**基于Scapy的分析与扩展**

**需求规格说明书**

Version 1.4

小组成员：

陈鸿超

李铎坤

刘颖

袁梦阳

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 |
| v1.0 | 2019/03/12 | 刘,袁,李,陈 |  | 刘,袁,李,陈 | 初稿 |
| 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 |
| v1.2 | 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 |
| v1.3 | 2019/0320 | 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 |
| V1.4 | 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 |

目录

[1 引言 1](#_Toc4154201)

[1.1 目的 1](#_Toc4154202)

[1.2 软件需求分析目的 1](#_Toc4154203)

[1.3 文档概述 1](#_Toc4154204)

[1.4 术语和缩略词 1](#_Toc4154205)

[2 整体概述 3](#_Toc4154206)

[2.1 项目概述 3](#_Toc4154207)

[2.2 项目包结构分析 3](#_Toc4154208)

[2.3 用户定义 4](#_Toc4154209)

[3 用户需求 5](#_Toc4154210)

[3.1 用户需求概述 5](#_Toc4154211)

[3.2 报文获取 5](#_Toc4154212)

[3.3 报文构造 5](#_Toc4154213)

[3.4 报文存储 5](#_Toc4154214)

[3.5 报文发送 5](#_Toc4154215)

[3.6 报文展示 6](#_Toc4154216)

[4 功能需求 7](#_Toc4154217)

[4.1 用例模型 7](#_Toc4154218)

[4.2 具体的功能需求和RUCM图 7](#_Toc4154219)

[4.2.1 端口监听 7](#_Toc4154220)

[4.2.2 数据解析 9](#_Toc4154221)

[4.2.3 数据构造 9](#_Toc4154222)

[4.2.4 数据展示 10](#_Toc4154223)

[4.2.5 数据导入 11](#_Toc4154224)

[4.2.6 数据导出 13](#_Toc4154225)

[4.2.7 数据发送 14](#_Toc4154226)

[5 非功能需求 16](#_Toc4154227)

[5.1 兼容性 16](#_Toc4154228)

[5.1.1 Python版本兼容性 16](#_Toc4154229)

[5.1.2 操作系统兼容性 16](#_Toc4154230)

[5.2 可扩展性 17](#_Toc4154231)

[5.3 容错性 17](#_Toc4154232)

[5.4 易用性 18](#_Toc4154233)

[6 运行要求 19](#_Toc4154234)

[6.1 软件要求 19](#_Toc4154235)

[6.2 硬件要求 19](#_Toc4154236)

[7 改进方案设想 20](#_Toc4154237)

[7.1 需求分析 20](#_Toc4154238)

[7.1.1 文件切分功能 20](#_Toc4154239)

[7.1.2 嗅探存储功能 20](#_Toc4154240)

[7.1.3 协议类型获取功能 21](#_Toc4154241)

[7.2 方案设想 22](#_Toc4154242)

[7.2.1 文件切分功能 22](#_Toc4154243)

[7.2.2 嗅探存储功能 23](#_Toc4154244)

[7.2.3 协议类型获取功能 24](#_Toc4154245)

[8 参考文献 26](#_Toc4154246)

# 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

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。软件的需求分析主要包括以下几个模块：用户需求、功能性需求、非功能性需求、运行要求，另外还需要对需求进行可行性分析。

## 文档概述

文档用途：本文档主要是介绍scapy的需求及规格说明。

主要内容如下：

* 描述Scapy的用户需求
* 以用例图的形式给出Scapy的功能需求，并对用例模型进行详细的描述
* 使用RUCM模型对功能需求进行建模
* 描述Scapy的非功能性需求
* 描述Scapy运行所需的硬件环境要求
* 描述Scapy运行所需的软件环境要求

## 术语和缩略词

|  |  |  |
| --- | --- | --- |
| 编号 | 术语 | 说明 |
| 1 | UCM | 用例建模 |
| 2 | RUCM | 限制性用例模型 |
| 3 | Python | 一种面向对象的解释型计算机程序设计语言 |
| 4 | 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。 |
| 5 | 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 |
| 6 | 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。端口监听还用于实现对共享目录访问的监测和控制 |
| 7 | 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 |
| 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 | pcapng | PCAPNG是PCAP Next Generation Dump File Format的缩写，是常用的数据报存储格式 |
| 10 | Eth | 物理层协议 |
| 11 | ARP | Address Resolution Protocol，地址解析协议，是根据 IP地址获取物理地址的一个TCP/IP协议 |
| 12 | IPV4 | Internet Protocol version 4，互联网通信协议第四版，是一种无连接的协议，操作在使用分组交换的链路层（如以太网）上。 |
| 13 | IPV6 | Internet Protocol Version 6，互联网通信协议第六版，是互联网工程任务组设计的用于替代 IPv4 的下一代IP协议 |
| 14 | 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 |
| 15 | TCP | Transmission Control Protocol ，传输控制协议，是一种面向连接的、可靠的、基于字节流的传输层通信协议 |
| 16 | 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 |
| 17 | SNMP | Simple Network Management Protocol，简单网络管理协议，由一组网络管理的标准组成，包含一个应用层协议（application layer protocol）、数据库模型（database schema）和一组资源对象 |
| 18 | SMB | Server Message Block，被用于Web连接和客户端与服务器之间的信息沟通 |
| 19 | eps | 用PostScript语言描述的一种ASCII图形文件格式，在PostScript图形打印机上能打印出高品质的图形图像 |

# 整体概述

## 项目概述

Scapy 是一个用于计算机网络的数据包处理工具，最初由Philippe Biondi用python编写。它允许用户能够发送，嗅探和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操作程序。

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。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hping，arpspoof，arp-sk，arping，p0f甚至是Nmap，tcpdump和tshark的某些功能模块。

Scapy的特点有如下几点：

* 可以完整地返回它收到的所有信息。
* 可以自由地构建所需的数据包，对数据包进行填充、任意修改字段值，并按照需要进行堆叠。
* 支持大量报文的解析和协议的构建。

## 项目包结构分析

* 插件调用目录：arch

Scapy的部分功能在实现时需要使用其他插件，比如tcpdump、winpcap等，同时也需要获取当前系统的一些硬件信息，arch目录下就封装了实现这些功能的各种模块。

* 网络标准目录：asn1

在计算机网络领域，编码、传输、解码采用的是ASN.1(Abstract Syntax Notation one)标准，asn1目录下封装了实现该标准的相应模块。

* 扩展模块目录：contrib

由非官方开发者开发的扩展功能模块，包括新的协议模块和一些扩展功能模块，经过官方审核测试之后，就好放在contrib目录下，在使用时可以显示的去调用这些模块。

* 报文格式目录：layers

Layers目录下封装了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，以及一些必要的解码方式。

* 基本功能模块目录：modules

Scapy的很多功能在实现时都需要调用一些共同的比较底层的基本模块，modules目录下就封装了需要显示加载的各种基本功能模块。

* 测试工具目录：tools

tools目录下包含为了测试各部分功能而准备的功能模块。

* 配置模块 ：config.py

Scapy在启动时会调用该模块，获取运行环境信息，初始化其他功能模块所需的配置参数。

* 协议字段格式模块：field.py

该模块封装了各种类型字段的格式信息，包括字段数据类型，大小等。

* 报文对象模块：packet.py

该模块封装了报文对象所拥有的各种参数和方法，所有的协议模块都继承于此模块。

* 报文发送/接收/嗅探模块：sendrev.py

该模块封装了与报文发送、接收和嗅探相关的所有功能函数。

* 工具库模块：utils.py

该模块封装了Scapy中常用的一些功能，比如pcap文件的读取和写入、数据类型转换、插件调用等。

## 用户定义

Scapy的用户定义为开发人员。

# 用户需求

## 用户需求概述

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开源项目扩展，面向人群为开发人员，主要目的是便于开发人员更方便快捷的进行报文的获取、解析和发送等操作。为了满足开发人员的需要，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，我们对Scapy进行了用户需求分析，用户的诉求主要有如下几点：报文获取、报文构造、报文存储和报文展示。用户通过报文获取的功能可以实现网路发现等功能，是后续对报文操作的基础。 用户通过报文构造的功能可以进行网络数据包的伪装以及构建探测、攻击网络的工具。同时，在获取报文之后，用户还需要对报文进行格式化存储，方便以后读取、分析报文。展示报文可以直观地展示用户关心的数据，是一项重要的辅助功能。

## 报文获取

Scapy应该支持尽可能多层面的数据拦截，包括网卡层面的拦截以及应用层和Socket层面的拦截。网卡层面可以截获和本机同网段的计算机发送的信息，包括送往自己的数据包。应用层的拦截主要是利用系统Hook方式进行实现。除此之外，Scapy还应该支持从文件中获取报文的功能。

## 报文构造

为了进行网络测试或分析网络数据等，用户需要模拟不同类型网络攻击，该功能的实现需要Scapy具有构造报文的功能。

## 报文存储

在进行报文的抓取之后，用户获得了目标数据，此时用户可以进行报文的解析等操作，同时也有报文存储的需求。存储的报文需要有可扩展性、可移植性以及可追加数据等特点。目前pcapng等主流报文格式满足该要求，因此，Scapy需要支持pcapng等格式的报文存储。

## 报文发送

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，用户可能会需求将该报文发送至特定主机，满足用户的特定需求，比如加密数据传输、攻击模拟等。因此。Scapy需要支持报文发送的功能。

## 报文展示

报文展示虽然是一项辅助功能，但却是一项重要的用户需求。简介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主要可以提供给开发人员以下七部分功能：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。

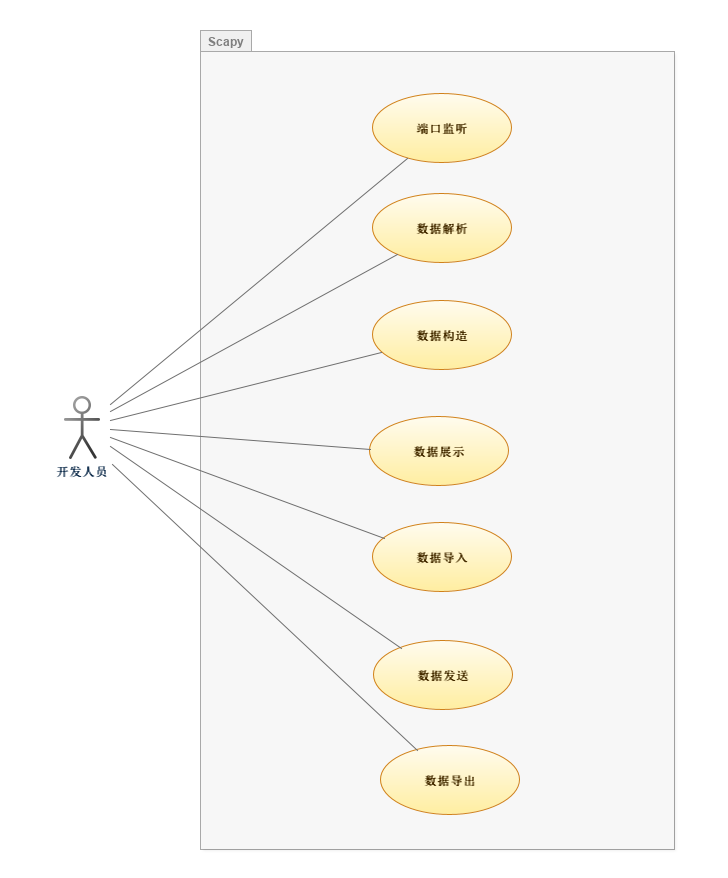


图 4.1 Scapy功能需求用例图

## 具体的功能需求和RUCM图

### 端口监听

开发人员可以通过调用函数，捕获一些数据包。开发人员提供给所调用的函数以下参数：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，最终返回给开发人员这段时间内监听到的已经解析过的报文数据包。



图 4.2 端口监听RUCM模型

### 数据解析

开发人员在使用Scapy进行读入本地数据文件或者是监听端口得到数据操作后，Scapy可以自动调用数据解析函数对报文进行解析，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。



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可以作为属性，并且都可以进行赋值操作。同时开发人员可以使用Scapy按层来构造数据包。Scapy需要依次实例化相应协议对象并赋值、对实例化对象进行拼接，最终返回给开发人员构造好的Scapy数据包。主要包括以下两个操作需求：

1）构造单层数据

开发人员可以对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

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



图 4.4数据构造RUCM模型

### 数据展示

开发人员需要可以对Scapy数据包进行展示。Scapy需要依次调用报文数据展示模块、在报文中提取所需字段、封装展示内容，最终返回给开发人员展示结果。数据包展示主要由以下两种需求：

1) 字符展示

开发人员可以使用调用数据包的方法来字符形式展示一个Scapy数据包。

2) 图形展示

开发人员需要可以对一个数据包或是一系列数据包进行图形展示，要求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，使得开发人员可以更直观地展示解析报文的结果。



图 4.5 数据展示RUCM模型

### 数据导入

开发人员需要可以将多种形式的数据转化为Scapy格式的数据包。Scapy需要依次调用转化模块、实例化转化对象、最终返回给开发人员Scapy格式的已经解析过的数据包。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

开发人员可以使用rdcap()函数来直接读取PCAP文件。开发人员需要提供给PCAP读取函数文件的位置来让Scapy去读取PCAP文件。同时开发人员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

开发人员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

开发人员可以将二进制字符串格式的数据导入为Scapy数据包。



图 4.6 数据导入RUCM模型

### 数据导出

开发人员需要可以将Scapy格式的数据包转化为多种形式的数据。Scapy需要依次调用转化模块、实例化转化对象、最终返回给开发人员数据格式。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文件导出

开发人可以将捕捉到的数据包存储为PCAP文件来作为以后的使用，或者应用到其他地方。开发人员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

开发人员可以将获取到的数据包转换为16进制编码形式。开发人员通过16进制转化函数来将数据转化为16进制格式。

3) Scapy数据包转为二进制字符串

开发人员可以将Scapy数据包格式导出为二进制字符串格式。开发人员将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。



图 4.7 数据导出RUCM模型

### 数据发送

开发人员可以将组织好的Scapy数据包发送出去。其中发送方式需要分为两种：一种是只发送数据不接收返回数据。

1）只发送数据包

开发人员可以通过发送函数将数据包发送出去。其中开发人员可以选定数据包发送的的IP地址以及链路层协议。同时开发人员需要可以查看有哪些报文成功已经发送。

2）发送数据包后接收回答

开发人员可以通过发送接收函数来发送数据包并且接受返回的回答。发送接收函数需要在发送报文后返回多个数据包和回答，以及没有被回答的数据包。





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会被开发者或者攻击实验人员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。

而提供Python兼容性用例的RUCM图如图5.1所示：



图 5.1 提供Python兼容性RUCM模型

### 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能力，而在Linux则是使用tcpdump命令来实现对网络数据包的截获。

为了实现操作系统层面的兼容性，需要Scapy提供对不同的操作系统有不同的底层访问接口，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的解析和构造，而计算机领域的已经施行网络协议数量很大，在有限的开发资源下，实现对庞大的协议群进行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，设计合适的类和接口，制定规范的要求。

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剖析字段方法，构建字段方法等。在方案完成的基础上，开发人员扩展新协议的用例RUCM图如图5.2：



图 5.2 扩展新协议RUCM模型

## 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剖析，若全部剖析失败，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界面，在此界面中用户可以在类似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。

该需求使得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

* 操作系统：

Windows 7及以上： 已安装Npcap/Winpcap，

Ubuntu 16.04及以上： 已安装tcpdump

* Anaconda（可创建多版本Python虚拟环境），Python 2.7+，Python 3.4+
* Pip库：matplotlib，pyx，vpython，cryptography等

## 硬件要求

* CPU：暂无要求，主流性能即可
* 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）
* 硬盘：20G硬盘及以上（windows 7 64位安装最低配置）

# 改进方案设想

## 需求分析

### 文件切分功能

在pcap文件中，报文是使用字节码存储的，所占用的存储空间是非常小的。但对于解析后的报文数据而言，每条报文都会包含非常多的对象、字段和方法，这就使得解析后的数据所占用的存储空间是原始报文大小的数十倍。

这个问题对于内存来说是一个极大的挑战，并且内存占用过多对程序的运行效率也有极大的负面影响。因此，对于比较大的pcap文件，将其切分成若干个小文件，然后再进行批处理是非常必要的。

该功能的RUCM模型如图7.1所示。



图7.1文件切分RUCM模型

### 嗅探存储功能

Scapy本身的嗅探模块sniff()提供了非常丰富的功能，包括BPF筛选规则、自定义筛选规则、多种停止方式、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。

当使用者需要获取所嗅探到的报文数据时，sniff()模块会将所有嗅探到的符合要求的报文进行解析处理后保存在内存中，等到嗅探结束时再返回给使用者。这就使得sniff()模块无法处理大规模的报文流量，否则就会因为内存泄漏被系统kill掉。

因此，在需要长时间运行，嗅探大规模流量的情况下，Scapy需要提供一个额外的功能，能够在嗅探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7.2所示。

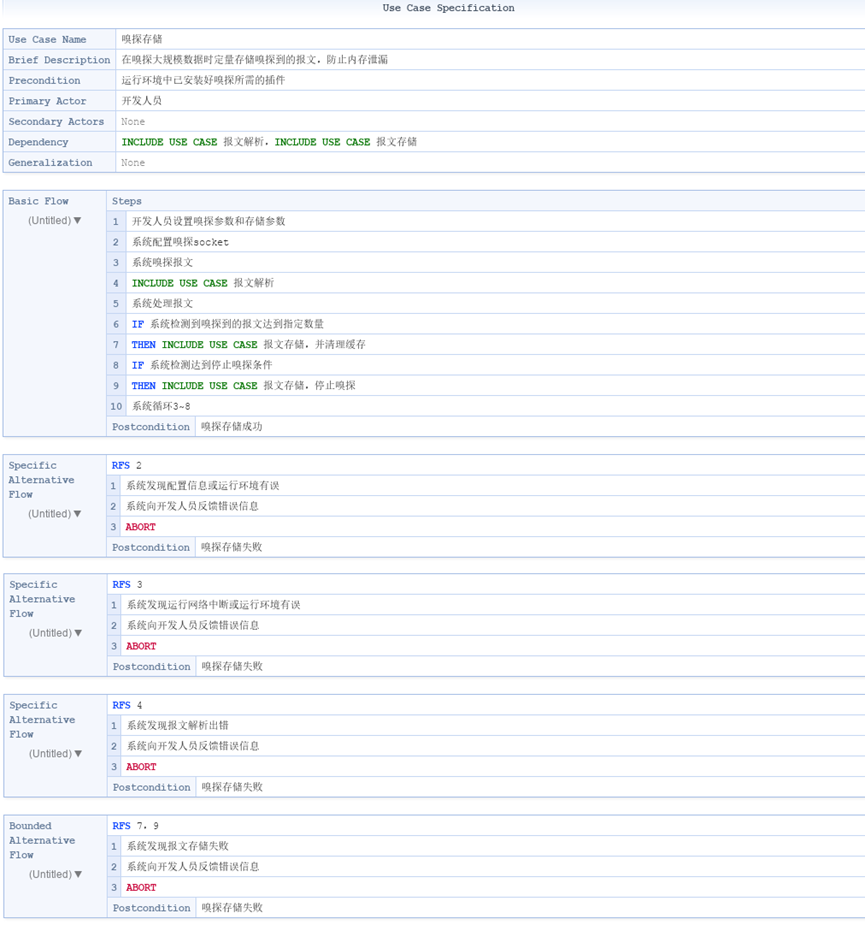


图7.2 嗅探存储RUCM模型

### 协议类型获取功能

在解析报文时，Scapy会根据协议层次划分一层一层的进行解析，解析结果也是一层层堆叠起来的。每一层都被封装成了单独的对象，该对象拥有一个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，其解析结果就是一个物理层的Ether对象实例，pkt.payload就是一个网络层的IP对象实例，pkt.payload.payload就是一个传输层的UDP对象实例，依次类推。

在处理多协议的流量数据时，使用者通常需要根据报文的协议类型对报文进行不同的处理操作。但在解析后的报文中，并没有协议类型这个字段，这就使得使用者不得不自己设计一种判定逻辑去判断报文的协议类型。因此，如果实现了这个功能，将其作为解析后的报文对象内置的一个函数，那么就可以为使用者节省掉大量的重复开发工作。

该功能的RUCM模型如图7.3所示。



图7.3 协议类型获取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，同时代码内部提供了一个PcapReader()类用于读取pcap文件的内容。

利用这两部分，我们可以封装一个split\_pcap()函数，函数内部的处理流程如图7.4所示。

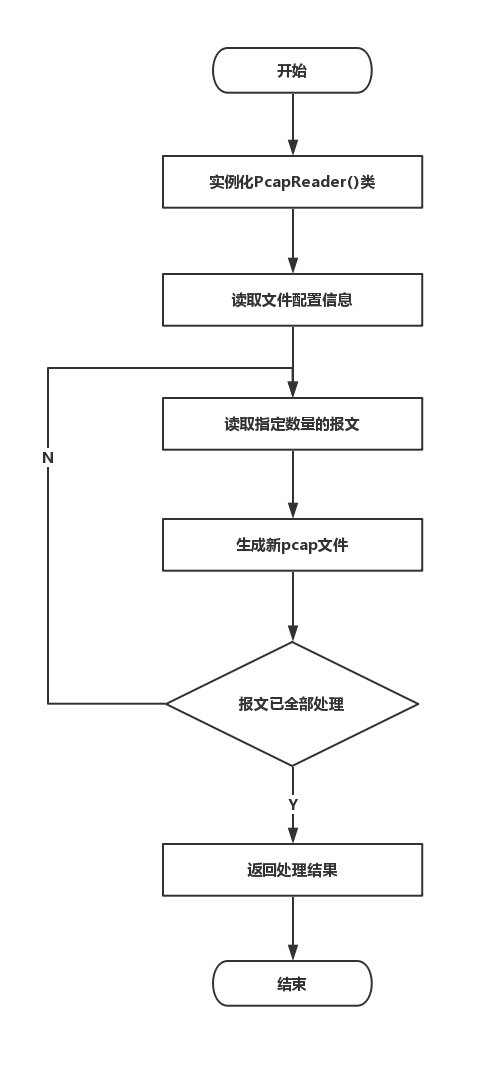


图7.4文件切分内部流程

### 嗅探存储功能

为了不影响已有嗅探模块，我们准备单独封装一个模块sniff\_store()，该模块在原有的sniff()模块基础之上增加两个参数batch\_count和res\_path，用来指定每次存储时的报文个数以及pcap文件的存放目录。同时，去掉store参数，不再将所有截获到的报文数据返回给使用者。

sniff\_store()模块内部的大致处理流程如图7.5所示。

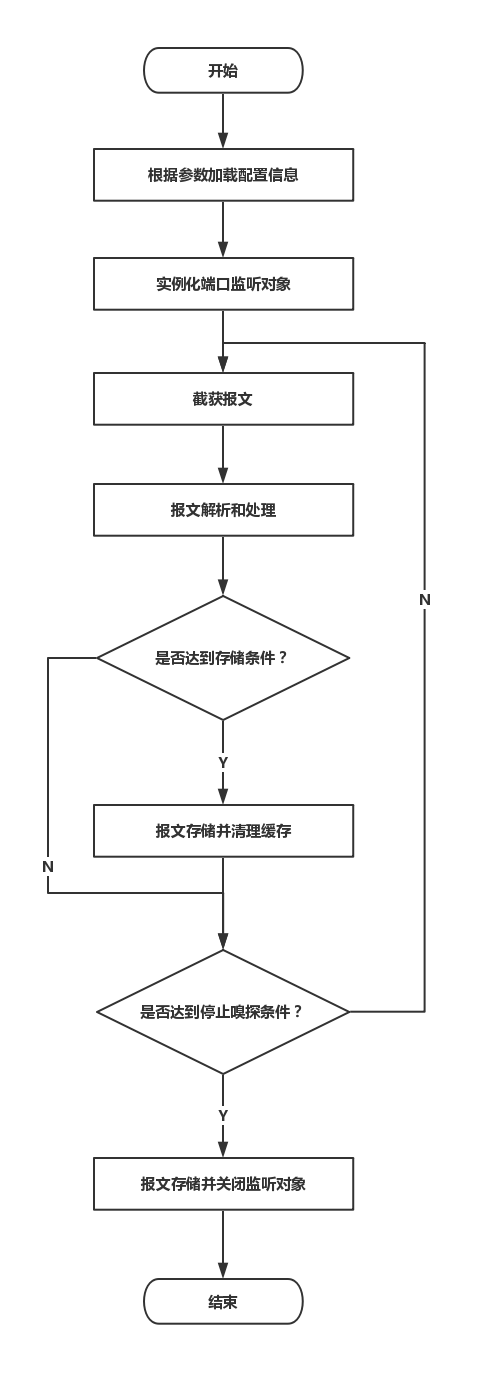


图 7.5 嗅探存储内部流程

### 协议类型获取功能

在Scapy中，所有的报文都是Packet()子类的一个实例，因此，只需要在Packet()类中实现一个get\_pro()函数，这样所有解析后的报文都可以直接调用自身的get\_pro()函数获取报文的协议类型。

而在get\_pro()函数内部，我们可以获取报文所有payload的name，也就是各层协议的名称，然后设计一个判断逻辑确定该报文的协议类型。

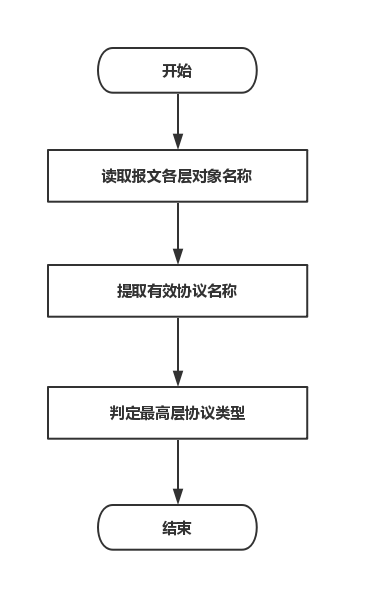


图 7.6 协议类型获取流程

# 参考文献