

2011 年全国大学生信息安全竞赛

参赛作品简介

作品名称： 一种新型隐秘访问系统

组 长： 麻凯

组 员： 廖湘科 吴一蔚 周雨晨

电 话： 15191880506

提交日期： 2011 年 6 月 5 日

摘 要

近年来网络及其相关技术飞速发展，给人们生活带来了极大的便利的同时，也在逐渐降低人们使用网络的技能要求，越来越多的人开始接触、使用攻击技术，使得网络攻击的数量飞速上升。

通过针对服务器开放端口的恶意扫描软件被广泛使用，对服务器进行恶意扫描从而引发了服务器安全问题层出不穷，多数的病毒和恶意软件常常利用开放端口对应服务的服务漏洞，进行相对应的攻击，达到窃取高级权限，破坏服务器相应服务，窃取资料的目的。

目前，建立链接的方式是用户主动向服务器的已知开放的端口发送建立链接请求，服务器回复该请求并建立链接，这种传统“三次握手”方式要求服务器长时间开放指定端口，造成了极大的安全隐患。

针对这一问题，本项目设计了一套隐秘访问系统，在服务器**不开放任何端口**的情况下，认证客户发送的数据包序列，并开放随机端口让指定用户访问，该端口对于其他非认证用户仍处于关闭状态。达到建立安全连接“无声无息”，即隐秘访问。

目录

摘要 2

作品介绍 5

 ● 认证并建立链接的过程..... 6

 ● 作品的安全性能..... 7

 ● 作品的总体简略框架图..... 8

一、研究背景 9

 1.研究现状 9

 ● 现有产品存在的问题 15

二、创新性 16

 1. 访问控制策略动态修改..... 16

 2. 带前向安全性的服务端口随机开放 17

 3. 基于 SHA 的新型序列生成算法..... 20

 4. 利用序列算法实现新型的双向身份认证..... 20

三、核心技术及实现方案 21

 1. 核心算法..... 21

 2. 动态访问控制模型..... 28

 3. 序列认证模型 32

 A. 数据包捕获模块..... 32

 B. 序列匹配模块 34

 C. 后台数据交互模块..... 37

四、性能测试 39

 1. 测试环境..... 39

 2. 测试项目..... 40

六、总结	48
------------	----

七、参考文献	48
--------------	----

作品介绍

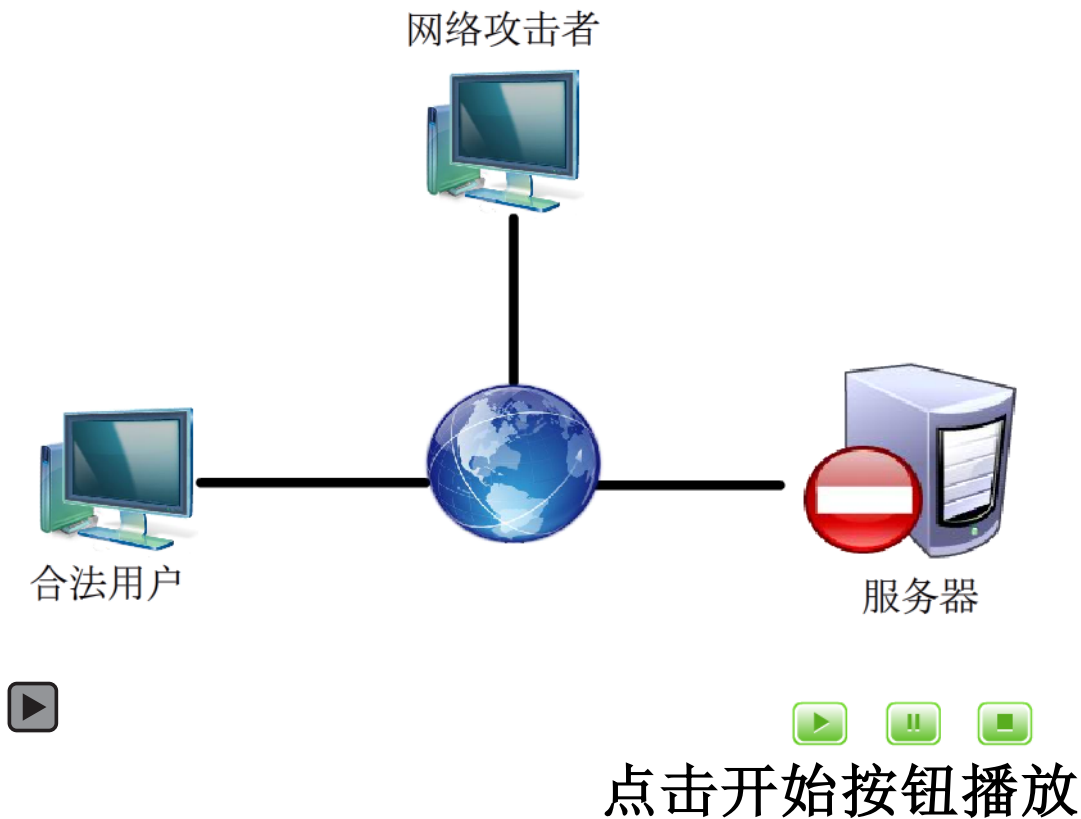
近年来，互联网的发展迅速，随之各种应用也得到了如火如荼的发展，而搭载着这些应用的服务器的安全问题却不容乐观。

目前的服务器长时间开放某个特定端口供用户建立链接，从而享用这项服务。这种状态极容易收到恶意软件的扫描攻击。非法用户可以监测固定端口以获取信息，可以建立链接而耗费系统资源并猜测密码 等等。对服务器而言，防火墙必须接受来自所有 ip 的链接，在接入后才通过密码来认证合法用户，管理员只有通过查看系统日志来添加抵制恶意访问的规则，过程极其繁琐。为了防止黑客监测固定服务的端口，管理员往往通过修改对应端口，使著名的服务不使用默认端口。但是这种方法必须通知所有合法用户新的服务端口，过段时间后再次重定向，过程也非常繁琐。而且服务器所面临的攻击远不止这些。

为了提高服务器对于这些攻击的防御能力并且在同时提供一种更为安全的访问控制机制，我们设计出隐秘访问控制系统。作品实现了：

- a) 服务器丢弃所有认证数据包，实现了对恶意扫描和恶意接入的有效的防御；当合法用户发来正确的认证序列，服务器动态改变访问控制策略，管理员不用手动添加规则；
- b) 固定的服务采用动态端口，用户每次使用服务，服务端口动态改变；管理员不用手动修改服务端口。黑客无法通过监测固定端口获取信息
- c) 认证序列具有伪随机性，实现了认证序列的不可预测；认证序列动态生成，具有前向安全性，有效抵制重放攻击；
- d) 在建立 tcp 连接后进行接入后认证，完成了对网关劫持或 ARP 欺骗等“冒充式”攻击的有效防御；
- e) 做完接入认证后，可以方便搭载各种网络应用；
- f) 断开连接后，删除该用户对用访问控制策略，恢复到初始状态，实现对虚假 IP 地址攻击的有效防御；
- g) 所有通信过程 AES 加密，保证信息的机密性；
- h) 客户端采用 Python 语言编写，方便跨越 windows Linux 和支持 Python 解释器的移动平台；

● 认证并建立链接的过程



Step 1: 合法用户需要访问服务器时，客户端中的根据用户输入的 KEY 和状态文件的内容计算出一组认证序列(20 个整数)，客户端用前 17 个整数充当 UDP 目标端口号，依次向服务器该端口发送 UDP 数据包。序列的生成算法改造自 SHA，具有前向安全性和伪随机性，无法推测出以后的序列。

Step 2: 服务器之前已生成多组序列，从网络设备驱动中捕获 udp 数据包并提取出目标端口号，与认证序列进行匹配。所有的数据包都在内核被过滤，服务器的应用程序不会收到任何数据包，有效抵抗恶意扫描和非法接入。

Step3: 若匹配成功，则服务器改变访问控制策略，不再过滤来自该用户 ip、端口号为第 18 个整数的数据包。从而允许该用户以该端口号建立 tcp/ip 链接。

Step4: 建立 tcp 连接后，客户端等待服务器回复第 19 个整数，以防止黑客冒充服务器。服务器等待客户端回复第 20 个整数，实现双向认证。使冒充式的攻击无法成功。

Step5: 完成双向认证后，客户端选择所需的服务，用户选择完成相应服务后，服务端把该端口重新定向到指定的服务程序，完成合法用户对该项服务的享用同时实现了服务端端口随机开放。

Step6: 服务运行完毕后，断开链接，服务器自主删除允许该用户访问该项服务策略，恢复到初始状态。

● 作品的安全性能

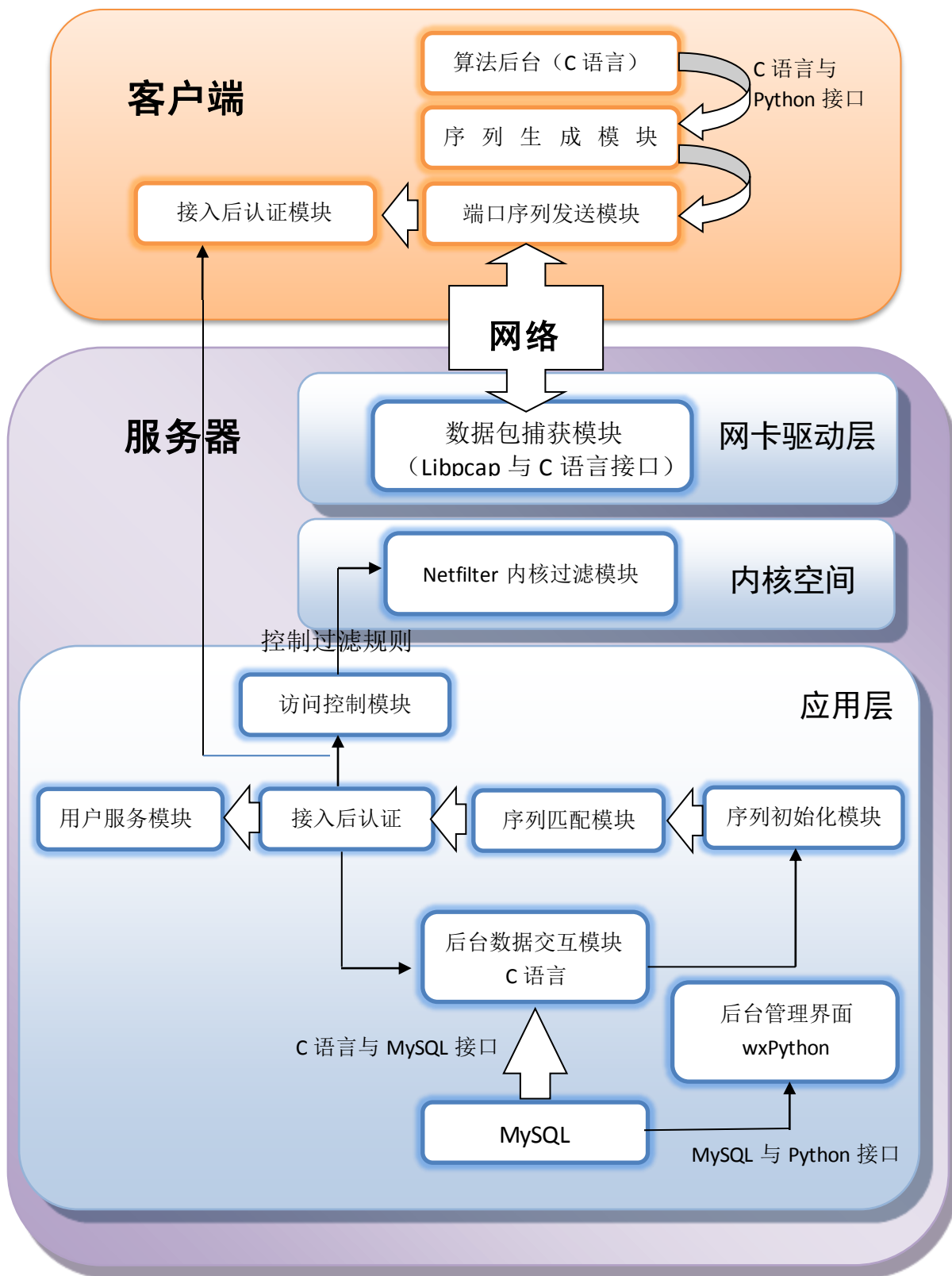
[1] UDP 端口号序列的猜测，需要从 65535^{17} 个端口序列猜测出一组序列，这样使得认证序列有很高的安全保障；

[2] 假设 UDP 端口号序列被成功猜测，服务器不会返回任何信息，该向哪个端口发送 SYN 数据包仍然不确定，被成功预测的几率是 $1/65535$ ；

[3] 即使猜测出认证序列，并且建立 TCP 链接，还有接入后认证，需要再次提供一个整数，猜猜对概率为 $1/65535$ ；

这种概率理论认为其不可实现。因为要冒充合法用户，概率为 $1/65535$ 的 20 次方。

● 作品的总体简略框架图:



一、研究背景

1.研究现状：

近年来网络及其相关技术飞速发展，给人们的生活带来了极大的便利的同时，也在逐渐降低人们使用网络的技能要求，越来越多的人开始接触、使用攻击技术，使得网络攻击的数量飞速上升。

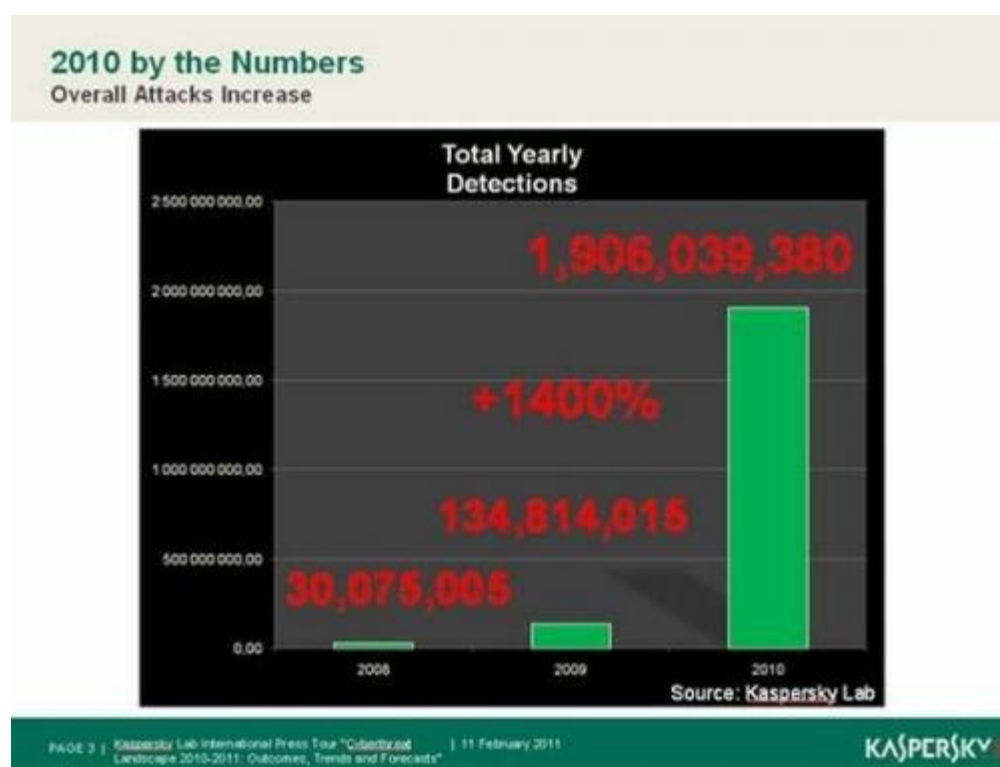


图 1 2008-2010 年网络攻击数量统计图

图 1 由卡巴斯基安全实验室提供的 2008-2010 年网络攻击数量统计图

通过针对服务器开放端口的恶意扫描软件被广泛使用，对服务器进行恶意扫描从而引发了服务器安全问题层出不穷，多数的病毒和恶意软件常常利用开放端口对应服务的已知或未知的服务漏洞，进行相对应的攻击，达到窃取高级权限，破坏服务器相应服务，窃取资料的目的。

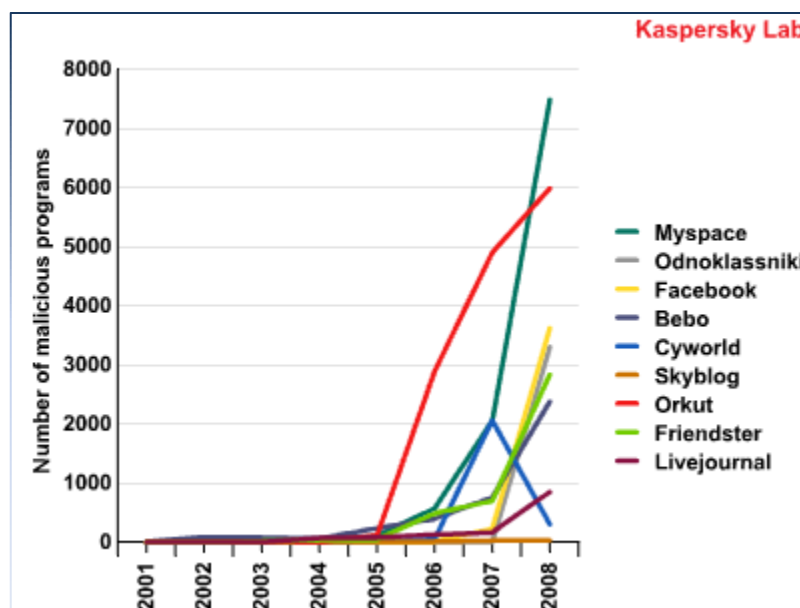


图 2 2001-2008 年某些知名网站日受到扫描等攻击的数量折线图

从安全管理角度来说，开启的服务端口越多，就越不安全，因此服务器在开放之前，会尽可能关闭一切不必要的端口，再对提供服务的端口做访问控制。而作为远程管理与维护的人员通常需要开启一些服务端口，如 FTP 或是 SSH。

若关闭这些服务，管理与维护人员就只能通过到达服务器所在地进行实地操作，这大大增加了网络管理人员的难度，浪费了网络管理人员的精力。

若开启这些服务，就要接受网络上接近无穷次的恶意扫描，暴力破解或是密码猜测，甚至是数据监听，这样使得网络管理员整天“提心吊胆，惶惶不可终日”，每天都必须仔细核对数十页，上百页甚至是数百页的管理日志，来检查远程管理用户是否是通过合法授权访问。简单的访问控制根本无法有效的控制、屏蔽这些攻击，反而还会给管理员的远程管理带来不必要的麻烦。

222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /UserReg.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /UserReg.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /lavery_EditWtlAdmin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /lavery_EditAdmin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /lavery_EditAdmin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /CmsEditor/ujkpadmin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /CmsEditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /CmsEditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /newsadmin/ubb/opebadmin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /newsadmin/ubb/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /newsadmin/ubb/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /asp_bin/webeditor/tdaadmin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /asp_bin/webeditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /asp_bin/webeditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /admin/webeditor/thcadmin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /admin/webeditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /admin/webeditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /manage/webeditor/ufbadmin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /manage/webeditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /manage/webeditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /webeditor/znlwadmin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /webeditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /webeditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /admin/SouthidcEditor/bdqiadmin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /admin/SouthidcEditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /admin/SouthidcEditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /ewindoweditor/cchxadmin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /ewindoweditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /ewindoweditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /eWebEditor/pvejadmin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /eWebEditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /eWebEditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /admin/eWebEditor/tomvadmin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /admin/eWebEditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /admin/eWebEditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /WebEdit/dzfmadmin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /WebEdit/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /WebEdit/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /admin/WebEdit/abuwadmin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /admin/WebEdit/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /admin/WebEdit/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /manage/eWebEditor/enyhadmin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"
222.189.238.53 -- [08/Mar/2011:05:18:45 +0800] "GET /manage/eWebEditor/admin_login.asp HTTP/1.1" 302 5 "-" "Mozilla/4.0"

图 3 服务器部分日志部分截图

图 3 为某服务器在一个小时内访问日志的 143 页中的某一页

而这些恶意扫描占用了大量原本提供给合法用户的带宽，致使合法用户在正常使用时服务质量大幅下降，消耗大量服务器资源。

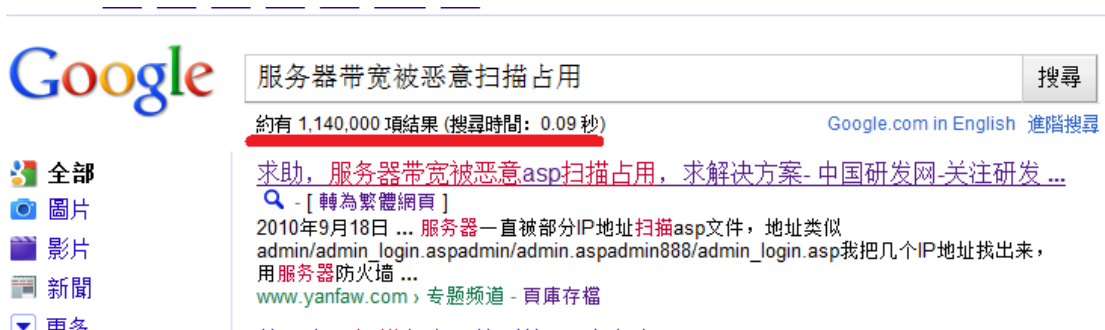


图 6 google 搜索带宽占用情况截图

图 6 为利用 Google 中搜索“服务器带宽被恶意扫描占用”关键字，搜索出 1140 万条相关信息，由此可见，服务器被恶意扫描软件攻击导致带宽被占用的情况非常普遍，管理员对此也十分头疼，束手无策，只能求助于广大网友。

为了能方便地建立链接以使用某项服务，人们在二十世纪中期提出用特定端口号默认标识某指定服务。由于该模式的简单便捷，得到大众的认可，一直沿用至今。

例如：

协议格式	端口号	默认对应服务
TCP	22	SSH (secure shell)
TCP	23	远程登录 (telnet)
TCP	80	超文本传输协议 (HTTP)
UDP	1027	HP 服务或 UC 聊天软件
UDP	1028	应用层网关服务

表格 1 服务默认端口号

现有的安全防护措施是给相应端口加安全措施例如密码等简单的安全认证，或者给服务器搭载防火墙等安全软件，企图达到有效控制端口安全的措施。但是也都存在大量漏洞，例如：

- 流光 5.0 等大量共享软件已经在网络上大量流传，利用其提供的功能，能深度扫描，暴力破解管理员密码；

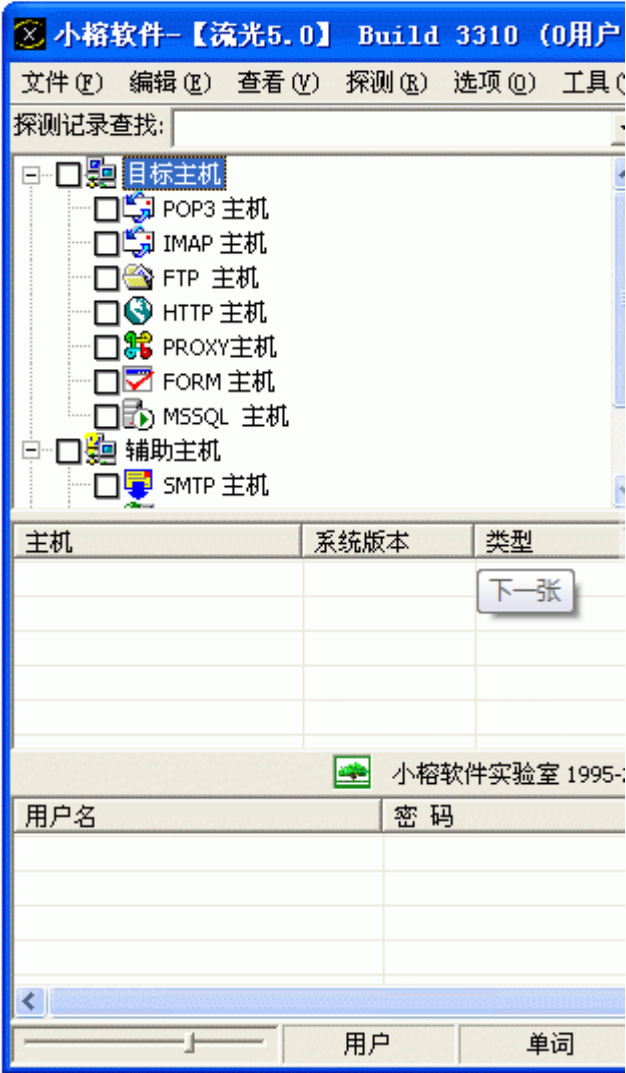


图 7 流光 5.0 版软件界面部分截图

- 防火墙也只是根据设定好的规则检查数据包并决定丢弃还是放行，这样的过程极容易受到一些黑客技术攻击，从而绕过防火墙等安全软件的防护，进行一些非授权的操作；



图 8 瑞星防火墙访问规则设置页面

而现有的服务器相关防护的安全软件，在广泛使用的同时，其漏洞等一系列不安全因素不断被发觉、熟知，即使在安全补丁不断更新，安全软件进一步完善的情况下，软件的安全性能仍然在不断降低。

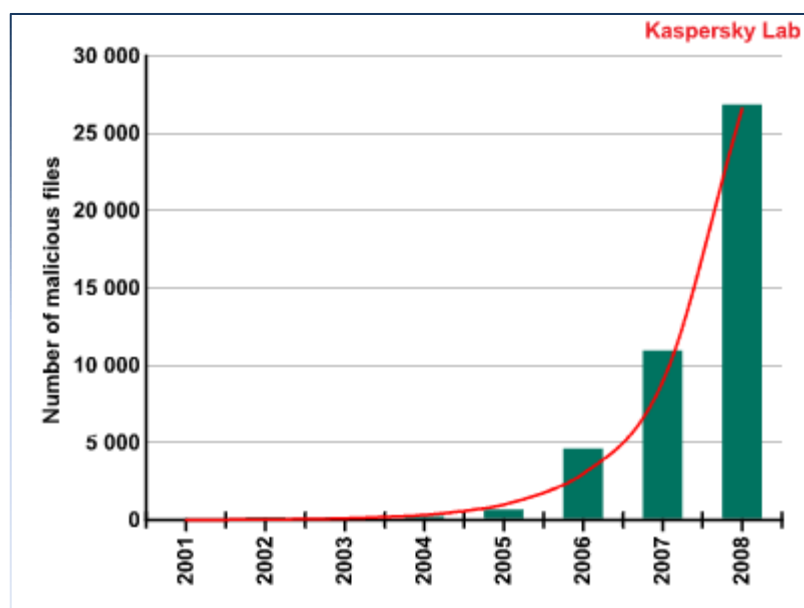


图 9 每年针对某特定服务的恶意攻击软件增长数量

由此可见，服务器就需要更全面，更安全的访问机制来保护主机及其所搭载的应用不受非授权用户的访问或者恶意攻击。隐秘访问控制系统因此而诞生，创新性地给服务器提供一种能自主修改的、隐蔽的、防御恶意扫描和其他恶意攻击的访问控制机制，去更好的提供对于服务器，搭载软件及其数据的安全保障。

端口碰撞技术是一种允许服务设备在用户按照约定的序列碰撞后，打开一个约定的服务端口提供服务的技术。所谓碰撞是由一个尝试访问系统中关闭端口的序列组成，也就是特定端口的连接请求。

● 现有产品存在的问题：

市面上的端口碰撞产品的设计，端口认证采用端口号的固定组合方式，他们的设计类似口令认证，安全性较差。端口序列本身没有含义，由使用者自己设置，用户按照这个指定序列向服务器发送数据包，服务器将收到的认证序列和记录于服务器内的序列进行比对，当全部序列成功认证后允许建立链接。

● 我们的创新

所以市面上的这个认证模式后隐藏着较多的不安全因素，例如：

- [1] 如果网络被人监视，监听者记录数据包，然后发动重放攻击，可以直接绕过访问控制，让安全机制形同虚设；
- [2] 访问控制规则过于简单，恶意监测者容易找到规律；
- [3] 不带双向认证过程，无法抵抗冒充式攻击；

隐秘访问控制系统，利用算法实现端口认证的随机性，让监听者无法成功预测出下一次的认证序列，实现了前向安全。

二、创新性

1. 访问控制策略动态修改

访问控制是按照用户的身份及其所归属的某预定义的组来限制用户对某些信息的访问，或限制对某些控制功能的使用。主要用于管理员控制用户对服务器、目录、文件等网络资源的访问。

目前的网络依靠防火墙控制，网络端口和节点的安全控制等几大访问控制策略来完成对网络的安全加固。

防火墙只是按照管理员设定的一系列安全策略来进行相应的数据过滤达到网络控制，但是预先配置固定的规则完全没有灵活性，无法适应多种情况：如对外出工作人员提供服务的服务器，必须接受陌生 ip 的接入，在接入后再通过密码识别合法用户，这样易受到恶意接入而浪费系统资源。

而**隐秘访问控制系统可以根据用户的认证情况随时修改相应的访问控制策略，达到动态访问控制，可以应对大多数情况，进而更好的达到对网络接入实时控制，加固网络安全的目的**。同时由于在初始状态时，默认丢弃所有数据包，大大节省了管理员查看数十页甚至上百页日志过程和添加相应规则以应对恶意攻击者的环节。

在客户端向服务器认证过程中，由于认证序列是基于用户持有 KEY 和当前状态联合计算得出，类似于伪随机数，认证序列相同概率极低，不会有相同认证序列产生。并且在发送数据包序列时不会携带任何有关用户信息的数据字段，就让恶意攻击者更难从所截获的数据包推算出认证序列，甚至无法知道登录用户是谁。

2. 带前向安全性的服务端口随机开放

由于 TCP 协议由端口号唯一对应某项服务，以前为了方便，端口号和特定服务做了默认映射，这样做存在着大量隐患，基于固定开放端口扫描的攻击层出不穷。

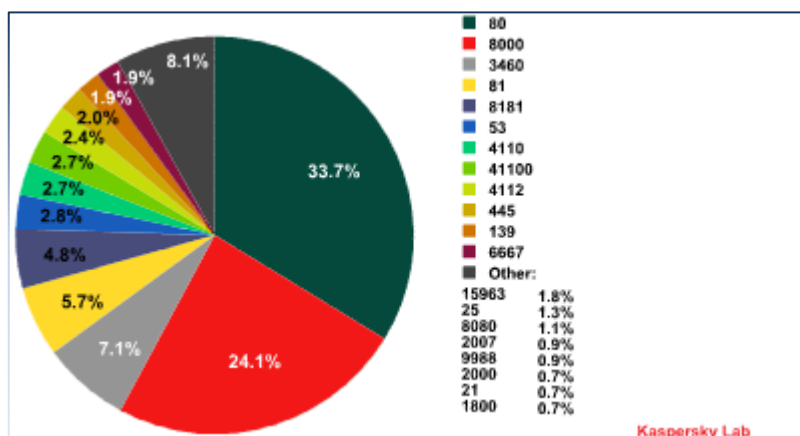


图 10 近几年恶意软件实现攻击端口分布图

由图 10 可知，恶意软件很大程度上依赖于默认端口映射，攻击的往往是著名服务的默认端口，以利用其所搭载服务的漏洞。

默认端口映射存在巨大隐患，这样，管理员就不得不在开启该项服务和保证服务器安全中抉择。

● 目前针对端口攻击的防御方式之一 ——手动修改端口号。

由于服务程序存在着较多的不足和缺陷带来了很大的隐患，一些网络管理员提出更改这些知名或是不知名端口号所映射的服务。例如：将 SSH（专为远程登录会话和其他网络服务提供安全性的协议，端口号：22）的默认端口号修改为 9999。

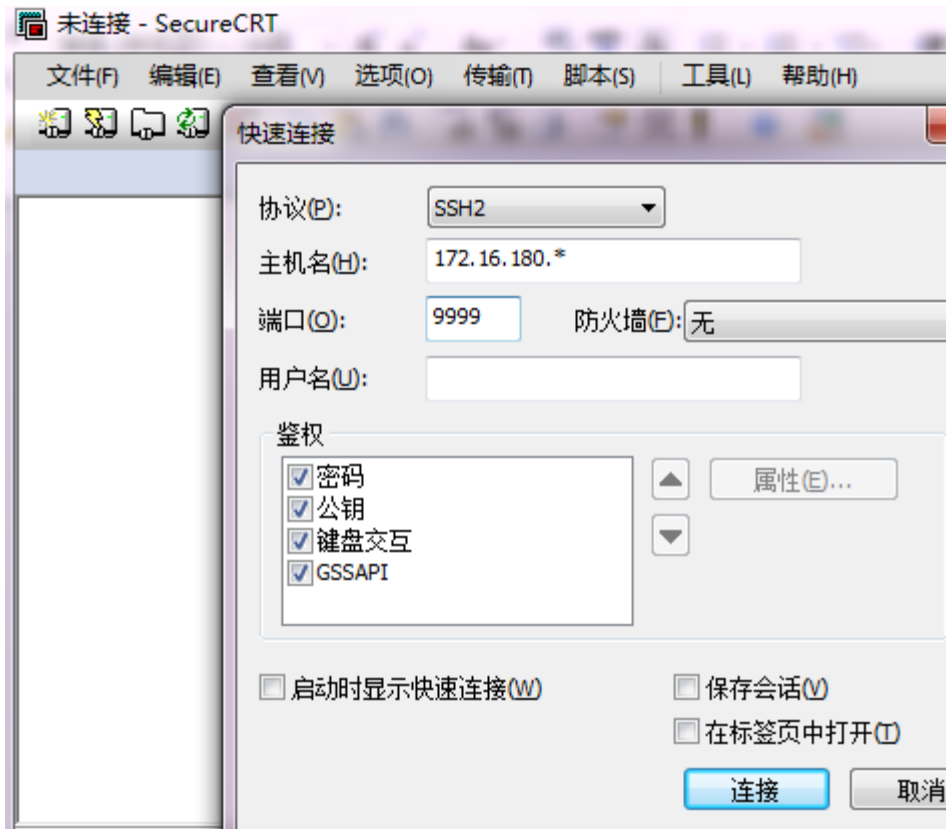


图 11 某软件手动修改链接端口设置页面

● 存在问题：

1. 这样做虽然能避免掉一些由恶意扫描带来的不安全因素，但是仍然存在着很大隐患，例如一个有经验的网络监听者就能从一些截取到的数据包并分析出服务器正在向合法用户提供何种服务，从而完成对该端口的攻击。

2. 手动修改端口号还需要跟其他管理员或是用户协商，这样又极大的增加了管理员的工作强度，增加了不必要的麻烦。

● 解决办法

手动跟换端口号的做法不能有效防御这种攻击，然而在隐秘访问系统中，我们提出的动态随机开放端口却能从很大程度上避免这种攻击，而且实现了动态端口的前向安全特征。

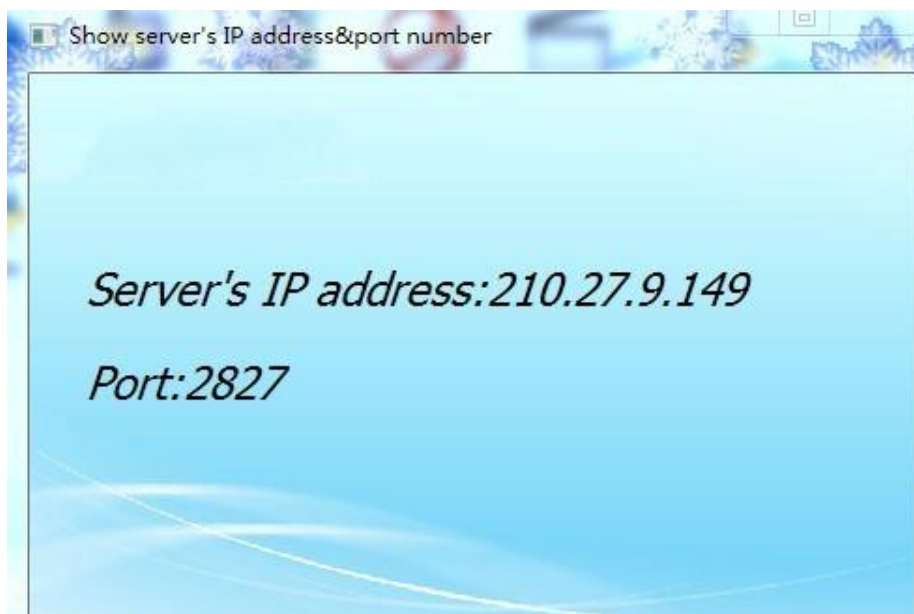


图 12 本软件产生随机端口号

在客户端和服务端计算出认证序列后，按照相同算法再计算出一个 16 位长整数，它就是这次建立链接的端口号，服务器和客户端通过本系统将数据提供给服务程序，从而隐藏了服务器提供给合法用户的服务。

假设有经验的网络监听者截获并成功得出数据包中包含的意义，进而发现是何种服务。由于端口号产生的伪随机性，下一次登录使用的端口号极难被猜出，从而下一次开启该服务的端口成为无法预知的，实现了前向安全。

假设即使下一次开启该服务的端口号被成功预测（概率 $1/65535$ ），由于访问控制策略中已经添加只接受来自合法用户地址的数据包，丢弃一切非合法用户数据包，致使该监听者无法攻击该服务。合法用户断开连接后，删除该用户一切相关策略，从而达到对于任何用户都拒绝服务的状态，进而从很大程度上避免了该种攻击。

3. 基于 SHA 的新型序列生成算法

SHA 全称安全散列算法，由美国国家标准和技术局发布，由于其良好的雪崩效应和更长的摘要等一系列特性，并且经过不断改进，现已经成为公认的最安全的散列算法之一，并被广泛使用。

由于 SHA 算法是接收一段明文，然后以一种不可逆的方式将它转换成一段定长的密文，也可以简单的理解为取一串输入码（称为预映射或信息），并把它们转化为长度较短、位数固定的输出序列即散列值（也称为信息摘要或信息认证代码），该算法已经开源供大众使用。

由于 SHA 算法可由上一状态直接计算出下一个状态，而不依赖其他的输入。假设非法用户通过某种方式窃取到用户的某一个状态，就可以通过该算法不断计算，计算出该用户的所有认证序列，从而获得该用户的权限，进行某些非授权操作。所以我们针对本项目的需要，基于 SHA 改造出合适的序列生成算法。

本开发小组利用 SHA 算法对用户输入的 KEY 进行处理，再经过一系列类似于伪随机算法的计算，得到多组接近于随机的认证序列。而每组认证序列的计算都要由缓存状态和原始 KEY 共同决定，缺一不可。假设攻击者窃取了某一个中间状态，因为不知道原始 KEY，该攻击者仍然不能计算出后续认证序列。这样就实现了对用户权限更安全的保护，进一步提高了该程序的安全性能。

4. 利用序列算法实现新型的双向身份认证

现如今，网络攻击迅速地发展，出现了例如网关劫持，ARP 欺骗等一系列“冒充式”网络攻击。

在传统的认证接入模式中，如果出现这种“冒充式”的网络攻击，攻击者切断合法用户与服务器的网络链接，修改、转发合法用户（或服务器）向服务器（或是客户端）的认证数据，使得攻击者自己顶替合法用户（或合法服务器），获得

对方信任，进行一系列非授权操作。

这样的攻击方式的出现造成了极大的安全隐患。

现有的防御这种攻击的方式是 挑战/应答方式的身份验证，该机制是每次认证时认证服务器端都给客户端发送一个不同的“挑战”字串，客户端程序收到这个“挑战”字串后，利用秘密密钥对挑战字串进行计算，做出相应的“应答”。然后客户端再向服务器发送挑战。

挑战应答机制虽然有效的防御了这种攻击，但是由于计算方式等一系列因素，反复进行这种较为复杂的运算，进行认证，就可能会给网络、客户和认证服务器带来太大的开销，效率并不高。

本开发小组基于本程序的认证算法，在计算出认证序列后，从该认证序列中抽出两个整数充当认证密码，并且保证这两个整数不会在建立链接前被发送。可以快速完成双向的接入认证，不用做多余的计算。

在服务器允许建立 TCP 链接，并且成功建立链接后，由服务器率先发送第一个认证密码，若被客户端认证，再由客户端发送另外一个认证密码，请求服务器认证。若两个认证过程任何一个没有认证成功，该 TCP 链接都会被中断。这样，就实现了客户端和服务器的双向认证，从很大程度上避免了该种“冒充式”攻击。

三、核心技术及实现方案

1.核心算法

A、安全散列算法(SHA)

安全散列算法（Secure Hash Algorithm）能计算出一个数字消息所对应到的，长度固定的字符串（又称消息摘要）。且若输入的消息不同，它们对应到不

同字符串的机率很高；而 SHA 是 FIPS 所认证的五种安全散列算法。这些算法之所以称作“安全”是基于以下两点（根据官方标准的描述）：

- 1) 由消息摘要反推原输入消息，从计算理论上来说是很困难的。
- 2) 想要找到两组不同的消息对应到相同的消息摘要，从计算理论上来说也是很困难的。任何对输入消息的变动，都有很高的机率导致其产生的消息摘要迥异。”

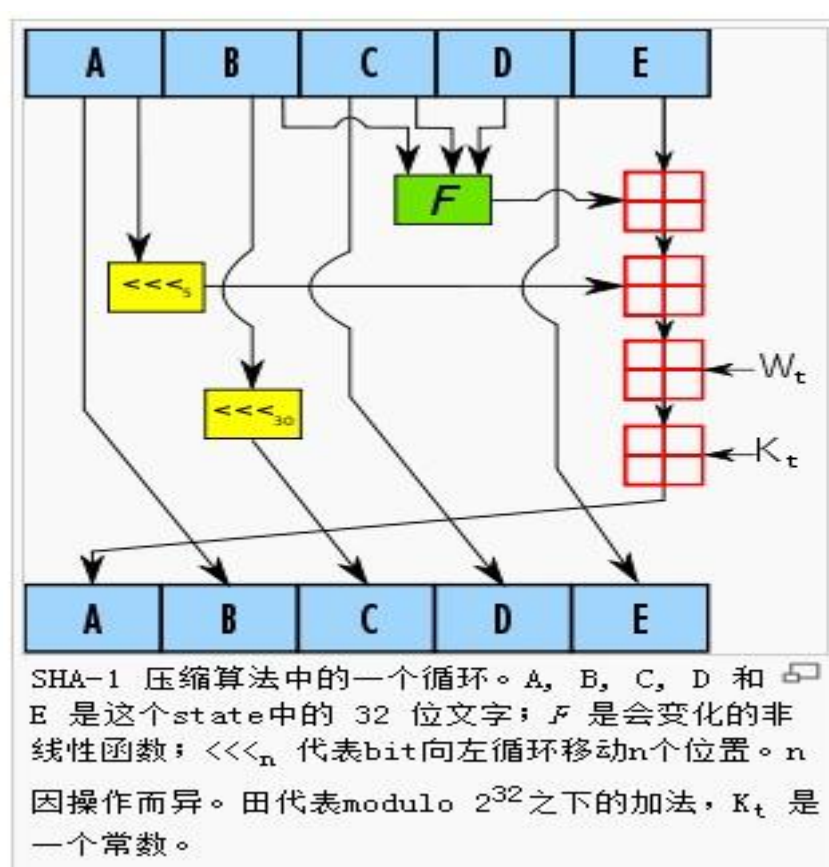


图 13 SHA 压缩算法单一循环示意图

● SHA 安全性能

我们采用了 SHA-256 和 SHA-1，最显著和最重要特点是 SHA 的报文摘要更长。使用强行技术，产生任何一个报文使其摘要等于给定报摘要的难度对 SHA 是 2^{160} 数量级的操作。这样，SHA-1 对强行攻击有更大的强度。并且，由

于设计，使其更不容易受到密码分析攻击。现在，SHA 已成为公认的最安全的散列算法之一。

由于 SHA 算法可由上一状态直接计算出下一个状态，而不依赖其他的输入。假设非法用户通过某种方式窃取到用户的某一个状态，就可以通过该算法不断计算，计算出该用户的所有认证序列，从而获得该用户的权限，进行某些非授权操作。所以我们针对本项目的需要，基于 SHA 改造出合适的序列生成算法。

我们的序列产生算法因为改造自 SHA 算法，所以每一次认证所用的序列必须保证随机性，而且保证不可推导出 key 以及下一组序列，实现了前向安全属性。

● 处理速度

由于在设计上，SHA 是模仿 MD4 设计的，但是由于更长的报文摘要，略微慢于 MD4，但仍然是高效 hash 算法。由于产生序列进行的计算较多，而客户端我们采用的 Python 语言比 C 语言慢，在 SHA 算法和序列生成算法的部分我们用 纯 C 实现，通过 Python 与 C 语言的接口实现数据传递。使客户端兼顾了跨平台无缝移植和注重效率的优点。

B、序列生成算法

● 变量简介

K	用户用来识别身份的 KEY，由管理员和用户共同创建，密钥分发是面对面分发。
S	用户当前状态，由 K 生成，每次计算会产生新的 S。

● 序列产生计算过程

Step1: 用户输入 K;

Step2: 检查存储 S 的缓存文件是否存在, 若不存在, 至 step3; 若存在, 跳至 step4;

Step3: 创建缓存文件, 通过 K 计算生成初始 S 并存入缓存文件中, 再回到 Step2;

Step4: 由 K 和 S 共同计算出认证序列和新的缓存状态;

Step5: 若认证通过, 将新的状态存入缓存文件中记录当前状态;

注: 服务器产生的序列与客户端一致, 服务器在初始时已经产生了序列, 在客户端序列到来时直接匹配, 以提高效率。

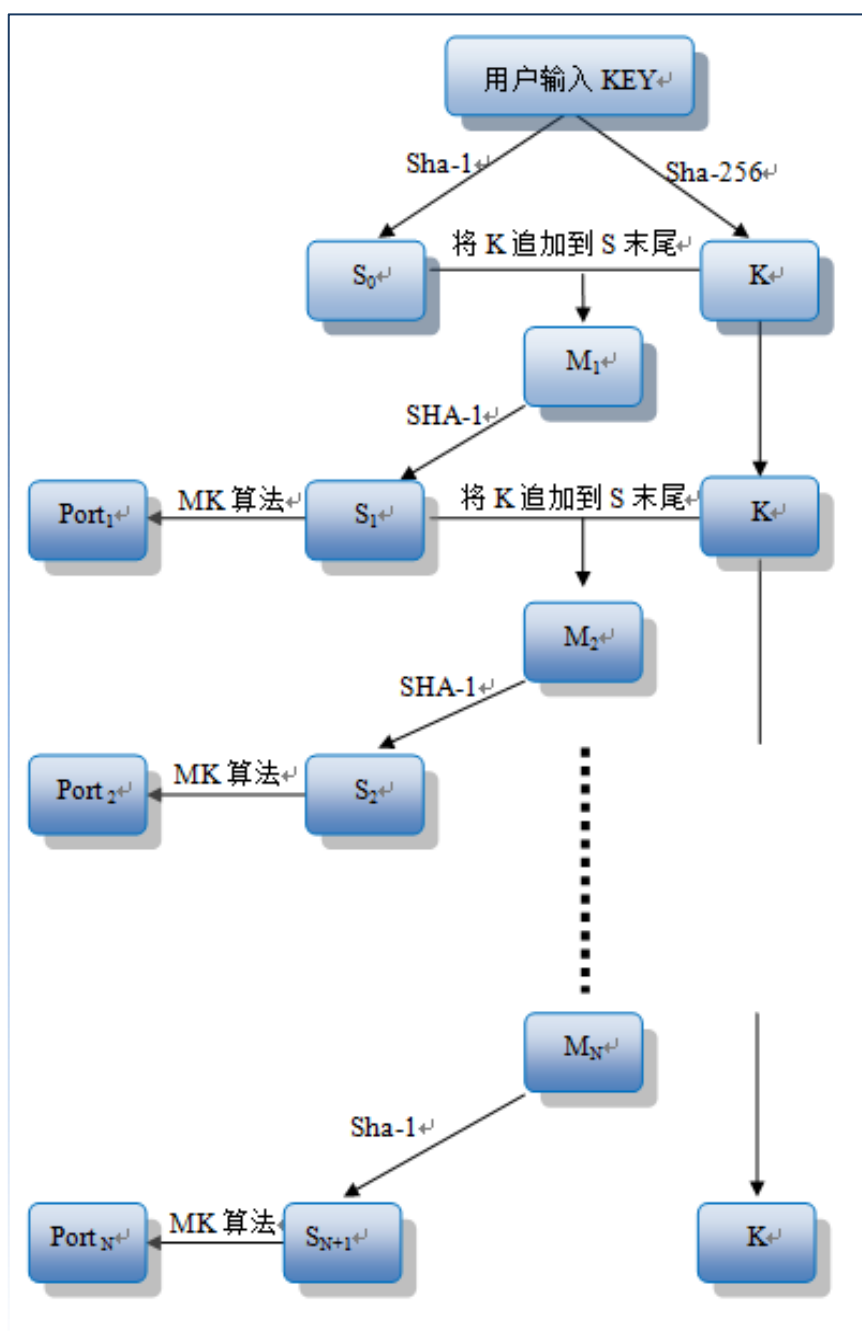


图 14 本软件序列生成算法流程图

图 14 为算法具体流程，S 为状态。可见，如果黑客通过恶意手段获得中间状态 S，因为没有初始 key，就无法通过 sha-256 算出 K，从而无法知道后续序列。SHA-1 极难逆推出消息原文，所以由 S 无法推出 M。

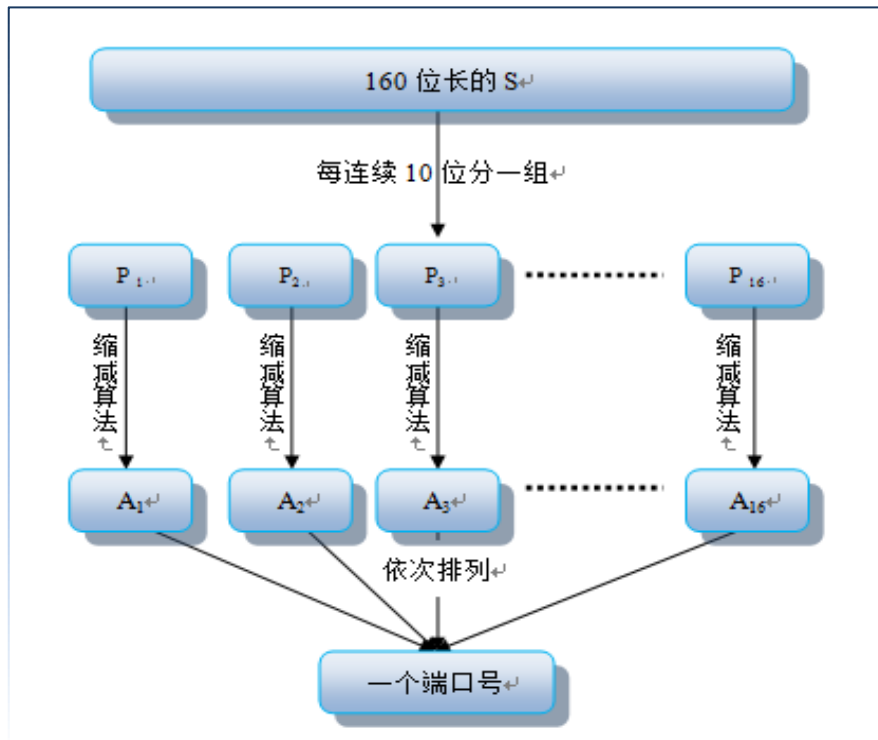


图 15 MK 算法流程图

图 15 为 MK 算法流程。缩减算法就是在数 P 中，计算出有多少 bit 是“1”，然后就将 P 的第几 bit 抽出，赋值予 A ， A 是 01 bit 位。16bit 组成新的短整数。由图可见，由于 SHA-1 运算结果的伪随机性， P 中 bit 位是 1 的个数也具有伪随机特征，从 P 到 A ，90% 的数据已被抛弃，无法由监听到的端口号来逆推出 P 和 S 。

图 14 与图 15 为认证序列产生的详细流程，可以看出，每计算出一个整数就会产生一个新的 S ，当计算出一组序列（20 个整数）后，最后的状态 S 作为后续序列所需状态而被保存。

C、AES 算法

密码学中的高级加密标准（Advanced Encryption Standard, AES），又称 Rijndael 加密法，是美国联邦政府采用的一种区块加密标准。这个标准用来替代原先的 DES，已经被多方分析且广为全世界所使用。经过五年的甄选流程，高级加密标准由美国国家标准与技术研究院（NIST）于 2001 年 11 月 26 日发布为 FIPS PUB 197，并在 2002 年 5 月 26 日成为有效的标准。2006 年，高级加密标准已然成为对称密钥加密中最流行的算法之一。

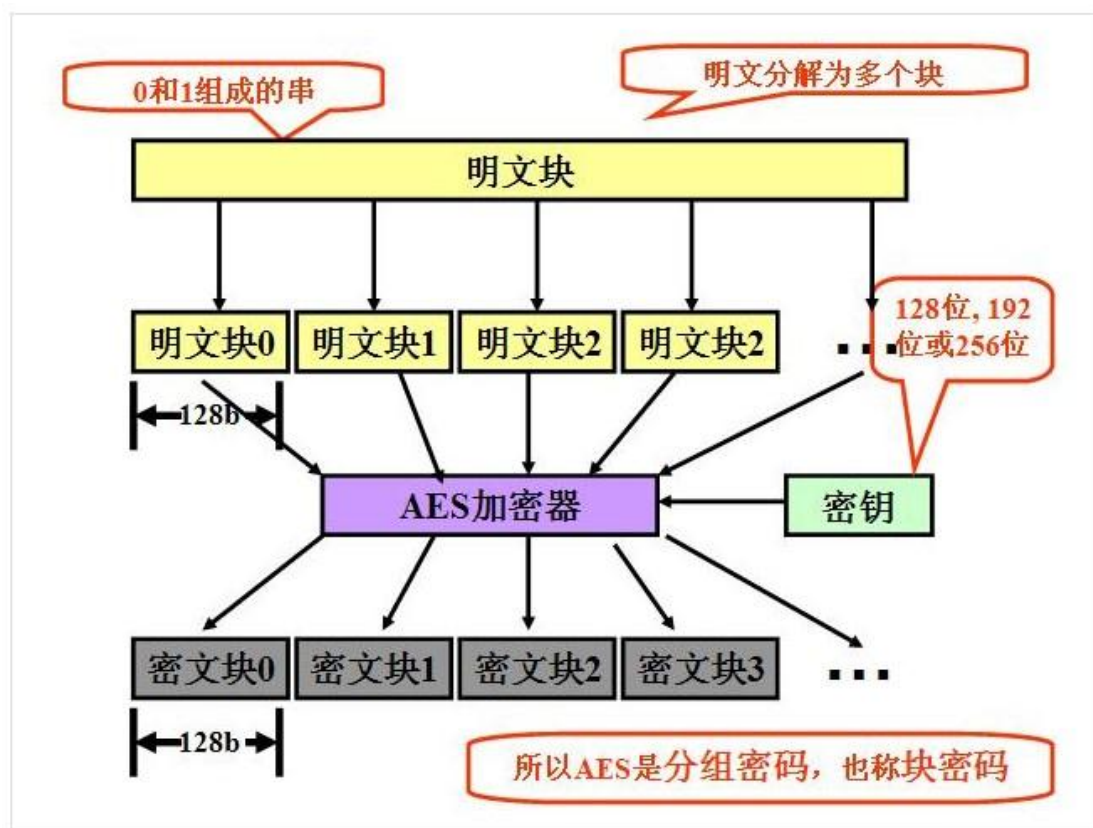


图 16 AES 算法简介

在用户与服务器应用程序进行通信的过程，数据全部使用 AES 进行加密，以保证信息的机密性。我们采用密钥为 256 位的 AES。

2. 动态访问控制模型

- 基于 `netfilter` 的服务端数据过滤

`netfilter` 组成 Linux 平台下的内核网络包过滤模型。 `netfilter/iptables` IP 信息包过滤系统是一种功能强大的工具， 可用于添加、编辑和除去规则， 这些规则是在做信息包过滤决定时， 防火墙所遵循和组成的规则。这些规则存储在专用的信息包过滤表中， 而这些表集成在 Linux 内核中。 在信息包过滤表中， 规则被分组放在我们所谓的 链（*chain*）中。

`netfilter` 组件也称为 内核空间（*kernel space*）， 是内核的一部分， 由一些信息包过滤表组成， 这些表包含内核用来控制信息包过滤处理的规则集。`iptables` 组件是一种工具， 也称为用户空间（*userspace*）， 它使插入、修改和除去信息包过滤表中的规则变得容易。

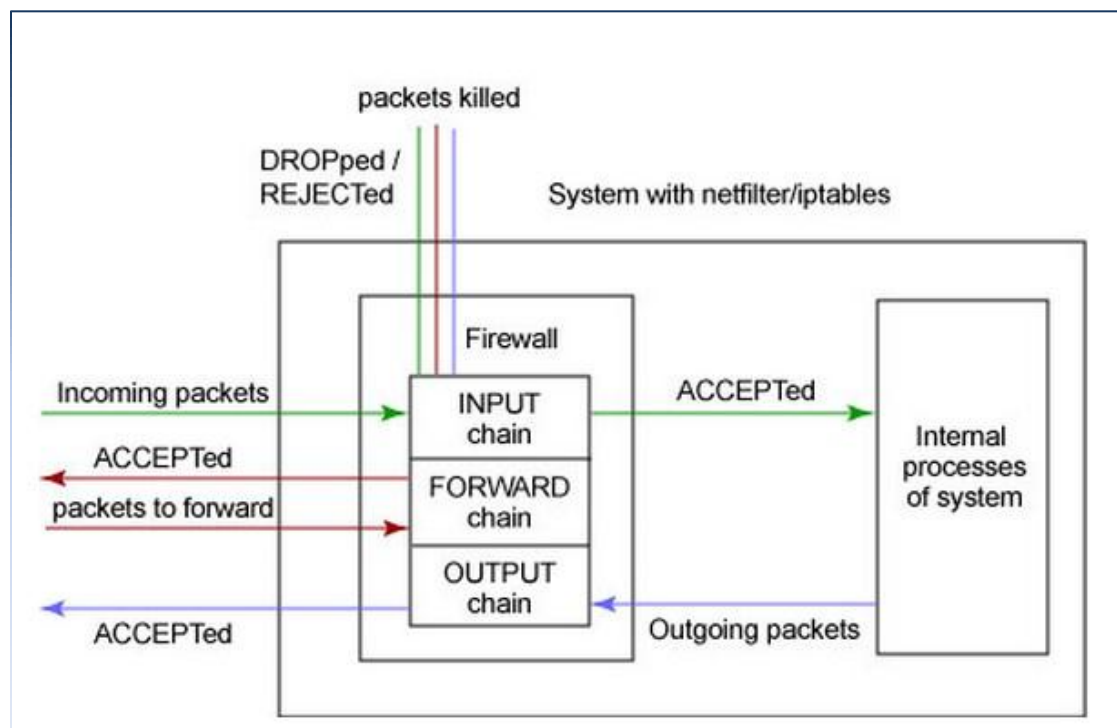


图 17 netfilter 数据包过滤过程

在初始化过程中，定义 netfilter 过滤（drop）所有的数据包，所有数据包在内核被过滤，无法到达应用层。令所有应用服务不会受到恶意链接和恶意扫描的困扰。

当用户发来的 UDP 包的目标端口序列与合法序列匹配时，程序自动修改信息包过滤表，实时动态地改变访问控制策略。不再过滤该合法用户的 ip 与某一个端口通信的数据包，使之能与该端口建立 tcp 链接。该端口由认证序列确定，认证序列中的第 18 个整数作为此次登录使用的服务端口。

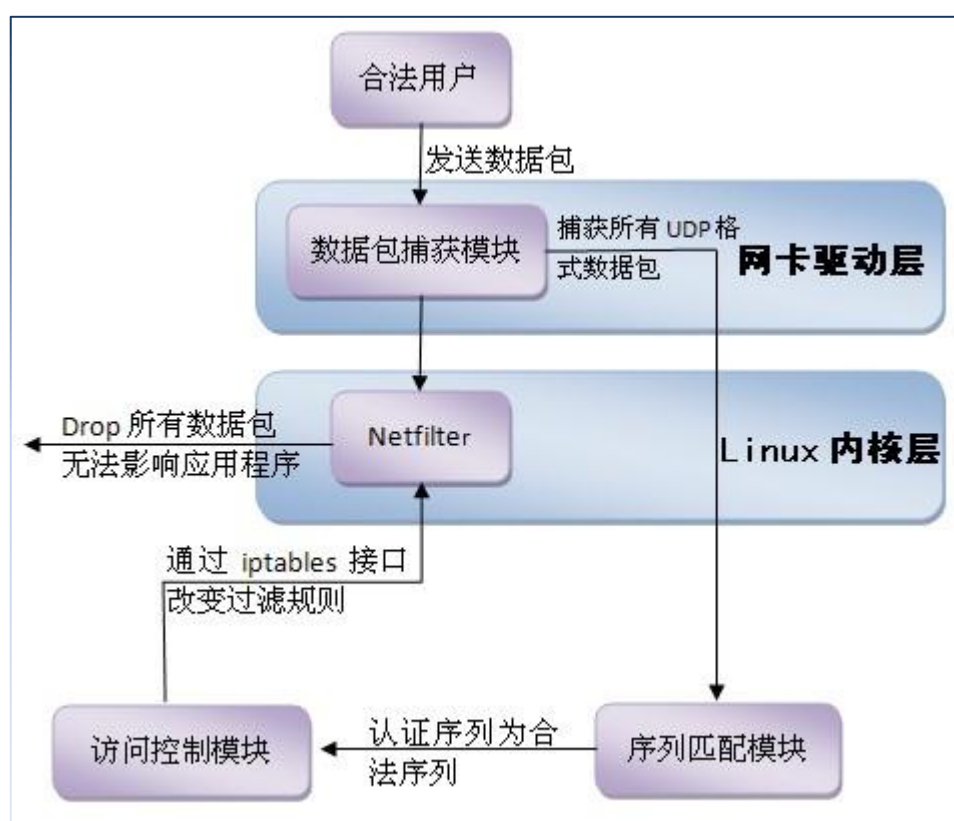


图 18 合法用户通过访问控制示例

假设用户成功通过这一步后，服务器允许该用户访问端口并成功建立 tcp/ip 链接。

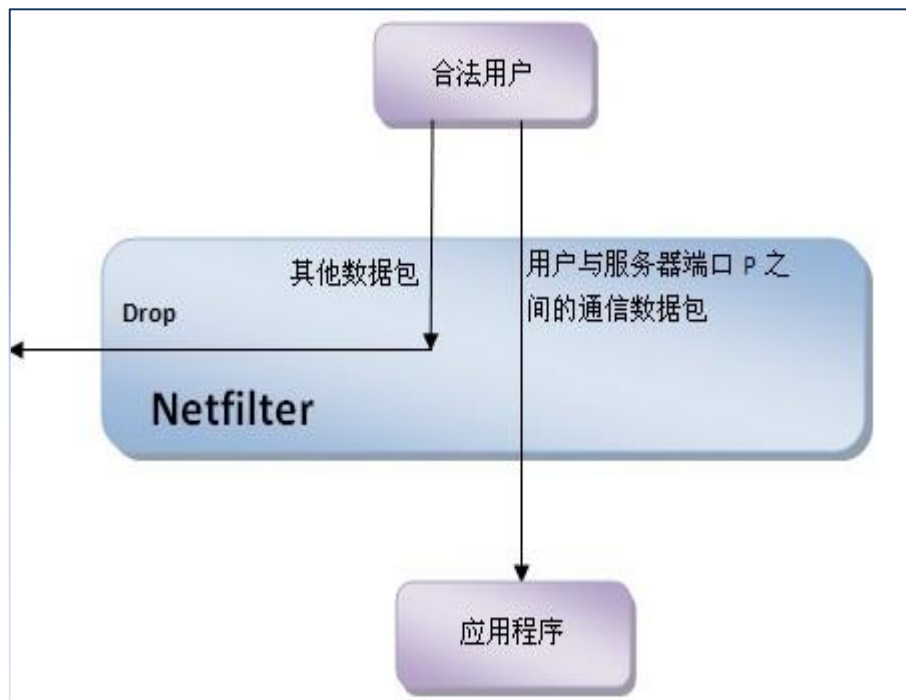


图 19 合法用户成功建立链接

现在用户可以使用端口 P 与服务器正常通信，在这一步该应用程序其实是接入后认证的程序。

在完成接入认证以后，服务器会让用户选择相应的服务应用程序，比如 SSH ftp 等等，对这些著名服务进行随机端口访问的实现策略是动态的端口重定向。

假如上图的例子在认证成功后用户选择 SSH 服务，则程序再次修改访问控制规则：

Step 1. 将用户与服务器端口 P 的通信过滤，相当于恢复到没有认证的状态。

Step 2. 添加一条 NAT 规则，将用户 ip 以 22 端口为目标地址的数据包重定向到一个随机端口，使得用户不能通过 22 端口访问 SSH 服务。

Step 3. 将用户 ip 以 P 为目标端口的数据包重定向到 22 端口。

Step 4. 添加一条规则，允许用户 ip 以 22 为目标端口的数据包不被过滤。

在做了以上 4 步之后，可以实现：

1. 用户不能通过 22 端口访问 SSH 服务，因为这些数据包会被重定向到随机端口，而被 netfilter 给过滤掉；
2. 用户以端口 P 可以成功使用 SSH 服务；
3. 因为除使用目标端口 P 之外，所有数据包仍然不能通过 netfilter，用户使用的端口 P 又被重定向到 SSH 服务的端口，所以用户只能使用 SSH 服务，而不能利用开放的 P 试图使用其他服务；

通过我们的动态访问控制技术，服务器对恶意用户的扫描和端口检测都有相当强的免疫力。管理员不用手动地添加防火墙过滤规则，不用为了找出恶意用户而查看成堆的系统日志，不用手动添加端口重定向规则和与用户协商端口。大大降低了系统管理员的工作量。

3.序列认证模型

A. 数据包捕获模块

——基于 libpcap 的认证数据包捕获

● 数据过滤流程摘要

Step1: 利用 BPF 算法对网卡接收到的链路层数据包进行过滤;

Step2: 利用旁路处理技术捕获所需数据包;

Step3: 将所需数据包的相关信息取出,追加到存储这些信息的双端队列的末尾;

● 组成部分

libpcap 主要由两部份组成: 网络分接头 (Network Tap) 和数据过滤器 (Packet Filter)。网络分接头从网络设备驱动程序中收集数据拷贝, 过滤器决定是否接收该数据包。

● 过滤算法

Libpcap 利用 BSD Packet Filter (BPF) 算法对网卡接收到的链路层数据包进行过滤。BPF 算法的基本思想是在有 BPF 监听的网络中, 网卡驱动将接收到的数据包复制一份交给 BPF 过滤器, 过滤器根据用户定义的规则决定是否接收此数据包以及需要拷贝该数据包的哪些内容, 然后将过滤后的数据给与过滤器相关联的上层应用程序。

● 包捕获机制

libpcap 的包捕获机制就是在数据链路层加一个旁路处理。当一个数据包到达网络接口时，libpcap 首先利用已经创建的 Socket 从链路层驱动程序中获得该数据包的拷贝，再通过 Tap 函数将数据包发给 BPF 过滤器。BPF 过滤器根据用户已经定义好的过滤规则对数据包进行逐一匹配，匹配成功则放入内核缓冲区，并传递给用户缓冲区，匹配失败则直接丢弃。如果没有设置过滤规则，所有数据包都将放入内核缓冲区，并传递给用户层缓冲区。

● 该模块使用的数据结构介绍

cap_info	存储一个 UDP 数据包的源 IP 和目标端口号；
Deque	以 cap_info 为元素组成的双端队列；

```
struct cap_info
{
    u_long sip; // 存储源 ip 地址
    u_short dport; // 存储目标端口号
};
```

服务器的 BPF 过滤器过滤掉除了目的地址为自己且传输协议为 UDP 的一切数据包，并且将所捕获到的数据包信息（cap_info 结构）追加到 Deque 的末尾。

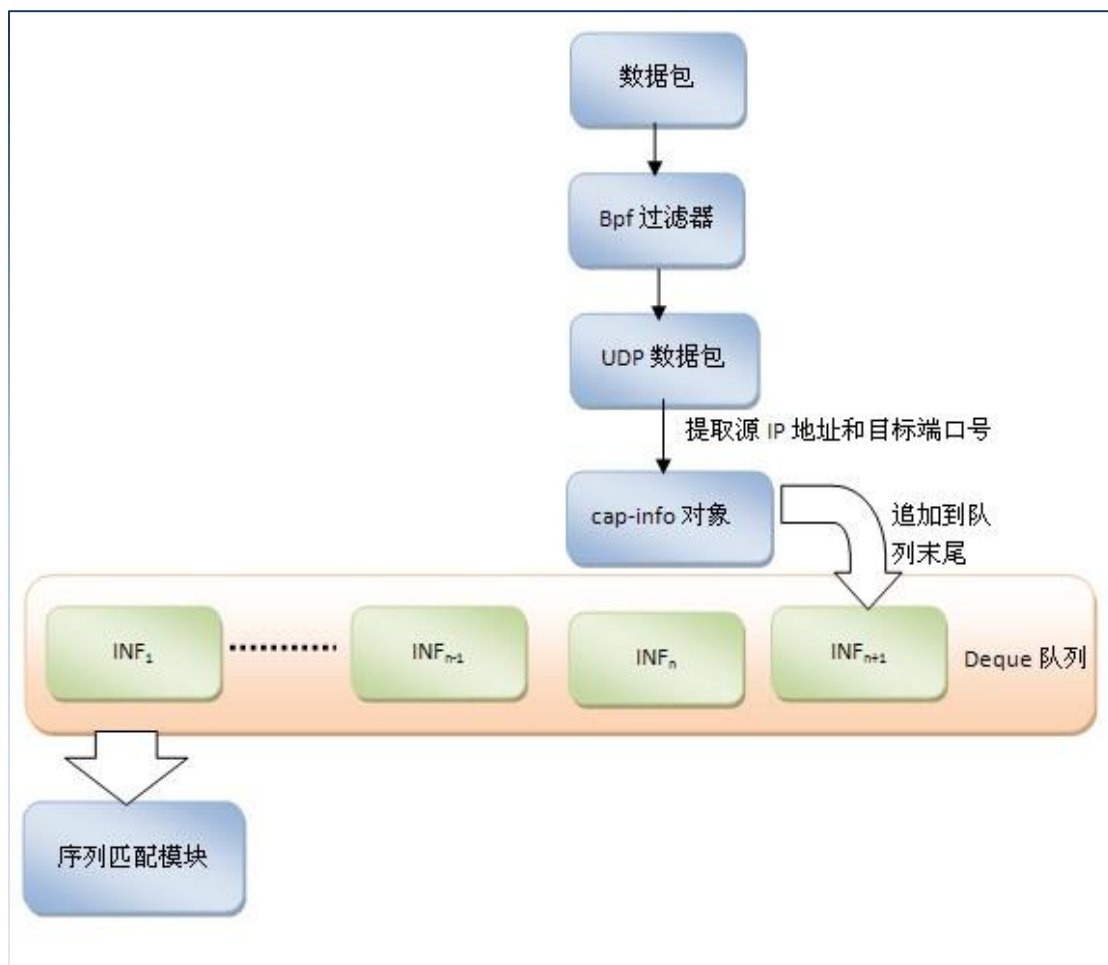


图 20 数据捕获模块框架图

B.序列匹配模块

● 全局参数介绍

Deque	所有 cap_info 组成的队列
right-seq	所有用户的合法认证序列
right-count	记录每个用户 ID 的序列被某些源 IP 匹配正确的次数

数据捕获模块不断从队列尾部填充 Deque，要进行序列匹配就要从 Deque 中取出各个 cap_info 的信息，并与 right-seq 中正确的认证序列相匹配。取下 Deque 中一个元素的信息时，必须删除 Deque 中的该元素以防内存泄露。

从 Deque 上取下信息花费的时间很短，而与 right-seq 中每一个用户的序列进行匹配需要花较长的时间。所以我们采用多线程技术，由多个序列匹配线程从 Deque 队列头上取下数据，并删除头元素，使得同时可以有多个线程在进行匹配。为了避免竞态条件带来的错误，我们需要建立健壮的程序模型。

● 竞态条件

竞态条件（race condition）是一个在设备或者系统试图同时执行两个操作的时候出现的不希望的状况，但是由于设备和系统的自然特性，为了正确地执行，操作必须按照合适顺序进行。

在计算机内存或者存储里，如果同时发出读写大量数据的指令的时候竞态条件可能发生，机器试图覆盖相同的或者旧的数据，而此时旧的数据仍然在被读取。结果可能是下面一个或者多个情况：计算机死机，出现非法操作提示并结束程序，错误的读取旧的数据，或者错误的写入新数据。

在序列匹配模块，多个线程都对 Deque 的队列头进行操作，在取下头部信息后删除头部元素。这会触发竞态条件使得匹配线程崩溃：假设某个匹配线程取下头元素后进行了删除操作，另一个线程却正在读取该头元素的信息并试图删除，然而该头元素早已被前一个线程删除，程序错误。为了解决竞态条件的问题，我们使用了互斥锁。

● 互斥锁实现安全的多线程访问

互斥锁只有两种状态，也就是上锁和解锁。可以把互斥锁看作某种意义上的全局变量。在同一时刻只能有一个线程掌握某个互斥上的锁，拥有上锁状态的线程能够对共享资源进行操作。若其他线程希望上锁一个已经上锁了的互斥锁，则该线程就会挂起，直到上锁的线程释放掉互斥锁为止。可以说，这把互斥锁使得共享资源按序在各个线程中操作。

在编程过程中，引入了对象互斥锁这个机制，来保证共享数据操作的完整性。创建一个互斥锁与 Deque 对应，这个互斥锁用来保证在任何一个时刻，只能有一个线程访问 Deque 的队列头。使得匹配线程不会发生竞态条件。

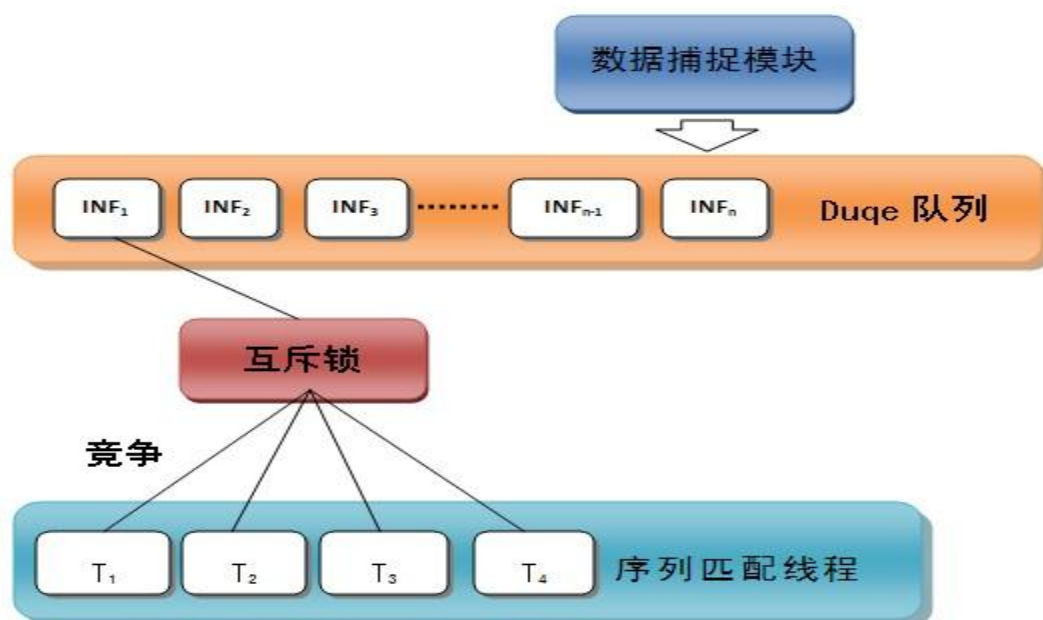


图 21 序列匹配模块框架图

● 序列匹配线程具体工作流程

Step1: 所有匹配线程争取“互斥锁”；

Step2: 争取到的线程从 Deque 队列头读取数据，并删除队列头；

Step3: 争取到的线程释放互斥锁，用读到的信息与 right-seq 中每一个序列进行匹配，匹配完成后执行 Step5；

Step4: 在互斥锁被释放的瞬间，其余多个匹配线程继续争取拿到互斥锁，拿到锁的线程执行 Step2 和 Step3；

Step5: 加入余下线程，继续争取互斥锁；

从开锁到解锁花费时间很短（只是读取和删除队列头），而与每个序列匹配的时间较长，删除头元素后立刻解锁并进行匹配，让其他几个线程争夺队列资源并执行同样的操作，大大增加了序列匹配的效率。

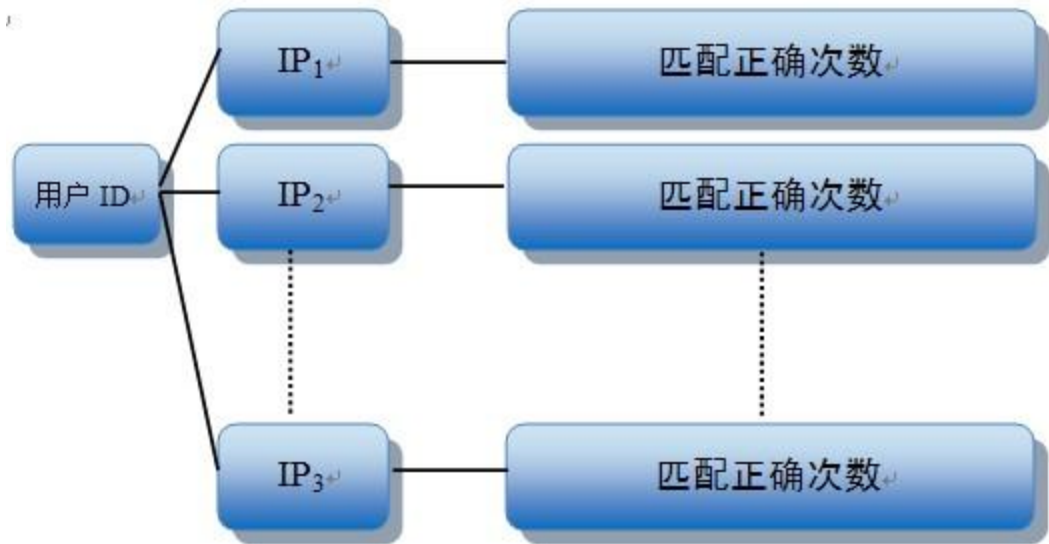


图 22 right-count 全局变量示意图

right-count 记录每个用户 ID 的序列被某些源 IP 匹配正确的次数。

当 UDP 包的目标端口号与某个用户的序列中某一数字相同，则该源 ip 对于这个用户的匹配正确次数加一，当某个匹配正确次数大于或等于 10 次时，该 ip 被认为具有合法的认证序列，可以通过访问控制规则。

C. 后台数据交互模块

后台数据库采用 MySQL, 后台数据交互模块通过 MySQL 与 C 语言的编程接口实现。服务器的序列初始化线程通过 MySQL 的 C 语言编程接口与之交互获得用户信息，包括 用户的缓存状态和多组认证序列。当有合法用户认证成功时，程序计算该用户新的序列与状态，通过后台数据交互模块更新数据库。

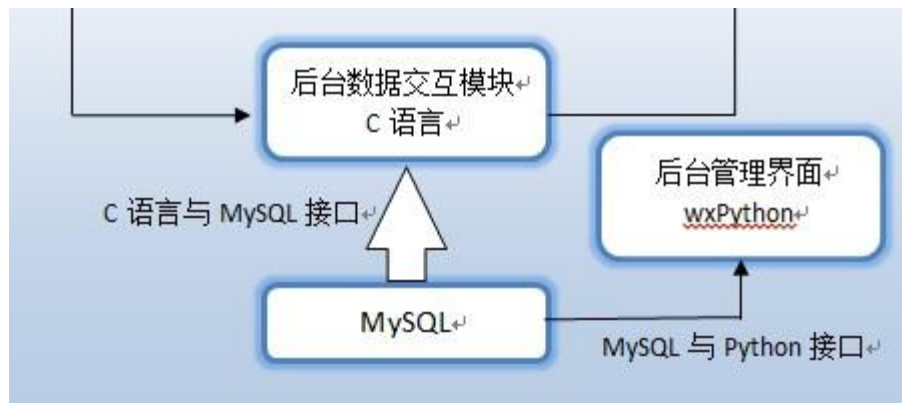


图 23 MySQL 编程接口示意图

MySQL 提供了简单的编程接口，使交互操作容易实现。

服务器后台数据库提供图形化管理界面，使管理员方便管理用户信息、查看在线用户、查看本软件日志等。数据后台的图形化界面使用 wxPython 开发，与数据库交互部分使用 MySQL 与 Python 的编程接口实现。如下图：



图 24 后台数据管理选择界面

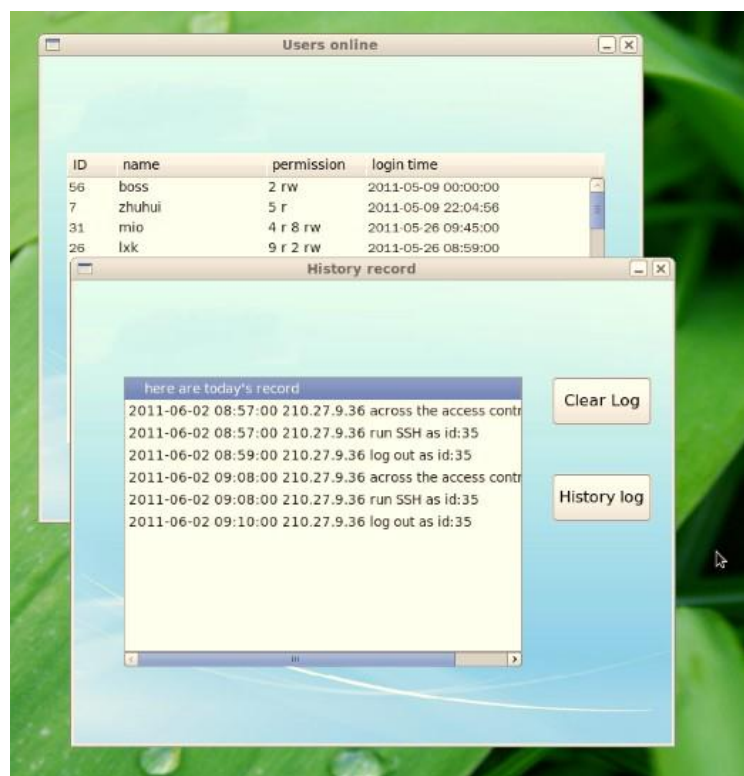


图 25 数据库后台操作界面

四、性能测试

1.测试环境

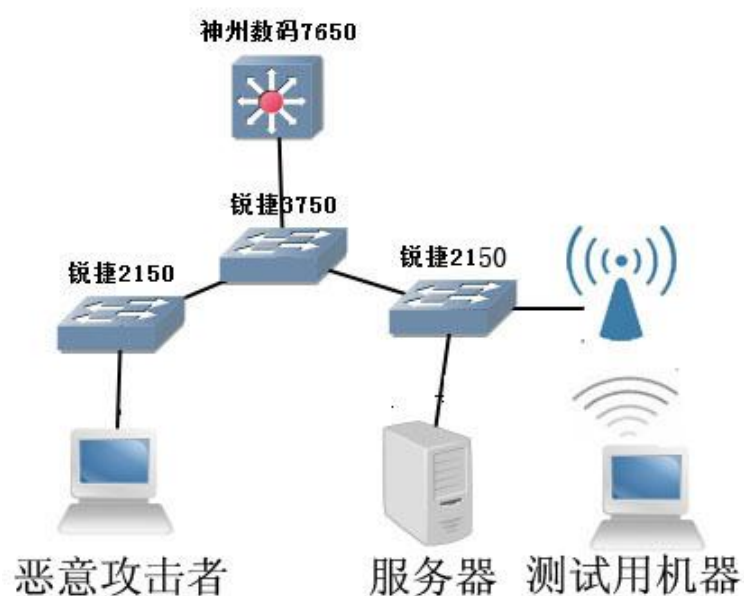


图 26 测试网络拓扑图

服务器：fedora 14 典型 Linux 系统；
测试用机器及模拟恶意攻击者：windows 7 旗舰版操作系统；

● 所搭建网络环境及特点

核心交换在此充当路由器功能，所有用户连接至接入层设备（交换机，无线交换机），然后在锐捷 3750 交换机上汇聚，通过光纤传输给该网络的核心交换。

搭建了一个微型的互联网，并且能够通过修改配置模拟在 Internet 中可能出现的网络问题，从而更有效的测试该款软件性能。

2. 测试项目

序号	测试名称	测试目的
a	防御扫描测试	检验软件对恶意扫描的防御能力
b	软件可用性测试	检验软件可用性
c	前向安全性测试	检验认证序列随机性和前向安全性
d	抵制身份欺骗测试	检验对“冒充式”攻击抵御能力
e	传输安全性测试	验证传输数据经过 AES 加密

表格 2 测试项目

a) 防御扫描测试

测试目的：此项目测试本软件对防御恶意扫描能力。

测试过程：在关闭和开启本软件状态下，利用“SuperScan”扫描服务器。

测试结果及分析：



图 27 关闭本软件扫描结果



图 28 开启本软件扫描结果

图 27 和图 28 说明在开启本软件状态下，能有效防御恶意扫描软件的攻击。

b) 软件可用性测试

测试目的：此项目测试本软件能否为合法用户提供服务，并初步展示软件。

测试过程：合法用户正常登录，并使用 SSH 链接到服务器。

测试结果及分析：

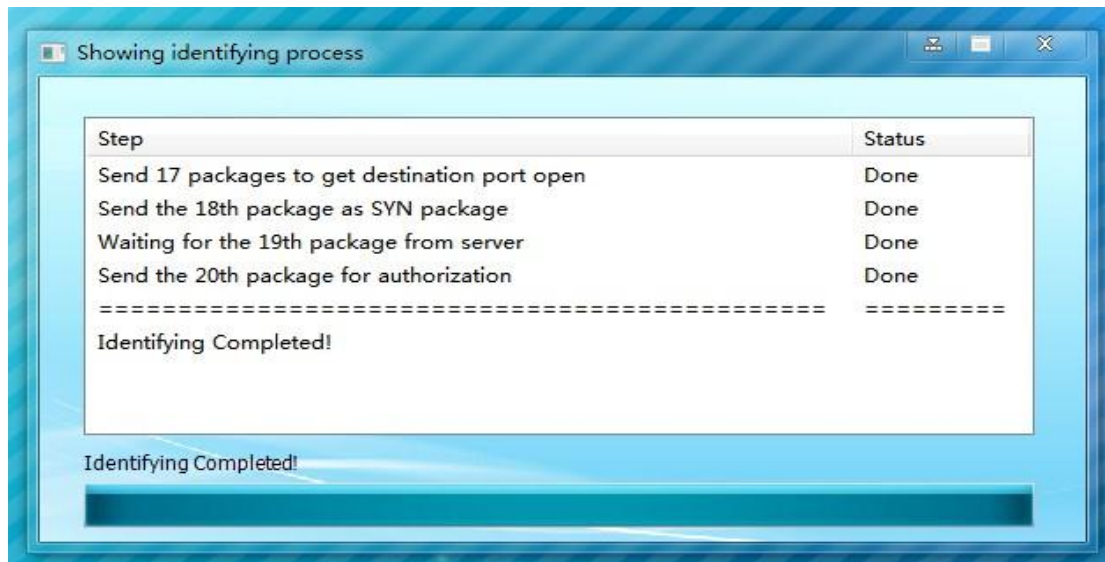


图 29 软件认证进度示意图

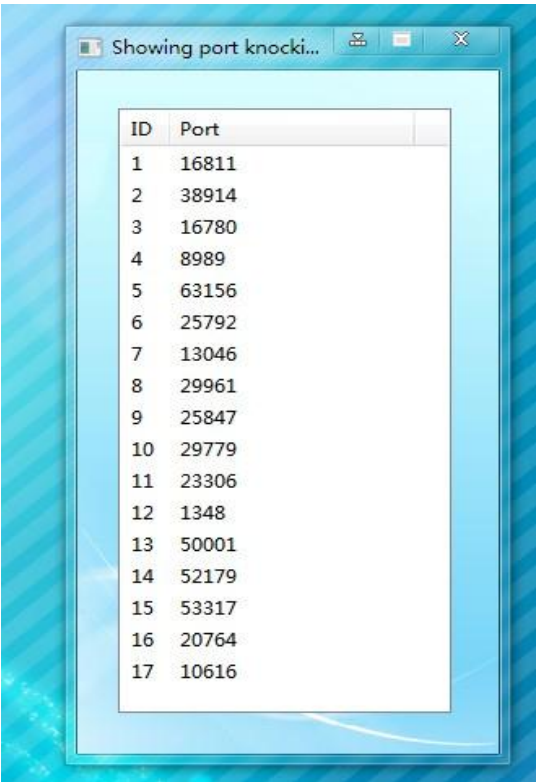


图 30 UDP 认证端口号序列



图 31 客户端可用 SSH 端口号

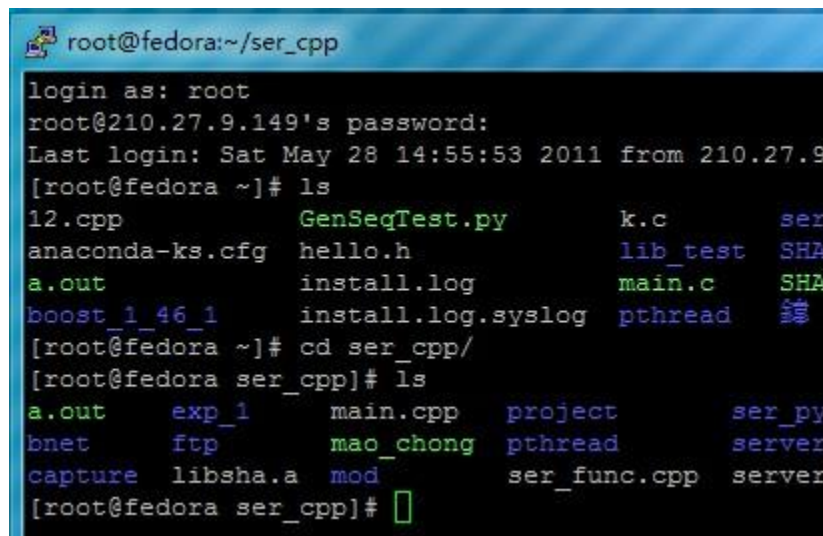


图 32 成功使用 SSH 访问服务器

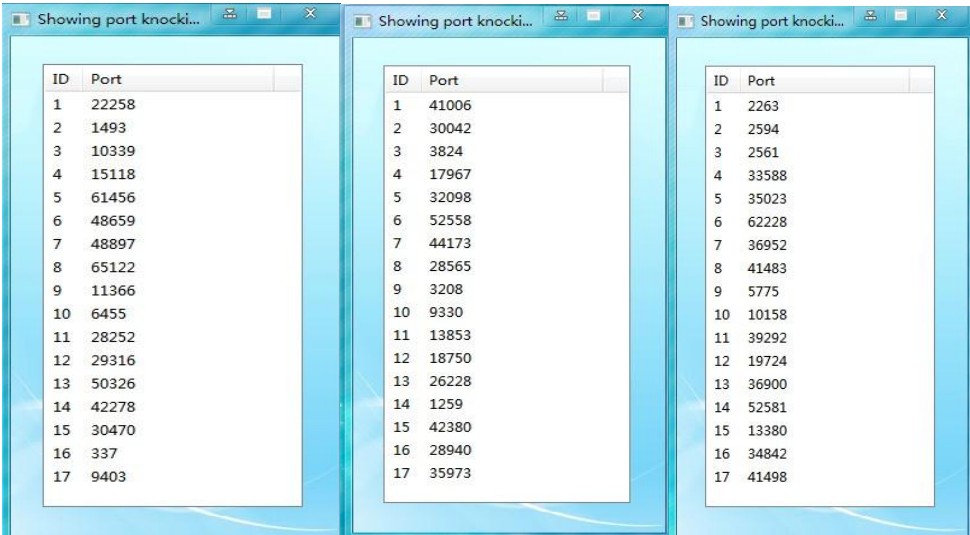
图 29 是用户输入完 KEY 后认证进度，图 28 是进行认证时使用的 UDP 端口序列，图 31 是选择 SSH 服务后，服务器开启该服务的端口，图 32 是成功利用 SSH 链接到服务器截图。说明用户已经通过认证，并使用 SSH 成功访问到服务器。

c) 前向安全性测试

测试目的： 检验每次登陆所需的认证序列各不相同，固定服务的端口号动态分配，且都具有随机特性。

测试过程： 同一个用户连续登陆 3 次都使用 SSH 服务，比对认证序列和服务端分配的 SSH 端口号。

测试结果及分析：

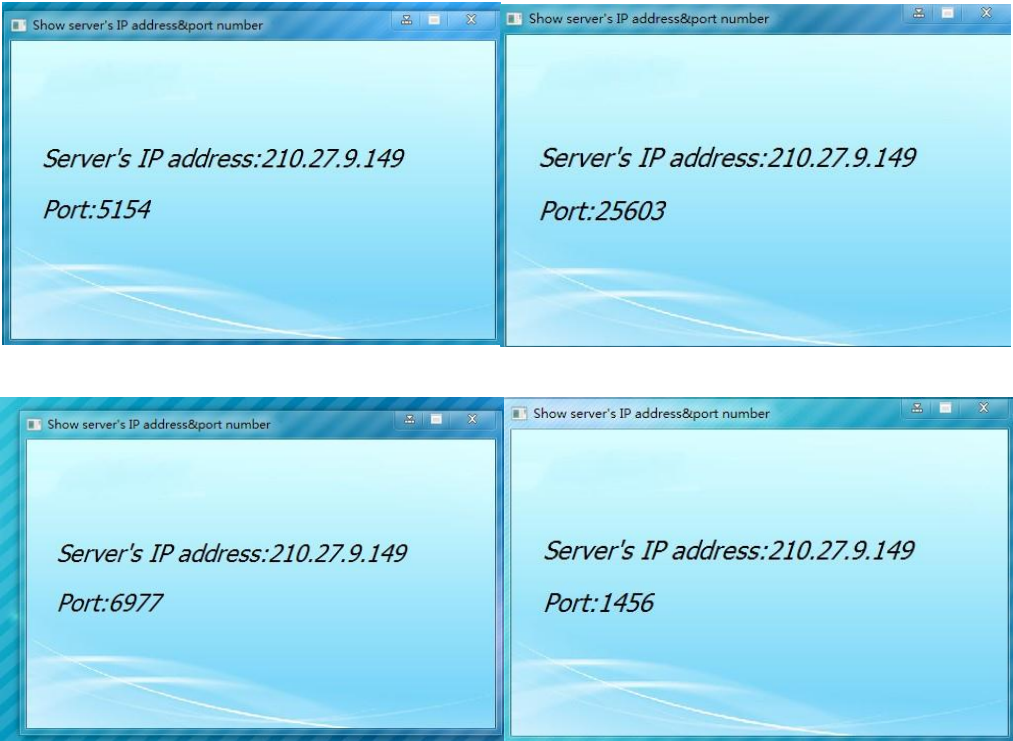


ID	Port
1	22258
2	1493
3	10339
4	15118
5	61456
6	48659
7	48897
8	65122
9	11366
10	6455
11	28252
12	29316
13	50326
14	42278
15	30470
16	337
17	9403

ID	Port
1	41006
2	30042
3	3824
4	17967
5	32098
6	52558
7	44173
8	28565
9	3208
10	9330
11	13853
12	18750
13	26228
14	1259
15	42380
16	28940
17	35973

ID	Port
1	2263
2	2594
3	2561
4	33588
5	35023
6	62228
7	36952
8	41483
9	5775
10	10158
11	39292
12	19724
13	36900
14	52581
15	13380
16	34842
17	41498

图 33 3 次认证使用的 UDP 端口序列



Server's IP address	Port
210.27.9.149	5154

Server's IP address	Port
210.27.9.149	25603

Server's IP address	Port
210.27.9.149	6977

Server's IP address	Port
210.27.9.149	1456

图 34 同一个用户四次使用 SSH 服务时服务器开放的端口号

图 33 中的三幅图，是同一用户连续三次访问服务器，进行认证时使用的 UDP 端口序列。

图 34 是同一用户连续四次使用 SSH 服务时，服务器开放的端口号。说明端口序列产生的随机性，使得认证具有前向安全性。

d) 抵制身份欺骗攻击

测试目的：检验本软件对网络“冒充式”攻击的抵御能力。

测试过程：用一个程序冒充合法服务器（使用服务器的 IP），客户端向该 IP 发送认证序列，并请求建立 tcp 链接，该程序接受请求并正常建立链接，观察客户端能否识别出服务器是否合法。

测试结果及分析：

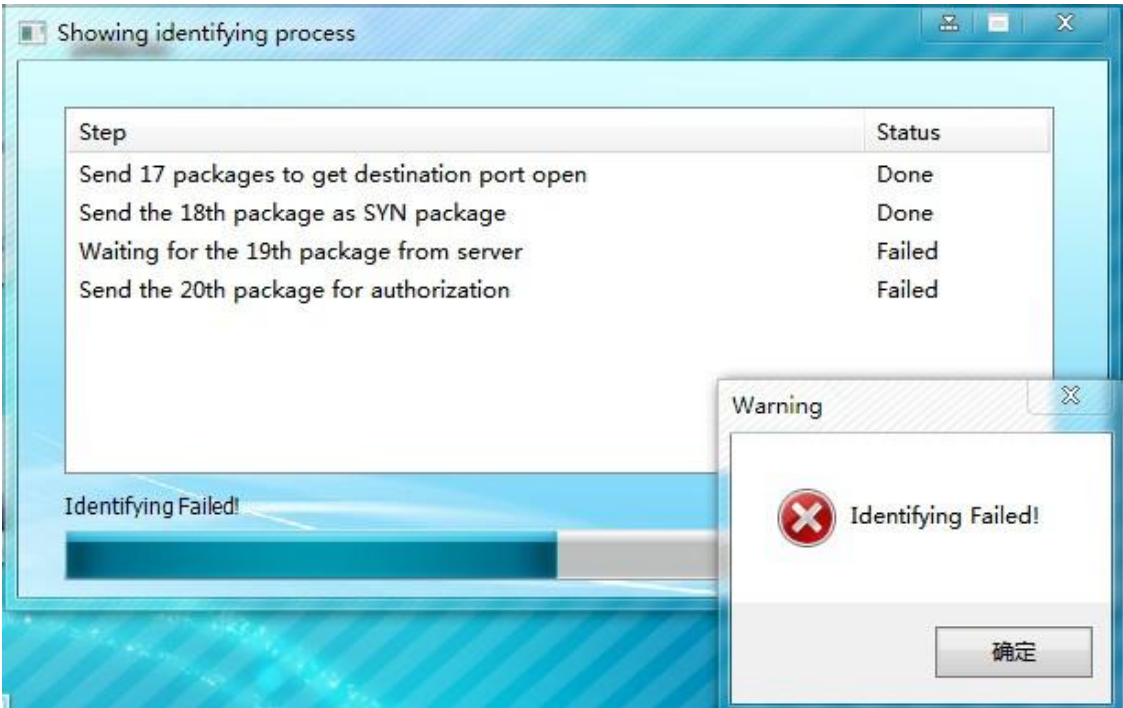


图 35 网络攻击者进行认证状态图

图 35 当网络攻击者通过“欺骗”合法用户，冒充服务器并建立链接后，客户端等待“服务器”回传第 19 个序列元素，但是假冒的服务器无法推测出该元素。所以在图示中的第三步客户端识别出假冒服务器。

e) 传输安全性测试

测试目的：验证传输过程中数据经过 AES 加密。

测试过程：客户端认证成功后，选择服务时选择“文件共享”，在服务器下载一个 c++ 文件。客户端使用 wireshark 进行捉包，查看数据包的内容。

测试结果及分析：

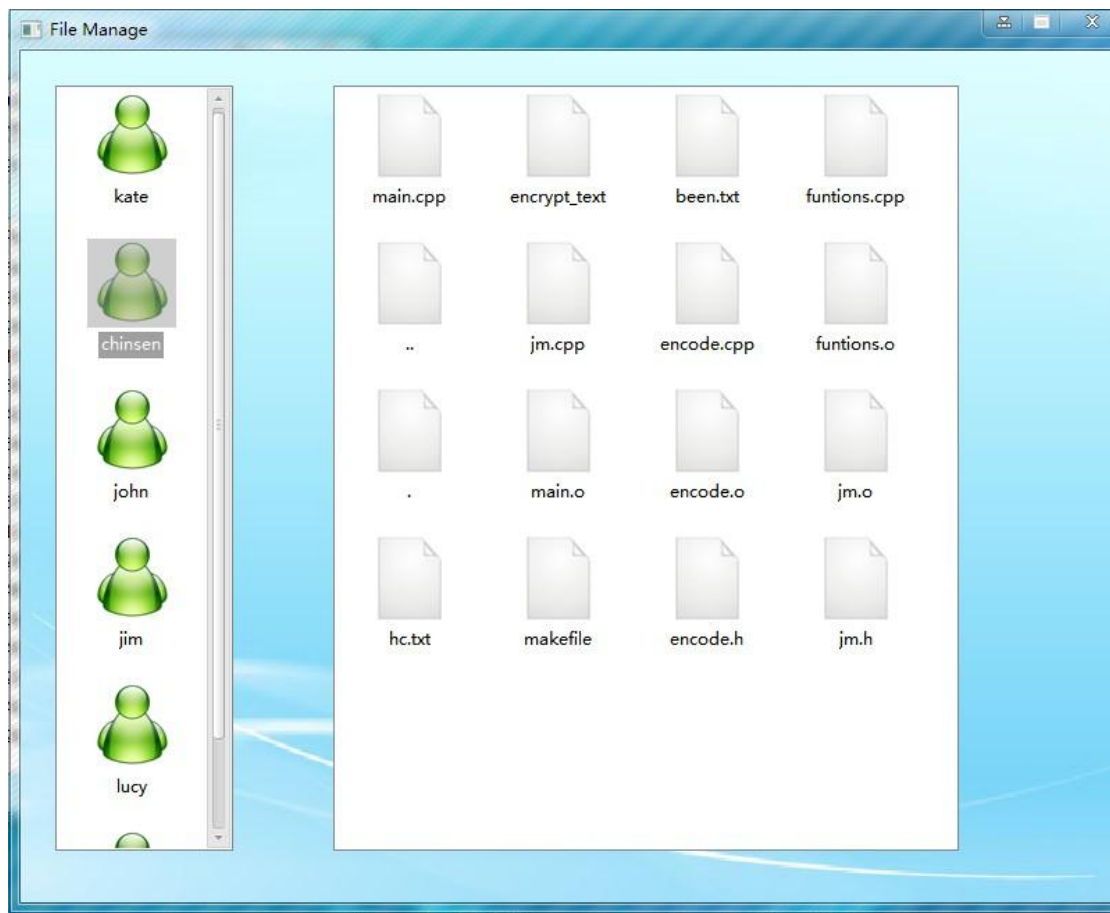


图 36 文件共享服务的客户端界面

图 36 为选择文件共享服务后，软件提供的文件选择界面。在测试时我们选择下载图中的“encode. cpp”进行下载。

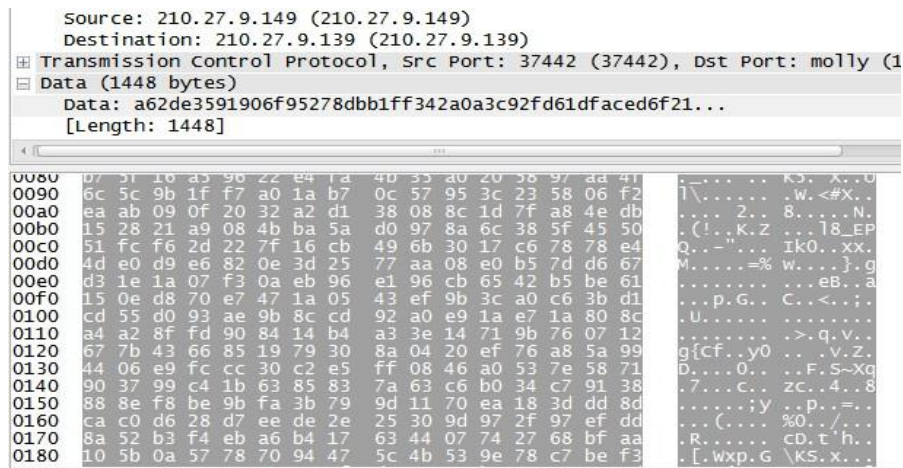


图 37 数据包的内容截图 1

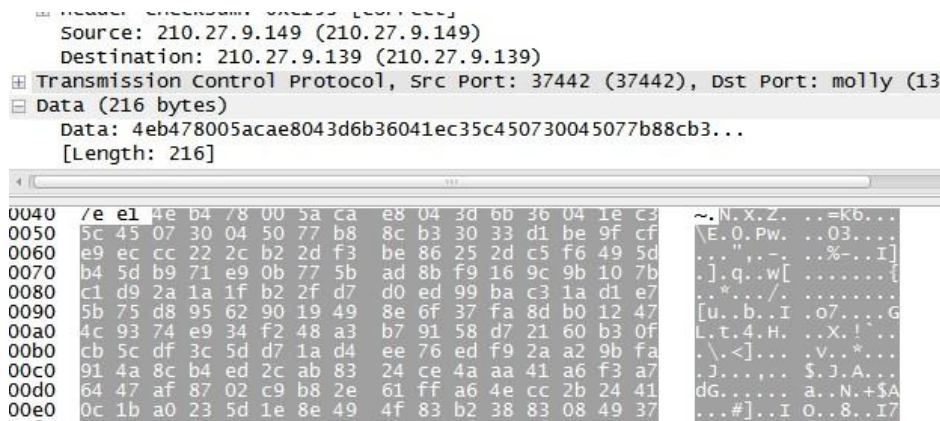


图 38 数据包的内容截图 2

图 37 和图 38 为 encode.cpp 在传输过程中, wireshark 捉到的数据包的内容截图, DATA 字段已全部加密。

```
#include "encode.h"
using namespace std;
void encode::get_new_key(vector<int>& k)
{
    int sum=0, aver, ct=0, rt=0;
    for(vector<int>::size_type i=0; i!=k.size(); ++i)
    {
        sum+=k[i];
        int s=k[i];
        while(s>10)
            s-=10;
        ct+=s;
        s=k[i];
        while(s/10!=0)
            s=s/10;
        rt+=s;
    }
}
```

图 39 文件内容截图

图 37 为客户端收到的文件内容截图,为正常程序软件。

由此可见, encode.cpp 在传输过程中传送的是加密后的密文。作品使用 256 位密钥的 AES, 保证了传送数据的安全。

六、总结

在网络攻击越发频繁的形势下, 访问控制和认证对于远程管理服务器、共享资源等网络应用尤其重要。现有的各种方法各自有着显著的优缺点, 但都不能满足现今的需求。防火墙可以控制数据包的进出, 但是它只是遵循一个规则集。各种认证方法都在完成接入之后再进行身份认证, 但多次的反复失败认证会耗费资源, 网络上恶意用户数量庞大, 接入后认证失败的频繁程度可想而知。且远程服务器的安全配置需要较多的网络管理经验, 配置疏忽和经验缺乏都会造成很大的风险, 服务器的访问控制和认证非常需要更自动化的解决方案。

基于这些问题, 我们设计开发了这种新型的隐秘访问系统, 以弥补这些方面的缺陷。它为访问控制策略的自动化动态改变、认证的前向安全性等提供了一种很有参考价值的解决方案。

七、参考文献

- [1] Charlie Kaufman , Radia Perlman, Mike Speciner 著
《网络安全: 公众世界中的秘密通信》 电子工业出版社。
- [2] 吴世忠 著 《信息安全评测认证理论与实践》 中国科技大学出版社。
- [3] 李晓航, 王宏霞, 张文芳 著 《认证理论及应用》 清华大学出版社。
- [4] Robert Love 著 康华, 张波 译 《Linux 内核设计与实现》 机械工业出版社。
- [5] W.Richard Stevens, Bill Fenner 著 杨继张 译 《Unix 网络编程》 清华大学出版社。
- [6] 李晖, 李丽香, 邵帅 著 《对称密码学及其应用》 北京邮电大学

出版社。

- [7] 杨波 《现代密码学》 清华大学出版社。
- [8] Wesley J. Chun 《Python 核心编程》 人民邮电出版社。