doi :10.3969/j.issn.1671-1122.2009.05.013

ClearBug 一种改进的自动化漏洞分析工具

刘波，文伟平，孙惠平，卿斯汉

（北京大学软件与微电子学院信息安全系，北京 102600）

摘　要：随着软件漏洞的危害性不断增强，软件漏洞分析已经成为了国内外安全研究的热点。已有的工作大致可以分为静态分析和动态分析两类。本文在开源的软件漏洞静态分析工具BugScam的基础上，提出了一种建立漏洞模型，映射漏洞模型为分析程序，并进行漏洞分析的思路。对于大量的软件漏洞，我们提出，将其分为函数漏洞和逻辑漏洞两类，并分别探讨了两种模型与程序之间的对应关系。最后，对我们编写的一个改进的自动化漏洞分析工具ClearBug进行了介绍，并用实验验证了模型与程序的正确性和有效性。

关键词：漏洞模型；自动化分析；ClearBug

中图分类号：TP31　　文献标识码：A

ClearBug An improved automatic tool for bug analysis

LIU Bo, WEN Wei-ping, SUN Hui-ping, QING Si-han

(Department of Information Security, SSM, Peking University, Beijing 102600, *China)*

Abstract: With the increasing harmfulness of software vulnerability, identifying potential vulnerabilities in software has become the focus of security research. The current analysis method can be roughly divided into two categories: static analysis and dynamic analysis. This paper presents an idea based on open source static analysis tool BugScam. First, set up vulnerability model. Then, map the model to program and begin vulnerability analysis. We classified vulnerability model to function model and logic model and research the corresponding relationship between model and program. Finally, we give an introduction of our improved automatic vulnerability analysis tool ClearBug. The experiment results show that our tool can effectively find out some software vulnerability.

Key words: Software Vulnerability Model; Automatic Analysis; ClearBug

0 引言

当前，大部分的恶意攻击或入侵都基于软件安全漏洞，一方面，安全研究者提出了一些对计算机系统安全进行评估的标准和模型，如 TCSEC [1]、SSECMM [2]，来使编制的软件更加的安全。但是，已有研究指出，即使是最严格的流程也会产生低质量的软件 [3]；因此，另一方面，研究人员也对安全漏洞进行了大量研究。早在 70 年代中期，美国南加州大学就开始了 PA 研究计划对操作系统的安全漏洞进行研究，之后，伊利诺伊大学、普渡大学 COAST 实验室也都进行了计算机漏洞方面的研究工作。漏洞的分析与收集工作也逐渐由个人行为转为组织的形式。在国内，绿盟科技、启明星辰是安全漏洞挖掘方面的代表。从 2000 年到 2007 年，绿盟科技安全小组发布了大量分析到的安全漏洞 [4]。然而，现有的研究都是使用一些传统的漏洞挖掘技术，针对具体漏洞，主要依靠安全研究者的经验进行判断，其中包含了大量的重复工作，研究效率受到了限制。如果能设计和使用自动化或半自动化的辅助工具，快速、高效和准确的挖掘软件中的安全漏洞，无论对于提高软件的安全性还是为国家安全提供保障都是意义重大的。

2003 年，Cheers Halvar 发布了一个开源的二进制漏洞分析工具 BugScam，该工具基于 IDA Pro 的反汇编结果进行汇编级的漏洞分析。但是 BugScam 只能查找一些很简单的缓冲区溢出漏洞。就像 Cheers 本人所说，该工具仅仅是一个开始 [5][6]。

本文在 BugScam 的基础上，提出了一种基于二进制代码进行漏洞分析的思路。首先建立漏洞模型，然后根据漏洞模型编写相应的模式匹配程序，最后对代码进行漏洞分析。

本文的贡献包括：(1) 提出了一种建立漏洞模型、编写模式匹配程序，然后对二进制代码进行漏洞分析的思路。(2) 首次提出了函数和逻辑两种漏洞模型，并分别介绍了其建模的方式。(3) 编写了 BugScam 的改进程序 ClearBug，该工具扩充了原有的漏洞分析函数，并能进行批量化的漏洞分析。

1 相关工作

当前的漏洞分析方法主要可以分为静态分析和动态分析两种 [7][8]。

其中静态分析包括：有向图分析、污点数据传播分析、IDC 脚本分析和整数约束分析；有向图分析是指构造函数调用关系图，在图中的各个节点记录子函数的起始位置、函数所分配的堆栈大小、函数局部变量的使用情况等信息 [9]。生成有向图后，可以对一些容易产生缓冲区溢出漏洞的函数，如 strcpy，根据其在调用前是否进行了参数边界检测，来判断是否可能产生漏洞。另外这种分析技术也可以用于补丁比较来发现已被软件厂商发现但未公开的漏洞 [10]。污点数据传播分析通常是建立双向数据流，首先是不可信数据的传播流，然后是 strcpy、strcat 等可疑函数的调用流，通过分析两条数据流的交点，并判断是否进行了污点数据的长度检查，来判断是否可能存在漏洞。IDC 脚本分析 [11] 也就是本文所采用的

分析方式，通过对IDA Pro反汇编后的结果进行分析，判断是否存在漏洞。D. Wagner等人在2000年提出了一种将缓冲区溢出的判断问题规约为整数范围分析的方法[12][13]，该方法基于C源代码，依据数据缓冲区的分配范围和实际数据的长度范围建立约束条件，然后利用约束条件对代码进行漏洞分析。

动态分析包括：黑盒测试[14][15][16]和虚拟堆栈分析[17]。黑盒测试是对程序功能是否按照需求正常运行，是否能接收输入数据而产生正确的输出信息进行检查。虚拟堆栈分析是对可疑函数的调用进行拦截，并同时创建该函数调用的虚拟堆栈，然后对堆栈内的信息进行分析。

2 漏洞分析模型

本文将漏洞模型分为函数模型和逻辑模型两种。其中函数模型是指针对特定的函数进行漏洞特征建模，然后在待分析文件中对该函数的调用点进行检查。这种模型的优点是易于建模，缺点是适用性不强，对于大量不使用该特征函数的文件无法分析。逻辑模型本质上包含函数模型，这里我们把它单独列出，特指对漏洞程序的逻辑结构进行特征提取，然后利用这种特征进行匹配分析。这种模型适用性更强，但是建模的准确性和粒度需要深入的研究。

2.1 函数模型

BugScam采用的全部都是函数模型，它总共包含8个函数(_strcpy、_strcat、_sprintf、_lstrcpyA、_lstrcatA、wsprintfA、sprintf、MultiByteToWideChar)，通过对这些函数进行漏洞特征的提取，然后进行函数特征匹配来分析文件中可能出现漏洞的位置。

当前的ClearBug版本在BugScam的基础上，添加了对strcpy、strcat和recv三个函数的特征模型。下面以recv函数为例，探讨函数模型的建立方法。

目前，ClearBug中的函数模型都是对缓冲区溢出漏洞进行分析，而缓冲区溢出的实质是一个缓冲区大小问题，因此对这个漏洞的检测可以规约为一个大小比对问题。recv函数的函数原型为int recv(SOCKET s, char* buf, int len, int flags)，其中buf为接收数据用缓冲区，len为可接收数据的长度，那么针对该函数的缓冲区溢出漏洞会在len大于buf的长度时发生。因此检测模型就可以定义为检查len的长度是否大于buf的长度：

Fuction Model of recv:
If len > SizeOf(buf):
Warning(It is likely to be a buffer overrun. This location should be investigated manually!)
Else:
Pass

上面给出的只是一个最基本的模型，为了提高检查的精度，可以对该模型进行进一步的细化，如判断在recv函数调用前是否进行了长度检查，来减少误报。

总的来说，函数模型的建立是相对比较简单的，但针对不同的分析目标，如源代码或者二进制代码，其具体程序的实现方法还是有很大不同的。另外，在检查的准确度和模型的复杂度之间也需要进行权衡。

2.2 逻辑模型

除了函数模型，更通用的模型是逻辑模型。由于很多漏洞的产生是由于程序编制人员在程序逻辑上没有考虑周全所导致的，其中的一些逻辑漏洞是具有共性的，如果我们对这些漏洞进行特征建模，就可以找到程序中可能会发生该漏洞的位置。

文献[18]提出了一种可用于脚本编程的逻辑关系，该逻辑关系其实就是对应了一个逻辑模型。当进行函数调用时，如果在getenv( )函数之后，strcpy( )、sprintf( )等函数之前，没有使用strlen( )函数进行长度检查，就认为该程序可能是带有漏洞的程序：

Logic Model One:
If (exists getenv()) and(exists strcpy() or sprintf() behind getenv()):
If not exits strlen() between them:
Warning(It is likely to be a logic bug. This location should be investigated manually!)
Else:
Pass

这个模型找到的漏洞实际上还是函数漏洞，但是其提出的分析方法却是逻辑分析。

下面我们再看一个与函数关系不大的逻辑模型，以图1中的程序为例[19]，程序的原意是当IsAccessAllowed( )函数返回ERROR_ACCESS_DENIED时，表明该用户没有访问权限，因此访问被拒绝。可是如果恶意用户通过构造输入等方式，使得IsAccessAllowed( )函数出错，其返回值比如说是ERROR，那么由于if / else默认允许访问这一逻辑上的漏洞，就可能发生非法用户获得访问权限的结果。该程序正确的逻辑应该是如图2所示，只有当IsAccessAllowed( )返回NO_ERROR时才赋予用户访问权限，其它情况下，访问都会被拒绝。

对于上面的程序，如果if语句中使用的是= =，而默认情况是执行if / else的第二个分支，并且该分支是授权分支，或者相反，if语句中使用的是!=，而默认情况下是执行if / else的第一个分支，而且该分支是授权分支，那么该程序就可能是带有漏洞的程序：

Logic Model Two:
If the judge statement contain" ==" and the authorized branch is the else branch
or the judge statement contain" !=" and the authorized branch is the first branch:
Warning(It is likely to be a logic bug. This location should be investigated manually!)
Else:
Pass

```
DWORD dwRet = IsAccessAllowed(...);
if (dwRet == ERROR_ACCESS_DENIED) {
    // security check failed.
    // Inform user that access is denied
} else {
    // security check OK.
    // Perform task.
}
```

图1 带安全隐患的程序段

```
DWORD dwRet = IsAccessAllowed(...);
if (dwRet == NO_ERROR) {
    // security check OK.
    // Perform task.
} else {
    // security check failed.
    // Inform user that access is denied.
}
```

图2 改进后的程序段

3 ClearBug 程序分析

这一节主要介绍 ClearBug 程序，该程序总体上分为两个部分：控制程序和分析程序。控制程序用 MFC 写成，其功能是调用 IDA 和 IDC 脚本，实现分析的自动化和批量化。分析程序用 IDA 提供的 IDC 脚本语言写成，在 IDA 反汇编的基础上，进行汇编级的漏洞分析。

就整体构架来说，控制程序和分析程序是基本独立的，因此 ClearBug 具有很强的扩展性。

3.1 控制程序

控制程序的界面如图 3 所示，由于 ClearBug 实际上实现的是对一个文件夹下的所有子文件夹和文件进行批量分析，因此采用了递归模式。首先用 FindFirstFile( ) 函数找到第一个文件或目录，如果是文件则进行分析，如果是目录则进行递归调用。分析完该文件或者目录后，通过 FindNextFile( ) 函数找到其余的文件或目录，并循环持续下去，直到分析完待分析文件夹下所有的子文件夹和文件。

对于单个文件的分析是通过 IDA 的命令行接口，通过命令行的方式调用 IDA 以及我们的 ClearBug 脚本进行漏洞分析，并生成漏洞分析报告。

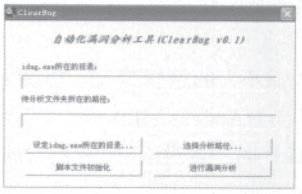


图3 控制程序界面

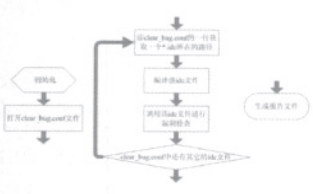


图4 分析程序框架

3.2 分析程序

在 IDA 反汇编的基础上，IDC 脚本会进行具体的漏洞分析。不同的漏洞模型（函数模型和逻辑模型）会对应不同的分析脚本，比如说 strcpy 函数就对应了一个针对该函数模型进行分析的脚本文件 strcpy.idc。

整个 ClearBug 脚本的框架基本和 BugScam 一致。如图 4 所示，首先是进行一些数组变量的初始化，该数组中的元素主要用于统计和保存分析的结果。然后会打开一个 clear_bug.conf 文件，该文件中保存了大量的路径，这些路径指向前面提到的针对不同的漏洞模型编写的各个分析脚本。通过循环，每打开一个 *.idc 文件，就会调用 IDA 提供的 compile( ) 函数进行脚本文件的编译，然后调用该脚本文件内的 audit( ) 函数进行漏洞检查，由于对每个漏洞模型，我们都定义了一个 audit( ) 函数，因此在主程序中对该函数的调用是通用的。该函数执行完以后，也就是对一个漏洞模型分析完以后，会判断循环是否结束，如果还有 *.idc 文件没有调用，就继续刚才的步骤，否则生成漏洞报告文件。漏洞报告文件是 html 格式的文档，如图 5 所示。

如何将第 2 节中提出的模型映射到程序的，这里我们仍然以 recv 函数模型为例，分析其具体实现时 audit( ) 函数的内容。在该函数中，首先是根据函数名（recv），调用 LocByName( ) 函数获得 recv 函数自身所在的地址。这里要注意的是，在有些 IDA 版本中，recv 函数对应的函数名为 __imp__recv@16。接着，在整个待分析代码段中，查找对该函数地址的引用位置，对每一个引用位置，如果为函数调用，即为 call 指令，则在该处调用 Audit_recv 函数进行漏洞分析。

在 2.2 节中，提出了针对 int recv(SOCKET s, char* buf, int len, int flags)的漏洞模型，Audit_recv 函数就是对该模型的一个具体实现。

图5 生成的漏洞报告

图6 对Winamp2.10的攻击

4 实验结果

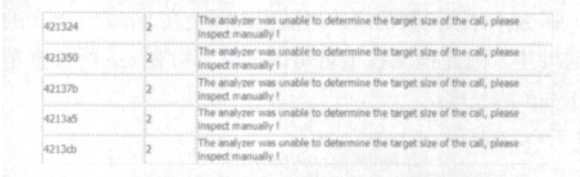
通过实验，对 ClearBug 程序的正确性和有效性进行验证。实验分析了三个带有漏洞的程序：一个 Winamp 2.10 的缓冲区溢出漏洞，一个简单的 strcpy 函数溢出漏洞以及一个 recv 函数的溢出漏洞。其中每个实验都会分别使用 BugScam 和 ClearBug 进行分析，并比较二者的不同。

4.1 Winamp 2.10缓冲区溢出漏洞

在 Winamp 2.10 版本中，支持播放列表，该列表实际上就是一些歌曲存放的路径。当加载一个列表时，正常情况下是列出一系列的待播放歌曲，但是如果该列表的长度过大，就会发生溢出。我们针对该漏洞编写了 shellcode，并成功的进行了攻击，如图 6 所示。

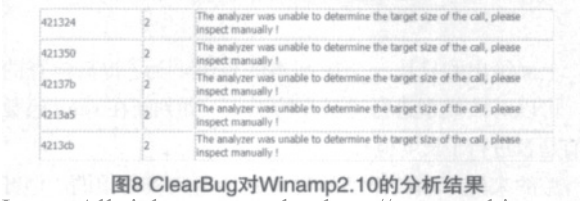
接下来，使用 BugScam 和 ClearBug 进行漏洞分析，其分析结果如图 7 和图 8。在这个漏洞中，实际发生溢出的是 wsprintfA( ) 函数，可以看到，两个程序都找到了发生溢出的函数在 IDA 反汇编结果中的地址。

通过这个实验可以得出结论，第 2 节提出的漏洞建模方式能够用于实际漏洞的分析。



421324	2	The analyzer was unable to determine the target size of the call, please inspect manually !
421350	2	The analyzer was unable to determine the target size of the call, please inspect manually !
42137b	2	The analyzer was unable to determine the target size of the call, please inspect manually !
4213a5	2	The analyzer was unable to determine the target size of the call, please inspect manually !
4213cb	2	The analyzer was unable to determine the target size of the call, please inspect manually !

图7 BugScam对Winamp2.10的分析结果



421324	2	The analyzer was unable to determine the target size of the call, please inspect manually !
421350	2	The analyzer was unable to determine the target size of the call, please inspect manually !
42137b	2	The analyzer was unable to determine the target size of the call, please inspect manually !
4213a5	2	The analyzer was unable to determine the target size of the call, please inspect manually !
4213cb	2	The analyzer was unable to determine the target size of the call, please inspect manually !

图8 ClearBug对Winamp2.10的分析结果

4.2 一个简单的strcpy函数缓冲区溢出漏洞

strcpy函数是典型的容易发生缓冲区溢出漏洞的函数，其函数原型为char *strcpy(char *strDestination, const char *strSource)，我们设定strDestination的大小为128，strSource的大小为384，则会发生溢出，如图9所示，对该漏洞编写相应的shellcode，可以成功的进行攻击，获得一个shell，如图10所示。

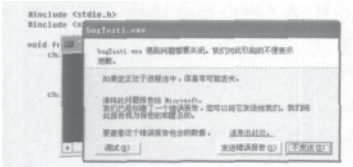


图9 strcpy缓冲区溢出漏洞

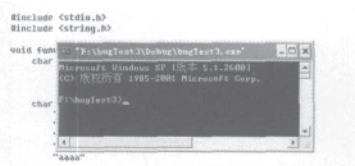


图10 对stropy漏洞的攻击

分别用BugScam和ClearBug进行漏洞分析，分析结果如图11和12所示。可以看出由于BugScam没有对strcpy函数进行建模分析，所以没有找到漏洞点，而ClearBug则成功的找到了漏洞点，而且给出了漏洞的一个描述。在该描述中成功的给出了strDestination和strSource的大小。

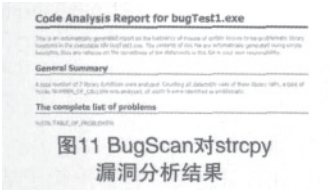


图11 BugScan对strcpy漏洞分析结果

图12 ClearBug对strcpy漏洞的分析结果

4.3 recv函数缓冲区溢出漏洞

在这个实验中，分别编写了一个客户端和服务端程序，在服务端程序中使用recv函数接收消息，recv函数的函数原型为int recv(SOCKET s, char* buf, int len, int flags)，我们设定len的大小为4096，而buf的实际大小为10，这样当服务端接收到一个过长的消息时，就会发生溢出。在客户端编写了攻击用shellcode，在和服务端通信的过程中，成功的发生了溢出并获得了服务端机器的控制权，如图13所示。

同样的，我们分别用BugScam和ClearBug进行漏洞分析，分析的结果如图14和15所示。可以看到我们的程序同样成功的找到了漏洞点。

图13 对recv函数缓冲区溢出漏洞攻击

图14 BugScan对recv函数漏洞的分析结果

图15 ClearBug对recv函数漏洞的分析结果

5 结论和未来工作

本文在开源软件BugScam的基础上，提出了一种建立漏洞模型，编制漏洞分析程序，然后对二进制文件进行漏洞分析的思路。在文中，我们将漏洞模型分为函数漏洞模型和逻辑漏洞模型两种，对于不同的模型，我们分别介绍了其与程序的映射关系。在此基础上，我们编写了一个改进的自动化漏洞分析工具ClearBug，并对该工具的控制部分和分析部分进行了介绍。最后，通过实验，我们对三个Winamp2.10、strcpy和recv函数的漏洞进行了检测，并成功的发现了其中漏洞点，验证了模型和程序的有效性。

接下来，我们的研究工作包括以下几个方面的内容：

大量分析已有的安全漏洞，建立漏洞模型库，并用于ClearBug的IDC脚本库中；研究如何改进当前的ClearBug程序架构，减少对IDA Pro的依赖性；考虑将IDC脚本分析与其它的漏洞分析方法相结合，得到更强的漏洞分析工具。

（责编 潘静）

参考文献：
[1] TCSEC. Trusted Computer System Evaluation Criteria. Department Of Defense. 1985.
[2] SSECMM. Systems security engineering capability maturity model. Software Engineering Institute. 1995.
[3] J.M. Voas and G. McGraw. Software Fault Injection: Inoculating Programs Against Errors. John Wiley and Sons, New York, 1998.
[4] 绿盟科技安全小组：http://www.nsfocus.net.
[5] BugScam IDC Package. http://sourceforge.net/projects/bugscam
[6] Kkqq, bugscam Analysis. Xfocus.2004.
[7] Watercloud, 利用IDA PRO挖掘和分析安全漏洞. Xfocus XCon.2003.
[8] Yichen Xie, Andy Chou, and Dawson Engler. ARCHER: Using Symbolic, Pathsensitive Analysis to Detect Memory Access Errors. ESEC/FSE'03, Helsinki, Finland. September 1-5, 2003.
[9] Funnywei, 缓冲区溢出漏洞发掘模型. Xfocus XCon. 2003.
[10] T.Sabin, Comparing Binaries With Graph Isomorphisms, http://razor.bindview.com/publish/papers/comparing-Binaries.html, 2004.
[11] 曹军，Windows危急级漏洞挖掘及分析技术研究. 四川大学硕士学位论文. 2006.
[12] D. Wagner, J. Foster, E. Brewer, and A. Aiken. A first step towards automated detection of buffer overrun vulnerabilities. In The 2000 Network and Distributed Systems Security Conference. San Diego, CA, February 2000.
[13] 杨小龙，刘坚. C/C++源程序缓冲区溢出漏洞的静态检测. 计算机工程与应用，2004；40（20）：108-110.
[14] Glenford J.Myers, The Art of Software Testing, Second Edition. John Wiley & Sons, inc, Hoboken, New Jersey2004.
[15] B.P. Miller, L. Fredricksen, B. So, "An empirical study of the reliability of Unix utilities," CACM, vol.33 no.12, Dec. 1990, 32-44.
[16] B.P. Miller, D. Koski, C.P. Lee, V. Maganty, R. Murphy, A. Natarajan, J. Steidl, Fuzz revisited: a re-examination of the reliability of Unix utilities and services, Tech. report CSTR-95-1268, U. Wisconsin, Apr. 1995.
[17] Anup K.Ghosh, Tom O'Connor, and Gary McGraw. An automated approach for identifying potential vulnerabilities in software. In Proceedings of the 1998 IEEE Symposium on Security and Privacy, pages 104—114, Oakland, CA, May 3-6 1998.
[18] 徐良华，孙玉龙，高丰，朱鲁华. 基于逆向工程的软件漏洞挖掘技术，微计算机信息，2006年24期：259-261.
[19] Michael Howard, David LeBlanc著，程永敬等译，编写安全的代码（第2版），机械工业出版社，2005

作者简介：刘波（1982—），男，硕士研究生，主要研究方向：网络安全；文伟平（1976—），男，副教授，主要研究方向：网络攻击与防范、恶意代码研究、信息系统逆向工程和可信计算技术等；孙惠平（1975—），男，讲师，博士，主要研究方向：身份和信任管理、访问控制、RFID安全与隐私保护等；卿斯汉（1939—），男，教授，博士生导师，主要研究方向：密码学、操作系统安全与安全集成、信息安全算法与协议等。