# 网络协议的自动化模糊测试漏洞挖掘方法

李伟明 张爱芳 刘建财 李之棠
(华中科技大学计算机学院 武汉 430074)

**摘 要** 随着网络应用越来越复杂和重要,对网络协议的安全性要求也越来越高.模糊测试(Fuzz Testing)作为一种重要的测试手段,通过大量数据的注入来测试网络协议的安全,能够发现拒绝服务、缓冲区溢出和格式化字符串等多种重要漏洞.但是手工进行 Fuzz Testing 需要精确了解网络协议细节并需要繁重的工作来构造大量测试数据集,导致覆盖率有限,效果也不好.为了解决这个问题,文中结合网络协议逆向工程和 Fuzz Testing 技术,提出了一种能够自动化识别各种网络协议并产生模糊器进行 Fuzz Testing 的漏洞挖掘方法.这种方法通过报文分类、多序列比对、特定域识别、模糊器生成多个阶段,自动识别网络协议报文结构并进行 Fuzz Testing.经过测试 FTP、TNS、EM、ISQLPlus 等多种已知和未知网络协议,结果表明这种方法在漏洞挖掘方面效果超过手工分析,并显著提升了测试效率,为提高网络协议的安全性提供了良好的基础,具有重要的应用价值.

**关键词** 协议逆向工程;模糊测试;漏洞挖掘
**中图法分类号** TP393 **DOI 号**: 10.3724/SP.J.1016.2011.00242

# An Automatic Network Protocol Fuzz Testing and Vulnerability Discovering Method

LI Wei-Ming ZHANG Ai-Fang LIU Jian-Cai LI Zhi-Tang
(*School of Computer Science, Huazhong University of Science and Technology, Wuhan* 430074)

**Abstract** Along with the increasing complexity of the network application, network protocol security is now become more and more important. Fuzz Testing often is used to discover DoS, buffer overflow, format string and other kinds of serious vulnerabilities of network protocols. But manually Fuzz Testing is very low effective and need adequate detail information about the protocols. The paper presents an automatic vulnerability discovering method which combines automatic Protocol Reverse Engineering technology and Fuzz Testing. The method is a four steps program, involving packets clustering, multiple sequences alignment, special fields recognition and fuzzer production, which find the structure of network packets and pursue Fuzz Testing. After testing FTP, TNS, EM and ISQLPlus protocols, the results show that this method is more effective and accurate than manually analysis. The method is of the important application value and can improve the security of network protocols.

**Keywords** protocol reverse engineering; fuzz testing; vulnerability discovering

## 1 引 言

网络协议只有安全的设计和实现才能保护用户传输的敏感信息.但是如何测试网络协议安全性一直是个非常困难的问题.特别是测试未公开网络协议(closed protocols)的安全性对于网络安全有着重要的意义,因为大量厂商并没有公开自己软件系统

收稿日期:2010-04-16;最终修改稿收到日期:2010-12-15.本课题得到国家自然科学基金(60573120)资助.**李伟明**,男,1975 年生,博士,讲师,主要研究方向为网络安全.E-mail: lwm@hust.edu.cn.**张爱芳**,女,1978 年生,讲师,主要研究方向为网络安全.**刘建财**,男,1985 年生,硕士研究生,主要研究方向为网络安全.**李之棠**,男,1951 年生,博士,教授,博士生导师,主要研究领域为系统结构、网络安全.

使用的网络协议细节，例如 MicroSoft 的网络文件共享 SMB 协议、Oracle 数据库访问的 TNS 协议、各种 IPTV 和及时通信软件使用的协议. 这些未公开网络协议在网络中被大量使用，一旦其未公开漏洞被攻击者利用，将产生巨大的危害. 因此，本文提出了一种自动化识别各种网络协议报文结构并进行模糊检测(Fuzz Testing)的方法，能够找到协议中拒绝服务、缓冲区溢出和格式化字符串等多种重要漏洞，既可以避免手工测试带来的繁重工作和低效率，又能深入测试网络协议的安全性.

本文第 2 节介绍网络逆向工程以及模糊检测漏洞挖掘方法的研究现状，并提出本文的总体思路和实现结构；第 3 节介绍数据报文分类方法；第 4 节详细介绍如何对未知网络协议进行识别，并对采用的多序列比对中的渐进比对和遗传算法进行比较，提出改进的渐进比对算法和优化的 Needleman Wunsch 算法；第 5 节则进一步介绍如何自动化识别网络协议的特定域；第 6 节描述如何将识别出的网络协议转化为 Fuzzer，构造针对目标协议的模糊测试器；第 7 节是实验测试，通过具体数据分析自动化漏洞挖掘方法的有效性；第 8 节对工作进行总结和展望.

## 2　介　绍

在网络协议逆向工程领域，国内外进行了较为深入的研究，主要分为两个大的方向：一个方向是单纯利用网络流量来推测网络协议，这类方法也称为基于“Network Trace”的方法；另外一个方向是对实现网络协议的服务器程序进行二进制的动态跟踪分析，通过跟踪二进制文件对报文的处理流程对报文进行解析，这种方法也称为基于“Tainted Data”的方法.

基于“Network Trace”的方法，典型的有 2004 年由 Marshall Beddoe 启动的“Protocol Informatics project”分析未知或者未公开的网络协议的结构①. 这个项目的目标是类比生物学从 DNA 中查找产生氨基酸的蛋白质的算法，通过对大量网络流量进行对比分析，最终分析协议结构. 具体方法是首先读取目标协议的抓包文件形成序列集. 然后通过局部序列比对算法计算这些序列之间的相对距离，形成距离矩阵. 接着使用非加权成对群算术平均法(unweighted pair group method with arithmetic mean)，构造一棵向导树，实现序列的聚类，每一类中的序列被认为是具有很大相似性. 最后，针对每一个聚类，利用渐进比对算法遍历系统树，完成多序列比对. 根据比对结果，就可以区分固定不变的区域和动态改变的区域. 但是这种方法仅仅提取了协议中的可变和不可变域，划分粒度很粗，同时分析结论也对流量的依赖性很强. 另外，其聚类方法以 LCS(Longest Common Subsequence)匹配数量作为参考标准，因此无法区分和提取序列中相对简短的协议域. 总之 PI 项目只是处于初步阶段，算法还很不成熟.

2005 年 Corrado Leita、Ken Mermoud 和 Marc Dacier 提出使用 PI 项目改造 Honeyd，实现 Honeyd 配置脚本的自动提取[1]. 文章的主要设计思路由 4 个步骤组成：首先从网络抓包文件中获取数据包的 Message 序列，然后分析这些序列，为每个请求维护一个状态机，记录客户端和服务器之间的信息交互过程. 然而，此时的状态机没有进行任何的语义分析，不能完成实际的交互动作，因此需要优化状态机，最后完成脚本的自动生成. 同样 Cui 和 Paxson 等人提出了一种网络协议识别和自动恢复的解决方案 RolePlayer[2]. 其设计思路是，在获取用户输入参数的条件下，通过分析少量样例数据流，采用算法定位终端信息(IP、host name)、用户参数、长度域、Cookie、任意域，然后根据这些信息生成交互脚本. 此后，该系统根据这些脚本信息识别任意一段新的流量是否符合已经学习的协议，并且可以通过关键域替换，实现自动交互的效果. 在另外一篇论文中[3]，他们结合 HoneyFarm 和上面提到的 RolePlayer 技术，实现了一种全新的恶意代码捕获框架 GQ：所有进入的流量，首先进行匹配，如果发现是已知协议，则直接由 RolePlayer 响应，一旦发现未知协议，则将这些流量重定向到高交互蜜罐中并扮演代理服务器的角色，以便捕获到最新的恶意代码和恶意攻击. 连续 4 个月的测试中，GQ 捕获了 66 种不同的蠕虫，这充分说明，RolePlayer 在 GQ 中确实发挥了重要而且有效的作用. 但是这种方法没有分析协议结构，可变性不高，流量稍微有点变化就导致无法正常响应. 在随后的研究中，他们引入了 Discover 系统[4]，试图从大量的网络报文中识别出同类型报文，然后分析报文格式，这种方法显然比 RolePlayer 更加实用，但是文章缺乏对于 Discover 所采用方法的精确定义.

① Network protocol analysis using bioinformatics algorithms. http://www.4tphi.net/~awalters/PI/PI.html, 2004

基于“Tainted Data”的协议识别方法是建立在动态污点分析技术基础之上的.动态污点分析技术是近几年兴起的一种解决方案,它可以在二进制代码层实现对不可信数据的传播情况的跟踪和分析,Argos[5]和 TaintCheck[6]就是两个典型的实例.基于“Tainted Data”的协议识别方法将所有网络传输数据作为不可信数据源,通过监控服务器处理这些不可信数据的流程,获取协议结构信息,因此其准确性要好于第一种方法.例如服务器程序在污点数据中搜索非污点数据“Username”,那么就可以肯定污点数据中的“Username”是报文结构中的一个关键字.Wondracek 详细描述了这种方法的机制[7]:利用网络报文作为输入的 Tainted Data,在指令级对其进行监控,然后根据监控信息分析对应报文的语义特征.最后根据多次监控的分析结果,将所有相同格式报文的语义信息融合,提取出一个通用的报文结构.Caballero 等人提出的 Polyglot 系统也是基于类似的思想实现的[8].Comparetti 设计的 Prospex[9]在引用以上方法识别单个报文结构的同时,也利用污点数据跟踪过程中获取的统计特征(包括系统调用特征、文件系统操作特征等)对报文进行聚类.此外,文中还引入状态机识别机制并简化,最终将客户机和服务器的交互过程以 DFA 的方式显示出来,达到解析交互流程和报文结构的目的.然而这种基于“Tainted Data”的方法也存在一些限制:它需要得到服务器程序并让其在指定环境中运行;其次,跟踪复杂程序运行也会产生大量数据需要分析并且降低服务器性能.

关于Fuzz Testing的研究也很活跃.Fuzz Testing 是一种通过外部输入数据影响内部程序执行的测试技术.它的基本原理是将尽可能多的可能导致程序出现问题的错误数据注入应用程序中,观察运行结果,通过观察和分析程序运行的错误来挖掘软件的脆弱点.Fuzz Testing 看似简单并且需要大量数据和注入点,但它却能揭示出程序中的重要漏洞,而且通过 Fuzz Testing 挖掘到的漏洞通常都比较严重.Miller[10]用 Fuzz Testing 测试了大量的 Unix 程序,结果发现超过 40%的命令行程序可以出现内存出错和死循环,同样超过 40%的 X-Windows 程序也会出错,显示了 Fuzz Testing 的有效性.在此基础上 Forrester[11]采用类似的方法对 Windows GUI 程序进行 Fuzz Testing,测试数据注入点是键盘鼠标输入和消息,同样使 21%的程序内存出错和 24%的程序陷入死循环,说明了 Fuzz Testing 对不同平台程序都是有效的.现在 Fuzz Testing 已经发展成为一种模块化的测试方法,典型的软件是 SPIKE[12].SPIKE 是 Immunitysec 公司的 Dave Aitel 写的一个黑盒安全测试工具,主要可以测试应用程序输入和网络协议报文.微软的 Godefroid Patrice 在 Fuzz Testing 方法中引入了白盒测试的方法,使得 Fuzz 更加精确,覆盖率也更高[13].国内研究者张玉清教授分别对 MP3 播放器、TFTP 客户端进行了 Fuzz Testing[14-15].

Fuzz Testing 的主要问题在于它是一种盲注入的方式,手工进行 Fuzz Testing 工作量过大,甚至几乎无法进行比较全面的测试,导致 Fuzz Tesing 的实用价值受到很大影响.

本文提出的方法建立在这两个领域的研究基础上,提出一个适用于 Fuzz Testing 的网络协议自动识别的方法.由于 Fuzz Testing 需要输入大量数据,为了避免在服务器端产生过量的二进制执行信息,本方法基于“Network Trace”,但是也引入了简单的服务器监控机制.重点在通过分析网络协议的报文,自动化精细识别网络报文中各个域,并推断其数据类型.然后根据域的划分,自动产生 SPIKE 所需的 Fuzz Testing 脚本,对每个域进行注入测试.测试过程中仅仅对服务器端运行结果进行监控,一旦发现服务器端出现拒绝服务或者溢出错误则记录相应的输入和输出.通过这种方式来发现网络协议设计和实现中的漏洞.经过对典型的公开和未公开网络协议的测试,证明这种方法是非常有效的,其主要的系统结构如图 1 所示.

图 1 描述了本方法的核心流程.本方法主要部分都采用自动化的方法,但是需要一个手工截获报文的阶段,即使用 Tcpdump 或者 WireShark 等工具,截获网络协议的报文.这个阶段需要用户对网络服务多次使用(一般 3～4 次),然后将每次截获的报文保存到一个 PCAP 文件中,以多个 PCAP 文件作为自动化阶段的输入.这个阶段不需要进行协议识别或者测试工作,因此易于操作和实现.在自动化部分主要分为两个阶段:协议识别和 Fuzz Testing.在协议识别阶段,首先采用类型匹配,将不同的 PCAP 文件中同类型的报文序列提取出来,作为一个报文组.然后对这个报文组进行多序列比对,将不变域和可变域分离出来,得到一个初步的报文域划分.报文域识别模块再进行进一步的 ANSII 字符串域、Unicode字符串域、二进制域、长度域的识别,得到一个较为准确的报文格式.报文格式得到后,可以根

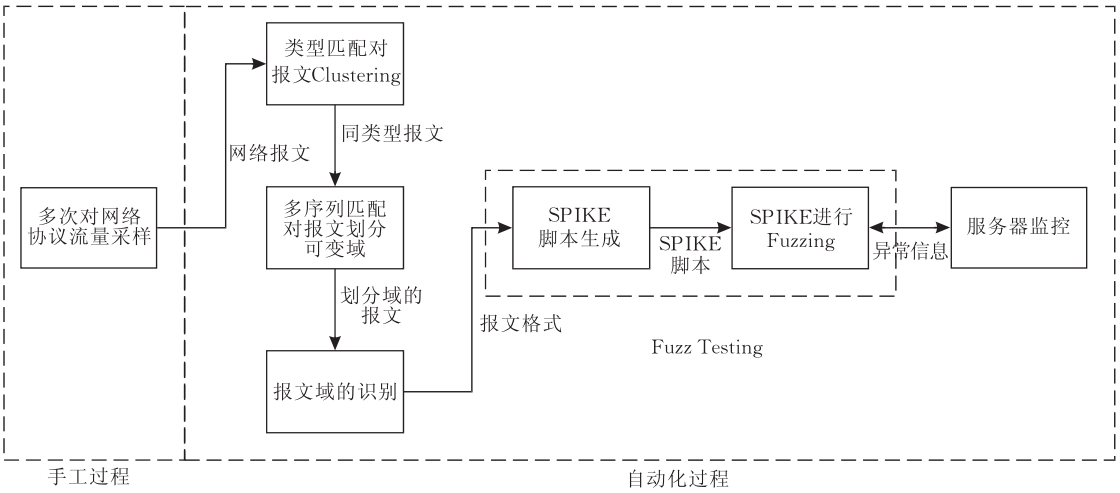



图 1　系统结构图

据此格式生成 SPIKE 脚本，由 SPIKE 根据脚本构造实际的报文对服务器进行测试. 而服务器主要安装 FaultMon 等监控软件，一方面监控内存使用情况，监控内存异常和系统服务状态；另外一个方面是记录被测试服务程序的异常和日志. 通过这两个部分的信息，可以反推漏洞可能的类型和存在的位置. 在这样的系统结构下，可以实现自动化地对网络服务器进行 Fuzz Testing，测试网络协议在设计和实现上的安全性.

## 3　报文聚类

为了能够识别未知网络协议，首先需要人工对网络协议进行多次访问，每次访问为一个 Session，每一个 Session 的报文都截获保存到一个 PCAP 文件中. 为了提取同类型报文，需要对多个 PCAP 文件中的报文进行聚类，将相同类型的报文序列组成一个报文组，属于同一个报文组的报文被认为是网络协议中相同格式的报文，它们将被作为多序列比对算法的基本输入.

具体方法为，提取 PCAP 文件中的非空负载报文，每个报文以各自原始序号标记，具有相同序号的报文作为一组. 此时的报文组，并不能保证它们是相同类型的，因为考虑到网络传输中可能出现丢包、乱序等情况，甚至相同类型的数据包的内容也极有可能有很大差异. 因此需要对每个报文组进行进一步分析. 对每个报文序列的每个字节识别相应的类型，如果是可打印字符用‘A’表示，非可打印字符用‘B’表示，此时这个报文序列对应的类型就形成一个由若干‘A’、‘B’组成的字符串. 考虑到字符串长度是可变的，我们将连续的‘A’合并成一个‘A’，如此形成一个新的序列，我们称这个序列为类型序列(Type Squence).

对类型序列进行聚类的算法如下.

```
def clusterSequences(typeSquences)
  typeCompare=False
  for seq in typeSquences do
    for byte in seq do
      if byte is not printable do
        typeCompare=true
        break
      done
    done
  done
  if typeCompare do
    model=
      GetHighestFrequencyTypeModel(typeSquences)
  else
    model=GetLongestCommonString(typeSquences)
  done
  for seq in typeSquences do
    if !MatchModel(model, seq) do
      typeSquences.remove(seq)
    done
  done
```

算法判断如果是纯文本的协议通过查找报文之间的最频繁最长共同串(longest common string)，找到具有共同类型序列的报文. 而具有不可打印字符的报文，说明不是纯文本协议，通过找到最频繁的类型序列，发现具有共同类型的报文，移除不符合要求的报文.

## 4　多序列比对

已有的多序列比对算法大体分 3 类：精确比对

算法、渐进比对算法和迭代比对算法[16]. 精确比对算法最为经典的是多维 Needlman Wunsch 算法,但其可行的计算维数为 3,Carrillo Lipman 算法通过减小计算空间,将计算维数提高到 10. 渐进比对算法由 Hogeweg 首先提出,Feng 和 Taylor 又加以完善,被广泛使用的多序列比对软件包 CLUSTAL W 就是基于渐进比对思想构建. 近年来,迭代比对算法被越来越多地用于求解多序列比对问题,基于模拟退火、遗传算法、Hidden Markov Model、Gibbs 抽样等的多序列比对算法被广泛应用于多序列比对问题的求解.

由于网络报文的多序列比对具有序列很长、报文数量多的特点,精确匹配需要大量的计算时间和内存空间,因此我们放弃了精确比对,主要考虑了基于序列长度的渐进比对算法和基于遗传算法的迭代比对算法. 通过比较这两种算法,我们发现简单的基于序列长度的渐进比对算法在效果和效率上大大优于后者,所以最终选择渐进比对算法作为协议分析的多序列比对算法. 在本节,分别介绍我们对这两种算法的改进,以及对两种序列比对算法的效果和性能测试.

### 4.1 基于遗传算法的迭代比对算法

本文利用基于遗传算法的多序列比对算法对多个数据包进行比对,得到反映这些报文共性和差异的比对结果. 具体的遗传算法设计如下.

**目标函数.**

在序列比对结果中,每个序列插入的空位(Gap)数量越少越好,因此目标函数加大了对 Gap 的惩罚力度,设多个序列为 $seqs=\{s_1,s_2,\cdots,s_n\}$,经过遗传算法填充序列长度相等都为 $len$,则

$$SP(seqs)=\sum_{i=1}^{len}SPCol(s_{1,i},s_{2,i},\cdots,s_{n,i}),$$

$$SPCol(c_1,c_2,\cdots,c_{len})=\sum_{i=1}^{i=len}\sum_{j=1,j!=i}^{j=len}Cmp(c_i,c_j),$$

$$Cmp(c_1,c_2)=\begin{cases}2, & c_1=c_2\ \&\ c_1!=gap\\ -1, & c_1!=c_2\ \&\ c_1!=gap\\ 2, & c_1!=c_2\ \&\ (c_1=gap\,|\,c_2=gap)\\ -2, & c_1=gap\ \&\ c_2=gap\end{cases}\tag{1}$$

式(1)即为 Sum-of-Pairs 目标函数的评分规则(简称 SP),注意如果遇到全部为 Gap 的列,直接删除,不计算分数. 可以看出,对于空位 Gap 的限制比其它字符要大,因此,最终比对结果中出现的 Gap 数量也相应减少.

**遗传算子.**

这里实现了 9 种遗传算子,假设每个个体包含 $N$ 个序列,下面分别简要介绍[17].

(1) CrossOver. 随机选择两个个体 $p1$ 和 $p2$,然后将 $p1$ 的前 $i$ 个序列和 $p2$ 的后$(N-i)$个序列进行组合,形成新的个体.

(2) Mutate. 随机选择一个个体及这个个体中的一个序列,删除这个序列中所有的空格,然后随机插入相同数量的空格,形成新的个体.

(3) LocalShuffle. 随机选择一个个体及这个个体中的一个序列,将这个序列中的一个 Gap 和相邻字符交换.

(4) BlockShuffle. 随机选择一个个体及这个个体中的一个序列,将这个序列中的一个 Gap 和相邻字符串交换.

(5) InsertGap. 随机选择一个个体及这个个体中的一个序列,随机删除这个序列中的一个 Gap 并随机插入一个 Gap.

(6) DeleteGap. 随机选择一个个体及这个个体中的一个序列,随机删除这个序列中的一个 Gap,并在序列末尾添加一个 Gap.

(7) Union. 随机选择一个个体及这个个体中的一个序列,将这个序列中两个相邻的字符串合并,即删除一个连续的 Gap 串,然后在相邻字符串的左边或右边插入相同数量的 Gap.

(8) Division. 随机选择一个个体及这个个体中的一个序列,将这个序列的一个字符串拆分,即随机删除一个 Gap,然后随机插入一个 Gap 到这个字符串中.

(9) CleanUpGapColumn. 检查是否存在全为 Gap 的列,如果存在的话,将此列删除并在每个序列的末尾添加一个全为 Gap 的列.

**动态退出流程.**

在遗传算法的实现中,所有序列的长度都是固定的(默认设置为最大长度的 1.2 倍). 在编程实现中,采用轮盘赌算法随机选择 SP 值比较大的个体进入下一代种群的演变. 我们知道,轮盘赌算法中,对应的特征值不能为负值. 但是根据以上算法计算的目标函数很有可能是一个小于 0 的整数. 为了避免出现负值的现象,在进行轮盘赌选择的时候,需要将所有值转换为正值,然后进行选择.

目前实现了 9 中遗传算子. 但是这些遗传算子的一个特点在于,它们随机性较强,增加了种群的多样性,不过这也是它们的一个缺点,即如果进入一个

很差的解空间，有可能一直在这个空间内迭代，无法再优化. 为了得到更优的结果，本文从每次产生的最优个体中提取免疫因子，通过免疫因子，产生新个体. 实验测试表明，免疫算子可以在很大程度上优化比对的结果. 在程序中增加实现了两种免疫算子：单一免疫因子和多免疫因子. 单一免疫因子每次提取最优个体中最长的连续的相同的列作为免疫因子，产生新个体，而多免疫因子则提取最优个体中多个连续的相同的列为免疫因子，产生新个体.

**遗传算法测试.**

为了测试遗传算法在多个报文序列对比中的效果，本文设计了 6 组测试数据，每组数据包含 4 个相似的网络报文，对每组数据进行基于遗传算法的多序列比对，共测 10 次，取平均结果. 表 1 为测试数据描述，表 2 左侧是不引入免疫因子的测试结果，右侧为引入免疫因子后的实验结果.

**表 1　多序列比对数据**

| 报文组 | 平均报文长度 | 相似性描述 |
|---|---|---|
| 1 | 200 字节左右 | 每行大约 0～10 个字节发生不同 |
| 2 | 200 字节左右 | 每行大约 10～20 个字节发生变化 |
| 3 | 200 字节左右 | 每行大约 0～10 个字节发生变化，最后一个报文多 10 个字节 |
| 4 | 200 字节左右 | 每行大约 10～20 个字节发生变化，最后一个报文多 10 个字节 |
| 5 | 200 字节左右 | 每行大约 0～10 个字节发生变化，最后一个报文少 10 个字节 |
| 6 | 200 字节左右 | 每行大约 10～20 个字节发生变化，最后一个报文少 10 个字节 |

**表 2　遗传算法测试结果**

| 普通遗传算法测试结果 | | | | 免疫遗传算法测试结果 | | | |
|---|---|---|---|---|---|---|---|
| 报文组 | 平均 SP 值 | 平均个体数量 | 平均种群数量 | 报文组 | 平均 SP 值 | 平均个体数量 | 平均种群数量 |
| 1 | 793.3 | 46639.6 | 446.5 | 1 | 1206.2 | 48839.4 | 727.7 |
| 2 | 500 | 48020.8 | 448.8 | 2 | 962.7 | 45829.7 | 666.3 |
| 3 | 353.3 | 52084.6 | 494.2 | 3 | 816.7 | 51674.6 | 770.2 |
| 4 | −64.7 | 37862.4 | 381.8 | 4 | 695.3 | 52858.6 | 775.2 |
| 5 | 570.9 | 52065.8 | 508.4 | 5 | 1096.8 | 57300.7 | 829 |
| 6 | 212.6 | 38786.7 | 389.8 | 6 | 906.5 | 70889.5 | 985.9 |

总体而言，遗传算法实现多序列比对还是存在一些不足. 首先，遗传算法本身存在很大的随机性. 即便是相同的序列，多次比对的结果也并不一定相同，最佳分析结果得到的 SP 值也较低. 而在协议自动分析中，对序列比对结果要求极为苛刻，任何错误的比对，都将直接影响后续分析工作. 此外，遗传算法消耗的时间相对比较长，当序列数量或者序列长度达到一定阈值之后，运行时间甚至无法忍受. 因此遗传算法不适合用于协议自动分析的多序列比对中.

### 4.2　基于序列长度的渐进比对算法

渐进比对算法建立在二维 Needleman Wunsch 算法基础上，基于相似序列通常具有进化相关性这一假设，它的思想是通过迭代地利用双序列动态规划比对算法，先由两条序列的比对开始，逐渐添加新序列，直到所有序列都加入为止. 但是不同的添加顺序会产生不同的比对结果. 因此确定适合的比对顺序是渐进比对算法的一个关键问题. 渐进比对算法主要由 3 个步骤组成：(1) 计算距离矩阵；(2) 构建向导树；(3) 依据向导树进行渐进比对. 针对报文序列比对的特点，我们实现了一种优化的 Needleman Wunsch 算法.

**优化的 Needleman Wunsch 算法.**

Needleman Wunsch 算法属于动态规划算法，算法结束时通过回溯得到一个全局的比对结果，根据该比对的结果计算出的序列相似度值最大. 算法有两个步骤：(1) 根据状态转换函数计算两个序列的相似分值，得到一个相似度矩阵. (2) 根据相似度矩阵，按照动态规划的方法回溯寻找最优的比对. 核心的状态转换函数定义如下：

$$M_{ij}=\max\begin{cases}M_{i-1,j-1}+S_{ij}\\ M_{i,j-1}+w\\ M_{i-1,j}+w\end{cases}\tag{2}$$

其中，$M_{ij}$ 表示当前状态得分，$S_{ij}$ 是字母匹配得分(如果不匹配就是罚分)，$w$ 是加入空位的惩罚. 因为空位不属于原始报文，为了保持报文长度和增加匹配的可信度，应该尽量少加入空位，所以我们设置字母匹配 $S_{ij}$ 得分为 2，不匹配 $S_{ij}$ 罚分为 $-1$，$w=-2$. 这样的设置在大部分情况下有效，但是经典的 Needleman Wunsch 算法的每个状态仅仅基于上一个状态得到，没有考虑对于连续匹配进行奖励，因此可能出现为了避免增加空位导致不能对齐的现象.

图 2 上侧截取了两个互相比对的报文中的部分序列，采用经典的 Needleman Wunsch 算法匹配后，发现这两个序列没有正确地对齐字符串 Referer：http://192.168.1.1/，这是因为为了避免过多的插入空位，算法宁可选择字符串不匹配. 那么最终得到的 Fuzz Testing 报文模板不会把这个部分作为不可变域，影响分析效果. 因此我们修改了 Needleman Wunsch 算法的状态转换函数，增加对于连续匹配的奖励.

$$M_{ij}=\max\begin{cases}M_{i-1,j-1}+S_{ij}+(n-1)\times b\\ M_{i,j-1}+w\\ M_{i-1,j}+w\end{cases}\tag{3}$$

其中，$n$ 是连续匹配的字母数目，$b$ 为连续匹配奖励，

设定为 2,这样鼓励算法将连续的字母比对在一起,即使插入更多的空位也不会产生比对的碎片,达到如图 2 下侧的效果,虽然插入了更多的空位,但是实际上更加符合网络报文匹配的要求,得到的结果更便于发现报文中的变化域.即Referer:http://192.168.1.1/在两个报文中正确地匹配在一起,而客户端识别的文件类型和服务器端的页面路径被标识出来,这两个区域可以分开进行 Fuzz Testing,增加了检测的粒度,容易触发更多漏洞,而且这样的报文更加容易通过服务器的格式检查.当然增加了连续奖励会增加算法的复杂度,因为必须更新状态,记住连续匹配是否产生以及匹配的字符个数.

```
_*/*Referer:http://192.168.1.1/console/wwwdir/logon/logon_by_us_ername_passwd_index.html
application/vnd.ms-excel, application_/x-shockwave-flash, */*Referer:http://192.168.1.1_
```

```
_____*/*Referer:http://192.168.1.1/
console/wwwdir/logon/logon_by_username_passwd_index.html
application/vnd.ms-excel, application/x-shockwave-flash, */*Referer:http://192.168.1.1_
_____
```

图 2 优化算法前后效果对比(用下划线代替空位)

**渐进比对算法流程.**

由于通过了报文聚类,我们认为剩下的报文应该是相似的,而且长度决定了报文变化的大小.因此我们提出了基于报文长度的渐进比对算法.算法的核心就是构建一棵二叉树,通过比较报文的长度,将长度最为接近的报文两两编组,

```
def ConstructTree(seqs)
for seq in seqs do
    node=new TreeNode(seq)
    nodeList.append(node)
done
while(len(nodeList)>1)
    node1, node2=
          FindAndRemove2ShortestSeqs(nodeList)
    newNode=new TreeNode()
    newNode.seq, node1.gapList, node2.gapList=
          NeedlemanWunsch(node1.seq, node2.seq)
     newNode.left=node1
    newNode.right=node2
    nodeList.append(newNode)
done
return nodeList[0]
```

算法的核心是每次查找最短的两个序列作为二叉树的叶子节点,然后进行 Needleman-Wunsch 算法比对,返回的三个值,第一个是插入 Gap 较少的比对结果,后两个是两个比对结果相对原始序列需要插入的 Gap 位置列表,这样递归下去,直到根节点.另外一个算法是构建多序列比对结果:

```
def SequenceResult(treeNode, resultSeqs)
  if treeNode.gapList !=null do
    GapListStack.push(treeNode.gapList)
    SequenceResult(treeNode.left)
    SequenceResult(treeNode.right)
    GapListStack.pop( )
  done
  if treeNode.left=null and treeNode.right=null do
    newSeq=
     ApplyGapListStack(treeNode.seq, gapListStack)
    resultSeqs.append(newSeq)
done
```

SequenceResult 是从根节点开始,递归调用,将所有的 gapList 都应用到叶子节点,最终形成结果序列保存在 resultSeqs 中.

**测试结果.**

为了测试渐进比对算法的效果,我们采用了和遗传算法同样的测试集合,进行了同样的测试.表 3 显示了对第 6 个报文组的测试结果.

**表 3 对比测试**

| 算法 | 平均时间/s | 最佳 SP 值 | 平均 SP 值 |
|---|---|---|---|
| 免疫遗传算法 | >60 | 1187 | 906.5 |
| 渐进比对算法 | 1.985 | 1782 | 1782.0 |

表 3 中数据显示,同样的程序多次分析同样的序列,渐进比对算法比对结果明显优于遗传算法结果.协议自动分析需要数据包序列中每个字节严格同序列中相同含义的字节对齐,否则分析无法成功进行.另外遗传算法还存在很大的随机性,每次执行的结果可能不一致,这意味着如果采用遗传算法进行序列分析的话,最终的协议结构也可能是不一致的.这在协议自动分析中是无法接受的.此外,遗传算法所消耗的时间也明显比渐进算法长.测试文件 6 中序列在 200 字节左右,最长运行时间已经达到 4 分 1 秒.随着序列长度的增加,比对时间也急剧增加.

综上所述,我们得出结论:在协议自动分析中不适合使用遗传算法实现多序列比对.因此在研究中

采用渐进比对算法作为协议自动分析中的多序列比对算法.

# 5　协议自动识别技术

为了方便描述，我们将数据包结构中的各个有意义的组成部分称为“域”(Field). 通过多序列比对，可以将多个报文进行对齐，然后根据每一列字节的变化，识别出不变域和可变域，其中不变域中每个字节对应的变化率都为 0. 从类型上看，域又可以分为二进制域和字符串域，其中字符串域对应可打印文本串. 此外，还需要识别出两种特殊的域：长度域和 Unicode 编码域. 长度域记录数据包中若干连续域的长度，而 Unicode 编码域指的是由 Unicode 编码的文本域. 由于我们的识别目标是应用层协议，而在下层协议中已经提供了校验和，因此就没有考虑在应用层出现校验和域的情况. 总体而言，我们希望通过协议自动分析，能够准确地识别出协议报文中的不变二进制域、可变二进制域、不变字符串域、可变字符串域、不变 Unicode 编码域、可变 Unicode 编码域和长度域.

识别各种域之后，域的集合就构成了数据包的模型，称为 Packet Model，而若干数据包模型和交互控制信息构成了目标 Protocol Model. 为了得到 Protocol Model，需要完成以下步骤：根据多序列比对结果划分域；进行类型纠错；按照类型划分标准数据包；识别 Unicode 编码；识别长度域；类型细粒度划分；格式化序列；输出 Packet Model.

另外还要强调一点，为了方便使用 Packet Model 产生模糊器，本文要求在参与比对的数据包中，必须有一个是进行漏洞测试的主机之间的流量抓包文件. 而这个数据包对应的序列称为默认序列，它是 Packet Model 对应的一个模板.

## 5.1　根据多序列比对结果划分域

在进行多序列比对时，为了能够使序列最大程度对齐，在必要的情况下需要插入空位，即前面提到的 Gap. 在比对完成之后，每个序列具有相同的长度. 统计每个序列中相同偏移的字节，计算对应的变化率. 如果相同偏移的所有字节数值都相等，则变化率为 0；如果不相等，计算不同数值的数量，它和序列个数的比值作为变化率. 根据变化率识别出不同的域，相同变化率的字节构成一个域，其中包括不变域和可变域. 接着识别每个域的类型. 对于每个域，除去 Gap 以外，如果找到一个不可打印字符，则表示这个域是二进制域，否则该域为可打印字符域.

## 5.2　Unicode 编码识别

这里的 Unicode 编码，指的是在每个字符添加一个“00”作为前缀或后缀，例如字符‘A’的 Unicode 编码为“41 00”. 很显然，Unicode 字符串的一个很明显的特征在于每个可打印字符之间以“00”间隔，这也是我们识别 Unicode 编码的标准. 遍历每个序列，查找连续间隔出现“00”、而每两个“00”之间是可打印字符的字节范围. 根据“00”相对可打印字符的位置确定 Unicode 编码类型. 然后将这个字节范围的字节合并成一个整体，作为一个 Unicode 域. 整个识别过程中，忽略 Gap 信息. 如果进行扩展可以识别报文中的汉字，例如 IIS 的 FTP 服务器采用 UTF-8 传输汉字.

## 5.3　类型纠错

前面的处理中可以根据字节的数值，确定一个域类型为二进制域或字符串域，但是这种方法在某些情况下是不正确的. 例如，对于数值 0x40，它既可以解释成‘@’字符，但是也可以表示一个二进制数值. 在这种情况下，很难直接通过数值来判断数值的类型. 考虑到每个字节和前后字节序列之间存在一定的联系，因此可以根据字节所处环境来判断. 这里假定所有的可打印字符域有一个最小长度值，所有长度小于这个阈值的字符串域，都被认为是二进制域. 理论上讲，这种策略并不是很精确，因为我们无法保证所有的字符串域的长度都大于或等于这个阈值，但是却可以在大多数情况下能够正确识别出域类型. 在实现中，我们将这个阈值作为一个参数供用户配置，默认值为 4.

## 5.4　长度域识别

在多序列比对时，已经向序列中插入 Gap，这些 Gap 在计算长度域时不应该存在. 为了保留细粒度划分片段所需要的信息，此时保存默认序列的 Gap 信息及各字节的变化率信息，然后删除所有的 Gap.

长度域用来描述数据包某一部分的长度，实际上也就是表示若干连续的序列片段长度之和. 在实际网络中，如果一个数据包中的长度域出现错误，那么极有可能是传输错误而要求重传，因此长度域的确定是协议结构自动分析的一个很重要的部分. 长度域的识别过程相当复杂，而且最终的分析结果可能遗漏，甚至不能保证分析结果一定是正确的.

长度域的识别策略基于以下几个假设：

(1) 首先，不考虑一个报文分为多个数据包到达的情况，此外数据包乱序到达也应该排除.

(2) 其次，长度域用于记录若干连续序列片段的长度之和. 而长度域都是 1～2 个字节长度的二

进制数值,而且高位在前.暂时不考虑以明文方式表示的长度域,例如 EM 协议中请求报文中存在“Content-Length:95”的长度域.

(3)长度域不可能出现在它所标记的序列片段范围之后.实际上,SPIKE 框架也不支持这种情况.

长度域的识别思路如下:对每个参与比对的序列,在已经识别的域的基础上,穷举所有可能的位置连续的域,并计算其长度,位置连续的域称为连续域集合.然后在整个序列的二进制域中搜索长度信息,一旦发现二进制域中存在匹配的连续域长度,就记录下来,这个信息成为长度向量(采用相对位移表示).每个序列都会产生一个长度向量集,对所有的长度向量集求交集,在解决长度向量交集的冲突之后,剩下的长度向量就代表最终的长度域.下面为详细识别过程.

在进行长度域定位之前,已经完成域的类型识别、类型纠错、Unicode 编码识别,这几个阶段的准确率直接关系到长度域定位的准确度.为了最大限度查找到长度域,需要穷举所有可能的连续的序列片段组合,计算每种组合的长度值.假设每个序列的域数量为 $n$,那么总共可能出现的组合数量为 $1+2+\cdots+n=n(n+1)/2$.当在二进制域中搜索到匹配的值时,就表示这个值可能记录了对应的域的长度.但是不能排除这是一种巧合,需要将所有匹配的信息以(起始域序号、终止域序号、长度域所在二进制域序号、长度域所在二进制域内偏移,长度域长度)的形式记录下来,形成一系列的长度向量,一个序列的所有长度向量组成一个长度向量集.一个长度向量集表示了在对应序列中可能出现长度域的组合.对所有的序列的长度向量集求交集.由于长度向量采用的是相对偏移,和具体序列无关,因此交集中的每个长度向量意味着它所标记的长度域在所有的序列中都是符合的.在实际测试中,我们发现长度域交集中可能存在冲突的现象.主要表现为

(1)一个长度域标记多个连续域集合;

(2)一个连续域集合可能被多个长度域标记;

(3)长度域本身重叠.例如,“00 ff”可以作为长度域标记一个连续域集合,但是“ff”本身也可能作为长度域标记另一个连续域集合.

遇到第 1 种冲突情况,总是优先选择距离长度域近的连续域集合作为长度范围.第 2 种冲突在实际数据包中出现过,因此被认为是一种正常情况,无须解决冲突.第 3 种冲突中优先选择 2 个字节长度域,而删除一个字节的长度域信息.当所有冲突解决之后,此时的长度向量集合的每一个向量,都标志着一个识别出来的长度域.

前面已经提过默认序列的概念.默认序列是需要进行漏洞挖掘的主机之间的数据包序列.从序列组中提取出默认序列,然后利用长度向量集,划分默认序列的二进制域,将长度域独立出来,并记录对应的片段组合信息.此时默认序列是我们进一步分析的对象,其它序列信息已经可以删除.

### 5.5 格式化输出

在报文序列的可打印字符串序列中,还可能出现一些特殊字符,例如“(”、“)”、“:”、“,”、“=”等,这些字符作为分隔符而存在.例如在报文中常出现“userName=sys”的情况,得到的是不变域“userName=”、可变域“sys”.相对来说,我们更希望得到的结果是不变域“userName”、不变域“=”、可变域“sys”.因此这一步操作就是将分隔符从默认序列中独立出来,从而更细粒度划分字符串域.

在格式化默认序列之后,就完成了协议自动识别过程.默认序列分析结构将以 HTML 的方式输出,称为 Packet Model.而进行模糊测试器也是在 Packet Model 的基础上完成的.表 4 为 Oracle 数据库的非公开协议 TNS 经过自动分析后,在默认序列上形成的 Packet Model.

**表 4 TNS 第 4 个报文的 Packet Model**

| 序号 | 域属性 | 域值 |
|---|---|---|
| 0 | Length Field(2):0~42 | x00xff |
| 1 | Binary-constant | x00x00x01x00x00x00x01x39x01 x2cx00x00x08x00x7fxff xc6x0ex00x00x01x00 |
| 2 | Length Field(2):5~42 | x00xc5 |
| 3 | Length Field(2):0~4 | x00x3a |
| 4 | Binary-constant | x00x00x02x00x61x61x00x00x00 x00x00x00x00x00x00x00 x00x00x00x00x00x00x00x00x00 x00x00x00x00x00 |
| 5 | String-constant | ( |
| 6 | String-constant | DESCRIPTION |
| 7 | String-constant | =( |
| 8 | String-constant | CONNECT_DATA |
| 9 | String-constant | =( |
| 10 | String-constant | SERVICE_NAME |
| 11 | String-constant | = |
| 12-0 | String-variable | orcl |
| 13 | String-constant | )( |
| 14 | String-constant | CID |
| 15 | String-constant | =( |
| 16 | String-constant | PROGRAM |
| 17 | String-constant | = |
| 18-0 | String-variable | F |
| 18-1 | String-variable | : |
| 18-2 | String-variable | \oracle\ |
| 19 | String-constant | 10.2.0\db_1\bin\ sqlplus.exe |

（续　表）

| 序号 | 域层性 | 域值 |
|---|---|---|
| 20 | String-constant | ) ( |
| 21 | String-constant | H O S T |
| 22 | String-constant | = |
| 23-0 | String-variable | 3 2 2 6 6 6 3 C C D 3 D 4 9 2 |
| 24 | String-constant | ) ( |
| 25 | String-constant | U S E R |
| 26 | String-constant | = |
| 27-0 | String-variable | A d m i n i s t r a t o r |
| 28 | String-constant | ) ) ) ( |
| 29 | String-constant | A D D R E S S |
| 30 | String -constant | =( |
| 31 | String-constant | P R O T O C O L |
| 32 | String-constant | = |
| 33 | String-constant | T C P |
| 34 | String-constant | ) ( |
| 35 | String-constant | H O S T |
| 36 | String-constant | = |
| 37-0 | String-variable | 1 9 2 . 1 6 8 . 1 . 7 8 |
| 38 | String-constant | ) ( |
| 39 | String-constant | P O R T |
| 40 | String-constant | = |
| 41 | String-constant | 1 5 2 1 |
| 42 | String-constant | ) ) ) |

## 6　构造模糊器

本文在 SPIKE 框架的基础上，自动生成模糊测试器 Fuzzer. SPIKE 是一个对网络协议进行 Fuzz Testing 的通用框架，也是使用最为广泛并且被人所熟知的模糊测试框架之一. SPIKE 使用数据块的网络协议分析测试方法，通过这种方法，可以自动完成数据块长度的计算和填充，提高了测试成功率，同时也使开发过程相对简单.

Fuzzer，实际上是进行 Fuzzing 测试的程序. 在 SPIKE 框架下，它由两部分组成：主控程序和数据包构造程序. 主控程序负责和目标服务程序进行交互操作，例如发送、接收数据包等. 数据包构造程序可以通过 SPK 脚本的形式实现. SPK 脚本存放着构造相应数据包的语句，主控程序根据 SPK 脚本构造具体的数据报文.

本文分析目标网络协议，得到了两方面的信息：

```
s_block_start("length_0");
s_block_start("length_3");
s_binary_block_size_halfword_bigendian_variable("length_0");
s_binary("00 00");
s_binary("01 00");
s_binary("00 00");
s_binary("01 39");
s_binary("01 2c");
s_binary("00 00");
s_binary("08 00");
s_binary("7f ff");
s_binary("c6 0e");
s_binary("00 00");
s_binary("01 00");
s_binary_block_size_halfword_bigendian_variable("length_2");
s_binary_block_size_halfword_bigendian_variable("length_3");
s_binary("00 00");
s_binary("02 00");
s_binary("61 61");
s_binary("00 00");
s_binary("00 00");
s_binary("00 00");
s_binary("00 00");
s_binary("00 00");
s_binary("00 00");
s_binary("00 00");
s_binary("00 00");
s_binary("00 00");
s_binary("00 00");
s_binary("00 00");
s_binary("00 00");
s_binary("00 00");
s_block_end("length_3");
s_block_start("length_2");
s_string("(");
s_string("DESCRIPTION");
s_string("=(");
s_string("CONNECT_DATA");
s_string("=(");
```

```
s_string("SERVICE_NAME");
s_string("=");
s_string_variable("orcl");
s_string(")(");
s_string("CID");
s_string("=(");
s_string("PROGRAM");
s_string("=");
s_string_variable("F");
s_string(":");
s_string_variable("\\oracle\\");
s_string("10.2.0\\db_1\\bin\\sqlplus.exe");
s_string(")(");
s_string("HOST");
s_string("=");
s_string_variable("3226663CCD3D492");
s_string(")(");
s_string("USER");
s_string("=");
s_string_variable("Administrator");
s_string(")))(");
s_string("ADDRESS");
s_string("=(");
s_string("PROTOCOL");
s_string("=");
s_string("TCP");
s_string(")(");
s_string("HOST");
s_string("=");
s_string_variable("192.168.1.78");
s_string(")(");
s_string("PORT");
s_string("=");
s_string("1521");
s_string(")))");
s_block_end("length_0");
s_block_end("length_2");
```

图 3　根据 Packet Model 自动生成的 TNS 协议第 4 个报文的 SPK 脚本

其一，默认序列，它反映了数据包的交互过程，可以通过读取默认序列了解网络协议的类型、IP 地址、端口、主机名、报文交互过程等特征. 另外一个是 Packet Model，它记录了数据报文应该如何构造以及报文中哪些部分是可以变化的. 因此可以从默认序列自动化产生主控程序，而从 Packet Model 自动化产生 SPK 脚本. 这样来产生 SPIKE 框架所需要的所有信息.

根据 TNS 协议第 4 个报文的 Packet Model 产生的 SPK 脚本如图 3 所示.

## 7 实验测试

为了测试系统的运行效果，我们选择了多种协议进行了深入测试. 通过对不同协议的测试，检验本方法对不同报文格式和数据类型的适应性. 为了分析比较，目标协议首先经过人工分析，设计出数据包构造脚本和交互控制信息，利用 SPIKE 挖掘相关协议漏洞. 然后在没有任何人工干预的情况下，利用协议自动分析技术，自动生成 SPIKE 模糊器，再次挖掘协议漏洞. 通过比较两次漏洞挖掘情况，来验证自动化方法的有效性.

### 7.1 FTP 协议测试

FTP 协议是一个公开的协议，它的应用也非常广泛. 在 FTP 服务器软件 ServU 4.0 上存在两个远程缓冲区溢出漏洞. 其中一个是列目录 LIST 命令：LIST -l:ParamString. 如果 ParamString 是一个超长的字符串，那么就会导致缓冲区溢出. 另一个存在溢出漏洞的是 MDTM 命令，格式为 MDTM time+timezone remote-filename，当“+”后面 timezone 的参数是超长字符串时，也会导致溢出.

通过对 FTP 协议进行自动化分析，能够正确地建立 LIST 和 MDTM 两个命令的报文格式，例如 MDTM 命令经过分析产生 Packet Model，并生成 SPIKE 构造脚本如图 4 所示.

```
s_string("MDTM ");
s_string_variable("20151207233859");
s_string("+");
s_string_variable("ZZZ");
s_string(" ");
s_string("/");
s_string_variable("test.txt");
s_binary("0d 0a");
```

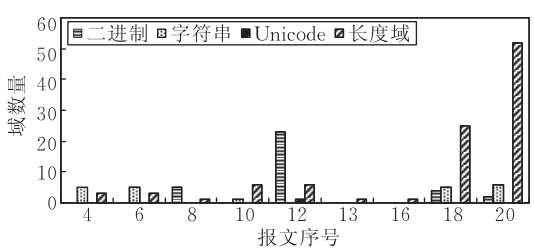
图 4 FTP 协议的 MDTM 命令 SPK 脚本

经过实际测试发现，当“20151207233859”或“ZZZ”被超长字符串替换的时候，都将导致服务器进程直接发生溢出异常而停止服务，即无论溢出“+”前面的时间还是“+”后面的时区都存在溢出漏洞. 而 LIST 超长参数溢出漏洞也能够被正确地检测到. 因此自动化 Fuzz Testing 漏洞挖掘方法对于 FTP 协议取得了很好的测试效果.

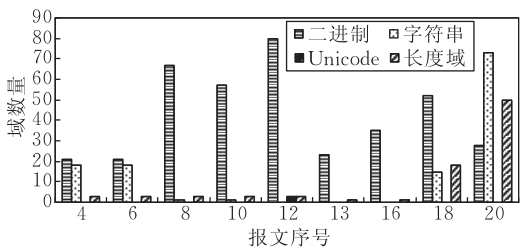
### 7.2 TNS 协议测试分析

TNS 协议是 Oracle 数据库管理系统服务端和客户端通信的协议，暂时没有官方文档进行详细介绍，属于一种未公开协议. TNS 协议传输可以使用多种方式传输，包括 TCP/IP 协议、SSL 的 TCP/IP 协议、命名管道和 IPC 等，其中 TCP/IP 协议传输部分使用明文传送. 根据观察，TNS 有 10 多种不同的报文格式，属于比较复杂的协议，而且不同的 Oracle 版本的 TNS 协议也有一定的差异，说明该协议仍然处于不断开发和改进中，同时也容易存在未公开漏洞，因此本文选择 Oracle 10g2 版本进行分析测试，既可以检验自动化 Fuzz Testing 漏洞挖掘方法对未公开网络协议的测试效果，又具有一定的应用价值.

TNS 协议在完成三次握手之后，从第 4 个数据包开始和服务器交互，在第 18 个数据包以明文的方式发送用户名到服务器，而在第 20 个数据包以密文的方式发送密码. 由于加密报文无法有效地进行 Fuzz Testing，本文主要对 TNS 的前 20 个数据包进行了分析(图 3 为第 4 个数据包的 Packet Model)，采样的报文一共是 7 组. 首先手工分析该协议，构造 SPIKE 测试脚本进行测试. 另外，利用自动化 Fuzz Testing 漏洞挖掘方法，同样产生了相应的脚本信息，进行自动化测试. 对两次的分析结果如图 5 所示.




(a) 自动识别的可变域




(b) 手工识别的可变域

图 5 自动识别和手工识别的可变域

图 5 中左侧为自动化识别出来的各种可变域，右侧为手工识别出来的各种可变域.可以看出自动化识别的可变域数量远远小于手工识别数量，主要是因为自动化识别的报文变化不够大，局限在一定的范围，导致很多可变域不能识别.人工分析时，可以根据分析者的经验，识别可以测试的域.对这些可变域分别进行测试，结果如表 5 所示.

**表 5　手工测试和自动化测试对比**

| 报文序号 | 手工分析 | | | 协议自动分析 | | |
|---|---|---|---|---|---|---|
| | 长度域 | 二进制 | 总和 | 长度域 | 二进制 | 总和 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 2 | 9 | 11 | 2 | 0 | 2 |
| 12 | 1 | 5 | 6 | 1 | 0 | 1 |
| 13 | 0 | 1 | 1 | 0 | 0 | 0 |
| 16 | 0 | 1 | 1 | 0 | 0 | 0 |
| 18 | 0 | 1 | 1 | 0 | 0 | 0 |
| 20 | 1 | 1 | 2 | 0 | 0 | 0 |
| 总和 | 4 | 18 | **22** | 3 | 0 | **3** |

表 5 中手工分析发现的漏洞而言，长度域导致的漏洞为 4 个，其它的异常均由于二进制域引发，而字符串则没有测试出任何异常情况，总的发现漏洞个数远大于自动化分析发现的 3 个漏洞.由于自动化分析的可变域和不可变域是根据采样的报文得到的，因此很可能很多不可变域其实是可变的，因此可以进一步地采样，当采样的报文组达到 15 组的时候，可以看到自动化识别的可变域增加非常显著，如图 6 所示.

根据图 6，在自动化识别的大量可变域中进行 Fuzz Testing，可以得到如表 6 所示的漏洞挖掘结果.

**表 6　增加采样报文组的漏洞挖掘结果**

| 数据包 | 手工分析 | | | | 协议自动识别 | | | |
|---|---|---|---|---|---|---|---|---|
| | 长度域 | 二进制 | 字符串 | 总和 | 长度域 | 二进制 | 字符串 | 总和 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 2 | 9 | **0** | 11 | 2 | 9 | **1** | 12 |
| 12 | 1 | **5** | 0 | 6 | 1 | **6** | 0 | 7 |
| 13 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 16 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 18 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 20 | **1** | 1 | 0 | 2 | **0** | 1 | 0 | 1 |
| 总和 | 4 | 18 | 0 | **22** | 3 | 19 | 1 | **23** |

实验数据表明，协议自动分析发现的漏洞为 23 个，已经超过手工分析发现的 22 个漏洞.就漏洞数量而言，二者性能基本类似，只有少量数据包中存在差别.这充分说明，利用协议自动分析技术来替换手工分析完成数据包的分析和 Fuzzer 的构造是可行的.通过分析两种方法发现漏洞的差异，可以进一步地了解各自的优势.

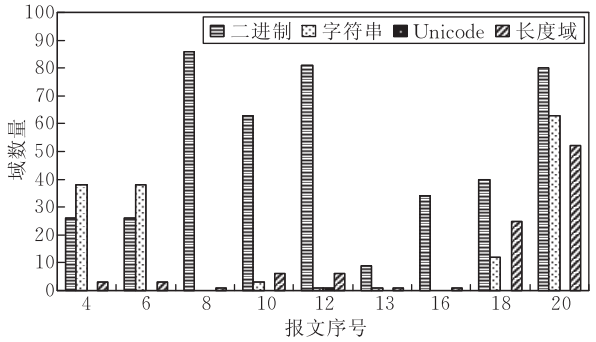



图 6　增加采样报文后识别的可变域

首先，在对第 10 号数据包进行分析的时候，协议自动识别发现了一个字符串产生的漏洞，而手工分析遗漏该漏洞.分析数据包构造脚本可以发现，在手工分析时，将字符串" NTLMSSP"作为二进制数据处理，虽然对这个域进行了测试，但是却遗漏了这个漏洞.而在协议自动识别中成功地识别这个结构并且进行了测试分析.实际上，在我们进行 TNS 协议测试中，这是唯一的一个由字符串引发的漏洞，而且这个字符串导致的是缓冲区溢出漏洞，可能会导致非常严重的安全问题.

在第 20 号数据包的分析中，协议自动识别产生的 Fuzzer 遗漏了一个长度域造成的拒绝服务漏洞.我们通过对比二者产生的数据包构造脚本分析产生这种情况的原因.在手工分析中，在传输"Session Key"的时候，有两个前置的长度域，这两个长度域由 3 个字节的 0x00 分隔.

而在协议自动识别过程中，这个结构完全被破坏：首先，传输的 SessionKey 被认为是字符串，而 SessionKey 对应的长度 0x40 刚好被认为是"@"字符(也就是图 7 中第 1 行表示的字节)，这样处理的主要原因是这两个 0x40 分别在两个字符串的末端.正是由于协议自动识别中将这个结构破坏了，所以导致最终遗漏这个长度域引发的漏洞.

```
s_binary_block_size_byte_variable("SESSKEY");
s_binary("00 00 00");
s_binary_block_size_byte_variable("SESSKEY");
s_block_start("SESSKEY");
s_binary("36 46 44 31 37 30 31 38 33 37");
s_binary("30 41 34 43 37 42 45 31 44 31 36 42 45 33 33 37");
s_binary("44 46 34 41 42 45 37 45 46 42 33 45 36 36 45 30");
s_binary("35 46 43 31 36 39 37 31 44 44 31 45 44 45 38 30");
s_binary("43 37 42 33 31 42");
s_block_end("SESSKEY");
```

图 7　手工分析的 20 号报文 SessionKey 结构

```
s_string_variable("AUTH_SESSKEY@");
s_block_end("length_13");
s_int_variable(0x0000, 5);
s_int_variable(0x00, 3);
s_block_start("length_27");
s_string_variable("@5C9B3EC4663B80A6E089BA77F48D7E1E652D59916E9DF9BBAA03CB76C81311A1");
```

图 8 自动化分析的 20 号报文 SessionKey 结构

另外在第 12 个数据包的环境中,自动化识别的划分粒度比人工识别的划分粒度更小,也导致自动识别报告的漏洞数量大于人工识别的数量.

### 7.3 其它协议测试分析

利用自动化网络协议漏洞挖掘方法,本文继续测试了 Oracle 数据库的 EM 和 ISQLPlus 协议. Oracle服务器本地可以通过 SQLPlus 登录数据库并执行数据库操作.为了方便管理,Oracle 服务器也提供了 HTTP 方式管理数据库.在 Oracle10G2 版本中,通过基于 HTTP 的 EM 协议登录远程服务器并执行管理操作,ISQLPlus 则是执行数据库语句的接口.

这两个服务都是基于 HTTP 的,而且传输过程没有加密,包括用户名和密码,本文仅对每个协议的第一个数据包进行了漏洞挖掘,测试结果表明协议自动识别能够正确地识别协议中的域.在 EM 协议中共测试出 2 个拒绝服务漏洞,在 ISQLPlus 中测试出 3 个拒绝服务漏洞.

## 8 结论和展望

本文描述了自动化识别网络协议并产生模糊器对网络协议进行 Fuzz Testing 的漏洞挖掘方法,并用这种方法对多种公开和未公开网络协议进行了测试.结论是这种方法具有快速、准确的优点.同时无需预先了解网络协议的语法,配置非常简单,能够节省大量的手工分析时间.在网络协议安全性检测、网络风险分析、模拟网络服务的蜜罐系统方面,具有重要的应用价值.本文主要做出的工作包含如下部分:首先提出了自动化网络协议 Fuzz Testing 的完整方案;其次,在网络报文多序列比对过程中提出了基于报文长度的渐进比对和连续匹配奖励的 Needleman Wunsch 算法,这两个方法在实验中比基于免疫的遗传算法要优越很多,适合网络报文的比对;对网络报文特殊域的识别做了较为深入的研究,例如长度域、Unicode 域和特殊分隔符等,并提出了二进制和文本网络协议应该区分处理,相比之前的研究成果更加深入和细致.最后,在测试部分,针对公开和未公开的 FTP、TNS、IM、ISQLPlus 进行了深入测试,给出了详细的测试结果.

在实验中,我们也发现本方法的效果依赖于采样报文本身的多样性,需要对网络协议多次采样,并尽量保证每次用不同参数来使用网络协议.如何以尽量少的采样次数得到最好的网络协议识别结果是我们未来需要深入研究的工作.

## 参 考 文 献


[1] Corrado Leita, Ken Mermoud, Marc Dacier. ScriptGen: An automated script generation tool for honeyd//Proceedings of the 21st Annual Computer Security Applications Conference (ACSA 2005). Tucson, USA, 2005: 202-214

[2] Cui Wei-Dong, Paxson Vern, Nicholas Weaver, Katz Randy. Protocol-independent adaptive replay of application dialog//Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS). San Diego, CA, 2006

[3] Cui Wei-Dong, Paxson Vern, Weaver Nicholas C. GQ: Realizing a system to catch worms in a quarter million places. International Computer Science Institute, Technology Report TR-06-004, 2006

[4] Cui Wei-Dong, Kannan Jayanthkumar, Wang Helen J. Discoverer: Automatic protocol reverse engineering from network traces//Proceedings of the 16th Usenix Security Symposium. Berkeley, CA, USA, 2007: 1-14

[5] Portokalidis Georgios, Slowinska Asia, Bos Herbert. Argos: An emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. ACM SIGOPS Operating Systems Review, 2006, 40(4): 15-27

[6] Newsome James, Song Dawn. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software//Proceedings of the NDSS'05. San Diego, California, USA, 2005

[7] Wondracek G, Kruegel C, Kirda E, Milani P. Automatic network protocol analysis//Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS). San Diego, California, USA, 2008

[8] Caballero Juan, Yin Heng, Liang Zhen-Kai, Song Dawn. Polyglot: Automatic extraction of protocol message format using dynamic binary analysis//Proceedings of the 14th ACM Conference on Computer and Communications Security. Alexandria, Virginia, USA, 2007: 317-329

[9] Comparetti Paolo Milani, Wondracek Gilbert, Kruegel Christopher, Kirda Engin. Prospex: Protocol specification extraction//Proceedings of the 2009 30th IEEE Symposium on Security and Privacy. Oakland, California, USA, 2009: 110-125

[10] Miller Barton, Fredriksen Louis, So Bryan. An empirical study of the reliability of UNIX utilities. Communications of the Association for Computing Machinery, 1990, 33(12): 32-44

[11] Forrester Justin, Miller Barton. An empirical study of the robustness of Windows NT applications using random testing//Proceedings of the 4th USENIX Windows System Symposium. Berkeley: USENIX Association, 2000, 4: 1-6

[12] Aitel Dave. MSRPC Fuzzing with SPIKE 2006. Technology Report, Immunity Inc, 2006

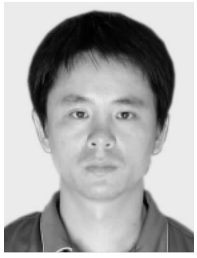
[13] Godefroid Patrice, Levin Michael, Molnar David. Automated whitebox fuzz testing//Proceedings of the 15th Annual Network and Distributed System Security Symposium. 2008: 1-16

[14] Wei Yu-Hao, Zhang Yu-Qing. MP3 vulnerability exploiting technique based on Fuzzing. Computer Engineering, 2007, 33(24): 158-160(in Chinese)
(魏瑜豪，张玉清. 基于 Fuzzing 的 MP3 播放软件漏洞发掘技术. 计算机工程，2007，33(24)：158-160)

[15] Liu Qi-Xu, Zhang Yu-Qing. TFTP vulnerability exploiting technique based on Fuzzing. Computer Engineering, 2007, 33(20): 142-144(in Chinese)
(刘奇旭，张玉清. 基于 Fuzzing 的 TFTP 漏洞挖掘技术. 计算机工程，2007，33(20)：142-147)

[16] Liu Li-Fang, Huo Hong-Wei, Wang Bao-Shu. PHGA-COFFEE: Aligning multiple sequences by parallel hybrid genetic algorithm. Chinese Journal of Computers, 2006, 29(5): 727-733(in Chinese)
(刘立芳，霍红卫，王宝树. PHGA-COFFEE:多序列比对问题的并行混合遗传算法求解. 计算机学报，2006，29(5)：727-733)

[17] Thomsen R, Fogel G B, Krink T. A clustal alignment improverusing evolutionary algorithms//Proceedings of the 4th Congress on Evolutionary Computation (CEC-2002). Honolulu, HI, USA, 2002: 121-126



**LI Wei-Ming**, born in 1975, Ph.D., lecturer. His main research interests include network architecture and network security.

**ZHANG Ai-Fang**, born in 1978, lecturer. Her main research interests focus on network security.

**LIU Jian-Cai**, born in 1985, master candidate. His main research interests focus on network security.

**LI Zhi-Tang**, born in 1951, Ph. D., professor, Ph. D. supervisor. His main research interests include network architecture, P2P network and network security.

**Background**

As we all know, network protocol robustness is the foundation of network security. How to test the security of a network protocol is remaining a difficult problem. Protocol Fuzz testing is a useful method by which we can find DoS, buffer overflow, format string and other kinds of serious vulnerabilities. But currently, Protocol fuzz testing is operated manually. Manual work needs priori knowledge and is notoriously slow.

In this paper, we present an automatic network protocol fuzz testing method, which greatly improves the efficiency of fuzz testing. Based on network traces, the method can automatically reverse engineer the structure of the network protocol more precisely than previous methods, and automatically produce SPIKE scripts to test the security of the network server. Experiments show that this method can work excellent and in some cases even exceed the results of manual analysis. It is an integrity and effective scheme to enhance the security of text and binary network protocols and is a part of the work to build a more reliable next generation Internet.