软件漏洞是指应用程序处理不同格式文件时由于设计上的缺陷，可能演变成为对系统产生威胁的安全漏洞。目前，办公软件应用广泛，因此，一些流行的办公软件如 MS Office 和 PDF 系列软件等成为文件型软件漏洞的重灾区，只要发现这类漏洞，就可以通过构造特定格式的文件触发漏洞，从而引发安全性问题，而作为隐患发现技术之一的漏洞挖掘技术，越来越受到人们的重视[4]。

RTF 文件作为微软 MS Word 可处理的文档格式之一，广泛应用于办公场合。由于 RTF 文件使用的广泛性，针对 RTF 格式的文件型漏洞也层出不穷，如堆溢出、格式化溢出、整数溢出和数组溢出等[5,6]。其中，数组溢出漏洞是由于 C++虚函数自身的安全性问题所造成的。根据多态性原理，虚函数通过虚表指针来寻找虚函数入口地址间接被调用。这种通过指针对象调用虚函数已经被发现存在漏洞利用[7~11]。本文结合 RTF 文件编目表属性和 C++虚函数机制，在 RTF 文件编目表属性和 C++虚函数机制分析基础上，研究二者的安全性问题。通过对二者的安全性分析，基于逆向工程提出一种改进的文件格式 Fuzzing 方法，实现对 RTF 数组溢出漏洞的挖掘技术，并设计和实现 RTF 数组溢出漏洞挖掘工具。实验测试结果表明，该 RAVD 工具能有效地挖掘 RTF 数组溢出漏洞。

2　研究背景

缓冲区溢出是通过重写函数在栈区的返回地址控制程序执行流程的一种传统漏洞[12]。目前，国内外许多专家已经提出了多种针对缓冲区溢出攻击的防御技术，包括金丝雀防护[13]、数据执行保护[14]以及自定义分配保护堆[15]。这些防御技术使传统缓冲区溢出攻击越来越难执行。近年来，出现一种通过替代重写函数在栈区的返回地址来控制程序数据流的漏洞，即数组溢出漏洞[7]。该漏洞通过控制堆中的虚表指针，当对象调用虚函数时，可以控制程序的执行流程，甚至可以执行 shellcode[8]。下面，将对 C++虚函数实现机制进行分析，研究 C++对象在内存中的布局和调用，并在此基础上研究如何通过数组溢出伪造虚表指针。

2.1　C++多态性与虚函数

在 C++中，多态性允许对派生类对象通过父类指针调用成员函数来执行代码。C++中实现多态的一个重要机制是虚函数[16]。其中，虚函数是通过虚表和虚表指针的引用来调用呈现运行时的多态性。对于有虚函数声明的类，它的每个对象都拥有一个指向虚表的指针，虚表中存放类的所有虚函数地址。下面，通过反汇编技术分析虚函数机制，这里，分别从单继承和多继承 2 种情况对虚函数机制进行分析。

2.2　单继承

首先，建立一个单继承下的 Win32 控制台程序。然后，通过反汇编工具 IDA pro 对该程序进行反汇编分析，得出虚函数的底层实现机制。C++源码和反汇编代码对应关系如图 1 所示。

在图 1 中，左边的 C++源码定义了 2 个虚函数 virtual void vf1()和 virtual void vf2()以及一个数据成员变量。右边为反汇编后的代码。从该汇编代码可以得出虚函数调用过程：1) 通过 LEA ECX, DWORD PTR SS:[EBP−8]获取对象 b1 地址，记为 addr；2) 在调用构造函数过程中，通过 MOV DWORD PTR DS:[EAX], OFFSET virtual.??_7base@@6B@ 得到虚表指针 vptr: virtual.??_7base@@6B@，并将其保存在对象内存空间中，即[addr]=vptr；3) 直接调用虚函数，如果已经明确要调用的是自身成员函数，由于没有构成多态性，如 b1.vf1()，将直接调用该虚函数 CALL virtual.0040106E；4) 间接调用虚函数，如果通过对象指针的方式调用虚函数，即通过对象指针引用（如 p->vf1()，p->vf2()）的方式调用，便会构成多态性，虚函数的调用将使用间接寻址的方式。通过 MOV EDX, DWORD PTR DS:[ECX]和 CALL NEAR DWORD PTR DS:[EDX]调用虚函数 vf1()。根据调用过程 2)可知，地址[EDX]其实就是[vptr]；vf2()的调用则是通过 MOV EDX, DWORD PTR DS:[EAX]和 CALL NEAR DWORD PTR DS:[EDX+4]完成，即[EDX+4]=[vptr+4]，虚函数 vf2 入口地址相对 vf1 偏移 4 byte。

从虚函数调用过程可以看出，每个包含虚函数的类创建一张虚表，用于放置各虚函数在函数跳转表中的地址。当创建带有虚函数的类对象时，编译

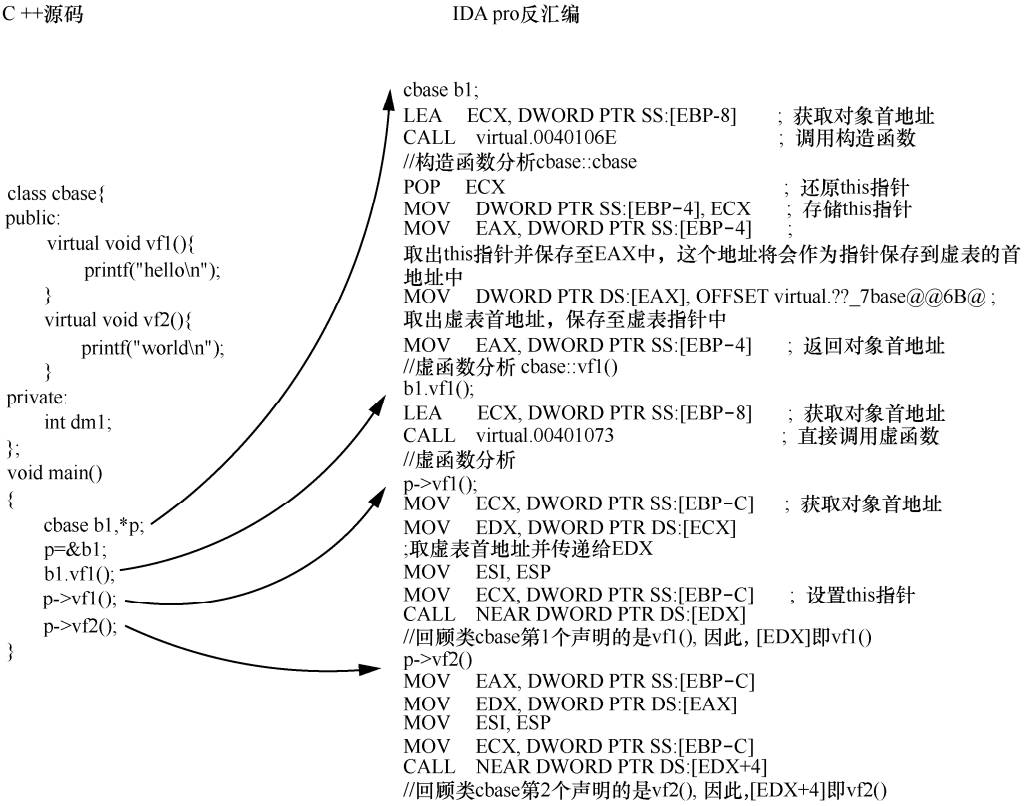

图 1 单继承 C++源码和反汇编代码对应关系

器会在对象的开始处放置一个 vptr 的指针，指向所属类的虚表。在 C++中，根据对象定义，普通成员函数、静态成员函数以及静态成员变量将不会占用对象空间的内存区域，而占用内存区域的是普通成员变量以及虚函数。因此，得出包含虚函数类的对象内存布局，如图 2 所示。

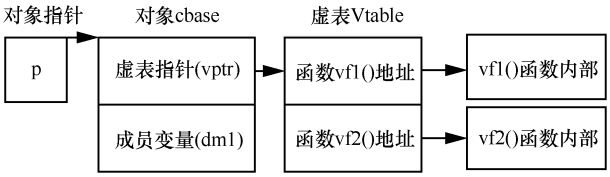

图 2 单继承对象在内存中的布局

从图 2 可以看出，在内存中虚表指针和成员变量的地址相邻，并且虚表指针地址在前，成员变量地址在后。虚表指针指向虚表，在虚表中存放虚函数地址，虚函数地址也彼此相邻。

2.3 多继承

在多继承下，一个类具有多个包含虚函数的父类，编译器会为它创建多个虚表。每个虚表中各虚函数的顺序与相应的父类一样，在类对象中各父类继承部分的开始处存放虚表指针，指向相应的虚表。利用父类指针可以指向子类的特性，可间接调用各子类中的虚函数。为分析多继承下虚函数的实现机制，创建一个包含继承类的情况，代码如下所示。

```
class base1{
public:
    virtual void vf1(){
        printf("hello\n");
    }
    virtual void vf2(){
        printf("hello\n");
    }
private:
    int dm1;
};
class base2{
public:
    virtual void vf3(){
        printf("world\n");
    }
    virtual void vf4(){
        printf("world\n");
    }
```

```
private:
    int dm2;
};
class derived:public base1,public base2{
public:
    virtual void vf5(){
        printf("hello world\n");
    }
    virtual void vf6(){
        printf("hello world\n");
    }
private:
    int dm3;
    int dm4;
}
```

在以上代码中，定义了包含成员变量的类 base1、base2 和 derived，其中，子类 derived 继承了父类 base1 和 base2。通过反汇编分析上述源码，得出多继承情况下对象在内存中的布局，如图 3 所示，p1 为父类 base1 的对象指针，p2 为父类 base2 的对象指针。

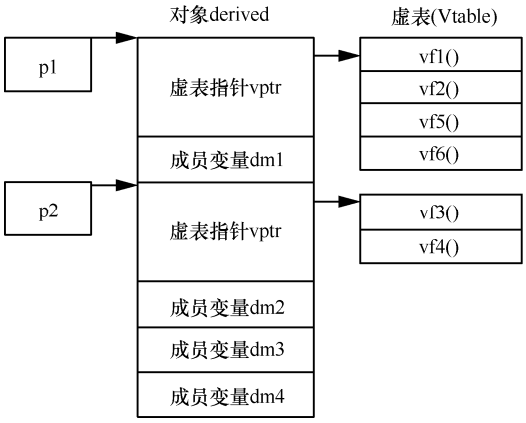

图 3　多继承对象在内存中的布局

从图 3 可以看出，多继承时内存中每个含有虚函数的类都有一张虚表 Vtable，子类虚表继承它各个父类的虚表，因此，父类虚表中包含的某项，子类虚表中也包含同样的项。

2.4　虚函数安全性分析

根据 2.2 节和 2.3 节分析，如果内存中存在多个对象，并且这些对象在内存中排列是连续性时，当某一个对象中的成员变量由于溢出，将多余数据覆盖到下一个对象空间中。由于相邻对象中的内容为虚表指针，溢出的结果就会修改邻接对象的虚表指针。一旦虚表指针被修改，在调用虚函数时就会造成内存访问出错。对象是在内存空间中，而且成员变量的溢出要求是连续性的覆盖，因此，这里的溢出称之为数组溢出，如图 4 所示。

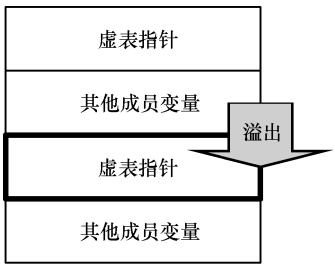

图 4　溢出邻接对象的虚表

当存在这种数组溢出漏洞时，就可以通过控制对象中的成员变量的溢出，修改对象中的虚表指针或修改虚表中的虚函数指针。当程序在调用虚函数时，就能够控制程序的执行流程，并实现让程序执行 shellcode，图 5 显示了数组溢出漏洞的攻击流程。

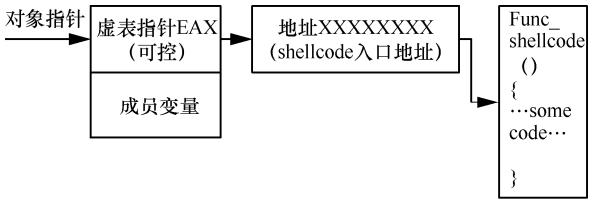

图 5　存在数组溢出漏洞的攻击流程

从图 5 可以看出，如果虚表指针 EAX 被溢出可控，就可以伪造一个虚表地址，而该虚表地址中存放着虚函数地址，如果这些虚函数地址指向 shellcode 的入口地址，那么在调用这些虚函数时就会执行 shellcode，最终实现对漏洞的攻击。

3　RTF 数组溢出漏洞分析

由于 MS Word 程序在解析 RTF 文件时，会使用 C++多态性特点，因此，需要调用并实现虚函数机制，如在解析 RTF 文件的编目表时就是利用多态性编程实现。根据 2.4 节的虚函数机制安全性分析，当 MS Word 在解析 RTF 文件时，尤其是解析编目表时就有可能产生数组溢出漏洞。因此，本节将通过逆向工程研究 MS Word 在解析 RTF 文件编目表时会存在虚函数调用，并且分析可能存在的数组溢出漏洞。

3.1　RTF 编目表

在 RTF 文件中，所有的项目符号和编号的格式信息都被存到编目表中[17]。因此，编目表担当了样

式表的角色，而每一个独立的段落只需存储一个到该编目表的索引，并不需要将这类格式的数据单独存在每一个段中。在 RTF 文件规范中存在 2 个编目表：编目表（\listtable 引用）和编目覆盖表（\listoverridetable 引用）。其中，编目覆盖表有少数顶级关键字，包括\listoverridecount，它包含格式要被覆盖的层次数目，其值依赖于这个将要被覆盖的列表是简单的还是混合、多级的。下面是编目表应用于 RTF 文件实例。

{*\listoverridetable{\listoverride\listid1026640128\listoverridecount0\ls1}

{\listoverride\listid1026640128\listoverridecount9

{\lfolevel\listoverridestartat\levelstartat0}{\lfolevel\listoverridestartat\levelstartat1}

{\lfolevel\listoverridestartat\levelstartat1}{\lfolevel\listoverridestartat\levelstartat1}

{\lfolevel\listoverridestartat\levelstartat1}{\lfolevel\listoverridestartat\levelstartat1}

{\lfolevel\listoverridestartat\levelstartat1}{\lfolevel\listoverridestartat\levelstartat1}

{\lfolevel\listoverridestartat\levelstartat1}\ls2}

以上实例中，关键字\listoverridetable 代表编目覆盖表。\listoverride 指的是编目覆盖项的列表。\listid*N* 是编目的 ID，需要精确地与编目表中的其中一个编目的\listid 相匹配。其中，*N* 值为长整型。\listoverridecount 是要被覆盖的层次数目。在 RTF 文件规范中，这个数目规定为 1 或 9。\lfolevel 是指编目覆盖层覆盖列表，所有的实际覆盖信息存储于编目覆盖层覆盖列表（\lfolevel 引用）中。\ls*n* 是编目覆盖项的索引，其中，*n* 是按照\listoverride 的顺序取值为 1~*N*。

3.2 编目表解析数组溢出漏洞分析

在 3.1 节中，对 RTF 文件的编目表进行了分析。本节将构造一个包含编目表的 RTF 文件，其 RTF 文件关键数据格式如下。

{\rtf1{{\listoverride\listid1094795585\listoverridecount9

{\lfolevel}{\lfolevel}{\lfolevel}{\lfolevel}

{\lfolevel}{\lfolevel}{\lfolevel}{\lfolevel}

{\lfolevel}

\ls1}}}

然后，调用 MS Word 对以上 RTF 文件进行解析，并通过 WinDbg 对其进行动态地逆向分析[18]，其关键解析代码如图 6 所示。

图 6 关键解析代码

从图 6 可以看出，该段代码中函数的调用方式和 2.2 节中虚函数的调用方式一样，首先通过 MOV EAX,DWORD PTR [EBP+8] 获取对象地址。然后，通过 MOV ECX,DWORD PTR [EAX]传递虚表指针 vptr，即 vptr=[EAX]。最后，通过 CALL DWORD PTR[ECX+4]间接调用虚函数，即[vptr+4]= [ECX+4]。因此，可以证明 Word 在解析 RTF 文件编目表时存在虚函数调用。

根据 RTF 文件关键源码以及 3.1 节可知，\listoverridecount 覆盖层次的数目和\lfolevel 编目覆盖层覆盖列表数目是相对应的。而\listoverridecount 的数目被限定为 1 或 9，因此，\lfolevel 的数目也将为 1 或 9。在本例中，\listoverridecount 的数目为 9，\lfolevel 的数目同样也是 9。进一步逆向分析，可以发现程序是通过\listoverridecount 声明的数目分配给\lfolevel 相应的内存空间，通过 WinDbg 内存窗口查看 MS Word 在解析\listoverridecount 长度时所分配的空间在内存中的布局，如图 7 所示。

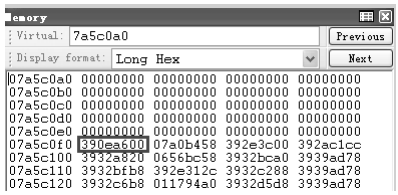

图 7 解析\listoverridecount 的数据在内存中布局

在图 7 中，0x00 区域即为\lfolevel 分配的内存空间，紧接着内存空间的数据（图 7 中方框内数据）0x390ea600 是对象指针。根据前面的分析可知，MS Word 在解析 RTF 文件编目表时存在虚函数调用，因此，这类对象指针所指向的内存空间是虚函数的入口地址，如图 8 所示。

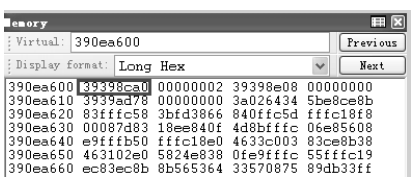

图 8 虚函数地址内存空间布局

在图 8 中，存放着一系列虚函数的入口地址，如 0x39398ca0（图 8 中方框内数据），因此，0x390ea600 对象指针即为虚表指针 vptr。如果\listoverridecount 数目与\lfolevel 数目不匹配，如\listoverridecount 数目小于\lfolevel 数目，当 MS Word 在解析编目表时，多余的\lfolevel 很可能就会造成数据溢出，而从图 7 的内存布局中可以看出，只要数据溢出就会覆盖邻接虚表指针 0x390ea600，根据 2.4 节可以知道，溢出结果就会造成数组溢出漏洞。

4　RTF 数组溢出漏洞挖掘

通过第 2 节和第 3 节的分析可知，MS Word 在解析 RTF 文件时存在数组溢出漏洞。因此，本节将重点研究如何挖掘 RTF 文件中数组溢出漏洞。

4.1　文件结构解析 Fuzzing

在漏洞挖掘中，本文使用 MS Word 对 RTF 文件进行解析，MS Word 是按照事先约定好的数据结构对文件中不同数据域进行解析，因此，可以采用文件格式 Fuzzing[19]方式进行漏洞挖掘。文件格式 Fuzzing 的关键就是如何有效地构造畸形数据文件，这种畸形数据通常称作半有效数据。目前，现有的文件格式 Fuzzing 一般无需了解文件本身格式，通过模糊模板文件中提供完全随机的输入或简单改变某些字节数据来生成畸形文件，称为暴力 Fuzzing。由于 MS Word 使用复杂的数据结构，当 MS Word 解析复杂的文件格式（如 RTF、DOC 文件）时，经常要经过多层解析[20]。如果采用现有的暴力 Fuzzing 方法盲目构造半有效数据，生成的畸形测试文件很可能在还没有到达指定层次时就被上层解析器阻断，而且容易产生大量无效文件，因此，漏洞挖掘效率和成功率非常低。

为弥补现有暴力 Fuzzing 中构造方法的不足，本文提出一种基于文件结构解析 Fuzzing 的漏洞挖掘方法。该方法通过逆向分析文件结构中指定数据块的解析逻辑，从而明确文件结构中指定数据块与解析该数据块代码之间的关系，实现定向 Fuzzing。这样不仅能够控制 Fuzzing 过程，而且降低 Fuzzing 工作的数据空间，相比暴力 Fuzzing，提高了漏洞挖掘效率和成功率。

本文采用这种改进的 Fuzzing 方法实现对 RTF 文件数组溢出漏洞的挖掘。首先，通过分析 RTF 文件结构确定不同字段在 RTF 文件结构中的组织方式。然后，通过逆向技术分析 MS Word 如何解析这些字段。一旦出现疑似数组溢出漏洞代码的情况，这些字段将会成为构建 RTF 文件结构的关键数据块。最后，通过 RTF 文件结构来定义文件中的关键数据块，并在此基础上对关键数据块中各个部分进行 Fuzzing，这样构造的模板文件不仅能够通过数据正确性以及一致性检测，而且也符合程序对关键数据块的解析逻辑。下面将详细介绍基于文件结构解析 Fuzzing 的 RTF 数组溢出漏洞挖掘工具（RAVD）的设计与实现。

4.2　RAVD 设计与实现

4.2.1　RAVD 总体设计

RAVD 基于文件结构解析 Fuzzing 技术，对解析 RTF 文档的 Word 程序进行安全性测试，寻找可能存在的 RTF 数组溢出漏洞。图 9 给出了 RAVD 的体系结构，它包含 RTF 文件输入数据结构化分析模块、RTF 测试文件构建模块、执行应用模块和异常检测模块。

从图 9 可以看出，RTF 文件输入数据结构化分析模块分析 RTF 文件结构中的各数据块字段的信息，包括表示长度信息、偏移值以及逻辑信息等，判断在解析时是否存在虚函数调用。RTF 测试文件构建模块根据 RTF 文件输入数据结构化分析模块提供的信息，按照一定的规则构建测试用例，进行定向 Fuzzing 测试。执行应用模块启动 MS Word 执行测试用例，同时，启用内建调试器监控符合数组溢出漏洞的异常。异常检测模块检测在执行应用模块所产生的异常，如果产生异常则记录该异常文件以及异常的具体信息，同时，过滤非文件结构解析所产生的无效异常。

4.2.2　RTF 文件输入数据结构化分析模块

依据文件结构解析原理，需要清楚 RTF 文件结构中指定数据块的解析逻辑，因此，在这个模块中首先需要对数据块中各个关键字段进行分析。通常针对 RTF 文件进行结构化分析的关键字段主要有：1) 表示数据长度字段；2) 表示数据偏移字段；3) 可变长度的数据字段；4) 可能引起应用程序执行不同逻辑的字段。出现以上 4 种字段都需要分析其解析逻辑。再分析该字段与解析该字段的代码之间的对应关系，如果在解析该字段的过程中出现虚函数调用的情况，如出现类似以下代码。

```
MOV EAX,对象指针
MOV EDX,[EAX]    //获取虚表指针
```

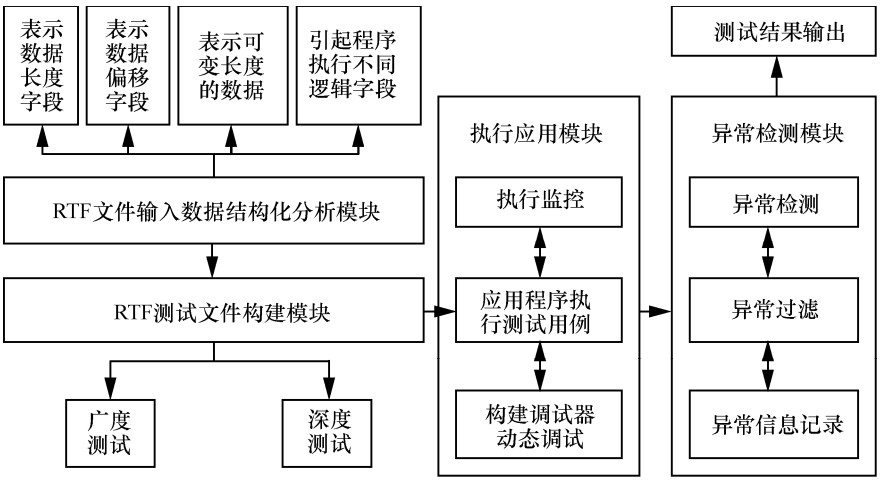

图 9 RAVD 体系结构

CALL [EDX+4*X] //调用虚函数，

其中，X 是类 A 中的第 X 个虚函数。X=0,1,2,3… 说明该字段存在数组溢出漏洞的可能，那么模板文件的文件结构中就必须包含该字段的数据块。

4.2.3 RTF 测试文件构建模块

传统文件格式 Fuzzing 工具，如 MiniFuzz[21]和 FileFuzz[22]，构建测试文件的策略都是通过对一个模板文件的指定字节、字或双字进行数据替换，在指定位置替换或插入数据等数据变化规则。这种 Fuzzing 方式存在数据变化空间难以控制、产生大量无效测试文件、尤其是对于复杂的文件格式通过随机修改数据的方式往往存在无法穿透软件过滤机制等局限性。为了弥补传统构建测试文件方法存在的不足，本文采用的方法是在输入数据结构化分析产生的模板文件基础上构建测试文件。在 4.2.2 节中，通过文件结构解析过程能够找到可能触发漏洞的字段，并以这些字段作为文件结构生成模板文件进行定向 Fuzzing。由于本文 RAVD 主要用于挖掘数组溢出漏洞，因此，在定向 Fuzzing 的时候结合广度测试和深度测试[14]。使用广度方法时，测试的是模板文件结构中的各个字段，包含多字段的 RTF 测试模板文件结构如图 10 所示。

顶级字段（标识）	字段1	数据内容 (length, value…)	…	字段n	数据内容 (length, value…)

图 10 RTF 测试模板文件结构

在图 10 中，模板文件存在多个可能触发漏洞的字段。为了提高测试效率，采取的方法是同时测试这些字段，而不是逐一对某个字段进行测试。在广度测试中，对于不同字段的数据内容所使用的 Fuzzing 方法也不一样，具体可以分为以下 3 种。

1) 边界值。在测试这类字段的数据内容时，选择极限边界值作为测试数据往往比较容易触发代码运行时潜在的安全问题，如选择整型值的上溢、下溢以及符号溢出，还可以选择边界值的 $\frac{1}{2}$ 或 $\frac{1}{4}$ 等，以及其附近的值作为测试数据。如对\listoverridecount 字段的内容进行测试数据选择时，可以选择 0x00、0x5A、0xFF 和 0x80 等。2) 替换字符串。除了常见的 ASCII 码用于替换数据，还可以选择当前数据本身替换，如可以用多个\lfolevel 替换原来数据。3) 字段分隔符。采用非字母数字字符，如空格和制表符等，这些字符通常被用作字段分隔符和终止符。

在广度测试过程中，一旦出现崩溃，从寄存器数据来看，并不能确认是否能够控制该崩溃发生，在这种情况下，需要重点关注产生崩溃的字段位置，针对这些字段在该位置处尝试所有可能的数据，这种测试方法称为深度测试。通过崩溃的结果，可以知道能够在多大程度上控制该崩溃，如果随着数据的变化，崩溃发生的位置或寄存器的值持续变化，说明在变异文件时使用的数据对异常结果产生一定的影响。

4.2.4 执行应用模块

构建好 RTF 测试文件之后，接下来，需要在 MS Word 中运行这些文件。如果在测试过程中 MS Word 产生崩溃，则需要确定是哪个测试文件导致 MS Word 发生崩溃，因此，需要在执行测试用例时对 MS Word 进行动态调试和异常信息监控。

识别异常信息最好的方法就是使用调试器。一种方法是直接利用 OllyICE 程序的实时监视功能来

实现对软件运行错误的监视[23]。在测试过程中，一旦发生错误，OllyICE 就会立即运行起来，这样就可以实时监视 MS Word 的运行状态。OllyICE 实时监视的优点在于既能识别已处理的异常，又能识别未处理的异常。虽然 OllyICE 能够立即发现被测试文件错误，但是并不能记录当前发生错误的测试文件。因此，本文提出一种基于内建调试器方法实现动态调试和异常信息监控，即利用 Windows 提供的调试接口实现动态调试功能，在执行测试的过程中内建调试器捕获异常信息，图 11 为该方法的操作流程。

从图 11 可以看出，首先需要调用 Windows API 函数 CreateProcess 创建子进程，该进程用于启动目标应用，接着在 CreateProcess 的 dwCreationFlags 参数中设置 DEBUG_PROCESS 标识，这样就可以在启动目标应用的同时调试进程。通过传入 CreateProcess 的命令行参数 lpCommandLine，开始调试进程。如果没有发生超时，那么就会一直监视调试事件，通过 API 函数 WaitForDebugEvent 等待调试事件。这里在 Windows 的调试事件中，只关注发生异常时的调试事件 EXCEPTION_DEBUG_EVENT，当发生这类调试事件时返回调试信息。通过这种内建调试器捕获异常信息的方式，RAVD 能够在每次启动目标应用后监视异常，不需要手动附加调试进程，同时可以记录异常信息。

4.2.5　异常检测模块

异常检测模块是在动态调试的基础上，对异常产生的结果进行过滤和记录。一般模糊测试需要识别每一种异常信息，但是对于漏洞利用，需要确定能否控制程序流程。目前，控制程序流程的方式主要有写可控的内存地址、写可控的数据、读可控的内存地址和读可控的数据等。根据数组溢出漏洞原理，异常是由于对象指针错误引起的，因此，在 RAVD 调试的异常信息中主要记录读写内存异常和访问内存异常。记录的异常信息包括以下 3 种：1) 异常发生的地址以及地址所在模块。由于 MS Word 在解析 RTF 文件时经常会调用动态链接库，因此，有必要记录异常的模块，以缩小后续分析的范围；2) 异常发生时所有的寄存器值。该寄存器值可作为测试数据造成异常的判定条件之一；3)异常发生地址附近处的汇编指令。数组溢出漏洞由于代码的特定性，因此需要对产生异常的汇编指令进行分析判断。

最后，通过 RAVD 测试所产生的异常信息以图形界面的形式反馈给用户，用户可以根据这些可视化的异常信息进一步分析是否能够获取程序的执行流程从而执行 shellcode。

5　测试与分析

为验证 RAVD 的正确性和有效性，本文实验环境为 E5-2620 CPU、32 GB RAM、操作系统为 Windows 7 SP1。采用暴力 Fuzzing（MiniFuzz 和 FileFuzz）和文件结构解析 Fuzzing（RAVD）2 种方法，并选取真实的测试用例对 RTF 文件数组溢出漏洞进行测试与分析。其中，FileFuzz 是 Michael Sutton 开发的开源 Fuzzer，本文用于测试的版本是

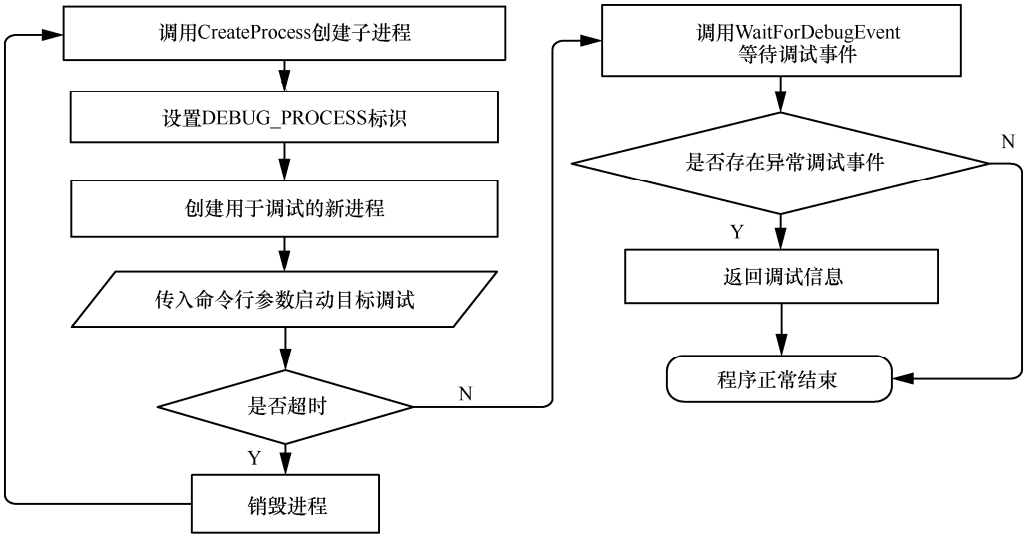

图 11　内建调试器调试过程

1.0.2510；MiniFuzz 是微软公司开发的文件 Fuzzing 工具，本文用于测试的版本是 1.5.5.0。

5.1 示例测试与分析

首先，针对不同的关键字段通过自主编写的测试用例进行评估。研究发现 Word 程序在解析 RTF 编目表时存在数组溢出漏洞，因此，针对 RTF 编目表的各个字段都应该进行测试，但是由于编目表包含太多的字段，并不能对所有的字段都进行结构化分析，因此，在测试时选择编目表项中一些典型的关键字段，然后在 RTF 模板文件中添加这些编目表字段，这些字段如表 1 所示。最后通过 FileFuzz、MiniFuzz 和 RAVD 这 3 款工具对构造好的模板文件进行测试。

在 RAVD 中使用广度测试和深度测试方法，对这些字段的数据同时进行测试，在 FileFuzz 和 MiniFuzz 中使用传统的盲测方法进行测试。如果检测出某个字段异常，用“√”标识，否则用“×”标识。测试结果如表 1 所示。

表 1 不同关键字段用例测试结果比较

RTF 关键字段	FileFuzz	MiniFuzz	RAVD
\listoverridecount	√	×	√
\lfolevel	×	×	√
\listid	×	×	×
\listoverride	×	×	×
\listsimple	×	×	×
\levelnfc	×	×	×
\levelnfc	×	×	×
\listhybrid	√	√	√
\leveljc	×	×	×
\levelnfcn	√	√	√
\levelold	√	×	√
\levelnumbers	×	×	×

从表 1 可以看出，RAVD 检测出了数组溢出漏洞中比较重要的 2 个异常字段\listoverridecount 和\lfolevel，而 FileFuzz 只能检测出一个异常字段\listoverridecount，MiniFuzz 未能检测到这 2 个异常字段。对于编目表中的其他字段，FileFuzz 能检测出的异常字段如\listhybrid、\levelnfcn 和\levelold，RAVD 都能检测到，而 MiniFuzz 只能检测到\listhybrid 和\levelnfcn 的异常。因此，RAVD 不仅能够有效检测出已知字段的异常，还能够检测出更多潜在的异常字段，具有更高的准确性。

5.2 模糊测试与分析

接着，采用上述 2 种方法对模板文件中的数据进行模糊测试。FileFuzz 和 MiniFuzz 采用盲测方法，因此，直接将文件中的所有数据都看成是二进制数据，对这些二进制数据进行字节级别的替换，通过控制测试用例的数量分别生成 100、300、500 和 1 000 例测试用例，RAVD 生成测试用例的策略则是通过 RTF 编目表关键字段的表示长度的数据、逻辑数据以及内容可变区数据进行广度测试和深度测试，同样通过控制测试用例的数量分别生成 100、300、500 和 1 000 例测试用例，最后在执行这些测试用例时记录 MS Word 出现异常的次数，测试结果如表 2 所示。

表 2 模糊测试异常次数比较

测试用例数量	FileFuzz	MiniFuzz	RAVD
100	0	0	2
300	2	0	6
500	7	3	16
1 000	13	12	42

从表 2 可以看出，使用 FileFuzz 工具每 1 000 例只能得到 13 例异常情况，MiniFuzz 也只能产生 12 例异常，而 RAVD 产生异常的数量是 42 例，是 FileFuzz 和 MiniFuzz 产生异常数量的 3 倍多。在前 500 例中，FileFuzz 只能产生 7 例异常，MiniFuzz 产生 3 例异常，而 RAVD 产生异常的数量是 16 例，是 FileFuzz 产生异常数量的 2 倍多，是 MiniFuzz 产生异常数量的 5 倍多。比较测试结果发现，FileFuzz 和 MiniFuzz 的畸形数据是采用随机函数生成伪随机数据作为输入变量，未结合虚函数调用机制和 RTF 文件结构分析来构造测试用例。因此，采用改进后的文件结构解析 Fuzzing 方法能够得到更多的异常，使 Fuzzing 效率得到提高。

对 RAVD 检测出的 42 例异常样本进行分析，发现由\listoverridecount 和\lfolevel 字段产生的异常样本中，当\listoverridecount 的数目与\lfolevel 的数目不匹配时，如当\lfolevel 数目大于或等于 29，且\listoverridecount 的参数值 N 为 25、26、27 或 28 时，RAVD 检测到的异常信息地址附近的汇编指令显示存在虚函数的调用。进一步分析发现，当 N 为这些不同的值时，不同位置的虚表指针被覆盖，因此属于数组溢出漏洞。对\levelnfcn 字段产生的异常样本分析发现，当\levelnfcn 的参数值 N 为 249、265、

278 或 279 时，RAVD 检测到的异常信息地址附近的汇编指令显示是由于指针错误引用导致程序产生崩溃，进一步通过逆向分析发现 MS Word 在解析\levelnfcn 的参数值 *N* 时，将 *N* 的值转化为相应的指针值，因此，*N* 的变化直接影响指针值，当 *N* 为某些值时可能会造成内存指针的错误引用从而产生异常。对\levelold 字段的异常样本分析发现，当\levelold 的值为 123、245 或 258 时产生异常，这种情况和\levelnfcn 字段的异常情况一样，都是由于指针错误引用产生的异常。对\listhybrid 产生的异常样本分析发现，当 *N* 的值为 35 510 时产生异常，造成异常的原因是由于微软的附件执行服务机制引发的异常处理机制。

此外，在测试过程中还需关注模板文件中各个关键字段的危险程度，因此，有必要对 RTF 文件中危险关键字段分布进行分析。下面，对表 2 中 RAVD 产生的 42 例异常中的关键字段进行分析统计。由于在广度测试时是针对模板文件每个字段进行同时测试，因此，可以根据不同字段在测试时产生异常的表现来衡量一个字段的危险程度，如图 12 所示，罗马数字 I 表示\listoverridecount 字段，II 表示\levelold 字段，III 表示\lfolevel 字段，IV 表示\levelnfcn 字段，V 表示\listhybrid 字段。

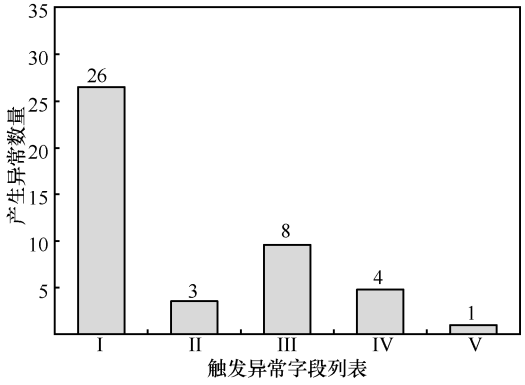

图 12　漏洞危险字段分布

从图 12 可以看出，在 RAVD 测出的异常数目中，\listoverridecount 和\lfolevel 字段占据 34 例，相比其他 3 个字段占据的比例最大，说明这 2 个字段在数组溢出漏洞中危险程度最高，同时能够检测出\levelnfcn、\levelold 和\listhybrid 字段的异常数量是 8 例，相比以上 2 个字段，这 3 个字段的危险程度较低。

5.3　实例测试与分析

最后，通过对已知漏洞的检测判断 RAVD 能否从已知漏洞中正确检测出数组溢出漏洞。这里选择 17 个典型的 MS Word 解析 RTF 文件产生的漏洞进行测试，包括早期的漏洞（CVE-2006-2492、CVE-2008-1091、CVE-2008-4028、CVE-2010-0134、CVE-2010-1902、CVE-2012-2528、CVE-2012-2539）、最常利用的漏洞（CVE-2010-3333、CVE-2012-0158、CVE-2012-1856、CVE-2013-3906、CVE-2014-1761），以及近年来新发现的漏洞（CVE-2015-1641、CVE-2015-1770、CVE-2015-2424、CVE-2015-2369、CVE-2016-0010）。本文测试采用的测试用例来自 Metasploit 渗透测试平台上的漏洞利用 POC[24]，如果某个漏洞样本可以被检测出，用“√”标识，否则为“×”，测试结果如表 3 所示。

表 3　　测试使用的 Word 漏洞

CVE ID	触发漏洞字段	触发漏洞模块	检测结果
CVE-2006-2492	\smarttags	mso.dll	√
CVE-2008-1091	\do	mso.dll	×
CVE-2008-4028	\dppolyline	mso.dll	×
CVE-2010-0134	\ls	rtfsr.dll	×
CVE-2010-1902	\do	mso.dll	×
CVE-2012-2528	\listid	winword.exe	×
CVE-2012-2539	\listoverridecount	wwlib.dll	×
CVE-2010-3333	\pFragments	mso.dll	×
CVE-2012-0158	\object	mscomctl.ocx	×
CVE-2012-1856	\object	mscomctl.ocx	×
CVE-2013-3906	\pict	ogl.dll	√
CVE-2014-1761	\listoverridecount	wwlib.dll	√
CVE-2014-1761	\lfolevel	wwlib.dll	√
CVE-2015-1641	\smarttags	msvcr71.dll	√
CVE-2015-1770	\object	osf.dll	×
CVE-2015-2424	\object	mscomctl.ocx	×
CVE-2015-2369	\object	rapi.dll	×
CVE-2016-0010	\pict	winword.exe	×

从表 3 可以看出，在 18 例测试样本中，总共有 13 例漏洞样本未检测出数组溢出，有 5 例漏洞样本检测出数组溢出漏洞。通过对测试样本的数据分析发现，在检测出的 5 例样本中，CVE-2014-1761、CVE-2006-2492 和 CVE-2015-1641 是由于对象指针错误引起的，进一步分析发现是由于虚表指针的错误引用造成的，因此，属于数组溢出漏洞。而 CVE-2013-3906 是由于 MS Word 在处理 RTF 文件

中嵌入的 TIFF 图形时造成的整数溢出漏洞，不属于数组溢出漏洞。因此，本次测试的错检率为 20%。

在未检测出的 13 例样本中，CVE-2008-1091 、CVE-2008-4028、CVE-2010-1902 和 CVE-2016-0010 是由于 MS Word 在处理 RTF 文件中的绘图对象造成的堆溢出漏洞；CVE-2010-0134 和 CVE-2010-3333 是由于 MS Word 分别在处理 RTF 文件中\ls 和 \pFragments 属性时产生的栈溢出漏洞；CVE-2012-2528 和 CVE-2012-2539 是由于 MS Word 分别在处理 RTF 文件\listid 和\listoverridecount 属性时造成的内存释放后的重用漏洞；CVE-2012-0158、CVE-2012-1856 和 CVE-2015-2424 这 3 个漏洞都是由于 MS Word 在处理 RTF 文件中嵌入的 MSCOMCTL.OCX 相应控件时造成的内存破坏漏洞；CVE-2015-1770 是由于无法正确处理内存中的对象产生的漏洞；CVE-2015-2369 是由于不正确处理动态链接库（DLL）文件造成的漏洞。以上 12 例都不属于数组溢出漏洞，因此，本文测试的漏检率为 0%。通过该实验得出本文的 RAVD 检测工具存在一定的错检率，但是却有很好的漏检率。

6 结束语

本文研究数组溢出漏洞挖掘技术，首先分析 C++虚函数存在安全性问题，进而研究 Word 在解析 RTF 文件的过程中存在的数组溢出漏洞问题。针对该问题，本文提出了一种改进的文件格式 Fuzzing 方法，即基于文件结构解析 Fuzzing 的漏洞挖掘技术，并实现了相应的漏洞挖掘工具 RAVD。实际测试结果验证了该工具在数组溢出漏洞挖掘中的正确性和有效性。

由于本文在文件结构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的方式去实现，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。此外当文件结构中的字段数量达到一定程度时，测试效率是否仍然能够保持高效性，如果不能保持高效性，需要对测试方法做相应的改进。最后，基于文件结构解析思想的模糊测试是否还能挖掘到其他类型的漏洞，也是在未来工作中要研究的内容。

参考文献：

[1] CABALLERO J，LIN Z. Type inference on executables[J]. ACM Computing Surveys, 2016, 48(4): 65.

[2] HUANG S K, HUANG M H, HUANG P Y, et al. Software crash analysis for automatic exploit generation on binary programs[J]. IEEE Transactions on Reliability, 2014, 63(1):270-289.

[3] 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.

[4] MASSACCI F, NGUYEN V H. An empirical methodology to evaluate vulnerability discovery models[J]. IEEE Transactions on Software Engineering, 2014, 40(12):1147-1162.

[5] 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.

[6] 乐德广, 章亮, 龚声蓉, 等.面向 RTF 的 OLE 对象漏洞分析研究[J]. 网络与信息安全学报, 2016, 2(1): 34-45.
LE D G, ZHANG L, GONG S R, et al. Research on OLE object vulnerability analysis for RTF file[J]. Chinese Journal of Network and Information Security, 2016,2(1):34-45.

[7] 王清. 0day 安全软件漏洞分析技术[M]. 北京市:电子工业出版社. 2011:345-346.
WANG Q. 0day security: software vulnerability analysis techniques[M]. Beijing: Publishing House of Electronics Industry. 2011:345-346.

[8] DEWEY D, GIFFIN J T. Static detection of C++ vtable escape vulnerabilities in binary code[C]//19th Annual Network and Distributed System Security Symposium (NDSS). 2012:1-14.

[9] JANG D, TATLOCK Z, LERNER S. SAFEDISPATCH: securing C++ virtual calls from memory corruption attacks[C]//21th Annual Network and Distributed System Security Symposium (NDSS). 2014: 1-15.

[10] PRAKASH A, HU X, YIN H. VfGuard: strict protection for virtual function calls in COTS C++ binaries[C]//22th Annual Network and Distributed System Security Symposium (NDSS). 2015: 1-15.

[11] BOUNOV D, KLCL R G, LERNER S. Protecting C++ dynamic dispatch through VTable interleaving[C] //23th Annual Network and Distributed System Security Symposium (NDSS). 2016:1-15.

[12] 李舟军, 张俊贤, 廖湘科, 等. 软件安全漏洞检测技术[J].计算机学报, 2015, 38(4): 717-732.
LI Z J, ZHANG J X, LIAO X K, et al. Survey of software vulnerability detection techniques[J]. Chinese Journal of Computers, 2015, 38(4): 717-732.

[13] COWAN C, PU C, MAIER D, et al. StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks[C]//7th Conference on USENIX Security Symposium (USENIX). 1998:5-15.

[14] STOJANOVSKI N, GUSEV M, GLIGOROSKI D, et al. Bypassing data execution prevention on microsoft Windows XP SP2[C]//The Second International Conference on Availability, Reliability and Secu-

rity. 2007:1222-1226.

[15] KHARBUTLI M, JIANG X W, SOLIHIN Y, et al. Comprehensively and efficiently protecting the heap[J]. ACM Sigops Operating Systems Review, 2006, 40(5): 207-218

[16] ZHANG C, CARR S A, LI T X, et al. VTrust: regaining trust on virtual calls[C]//23th Annual Network and Distributed System Security Symposium (NDSS). 2016:1-15.

[17] RTF 1.9.1. Rich text format (RTF) specification[S]. Microsoft Corporation, 2008.

[18] VOSTOKOV D. Windows debugging: practical foundations[M]. Monkstown: Opentask Publisher, 2009:79-81.

[19] LI J X, XU X, LIAO L J, et al. Concolic execute Fuzzing based on control-flow analysis[C]//11th International Conference on Computational Intelligence and Security (CIS). 2015: 385-389.

[20] MS-DOC 6.1. Word (.doc) binary file format[S]. Microsoft Corporation, 2017.

[21] OUYANG Y J, ZENG S, CAO Y, et al. A region-sensitive Fuzzing test based on multi-objective programming[J]. Lecture Notes on Software Engineering, 2016, 4(2): 116-122.

[22] HU C J, Li Z J, MA J X, et al. File parsing vulnerability detection with symbolic execution[C]//6th IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE), 2012: 135-142.

[23] COHN R, RUSSELL J. OllyDbg[M]. VSD Publisher, 2012:24-26.

[24] KENNEDY D, O'GORMAN J, KEARNS D, et al. Metasploit: the penetration tester's guide[M]. San Francisco: No Starch Press, 2011: 56-58.

作者简介：

乐德广（1975-），男，福建三明人，博士，常熟理工学院副教授，主要研究方向为信息安全与下一代互联网技术等。

龚声蓉（1966-），男，湖北天门人，博士，常熟理工学院教授、博士生导师，主要研究方向为图像处理与信息安全等。

吴少刚（1973-），男，安徽宿松人，博士，中科梦兰电子科技有限公司研究员，主要研究方向为计算机系统结构、并行与分布式计算等。

徐锋（1981-），男，江苏常熟人，中科梦兰电子科技有限公司高级工程师，主要研究方向为计算机体系结构及自主安全。

刘文生（1969-），男，福建泉州人，泉州公安局高级工程师，主要研究方向为网络安全。