目录

[1 项目概述 2](#_Toc500494621)

[1.1 前言 2](#_Toc500494622)

[1.2 开发方式 2](#_Toc500494623)

[1.3 术语与缩略语 2](#_Toc500494624)

[1.4 系统组成 2](#_Toc500494625)

[1.5 系统部署图 2](#_Toc500494626)

[1.6 总体框架 3](#_Toc500494627)

[2 电站安全防护模块设计 3](#_Toc500494628)

[2.1 支持平台 3](#_Toc500494629)

[2.2 基本框架 3](#_Toc500494630)

[2.3 与客户端交互端口 5](#_Toc500494631)

[2.4 模块设计 5](#_Toc500494632)

[2.4.1 初始化模块 6](#_Toc500494633)

[2.4.2 通信模块 7](#_Toc500494634)

[2.4.3 客户端数据处理模块 7](#_Toc500494635)

[2.4.4 数据包处理模块 8](#_Toc500494636)

[2.4.5 DPI检测模块 9](#_Toc500494637)

[2.4.6 日志管理模块 10](#_Toc500494638)

[3 客户端设计 11](#_Toc500494639)

[3.1 开发平台和工具 11](#_Toc500494640)

[3.2 支持平台 11](#_Toc500494641)

[3.3 部署安装 11](#_Toc500494642)

[3.4 基本框架 15](#_Toc500494643)

[3.5 与电站防护模块交互 15](#_Toc500494644)

[3.5.1 交互接口设计 16](#_Toc500494645)

[3.6 日志模块 18](#_Toc500494646)

[3.6.1 窗口设计 19](#_Toc500494647)

[3.7 服务进程控制 20](#_Toc500494648)

# 1 项目概述

## 1.1 前言

随着信息化与工业化的深度融合，越来越多的工业控制系统产品（如SCADA、DCS等）采用通用协议、通用硬件和通用软件，以各种方式与MIS网络、因特网等公共网络连接，但是由于工业控制系统与传统的IT系统之间有着很大的区别，工业控制系统方面还存着很多不足，而由于防护不足会造成很多严重的问题，其中包括核心数据被窃取、关键工控流程被破坏、对工业系统功能未经授权的访问甚至造成工厂停产等，所以增加工业控制系统的防护功能有非常重要的意义。

## 1.2 开发方式

自行研制与技术引进相结合。软件平台基于Linux操作系统，硬件平台基于工控机。

## 1.3 术语与缩略语

DCS（分布式控制系统）、SCADA（数据采集与监视控制系统）、DPI（深度包检测技术）、DDOS攻击（分布式拒绝服务攻击）

## 1.4 系统组成

系统包括软件和硬件两部分。其中硬件为工控机，软件包括配置管理客户端及工控机中各种安全防护模块。

## 1.5 系统部署图

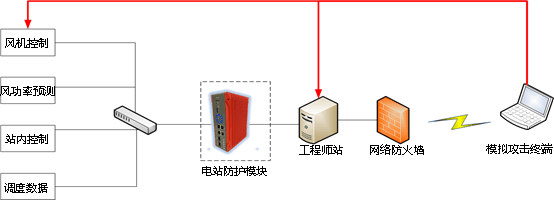


图1-1 系统部署图

## 1.6 总体框架



图1-2 系统整体架构

# 2 电站安全防护模块设计

## 2.1 支持平台

Ubuntu 12.04 LTS操作系统

## 2.2 基本框架

根据实际需要采用位于Linux网络层和安全防护内核功能模块之间的Netfilter架构进行开发。

Netfilter 是Linux内核中进行数据包过滤、连接跟踪、地址转换等的主要实现框架。当我们希望过滤特定的数据包或者需要修改数据包的内容再发送出去，这些动作主要都在netfilter中完成。

Netfilter框架在内核中的位置如图2-1所示。

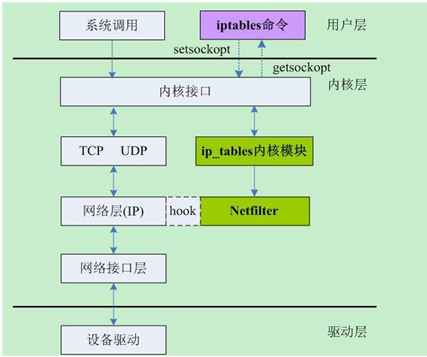


图2-1 Netfilte框架

iptables工具就是用户空间和内核的Netfilter模块通信的手段，iptables命令提供很多选项来实现过滤数据包的各种操作，所以，我们在定义数据包过滤规则时，并不需要去直接修改内核中的netfilter模块，后面会讲到iptables命令如何作用于内核中的netfilter。

Netfilter的实质就是定义一系列的hook点（挂钩），每个hook点上可以挂载多个hook函数，hook函数中就实现了我们要对数据包的内容做怎样的修改、以及要将数据包放行还是过滤掉。数据包进入netfilter框架后，实际上就是依次经过所有hook函数的处理。

对于IPv4协议来讲，一共有5个hook点，这5个hook点上注册的hook函数放在表2-1中：

|  |  |
| --- | --- |
| Hook点 | 该hook点上注册的hook函数链表的表头 |
| NF\_INET\_PRE\_ROUTING | nf\_hooks[NFPROTO\_IPV4][ NF\_INET\_PRE\_ROUTING] |
| NF\_INET\_LOCAL\_IN | nf\_hooks[NFPROTO\_IPV4][ NF\_INET\_LOCAL\_IN] |
| NF\_INET\_FORWARD | nf\_hooks[NFPROTO\_IPV4][ NF\_INET\_FORWARD] |
| NF\_INET\_LOCAL\_OUT | nf\_hooks[NFPROTO\_IPV4][ NF\_INET\_LOCAL\_OUT] |
| NF\_INET\_POST\_ROUTING | nf\_hooks[NFPROTO\_IPV4][ NF\_INET\_POST\_ROUTING] |

表2-1 IPv4协议hook点函数

对于发往本地的数据包，会依次经过NF\_INET\_PRE\_ROUTING和NF\_INET\_LOCAL\_IN两个hook点的处理。

对于本地向外发出去的数据包，会依次经过NF\_INET\_LOCAL\_OUT和NF\_INET\_POST\_ROUTING两个hook点的处理。

对于经过本机转发的数据包，会依次经过F\_INET\_PRE\_ROUTING、NF\_INET\_FORWARD和NF\_INET\_POST\_ROUTING三个hook点的处理。

Netfilter功能框架如图2-2所示。



图2-2 Netfilter功能框架

每个钩子函数最后必须向Netfilter框架返回下列几个值其中之一：

1. NF\_ACCEPT继续正常传输数据报。这个返回值告诉Netfilter：到目前为止，该数据包还是被接受的并且该数据包应当被递交到网络协议栈的下一个阶段。

2. NF\_DROP丢弃该数据报，不再传输。

3. NF\_STOLEN模块接管该数据报，告诉Netfilter“忘掉”该数据报。该回调函数将从此开始对数据包的处理，并且Netfilter应当放弃对该数据包做任何的处理。但是，这并不意味着该数据包的资源已经被释放。这个数据包以及它独自的sk\_buff数据结构仍然有效，只是回调函数从Netfilter获取了该数据包的所有权。

4. NF\_QUEUE对该数据报进行排队(通常用于将数据报给用户空间的进程进行处理)

5. NF\_REPEAT 再次调用该回调函数，应当谨慎使用这个值，以免造成死循环。

## 2.3 与客户端交互端口

端口22222用于接收客户端发送的规则配置消息；

端口33333用于接收客户端扫描设备消息；

端口30330用于向客户端发送设备扫描结果确认消息；

端口30332用于向客户端发送规则配置结果确认消息；

端口8000用于向客户端发送日志信息。

## 2.4 模块设计

电站安全防护模块由初始模块、通信模块、客户端数据处理模块、日志管理模块、数据包处理模块和DPI检测模块组成。

模块层次划分如下图：



图2-3 电站安全防护模块层次划分

### 2.4.1 初始化模块



图2-4电站防护模块运行流程图

1）程序启动。

2）通过文件加载iptables默认规则。

3）连接规则数据库。

4）加载数据库中已保存的所有规则。

5）启动监听端口监听是否有客户端配置信息到达。

### 2.4.2 通信模块

通信模块实现功能如下：

1）启动监听通信端口。

2）监听客户端的扫描及配置信息请求。

3）实现客户端与电站防护模块的通信交互。

2.4.2.1 通信模块类设计

void ProcessData::processData()

|  |  |
| --- | --- |
| 定义 | void ProcessData::processData() |
| 功能 | 启动数据包获取 |
| 参数 | static int lb\_nflog\_fd 打开nflog;static struct nflog\_handle \*handle 绑定地址族;static struct nflog\_g\_handle \*group\_handle 绑定netlink组; |
| 返回值 | 无 |

### 2.4.3 客户端数据处理模块

实现功能：处理客户端发送的扫描设备信息及规则配置信息并将信息分类存入数据库中，同时返回处理结果给客户端。

2.4.3.1 客户端数据处理流程



图2-5 客户端数据处理流程图

2.4.3.2 客户端数据处理模块类设计

int ProcessData::packet\_callback(struct nflog\_g\_handle \*gh, struct nfgenmsg \*nfmsg,

struct nflog\_data \*nfa, void \*data)

|  |  |
| --- | --- |
| 定义 | private:  static int packet\_callback(struct nflog\_g\_handle \*gh, struct nfgenmsg \*nfmsg,struct nflog\_data \*nfa, void \*data); |
| 功能 | 处理数据包 |
| 参数 | data 捕获的数据包内容 |
| 返回值 | 1 |

### 2.4.4 数据包处理模块

实现功能：电站安全防护模块会对通过的所有数据包进行安全性检查，判断数据包是否合法的条件是基于用户通过客户端配置的规则。

2.4.4.1 数据包处理流程



图2-6 数据包处理流程

其中，访问控制是网络安全防范和保护的主要策略，其目的是防止网络资源被非法用户访问和使用，访问控制流程图如图2-7所示。



图2-7 访问控制流程

### 2.4.5 DPI检测模块

实现功能：把当前数据包中应用层载荷的特定内容与协议标准定义的期望值进行比较，检查其是否符合协议标准的规定，从而确定应用层协议的种类以及应用的内容。

2.4.5.1 DPI检测流程



图2-8 DPI检测流程

2.4.5.2 DPI内核module设计

|  |  |
| --- | --- |
| 函数名称 | 功能 |
| bool IsTcpPacket(const struct sk\_buff \*) | 判断是否为TCP数据包 |
| bool IsUdpPackst(const struct sk\_buff \*) | 判断是否为UDP数据包 |
| bool IsIcmpPacket(const struct sk\_buff \*) | 判断是否为ICMP数据包 |
| \_\_u32 GetSrcIp(const struct sk\_buff \*) | 获取数据包源IP地址 |
| \_\_u32 GetDstIp(const struct sk\_buff \*) | 获取数据包目的IP地址 |
| \_\_u32 GetSrcPort(const struct sk\_buff \*) | 获取数据包源端口 |
| \_\_u32 GetDstPort(const struct sk\_buff \*) | 获取数据包目的端口 |
| \_\_u8\* GetAppData(\_\_u8\* , const struct sk\_buff \*) | 获取应用层数据 |
| \_\_u16 GetAppDataLength(const struct sk\_buff \*) | 获取应用层数据长度 |
| modbus\_decode.c | modbusTcp协议解码文件 |
| modbus\_hook.c | modbusTCP 协议钩子函数 |
| OPC\_decode.c | OPC协议解码文件 |
| OPC\_hook.c | OPC 协议钩子函数 |
| libxt\_modbusTcp.c | modbusTcp生成内核模块相关信息 |
| xtmodbusTcp.mod.c | 生成modbusTcp DPI内核模块 |
| xtOPC.mod.c | 生成OPC DPI内核模块 |
| Libxt\_OPC.c | OPC 生成内核模块相关信息 |

### 2.4.6 日志管理模块

实现功能：电站安全防护模块拦截异常数据产生的报警日志发送给客户端，保存到客户端数据库中。

2.4.6.1 日志类设计

|  |  |
| --- | --- |
| 定义 | void sendlog(std::shared\_ptr<SecurityDev> s\_ptr, std::shared\_ptr<ClientDev> c\_ptr) |
| 功能 | 将报警日志发送到客户端 |
| 参数 | SecurityDev 被保护设备，ClientDev客户端设备（包含设备IP及MAC地址） |
| 返回值 | 无 |

# 3 客户端设计

实现功能：用户权限管理、网络拓扑组态、安全规则组态、自定义安全规则、

网络设备类型识别、安全规则自学习、实时报警及日志查询分析。

## 3.1 开发平台和工具

平台：Windows 7操作系统

工具：Visual Studio 2015，WPF（Windows Presentation Foundation）框架

## 3.2 支持平台

MicroSoft Windows 7及以上所有版本操作系统。

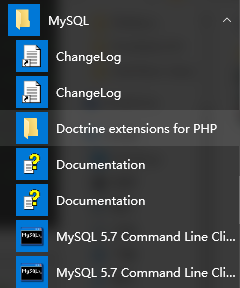
包括：Windows 7/2008/2008R2/Vista/8/2012/10。

## 3.3 部署安装

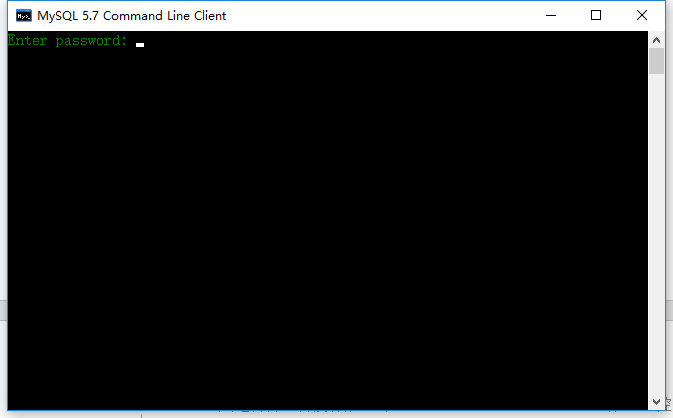
使用电站防护模块配置管理平台需要提供mysql数据库服务。

1、从安装包中找到安装mysql数据库，安装期间会设置数据库的用户名和密码，请牢记！

2、 Mysql安装完成后，从已安装应用中找到，右键选择以管理员身份运行MySQL 5.7 Command Line Client。

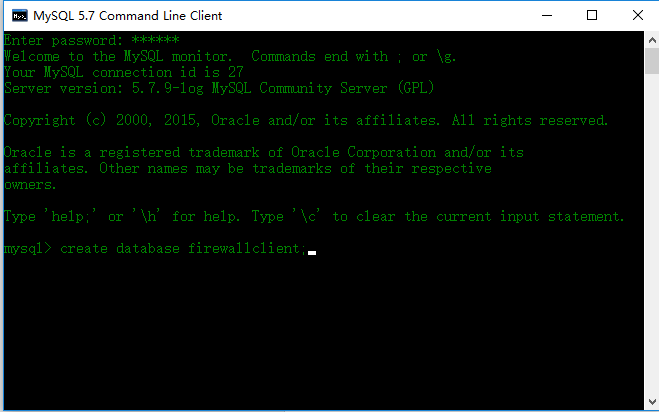


3、输入安装mysql时的用户密码然后执行以下操作：

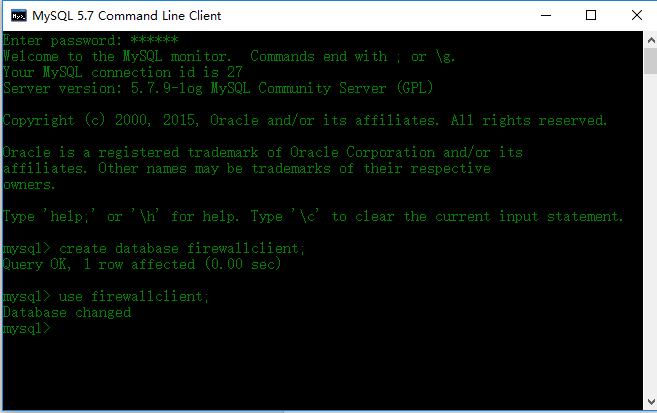


1. create database firewallclient;

（firewallclient为数据库名称，可更改）

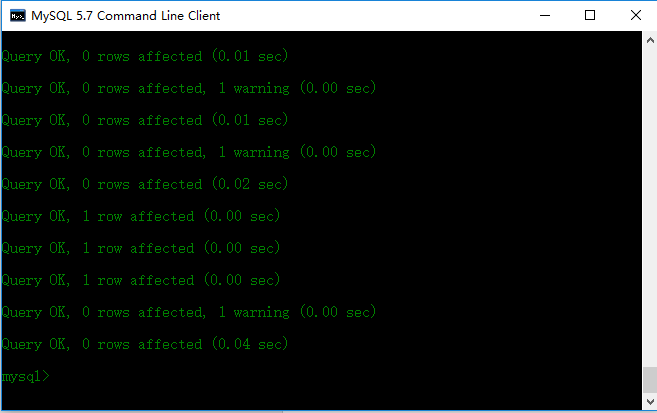


（2） use firewallclient;



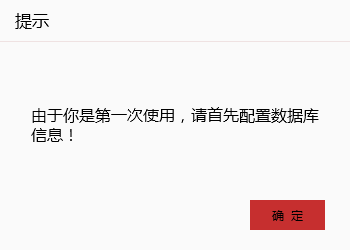
(3) source E:/firewallclient.sql;

（E:/firewallclient.sql为.sql文件所在地址，请根据实际文件所在地址输入，回车后等待执行完毕即可。）



4、启动CMS客户端，

（1）、提示第一次使用CMS客户端需要配置数据库信息。



（2）、填写数据库信息（根据步骤2、3的数据库信息填写）。



数据库信息可在主界面的系统设置中更改。

（3）、数据库配置成功后即可正常使用客户端管理配置电站防护模块。

## 3.4 基本框架

客户端设计采用WPF（Windows Presentation Foundation）框架。

WPF是微软推出的基于Windows 的用户界面框架，属于.NET Framework 3.0的一部分。提供了统一的编程模型、语言和框架，同时它提供了全新的多媒体交互用户图形界面。

3.5 与服务器交互接口

WPF 由两个主要部分组成：引擎和编程框架。

1、 WPF引擎。引擎统一了开发人员和设计人员体验文档、媒体和 UI 的方式，为基于浏览器的体验、基于窗体的应用程序、图形、视频、音频和文档提供了一个单一的运行时库。WPF使得应用程序不仅能够充分利用现代计算机中现有的图形硬件的全部功能，而且能够利用硬件将来的进步。例如，WPF 的基于矢量的呈现引擎使应用程序可以灵活地利用高 DPI监视器，而无需开发人员或用户进行额外的工作。 同样，当WPF 检测到支持硬件加速的视频卡时，它将利用硬件加速功能。

2、WPF 框架。框架为媒体、用户界面设计和文档提供的解决方案远远超过开发人员现在所拥有的。WPF 的设计考虑了可扩展性，使开发人员可以完全在 WPF引擎的基础上创建自己的控件，也可以通过对现有 WPF 控件进行再分类来创建自己的控件。WPF 框架的核心是用于形状、文档、图像、视频、动画、三维以及用于放置控件和内容的面板的一系列控件。这些“自有控件”为开发下一代用户体验提供了构造块。

## 3.5 与电站防护模块交互

22222端口用于向局域网发送扫描电站防护模块信息。

33333端口用于向电站防护模块发送规则配置信息。

30331端口用于接收电站防护模块返回的设备扫描信息。

30333端口用于接收电站防护模块返回的配置结果信息。

8000端口用于接收电站防护模块发回的报警日志信息。

### 3.5.1 交互接口设计

int ScanDevice(string start\_IP, string end\_IP)

|  |  |
| --- | --- |
| 定义 | int ScanDevice(string start\_IP, string end\_IP) |
| 功能 | 扫描一个IP范围内的防护墙设备 |
| 参数 | string start\_IP 起始IP地址，end\_IP结束IP地址 |
| 返回值 | int |

List<FWDeviceForm> CheckDevices(string start\_IP, string end\_IP)

|  |  |
| --- | --- |
| 定义 | List<FWDeviceForm> CheckDevices(string start\_IP, string end\_IP) |
| 功能 | 确认网络中存在的防火墙设备，并返回其IP地址列表 |
| 参数 | string start\_IP 起始IP地址，end\_IP结束IP地址 |
| 返回值 | List |

public void SendCheckInfo()

|  |  |
| --- | --- |
| 定义 | public void SendCheckInfo() |
| 功能 | 发送扫描设备的数据 |
| 参数 | 无 |
| 返回值 | 无 |

public bool SendConfigInfo(string cmd)

|  |  |
| --- | --- |
| 定义 | public bool SendConfigInfo(string cmd) |
| 功能 | 发送配置信息 |
| 参数 | string cmd 配置信息 |
| 返回值 | bool |

bool ChangeModbusTcpRules(string dst\_IP, string src\_IP, string min\_addr, string max\_addr, string func, int min\_data, int max\_data, string dev\_IP, bool log\_record, bool add\_delete)

|  |  |
| --- | --- |
| 定义 | bool ChangeModbusTcpRules(string dst\_IP, string src\_IP, string min\_addr, string max\_addr, string func, int min\_data, int max\_data, string dev\_IP, bool log\_record, bool add\_delete) |
| 功能 | 配置ModbusTcp DPI规则 |
| 参数 | string dst\_IP 目的地址IP, string src\_IP 源地址IP, string min\_addr 最小线圈地址, string max\_addr 最大线圈地址, string func 功能码, int min\_data 最小数据, int max\_data 最大数据, string dev\_IP 被保护设备IP地址, bool log\_record 日志记录标志, bool add\_delete 添加/删除规则标志 |
| 返回值 | bool |



图3-1 modbusTcp DPI配置窗口

bool ChangeOPCRules(string dst\_IP, string src\_IP, string dev\_IP, bool log\_record,bool add\_delete)

|  |  |
| --- | --- |
| 定义 | bool ChangeOPCRules(string dst\_IP, string src\_IP, string dev\_IP, bool log\_record,bool add\_delete) |
| 功能 | 配置OPC DPI规则 |
| 参数 | string dst\_IP 目的地址IP, string src\_IP 源地址IP, string dev\_IP 被保护设备IP地址, bool log\_record 日志记录标志,bool add\_delete添加/删除规则标志 |
| 返回值 | bool |



图3-2 OPC DPI规则配置窗口

## 3.6 日志模块

实现功能：开启8000端口监听电站安全防护模块发回的报警日志信息，实时显示并保存到本地数据库中。

public void Save\_DisplayLog(bool start)

|  |  |
| --- | --- |
| 定义 | public void Save\_DisplayLog(bool start) |
| 功能 | 接收并保存日志到数据库 |
| 参数 | bool start 开始标志 |
| 返回值 | 无 |

### 3.6.1 窗口设计



图3-1 实时报警日志窗口

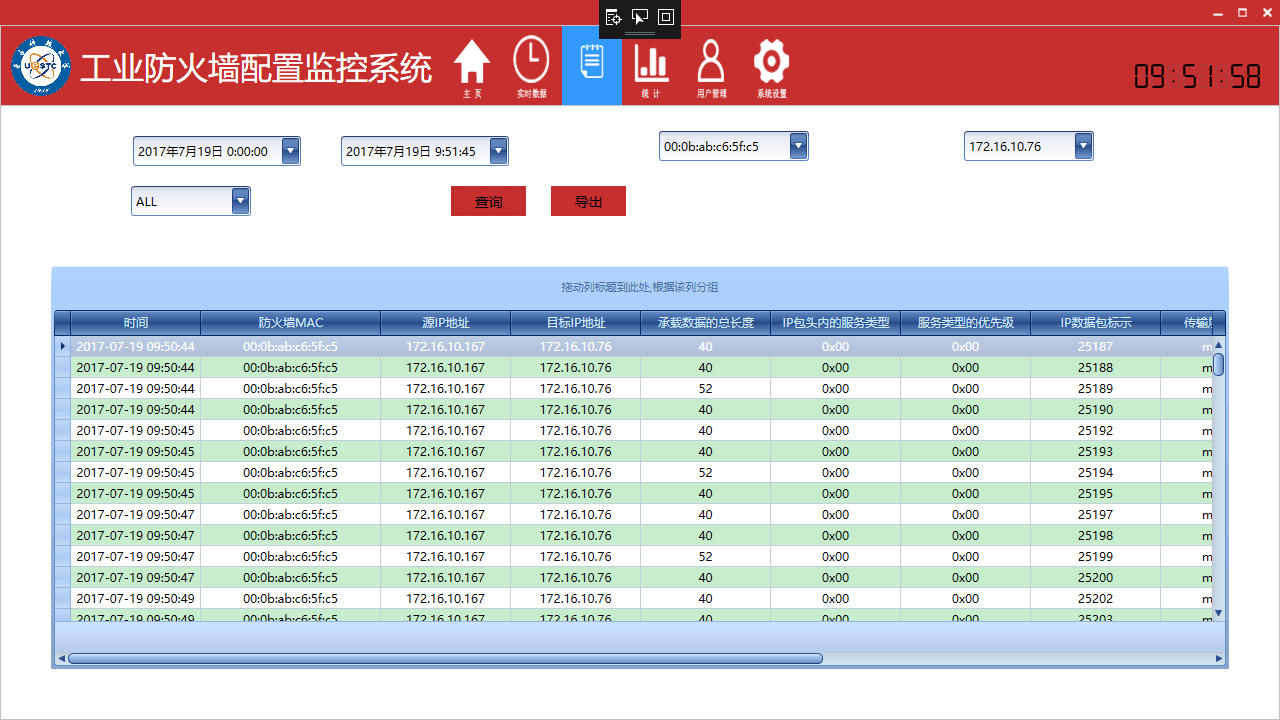
****

图3-2 查询历史日志窗口

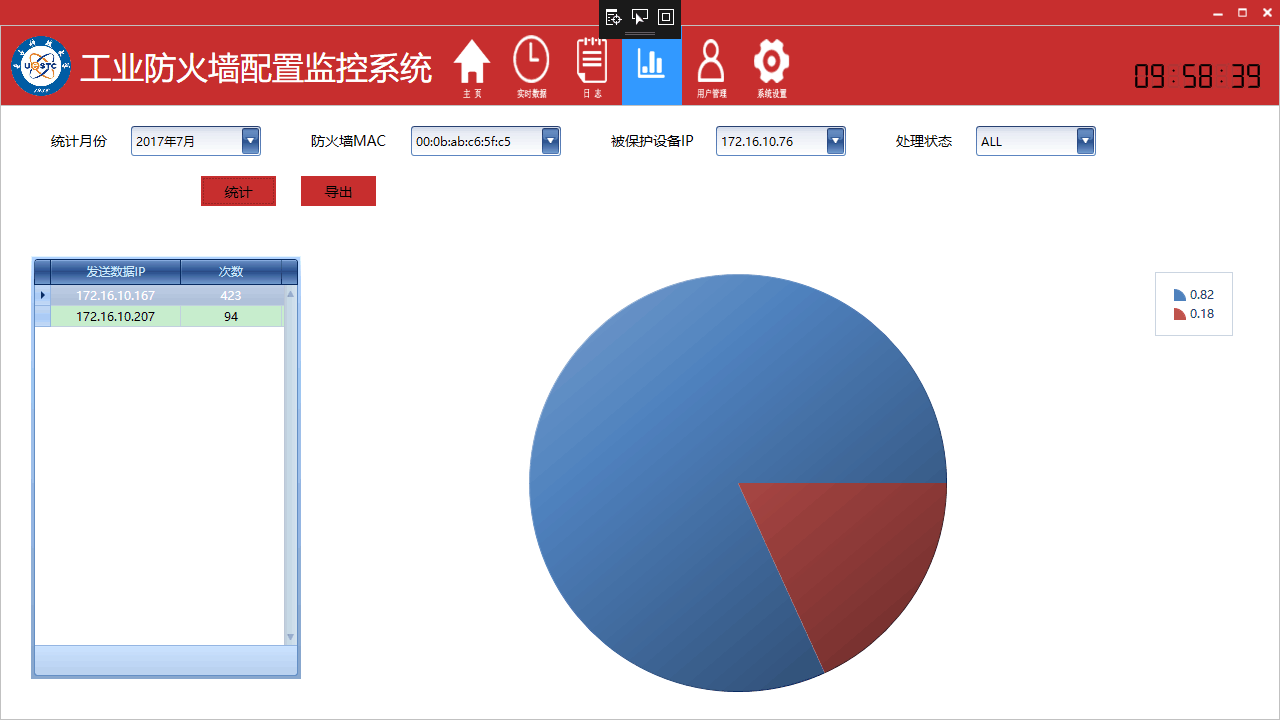


图3-3 报警日志统计窗口

## 3.7 服务进程控制

客户端与电站防护模块通信时会监听相应的端口，因此每台计算机仅能运行一个客户端程序。