一种基于 fuzzing 技术的漏洞发掘新思路*

邵 林, 张小松, 苏恩标
(电子科技大学 计算机科学与工程学院, 成都 610054)

摘 要: 目前检测软件缓冲区溢出漏洞仅局限于手工分析、二进制补丁比较及 fuzzing 技术等, 这些技术要么对人工分析依赖程度高, 要么盲目性太大, 致使漏洞发掘效率极为低下。结合 fuzzing 技术、数据流动态分析技术以及异常自动分析技术等, 提出一种新的缓冲区溢出漏洞发掘思路。新思路克服了已有缓冲区溢出漏洞发掘技术的缺点, 能有效发掘网络服务器软件中潜在的未知安全漏洞(0day), 提高了缓冲区溢出漏洞发掘效率和自动化程度。

关键词: 自动化; 缓冲区溢出; 黑盒测试; 安全漏洞; 漏洞发掘
中图分类号: TP319 **文献标志码:** A **文章编号:** 1001-3695(2009)03-1086-03

New method of software vulnerability detection based on fuzzing

SHAO Lin, ZHANG Xiao-song, SU En-biao
(School of Computer Application Technology, Technology University of Electronic Science & Technology of China, Chengdu 610054, China)

Abstract: The techniques of buffer overflow vulnerabilities detection was single and limited to manual analysis, binary-patch comparison, fuzzing and so on. These techniques of vulnerabilities detection were either too dependent on manual analysis or too blind, leading up to the low efficiency of vulnerabilities detection. Introduced a new method of buffer overflow vulnerabilities detection, which was based on fuzzing, data-flow dynamic analysis and automated exception analysis. Overcame the disadvantages of old techniques, this new method effectively improves the detection of potential unknown security vulnerabilities (0day) in software. Besides, this method is more automated and performs better in finding new security vulnerabilities.

Key words: automation; buffer overflow; fuzzing; security vulnerability; vulnerability detection

自从 A. One[1] 在著名黑客杂志《Phrack》上发表了开创性文章之后, 缓冲区溢出漏洞利用技术被网络黑客广泛采用。虽然后来兴起的入侵检测系统(IDS)和入侵保护系统(IPS)能根据攻击数据包指纹对黑客攻击进行拦截, 但这种防御措施极为被动, 只有在相关漏洞利用程序公布后才能拦截这些缓冲区溢出攻击。因此, 利用软件的未知安全漏洞进行网络攻击一直都是网络攻防的重点。为了确保计算机网络的安全, 安全人员应尽可能地发掘软件潜在安全漏洞并及时修补, 这样才能有效防止利用软件安全漏洞的攻击行为。

目前, 软件安全漏洞发掘技术大致分为基于源代码的漏洞发掘技术和基于可执行文件的漏洞发掘技术。然而, 现在大多数软件都是不开源的, 如各种商业软件和操作系统软件 Windows、MacOS 等。因此, 针对可执行文件的漏洞发掘技术显得尤为重要。本文将提出一种在协议层面上, 利用 fuzzing 技术对第三方服务器软件进行缓冲区溢出漏洞发掘的新思路。

1 传统的漏洞发掘技术

1.1 Fuzzing 技术(黑盒测试)

Fuzzing 技术是一种高度自动化的测试技术, 它以大量半有效(semi-valid)的数据作为程序输入, 从而发掘程序内部潜在的缺陷或错误。P. Oehlert[2] 对 fuzzing 技术以及如何构造 fuzzing 工具(fuzzer)进行了详细说明; Miller 等人[3,4] 首先将 fuzzing 技术应用于软件的可靠性测试, 在 UNIX 系统上利用 fuzzing 技术对 UNIX 系统及该系统上的工具(utilities)进行了可靠性测试; 在此基础之上, Miller 等人[5] 进一步将 fuzzing 技术应用到 Windows 平台上以及 MacOS 上进行了类似的测试。迄今为止, 国外学者已对 fuzzing 技术进行了大量研究, 并取得了很多成果。芬兰 Oulu 大学研究并实现了针对网络协议实现软件进行漏洞发掘的 PROTOS 系统; 此后, 美国加州大学 Sant Babara 分校又提出了针对状态协议的漏洞发掘软件[6], beSTORM、SPIKE 等都是目前为止较为成功的商业漏洞发掘软件, 已成功发掘出多个软件安全漏洞。

目前国内安全领域在 fuzzing 技术方面的研究并不多见, 相关文献也较少。文献[7]中介绍了如何利用 fuzzing 技术发掘 MP3 播放软件中存在的安全漏洞; 文献[8]则介绍了利用 fuzzing 技术发掘 TFTP 服务器软件中安全漏洞的方法。

1.2 源码审计(白盒测试)

源码审计是通过对程序源代码进行检测, 以发现代码中可能的错误。源码审计主要针对 C/C++ 代码, 而且所能检测出的漏洞局限于缓冲区溢出漏洞和格式化字符串漏洞。文献[7]对当前已有的知名源代码审计工具进行了详细的分析, 并指出当前的源代码审计工具应用范围较窄, 而且大多数商业软件都是不开源的。目前, 源码审计技术多与开发平台相结合或由编译器实现, 如 Visual Studio 6.0、Visual Studio 7.0 以及 DDK 编译器等。

1.3 IDA 反汇编审计(灰盒测试)

该技术与源码审计较为相似, 所不同的是该技术是针对可

收稿日期: 2008-06-08; 修回日期: 2008-08-14 基金项目: 国家自然科学基金资助项目(60673142)

作者简介: 邵林(1985-), 男, 四川自贡人, 硕士, 主要研究方向为软件安全技术(headium2006@gmail.com); 张小松(1968-), 男, 副教授, 博士, 主要研究方向为网络安全; 苏恩标(1985-), 男, 硕士, 主要研究方向为网络安全.

执行文件进行漏洞发掘。很多时候软件的源代码无法获取,但通过类似 IDA 这样的强大反汇编平台,可以实现基于可执行文件的漏洞发掘。由于汇编代码可读性差,且软件的反汇编代码数量极为庞大,该技术对于漏洞发掘人员的要求极高。

1.4　二进制补丁比较

一旦发现软件存在安全漏洞,软件厂商就会通过打补丁的方式对软件进行修复。这样,通过对比打补丁前后的可执行文件(或反汇编码)就能知道漏洞的具体细节。然而,二进制补丁比较技术只能用于发掘已经公布的安全漏洞,这种事后分析方法的现实意义并不大。

1.5　逆向分析安全产品

在 2007 年世界黑客大会(DefCon)上,G. Robert 等人[9]提出了通过逆向分析安全产品以获取未知安全漏洞的方法。众所周知,在世界范围内存在着一个巨大的地下漏洞交易市场,通过高额价格不仅可以买到安全漏洞描述,甚至还可以买到漏洞利用程序。因此,一些商业厂家就通过地下交易方式购买未知漏洞利用程序,提取特征代码加入到杀毒软件病毒库或 IDS 的指纹库中。可见,通过逆向分析安全产品有可能获得未知安全漏洞。虽然该技术与前面的二进制补丁比较技术相似,但是该技术所发掘到的漏洞很可能是未知安全漏洞,因此该技术应用价值较大。

2　Internet 文本消息协议

目前,网络应用层协议大致可以分为基于文本消息的协议和基于数据块的协议。顾名思义,基于文本消息的协议就是使用文本消息来描述协议命令以及协议命令参数,这类协议的典型包括 Telnet、FTP、POP3、SMTP 以及 IMAP 等;而基于数据块的协议则包括了 SMB 协议(即 CIFS 协议)和 RPC 等,该类协议的特点是协议命令和参数往往都是以数据块中某个域的数值来表示的。与文本消息相比,基于数据块的协议功能更强大,同时也更为复杂。

在互联网发展初期,计算机及网络硬件条件较差,文本消息协议得到广泛应用,尤其是 Telnet、FTP、POP3 以及 SMTP 等协议。这些协议作为 Internet 发展的遗留物一直保存了下来,像 FTP、SMTP 以及 POP3 等协议实现软件仍旧被广泛使用。

近年来,随着操作系统级别安全漏洞的不断公布以及操作系统的不断完善,系统级安全漏洞发掘难度日益增加。网络黑客开始转向网络协议实现软件,通过这些软件,网络黑客也能实现对目标系统的入侵。而这些软件则主要集中于 FTP、HTTP 以及 SMTP 等服务器软件。因此,针对第三方网络协议实现软件进行漏洞发掘是很有必要的,能够有效防范网络入侵。

如上所述,Internet 文本消息协议采用文本来描述协议命令以及命令参数,因此更为简单。为了便于描述漏洞发掘模型以及实现模型原型,本文将针对基于 Internet 文本消息的协议进行说明和开发原型系统。

3　在协议层面上实现的 fuzzing 技术

从前面对各种漏洞发掘技术的分析来看,只有 fuzzing 技术具有较高的自动化程度,其他漏洞发掘技术则过多地依赖于漏洞发掘人员的经验。然而 fuzzing 技术也存在着一定的缺陷,如漏洞发掘时间开销大,存在测试过程中的运行时开销等。针对 fuzzing 技术所存在的优缺点,本文提出了在协议层面上使用 fuzzing 技术来实现对服务器软件漏洞发掘的思路。具体而言,该思路要求在详细分析协议文档后,按照特定的数据结构来描述整个测试过程,然后测试控制模块根据该结构实施测试。在整个测试过程中,需要对被测试的服务器进行监控,一旦服务器内部出现异常,才能够立即捕获并分析异常。由于整个测试过程中,测试模块记录下了测试数据的长度等信息,在异常分析时能较为精确地定位异常信息。基于该思路实现的漏洞发掘系统原型框架如图 1 所示。

3.1　文本消息协议的描述

为了便于说明,本文将以 FTP 为例来详细说明对文本消息协议的描述。FTP 中的所有命令、命令参数以及分隔符都是特定的字符或字符串。而且这些命令及分隔符又可以分为一定的类型。这样就可以通过几张表格将这些字符或字符串关联起来,从而很好地描述协议命令。用于描述协议命令、参数及分隔符的各表格如图 2 所示。

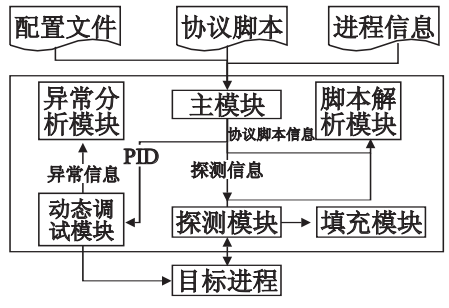

图1　软件漏洞发掘系统框架图

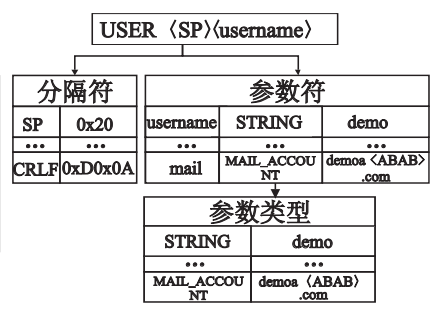

图2　协议描述文件

3.2　异型数据包的生成

异型数据包是指满足协议语法规则但又是无效的测试数据。这些数据能够有效地覆盖程序执行路径,从而尽可能地触发程序内部的漏洞。构造异型数据包的常用方法如下:

a) 针对缓冲区溢出漏洞而言,异型数据包的构造很简单,只需填充超长字符串即可。但是,为了便于异常分析模块分析,在填充时往往都是重复填充相同字符串,如“ABAB”“\A\A”或“\...\”等。

b) 对于格式化字符串漏洞探测而言,填充字符串往往是“%s%d”“%n%d”以及诸如此类的字符串。这些用于测试格式化字符串漏洞的字符串可以通过参数类型配置文件来设置。

c) 整数溢出也是软件安全漏洞中常见的一类软件漏洞。在测试软件整数溢出漏洞时,所采用的方法往往是填充整数的边界值,如 -1、0、1、0xff、0xffff、0xffffffff 等。

上述三种测试数据构造方法是 fuzzing 技术实现时最常采用的方法,根据测试目的的不同,还有其他的异型数据包构造方法。此外,针对不同的漏洞测试,fuzzer(基于 fuzzing 技术实现的漏洞发掘工具)所采用的测试数据构造方法也有所不同。比如本文所实现的原型系统所针对的是缓冲区溢出漏洞,因此系统没有必要构造用于发掘整数溢出漏洞的异型数据包。

3.3　Fuzzing 过程的控制

在漏洞发掘过程中,各个测试命令的顺序是极为重要的。根据笔者对已经报告的软件安全漏洞的分析,大多数安全漏洞的触发都需要特定的上下文环境,即只有在发送特定协议命令序列时,软件中的安全漏洞才可能被触发。Golden FTP Server (<=2.7)版本中,NLST 及 CWD 命令漏洞都必须在登录 FTP 服务器之后才可能触发。由此可见,在整个测试过程中,协议命令的测试顺序(即异型数据包的发送顺序)是极为重要的。

系统采用图 3 所示的有向图来描述测试过程中各个协议命令的发送顺序。从起始节点到终端节点(可以包括多个终

端节点) 的每一条路径都对应于一个测试序列。这样, 每一个测试序列就对应于一个特定的协议上下文。根据该有向图, 原型系统将依次构建并发送测试数据包。在原型系统中, 该有向图采用 XML 语言进行描述, 然后由脚本解析模块进行解析并转换为程序内部约定好的结构。

3.4 异常的捕获及分析

当前有不少的 fuzzer 都只是简单地提供测试数据, 对于异常的分析几乎不涉及。利用这些 fuzzer 进行漏洞发掘时, 漏洞发掘人员往往需要第三方调试器配合才能有效地发掘软件漏洞。笔者在设计原型系统时考虑到了该问题, 因此原型系统采用 Windows 操作系统自身提供的调试 API(如 DebugActiveProcess 等) 对服务器进程进行全程监控, 一旦发现服务器进程内部异常, 系统将暂停发送测试数据包并立即保存与异常相关的信息。此后, 系统尝试将处于崩溃状态的进程恢复到正常状态。异常恢复是漏洞发掘自动化的一个关键所在, 但本文的重点不在于此, 在此将不对该问题进行过多说明。

在捕获并保存服务器进程内部的异常之后, 原型系统将对异常信息进行自动分析, 从而智能地给出异常分析报告。原型系统所实现的异常自动分析算法是笔者在分析已有的缓冲区溢出漏洞之后设计出来的算法, 该算法能够有效地分析出导致异常的命令、异常类型以及异常风险程度等。根据该算法生成的异常分析报告在一定程度上能减少漏洞分析人员的后期分析工作, 缩短漏洞分析时间, 从而提高漏洞发掘的效率。

4 实验与结果

4.1 对弱点软件进行的漏洞发掘

利用原型系统, 笔者对 ServU FTP Server(< 5.0) 、Golden FTP(≤ 2.7) 、WSFTP Server(5.05) 以及 Titan FTP Server (6.04.0545) 等多款 FTP 服务器软件进行了安全性测试。在对 Titan FTP Server(6.04.0545) 进行测试时, 笔者还发现了两个未知安全漏洞。图 4 是原型系统对 XXX 服务器软件进行测试时, 发现服务器内部异常时的截图。

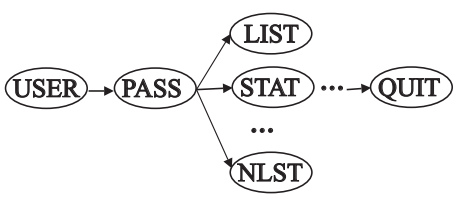

图3 描述协议测试顺序的有向图

图4 原型系统截图

4.2 实验结果

原型系统在设计之初就充分考虑到系统的重用性和可扩展性。具体而言, 原型系统的使用者只需对脚本文件进行适当的修改就可以构造出满足特定要求的数据包。正是由于该原因, 原型系统几乎可以发掘已经公布的所有缓冲区溢出漏洞。本文利用原型系统对已经公布的 ServU、Golden FTP Server 以及 WS FTP Server 等服务器漏洞都进行了验证性测试, 除了 Golden FTP Server(2.7) 及其后续版本中的漏洞无法发掘之外, 其他的漏洞本文的原型系统都能发掘。表 1 是原型系统对 ServU、Golden FTP Server 以及 WS FTP Server 等 FTP 服务器软件进行漏洞发掘的结果。

表 1 原型系统测试结果

名称(版本)	漏洞类型	探测字符串	探测长度
ServU(< 5.0)	栈溢出	SITE CHMOD [ABAB] / test. txt	256 B
Golden FTP(< = 2.7)	同上	NLST [/A /A]	5 836 B
Golden FTP(< = 2.7)	同上	CWD [//A:]	1 400 B
WSFTP(< = 5.05)	同上	XCRC [ABAB] XMD5 [ABAB] XSHA1 [ABAB]	1 KB
Titan FTP(6.04.0545)	堆溢出	USER [ABAB] ACCT [ABAB]	64 KB

表 1 所给出的漏洞中, Titan FTP 服务器漏洞都是原型系统所发掘出的。此外, Titan FTP 服务器中 USER 命令所导致的堆溢出漏洞, 在黑客网站 http: //www. milw0rm. com 上已经公布了相关利用代码。而 ACCT 命令所导致的漏洞仍旧未公布, 因此该漏洞仍旧是一个 0day 漏洞。

4.3 原型系统所不能处理的软件

为了提高漏洞发掘效率, 本文的原型系统采用调试器来捕获测试过程的异常。然而一些商业软件在发布版本中加入了反调试功能, 这些具有反调试功能的软件在运行过程中一直在检测是否有调试器附着到自身, 一旦发现有调试器附着到其上, 软件对应的进程就抛出异常并自动中止运行。对于这些软件而言, 本文的原型系统无法进行漏洞发掘工作, 因为原型系统会将程序运行时抛出的异常认为是“可疑漏洞”。此外, 一旦该类程序抛出异常, 服务器进程就中止运行。

5 结束语

目前, 原型系统的自动化程度和智能化程度与本文的预期目标仍有差距。在后续的研究工作中, 笔者将从以下几方面进一步提高系统的自动化和智能化程度: a) 在测试输入数据构造方面, 准备借鉴人工智能中的遗传变异和启发式算法来生成测试数据, 从而提高漏洞发掘效率和自动化程度; b) 采用内存断点技术对弱函数调用进行跟踪, 从而捕获服务器从接收数据到系统崩溃过程中的函数调用图; c) 在现有的异常自动分析算法上进行改进, 提高异常分析的智能性和正确性。

参考文献:

[1] NE A. Smashing the stack for fun and profit[J] . Phrack Magazine, 1996, 7(49) 14-16.

[2] OEHLERT P. Violating assumptions with fuzzing[J] . IEEE Security & Privacy, 2005, 3(2) : 58-62.

[3] MILLER B P, FREDRIKSEN L, SO B. An empirical study of the reliability of UNIX utilities[J] . Communications of the ACM, 1990, 33(12) : 32-44.

[4] MILLER B P, KOSKI D, LEE C P, *et al*. Fuzz revisited: a reexamination of the reliability of UNIX utilities and services[R] . Madison: University of Wisconsin-Madison, 1995.

[5] MILLER B P, MILLER C, FREDRICK M. An empirical study of the robustness of MacOS applications using random testing[C] //Proc of the 1st International Workshop on Random Testing. 2006: 46-54.

[6] BANKS G, COVA M, FELMETSGER V, *et al*. SNOOZE: toward a stateful Network protocol fuzzer[C] //Proc of the 9th International Conference on Information Security. 2006: 343-358.

[7] 魏瑜豪, 张玉清. 基于 fuzzing 的 MP3 播放软件漏洞发掘技术[J] . 计算机工程, 2007, 33(24) : 158-167.

[8] 刘旭奇, 张玉清. 基于 fuzzing 的 TFTP 漏洞挖掘技术[J] . 计算机工程, 2007, 33(20) : 142-147.

[9] GRAHAM R, MAYNOR D. A simpler way of finding 0day[R/OL] . (2007) . http: //www. black hat. com/presentations/bhusa_07_Maynor and Graham/whitepaper/bh_usa_07_mayor and_graham_wp. pdf.

[10] JON H, PASCAL M. Can source code auditing software identify common vulnerabilities and be used to evaluate software security[C] //Proc of the 37th Annual Hawaii International Conference on System Sciences. 2004: 4405-4414.