摘 要

传统的漏洞挖掘技术并未体现出应有的价值和潜力，针对具体漏洞的挖掘，安全研究者所进行的工作中存在大量的重复内容，同时在研究效率、效果上也存在相当的缺陷。基于模糊的漏洞挖掘技术可以很好的解决传统漏洞挖掘技术的问题。

采用了先进的协议描述语言对网络协议格式进行描述，解决了协议格式分析中动态计算字符串长度的问题，使得产生的测试数据都是有效的。建立了字符串替换库，可以产生大量的测试数据，达到动态产生测试数据的目的。技术方案中的基于参数权值衡量技术可以对模糊数据进行动态的跟踪，进而得到它们可能会导致溢出的概率。基于以上的统计，发送带有权值的测试数据到服务器端，可以减少测试时间，降低测试数据数量，提高测试效率。

在 linux 平台上实现了相应的漏洞挖掘器。漏洞挖掘器主要由产生脚本模块、调试跟踪模块和产生并发送测试数据模块等构成。产生脚本模块可以实现协议的自动化解析和本自动的测试脚本的撰写，可以减少测试人员撰写脚本的时间；调试跟踪器和发送数据端相连接起到对参数进行评估的作用，可以实现参数的自动选择；产生并发送测试数据模块是实验的核心部分，可以用来解析脚本，产生并发送测试数据等，具有一定的先进性和实用性。利用挖掘器对基于网络协议的软件进行测试，达到了预期的效果。

关键词：漏洞挖掘，模糊，网络协议，信息安全

Abstract

The traditional discoverying Vulnerability technology should not reflect the value and potential capacity. Researchers usually do a large number of duplicate works when they are trying to discovery a specific vulnerability ,and what's more,they will have some faults in the effectiveness of research and considerable shortcomings. Discovering the ulnerability of Protocol Using Fuzzing Method can resove the problem that exsist in the traditional technology.

First ,an advanced description language to describe the format of network protocols was adopted, which can resolve the length problems which was the difficulty of the analyse protocol and can make the test data valid. Then a library which can create a great deal of testing data was estabilished to the purposes of producing dynamic test data. Finally, Based on the programme of technical parameters of the right technology can measure the value of fuzzy data dynamic track, then they may be led to the probability of spills. Based on the above statistics, sending the test data to the server can reduce testing time, reduce test data volume and improve the efficiency of testing.

At last a software based on the above method was realized on the Linux platform which was consist of three components:PD2AD,Debugger and autoFuzzer.PD2AD can automatically resolve the protocol and generate the test scripts; Debugger connects to the Fuzzer and tracking markers which were used in the dangerous functions of the tested programme,and then mark values on the markers depending on the useness of the dangerous functions.Then the markers with heaveier weights ware fuzzed first by Fuzzer.

Key words: Vulnerability Discovery, Fuzzing, Protocols, Information security

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　月　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密□ ，在_____年解密后适用本授权书。

本论文属于

不保密□。

(请在以上方框内打“√”)

学位论文作者签名：　　指导教师签名：

日期：　　年　月　日　　日期：　　年　月　日

1 绪论

随着攻击事件的不断上升，病毒、蠕虫和其它恶意程序在 Internet 上大肆泛滥，信息安全面临诸多挑战，逐渐成为人们眼中的热点和焦点。信息安全的一个核心问题就是计算机系统中存在着软件安全漏洞，恶意的攻击者可以利用这些安全漏洞访问未经授权的资源，破坏敏感数据，进而威胁信息系统的安全。基于模糊(Fuzzing) [1] 的漏洞挖掘技术作为一种高效的黑盒子测试技术，逐渐受到信息安全工作者的重视。

1.1 课题的研究背景

覆盖全球的因特网，以其自身协议的开放性方便了各种计算机网络的入网互联，这极大地提高了人们的工作效率和生活的便利性。但是，由于早期网络协议对安全问题的忽视，以及在使用和管理上的无序性，使得网络安全受到严重威胁，安全事故频频发生。近几年针对 Windows 系统的蠕虫层出不穷，造成的十分严重的影响，其中典型代表是 2006-7 年爆发的“熊猫烧香”病毒。2006 年 12 月下旬至 2007 年 1 月初，“熊猫烧香”病毒在全国范围内大肆传播。据国家计算机病毒应急处理中心以及瑞星、江民、金山等防毒厂商技统计数据显示，截止 2007 年 2 月 2 日，共监测发现“熊猫烧香”病毒变种 700 余个，被感染计算机达 445 万台。导致巨大的经济损失 [2]。类似的蠕虫攻击事件几乎每天都会发生，只是造成的影响程度有轻重之分。

病毒蠕虫盛行的主要原因在于软件漏洞被黑客所利用。根据中国 CERT 的统计，从 2000 年到 2007 年，每年都新增大量的漏洞报告。2003 年漏洞报告总数为 3784，2004 年为 3780，2005 年为 5990，2006 年为 26476，而 2007 年在则在 2006 年的基础上上涨了 2 倍。在国内，绿盟科技维护着国内最大、最全面的安全漏洞库，几乎每天都更新着最近的 8000 多条漏洞信息。黑客往往在极短时间内就能成功利用这些漏洞，当人们意识到应该增强安全意识的时候，由这些漏洞产生的蠕虫及

其变种早已泛滥。蠕虫传播的形式也从原来的主动感染，扩展到兼有网页、电子邮件的多种被动感染形式。

相比于黑客，安全研究者们在漏洞研究工作的影响方面显得被动和滞后。很多情况下，我们都通过拦截并分析特定的蠕虫及其变种，进而了解黑客的采用漏洞利用技术。为此，国家高技术发展计划(863)的信息安全技术主题主要研究信息安全核心、关键和共性技术，以形成自主的信息安全防护能力、隐患发现能力、应急反映能力以及信息对抗能力，为建立国家信息安全保障体系提供技术支撑。作为隐患发现技术之一的漏洞研究技术，理应受到重视。而漏洞挖掘技术作为漏洞研究技术的重要组成部分，近年来得到了长足的发展。

但是传统的漏洞挖掘技术存在一定的缺陷和不足，导致漏洞挖掘研究效率低下，基于 Fuzzing 的漏洞挖掘技术作为动态挖掘方法的一种，近几年来在漏洞挖掘领域得到了长足的发展，同时也具有很高的研究效率。本论文的选题就是在该类漏洞研究技术的基础上提出的。

1.2 国内外研究概况

20 世纪 70 年代中期，美国启动的 PA (Protection Analysis Project) 和 RISOS(Research in Secured Operating Systems)计划被公认为是计算机安全研究工作的起点。1980 年，美国密执安大学的 B.Hebbard 小组使用“渗透分析” (Penetration Analysis) [3] 方法成功地发现了系统程序中的部分漏洞。1990 年，美国伊利诺斯大学的 Marick 发表了关于软件漏洞的调查报告 [4] ，对软件漏洞的形成特点做了统计分析。1993 年，美国海军研究实验室的 Landwher 等人收集了不同操作系统的安全缺陷，按照漏洞的来源、形成时间和分布位置建立了 3 种分类模型。普渡大学 COAST 实验室的 Aslam 和 Krsul 在前人成果的基础上，提出了更为完整的漏洞分类模型，并建立了专用漏洞数据库。MITRE 公司从事的"公共漏洞列表" (Common Vulnerability Enumeration,CVE)工作，为每个漏洞建立了统一标识，方便了漏洞研究的信息共享及数据交换。这些研究都代表了当时的最高成果，但到目前为止对于安全漏洞的定义、分类、特征仍未形成统一和公认的结论。

传统的漏洞挖掘技术分为静态分析和动态分析两大类[5]。其中，静态分析技术主要包括:有向图分析[6]、污点数据传播分析、IDC 脚本分析、整数限制分析和补丁比较[7]分析;动态分析技术主要包括:格式分析、黑盒测试[8]和虚拟堆栈分析[9]等。Fuzzing 技术作为黑盒子测试技术，是一种高效的漏洞挖掘技术，有很高的研究价值。Fuzzing 通过发送随机数据到目标程序用以找出漏洞。Fuzzing 不只是进行简单的字符串替换，而是利用一些高效的工具来发掘软件漏洞。利用 Fuzzing 已经发现了许多软件的漏洞[1,10-12]。本文正是基于国外对 Fuzzing 技术的研究热度和 Fuzzing 技术本身具□的研究价值而写。

目前利用自动化方法发现漏洞基本上可以归为两类，一类是基于软件测试中的黑盒测试原理，这类方法主要针对软件提供的接口，而不是针对软件的实现进行测试。测试引擎通过发送大量的测试数据对软件进行测试，如果检测到被测试软件有异常情况(例如软件崩溃、产生异常等)，就可以使用一些判断条件来确定某个测试数据是否可以触发软件漏洞。这一类的典型方法的代表技术有 SPIKE[13]，SPIKE 在其不断的开发过程中，发现了微软 Windows 操作系统，以及微软 SQL Server 中多个远程缓冲区溢出漏洞。

另外一类是使用源码审核技术，对软件的源代码进行扫描，针对不安全的库函数使用以及内存操作进行语义上的检查，从而发现安全漏洞。其中的典型有静态检查技术[14-16]，但是这些检查方法只能局限于库函数使用错误，不能识别动态内存错误。文献[16]中提出了使用静态方法检查动态错误的简单模型，但是检测效果不是很明显，难以进行实用化的操作。

微软的 Visual C++.NET 编译器[17]中使用了相关的方法进行安全性检查以尽可能地减少软件中的安全隐患。除了针对开源软件源代码扫描的方法之外，bugscam[18]和 bugaudit[19]在反编译平台 IDA Pro[20]基础上，针对编译后的二进制可执行文件进行静态检查，在针对非开源软件方向上做出了有益的尝试。同样地，针对二进制可执行文件的检查也存在着缺乏运行时数据带来的一系列问题。

诚然，上述自动化方法确实很有成效地发现了一些安全漏洞，但同时这些检测方法还存在着许多不足。黑盒测试相对比较盲目，难以达到比较理想的分支覆盖测

试，同时测试的结果依赖于测试数据，而测试数据的构造需要对被测试软件使用的协议、接口信息有很深入的了解。基于编译技术的方法由于缺乏软件运行时的数据，难以确定软件在实际运行时的数据以及执行状态，造成无法进行漏洞判断，或者判断的失误率过高。

基于上述考虑，一些安全分析人员对运行时检查安全漏洞的方法进行了探讨。一些运行时的检测方法[21]被提出，但是其中描述的技术过于分散，没有形成系统的模型和工具，而且没有对如何在运行时匹配漏洞模型给出一个完整的构想。

1.3 课题研究的意义

对软件安全漏洞确切、统一的定义有利于刻画漏洞的根本特征，有助于规范对漏洞本质的描述，而对软件安全漏洞的统一描述有助于各研究团体之间漏洞研究成果和学术交流，有利于漏洞研究的不断深化和发展[22]。从某种程度上讲，对软件安全漏洞挖掘技术的研究扩展了系统安全研究的广度和深度。

根据科学研究的一般规律，建立科学合理的软件漏洞挖掘模型和挖掘流程，能够增强对软件安全漏洞本质的进一步了解，有助于防止程序设计人员在编写程序时产生安全漏洞；可以使软件管理和使用人员了解其系统中可能存在的漏洞隐患，从而有针对性地消除或阻止安全漏洞的存在；可以帮助安全分析人员更具针对性地寻找、分析、发现未知的漏洞，达到防范于未然的目的。

此外，漏洞挖掘的自动化程度是漏洞挖掘技术发展的重要方向。摆脱繁复的人工干预和判读，自动化或半自动化的漏洞辅助挖掘工具的实现将会在很大程度上提高漏洞挖掘的效率。

然而，从现有的资料来看，大多组织都是都是侧重于漏洞扫描等工作的研究，就漏洞挖掘技术而言还显得不够。针对具体的漏洞，安全研究者进行的研究工作往往存在大量的重复内容，在研究效率和效果上具有相当大的局限性。因此，高效、实用的漏洞挖掘技术将是本文研究的重点。

1.4 研究内容

研究高效、实用的漏洞挖掘技术是本文的主要内容。基于论文思路的考虑，本文共分为六章,主要内容分为以下几点。

第一章是引言部分，介绍了漏洞挖掘的研究背景和研究现状。

第二章介绍了本文所用到的基本理论，以及传统的漏洞挖掘技术原理。

第三章是文章的核心部分，重点介绍了基于 Fuzzing 的网络协议漏洞挖掘技术，并提出了基于 Fuzzing 的漏洞挖掘器的设计方案。

第四章在上一章的基础上，实现了漏洞挖掘器，并对软件的关键技术进行了进一步的分析。

第五章实验仿真部分，利用挖掘器进行实际的漏洞挖掘工作，并对结果进行分析，对比得出漏洞挖掘器的优点和不足。

第六章对本文进行了总结，并对未来做了展望。

2 相关理论及技术

随着网络化和信息化的快速发展，信息系统的安全问题日益严重。网络攻击、入侵等安全事件频繁发生，而这些事件多数是由于系统或应用软件存在安全漏洞造成的。对安全漏洞的研究，有利于及时修补已知的漏洞、发现未知的漏洞，从而减少系统安全漏洞被攻击者利用的可能性，降低安全事件的发生率。因此，深入研究漏洞的相关知识是非常有必要的。

2.1 漏洞概述

2.1.1 漏洞的概念

漏洞，是计算机系统在硬件、软件、协议的涉及于实现过程中或系统安全策略上存在的缺陷与不足。任何一个系统，无论是软件还是硬件都不可避免的存在漏洞，所以从来都是没有绝对的安全。当然漏洞的存在本身并不能对系统安全造成什么损害，关键的问题在于攻击者可以利用这里漏洞引发安全事件

漏洞的定义[23]，从不同的角度会有不同的表达方式。下面列举几种到目前为止具有代表性的几种表达方式。

(1).基于模糊概念角度的描述

从模糊概念角度来讲，漏洞是指系统中存在的任何不足或缺陷，是在许多层次和角度下可以察觉到的违背功能，是用户、管理员和设计者意愿的一种违背，并且这种违背背景是由外部对象触发的。

(2).基于访问控制角度的描述

从访问控制角度来讲，漏洞是指导致操作系统执行操作与访问矩阵所定义安全策略之间相冲突的因素。按照这一定义，需要清楚地指明一个访问控制矩阵，即指明对系统中的哪些操作是允许的，哪些是不允许的。

(3).基于状态空间角度的描述

从状态空间角度来讲，漏洞就是指区别于所有非受损状态的容易受攻击的状态特征。漏洞就是描述容易受攻击的状态。

上面从不同角度给出了漏洞的概念。简而言之，漏洞就是指系统中存在的缺陷和不足，这种不足可以导致潜在的安全威胁。

2.1.2 漏洞的分类

对漏洞的分类分级研究，目的在于更好地描述、理解、分析并管理已知的系统安全漏洞，在此基础上进一步地预测或者主动发现未知的安全漏洞。

(1) 漏洞分类

本文根据漏洞级别、漏洞成因、漏洞影响、受影响的系统和漏洞利用方法对漏洞进行了分类,主要描述如下:

1. 根据漏洞成因分类

漏洞成因[24]指的是漏洞是由于何种原因造成的，大体分为两类，编码错误和环境配置错误。如果进一步划分，可以分为：

输入验证错误(input validation error)，对用户输入的数据进行合法性检验；

边界条件错误(boundary condition error)，对边界条件进行有效性验证；

缓冲区溢出错误(buffer overflow)，对输入缓冲区的数据进行长度和格式的验证；

访问验证错误(access validation error)，访问验证存在逻辑错误；

意外条件错误(exceptional condition error)，程序逻辑未考虑意外和特例；

环境错误(environment error)，系统或软件环境错误或不匹配；

配置错误(configuration error)，系统或软件的参数或策略配置错误；

竞争条件错误(race condition error)，时序或同步存在错误；

其他错误(others)。

2. 根据漏洞影响分类

漏洞影响是指漏洞被利用后会对系统造成哪些危害。这些危害可以分为四类：

对数据的访问(access to data)；

对命令的执行(execution of commands)；

对代码的执行(execution of code)；

拒绝服务(denial of service)。

3. 根据漏洞利用类型分类

对漏洞的利用可以分为远程和本地两种。

4. 根据受影响的系统分类

现存的操作系统多种多样，并且每种系统都有多个版本，但是这些系统可以大体上非 5 类：Dos 系统；Windows 系统；Unix 系统；Linux 系统；其他系统。漏洞数据库采取了这种不太精确的系统分类方式，其实操作系统根据类型、版本、发行商、补丁号可以分为许多类型，例如 Windows xp home sp2 等。

2.2 漏洞挖掘技术原理

研究漏洞挖掘技术的主要目的是尽可能地找出软件中潜在的漏洞，方法多种多样，采用的技术也因人而异。传统的漏洞挖掘技术模型如图 2-1 所示。

由图 2-1 可知，传统的漏洞挖掘技术分为静态分析和动态分析两大类。其中，静态分析技术主要包括:有向图分析、污点数据传播分析、IDC 脚本分析、整数限制分析和补丁比较分析等;动态分析技术主要包括:格式分析、黑盒测试和虚拟堆栈分析。

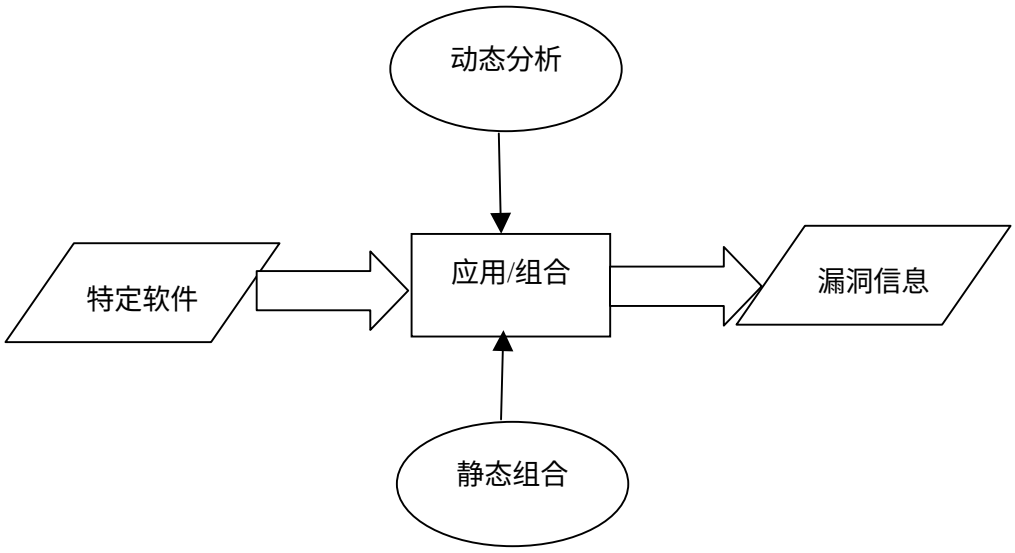


图 2-1 漏洞挖掘技术模型

2.2.1 静态分析技术

静态分析技术是指在程序非运行情况下，对软件汇编代码进行分析，以发现一些潜在的漏洞。目前主要的静态分析技术包括有向图分析、污点数据传播分析、IDC脚本分析和整数限制分析以及补丁比较技术等。这些技术的应用都需要通过特定的逆向平台对软件程序进行反汇编，得到相应的反汇编文本。实际工作中，常用的反汇编工具包括 IDAPro、W32Dasmv [25]、HVIEW 等。其中， IDAPro 是目前功能最强大的、具有图形交互界面的反汇编工具，它能自动识别各种处理器和编译器，分析软件的隐藏功能以及识别无符号的库调用。

(1)有向图分析

有向图分析是最直接的一种静态分析技术，实现思路也很简单。利用 IDAPro 得到软件的反汇编文本，搜索汇编代码中的 Call、Ret 指令，即搜索所有的函数调用与定义。搜索算法既可采用广度优先搜索算法，也可以采用深度优先搜索算法[26]。然后，按照软件逻辑流程构造出函数调用关系图如图 2-2 所示。每个函数调用，包括那些可疑的字符串操作函数(如 strcpy、printf 等)，都作为有向图的一个节点。每个节点记录着函数入口地址、分配的堆栈大小、局部变量使用情况、调用者传递的参数、返回地址等信息。

有向图分析的难点源于编译器的优化。诸如 strcpy、sprintf 等字符串操作函数的调用在 VC6.0 及以上版本编译器编译的时候都被硬编码到了软件程序中。要在汇编代码中找到这些函数调用，必须对函数相关指令进行特征匹配。

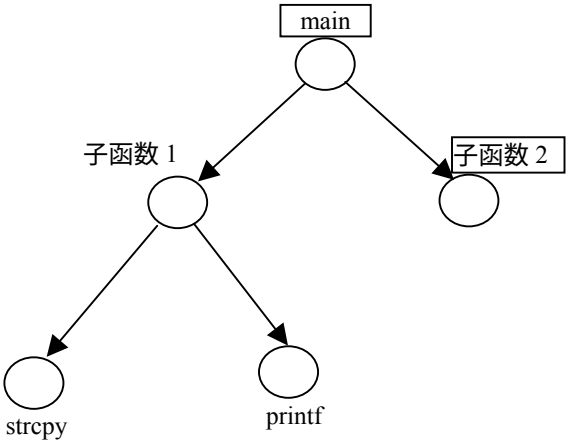


图 2-2 有向图示例

(2)污点数据传播分析

污点数据传播分析是挖掘漏洞的另一种静态分析技术，来源于 Perl 中“ tainting ”的机制。与有向图法注重检查可疑函数的调用不同，污点数据传播分析的思想在于监控所谓的“ 不可信数据 ”，即污点数据的传播轨迹。将污点数据标记为“ tainted ”，“ tainted ”数据通过程序语句进行传播，任何对“ tainted ”数据的操作(如添加，合并，插入等)，其结果值也是“ tainted ”。

针对非开源软件，污点数据传播分析又称为双向数据流分析。所谓双向，是从污点数据输入开始，自上而下形成一棵传播树;二是反汇编文本中搜索 strcpy、sprintf 等可疑函数的调用，然后从这些函数调用的地方开始向上回溯，找到两者相交的位置，从而产生污染流路。分析交点处指令，判断是否进行了污点数据的长度检查，进而判断漏洞是否存在。

显然，由于存在字符串操作函数的搜索，污点数据传播分析仍然受到编译器优化的影响。另一个难点是完整污点数据传播树的构造。从确定污点数据输入处开始，到理清所有相关处理代码之间的逻辑关系，加上大量的分支指令，构造传播树是一个相当复杂的过程。

(3)IDC 脚本分析

同样是基于 IDAPro 的静态分析技术，IDC 脚本分析在挖掘漏洞，尤其是缓冲区溢出漏洞方面相比前两种技术更有效。IDAPro 除了具有极强的反汇编和图形交互界面外，还提供了 IDC 自动化脚本功能，使得用户可以编写特定目的的自动化脚本对反汇编数据库内的内容进行处理。

IDC 漏洞挖掘脚本的编写思想并不复杂。通过 FindText 函数搜索汇编代码中的数据输入函数(如 RaedFile、Recv 等)调用，在这些代码地址后面一段小范围内搜索诸如 strcpy、sprintf 等数据操作函数调用。如果找到，则记录调用者地址，并检查在数据输入函数与字符串操作函数调用之间是否进行了数据长度检测。如果没有则说明可能存在漏洞。

IDC 脚本分析在理论上可以发现不少简单的漏洞。Halvar Flake 开发了基于 IDC 的漏洞挖掘脚本工具 BugScam[27]、BugScan，但受到目前反汇编技术的影响，这些

工具只能找出一些简单错误引发的漏洞。一个改进的方法是加入更多的可以匹配的模式，以及在反汇编技术允许的范围内完善确定缓冲区大小的算法。Halvar Flake 认为静态分析如果想更有效的话，需要一个更为强大的逆向工程平台，现在 IDAPro 是远远不够的。

(4)整数限制分析

整数范围限制是另一种静态分析技术，是指将漏洞挖掘抽象为整数范围分析的问题[28]。针对非开源软件，整数范围限制的挖掘模型包括汇编代码分析和建立数学模型。首先是对 IDAPro 得到的汇编代码进行分析，产生整数范围限制，然后对产生的限制进行分析从而筛选出存在安全漏洞的代码[29]。这里的整数范围，包括数据缓冲区的分配范围和实际数据的长度范围。

定义数据 s 的缓冲区分配长度 A(s)的范围为[a,b]，实际数据的长度范围 L(s)为[c，d]。那么就会出现三种可能的情况。

1 如果 b≦c，可以判断不存在溢出漏洞;

2 如果 a>d，判断存在溢出漏洞;

3 如果两范围重叠在一起 d>b>c>a，就不能得出肯定的判断，需进一步分析。

整数限制分析难点在于对整数范围的分析。如果是简单的数据使用引起的漏洞，是比较容易挖掘的;但很多漏洞往往是由一个错综复杂的关系而产生的，因此容易出现由不精确的范围分析引起的误报。但作为一个自动化的软件漏洞静态分析技术，它已经不错了，毕竟可以减轻人员审计的大量时间和精力。

汇编文本来源于 IDAPro，这就必然会产生大量的无用信息，对这些无用信息的分析将消耗大量的时间。如果软件体积很大的话，例如在 500KBYTE 以上，则可能根本无法保证能在相应的多项式时间内完成漏洞挖掘工作。要进一步改进的话，必须引进更加强大的反汇编工具，但现在尚未出现。

在实际的工作中，静态分析技术往往只作为漏洞挖掘的辅助手段，我们更多的还是采用动态分析技术。

2.2.2 动态分析技术

动态分析技术是指在程序运行状态下，通过测试、调试等手段来挖掘软件中潜在的漏洞。传统的动态分析技术主要包括:格式分析、虚拟堆栈分析和黑盒测试。

(1)格式分析

本文将常规漏洞定义为两种类型:协议格式解析漏洞与文件格式解析漏洞。如果漏洞是软件在处理网络通信数据时引发的，则称为协议格式解析漏洞，这类漏洞涉及的协议特指各 TCP/IP 护应用层协议;如果漏洞是软件解析本地文件格式时引发的，则称为文件格式解析漏洞。

1 协议格式

本文中协议特指各 TCP/IP 应用层协议，如 HTTP[29]、RPC[30]、Telnet[31]、FTP[32]、SMTP[33]、POP3[34]等几类。协议格式分析[35]目的是根据协议格式的标准定义，充分了解其中各字段的含义、尺寸等信息。软件之间通信时，程序将从整个数据包中对协议所有字段一一解析。解析某些字段(如 HTTP 协议的应用数据部分)时，会涉及到对字符串等数据的操作，而漏洞也就是在这些操作过程中引发的。

实际工作中，要研究协议格式，除了参考相应的 RFC 文档外，对通信数据包的分析也是很有必要的。通信数据包分析通过一些网络协议分析工具来实现，典型的网络协议分析工具包括 Ethereal[36]、Sniffer Pro[37]、Iris 等，它们都需要 winpcap 驱动来支持。这些网络协议分析工具不仅能有效地截获通信数据包，还能对不同的字段进行详细解析。

2 文件格式

本文涉及的文件格式泛指 Windows 支持的各种文件格式，如.doc、.pdf、jpg 等，统称它们为客户端文件格式。客户端文件格式的共性就是数据字段大多都以“ 标识 长度 数据 ”的形式出现。分析清这些数据字段，再结合动态分析技术，就可能挖掘出漏洞。实际工作中，具体文件格式的分析可通过一些辅助工具来实现。例如，通过 VC6 自带的 DFView 具查看 word 的.doc 文件内容，如图 2-3 所示。但是，这些工具只能按照格式给出文件的大致分块信息，而不提供各字段详细的解释。要深入分析文件格式，可以将新旧文件版本对照研究。

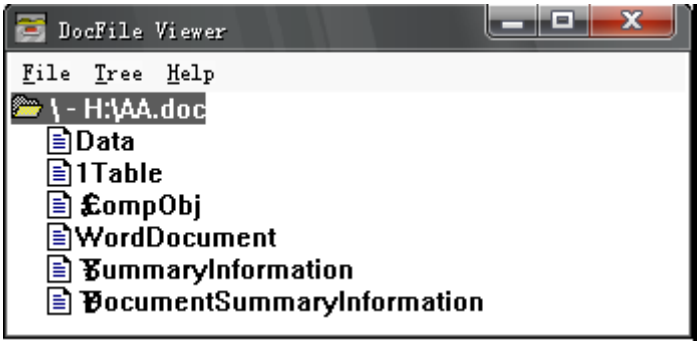


图 2-3 DFView 格式查看功能

(2)虚拟堆栈分析

虚拟堆栈分析是挖掘缓冲区溢出漏洞常用的动态分析技术，其思路是:在格式分析的基础上，动态拦截[38]软件运行时所有的可疑函数调用，并同时创建该函数调用的虚拟堆栈[39]。在虚拟堆栈中，记录所有函数缓冲区的使用情况。对于可疑函数所使用的缓冲区，先分析它是位于堆中还是在堆栈中，针对堆和堆栈的处理有所不同。接着，获得相应的数值化描述信息，例如缓冲区位置、大小、堆栈中的函数返回地址等信息，最后与限制条件相比，判断是否存在安全错误，最后通过人工分检，进而挖掘漏洞，分析流程如图 2-4 所示。

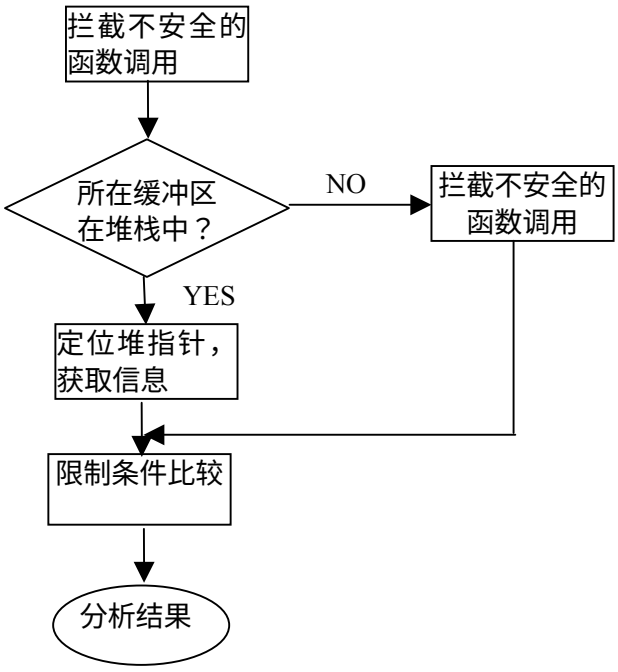


图 2-4 虚拟堆栈分析流程

虚拟堆栈分析流程比较简单，但实现起来需要解决下面两个问题。

1 拦截可疑函数

可疑函数包括 strcpy 等字符串操作函数，通过 HOOK API 技术就可以完成拦截。但正如前文所述，从 VC.60 开始的编译器都进行了优化，一些字符串操作函数都被硬编码到了程序中，我们看不到函数名，而只是一些指令，解决的方法是指令配对。

2 虚拟堆栈

虚拟堆栈用于记录堆栈的调用情况而新开辟的一片内存。分析过程中，通过虚拟堆栈可以很方便的定位出所分析的字符缓冲区在堆栈中的位置，生存周期等。虚拟堆栈如图 2-5 所示。

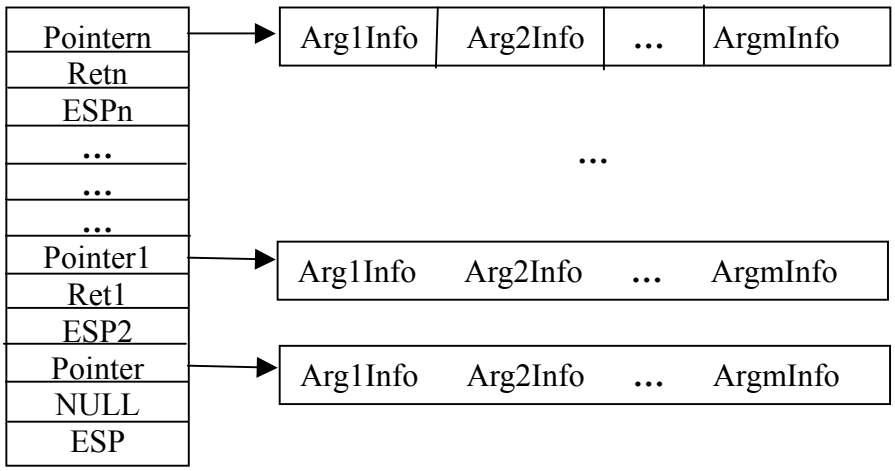


图 2-5 虚拟堆栈

图 2-5 中，ESPn 表示系统为第 n 个函数的分配的堆栈起始地址;Retn 表示第 n 个函数的返回地址;Pointern 表示第 n 个函数的各变量的堆栈信息指针;ArgInfo——ArgmInfo 表示变量的堆栈信息，包括变量的存储地址和大小。

虚拟堆栈分析难度在于拦截之后的分析判断，如果没有人工分析的介入，很难直接挖掘出漏洞。因此，漏洞挖掘的成功率在很大程度上依靠人的经验，要实现系统化困难很多。

(3)黑盒测试

黑盒测试是最直接的动态分析技术。黑盒测试的前提是可以与程序进行交互，故而不需要源代码。这种测试的原理是将程序看作一个只具有输入和输出的黑盒。测试人员对程序的了解只能是基于输入和输出的关联性进行分析。

黑盒测试一个很关键的问题是测试数据的选取。既然知道软件能做什么，那么也就大概知道该软件的安全输入域的范围，所以测试最好是选择软件安全输入域外的数据。还有溢出漏洞在漏洞中占了很大比例，并且这类漏洞多是由特殊字符或超长字符串导致的，如果分析总结以往的溢出漏洞的利用方法，研究这些超长字符的构造方式同样有利于对测试数据的选取。此外，在协议测试中，例如 FTP、SMTP 和 POP3 等，测试人员必须要了解这些协议的命令和命令格式，还要分析这些命令在什么情况上会产生溢出错误，如此才能构造有效的测试数据。

黑盒测试的步骤是这样的，首先，分析相同领域或者相似软件的安全问题，归纳总结出一些规则或者模板；其次就是工具规则或模板构造测试数据对软件进行测试；再次验证软件输出的正确性；最后确定一些疑似的漏洞点。

1 协议格式解析漏洞的挖掘

本文涉及的协议都是 TCP/PI 应用层协议，它们格式公开，并存在一些共同点。例如，各协议数据包都是按照以太网头、PI 头、TCP 或 UDP 头、应用头、应用数据的顺序封装的，如图 2－6 所示。同时，大多数软件通信都是在客户端(Client)与服务端(Server)之间发送数据包，而漏洞往往都在服务端软件产生。因此，构造通信数据包，进行黑盒测试是比较容易实现的。只需要将数据包的各字段按照测试方案依次变换为一些非常规的形式，然后发送到服务端，最后接收并分析返回信息，分析服务端异常现象。

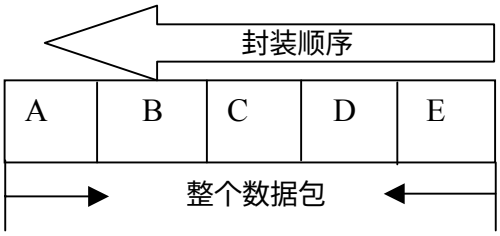


图 2-6 应用层数据包结构

图 2-6 中，定义

A：以太网头

B：IP 头

C：TCP 或 UDP 头

D：应用头

E：应用数据

2 文件格式解析漏洞的挖掘

相比协议格式，大多数文件格式[40]要复杂得多，而且格式细节都未公开。要通过深入分析文件格式后构造特定的文件数据，进行黑盒测试，难度要比挖掘协议格式解析漏洞大很多。例如，要研究 word 解析.doc 文件格式中的漏洞，我们必须先研究.doc 文件格式。但是最新.doc 文件格式微软并未公布，要查看文件格式，只能通过 UltarEdti32 等软件查看十六进制数据，或通过 DFView 查看简单的格式分块。因此，黑盒测试一般不适合文件格式解析漏洞的挖掘。

总体说来，黑盒测试通过测试用户输入非常规的命令行参数、图形界面交互数据、网络数据、环境变量等数据来触发软件潜在的漏洞。如果测试对象出现地址访问非法的异常现象，并导致程序退出，则可能存在漏洞。黑盒测试实现简单，但由于它无法真正理解程序的流程，并不能够将测试对象的所有问题同时暴露出来，协议格式解析漏洞的挖掘效率也往往由于协议分析的时间长而比较低下。

2.3 小结

分析表明，在效率上动态分析技术要比静态分析技术更为实用和有效。所以一般漏洞挖掘采用动态分析技术，而从上可以发现它们存在一个共同缺陷:漏洞研究的协议格式相关性较强，重复性工作多，效率较低。尤其是协议格式解析漏洞挖掘，总需要先熟悉漏洞涉及的协议，然后才进行各种分析，从而产生大量的重复性操作。

采用 fuzzing 的方法进行漏洞挖掘是近年来的信息安全研究领域的热点问题，本文将采用 fuzzing 的方法对网络协议的漏洞挖掘进行研究，可以解决传统漏洞挖掘技术存在的一些不足，具体的解决方法将在第 3 章详细描述。

3 基于 Fuzzing 的网络协议漏洞挖掘技术研究

第 2 章描述了传统漏洞挖掘技术的基本原理及难点，本章在分析传统漏洞挖掘技术缺陷的基础上，提出一种基于 Fuzzing 的网络协议漏洞挖掘技术。

3.1 传统漏洞挖掘技术的缺陷

基于网络协议的漏洞挖掘技术研究中，协议格式的分析是非常关键的，本文将网络协议格式抽象为如图 3-1 所示的模型。

<QUES>	<LENGTH>	<STRING>	<END>

图 3-1 协议格式

图 3-1 所示的协议格式中的变量表示如下。

QUES ：数据包的格式部分，每种协议具有不同的格式，是固定值。

LENGTH：是后面的变量 STRING 的长度，随 STRING 的不同而不同，可变。

STRING ：数据部分，可以填充任意内容，是可变的。

END ：数据包的格式部分，是固定值。

研究发现基于网络协议格式的漏洞挖掘技术的关键是如何构造测试数据包的问题。而传统的黑盒子测试技术的实现是通过建立基于网络协议的客户端，然后测试者根据自己的经验构造一些他们认为会导致溢出的数据，并将其发送到目标系统，进而检测目标软件的漏洞。这种方法具有明显的缺陷：

如果网络协议是由客户端和服务器端的 API 共同决定的，或者测试者拥有客户端的源代码，那么在构造数据包的时候，这些 API 和客户端的源代码就会影响测试者的判断，导致测试的结果和开发者的一些假设判断有很大的差别。

尽管测试者掌握了已知协议的所有知识，但是有构造一个基于此协议的客户端测试程序还是很难的，况且这样的测试程序对于别的协议未必可用，这样很大程度上浪费了资源。

上面的几个缺陷总的来说是如何方便的构造有效的测试数据包的问题。基于图 3-1 所示的协议格式的模型，传统的通过故障注入[41]的方法来挖掘漏洞的模型图如图 3-2 所示。即是在客户端构造大量的随机数据，再将它们发送到服务器端用以检测目标软件的漏洞。但是发送的大量数据会因为其不合法而被拒绝。

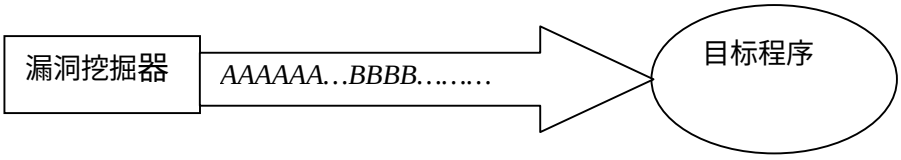


图 3-2 传统漏洞挖掘技术

服务器端对每个数据包都会做检查：

If (开始头 4 bytes !=”QUES”)

{

拒绝数据包；

}

可以看出，构造的大量的随机数据会被服务器拒绝，导致漏洞挖掘的低效性。表明了进行漏洞挖掘必须要对目标软件的协议格式进行分析，得出数据包的固定部分，在图 3-1 中是<QUES>和<END>,然后基于固定的值对 STRING 进行随机的构造，再发送数据到目标程序，如图 3-3 所示。

图 3-3 传统的 Fuzzing 技术

由上可知，基于 Fuzzing 的漏洞挖掘技术较传统的漏洞挖掘技术有了很大的进步，因为构造的数据比随机发送的数据更为有效，可以部分通过服务器的检查，但是服务器端还会做出如下检查：

If (开始 4 bytes !=”QUES”)

{

拒绝数据包；

}

```
if(LENGTH!=strlen(STRING))
{
        拒绝数据包；
}
```

很显然，服务器端除了要检查数据包的固定格式外，还要对可变部分的长度做计算并检查。但是由于 LENGTH 是随填充的变量 STRING 而变的，所以利用传统的方法很难动态的得出 LENGTH 的值，最后也是很难通过服务器端的检查的。所以需要构建新的更为有效挖掘模型，如图 3-4 所示。

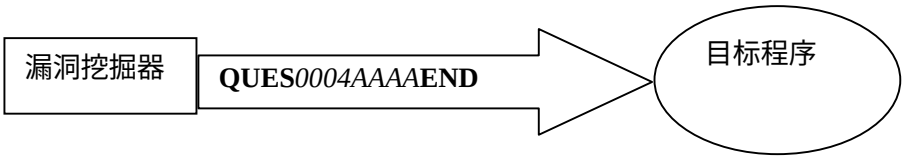


图 3-4 基于 Block-based 的 Fuzzing 框架

这种模型的实现代表是 SPIKE,提出了基于 Block-Based[42]协议分析的方法，可以高效地实现模型 3-4 提出的要求，动态地计算 STRING 的大小并填充到 LENGTH 域，这样构造的数据就可以通过服务器端的检查，使得发送到目标程序的数据变的非常有效。这种方法很大程度上提高了漏洞挖掘的效率。

然而在效率得到保证后，我们来看挖掘器的参数输入空间：

$$N = 256^m = 2^{8m} \qquad (3-1)$$

其中 N 为参数输入空间，m 为字符串的长度；由(3－1)式可知，图 3-3 的表示中 STRING=’AAAA’，得到 m＝strlen(STRING)＝4；$N=256^4=2^{32}$ 可以看出 N 的数量是非常大的，并且输入的参数的空间大小 N 是由需要被替换的字符串的长度 m 决定的。

传统的基于 fuzzing 的漏洞挖掘技术比原始的故障注入技术拥有的优点：不需要源代码；对二进制代码的需求也不高；挖掘到的每一个漏洞都是实际存在的；不需要非常高深的专业知识；自动和半自动化程度高；可以总结为实用性很好。但是传统的基于 fuzzing 的漏洞挖掘方法也具有如下缺陷：构造的数据大多是无用的；漏洞挖掘会耗费大量的时间；不能够发现所有的漏洞。

3.2 基于 Fuzzing 的协议格式漏洞挖掘技术

3.2.1 技术模型

在分析了传统漏洞挖掘技术的缺陷后，本文在原有的 Fuzzing 技术框架的范围内，构造了新的技术模型：基于权值衡量的 Fuzzing 技术。由 3.1 可知，基于 Fuzzing 的网络协议的漏洞挖掘技术得以顺利进行的前提是：基于网络通信的原则进行漏洞挖掘；理解并解析协议格式；解决长度域匹配的问题，构造自己的参数替换库。本文在这四个前提下，采用了基于参数权值衡量的技术，对输入参数进行权值衡量，达到了减少输入空间的目的。在完成上面的技术模型的基础上，自动构造满足协议格式的网络数据包并发送到目标程序，显示漏洞信息。

(1)基于协议的通信

网络通信基本上都是基于客户端-服务端(C-S)模式的，如图 3-5 所示。

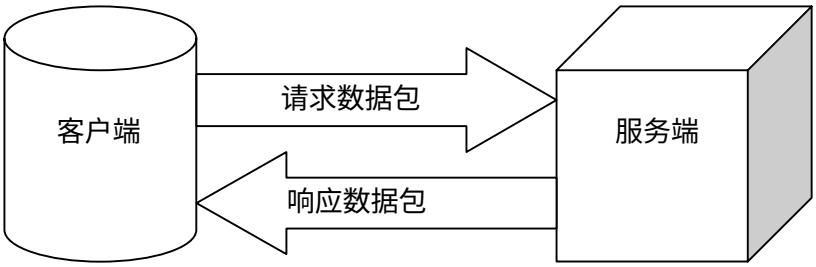


图 3-5 C-S 通信模型

两端利用 TCP/IP 协议进行通信，客户端发送请求数据包，服务端返回响应数据包，即 Request-Response 方式。因此，网络通信的共性就是数据包的通信。每个服务软件都有相应的应用层协议，应用层协议决定了服务端对请求数据包里的内容如何解释，而协议细节也就体现在数据包中。

(2)理解并解析协议格式

挖掘器必须可以识别已知和未知的协议格式,网络数据包格式抽象为：

<QUES><LENGTH><STRING><END> (3－2)

由(3－2)可知，数据包中的固定值 QUES 和 END 是必须要知道的，只有在理解协议格式的基础上，才能构造出合法的数据包，这是区别于随机发送数据到目标程序的原始的故障注入式的方法，实验证明这种方法是非常有效的。

(3)解决长度域匹配的问题

传统的黑盒测试是通过采用 Perl 脚本语言来设计一些网络数据，并用一些较大的字符串来替换小的字符，用以发现缓冲区溢出。但是，当将这项技术扩展到更为复杂的协议应用的时候，比如协议由几层构成(比如 OSI 七层协议模型)，那么，它们每层的大小是相关联的目的是为了防止被简单的网络数据包模仿。事实上，当今的大多数协议都依赖于别的协议，比如 HTTP。

可以将任何协议解析为长度域和数据域，但是，当测试者试图构造他们的脚本程序的时候，他们会发现在构造底层数据包的时候，必须要知道所有高层协议的长度。如果不能很好地处理层与层间长度域的相互关系的话，构造的数据包就会被服务器拒绝。

例如，当测试者试图发送一超长字符串到特殊的应用程序的时候，不是简单的用一超长字符串来替换字符就可以的。这测试者必须要动态更新长度域：处于 HTTP 报文头里面的。对于更为复杂的协议，会有更多的长度域需要被更新。盲性的 Fuzzing，即传统的 Fuzzing 技术通过发送随机数据和不符合格式的数据到服务器端的方法来发现漏洞是会浪费大量的时间的。

但是，通过创建一函数来计算每一字符串长度的工程量是巨大的，并且，它的重用性也不高。因此，需要一技术框架可以将底层协议和已知的高层协议的长度域隔绝起来。对于人工模拟协议格式，最好的方法是将协议看作是一长串的字符，而不是蛋壳包蛋壳的协议模型。Block-Based 协议描述语言允许测试者在创建数据包的时候，创建许多数据块并将它们绑定到每一长度域，这样一旦那些数据块的大小因为被较大的字符串替换的时候，挖掘器就能够重新计算一个准确的数据块的大小并发送一个正确的长度值请求。本文技术采用基于 Block-Based 的协议描述语言来解决长度域动态匹配的问题，由此可以很大程度上减少输入空间的大小。

(4)构造自己的参数替换库

不是通过随机构造字符串来替换数据包里面的可变数据，而是构造自己的替换库文件，这样的效率比较高。基于 Fuzzing 的技术比较经典的框架 SPIKE2.9 有 681 个这样的替换条目。

(5)基于参数权值衡量的 Fuzzing 技术

传统的黑盒子测试技术包括 Fuzzing 技术的数据输入空间是无限的，通过输入随机数据来进行漏洞挖掘。这种测试的方法是最为原始的，会带来时间和空间的巨大浪费。下面以数据的角度来描述这个问题：

假设用来进行替换的字符串或字符的数目为 L，被测试的字符串的个数为 F，那么漏洞挖掘的复杂度 C。

$$C=L*F \tag{3-3}$$

由(3－3)可知，要降低复杂度 C，必需要通过减少 L 或 F 来做到。但是，由于 L 里面的替换数据或字符都是不可缺少的，减少 L 的数目会降低效率，而盲目的减少 F 的数据也会影响到挖掘的效率。

衡量测试数据权值技术，通过修改 F 的大小，或者对 F 的变量数据进行重新的排序，可以使得 F 的数据变得更为有效，降低 C 的复杂度。如果 L 的有上千条的数目，可以想象哪怕是减少 F 的一个输入，都可以取得很高的效率回报。

为了便于本文的概念性的表述，对一些常用语做一下定义：

定义 3－1(Tracer/Debugger) 一个 Tracer 是可以列出目标软件的所有动态执行函数的调试器。通过绑定动态分析工具到目标软件上，可以知道哪些参数会被不安全的函数调起。如果一旦发现了漏洞，可以用调试函数列出对应的详细信息。当用户可以控制那些不安全的函数的时候，利用这些函数进行 Fuzzing 是非常关键的。

定义 3－2(Marker) 用户输入的每一个变量或字符都可以看着是 Marker。

定义 3－3(Fuzzer) 产生并发送测试数据到目标程序，是系统的核心部件。

基本上讲，整个过程是自动地进行分析的，当一个 marker 作为一个字符或数值被不安全函数调用的时候，对应的这个 marker 的权值就会增加，并把结果实时地传输的 fuzzer。当所有的 marker 都被测试和权值衡量后，fuzzer 将会首先调用权值最大的进行测试。在这种情况下，会加大首先找到缓冲区溢出漏洞的概率。 综上，基于 fuzz 的网络协议漏洞挖掘技术模型如图 3-6 所示。

由图 3-6 可知，技术模型主要包括：解析协议格式；自动构造测试脚本；调试跟踪输入参数以及测试数据包的构造。

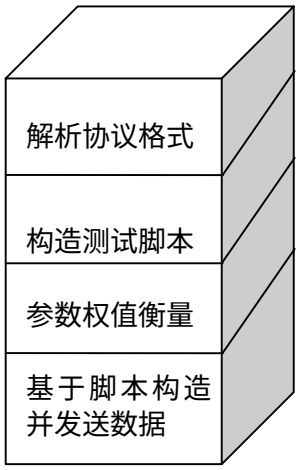


图 3-6 基于 Fuzzing 的网络协议漏挖掘技术模型

3.2.2 解析协议格式与构造测试脚本

基于 Fuzzing 的漏洞挖掘技术有别于传统的漏洞挖掘技术的关键点在于对网络协议的解析，而不是发送随机数据。采用人工的方法对协议进行解析会耗费大量的时间和人力，是低效的做法。因此本文采用自动化的方法解析网络协议。协议分析的一般方法：抓取协议样本包，然后再解析协议格式。

本文的思路在于采用自动化的抓包工具比如 Wireshark，Sniffer pro 等。Wireshark1.0 利用自带的协议识别引擎称为解析器可以识别出 935 种以上的协议格式，因此可以利用 Wireshar 来解析网络协议。

在理解协议格式的基础上，下一步要做的工作是撰写测试脚本。通常，人工抓写的脚本常常会出现这样或那样的错误，传统的漏洞挖掘技术表明测试脚本的撰写是最耗费时间的一项工作。本文采用基于 Wireshark 工具抓包保存的 PDML 文件格式，自动地实现测试脚本的构造。当然有的协议 Wireshark 不能识别，但是同样也可以被当作二进制文件进行自动的测试脚本撰写。模型图如图 3-7 所示。

(1)抓取协议样本包

通过数据包截获提供协议样本包，可以方便到具体应用环境测试。实际工作中，数据包截获可以在系统用户态截获和核心太截获两个层面进行。在用户态的数据包截获技术主要包括原始 Socket[44]、Winsock Layered Service Provider(LSP)、Windows 包过滤接口、以及替换系统自带的 Winsock 库等，这些技术共有缺陷是无

法处理 TCP/IP 协议栈中底层的数据信息。在核心态的数据包截获技术主要通过 TDI、NDIS、WinpCap[44]等各种驱动接口来实现。图 3-8 显示了基于 WinpCap 驱动的 Wireshark 工具截获的 TCP 包。

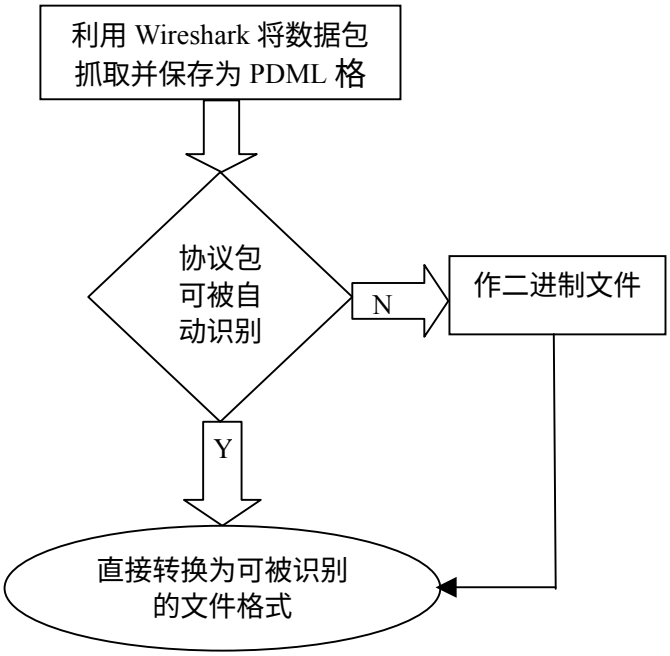


图 3-7 解析协议格式与构造测试脚本文件模拟图

由图 3-8 可看到，数据包截获生成一个完整的数据包文件，包括以太网头、IP 头、TCP 或 UDP 头、应用头以及应用数据。

```
    6 0.003904    211.69.192.86      211.69.192.101     TCP      fpitp > t
⊞ Frame 5 (566 bytes on wire, 566 bytes captured)
⊞ Ethernet II, Src: Giga-Byt_34:0c:61 (00:20:ed:34:0c:61), Dst: Internet_03
⊞ Internet Protocol, Src: 211.69.192.101 (211.69.192.101), Dst: 211.69.192.
⊟ Transmission Control Protocol, Src Port: italk (12345), Dst Port: fpitp (
    Source port: italk (12345)
    Destination port: fpitp (1045)
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 513    (relative sequence number)]
    Acknowledgement number: 513    (relative ack number)
    Header length: 20 bytes
  ⊞ Flags: 0x18 (PSH, ACK)
    Window size: 65023
  ⊞ Checksum: 0x96c6 [correct]
  ⊞ [SEQ/ACK analysis]
⊞ Data (512 bytes)

0000  00 e0 4d 03 02 b9 00 20  ed 34 0c 61 08 00 45 00   ..M.... .4.a..E.
0010  02 28 7a 3c 40 00 40 06  97 4c d3 45 c0 65 d3 45   .(z<@.@. .L.E.e.E
0020  c0 56 30 39 04 15 58 fb  dd d6 02 a8 bd 47 50 18   .V09..X. .....GP.
0030  fd ff 96 c6 00 00 00 00  00 00 e3 00 28 00 00 00   ........ ....(...
0040  00 00 30 7d 14 00 68 00  00 00 60 fd 00 00 00 00   ..0}..h. .. ......
0050  00 00 02 00 00 00 01 00  00 00 06 00 62 00 00 00   ........ ....b...
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 00 00 00 00 03 00 00  00 08 00 00 00 35 2e 30   ........ .....5.0
0080  2e 31 2e 33 37 00 00 00  00 00 00 0b 00 00 00      .1.37... .........
```

图 3-8 wireshark 截获的 TCP 数据包

(2)构造测试脚本

回到协议数据包的构造问题上来。要提高漏洞挖掘效率，需要灵活构造、自动发送基于协议格式的同时又具有畸形数据的数据包，这里的“畸形”是指有引发漏洞的可能。为此，我们引入脚本，脚本指定测试数据包的构造方式和内容。

采用 Block-Based 协议描述语言，将协议格式划分长度域和数据域，对于人工分析而言，就可以把协议看成是一串长字符，而不是传统的蛋壳包蛋壳的模型。

基于 Block-Based 的协议描述语言包含如下函数：

```
string("dummy"); /* 定义一常量字符 */
string_uni("dummy"); /* 定义一常量宽字符*/
hex(0x0a 0a \x0a); /* 定义一十六进制常量值 */
block_begin("block"); /* 定义块的开头 */
block_end("block"); /* 定义块的结尾 */
block_size_b32("block"); /* 一个大小为 32 bits 的 big-endian 数据块 */
block_size_l32("block"); /* 一个大小为 32 bits 的 little-endian 数据块 */
block_size_b16("block"); /*一个大小为 16 bits 的 big-endian 数据块*/
block_size_l16("block"); /*一个大小为 16 bits 的 little-endian 数据块*/
block_size_8("block"); /* 一个块的大小为 8 bits */
block_size_hex_string("block"); /*一个十六进制字符大小的块 */
block_size_dec_string("block"); /* 一个十进制字符大小的块 */
block_crc32_b("block"); /* big-endian 中的块的 crc32 */
block_crc32_l("block"); /* little-endian 中的块的 crc32 */
send("block"); /* 发送数据块 */
recv("block"); /* 接收数据块 */
fuzz_string("dummy"); /*fuzz 字符"dummy" */
fuzz_string_uni("dummy"); /* fuzz t 宽码字符 "dummy" */
fuzz_hex(0xff ff \xff); /* fuzz 十六进制值*/
```

利用上面的协议描述函数基本上可以写出任何二进制协议和字符串协议。但是即便有好的协议描述语言，但是由人工撰写协议格式所耗费的时间也是很大的。为此本文构建一个 adc 工具用来将识别脚本语法并将指定的文件转换为 fuzzer 可以识别的格式。同时基于 wireshark 工具可以识别高达 930 多种协议格式，因此也撰写了工具 pdml2ad，用于将 wireshark 抓包保存的格式文件 pdml 转换为 ad 文件，可以被 fuzzer 识别。

3.2.2 调试跟踪输入数据

对于输入参数而言，哪些导致缓冲区溢出的可能性比较大，是根据具体的软件而言的。很多函数没有对参数的长度进行检查。跟踪 marker 被调用的情况，对 marker 的权值进行实时评估，并把相应的结果发送到 fuzzer。跟踪器和挖掘器相连，其主要的作用是对 F 进行重新排序，将那些被调用的 marker 进行标记，同时对 marker 被目标程序不安全函数调用的情况进行记录，这样的好处是，可以增大发现缓冲区溢出的概率。

调试技术，通过读取目标程序的调试文件，对目标软件的危险函数进行定位，并对每一个危险函数设置断点。同时检测目标软件危险函数对 marker 的调用情况。基于上面的分析原理，调试跟踪模块的结构如图 3-8 所示。

由图 3-5 所示，整个调试跟踪模块主要分为建立连接模块，调试模块，发送 marker 到 fuzzer 模块。

(1)与 Fuzzer 建立 TCP/IP 连接

在 TCP/IP 网络应用中，通信的两个进程相互作用的主要模式是客户机/服务器，即客户端向服务器发送请求，服务器接收到请求后提供相应的服务。客户机/服务器的建立基于以下两点：首先，建立网络的起因是网络中断、硬件资源、运算能力和信息不均等，需要共享，从而拥有众多资源的主机提供服务，资源较少的客户请求这一非对称等作用。其次，网间进程通信完全是异步的，相互通信的进程间既不存在父子关系，又不共享内存缓冲池[43]，因此需要一种机制为希望通信的进程间建立一种联系，为二者的数据交换提供同步，这就是基于客户机/服务器模式的 TCP/IP。

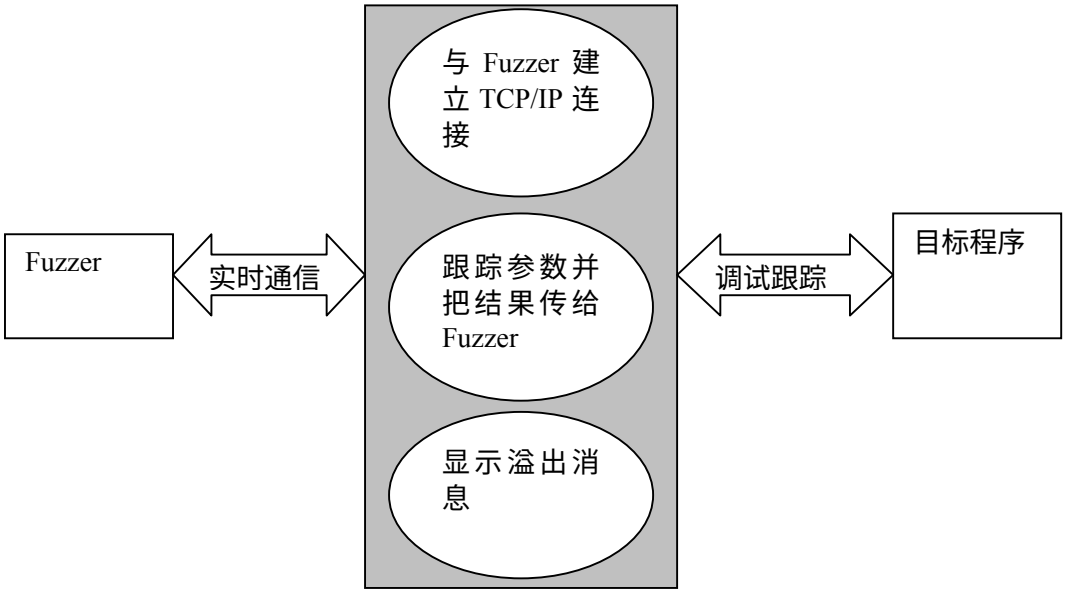


图 3-8 调试跟踪参数模块

客户机/服务器模式在操作过程中采用的是主动请求方式。

首先服务器方要启动，并更具请求提供相应的服务，具体情况如下所述。

1 打开一通信通道并告之本地主机，它愿意在某一公认地址端口上(如 http 为 80)接收客户请求。

2 等待客户请求到达改端口。

3 接收到重复服务请求，处理改请求并发送应答信号。接收并发送服务请求，要激活一新进程来处理这个客户端请求。新进程处理此客户请求，并不需要对其它请求做出应答。服务完成后，关闭此新进程与客户的通信链接，并终止。

4 返回第二步，等待另外的客户请求。

5 关闭服务器。

客户方，对应的情况可描述如下。

1 打开一通信通道，并连接到服务器所在主机的特定端口。

2 向服务器发出服务请求报文，等待并接收应答，然后继续提出请求。

3 请求结束后关闭通信通道并终止。

从上面的描述过程可得出两点。

1 客户与服务器进程的作用是非对称的。因此编码不同。

2 服务进程一般是先于客户请求启动的。只要系统运行，该进程就一直存在，直到正常终止或强迫终止。

上面是连接的通信过程，在 Debugger 和 Fuzzer 建立连接后，Fuzzer 首先向 Debugger 传送需要被跟踪监测的数据 Markers，Debugger 也通过此连接将监测的结果实时地发送到 Fuzzer，让 Fuzzer 可以很好的执行漏洞挖掘，整个过程如图 3-9 所示。

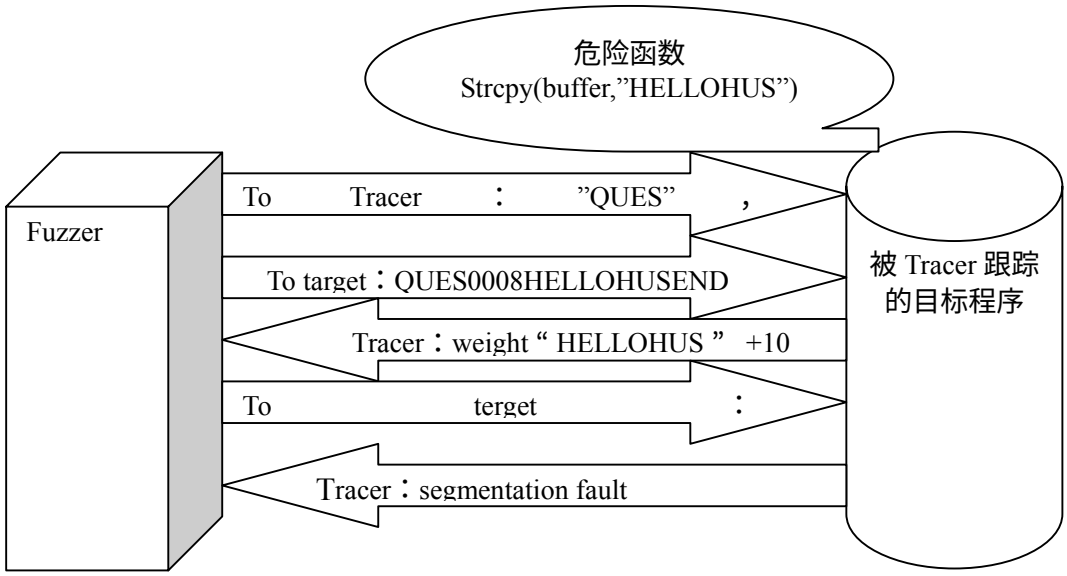


图 3-9 模拟跟踪器在系统中的使用

(2)对目标程序进行调试跟踪

对 markers 的调试跟踪是调试模块的核心部分，根据目标软件的调试文件，对目标软件的危险函数的运行情况进行检测，包括对其设置断点，记录调用参数 markers 的情况。根据需要把这些记录信息提供给漏洞挖掘器进行数据发送，其设计目标如下：

1.对目标程序的危险函数设置断点；

2.记录 markers 被危险函数的调用情况。

那么什么是 marker 呢？marker 可以看作是每一个被输入的参数，比如

<QUES><LENGTH><STRING><END>中的 ' QUES',<STRING>和<END>可以看作是 Marker。对发送的字符串"QUE0009HELLO22C3END 过程"如图 3-9 所示。

3.2.4 测试数据包的构造

基于 Fuzzing 的网络协议漏洞挖掘技术最终体现形式是灵活构造满足特定需要的测试数据包。因此，测试数据包的构造，是整个技术的核心所在，处理流程如图 3-10 所示。

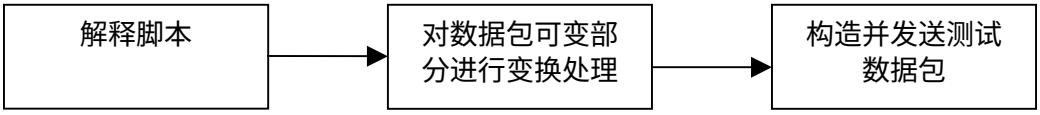


图 3-10 测试数据包构造流程

(1)解释脚本，读取协议包内容

脚本的解释应该与脚本编写规范相配合。由于协议数据包是由脚本直接提供，所以必须严格按照格式对脚本中的协议数据包内容逐行进行一一读取，直到完毕为止，将读取的内容保存在缓冲区中以备调用。

(2)构造测试数据包

读取协议数据包内容后，需要对协议数据包中的可变换部分进行变换处理，然后构造测试数据包。确定漏洞挖掘目标，发送测试数据包这一步主要分为三个步骤进行，具体流程图如图 3-11 所示。

在图 3-11 中：

A:发送原始数据包到目标程序；

B:发送 Marker 到调试器；

C:调试器返回带有权值的 Marker；

D:挖掘器发送带有权值的 Marker 到目标程序。

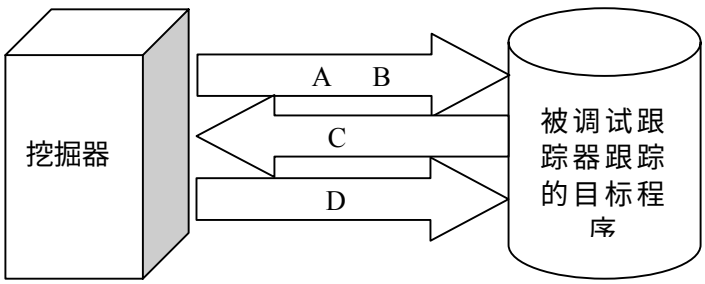


图 3-11 发送数据包

可以知道，发送数据的目标有两个：调试器和目标程序，为此，它们分别有几个要素需要确定：

对调试器

目标 IP：该 IP 地址必须可以访问，一般和发送机器处于相同局域网中；

端口：和调试器他哦那些的端口，一般不被系统常用

对目标程序

目标 IP：该 IP 地址必须可以访问，一般和发送机器处于相同局域网中；

协议：分为 TCP 和 UDP 协议；

端口：正常的应用层服务端口，如 IIS 的 80 端口，RPC 的 135 端口等。

上述要素在测试的时候由测试员手工输入。为了测试方便，目标机器应该使用物理上可控，同时从 IP 也是可以访问的。例如，目标机器和测试数据包发送机器位于相同的内部网中，最好是物理位置相邻。

构造好测试数据包和发送测试数据包是一起进行的。

3.3 小结

针对传统技术存在的缺陷，基于 fuzzing 的协议漏洞挖掘技术做出了相应的改进，其具体优势主要表现在以下两个方面：

(1)漏洞挖掘效率提高

提高漏洞挖掘效率是研究该技术的出发点。不论针对何种通信协议的服务端软件，只要利用抓包工具抓取到对应的数据包格式，就可以方便，灵活的构造测试数据包，进行漏洞挖掘，同时避免了大量的重复性、繁重性的工作，较少的实现了漏洞挖掘的自动花。

(2)灵活性强

对于任何基于协议格式的目标软件，只要可以得到对于的数据包，就可以构造对于的测试数据，就有可能挖掘目标软件漏洞。这样对于任何一个安全研究工作者来说这可以很大限度的处理更多的协议。

4 基于 Fuzzing 的网络协议漏洞挖掘器设计

本章在基于 fuzzing 的网络协议漏洞挖掘技术基础上，给出了相应的漏洞挖掘器在 linux 平台上的设计方案，并将其实现。

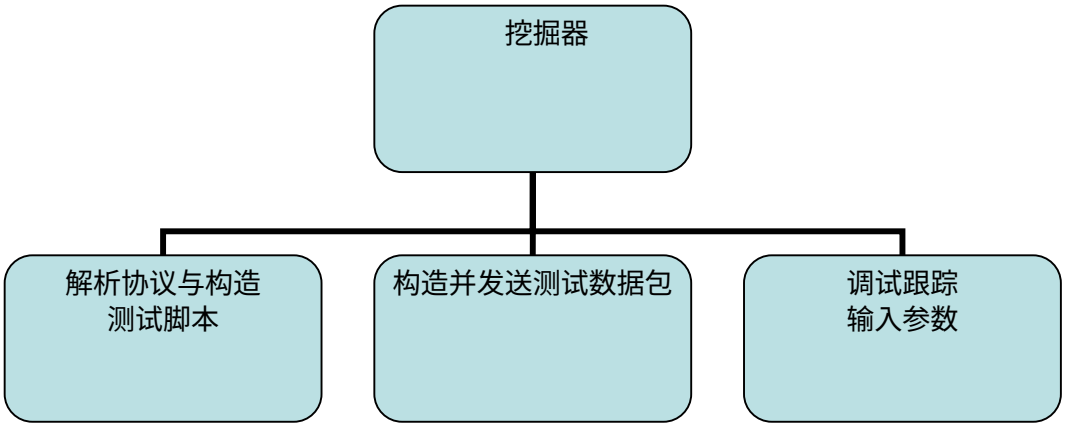
4.1 挖掘器设计构思

挖掘器设计的出发点是提高漏洞挖掘效率，并对协议格式的分析采用自动化的方式进行，大大地减少了漏洞工作这对协议细节的掌握和测试脚本的撰写。进行漏洞挖掘，首先利用转换软件将协议分析工具 wireshark 抓到的协议包进行转换，将转换的结果作为参数为挖掘器所使用，挖掘器再将脚本参数传递给跟踪器，跟踪器触发目标软件，检测参数被目标软件的不安全函数调用的情况。将结果传递给挖掘器，挖掘器根据参数权值的大小，依次对目标程序进行 Fuzzing。最后通过分析跟踪器的信息，判断软件的漏洞信息。

4.2 挖掘器总体结构

(1)挖掘器结构图

漏洞挖掘器由测试脚本构造、调试 markers 值和解析脚本构造并发送测试数据包构成，漏洞挖掘器结构图如图 4-1 所示。



4-1 挖掘器结构框图

(2)挖掘器组成部分描述

挖掘器主要由以下几个部分组成。

脚本产生：包括 PDML 文件[44]转换，和脚本语法检查。脚本产生器的功能是直接构造测试脚本。

Marker 跟踪：在本文中描述为 Debugger/Tracer。采用了 GDB 技术[45]，对 marker 在目标程序中被危险函数的调用进行跟踪，并实时的反馈到 Fuzzer；并显现响应的漏洞信息。

构造并发送测试数据包：在本文中描述为 Fuzzer。根据 Debugger 返回的带权值的参数的值，对目标程序进行 Fuzzing。

4.3 挖掘器逻辑流程

挖掘器模块关系如图 4-2 所示。

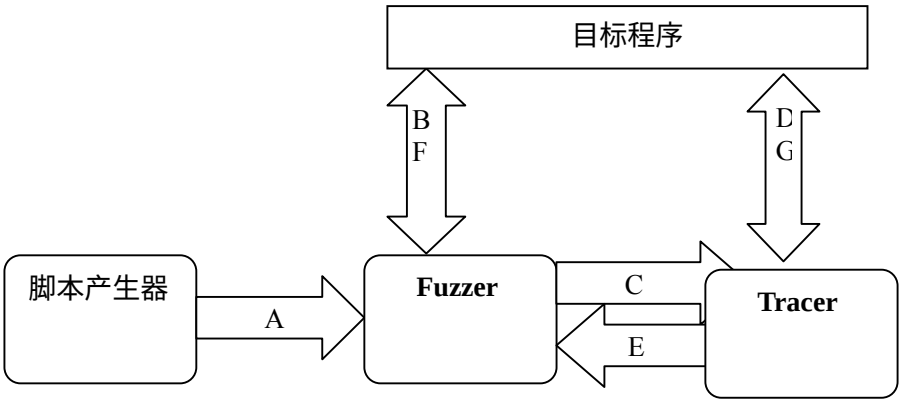


图 4-2 挖掘器逻辑流程

图 4-2 中的定义如下：

A：利用 Block-based 协议语言将协议描述成为 Fuzzer 可以识别的语言，数据包源 Marker

B：与目标程序建立连接，并发送原始数据包到目标程序

C：发送与原始数据包对应的 Markers 到 Tracer

D：Tracer 监测 Marker 在目标程序中被危险函数的调用情况，并设置 Marker 的权值

E：将对 Marker 权值设定后的结果发送给 Fuzzer

F：Fuzzer 利用 Marker 对目标程序进行 Fuzzing

G：在 F 的基础上，显示对应的漏洞信息

4.4 挖掘器主要模块设计

4.4.1 测试脚本模块设计

测试脚本是挖掘器的重要组成部分，功能利用 Block-based 协议描述语言对协议进行描述，屏蔽协议的相关内容，可以很方便的构造数据脚本。整个模块由两个部分构成：利用 Block-Based 协议描述语言对 PDML 文件进行转换,添加测试数据并转换为 Fuzzer 可识别脚本文件;

(1)转换 PDML 文件

PD2AD 的功能是将 wireshark 抓到的协议包保存的 PDML 文件转换为 AD 格式。只要是 Wireshark 支持的协议，并保存为 PDML 文件格式，就可以很好地被转换。Wireshark1.0 可以识别的协议达到 935 种。对 wireshark 不能识别的协议，PDML 可以调用自己的函数自动解析里面的明文标识并转换为相应的文件。

由于需要对 PDML 文件进行解析，因此本文引入了 libxml2 库。Libxml2 是一个 xml c 语言版的解析器，本来是为 Gnome 项目开发的工具，是一个基于 MIT License 的免费开源软件。它除了支持 c 语言版以外，还支持 c++、PHP、Pascal、Ruby、Tcl 等语言的绑定，能在 Windows、Linux、Solaris、MacOsX 等平台上运行。

引入 libxml2 库文件，并对文件的名字，IP 地址，包文件等关键地方进行转换。下面说明本模块的主要实现部分：

```
//定义主要的结构体
typedef struct configuration {
   char * xml_filename;                  // 指向 XML-PDML 文件指针
   xmlDocPtr doc;                        // 包含树结构体指针
   xmlNodePtr cur;                       // 指向单一节点的指针
   unsigned int packet_counter;          // 抓到的包的数目
   unsigned int proto_counter;           // 协议种类数目
```

```
    unsigned int transport_type;          //  传输协议的类型;tcp = 1; udp = 2;
    int invert;                          //  包定义  packet > send = 0; second packet > send = 1
    unsigned int ip_client;                 //客户端的 IP 地址·
    unsigned short port_client;            //  客户端的端口号
    unsigned int ip_server;                 //  服务器端的 IP 地址·
    unsigned short port_server;            //  服务器端的端口号
    unsigned int ip_pkt;                     //当前数据包的 IP 地址
    unsigned short port_pkt;                //当前数据包的端口号
    unsigned int send;            // send＝1 表述发送数据，否则接收数据
    ……
} config;
//解析协议数据包
void xml_parse_packet(config *conf) {
    定义 xmlChar 指针  *name;
    保存副本 cur = conf->cur;
    进入数据包 cur = cur->xmlChildrenNode;
    初始化协议计数器 conf->proto_counter = 1;
      while  循环(cur  不为空) {
      如果  (名称= "proto") {
        获取协议数据包名称；
       //判断是否为 IP 类型
         if (!strncmp(name, "ip", strlen(name))) {
       解析 IP 信息 xml_parse_ip(conf, cur);
        }
        否则为 TCP 类型：{
                          保存端口信息
       }
        如果是 UDP 类型{
```

保存端口信息

……

}

否则为的数据包格式{

//解析协议包内容 xml_parse_proto(conf, cur);

}

协议类型指针++;

}

解析原始数据;

进入下一个数据包 cur = cur->next;

}

}

(2)ADC

ADC 可以检测脚本的正确性，即是可以用 ADC 来编译脚本。对那些比较复杂的协议而言，测试者会人为的添加 Fuzzing 数据，难免会有错，ADC 可以检查一般的语法错误。

4.4.2 跟踪参数权值模块设计

本块是整个模块的核心部分之一，目的是实现减少 Fuzzing 数据的输入，降低 Fuzzing 漏洞挖掘的复杂度。主要的思想是通过衡量输入数据被目标程序的不安全函数调用的情况，参数被危险函数调用的次数越多，那么这个参数的权值就会相应的增加。没有被调用的参数就没有权值。

初始化：在 strcpy,strcat,gets,sprintf,getenv,printf,syslog 等容易导致溢出的地方设置断点，然后建立和 Fuzzer 之间的连接通信。最后处于调试等待状态。

用到的数据结构：

//装载危险函数的相信信息

struct struct_bp {

链表指针* next;

```
    断点的号 id;
    ESP 的类型如. 0-> addr, 1-> string
    有用值和 ESP 的距离 esp e.g. strcpy = 0x8 esp;
    函数的名称  *name;
    函数的地址  addr;
} bp;
//marker 输入参数的详细信息
struct struct_string {
    链表  next; /
      fuzzer 传递过来的 marker 字符串的 id;                    /* */
    包含一字符串的缓冲*string;          /*区*/
} string;
//configure 结构体
typedef struct configuration {
    被调试跟踪进程的 pid
    gdb 执行字符串  *gdb_exe;
    输入参数的文件  *input;
    输入文件  *f_input;
    指向程序名字的指针  *program;
    gdb dump  文件  *f_gdb_dump;
    gdb dump  文件名称  *gdb_dump;
    string dump  文件  *f_string_dump;
    string dump 文件名称  *string_dump;
    指向程序参数的指针  *args;
    总端主机 master;
    总端客户端 slave;
    包含 gdb 输出文件的缓冲区  * gdb_buf;
    断点链表  *bp;
```

Marker 输出参数链表 *string;

和 Fuzzer 连接端口 port;

} config 机构体;

init_configuration()进行初始化操作

gdb_break()对目标程序的危险函数设置断点，并对指定的函数指定危险系数，同时将这些函数设置的断点以链表的形式保存下来。

4.4.3 测试数据包模块的实现

测试数据包模块是整个系统的核心模块，功能是解释 AD 脚本文件，从脚本中读取协议样本内容；分别对协议数据包中的若干数据项进行变换，构造测试数据包，并将数据包发送到目标程序端。

测试数据包模块的四个主要组成部分：脚本处理、数据项变换、和调试器通信、数据包构造及发送，如图 4-4 所示。

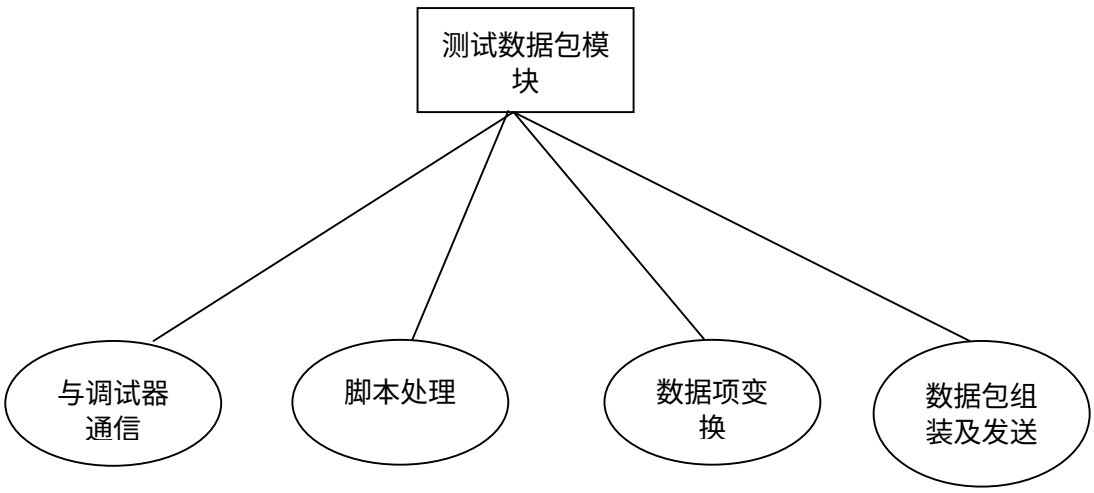


图 4-4 测试数据包模块构成

(1)与调试器通信连接

与调试器建立连接并进行实时的通信，首先将测试数据发送给调试器，调试器将测试数据发送到目标程序并监测它们被使用的情况，对每个被危险函数调用的测试数据增加相应的权值，并实时地发送给挖掘器模块。

相应的处理函数原型声明为：

int dbg_connection(config *conf)

int dbg_send_msg(config *conf, unsigned int type, unsigned int id, void *string, unsigned int size);

int dbg_read_msg(config *conf);

函数 dbg_connection 中，输入参数 conf 是整个挖掘器的核心数据结构，包括相关的所有信息，函数返回 0 表示连接成功，-1 表示失败。函数 dbg_send_msg 是用来接收来自于调试器的信息；函数 dbg_read_msg 用来发送信息到调试器。连接函数通过 conf 结构体提供的信息包括调试器的 IP 地址和端口号和调试器建立连接。对应的 conf 结构体内关于连接的信息：

typedef struct configuration {

fuzzer 的类型: 0=客户端 (默认) mode

协议类型： 0 = tcp; 1 = udp;

开始连接前的等待时间(秒);

调试器的主机名(IP 地址);

与调试器建立连接的端口号;

dbg socket 描述;

用于连接的主机名;

……;

} config 结构体;

建立连接后，就可以和调试器进行通信，发送和接收相应的数据。

(2)脚本处理

脚本处理是指从 AD 格式文件中读取协议数据包。相应的处理函数原型声明为：

unsigned int read_adc_file(config *conf, char *filename)

函数 read_adc_file 的参数为 config *,和文件名指针。函数的功能是读取 AD 文件，并将其内容拷贝一份到内存中，以备分析。Filename 表示函数名称，对于参数 conf，下面表示其内的关于脚本处理的相关数据结构：

//adc 文件结构

```
 struct struct_adc {
   缓冲区地址;
   开始读取缓冲区数据的地址;
   缓冲区的偏移;
   缓冲区的大小;
};
//核心数据结构
typedef struct configuration {
   指向当前 fuzz 缓冲区的指针;
   当前缓冲区 fuzz 文件的大(拷贝的 ad 文件
   adc  文件指针
   ……
} config;
```

(3)数据项变换

发送数据并挖掘软件漏洞

本模块是整个系统的核心部分，采用了故障注入的方法，发送经过数据项变换后的数据到目标软件，用以挖掘热软件的漏洞。主要用到函数和数据结构如下：

```
struct struct_block {
   unsigned int id;                  /* block  的 id 号*/
   unsigned int size;                /* block 的大小*/
   unsigned int offset;             /*在包中的偏移*/
   unsigned int state;               /* 0:  开启  1:  关闭  */
   struct struct_block *next; /* block 链表*/
};
int send_fuzz(config *conf, struct struct_block *block) {
   unsigned int bytes;
      bytes = sendto(conf->socket,
            conf->buf_fuzz + block->offset,
            block->size,
```

```
            0,
            (struct sockaddr *)&(server),
            sizeof(server));
        if (bytes == -1) return -1;
         /*      与调试器的连接通信，等待 DBT_TIMEOUT 信号*/
        if (conf->dbg_mode) {
           fd_set rfds;
           struct timeval tv;
           int retval;

           tv.tv_sec = DBG_TIMEOUT_SEC;
           tv.tv_usec = DBG_TIMEOUT_USEC;
           FD_ZERO(&rfds);
           FD_SET(conf->dbg_socket, &rfds);
           retval = select(conf->dbg_socket+1, &rfds, NULL, NULL, &tv);
           /* 错误退出 */
           if (retval == -1) {
          return -1;
           }

           /*dbg 打印漏洞信息 */
           if (retval) {
 if (dbg_read_msg(conf)) bytes = -1;
           }
        }
     }
      return bytes;
}
```

4.5 小结

基于 Fuzzing 的网络协议漏洞挖掘系统方案具有自身的优点和缺点, 优点是带有自身的调试器，可以对参数进行权值的衡量，这样可以很大限度的减少参数的输入，降低复杂度。另外实现了测试脚本的半自动化的撰写，减少了人工撰写脚本的麻烦。

缺点是考虑到效率和安全性的问题，整个过程的开发是基于 linux 平台的。缺乏 windows 平台下面的支持，没有友好的图形界面，使得其应用具有一定的局限性和不方便性。

5 实验与仿真

本章主要讲解漏洞挖掘器的应用，漏洞挖掘器包括几个部分：协议解析及脚本撰写(PD2AD)、参数调试跟踪(ADBG)和构造并发现测试数据(autoFuzzer)，下面分别介绍这几个工具的使用方法，并对测试结果进行分析，得出漏洞挖掘器的优点。

5.1 测试用例

5.1.1 介绍

基于网络协议的漏洞挖掘，被测试的目标软件为可以用来远程控制计算机。由于 Fuzzing 技术本身也是黑盒子测试技术，这在前文中有介绍，因此，不需要目标程序的源代码就可以进行漏洞挖掘。

vuln2 是一个服务程序工具用到了基于协议的传输。一般的基于协议格式的漏洞挖掘，进行的首要步骤是对目标协议格式进行分析，然后基于协议格式写出相应的数据包进行 Fuzzing。利用本文描述的工具 PD2AD 和 ADC 可以很方便的实现协议脚本的撰写，免去繁琐的人工分析协议和撰写测试脚本。 最后用调试器 ADBG 和 fuzz 引擎 autoFuzzer 对 vuln2 进行 Fuzzing 测试。

5.1.2 测试流程

(1) PD2AD

通常的解析协议的方法是利用抓包工具在客户端和服务器端的通信过程中抓取数据包，然后对其进行分析。当前比较实用的抓包工具很多，可以选择 Wireshark，Wireshark 到目前为止的最高版本是 1.0，可以识别 930 种以上的协议格式。Wireshark 抓包并保存为 PDML 文件，如图 5-1 所示。但是 vlun2 所用到的协议 Wireshark 却不能识别。利用 PD2AD 可以将 PDML 文件直接转换为 AD 文件。

对目标程序的 pdml 文件进行查看，如图 5-2 所示。

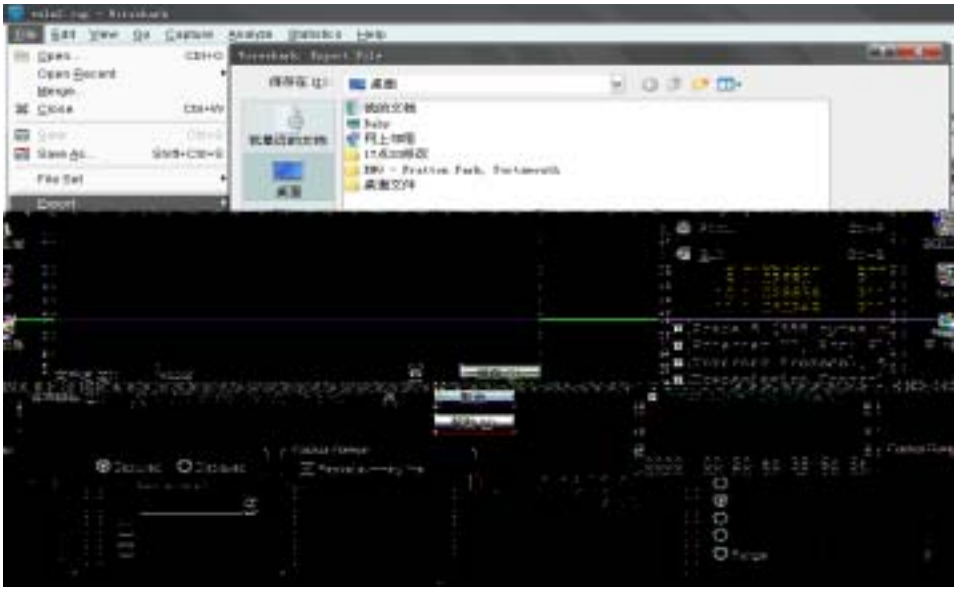


图 5-1 利用 Wireshark 抓包并保存为 PDML 文件

```
[root@hust dummy]# cat ./vuln2.pdml
<?xml version="1.0"?>
<pdml version="0" creator="ethereal/0.10.6">
<packet>
  <proto name="geninfo" pos="0" showname="General information" size="81">
    <field name="num" pos="0" show="1" showname="Number" value="1" size="81"/>
    <field name="len" pos="0" show="81" showname="Packet Length" value="51" size
="81"/>
    <field name="caplen" pos="0" show="81" showname="Captured Length" value="51"
 size="81"/>
    <field name="timestamp" pos="0" show="Sep 25, 2004 15:59:22.350469000" shown
ame="Captured Time" value="1096131562.350469000" size="81"/>
  </proto>
  <proto name="frame" showname="Frame 1 (81 bytes on wire, 81 bytes captured)" s
ize="81" pos="0">
    <field name="frame.marked" showname="Frame is marked: False" hide="yes" size
="0" pos="0" show="0"/>
    <field name="frame.time" showname="Arrival Time: Sep 25, 2004 15:59:22.35046
9000" size="0" pos="0" show="Sep 25, 2004 15:59:22.350469000"/>
    <field name="frame.time_delta" showname="Time delta from previous packet: 0.
000000000 seconds" size="0" pos="0" show="0.000000000"/>
    <field name="frame.time_relative" showname="Time since reference or first fr
ame: 0.000000000 seconds" size="0" pos="0" show="0.000000000"/>
```

图 5-2 vuln2.pdml 文件内容

基于语法解析和 Block-based 协议描述语言而开发的 PD2AD 工具可以很方便的将 pdml 格式转换为 autoFuzzer 可识别的测试脚本。这种脚本和 SPIKE 框架脚本很相似。

将 vuln2.pdml 文件格式转换为 myvuln2.ad 格式文件,如图 5-3 所示。

可以看出这不是字符串，因为此协议不是基于 ASCII 字符的。但是末尾的\0X 可以看出协议的某一个部份是基于字符串的。由于 Wireshark 不能识别的协议，首先

要理解 AutoFuzzer 所用协议脚本描述语言。这是来源域 Dave Aitel 的创意，基于 Block-Based 描述语言。将协议分为块，并把块划分为数据域和长度域，域的大小在其数据被添加后被动态的更新。

```
[root@hust dummy]# pdml2ad -s ./vuln2.pdml ./vuln2.ad
[root@hust dummy]# cat ./vuln2.ad
block_begin("packet_1");
hex(
51 55 45 53 00 00 00 03    41 41 41 45 4e 44 0a
);
block_end("packet_1");
recv("packet_1");  /* tcp */

block_begin("packet_2");
block_end("packet_2");
send("packet_2");  /* tcp */
```

图 5-3 PDML2AD

由图 5-3 的内容可知，最后的一块"packet_2" 由于是空的，所用没有用处，可以手动去掉.，并添加 Fuzzing 字符串内容。最后用 AutoFuzzer 协议脚本描述语言描述的结果如下：

```
block_begin("packet_1");
string("QUES");//固定值
block_size_b32("string_1"); /* big endian 32 bits size */
block_begin("string_1");
fuzz_string("AAA");//fuzzing 数据
block_end("string_1");
string("END");
hex(0a); /* \n */
block_end("packet_1");
send("packet_1");    /* tcp */
```

这里用函数“ fuzz_string ”来替换“ string ”，可以达到替换 fuzz 的目的，这里 fuzz 的字符串是“ AAA"。除此外的别的字符串或值将不会被 fuzzer 所替换。如果目标协议是可以被 wireshark 识别的，那么就可以利用 PDML2AD 直径进行转换。

(2) ADC

ADC 可以检查人工撰写的脚本可能存在的错误，还将 ad 文件 转换为可以直接被 fuzzer(autoFuzzer)使用转换为 adc 文件情况如图 5-4 所示。

```
[root@hust dummy]# adc ./bad-vuln2.ad
[YY-ERROR] block: "string_1" has not been ended
[!] block: "string_1" size: 7 (0x7)
[!] block: "packet_1" size: 15 (0xf)
[YY-ERROR] at line[20]: See block(s) error(s) above
[root@hust dummy]#
```

图 5-4 adc 检测到错误语法

在描述了协议并有了相应的测试脚本后，下一步调用 ADBG.。

(3) ADBG.

ADBG 利用 gdb 来跟踪程序的运行，并衡量输入参数的权值。ADBG 将检测哪些危险函数被目标程序调用，并对每一个 fuzzed 变量基于其被危险函数调用的次数给出一定的权值，这些参数包括：fuzz_string，fuzz_hex 等。调试跟踪器 ADBG 将监听本地端口号，为了实时的和 fuzzer 建立通信连接，如图 5-5 所示。

```
[root@hust dummy]# adbg -v -p 31337 ./vuln2 2000
[*]-+-> breakpoint on: "strcpy"
[*]-+-> breakpoint on: "strcat"
[*]-+-> breakpoint on: "gets"
[*]-+-> breakpoint on: "sprintf"
[*]-+-> breakpoint on: "getenv"
[*]-+-> breakpoint on: "stpcpy"
[*]-+-> breakpoint on: "printf"
[*]-+-> breakpoint on: "syslog"
[*] waiting for a connection on port: 31337
```

图 5-5 adbg

图 5-6 显示了 adbg 检测程序 vuln2，并在那些危险函数地方设置的断点，同时监听 31337 端口，等待和 fuzzer 建立连接。

(4) Fuzzer

Fuzzer 是 fuzz 的核心引擎，用来解析 ADC 文件并发送测试数据到目标程序：

1)将对目标协议的描述(利用 block-based 协议描述语言)传给 Fuzzer。每一个规范的元素都将被看作 marker。在本例种 marker 是“ AAA ”

2)fuzzer 利用这个描述（包含在 adc 文件中）建立和和目标程序间的正常连接

3)调试跟踪器 adbg 收到来自 fuzzer 的 marker 参数列表，并将它们运行在目标程序上。Marker 只是一些测试者想测试的字符串而已。在本例中只有一个 marker："AAA".

4)调试跟踪器分析目标程序的运行情况，检测出哪些危险函数在调用 markers。

5)如果一个 marker 被一个危险函数调用，那么调试跟踪器将为此 marker 增加一个更大的权值，并将结果传递到 fuzzer。

6)根据 marker 权值的大小，fuzzer 选择哪些 merker 会被用于测试。哪些没有权值的 marker 将不会被用于第一轮的 fuzzing 测试。

7)如果一个变量激发了一个缓冲区溢出，调试跟踪器将给出此漏洞的信息。

开启 autoFuzzer 程序如图 5-6，5-7 所示。

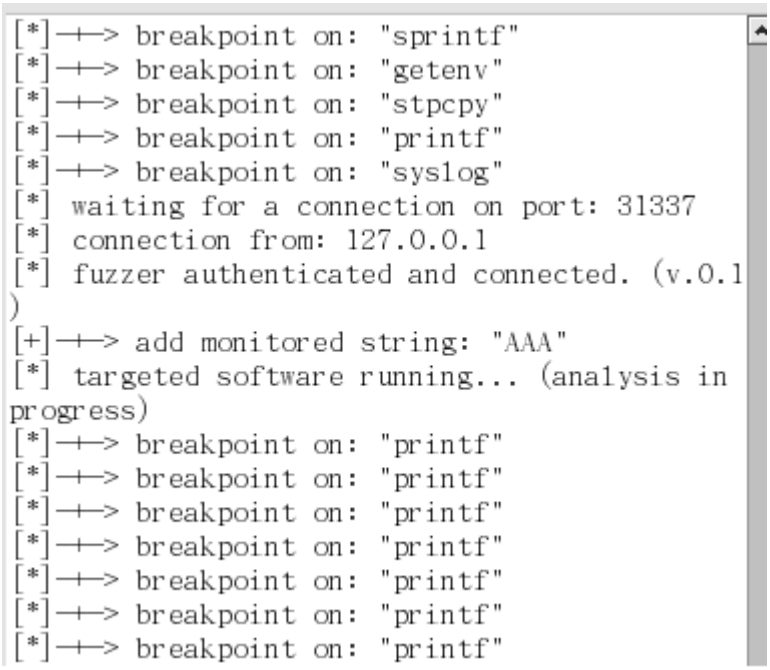


图 5-6 adbg 运行

可以识别第一个数据包包，fuzzer 为了建立和目标程序的连接：

QUES<size of the string "AAA" in 32-bit big-endian>"AAA"END

图 5-7 的文件组成了替换 fuzz 库文件。每一个 fuzzed 字符串(本例子的字符串是“ AAA ”)将被这些文件所替换。第一个文件被称为"string-1-x3"，它包含了 3-byte 字符串 "AAA". 和原始字符串一样的，本例中没有用处。

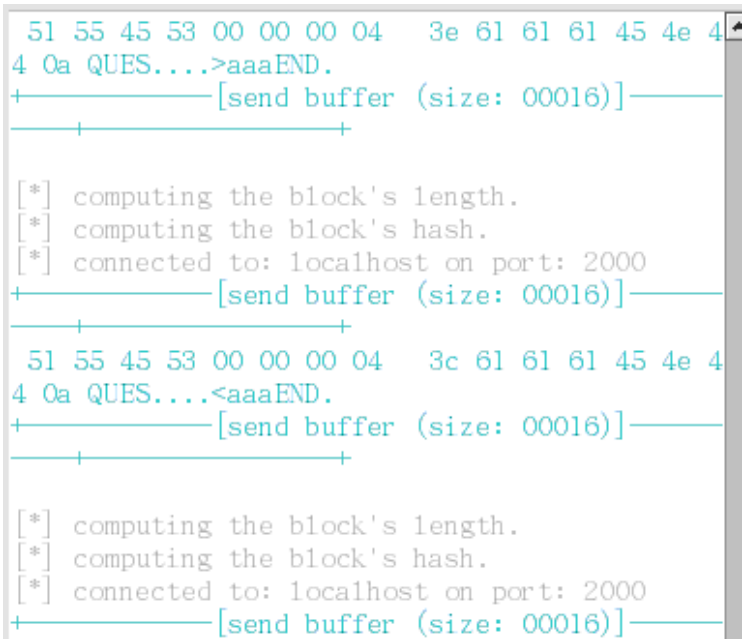


图 5-7 autoFuzzer 运行

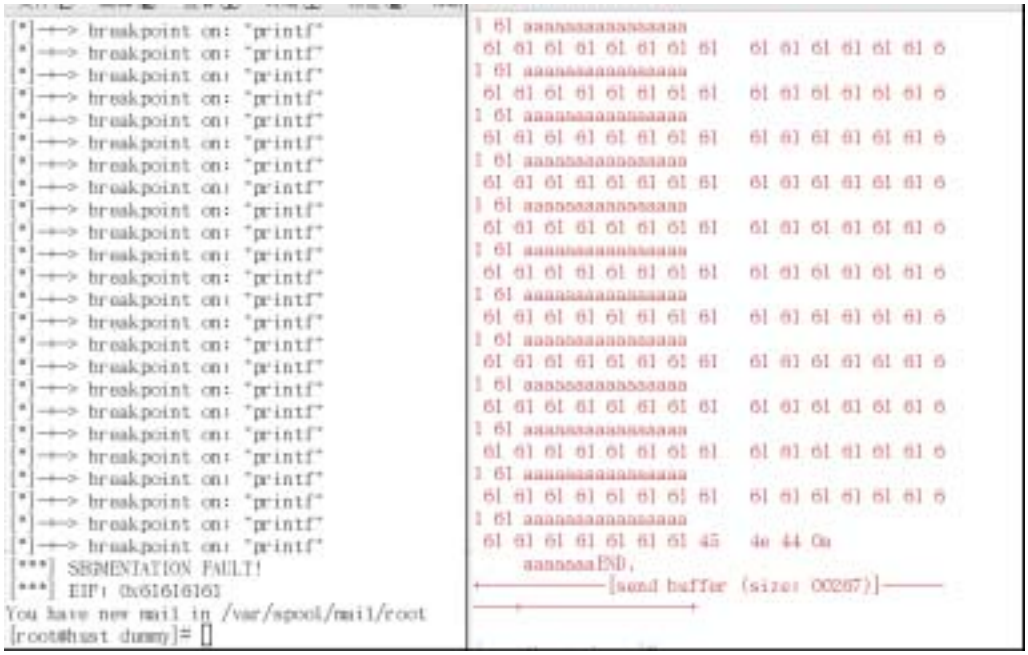


图 5-9 ADBG 提示漏洞信息

当"AAA"被替换为 16-byte 长度的字符串 "AAAAAAAAAAAAAAAA"时候.可以看到长度域值被改变了：00 00 00 10，在 32bit big endian 中是 16. 可以看到，当字符串被长 255byte 字符串替换的时候，调试跟踪器显现了消息如图 5-8 所示。

查看 adgb 调试跟踪器的输出文件：

[***] SEGMENTATION FAULT!

[***] EIP: 0x41414141

OK，发现了基本的栈溢出漏洞。

5.2 实验结果分析

软件名称：vuln2

影响版本：1.11

漏洞描述：vuln2 可以用非法获取远程计算机的权限。通过测试发现，vuln2 在执行远程代码时存在缓冲区溢出漏洞。

漏洞调试：用 Vuln2 执行远程代码当参数信息 AAA□AA 超过 2678 B 的长度，就会造成程序崩溃，当本字段长度为 2 212 B 时，EIP 被填充的字符串覆盖，精心构造字符串可以在本机执行攻击代码。

5.3 小结

通过上面的测试用例，可以看出本文提出的基于 Fuzzing 的网络协议漏洞挖掘技术的可行行与实用性。脚本的撰写，Fuzzing 数据的选择以及 Fuzz 引擎的设计等都体现了自动化的思想，对信息安全研究工作者来说，可以提高他们的工作效率。

6 总结与展望

随着通讯和计算机技术的迅猛发展，计算机网络向世界各个角落延伸，人们通过网络互联享受着科技带来的巨大便利。通过网络，拓展了信息资源共享时间和空间，提高了利用率。在得益于计算机互联网络所带来的新的巨大机遇的同时，也充分感受到信息安全面临的严峻考验。

6.1 工作总结

本文是“漏洞挖掘技术研究”中的一部分。通过研究传统的漏洞挖掘技术的缺陷，针对软件领域基于网络协议格式的漏洞占主导的现实情况下，提出了新的漏洞挖掘技术，主要做了以下工作：

(1)深入研究了传统的漏洞挖掘技术：静态检测技术和动态分析技术，对漏洞挖掘技术的实用性、可行性进行了探讨，得出了传统漏洞挖掘技术的缺陷，提出了需要改进的地方。

(2)采用了基于 Block-Based 协议描述语言对网络协议数据包进行分析，并自动生成测试脚本；建立了 Fuzzing 测试库文件，可以实现 Fuzzing 测试数据的自动化生成；提出了基于权值衡量的 Fuzzing 技术，采用调试器和 Fuzzer 连接的方式对 Marker 参数值进行跟踪，达到了减少参数输入空间的目的。以上几点最终形成了基于 Fuzzing 的网络协议漏洞挖掘技术。

(3)在基于 Fuzzing 的网络协议漏洞挖掘技术可行性原理的基础上，实现了 Fuzzer 漏洞挖掘器，通过采用本文实现的漏洞挖掘器对软件进行漏洞挖掘测试，证明了本文提出的参数权值衡量的技术具有一定的先进性和很高的实用性。

6.2 工作展望

本文还有一些工作需要进一步的完善，在挖掘器方便性方面需要拓展到 Windows 平台下，毕竟 Windows 平台下的漏洞挖掘工作也是整个漏洞技术研究的重要组成部分，同时图形化界面也是需要的，这样可以方便安全研究工作者的使用。

另外作为漏洞技术研究的重要组成部分之一的漏洞信息分析技术在本文中没有得到很好的研究，这是需要解决的重点之一。

除以上提出的需要拓展的内容外，基于 Fuzzing 技术在漏洞挖掘研究方面的实用性和高效性，对 Fuzzing 技术的深入研究也是必须要继续。最后，需要进一步研究漏洞挖掘技术，建立起一套更科学、实用性更高的漏洞挖掘技术体系。深入研究本文提出的研究方案，提高漏洞挖掘软件的利用率和实用性。

致 谢

首先，我由衷的感谢我的导师崔国华教授，实验室的付才老师，徐鹏师兄。他们在我毕业学习期间为我提供了充分的发展空间和研究氛围，使我有机会直接参与高水平的科研项目，掌握相关研究领域的最新科研动态。从开始研究生阶段的学习以来，崔国华教授始终给予我悉心的指导，并以他超前敏锐的洞察力，创造性的思维，渊博的知识，严谨的治学态度，卓越的管理才能以及对科技发展方向的把握给我以熏陶。崔国华教授将是我今后在科研学习生活中的榜样，在此谨向崔国华教授致以崇高的敬意和深深的谢意。

同时感谢计算机学院信息安全实验室其他老师和同学在学习工作中予以了我的很多关心和帮助，这里一并向他们表示诚挚的谢意。

最后，向本论文引用过的文献作者们表示谢意，向百忙中抽出时间评议、审阅论文的各位尊敬的老师表示感谢。

参考文献

[1] B.P.Miller,L.Fredriksen.An Empirical Study of the Reliability of UNIX Utilities.Communications of the ACM 33,December 1990.32~44

[2] 马杰.也谈“熊猫烧香”.程序员,2007 年,第 3 期:135~135

[3] B. Hebbard, P. Grosso, T. Baldridge et al. A Penetration Analysis of the Michigan Terminal System. Operating Systems Review.1980.14(1):7~20

[4] Marick B.A Survey of Software Fault Survys.University of Illinois at Urbana-Champaign,1990.14~18

[5] 曹军.Windows 危急级漏洞挖掘及分析技术研究：[硕士论文].四川大学,2006.6~10

[6] 罗景,赵伟,秦涛等.基于有向带权图迭代的面向对象系统分解方法.软件学报,2004 年,15(9) :1292~1300

[7] 曾鸣,赵荣彩,姚京松等.一种基于反汇编技术的二进制补丁分析方法.计算机科学,2006 年,第 10 期:283~287

[8] John Wiley,Hoboken ,New Jersey.Art of Software Testing.Second Edition.USA:Word Association.2004.9~11

[9] Anup K.Ghosh,Tom O,Connor,&Gary McGraw.An Automated Approach for Identifying Protential Vulnerabilities in Sofeware. IEEE Transactions On Information Theory.Phrack.2002.104~114

[10] B.P.Miller,D.Koski,C.P.Lee,V.Maganty,R.Mruthy,A.Natarajan,and J Steidl. Fuzz Revisited:A Re-examination of the Reliability of UNIX Utilities and Services[R]. Technical report,CS-TR -95-1268,Computer Sciences Department, University of Wisconsin,1995.1~23

[11] J.E.Forrester and B.P.Miller.An Empirical Study of the Robustness of

Windows NT Applications Using Random Testing.4th USENIX Windows Systems Symposium. Washington, USA ,Seattle, August 2000.

[12] B.P.Miller,G..Cooksey and F.Moore.An Empirical Study of the Robustness of MacOS Applications Using Random Testing.ACM New York,NY,USA. Volume 41,Issue 1(January 2007):78~86

[13] Michael Sutton,Adam Greene,Pddram Amini.Fuzzing Brute Force Vulnerability Discovery.USA:Pearson Education,Inc,2007. 236~242

[14] J.Viega,J.T.Bloch,Y,Kohno,G.McGraw.ITS4:A static vulnerability scanner for C and C++ code. Computer Security Applications, 2000. ACSAC '00. 16th Annual Conference .USA,IEEE Computer Society Press, 2001.257~267

[15] Brian V.Chess.Improving Computer Security Using Extended Static Checking. Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on.UEA,IEEE Computer Society Press, March 2006.160~173

[16] David Larochelle,David Evans.Statically Detecting Likely Buffer Overflow Vulnerabilities.Proceedings of the 10th USENIC Security Symposium-Volume 10.USA,USENIC Press,2001.14~14

[17] Brandon Bray,Compiler Security Checks In Depth,Microsoft Corporation,2006.

[18] 屈晔,张昊.Bugscam 自动化静态漏洞检测的分析.信息安全与通信保密,2006 年,24(4):41~45

[19] JEFF.Kill bugs Source-code assessment tools identify security holes during shoddy development. Applications and Stardards . Pearson Publication,October,2005.5~8

[20] Mike Van Emmerik,Trent Waddington.Using a Decompiler for Real-World Source Recovery.Reverse Engineering,2004.Proceedings.11th Working Conference.IEEE,2004.27~36.

[21] 赵巾帼,罗庆云.程序调试运行时的错误及检测方法.网络安全技术与应

用,2006 年,04 期:46~47

[22] 许治坤,王 伟,郭添森. 网络渗透技术.第一版.北京:电子工业出版社,2005 .260~276

[23] 张玉清,戴祖锋,谢崇斌.安全扫描技术.北京:清华大学出版社,2004,126~136

[24] William Stallings. Network Security Essentials,Applications and Stardards . Pearson Publication,July,2004 .23~25

[25] 冰雪黑鹰,三剑合壁.见招拆招——W32Dasmv、UltraEdit 及 CodeFusion Wizard 综合使用手记.软件指南,2005 年 7 期,69~71

[26] Funnwyei.缓冲区溢出漏洞发掘模型.XfocusXCon,2003.12(5):8~10

[27] Halver Flake.Automated Software Security Analysis.Xfocus.2004,20(7)12~15

[28] Brief.缓冲区溢出漏洞发掘之整数范围限制浅析.XfocusXCon.2003,13(2)20~24

[29] [美]史蒂文斯(Stevens,W.R.)著, 胡谷雨等译.TCP/IP 详解卷 3：TCP 事务协议、HTTP、NNTP 和 UNIX 域协议.北京:机械工业出版社, 2000 年 9 月第 1 版.120-134

[30] R.Srinivasan.RPC-Remote Procedure Call Protocol Specification Version 2(Sun version).IEEE, ISO/IEC CD 11578 N6561, ISO/IEC, November 1991.24~32

[31] J.Postel,J.Reynolds.Telnet Protocol Specification. Obsoletes:NIC 18639, May 1983.4~14

[32] J.Postel,J.Reynolds.File Transfer Protocol.Obsoletes:NFC 765,October 1985.35~40

[33] Jonathan B.Postel. Simple Mail Transfer Protocol.Information Sciences Institute University Publication,August 1982.19~42

[34] Duffy,Robert A.POP3 mail client using winsock.Dr.Dobb's Journal, 1995,12(2):16~25

[35] Reed KD.协议分析[M].第 7 版.孙坦,张学锋,杨琳等译。北京：电子工业出版社,2005.51~85

[36] 王丽苹,孙蕾.基于 Ethereal 开源代码构建协议解析器的方法研究.计算机技术与发展,17(10) :27~30

[37] 王石.局域网安全与攻防--基于 Sniffer Pro 实现.第一版.北京:电子工业出版社,2006.123~136

[38] 王远.Windows 系统 API 函数拦截技术研究.微计算机信息.2006 年第 22 卷第 10-3 期:224~226

[39] 潘琦,王澄,杨宇航.堆栈溢出攻击的分析及防范.上海交通大学学报.2002,36(9):1346~1350

[40] Michael Stton,Adam Greene,Pedram Amini.Fuzzing Brute Force Vulnerability Discovery. First Edition USA:Addison-Wesley Professional, 2007.169~179

[41] 朱鸿宇,谢余强,刘瑰.基于故障注入发现缓冲区溢出漏洞的研究.微计算机应用,2005 年 06 期.676~678

[42] Dave Aitel. The advantages of block-based protocol analysis for security testing.IEEE, 2002.3~9

[43] 余翔湛 ,殷 丽 华.动态共享内存缓冲池技术.哈尔滨工业大学学报,2004 年,36(3) :383~383

[44] D.P IPER D. Wireshark (Ethereal) PDML to Sequence Diagram Conversion Tutorial. RFC2407. 1998, 16~23

[45] 杜华.Linux 编程技术详解.第一版.北京:人民邮电出版社,2007.84~97

基于模糊的网络协议漏洞挖掘技术研究

作者： 余洪航
学位授予单位： 华中科技大学