

软件安全漏洞挖掘的研究思路及发展趋势

文伟平^{1,2}, 吴兴丽¹, 蒋建春³

(1. 北京大学 软件与微电子学院信息安全系, 北京 102600;

2. 北京大学 软件工程研究所网络与信息安全研究室, 北京 100871;

3. 中国科学院软件研究所, 北京 100190)

摘要: 软件安全漏洞发掘作为一项预先发现软件潜在安全漏洞来保证软件安全的重要技术, 日益受到人们的重视。本文首先对软件安全漏洞发掘研究的背景及相关技术进行了充分调研, 然后针对当前进行软件安全漏洞挖掘提出新的研究思路, 从漏洞模型、补丁比对、序列搜索算法等四个方面进行了详细描述。

关键词: 软件安全; 漏洞挖掘; 补丁比对

中图分类号: TP309.5 **文献标识码:** A

0 引言

早在上世纪 70 年代中期, 美国南加州大学开始了 PA (Protection Analysis Project) 研究计划, 主要针对操作系统的安全漏洞进行研究, 以增强计算机操作系统的安全性。后续有 RISOS 计划、SDC 的渗透分析、Brian Marick 的软件漏洞分析、Landwher 的漏洞分类、普渡大学 COAST 实验室的计算机漏洞研究^[1]。国外著名的安全团队如 Eeye、LSD、W00W00 等^[2]组织对最新的漏洞进行及时跟踪分析, 并给出相应的漏洞解决方案。

然而, 到目前为止对于安全漏洞的定义、分类、特征及挖掘方法仍未形成统一和公认的结论。国内外安全漏洞研究组织公布的都是一些传统的漏洞挖掘研究技术。针对具体漏洞, 安全研究者往往进行大量的重复工作, 研究效率和效果上也有相当的局限性。因此, 如何设计和使用自动化或者半自动化的辅助工具, 快速、高效和准确挖掘软件中的安全漏洞成为最近相关研究的热点和难点。

1 软件安全漏洞挖掘相关技术调研

1.1 补丁比对技术

基于补丁比对的漏洞分析技术较早得到了应用, 其理论模型的创建始于 2004 年 2 月, 由 Halvar Flake 第一次提出了结构化比对的^[3]补丁比对算法, 同年 4 月 Tobb Sabin 提出图形化比对算法作为补充^[4], 之后发展比较迅速。2008 年 1 月 David Brumley 等人在 IEEE 上发表论文^[5], 肯定了补丁比对技术在现实环境中的应用价值。在工业界, 基于补丁比对的漏洞分析软件得到了迅速发展。国外方面, 在 2005 年 12 月 iDefense 发布了 IDACCompare; 2006 年 11 月 eEye 发布了 eEye Binary Diffing Suite (EBDS)^[6]; 2007 年 09 月 Sabre 发布了 Bindiff2。这些工具的出现和改进, 使得补丁比对技术在漏洞分析过程中得到了越来越广泛的应用。在国内, 开展补丁比对理论和技术研究的相关单位逐步增多。具有代表性的包括: 2006 年 10 月, 解放军信息工程大学信息工程学院的罗谦、舒辉等人, 在串行结构化比对算法基础上, 提出了一种基于全局地址空间编程

模型实现的并行结构化比对算法; 2007 年 12 月, 国家计算机网络入侵防范中心 (NCNIPC) 完成了 NIPC Binary Differ (NBD) 补丁比对工具。

补丁比对提高了定位二进制文件安全漏洞的效率。目前常用的二进制补丁比对方法主要分为三类:

(1) 基于文本的比对。基于文本的比对是最为简单的一种补丁比对方式, 通过对两个二进制文件 (补丁前和补丁后) 进行对比, 对文件比对中出现的任何一点差异, 都不做处理地写入结果之中。这种方法的后果是最后输出的结果范围很大, 容易出现极多的误报情况, 漏洞定位精度极差且结果不容易被漏洞分析人员理解, 因此仅适用于文件中产生变化较少的情况。

(2) 基于汇编指令的比对。基于汇编指令的二进制文件比对是先对二进制文件进行反汇编, 然后将两个反汇编之后的文件进行对比, 具有代表性的工具如 eEye 发布的 eEye Binary Diffing Suite (EBDS) 软件工具中的 Binary Diffing Starter。这种方式虽然较直接的二进制文本比对要进步, 比对结果更容易被分析人员理解, 但是仍然存在输出结果范围大, 误报情况多和漏洞定位不精确的缺点。更重要的是基于汇编指令的补丁比对方法很容易受编译器编译优化的影响, 结果会变得非常复杂。

(3) 基于结构化的比对。基于结构化比对的方法是 Halvar Flake 在 2004 年提出的, 这种方法的基本思想是: 给定两个待比对的文件 A1 和 A2, 将 A1 和 A2 的所有函数用控制流图来表示, 通过比对两个图是否同构来建立函数之间一对一的映射。该方法从逻辑结构的层次上对补丁文件进行了分析, 但当待比对的两个二进制文件较大时, 由于提取签名信息、进行结构化比对的运算量和存储量非常巨大, 程序的执行效率非常低。D. Brumley 等人在此基础上, 提出了基于程序控制流程图 (CFG) 的

约束规约分析方法,一定程度上提高了漏洞定位精度。总之,目前基于结构化的补丁比对在执行效率和漏洞定位的精确性方面还存在很大的发展空间。

1.2 Fuzzing测试技术

Fuzzing 测试是通过提供非正常的输入并监测系统异常发现软件安全漏洞的自动化方法。Fuzzing 测试的概念最早在1989年由Wisconsin大学的Barton Miller教授提出,用于测试UNIX系统应用程序的健壮性。1999年Oulu大学开始研发基于Fuzzing的测试工具软件PROTOS,并于2002年发布了用于SNMP测试的版本。2002年美国黑客大会上,Dave Aitel演示了名为SPIKE的Fuzzing测试工具,这是一种流传很广的开源工具。2004年Michal Zalewski发布了针对浏览器进行Fuzzing测试的工具Mangleme。2005年Mu Security公司发布了商业的Fuzzing测试工具。专用于文件格式Fuzzing测试的工具产生于2004年,当时微软发布了MS04-028漏洞公告,这是第一次发布文件格式漏洞,此后文件格式漏洞的发掘和利用一直受到广泛的关注。出现了FileFuzz、FFuzzer、UFuz3、Fuzzer等一系列专用于文件格式的Fuzzing测试工具。

在Windows操作系统下,出现了大量基于文件格式的软件安全漏洞,影响较大的有Microsoft Windows GDI+JPG解析组件缓冲区溢出漏洞(MS04-028)、Microsoft Word 6.0/95文件转换远程缓冲区溢出漏洞(MS04-041)、Microsoft Windows动画光标文件处理远程缓冲区溢出漏洞(MS05-002)、Microsoft MSN Messenger和Windows Media Player PNG图片解析远程代码执行漏洞(MS05-009)、Microsoft Excel畸形字符串远程代码执行漏洞(MS07-015)、Microsoft Excel BIFF记录远程栈溢出漏洞(MS07-023)、Microsoft Visio文档封装远程代码执行漏洞(MS07-030)、Microsoft Publisher任意指针引用

远程代码执行漏洞(MS07-037)和Microsoft GDI+ WMF图形处理缓冲区溢出漏洞(MS08-052)等,上述漏洞的相关情况可以参考微软网站每月第一个周二发布的漏洞公告。

虽然Fuzzing测试工具已经发现了大量的安全漏洞,但是Fuzzing测试的缺点也是比较显著的。在Fuzzing测试中要想完成完整的分支覆盖、状态测试,必须构造庞大的测试用例,而测试用例的数量是以指数级增长,所以Fuzzing测试中要想实现完整测试不大可能。这种情况下需要对测试用例进行精简,而在不了解系统内部实现的前提下进行测试用例的精简,容易造成大量的测试用例分布在某些特定的分支和状态上,而一些特殊的分支或者状态得不到覆盖。考虑到上述缺陷,针对接口进行的Fuzzing测试有一定程度的盲目性,不能针对同一接口或者协议实现的不同软件进行有针对性的测试。

1.3 自动化的静态分析

静态分析根据软件类型分为两类:针对开源软件的静态分析和针对非开源软件的静态分析。针对开源软件的静态分析,使用编译技术在代码扫描或者编译期间确定相关的判断信息,然后根据这些信息对特定的漏洞模型进行检查。而针对非开源软件的静态分析,主要是基于反汇编平台IDA Pro,使用自下而上的分析方法,对二进制文件中的库函数调用,循环操作等做检查,其侧重点主要在于静态的数据流回溯和对软件的逆向工程。

通过扫描源码检查安全漏洞的主要工具有FLAWFINDER、ITS4、RATS、SPLINT。其中FLAWFINDER、ITS4、RATS使用词法分析检查安全漏洞,检查的原理是如果发现对某些函数的调用与指定的方法不同,就给出警告,如果进一步发现问题,就报告一个错误。但是这些工具难以确定被检查的函数调用是否真的是一个安全漏洞,而仅仅

是报告一个潜在的安全错误。相比之下SPLINT通过代码解析提供了更为详细的函数参数值确定的方法,提供了发现安全漏洞更为精确的依据。

针对二进制文件进行安全漏洞检查比较典型的工具有Bugscam。基本的工作原理和扫描源码的工具相同。但是由于不能对二进制文件进行词法分析和代码解析,所以核心的问题在于如何确定参数类型和获得参数的值。由于缺乏比较系统的理论基础,目前针对二进制文件的检查工具还只是基于人为经验,在检测模型和算法上的研究并不完善。

除了针对不安全库函数和错误使用安全库函数的检查之外,静态分析还在动态内存分配方面作出了不错的工作。通过代码扫描或者通过在源码编译期间获得的信息对内存分配设计一个安全模型,然后在这个安全模型中进行逻辑检查,从而发现安全漏洞。但是考虑到现代软件的复杂性,内存管理并不单纯的使用Malloc、New、HeapAlloc函数进行,而使用了更为复杂的例如内存池等技术,这些模型并不是很完善,在一些复杂问题的判断上并不实用。

虽然静态分析由于缺乏运行时数据,缺乏动态的测试过程以及细粒度的安全评估,并不能确定所有的安全漏洞,而且在检测漏洞的方面并不是很准确和完善。但是静态分析的优势在于容易集成于开发过程中,例如编译器中,使软件在开发过程中就可以发现一些潜在的安全隐患。微软的编译器中已经集成了相关的功能,是一个很典型的应用。

2 软件漏洞挖掘的主要研究思路

2.1 软件安全漏洞挖掘模型及流程研究

漏洞挖掘过程主要分为两大部分:漏洞代码的粗定位和精确定位。粗定位的目标是发现被分析程序中所有可能存在安全漏洞的代码位置(简称漏洞点)。精确定位是在粗定位的基础上,利用动

