一种基于反汇编技术的二进制补丁分析方法*)

曾 鸣[1] 赵荣彩[1] 王小芹[2] 姚京松[1]

(中国人民解放军信息工程大学计算机科学与技术系 郑州 450002)[1]

(清华大学计算机科学与技术系 北京 100084)[2]

摘 要 软件开发商通过向用户提供补丁程序来修改软件中存在的安全漏洞。但随着安全漏洞研究者不断提高分析安全补丁的能力和速度,厂商开始向公众封闭与安全补丁相关的漏洞技术细节,仅提供软件打补丁前后的二进制代码,由此引发了二进制代码比较技术研究的热潮。二进制代码比较技术的目的是定位执行代码间的差异,从而获得补丁所修补的漏洞细节。本文提出了一种基于反汇编技术,定位执行代码间语义差异,从而完成二进制安全补丁分析的方法。描述了该技术模型、系统框架和关键技术,并通过实践证明此方法可以快速有效地定位安全补丁所修补的软件漏洞。

关键词 补丁分析,反汇编,安全漏洞,IDA

Security Patches Analyzing Through Disassemble Technology

ZENG Ming[1] ZHAO Rong-Cai[1] WANG Xiao-Qin[2] YAO Jing-Song[1]

(Department of Computer Science, the PLA Information and Engineering University, Zhengzhou 450002)[1]

(Department of Computer Science, Tsinghua University, Beijing100084)[2]

Abstract Software vulnerability is a primary cause of information system insecurity. Patches are always used by software vendors to amend mistakes in software-system which causes safety problems. Patches analyzing technology tries to find out details of the vulnerability that has been corrected. To avoid this, software vendors refused to offer details of patches to the public, usually only binary versions before and after patching is provided. This drives researchers to analyze patches through binary comparing. A method based on disassemble technology to do patches comparing is presented in this article. Following crucial technologies are discussed in detail: 1)function reorganization and description in binaries. 2)For a function in one binary version, how to find its peer in another version with the same function. 3)how to remove difference caused by compiler. A framework implementing the described method is presented. As a practical example, a security update that fixes a crucial vulnerability in Internet Explore is analyzed.

Keywords Patches analyzing, Disassemble, Security vulnerability, IDA

1 引言

软件系统在发行之初,总是难以避免各种安全上的缺陷和漏洞。为了修补软件的这些 bug,软件开发商会提供相应的修补程序,这种修补程序一般称为"补丁"。补丁分析技术的目的就是通过分析软件开发商提供的补丁相关信息,定位出补丁所修补的软件漏洞。对于开放源代码的软件,补丁本身就是一些程序源代码,打补丁的过程就是用补丁中的源代码替换原有的代码。但对封闭源代码的软件系统,厂商只提供修改之后的二进制代码,微软的安全补丁公告就是典型的例子。这一类的补丁分析具有相当大的难度,本文主要的讨论就是针对这一类补丁的分析技术。

从某种意义上来说,补丁以及伴随其发布的相关信息,就是漏洞研究者的"指南针"。漏洞研究者根据所提供的蛛丝马迹,经过反复的跟踪、测试和试验,是有可能定位软件漏洞所在的。为了避免这种情况,厂商近些年开始向公众封闭与补丁相关的漏洞技术细节。漏洞研究者唯一能够得到的就是软件系统在打补丁前后的二进制代码,由此掀起了补丁二进制代码比较技术研究的热潮。补丁二进制比较技术通过比较同一个软件系统的不同二进制版本,揭示出它们的差异信息,找到它们在语义上的变化。这种技术在安全防护、病毒变种分析、利用其他产品未公开的特性、提防别有用心的黑客等方面都有着广泛的应用。补丁二进制比较有着诸多的难点:①由于信息的不对称,对源码简单修改后重新编译,就会导致逆向工程中对目标代码的分析需要完全重做来检测修改。②一个补丁往往补多个问题,使得代码变化较多;③编译器的一些优化,如指令顺序的颠倒、寄存器选择的不同等,增加了比较的难度。这些难点使得传统的补丁比较方法不能很好地工作,如二进制字节对应比较只适用于若干字节变化的补丁比较。而通过直接对反汇编后的文本进行比较,由于缺乏对程序逻辑的理解,只适用于小文件和少量变化。针对这些问题,国际上已出现了一些新的方法,见文[1,2]。本文提出了一种基于反汇编技术的二进制补丁分析方法,描述了其模型、系统框架和关键技术。它将补丁分析分在两个层面进行:一个是函数层面,一个是汇编指令层面。相对于文[1]基于图的同构的指令比较分析方法,本算法的复杂度比较低,能够兼容编译器对补丁分析的影响,正确识别出一些编译器造成的二进制代码间的差异(如指令重排序),减少差异信息的误报。相对于参考文[2]中结构化的比较方法,它能够报告出一些指令层次的差异信息,从而识别出通过变化静态变量或者增加缓冲区大小而被"补上"的漏洞,防止漏报。

2 模型描述和系统框架

2.1 模型描述

*)国家"863"计划基金资助项目"网络安全积极防御关键技术"(2003AA146010)。曾 鸣 博士,主要研究方向:信息安全、软件系统安全;赵荣彩 教授,博导;王小芹 高工;姚京松 高工。

本模型是基于反汇编技术的，采用了编译原理中的几个概念。

函数调用图：描述一个二进制文件函数之间的调用关系的有向图，可以表示为 $a=\{\{f_1,\cdots,f_n\},\{f_1^e,\cdots,f_m^e\}\}$，其中 $f_1,\cdots,f_n$ 是有向图中的节点，表示二进制文件中的 n 个函数；$f_1^e,\cdots,f_m^e$ 是有向图中的 m 条边，表示函数之间的调用关系，f_i^e 代表了从函数 f_i 到函数 f_e 的一条边，说明 f_e 是 f_i 的子函数。

控制流程图(CFG)：一个函数所有的语句序列可以被分为若干基本块。基本块是一个连续的语句序列，控制流从它的开始进入，并从它的末尾离开，不可能有中断或者分支(末尾除外)。划分程序指令序列形成基本块的方法见参考文[3]。控制流图的节点由基本块组成，表示计算；节点之间的边表示控制流向。每一个函数控制流图都只有一个入口(入度为 0 的点)，但可以有多个出口(出度为 0 的点)。图 1 给出了一个函数控制流图的示例，包含了 5 个基本块、6 条边。

补丁分析过程中，二进制文件经过反汇编技术的处理，可以被抽象为上述的两张有向图。使用 A 代表软件打补丁前的版本，其函数调用图可表示为：$a=\{\{a_1,\cdots,a_n\},\{a_1^e,\cdots,a_l^e\}\}$；使用 B 来描述软件打补丁后的版本，函数调用图表示为：$\beta=\{\{b_1,\cdots,b_n\},\{b_1^e,\cdots,b_k^e\}\}$。A 和 B 中的每一个函数都采用一个 CFG 来描述。补丁分析的目的是找出 A 和 B 之间语义上的差异之处。

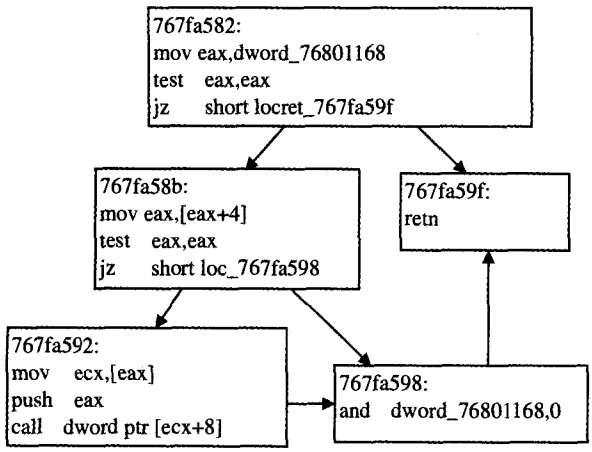

图 1　函数控制流图

2.2　系统框架

在上述模型之下，按照以下的步骤进行补丁分析：

①利用反汇编技术分析二进制代码，分别为补丁前后的软件版本 A 和 B 生成函数调用图和函数控制流图。

②利用两种有向图信息，对于 A 中的每一个函数 a_i，找到它在 B 中功能相对应的函数 b_j。考虑到补丁自身的特殊性，A 和 B 版本的软件中，大部分代码是相同的，所以对于 A 中绝大多数的函数，在 B 中都能找到与其相应功能的函数同其配对。对于无法配对的函数，应当被视为两个版本的差异，代表新增的功能或者修改。

③对于配对的函数 (a_i,b_j)，进一步通过函数流程图提供的信息进行比较，确定这两个对应函数之间有没有语义上的变化。比较过程中如果发现函数配对发生错误，要返回第二步，调整中间配对结果。

系统架构如图 2 所示，其中虚线框内是补丁分析的主要流程，其他模块是为了提高分析准确率和有效性的辅助模块。编译器优化处理消除编译器造成的版本 A 和 B 之间的差异，避免误报。在整个分析过程中，人工专家可以调整每一步的中间分析结果，使得最终的结果更准确。

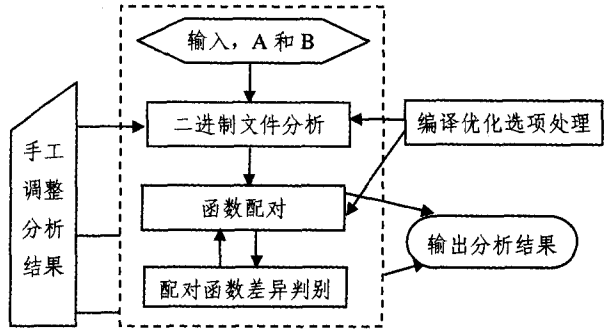

图 2　系统框架图

3　系统中的关键技术

3.1　二进制文件的函数识别和配对问题

3.1.1　函数识别和描述

在二进制代码的层面上，不存在源码中一个个清晰的函数。为了对版本 A 和 B 中的函数进行配对，首先要对二进制文件进行反汇编。通过诸如 IDA-Pro 之类的反汇编工具[4]，可以分析出二进制文件所包含的函数集合。将 A 的函数集合记作 $f_A=\{a_1,\cdots,a_n\}$，B 的函数集合记作 $f_B=\{b_1,\cdots,b_m\}$。

如果可以获得 A 和 B 的符号表，对于函数配对是非常有帮助的。在对二进制代码进行反汇编的时候加载符号表，可以得到函数名及参数、变量、调用的库函数名。名称相同的函数，以及唯一包含相同名称静态串的函数，都可以直接配对。

针对无法获取符号表，或者符号表不全的情况，这里介绍一种利用函数结构化特征进行函数配对的方法。这种方法用到了上述函数控制流图的概念。从函数流程图中提取能够代表函数功能特征的信息，称作函数的签名。A 和 B 中签名相同的函数，可以认为在功能上是相同的，从而进行配对。很明显，签名因素的选取至关重要，直接影响到配对的准确性。可以作为函数特征的信息有：函数参数、局部变量、函数控制流图中基本块和边的个数，以及函数调用子函数的个数。

函数参数的个数、类型，局部变量的个数或占用空间的大小，这些函数特征在源代码层面上可以很好地用来区分函数。但是在目标码层面上会出现一定的问题。比如，函数之间参数的传递有多种方式，通常编译器采用栈(push)的方式传参。但有的编译器采用寄存器直接传参，这时，参数的个数就无法准确确定，而字节对齐的要求又无法保障局部变量的个数或占用空间的大小得到统一。所以，参数和局部变量作为签名因素是不合适的。函数控制流图中的基本块和边的个数，较好地反映了函数的规模和流程。相当数量的补丁对软件漏洞修改都是通过判断某些条件，从而增加一些异常处理来进行的，此时其基本块和边的信息必然改变，将这两个特征是作为签名因素是能够有效显示差异的。另外，子函数个数也是一个反映函数特征不可缺少的因素。

基于以上的讨论，定义一个函数的结构化特征信息为一个函数的签名。具体描述如下。

定义 1　一个函数的签名为一个三元组 $\mathrm{Sig}(f_i)=(\alpha_i,\beta_i,r_i)$，其中 α_i 代表了函数 f_i 控制流图中的基本块个数，β_i 代表了 f_i 控制流图中的边数，γ_i 代表了 f_i 调用的子函数个数，即是函数调用图中节点 f_i 的出度。

3.1.2 基于函数签名的配对算法

在3.1.1节中函数签名定义的基础上，定义版本A中的函数α_i与版本B中的函数b_j配对，即$P(\alpha_i)=b_j$，当且仅当它们同时满足以下3个条件：

- α_i与b_j的签名是相同的；
- B中除了b_j，没有别的函数与α_i的签名相同；
- A中除了α_i，没有别的函数与b_j的签名相同。

满足以上条件的两个函数即可认为是相同的函数，表示为$p(\alpha_i)=b_j$且$p(b_j)=\alpha_i$。形式化描述如下：

$$P(a_j)=b_j \Leftrightarrow |\mathrm{Sig}^{-1}(\mathrm{Sig}(\alpha_i))|=1=|\mathrm{Sig}^{-1}(\mathrm{Sig}(b_j))| \wedge \mathrm{Sig}(\alpha_i)=\mathrm{Sig}(b_j) \quad (1)$$

根据式(1)进行函数配对之后，得到了从版本A中函数到版本B中函数的一个初始映射$p_1:A_1\rightarrow B_1$，其中$A_1\subset\{a_1,\cdots,a_n\}$，$B_1\subset\{b_1,\cdots,b_m\}$。

下面利用2.1节中介绍的函数调用图扩充p_1，生成一个映射序列$p_2,\cdots,p_h$，其中$A_1\subset A_2\subset\cdots A_h$，$B_1\subset B_2\cdots B_h$。扩充$p_t$的目的是使得版本A和B之间的函数尽可能多地配对。

由p_{i-1}计算p_i的迭代算法如下：

①取已经进入p_{i-1}的所有a_i和$p_{i-1(a_i)}$；

②通过函数调用图分别得到函数a_i和$p_{i-1(a_i)}$调用的子函数集合，分别记作A'_i，B'_i；

③使用类似构造p_1的方式构造$p'_i:A'_i\rightarrow B'_i$；

④对于$a_j\subset A_{i-1}$，令$p_{i(a_j)}=p_{i-1(a_j)}$。如果$a_j\not\subset A_{i-1}$但是③中增加了新的配对，则令$p_{i(a_j)}=p'_{i(a_j)}$。

由p_i可以计算p_{i+1}，这种计算一直持续到可以配对的函数对不再增加为止。

配对结果经过人工检查，有错误的地方可以进行调整，然后利用迭代算法扩充配对函数集合，这样可以防止错误的瀑布式传播。通过对多个补丁的分析表明，上述算法函数配比率达到95%，而在配比的近3万对函数中，发生配比错误的不超过10对。

3.2 已配对函数的差异判别

已经配对的函数(a_i,b_j)在新旧版本的二进制文件中实现相应的功能。对它们进行进一步的比较，目标是发现函数之间指令级别的差异。

3.2.1 基本块判同技术

对于已经配为一对的函数a_i与b_j，判断它们之间究竟有没有差异，可以在函数流程图的基础上，进行基本块中指令之间的比较。从入口基本块开始，逐一进行，将语义相同的基本块判同。无法判同的基本块中包含了函数的差异。系统中的基本块判同标准可以分为3个等级：第一等级最为严格，要求基本块内的所有指令除了自身地址以外的其他所有信息都被比较，包括命令符和参数，两个基本块被判同的要求是这些信息均相同。这种方法也可以用于一些未配对小函数的比较。补丁分析中常会遇到多个小函数，它们包含的汇编指令都是完全相同的，只有通过带操作数的比较才有可能将它们区分开来，这种方法可能会增加新的函数配对。第二等级是仅对指令的操作符进行比较，而不去判断操作数。第三等级是容许基本块内的指令顺序有所变化，由于编译器的优化经常会对基本块内部的指令顺序进行重排序，而重排后的指令在功能上与原序列并没有本质的差别，所以能够兼容指令顺序重排的基本块判同算法具有重要的意义。这3种等级的基本块判同算法在补丁的分析过程中综合使用，用在函数差异判断的不同子阶段。

第一和第二等级的基本块判同方法比较容易实现，可以采用简单的字符串比较。实现第三等级的判同具有一定的难度，Thomas Dullien在文[2]中提出了一种小素数乘积法，其原理是素数乘积分解的唯一性。

3.2.2 小素数乘积算法

用$\beta:=\{a_1,\cdots,a_m\}$表示汇编指令集合，共有$m$种汇编指令。定义$p_m=\{3,\cdots,\rho_m\}$为前$m$个素数，为$\beta$中每一个元素分配一个唯一的小素数，即构造如下的映射：$\tau:\beta\rightarrow p_m$，$\tau(\alpha_i)\rightarrow\rho_i$。

对于n条汇编语句组成的基本块a和b，分别计算其素数乘积$\prod_{i=1}^{n}\tau(\alpha_i)$，$\prod_{i=1}^{n}\tau(b_i)$。很明显，如果$\prod_{i=1}^{n}\tau(\alpha_i)=\prod_{i=1}^{n}\tau(b_i)$的话，说明$\mu$和$\nu$所包含的指令集合是相同的，指令次序可以相同，也可以不同。在补丁分析的大部分场合，这两个基本块可以被认同。这样就解决了3.2.1节中第三等级的基本块判同问题。

素数乘法是高开销的，乘积增长得非常快，在计算机中难以表示。为了实际应用算法，可以将乘积与2^{64}取模，这样就可以利用通用的x86-cpu寄存器来进行计算了。简化计算的风险是漏报：在$\prod_{i=1}^{n}\tau(\alpha_i)\neq\prod_{i=1}^{n}\tau(b_i)$的情况下，式子$\prod_{i=1}^{n}\tau(\alpha_i)=k2^{64}+c$和$\prod_{i=1}^{n}\tau(b_i)=j2^{64}+c$却可能成立。这样，实际有差异的$a$和$b$两个基本块可能被认为相同，从而造成差异信息的漏报。

可以证明上述情况成立的概率小于$(\frac{p_m^n}{2^{64}}-1)\frac{(m-1)!\,n!}{(n+m-1)!}$。在实际的补丁分析中，对于有$m=100$规模的汇编指令种类，基本块内的语句在14条以内的情况，上述算法都是安全的。在实际补丁分析中，绝大部分情况都是满足这个条件的。

将相应函数的差异信息图形式化显示在函数控制流程图中，有利于直观地分析，便于漏洞定位。

3.3 编译优化处理

在补丁分析过程中，编译器造成的影响是不容忽视的。如果二进制软件版本A和B版本在生成的时候，采用的编译器优化选项不同，那么它们之间存在许多差异，都是由编译器造成的，而不是由补丁对漏洞的修补造成的。这些差异信息并非补丁分析技术所感兴趣的，属于是误报，必须予以剔除。

无论是在函数配对的过程中，还是在基本块判同的算法中，不同编译器优化选项都将对补丁分析造成许多干扰。图3和图4是在分析微软ms05-015补丁中遇到的一对典型函数。它们在形式上有很大的区别，但实质上却是完全相同的。在图3中，每个出口基本块都有自己独立的retn语句，而图4中采取了一个优化措施，它提取了所有的retn语句，形成一个公共的出口基本块，使得函数只有一个出口，这是近年来编译器优化的一个趋势。这导致了图3的签名因素是(6,8,0)，而图4的签名因素是(7,10,0)。对于这种情况，在根据签名对函数进行配对之前，必须加以处理，否则无法将这一对本应该配上的函数正确地配对。

对于一种可能会影响签名因素的编译优化策略φ，可以通过以下两种方式来进行处理：一是把编译优化策略φ对控制流图产生的影响等价地作用于版本A和版本B中函数的控制流图，将经过处理后的新的控制流图记作$f(A)$和$f(B)$。

如果生成目标码 A 时没有采用优化 φ，而 B 采用了，那么 f(A)就等同于在 A 的编译器中加上 φ 之后形成的函数流程图。这样，版本 A 和 B 比较的基准就一致了，等同于编译器采用了相同的优化策略。第二种方式同第一种对称，可以将编译优化策略 φ 对版本 A 和 B 中函数流程图的影响都去掉。总之，目标就是使得一种策略 φ，要么都影响了补丁前后函数的控制流图，要么都不影响。例如，针对图 3 和图 4 的例子，可以采用将图 3 中所有的 retn 语句都提出，单独成块，或者将图 4 中的 retn 语句复制到它所有的父亲节点中。针对其他类型的编译优化选项，还可能采用指令翻译、基本块合并等方法。

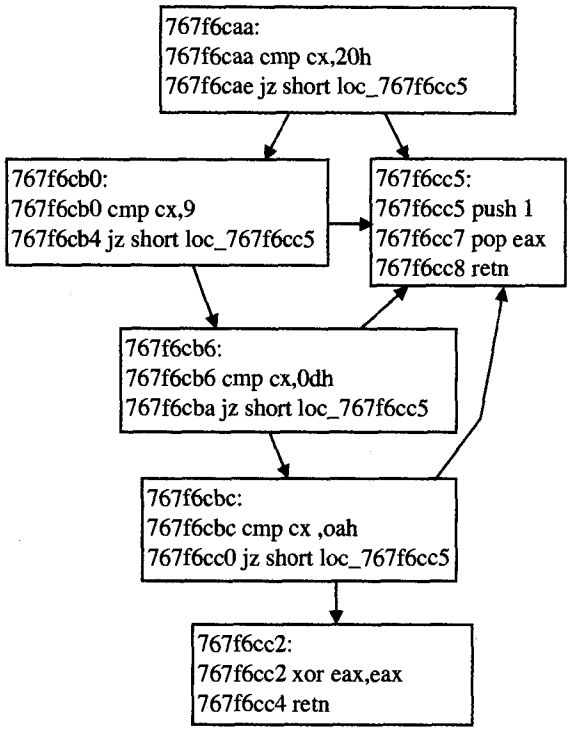

图 3　分析微软 ms05-015 补丁时碰到的一个函数流程图

基本块判同的算法，同样受到编译器优化选项的很大影响。相同的源代码经过编译器生成后可能具有不同的二进制形式，仍然以图 3、图 4 为例子来说明：从传参方式来说，图 3 采用了寄存器直接传参的方法，而图 4 中通过栈传递参数到寄存器，这样导致两者的入口基本块形式不同；从寄存器赋值方式来说，图 3 在编号为 767f6cc5 的基本块中为寄存器 eax 赋 1 值，采用 push 1，pop eax 的组合。而图 4 在标号为 767f60ab 的基本块中则采用 xor eax，eax，inc eax，不同的语句组合实现了完全相同的功能。除此之外，还存在指令重排序的问题，这种编译器的影响在图 3、图 4 中的例子没有体现，但在补丁分析中是非常普遍的。3.2.2 节中小素数乘积算法就是针对指令重排序优化的一种处理方法。以上列举的几种情况都是汇编语句形式不同而语义相同的例子，为了避免差异信息的误报，在已匹配函数差异判别阶段应该对这些编译器造成的区别加以处理，将它们认同，因为这些形式上的差别并非补丁分析感兴趣的目标。

编译优化验证技术是编译科学中的一个分支，它的目的是对经过优化的代码与原代码的等价性进行判断。此种技术用在补丁分析中，将编译器导致的补丁间的差异与真正的代码间的语义差别区分开来，可以有效减少补丁分析的误报率。编译优化验证技术的方法可以查阅文[6～9]。

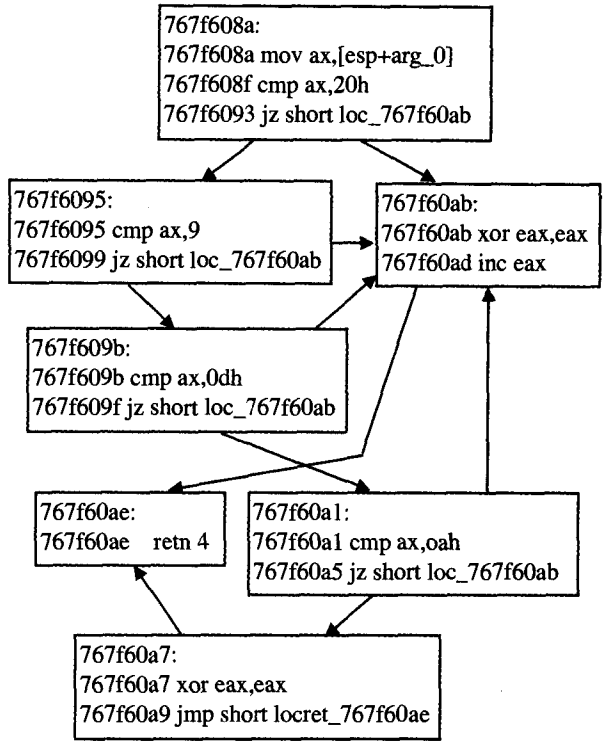

图 4　分析微软 ms05-015 补丁时碰到的与图 3 配对的一个函数流程图

4　系统的实现

系统的实现是基于反汇编技术的。比较好的一个反汇编平台可以采用 IDA Pro，这是 DataRescue 公司出品的一款功能强大的专业级反汇编工具。IDA(The Interactive Disassembler)的全名叫做交互式反汇编工具。理论上讲，能够支持语言特性的完美的反编译需要强大的人工智能的支持才能够实现，但是 IDA 凭借其交互的特性，在人为干预的前提下，也能够实现相当强的功能。IDA 在分析二进制文件时，也会发生错误，通常表现为部分函数无法识别。人工专家可以通过交互特性来对这种错误进行改正。

通过编写 IDA 插件，可以得到 2.1 节中提到的二进制文件调用关系和函数控制流程图。然后根据上文描述的方法，就可以进行补丁分析了。在已配对函数差异判别的过程中，将函数流程图中相同的基本块染为深色，并且编上相应的号码。这种将差异信息图形化的显示，有利于人工理解分析结果，快速定位漏洞。

5　一个具体的补丁分析例子

微软在 ms05-025 中报告了 Internet Explorer 由于处理 PNG 图像的方式存在 bug 而导致一个远程执行代码漏洞。攻击者可以通过构建恶意的 PNG 图像来利用此漏洞，如果用户访问了恶意网站或查看了恶意的电子邮件，其用户系统就可能被完全控制。

根据此漏洞报告提供的简单信息，首先获得新旧版本的 Pngfilt.dll，通过反汇编工具分析二进制文件，得出新旧版本的函数个数均为 142。由于新旧版本的 Pngfilt.dll 均存在符号表，每个函数都能通过反汇编技术获得其名字，所以全部配对。为了验证签名算法的有效性，不再利用函数名称，而是生

成所有函数的签名信息，并进行编译优化的处理，利用 3.1.2 节中的签名算法进行配对，函数配对 128 对。无法配对的主要原因是多个函数拥有相同的签名，经过简单的人工分析，可以很容易地将剩下的函数配上。

配对之后，比较对应的函数，经过差异判定，定位到 3 个函数。函数 WriteScanLine 和 ProcessTRNS 同属类 CPNGFilter，另外一个 IID_Iregist 函数，经过人工分析，发现其差异是由编译器造成的。

图 5 和图 6 是函数 WriteScanLine 打补丁前后的函数流程图。由于函数较大，主要截取了差异的部分。图 7 和图 8 是函数 ProcessTRNS 打补丁前后的函数流程图。在两对函数流程图中，每个深色的基本块都有一个与其编号相同的深色的基本块，这些基本块都是对应相同的。

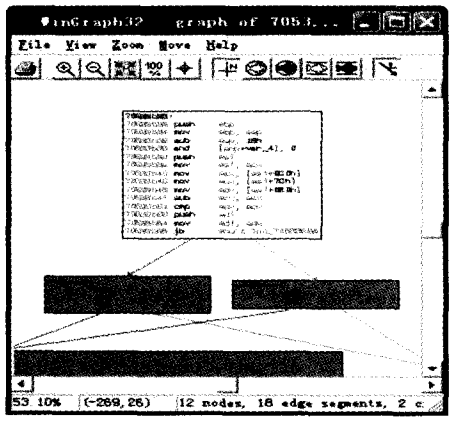

图 5 函数 WriteScanLine 打补丁前的函数流程图

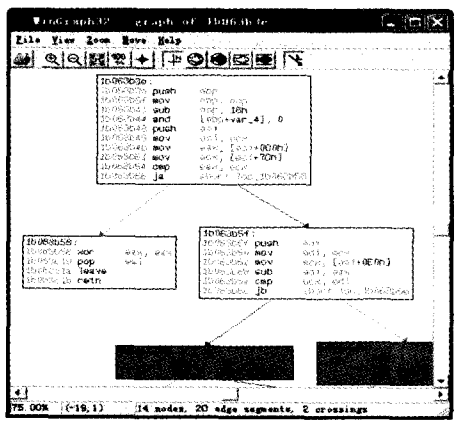

图 6 函数 WriteScanLine 打补丁后的函数流程图

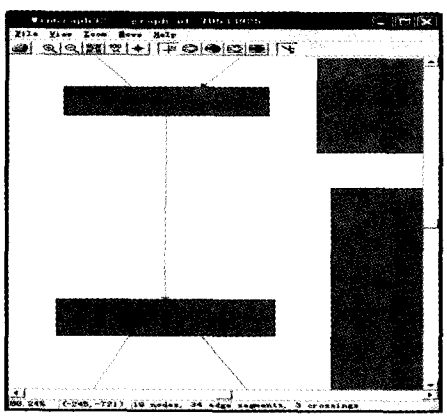

图 7 函数 ProcessTRNS 打补丁前的函数流程图

很容易看出补丁前后的 Pngfilt.dll 的差异，新版本中增加了对两个变量值的合法性判断。经过跟踪分析，确定旧版本中缺少对一个无符号参数的大小判断，可能导致整数溢出。旧版本中还缺少对一个循环控制变量的合法性检查。这两个漏洞可能被恶意的 PNG 图像触发，从而导致远程代码执行。

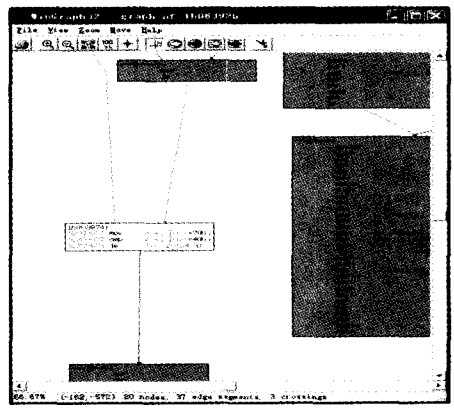

图 8 函数 ProcessTRNS 打补丁后的函数流程图

总结 二进制补丁分析是一件复杂的工作，涉及了反汇编、图的同构、编译优化验证等多方面的问题。本文提出了一种基于反汇编技术进行安全补丁分析的方法，并且在实际的工作中对此方法进行了实现，以此为基础分析了多个补丁。实际的补丁分析工作证明基于此方法的系统可以快速、准确地定位补丁前后软件的差异，从而定位漏洞。

在多个实际补丁分析的过程中，总结出的重要经验是在补丁分析的过程中必须结合人工专家的经验和智慧。在分析的每一步，人工专家都可以对中间结果进行调整，进行人机交互，并根据实际的经验进一步完善算法。另外，编译器对补丁分析造成的影响仍然造成了相当数量的误报，尤其是在配对函数差异判定部分。对这方面性能的改进是下一步工作的重点。

参 考 文 献

1 Sabin T. Comparing binaries with graph isomorphisms. http://razor.bindview.com/publish/papers/comparing-binaries.html. 2004

2 Dullien T, Rolles R. Graph-based comparison of executable objects. http://www.sabre-security.com/files/BinDiffSSTIC05.pdf. 2005

3 Aho A V, Sethi R, Ullman J D. Compilers Principles, Techniques, and Tools. 北京：机械工业出版社，2003

4 DataRescue. IDA Pro disassembler. http://www.datarescue.com/idabase. 2005

5 Hoqlund G, Mcgraw G. Exploiting Software : How to Break Code. Addison Wesley, 2004

6 Currie D W, Hu A J, Rajan S. Automatic formal verification of DSP software. In: Proceedings of the 37th Annual ACM IEEE Conference on Design Automation(DAC'00), ACM Press, 2000. 130～135

7 Feng X, Hu A J. Automatic formal verification for scheduled VLIW code. In: Proceedings of the JointConference on Languages, Compilers and Tools for Em-bedded Systems & Software and Compilers for Embed-ded Systems (LCTES/SCOPES'02), ACMPress, 2002. 85～92

8 Necula G C. Translation validation for an optimizing compiler. In: Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation (PLDI'00), ACM Press, June 2000. 83～94

9 Pnueli A, Siegel M, Singerman E. Translation validation. In: Proceedings of the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'98). vol 1384, Lecture Notes in Computer Science. Springer-Verlag Heidelberg, Mar 1998. 151～166