****

软 件 工 程 实 验

软件需求规格说明书

基于Scapy的分析与扩展

Version 2.6

实 验 小 组 B组

小 组 成 员 陈鸿超（SY1806214）

李铎坤（SY1806219）

刘 颖（SY1806418）

袁梦阳（BY1806157）

编制时间： 2019年3月

版本变更历史

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 版本 | 提交日期 | 编制人 | 修改说明 | 审核人 | 版本说明 |
| v2.6 | 2019/04/24 | 李铎坤 | 完善第七章 |  | 完善了会话相关RUCM |
| v2.5 | 2019/04/24 | 刘颖 | 完善第七章 | 李铎坤 | 完善了会话提取的相关内容 |
| v2.4 | 2019/04/09 | 李铎坤 | 页码丢失，添加页码 | 刘颖 | 修改了页码问题 |
| v2.3 | 2019/04/08 | 陈鸿超 | 根据A/E组修改意见进行修改 | 李铎坤 | 修改了文档中的语句、术语和图的问题 |
| v2.2 | 2019/04/02 | 袁梦阳 | 修改RUCM | 陈鸿超 | 修改RUCM中VALIDATES THAT表达 |
| v2.1 | 2019/04/01 | 李铎坤 | 修改版本格式 | 袁梦阳 | 更新封面和参考文献及版本记录表 |
| v2.0 | 2019/04/01 | 刘颖 | 修订第三章内容 | 陈鸿超 | 丰富用户需求，补充术语缩略语 |
| v1.10 | 2019/04/01 | 陈鸿超 | 修改2.2节内容 | 李铎坤 | 将2.2节改为系统结构 |
| v1.9 | 2019/03/28 | 袁梦阳 | 按照CD两组问题清单进行修改 | 陈鸿超 | 修改了用例图和RUCM图 |
| v1.8 | 2019/03/28 | 李铎坤 | 按照CD两组问题清单进行修改 | 袁梦阳 | 修改了用户定义，修改了硬件中cpu的要求 |
| v1.7 | 2019/03/28 | 陈鸿超 | 按照CD两组问题清单进行修改 | 李铎坤 | 修改了语句问题、一致性问题、流程图问题 |
| v1.6 | 2019/03/24 | 刘颖 | 给图、表格加引用，修改第七章的叙述方式。 | 陈鸿超 | 修订第七章 |
| v1.5 | 2019/03/24 | 刘,袁,李,陈 | 对整个说明书存在的问题进行修改 | 刘,袁,李,陈 | 第二次修订 |
| v1.4 | 2019/03/22 | 陈鸿超 | 将业务需求改为了用户需求，并对相应内容进行修改 | 袁梦阳 | 根据不同需求的定义对文档进行了调整 |
| v1.3 | 2019/03/20 | 袁梦阳 | 删除了业务需求中的报文解析需求。 | 陈鸿超 | 按照老师要对业务需求与功能需求进行了修正 |
| v1.2 | 2019/03/13 | 刘,袁,李,陈 | 添加部分图注，更改错别字和语句，添加用例图说明，重新整理部分需求的叙述。 | 陈鸿超  李铎坤 | 第一次修订 |
| v1.1 | 2019/03/13 | 袁梦阳 | 修改了功能需求的叙述方式。 | 陈鸿超 | 修改第4章 |
| v1.0 | 2019/03/12 | 刘,袁,李,陈 | 完成第一版初稿 | 刘,袁,李,陈 | 初稿 |

目录

[版本变更历史 I](#_Toc7108663)

[1 引言 1](#_Toc7108664)

[1.1 目的 1](#_Toc7108665)

[1.2 软件需求分析目的 1](#_Toc7108666)

[1.3 文档概述 1](#_Toc7108667)

[1.4 术语和缩略词 1](#_Toc7108668)

[2 整体概述 3](#_Toc7108669)

[2.1 项目概述 3](#_Toc7108670)

[2.2 系统架构 3](#_Toc7108671)

[2.3 用户定义 4](#_Toc7108672)

[3 用户需求 5](#_Toc7108673)

[3.1 数据获取 5](#_Toc7108674)

[3.2 数据构造 5](#_Toc7108675)

[3.3 数据存储 5](#_Toc7108676)

[3.4 数据发送 6](#_Toc7108677)

[3.5 数据展示 6](#_Toc7108678)

[4 功能需求 7](#_Toc7108679)

[4.1 用例模型 7](#_Toc7108680)

[4.2 具体的功能需求和RUCM图 7](#_Toc7108681)

[4.2.1 端口监听 7](#_Toc7108682)

[4.2.2 数据解析 8](#_Toc7108683)

[4.2.3 数据构造 9](#_Toc7108684)

[4.2.4 数据展示 10](#_Toc7108685)

[4.2.5 数据导入 11](#_Toc7108686)

[4.2.6 数据导出 12](#_Toc7108687)

[4.2.7 数据发送 13](#_Toc7108688)

[5 非功能需求 15](#_Toc7108689)

[5.1 兼容性 15](#_Toc7108690)

[5.1.1 Python版本兼容性 15](#_Toc7108691)

[5.1.2 操作系统兼容性 15](#_Toc7108692)

[5.2 可扩展性 15](#_Toc7108693)

[5.3 容错性 15](#_Toc7108694)

[5.4 易用性 16](#_Toc7108695)

[6 运行要求 17](#_Toc7108696)

[6.1 软件要求 17](#_Toc7108697)

[6.2 硬件要求 17](#_Toc7108698)

[7 改进方案设想 18](#_Toc7108699)

[7.1 改进后的用例模型 18](#_Toc7108700)

[7.2 需求分析 18](#_Toc7108701)

[7.2.1 文件切分功能 18](#_Toc7108702)

[7.2.2 监听持久化功能 19](#_Toc7108703)

[7.2.3 协议类型获取功能 21](#_Toc7108704)

[7.2.4 会话提取功能 21](#_Toc7108705)

[7.3 方案设想 23](#_Toc7108706)

[7.3.1 文件切分功能 23](#_Toc7108707)

[7.3.2 监听持久化功能 24](#_Toc7108708)

[7.3.3 协议类型获取功能 25](#_Toc7108709)

[7.3.4 会话提取功能 26](#_Toc7108710)

[8 参考文献 28](#_Toc7108711)

# 引言

## 目的

本文旨在通过对Scapy的功能需求和用户需求等进行分析，并结合软件工程综合实验的要求，来撰写软件需求规格说明书，作为软件开发和测试的指导依据。

## 软件需求分析目的

软件需求分析是在可行性研究的基础上，将用户对系统的描述，通过开发人员的分析概括，抽象为完整的需求定义，系统地将软件的功能性需求等内容形成一系列文档的过程[1]。

需求分析的具体内容可以归纳为五个方面：软件的整体概述，软件的用户需求，软件的功能需求，软件的非功能性需求，软件的运行要求。

## 文档概述

文档用途：本文档主要是对Scapy进行需求及规格说明。

主要内容如下：

1. 描述Scapy的用户需求
2. 以用例图的形式给出Scapy的功能需求，并对各功能需求进行详细的描述
3. 使用RUCM模型对功能需求进行建模
4. 描述Scapy的非功能性需求
5. 描述Scapy运行所需的硬件环境要求
6. 描述Scapy运行所需的软件环境要求
7. 进行改进方案设想

## 术语和缩略词

表格 1.1 术语与缩略语

|  |  |  |
| --- | --- | --- |
| 编号 | 术语 | 说明 |
| 1 | Anaconda | 开源的Python包管理器 |
| 2 | ASN.1 | Abstract Syntax Notation one，计算机网络领域的一种编码、传输、解码标准 |
| 3 | DNS | Domain Name System，域名系统，是互联网的一项服务。它作为将域名和IP地址相互映射的一个分布式数据库，能够使人更方便地访问互联网 |
| 4 | Ether | 物理层协议的简称 |
| 5 | IP | 互联网协议地址，是分配给用户上网使用的网际协议的设备的数字标签 |
| 6 | Npcap | Windows系统中的数据包嗅探和发送工具 |
| 7 | payload | 有效载荷，各协议所携带的需要传输的原始数据 |
| 8 | pcap | pcap 是packet capture 的缩写，pcap文件格式是常用的数据报存储格式 |
| 9 | pkt | Scapy内部的变量名，表示解析后的报文对象 |
| 10 | Python | 一种面向对象的解释型计算机程序设计语言 |
| 11 | RUCM | Restricted Use Case Modeling，限制性用例模型 |
| 12 | tcpdump | Linux系统中功能强大的网络数据采集分析工具 |
| 13 | UDP | User Datagram Protocol，用户数据报协议，开放式系统互联参考模型中一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务 |
| 14 | WinPcap | Windows系统中提供底层网络接口访问的工具 |
| 15 | 报文 | 报文(message)是网络中交换与传输的数据单元，即站点一次性要发送的数据块 |
| 16 | 端口 | 是一种经由软件创建的服务，在一个计算机操作系统中扮演通信的端点 |
| 17 | 端口监听 | 端口监听是指对客户端(个人机器)所操作的一种信息记录，还用于实现对共享目录访问的监测和控制 |
| 18 | 接口 | API，是一些预先定义的函数，目的是提供应用程序与开发人员基于某软件或硬件得以访问一组例程的能力，而又无需访问源码，或理解内部工作机制的细节 |

# 整体概述

## 项目概述

Scapy[2]是计算机网络领域的数据包处理工具，最初由Philippe Biondi用Python编写。它能够帮助用户发送、嗅探、剖析并伪造网络数据包。与此同时，它也允许用户构建可以探测，扫描或攻击网络的工具。简单来讲，Scapy是一个功能强大的交互式数据包处理工具。

Scapy能够伪造或者解码大量协议的数据包，在线路上发送和捕获数据包，并匹配请求和回复[3]。它同样也能轻松处理扫描、路由跟踪、探测、单元测试、攻击和网络发现等任务。从功能上讲，Scapy可以取代很多攻击模拟工具和报文处理工具的部分甚至是全部的功能模块。

Scapy的特点有如下几点：

1. 可以完整地返回获取报文的所有信息。
2. 可以自由地构建所需的数据包，对数据包进行填充、修改任意字段值，并按照需要进行堆叠。
3. 支持多种不同协议类型报文的解析与构建。

## 系统架构

作为一个数据包处理工具，Scapy的系统架构比较简单，主要由底层系统、基础模块、应用模块三部分组成。其中，底层系统是操作系统本身自带的功能，基础模块和应用模块则是由Scapy所实现。Scapy的系统架构如图 2.1所示。



图 2.1Scapy系统架构简图

基础模块是应用模块的实现基础，对于用户而言是不可见的，主要包括环境配置模块、插件调用模块、网络标准模块、协议格式模块。其中环境配置模块主要负责获取运行环境信息，初始化其他模块所需的配置参数；插件调用模块主要负责调用各种插件，比如tcpdump、WinPcap等；网络标准模块主要负责封装计算机网络领域所使用的编码、传输、解码标准；协议格式模块主要封装了Scapy所支持的所有协议的具体格式，包括所有协议字段名称、数据类型、大小、位置等信息。

应用模块是用户可以直接使用的模块，主要包括数据导入模块、数据导出模块、数据展示模块、数据解析模块、数据构造模块、端口监听模块、数据传输模块。这些模块实现了用户进行数据包处理所需的各种功能，包括数据获取、数据构造、数据存储、数据发送和数据展示等。

## 用户定义

Scapy的用户定义为计算机网络方面的开发与测试人员。

# 用户需求

软件的用户需求指的是软件的使用者希望通过该软件能够完成的操作或者目标，是一个软件体现其价值的基本需求。本项目基于Scapy开源项目进行扩展，面向人群为计算机网络方面的开发与测试人员，主要目的是便于用户快捷地进行报文的获取、解析和发送等操作。为了满足用户的需要，本项目考虑实现一个更全面、快速的报文操作工具。为了达到这个目标，本章对Scapy进行了用户需求分析，用户的诉求主要有如下几点：数据获取、数据构造、数据存储、数据发送和数据展示，下面将详细介绍这几点用户需求。

## 数据获取

为了获取数据，用户需要Scapy支持尽可能多层面的数据拦截，包括网卡层面的拦截以及Socket层面的拦截。在网卡层面，用户需要截获和本机同网段的计算机发送的报文数据，包括发往自己的数据包。在Socket层面，用户需要截获本机指定端口上发送和接收到的报文数据。

除了使用Scapy去截获报文数据之外，用户可能还需要从文件中读取报文数据，比如从常见的pcap文件中读取报文数据。

## 数据构造

为了进行网络测试、攻击模拟等操作，用户需要根据自己的需要构造不同协议类型的报文数据，并为各层的协议字段赋予指定的值。报文数据主要是指数据包。数据包是通信传输中的数据单位，由包头和包体构成。包头的长度固定，包含整个数据包的全局信息；包体的长度不定，包含协议的各个字段信息。双方的请求数据包和应答数据包的包头结构是一致的。数据包的内容主要包括目的IP地址、源IP地址、净载数据等部分[4]。用户对数据包的各个字段赋值可以伪造数据包，以构建探测、攻击网络的工具。

## 数据存储

用户在获取到报文数据之后，除了需要对报文数据进行解析处理之外，也需要对报文数据进行存储，以便后续的分析处理。常见的报文存储格式为pcap文件，它主要由文件头和数据块构成。文件头中规定了数据的存储格式、时区、报文的链路类型以及报文最大长度等。数据块可以有很多个，每个数据块都由块头和数据区组成。块头中指定了数据包的截获时间、数据包的长度等信息，数据区则是具体的数据包字节码。用户通过数据存储可以对报文数据进行格式化存储，方便以后读取、分析报文数据。

## 数据发送

对于用户自己构造的报文或者是从其他途径获取到的报文数据，用户需要将该报文数据发送至特定主机，以满足用户的特定需求，比如加密数据传输、攻击模拟等。数据发送包括两种，仅发送数据包和发送数据包并等待接收一个或多个数据包。同时，若按照数据发送的协议层，还可以分为从第二层数据链路层发送数据包或从第三层传输层发送数据包。

## 数据展示

报文数据展示虽然是一项辅助功能，但却是一项重要的用户需求。简洁美观的报文展示可以给用户带来良好的体验，让用户快速了解报文内容，并方便用户进行交流和展示。

# 功能需求

## 用例模型

Scapy主要可以提供给用户以下七部分功能：端口监听、数据解析、数据构造、数据展示、数据导入、数据发送、数据导出。用例图如图 4.1所示。

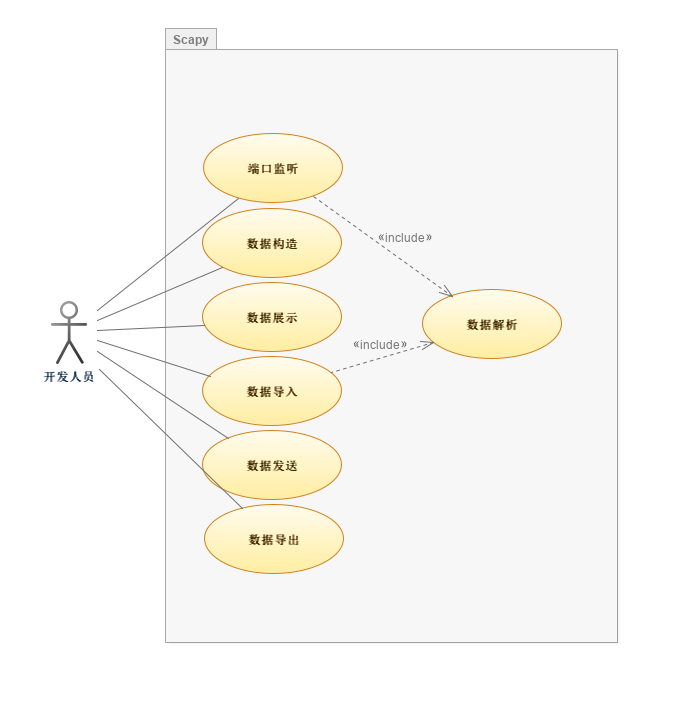


图 4.1 Scapy功能需求用例图

## 具体的功能需求和RUCM图

### 端口监听

用户可以通过调用函数，捕获指定类型的数据包。在调用相应函数时，用户需要提供以下参数：需要监听的端口或者一个端口列表（默认监听所有端口）、需要监听的报文的个数或是监听的结束时间，以此来作为停止监听的标志。Scapy需要依次配置嗅探socket、嗅探报文、处理报文，最终返回给用户这段时间内监听到的已经解析过的报文数据。

该功能的RUCM模型如图 4.2所示。



图 4.2 端口监听RUCM模型

### 数据解析

用户在调用相关函数获取报文数据之后，Scapy要能够自动调用数据解析函数对报文进行解析。在解析报文数据时，首先调用判别函数计算出协议类型，然后实例化相应的协议对象并填充协议字段值，最后返回的是已经解析好的Scapy数据包。

该功能的RUCM模型如图 4.3所示。



图 4.3 数据解析RUCM模型

### 数据构造

报文中每一层都是一个可以实例化的对象，每一层的不同的字段都可以作为属性，并且都可以进行赋值操作。Scapy需要支持按层构造数据包的功能，能够根据用户的要求实例化相应协议对象并赋值、对实例化对象进行拼接，最终返回给用户构造好的Scapy数据包。主要包括以下两个操作需求：

1）构造单层数据

用户可以对Scapy中数据包的属性进行赋值操作，通过这些对属性的赋值操作，来构造一个完整的单层数据。

2）层的堆叠

用户可以使用Scapy中的操作来将两层数据进行堆叠。用户这样做时，较低层可以根据上层重载其一个或多个默认字段（当然用户也可以单独设置想要的值）。同时用户可以将一个字符串直接当做一个原始的数据层进行操作。

该功能的RUCM模型如图 4.4所示。



图 4.4 数据构造RUCM模型

### 数据展示

用户在分析数据时需要对Scapy解析后的数据包进行展示。在调用报文数据展示模块时，Scapy会提取报文中的所需字段，封装展示内容，最终返回给用户展示结果。数据包展示主要有以下两种需求：

1) 字符展示

用户可以调用封装好的函数以字符形式展示一个Scapy数据包的所有信息。

2) 图形展示

用户可以对一个数据包或是一系列数据包进行图形展示，将报文数据中的每一层分协议展示出来，并可以将解析之后的字符串与16进制编码格式的数据对应起来，使得用户可以更直观地展示解析报文的结果。

该功能的RUCM模型如图 4.5所示。



图 4.5 数据展示RUCM模型

### 数据导入

用户需要将多种形式的数据转化为Scapy格式的数据包，Scapy要提供转化模块，实例化转化对象、最终返回给用户Scapy格式的数据包。需要转化的类型主要有：pcap文件格式、16进制编码格式以及二进制字符串格式。

1) pcap文件读取

用户可以使用封装好的函数来直接读取pcap文件。用户需要提供给pcap读取函数文件的位置来让Scapy去读取pcap文件，同时用户可以设置参数来控制读取的报文个数，默认读取pcap中的所有报文。

2) 16进制编码格式

用户可以通过转化函数将16进制的编码格式再转回为Scapy的数据包格式。

3) 二进制字符串格式

用户可以将二进制字符串格式的数据导入为Scapy格式的数据包。

该功能的RUCM模型如图 4.6所示。



图 4.6 数据导入RUCM模型

### 数据导出

用户需要将Scapy格式的数据包转化为多种形式的数据。Scapy要提供转化模块，实例化转化对象，最终返回给用户所需的数据格式。需要支持的数据类型主要有：pcap文件格式、16进制编码格式以及二进制字符串格式。

1) pcap文件导出

开发人可以将捕捉到的数据包存储为pcap文件以便以后使用，或者应用到其他地方。用户在使用时只需将存储的文件的地址以及所需要存储的数据包作为参数交给转化函数即可。

2) 转为16进制编码格式

用户可以调用转换函数将获取到的数据包转换为16进制编码形式。

3) Scapy数据包转为二进制字符串

用户可以将Scapy数据包导出为二进制字符串格式，只需要将Scapy数据包传递给二进制转化函数，即可返回数据包对应的二进制字符串格式的数据。

该功能的RUCM模型如图 4.7所示。



图 4.7 数据导出RUCM模型

### 数据发送

用户可以将组织好的Scapy数据包发送出去，发送方式需要分为两种：一种是只发送数据不接收响应数据，另一种是在发送数据的同时接收响应数据。

1）只发送数据包

用户可以通过发送函数将数据包发送出去，同时用户可以选定数据包发送的的IP地址以及链路层协议，并且可以查看有哪些报文已经成功发送。

2）发送数据包并接收响应数据

用户可以通过发送接收函数来发送数据包并且接收响应数据，发送接收函数需要在发送报文后返回有应答的数据包及其应答数据，同时也要返回没有被应答的数据包。

该功能的RUCM模型如图 4.8所示。



图 4.8 数据发送RUCM模型

# 非功能需求

## 兼容性

作为一个数据包处理工具，Scapy会被用户使用在不同的环境和平台中，因此Scapy需要具有适应多种环境的兼容性。

### Python版本兼容性

Scapy是一个基于Python实现的功能库，而Python具有多种不同的版本，并且Python2和Python3之间的标准差异是相当大的，为了适应实际的Python生产环境，Scapy必须提供多种不同版本下的安装包。

### 操作系统兼容性

Scapy在运行时所依赖的抓包模块在不同的操作系统环境下是不同的，例如在Windows下系统提供了WinPcap/Npcap模块来提供访问网络底层的接口，而在Linux下则是由tcpdump命令来实现类似功能。

为了实现操作系统层面的兼容性，需要Scapy提供不同操作系统下的底层访问接口，从而实现在Windows下调用WinPcap/Npcap截获数据包，在Linux调用tcpdump截获数据包。

## 可扩展性

Scapy的一个主要功能需求是实现对多种协议报文的解析和构造，而计算机领域已经施行的网络协议数量很大，在有限的开发资源下，对庞大的协议群实现完全的支持是不可行的，而且随着物理层面设备升级和网络标准的不断扩展，新的网络协议的出现是很有可能的；另一方面，在一些比较大的单位、企业和机构为了实现某些功能或者提高安全性，会采用一些自主设计的私有协议。因此Scapy需要能够提供一套协议扩展的方案，包括设计合适的类和接口，制定规范的扩展要求。

一般地，Scapy需要提供限制扩展协议的父类，并且要求扩展协议需要实现指定的方法，典型地，一个扩展协议需要实现的方法有数据解析方法，数据构造方法等。

## 容错性

某些情况下，例如某些攻击工具产生的数据包，会将中上层协议的部分填充为非法数据，在解析此类数据包时，Scapy需要将中上层非法协议层的数据保留起来，或者封装成为一个可见原始字节的对象中，这样对于用户就可直接获取到非法填充字节然后进行分析。

一般地，Scapy可提供一个保留报文原始字节信息的类，在解析中上层协议时，可先按照该层已支持的协议尝试解析，若全部解析失败，则将该层原始字节保存在该类中，用户可通过访问对象获取“非法”数据。

## 易用性

为了让用户在使用Scapy编写代码时可以进行简单的代码调试和验证，Scapy可提供一个命令行交互模式，在此模式中用户可以在类似Python交互模式的环境下执行一行简单的代码且给出运行结果，并能调用Scapy的功能函数或模块。

该模式的主要用途在于可以不用编写完整代码，就能方便快捷地验证一些结果，查看Scapy功能函数的效果。

该模式使得Scapy既可以当做Python库引入已有的工程，又能当成一个独立的命令行小工具使用。

# 运行要求

## 软件要求

1. 操作系统：
   * 1. Windows 7及以上： 已安装Npcap/WinPcap，
     2. Ubuntu 16.04及以上： 已安装tcpdump
2. Anaconda（可创建多版本Python虚拟环境），Python 2.7+，Python 3.4+
3. 第三方Python库：matplotlib，pyx，vpython，cryptography等

## 硬件要求

1. CPU：Intel i5-4590及以上，AMD Ryzen 3 1200及以上
2. 内存：4G内存及以上（系统最低2GB，Scapy报文解析较占用内存空间）
3. 硬盘：20G硬盘及以上（Windows 7 64位安装最低配置）

# 改进方案设想

## 改进后的用例模型

在Scapy原有功能的基础之上，本项目准备增加三个新的功能：文件切分、监听持久化和协议类型获取。改进后Scapy的用例模型如图 7.1所示。

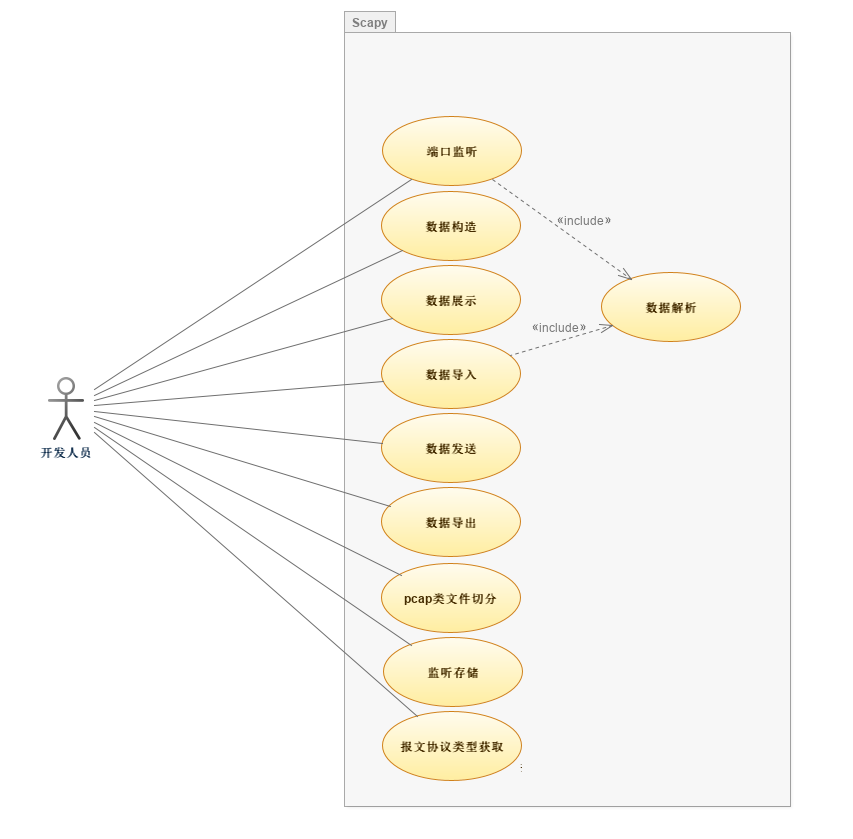


图 7.1 改进后的用例图

## 需求分析

### 文件切分功能

pcap文件的报文使用字节码存储，所占用的存储空间很小。但对于解析后的报文数据而言，每条报文都包含大量对象、字段和方法，使得解析后的数据占用的存储空间是原始报文大小的数十倍。

由于解析pcap文件对内存造成极大的负担，而且内存占用率过高会影响程序的运行效率，那么，对于较大的pcap文件，将其切分成若干个小文件，再进行批处理非常必要。

该功能的RUCM模型如图 7.2所示。



图 7.2文件切分RUCM模型

### 监听持久化功能

Scapy的端口监听模块sniff()提供了非常丰富的监听功能，包括BPF筛选规则、自定义筛选规则和针对每条报文的自定义处理函数等，但其在设计上有一个很大的缺陷。当用户需要获取Scapy监听到的报文数据时，sniff()模块会将所有监听到的符合要求的报文进行解析处理后保存在内存中，等到监听结束时再返回给使用者。这使得sniff()模块无法处理大规模的报文流量，否则会因为内存泄漏被系统停止运行。

因此，当用户需要长时间运行端口监听模块或者监听大规模流量时，Scapy需要提供一个额外的功能，使用户能够在监听的过程中分批存储截获到的报文，及时释放内存。

该功能的RUCM模型如图 7.3所示。



图 7.3 监听持久化RUCM模型

### 协议类型获取功能

Scapy根据报文的协议层次将报文数据分层解析，解析的结果也按照协议层堆叠起来。解析结果的每一层是一个单独的对象，拥有一个payload变量，用于保存下一层协议的对象实例。

比如一条DNS报文pkt，其解析结果是一个物理层的Ether对象实例，pkt.payload是一个网络层的IP对象实例，pkt.payload.payload是一个传输层的UDP对象实例，依次类推。

在处理多协议流量数据时，用户通常需要对不同协议类型的报文进行不同的处理操作。但用户无法直接从Scapy解析的结果中获取报文的协议类型，必须自行设计一种判定逻辑来得到报文的协议类型。因此，为解析后的报文对象内置一个获取协议类型的函数可以为用户提供极大的方便。

该功能的RUCM模型如图 7.4所示。



图 7.4 协议类型获取RUCM模型

### 会话提取功能

首先我们需要对会话的概念加以解释，此处的会话与HTTP中的会话有所区别。一次会话由互相通信的两个实体之间的交互过程组成。根据会话的通信性质，可将会话的两个实体分为客户端和服务端。一次会话过程由客户端和服务端的多次交互组成，并且客户端的一次请求总是对应于服务端的一次回应。图 7.5展示了客户端和服务器端的两次会话，在客户端向服务端发起TCP连接请求并被接受后，服务端和客户端开始了基于TCP和SMTP会话，图片中反映了客户端和服务端的多次交互过程。可以看到，由一个或多个相互交互的报文构成了会话，两次通信的时间间隔如果超过某一个阈值，将会认为是两次会话。

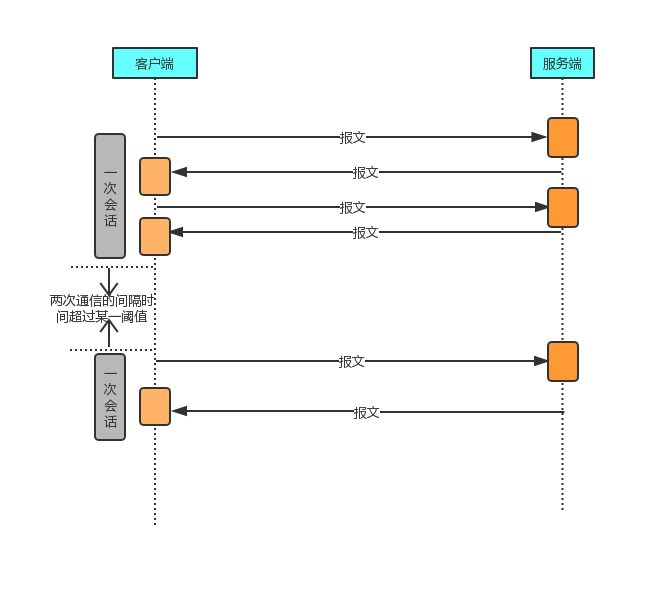


图 7.5 会话示意图

一个pcap文件中包含大量会话，有时我们只需要其中一部分或是只需要从某一IP发往另一个IP的会话。基于这种应用场景，我们认为有必要增加会话提取功能：从指定的pcap文件中提取出符合特定条件的会话，并进行存储。

最常用的会话提取条件是四元组提取，四元组指的是源IP地址、源端口、目的IP地址和目的端口四项条件。根据这四个条件用户可以将指定IP和端口的会话提取出来，以进行进一步的分析。另外，有时用户需要将一定间隔时间内的会话数据提取出来放入一个文件中。比如当用户做像Web服务流量监控的任务时，希望分析所观察到的传入数据流，采用时间窗口进行处理比较适当。应用时间窗后一次会话不仅受到两次通信间隔时间的影响，用户设置的时间窗大小还会限制一次会话的最长时间。在实际应用中，用户可以根据应用场景调整窗口大小。

该功能的RUCM模型如图 7.6所示。



图 7.6 会话提取RUCM模型

## 方案设想

### 文件切分功能

Scapy提供了wrpcap()函数用于生成pcap文件，以及PcapReader()类用于读取pcap文件的内容。本项目拟通过这两部分封装split\_pcap()函数，函数流程如图 7.7所示。

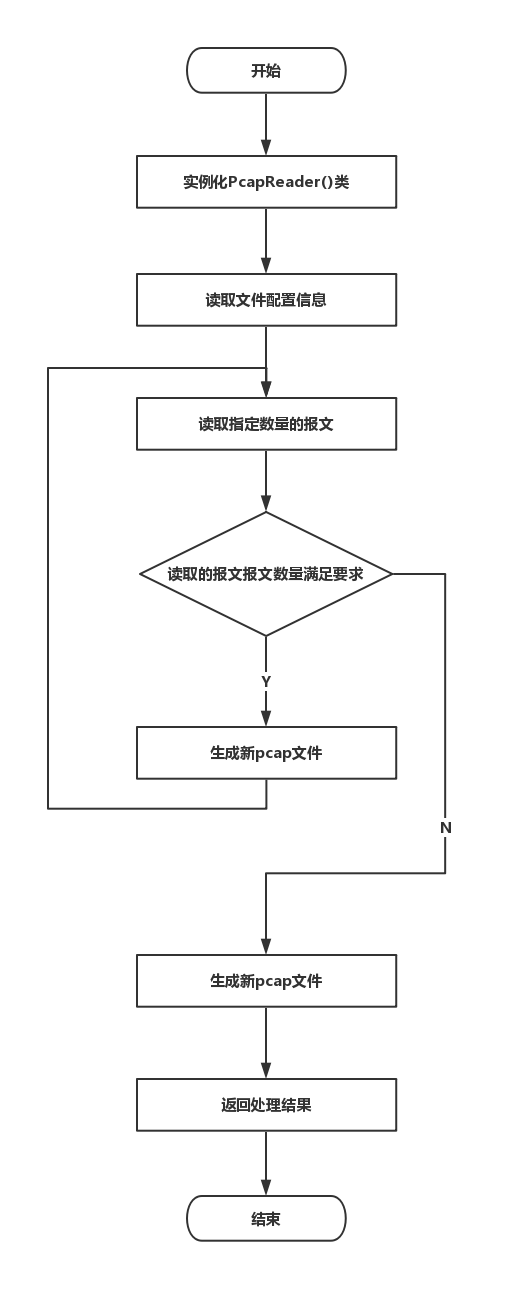


图 7.7 文件切分内部流程

### 监听持久化功能

为了不影响已有端口监听模块，本项目拟单独封装一个模块sniff\_store()，并在原有的sniff()模块基础上增加两个参数batch\_count和res\_path，分别指定存储的报文个数以及pcap文件的存放目录。另外，在原有的sniff()模块中有一个store参数，该参数决定是否将截获到的报文保存至内存中，最后再返回给用户。对于新设计的sniff\_store()模块，本身的目的就是为了防止内存泄漏，因此不再需要store参数。

sniff\_store()模块的流程如图 7.8所示。

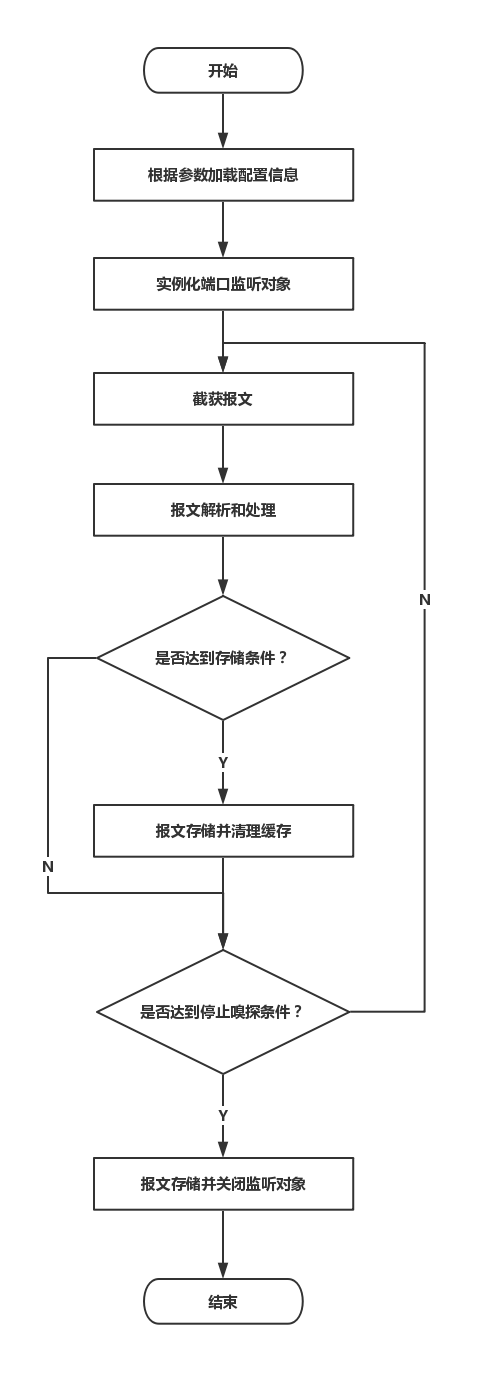


图 7.8 监听持久化内部流程

### 协议类型获取功能

Scapy的报文是Packet()子类的实例，本项目拟在Packet()类中实现get\_pro()函数，这样解析后的报文可以直接调用自身的get\_pro()函数获取报文的协议类型。

get\_pro()函数可以获取报文所有payload的name，也就是各层协议的名称，再设计一个判断逻辑以确定该报文的协议类型。函数的判断流程如图 7.9所示。

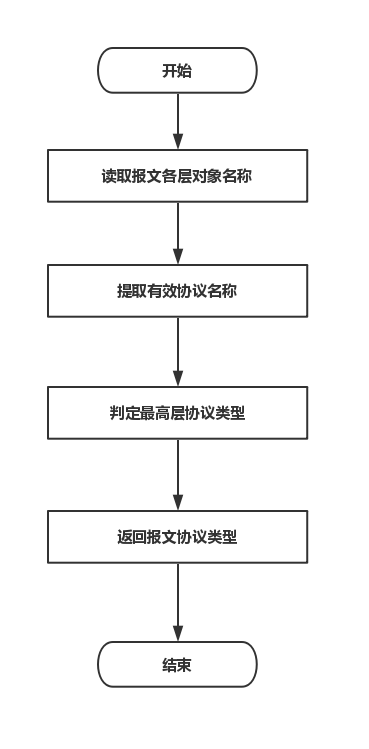


图 7.9 协议类型获取流程

### 会话提取功能

本项目拟采用字典的方式实现报文会话提取功能。用户设置的源端口、目的端口、源IP和目的IP作为字典的主键，再将pcap数据中的报文写入字典中。结合该字典和时间窗口可以将pcap数据划分为多个独立的会话，再以文件形式进行存储。模块的简单流程设计如图 7.10所示：

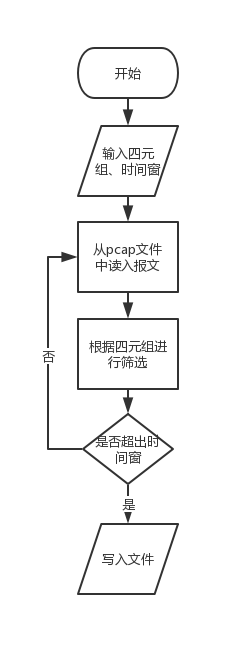


图 7.10 会话提取流程图

# 参考文献

1. 邹欣.构建之法 现代软件工程[M]. 北京:人民邮电出版社, 2017. 157-158.
2. Scapy Documentation [EB/OL]. <https://scapy.readthedocs.io/en/latest/>
3. BIONDI P. Packet generation and network based attacks with Scapy[J]. CanSecWest/core05, 2005.
4. 谢希仁, 谢钧. 计算机网络教程[M]. 人民邮电出版社, 2002.