深度报文解析

# 项目课程目标

本次项目实例为一个真实企业中的项目裁剪版本。 取自该完整项目中最核心的“深度报文检测”模块，略加删减。

本次课程期望在学员就业之前，能够尽可能模拟企业中真实项目的开发场景。树立“项目” 不等于“多写了几行代码”的概念。

在一个项目“产品”的开发周期中，按照 4-2-4的原则，就是前期项目设计（市场分析、需求定义、概设、详设）， 中间编码， 后期验证（单元测试（bug解决）、集成测试（bug解决）、试运行） 三部分的时间划分， 整个编码仅占项目开发周期的20%。

所以，本次项目具备如下特定（目标）

* 模拟一个真实的企业项目开发场景
* 重视文档的编写，
* 了解项目开发过程中，源码组织、管理（配置管理）等的一些概念，会简述。
* 了解项目开发过程中，Bug管理的一些流程。
* 树立“入狱"10天的精神准备，切实克服困难，完成整个项目
* 深入理解TCP/IP协议栈，掌握网络报文的深度解析方法和技巧。

# 项目介绍

**目标**

* 了解项目的应用场景
* 掌握项目的关键实现技术
* 掌握项目的部署场景

## 项目名称

* 上网行为管理系统
* 网络私有应用检测系统
* 网络隐匿流量侦控系统

我们本次实现其中最核心的dpi（Deep Packet Inspection深度报文检测）模块

## 产品概述

### 为什么要研制网络隐匿流量侦控系统

隐匿通道是一种允许违背合法的安全策略进行通信的通道。网络隐匿通道中的通信具有加密和匿名性，由于普通的网络安全产品自身的局限性，这使得网络隐匿通道中的信息传递活动很难被检测到；与周知或公开协议区别，私有通信协议指自定义非公开的通信协议。目前绝大数互联网应用采用私有协议。



目前，国内外敌对势力、异见人士、不法分子通过使用隐匿通道工具（如自由门、VPN、即时通信软件等）进行境内外勾连，在境内外网站发表和散布各种谣言、煽动性信息，使用范围逐渐扩大，有些工具甚至得到了反华国家与机构的资金赞助开发，面向全球免费提供下载，而业务部门目前缺少对这些隐匿通道工具的有效监控手段，急需新的技术侦察工具。

### 产品简介

网络隐匿流量侦控系统的产品图片（参考图片）



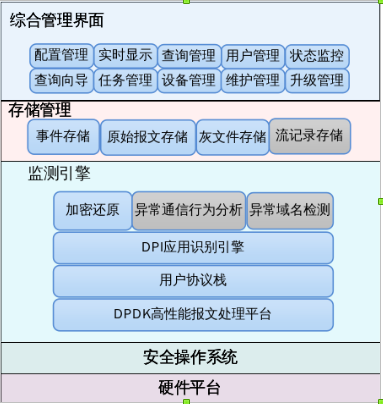
系统应基于旁路部署，通过流量监测实现对VPN类、即时通信类、翻墙类、特种木马等隐匿通道的流量识别，并在识别的基础上实现对部分应用流量的破解和还原。

### 产品定位

* 技侦：针对国内外敌对势力通过隐匿通道（如VPN、即时通信软件）进行境内外勾连而没有有限监控手段的现状，急需新的技术侦察工具。
* 网保: 需要具备针对特种木马窃密数据回传通道监测能力。
* 企业安全：针对数据防泄露（DLP）的需求，需要有可以识别私有加密应用和网络异常通信行为分析的监测产品，并能对加密流量进行还原和审计。

## 产品架构

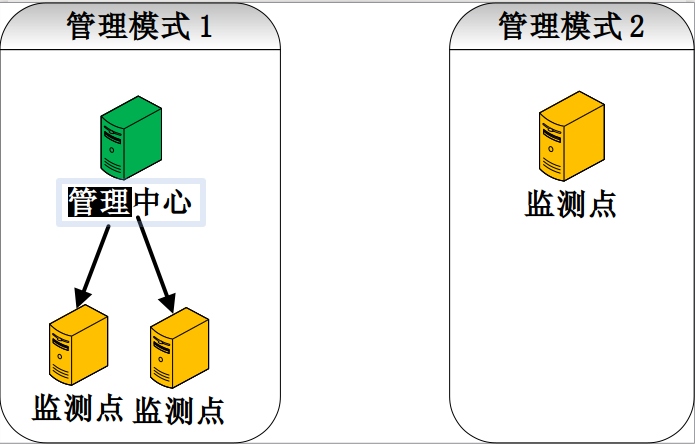
网络隐匿流量侦控系统主要由多核硬件平台、安全操作系统、监测引擎、存储管理、数据关联分析、协同分析、升级管理、维护管理、综合管理界面等功能模块组成，如下图所示：



* 硬件平台：通过硬件选型，选择高性能的网络应用处理平台，满足关口系统软件运行要求，具有良好的环境适应性和软件兼容性。
* 安全操作系统：在开源的Linux操作系统的基础上，通过内核裁剪、安全加固和强访问控制配置，满足系统的安全性和高效性要求。
* 监测引擎：完成网络报文的预处理、并提供插件管理机制加载业务支撑插件，业务支撑插件主要有原始流记录、报文记录、文件还原、域名提取、URL提取等，其主要功能是生成业务模块的数据源。对采集到的netflow原始流记录或录制的原始报文等组合应用多种网络通信行为分析技术，可发现未知攻击和未知木马。
* 存储管理：实现对多元异构数据的存储，数据包括报文特征事件、恶意代码感染事件、域名访问记录、原始流记录、异常流量事件、异常域名事件、异常行为事件等，至少存储半年的数据，并提供查询接口。
* 综合管理界面：主要包括配置管理、历史查询和实时显示、状态监控、设备管理、升级管理、维护管理等的前台配置与展示功能。

## 管理模式

网络隐匿流量侦控系统支持单点部署或自上至下的多级集中管理，监测节点部署在重要部门关口，管理中心可以管理多个本地或异地的监测点。



## 主要功能

### DPI(深度包检测)

* 支持不少于15大类，1000种以上的网络应用的识别。
* 支持HTTP、SMTP、POP3、IMAP等几十种周知协议的解析与还原。
* 加密应用识别：
  + VPN识别：支持PPTP、L2TP、OpenVPN、SSTP、IPSec、Cisco VPN的识别和上下线检测；
  + 即时通信应用识别：支持微信、QQ、陌陌、WhatsAPP、Telegram、Skype等识别。
  + 翻墙软件识别：支持Lantern、Turbo\_vpn、穿云vpn、红叶vpn、vpn\_speed、极速vpn、自由门、猎豹vpn、Shadowsocks等的识别和上下线检测。
  + 通用加密应用识别：支持HTTS、FTPS、POP3S、SMTPS、IMAPS、Gmail等的识别。

### 加密流量还原

* 支持自由门、极速VPN、PPTP（开发中）、Shadowsocks、极牙等隐匿通道的流量破解还原能力，支持Talkbox应用流量的破解还原。

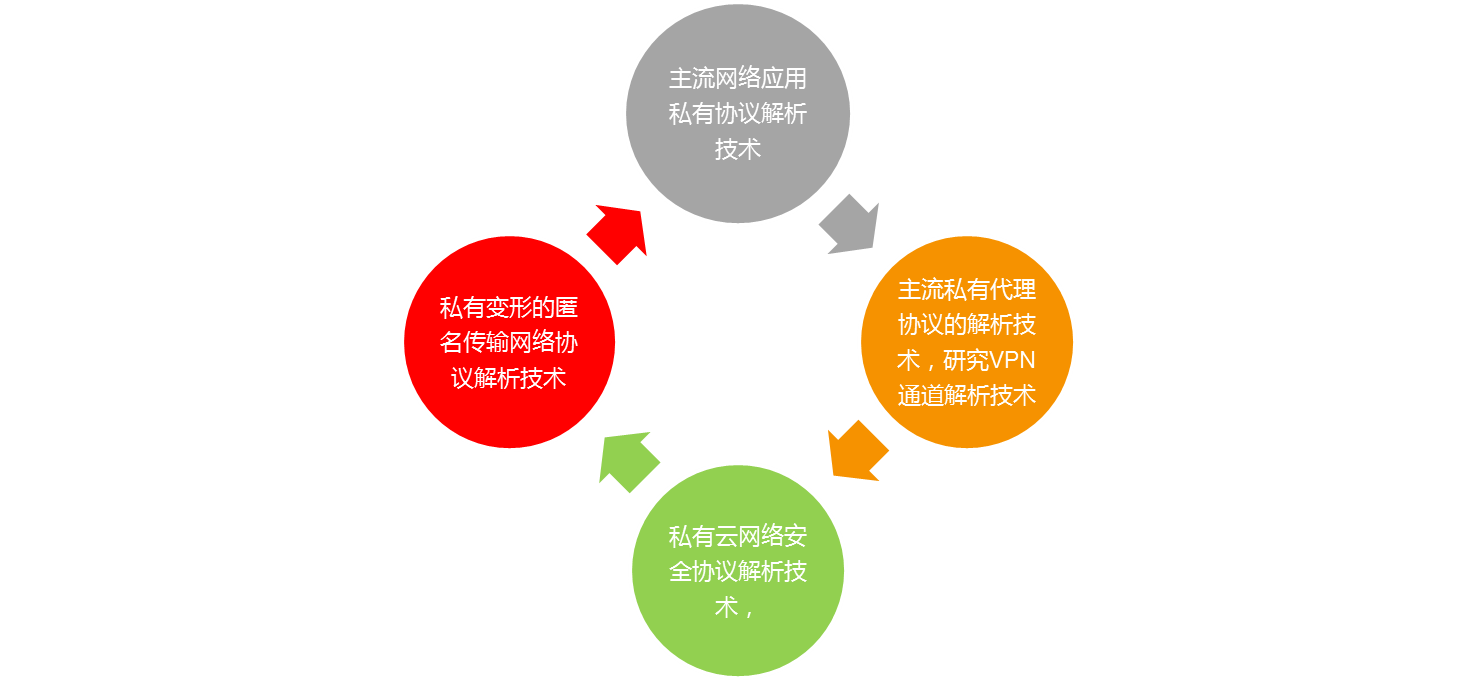
### 异常通信行为分析

* 对采集到的netflow原始流记录或录制的原始报文等组合应用多种网络通信行为分析技术，可发现未知协议和未知木马。
* 异常域名检测：通过分析域名请求、应答记录发现异常域名，留存域名访问记录元数据。
* 异常通信行为分析：发现网络异常通信行为，如内部主机和外部主机的间歇性连接、服务器主动外联、工作主机非工作时间主动外联等。

## 关键技术

### 私有应用流量深度监测技术

主要基于DPI 和文件还原与解析技术、逆向技术、关键字匹配、自然语言理解技术，重点关注四大技术：



* 本系统采用自主研发的DPI引擎来实现对网络报文的高速处理和网络应用的精确识别，采用自动化应用指纹识别技术来收集互联网应用（网站、数据库、应用系统、工业设备）的指纹信息，并构建应用指纹库。
* 采用协议的逆向技术来实现对网络应用的深度监测：
  + 对于标准公开协议，可按照协议规范实现细粒度的协议分析，提取协议中特征字段和还原协议传输内容；
  + 对于私有未公开和加密协议，通过搭建仿真环境，采用软件逆向和协议逆向方法识别其交互过程和加密算法，生成协议规范，并按照规范提取协议中的特征字段和解密还原传输内容；

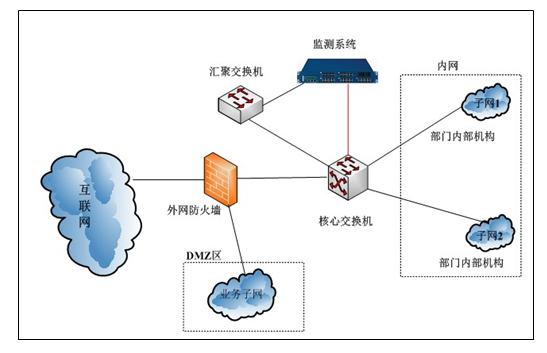
### 异常通信行为分析技术

网络异常通信行为分析是一种有效检测未知攻击的技术，重点关注攻击成功后续通信行为，包括外联行为分析、间歇会话连接分析、加密通道分析、异常域名分析、上下行流量分析等技术，

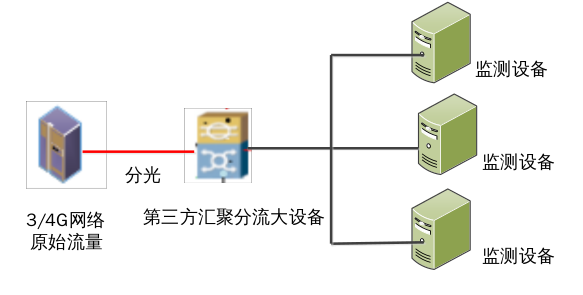
* 外联行为分析：对主要对可疑的外联行为进行分析，我们的产品内置了几十种的外联行为分析场景，如：
  + 服务器主动主动发起对外的网络连接请求和域名请求行为;
  + 邮件服务器发起的DNS请求不是mx记录；
  + 内网工作主机在非工作时间连接外部服务器行为；
  + 内网主机发起对非周知端口的域名请求；
  + URL中包含特殊字符串；
  + 重要主机通过非周知端口连接国外或台湾服务器行为；
  + 内网通过周知端口对外发起私有协议通道。
  + ……
* 间歇会话连接分析：木马的被控端一般会主动、周期性地连接控制端，以更新状态、获取新的指令，连接的周期和频率较为固定，间歇会话连接分析可以检测到主机的间歇性链接行为，从而发现木马被控端。
* 上下行流量比对比分析：主要是对比分析上下行的通信流量，正常的通信行为，下行流量较高或基本相当（如P2P流量），而窃密木马因为需要回传数据，在主动外联后，大多数时候上行的流量会比下行流量高。
* 加密通道分析：基于网络加密流量的识别技术，可以识别网络流量为加密还是非加密的，如果某个非周知端口突然出现了加密流量，或者某周知端口本应传输明文流量，突然出现了加密流量，可以认为存在异常。
* 异常域名检测：主要用于发现僵尸网络或木马等恶意代码被控制端对其控制端服务器的动态域名请求行为，是依据域名名字长度、域名名字中存在的特殊字符数、域名对应的IP地址为特殊地址的个数、域名对应的解析次数的突变度、域名解析次数的周期性、域名对应的IP池个数、域名对应IP池的跳变时间属于特定时段的次数、 域名对应的IP池的变化次数等异常特征对域名记录进行检测，发现异常域名。

## 典型部署

* 典型部署方式1如下图所示，网络隐匿流量侦控系统部署在通过交换机镜像（或者分光）的方式将核心交换机的流量接入网络隐匿流量侦控系统的捕包网卡。网络隐匿流量侦控系统通讯网卡需要接到内网汇聚交换机上， 并可以通过防火墙设置访问外网的权限。



* 典型部署方式2，适合骨干网监测，将骨干网上的数据通过分光点引接到汇聚分流设备，由汇聚分流设备过滤后将流量分流到多个监测设备。



## 随堂练习

* **问答**
  + 关于“深度报文检测”模块， 你能再想出几种可能利用DPI模块的项目
* 一般企业中会从“高层设计”来开启一个项目的研发。
  + 你如何理解“高层设计”文档， 或者， 你编写过类似文档吗？
    - 高层设计文档，目的在于说清楚要做的“这个产品”， “这个项目”是什么， 在某种意义上，等同于需求定义。

# Bub追踪系统

* **目标**
  + 掌握在项目开发过程中， 缺陷（Bug)的处理流程

## Bug追踪系统简述

在软件开发生命周期（SDLC）跟踪错误是其中一个最重要的步骤没有这一步软件可以是不完整的。

这是一个很大的节省时间的添加和管理的错误是在Bug跟踪系统。很少的错误追踪系统不仅跟踪的错误，但也完全基于软件的项目管理与可用于许多其他任务。

## Bubzilla

Bugzilla是基于Web的项目管理软件作为开源软件发布。

Bugzilla是用于管理软件的开发，并帮助您在软件开发过程中得到一个手柄。

Bugzilla是强大的指挥工具，让你的团队组织和有效的沟通。它是允许有效跟踪漏洞和代码更改。这是由Mozilla基金会开发的。此问题跟踪工具使用了许多像Mozilla中，脸谱，美国航空航天局，开放式办公，红帽等最受好评的组织

Bugzilla的成功的项目往往是成功的组织和沟通的结果。

Bugzilla的好处：使用Bugzilla的帮助，以提高产品质量，更好地与团队成员沟通，有助于提高客户满意度，Bugzilla的可以提高软件开发过程中的生产力。



**特点**

* Bugzilla是在较短的时间内得到普及，因为它的特点。
* 提高可扩展性和性能，因为优化数据库结构。
* 高安全性
* 支持先进的查询工具和保存查询
* 注册成立的Bugzilla的电子邮件通知功能
* Bugzilla的允许基于用户配置文件，用户也可以添加其他的电子邮件ID设置电子邮件偏好。
* 优秀的权限系统

**参考链接**

* <https://bugs.kali.org/my_view_page.php?refresh=true>
* <https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=904821>
* <https://www.debian.org/security/>

## mantisbt

中文译作 “蟑螂Bug追踪”

螳螂Bug跟踪是一个开源的基于web的Bug跟踪系统。这是写在PHP和多个数据库，如MS SQL，MySQL和PostgreSQL的作品。

螳螂有多个项目，潜入多层次如下：

项目 - >子项目 - >分类 - >错误

基于用户访问权限，权限的用户可以对每个项目做出贡献。

螳螂是强大的工具，集成了几个应用，如实时跟踪，聊天，维基，RSS订阅多。



* **特点**
  + 开源工具（GPL许可）
  + 支持运行PHP的任何平台（Windows，Linux和Mac上的Solaris，AS400/i5等）
  + 可自定义的发行页数
  + 用户可以有不同的访问级别，每个项目
  + 支持的项目，子项目和类别。
  + 支持全面的电子邮件通知
  + 搜索和过滤器 - 简单/高级筛选，全文检索，共享的过滤器（跨用户/项目）
  + 支持报告和图表报告
  + 每个实例的多个项目
  + 支持自定义字段
  + 允许自定义问题的工作流程
  + 允许看问题更改历史记录
  + 我的视图页面
  + 源代码控制集成
  + 无限数量的用户，问题或项目。
  + 设置匿名访问
  + 支持时间跟踪管理
  + 有本地化。
  + 变更支持
  + 简单的用户体验
  + 容易评估
  + 允许看路线图
  + 易于安装（内部和在托管环境中)

**参考链接**

* <https://bugs.kali.org/my_view_page.php?refresh=true>

## 随堂练习

* 你过去在项目开发过程中， 使用过哪些“缺陷管理系统”？
* 你能大致简述一下你们公司Bug管理的流程吗？
  + 从bug提交、Fixed解决、 挂起、打回、关闭。
  + 此问题中，Bug关闭由测试人员确认问题解决后，选择关闭。
* 你们公司中， 随机Bug如何如何关闭（解决）
  + 随机bug难点在于bug的复现。
  + 一般在最近连续三个release版本中，没有出现， 可视为bug解决，可选中关闭。

# 源码管理系统

* **目标**
  + 了解源码管理
  + 了解源码管理的工具
  + 了解源码管理流程
  + 了解git源码管理的分支

## 源码管理简述

在现代软件开发项目中，要成为一个有效的软件开发人员，我们必须能够与其他项目贡献者并行进行开发。

源代码管理（SCM）系统不是什么新思想。为了编写一些能够更快速、简单地开发以后软件项目的软件，已经进行了很多尝试。

最新的源代码解决方案都包含了版本控制系统，它可以对源代码的修改进行回滚，从而将有害的代码剔除出项目之外，或者简单地跟踪哪些人修改了代码的哪些行的内容。版本控制系统试图解决开发人员在试图同时对某个文件进行修改时所出现的冲突问题，可以防止用户覆盖其他人所作的修改。源代码管理使用的很多流行解决方案都试图解决以前 SCM 解决方案中的失效问题。

集中化的版本控制系统通常采用两种方式：

* 有些提供了文件锁来防止多个用户的并行访问。这些系统对文件进行加锁，这样在某个时间只有一个开发人员对中心仓库具有写入权限。
* 另外一些工具，例如 CVS，允许多个开发人员同时对相同的文件进行编辑，并提供了一些机制稍后合并这些修改。

流行的版本控制系统包括：

* CVS
* Subversion
* Arch
* Bazaar
* BitKeeper

## Git

### 什么是 Git？

非常简单地说，Git 是 Linus Torvalds 最近实现的源代码管理软件。正如所提供的文档中说的一样，“Git 是一个快速、可扩展的分布式版本控制系统，它具有极为丰富的命令集，对内部系统提供了高级操作和完全访问。”

Torvalds 开始着手开发 Git 是为了作为一种过渡方案来替代 BitKeeper，后者之前一直是 Linux 内核开发人员在全球使用的主要源代码工具。开放源码社区中的有些人觉得 BitKeeper 的许可证并不适合开放源码社区的工作，因此 Torvalds 决定着手研究许可证更为灵活的版本控制系统。尽管最初 Git 的开发是为了辅助 Linux 内核开发的过程，但是我们已经发现在很多其他自由软件项目中也使用了 Git。例如，X.org 最近就迁移到 Git 上来了，很多 Freedesktop.org 的项目也迁移到了 Git 上。

Git 目前主要由寻找 CVS 或专有代码管理解决方案替代物的软件开发人员所使用。Git 与 CVS 有很多区别：

* 分支更快、更容易。
* 支持离线工作；本地提交可以稍后提交到服务器上。
* Git 提交都是原子的，且是整个项目范围的，而不像 CVS 中一样是对每个文件的。
* Git 中的每个工作树都包含一个具有完整项目历史的仓库。
* 没有哪一个 Git 仓库会天生比其他仓库更重要。

## Git源码管理的中的流程

* 源码复制和传播
  + 源代码向研发部门以外复制必须获得”总经理“的书面授权。并必需记录复制人、批准人、复制时间、复制目的、文件流向、文件版本或内容。
  + 源代码的借阅、复制必须进行详细的登记，必需记录借阅人、批准人、借阅时间、借阅目的、文件流向、文件版本或内容、归还时间。
  + 对于因合作需要，需要向外复制、传播、分发源代码的，不论是全部还是部分代码和资料，均必需和对方签订技术、源码的保密协定，明确对方应当承担的对源码保密的责任和义务。
* 源码的版本管理
  + 版本号：由“＜主版本号＞.＜次版本号＞.＜修订号＞”三段组成，中间是点号分开。
* 源码的提交管理
  + 源码修改完成后，提交到”非主分支“， 分支命名可关联bug编号（建议）
  + 提交源码评审， 一般评审人员具有一票否决。
  + 评审通过后，提交源码merge申请，
  + 专人（配置管理员）或源码的Maintainer负责合并

## 成功的Git分支模型

（本文参考自网络， ）

下图是一个典型源码分支模型：



### 主要分支

在核心上，开发模式受到现有模型的极大启发。中央回购拥有两个主要的分支，无限生命：

* 主分支
  + master
* 开发分支
  + develop
* 发布分支
  + release-\* 每个Git用户应该熟悉该master分支origin。与master分支并行，存在另一个分支develop。

我们认为origin/master是源代码HEAD总是反映生产就绪状态的主要分支 。

我们认为origin/develop是主要的分支，源代码 HEAD总是反映出下一个版本的最新交付发展变化的状态。有人会称之为“整合分支”。这是每个自动夜间建造的地方。

当develop分支中的源代码达到稳定点并准备释放时，所有这些更改都应该以master 某种方式合并，然后用发布号标记。将详细讨论如何做到这一点。

因此，每次将更改合并到一起时master，根据定义，这是一个新的生产版本。我们往往对此非常严格，所以理论上，我们可以使用Git钩子脚本在每次提交时自动构建和推出我们的软件到我们的生产服务器 master。



## 随堂练习

* **思考**
  + 在你们原来的项目源码的管理中，源码分支是如何组织的？

# pcap文件解析

* **目标**
  + 了解 使用tcpdump生成pcap文件的方式
  + 掌握 使用wireshark生成pcap文件的方式
  + 掌握 pcap文件的格式，
  + 应用编码会解析pcap文件

## 获取pcap文件

### pcap是什么

pcap是一种数据流格式，wireshark软件可以直接把网络数据流变成这种格式。

在Linux里，pcap可以说是一种通用的数据流格式，很多开源的项目都需要用到这种格式的文件。

### 如何获取 pcap文件

一般通过tcpdump或者 wireshark工具获取pcap文件。

* **使用tcpdump命令获取pcap文件**
* tcpdump基本用法 tcpdump [ -i interface ] [ -w file ] [ expression ]
  + -i ： 监听 interface. 如果 不指定 接口, tcpdump 在 系统 的 接口 清单 中, 寻找 号码最小, 已经 配置好的 接口 (loopback 除外)
  + -w 把 原始报文 存进 file, 不做 分析 和 显示. 它们 可以 以后 用 -r 选项 显示. 如果 file 是 -'', 就 写往 标准输出.
  + -v (稍微多一点) 繁琐的输出. 例如, 显示 IP 数据报 中的 生存周期 和 服务类型.
  + -vv 更繁琐的输出. 例如, 显示 NFS 应答报文 的 附加域.

Panda.Guo@2018-07-26 20:56:13 $ sudo tcpdump -i wlp2s0 -w ~/wl.pcap -vv

tcpdump: listening on wlp2s0, link-type EN10MB (Ethernet), capture size 262144 bytes

^C69 packets captured

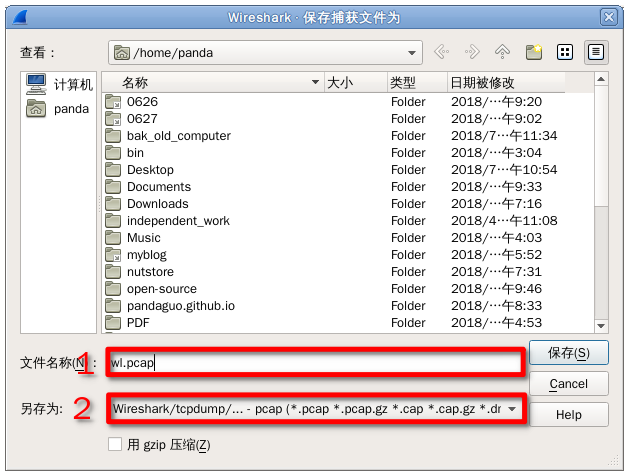
69 packets received by filter

0 packets dropped by kernel

Panda.Guo@2018-07-26 20:56:51 $

通过Ctrl+c随时选择终止。

* **使用wireshark工具抓取pcap文件**
  + 启动 wireshark，
  + 选择 “接口”后， 点击“开始捕获分组” 按钮，开始抓包
  + 。。。。
  + 选择 “停止捕获分组”
  + 选择“文件”-> "保存"， 弹出对话框,如下图所示



* + - 在1位置，输入pcap文件名称
    - 在2位置， 选则pcap类型。

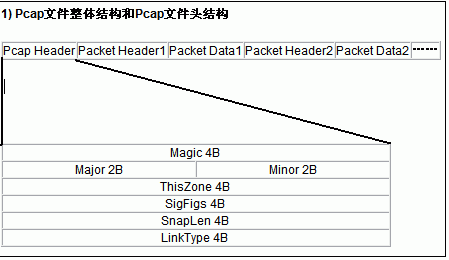
## pcap文件格式

### pcap格式

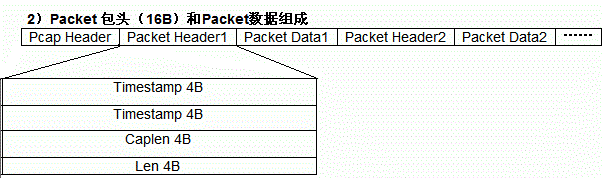


上图是pcap的格式结构图，分成两大部分：

* Global Header （共 24 Byte）整个数据流文件，只会有一个 Global Header，它定义了本文件的读取规则、最大储存长度限制等内容；



* + Magic：4Byte：标记文件开始，并用来识别文件自己和字节顺序。0xa1b2c3d4用来表示按照原来的顺序读取，0xd4c3b2a1表示下面的字节都要交换顺序读取。考虑到计算机内存的存储结构，一般会采用0xd4c3b2a1，即所有字节都需要交换顺序读取。
  + Major：2Byte： 当前文件主要的版本号，一般为 0x0200【实际上因为需要交换读取顺序，所以计算机看到的应该是 0x0002】
  + Minor：2Byte： 当前文件次要的版本号，一般为 0x0400【计算机看到的应该是 0x0004】
  + ThisZone：4Byte：当地的标准时间，如果用的是GMT则全零，一般都直接写 0000 0000
  + SigFigs：4Byte：时间戳的精度，设置为 全零 即可
  + SnapLen：4Byte：最大的存储长度，如果想把整个包抓下来，设置为 ffff 0000，但一般来说 ff7f 0000就足够了【计算机看到的应该是 0000 ff7f 】
  + LinkType：4Byte：链路类型，常用类型有以下几种，其他的，需要用的时候再查就行了。
    - 0 BSD loopback devices, except for later OpenBSD
    - 1 Ethernet, and Linux loopback devices
    - 6 802.5 Token Ring
    - 7 ARCnet
    - 8 SLIP
    - 9 PPP
    - 10 FDDI
    - 100 LLC/SNAP-encapsulated ATM
    - 101 "raw IP", with no link
    - 102 BSD/OS SLIP
    - 103 BSD/OS PPP
    - 104 Cisco HDLC
    - 105 802.11
    - 108 later OpenBSD loopback devices (with the AF\_value in network byte order)
    - 113 special Linux "cooked" capture
    - 114 LocalTalk
* Packet Header（共 16 Byte） Packet Header可以有多个，每个Packet Header后面会跟着一串Packet Data，Packet Header定义了Packet Data的长度、时间戳等信息。



* + Timestamp：被捕获时间的高位，单位是seconds
  + Timestamp：被捕获时间的低位，单位是microseconds
  + Caplen: 数据包长度：32位 ，标识所抓获的数据包保存在pcap文件中的实际长度，以字节为单位。
    - a 32-bit value giving the number of bytes of packet data that were captured;
  + Len：数据包实际长度： 所抓获的数据包的真实长度，如果文件中保存不是完整的数据包，那么这个值可能要比前面的数据包长度的值大。
    - a 32-bit value giving the actual length of the packet, in bytes (which may be greater than the previous number, if you are not saving the entire packet).
* Packet Data（共 Caplen Byte）在包头之后，就是数据包的数据了，数据长度就是Caplen个Byte，
  + 在这之后就是一个新的Packet Header，定义一个新的Packet Data属性，再接一个新的Packet Data，如此循环。
  + PCAP文件里面并没有规定捕获的Packet数据包之间有什么间隔字符串，下一组数据在文件中的起始位置。我们需要靠第一个Packet包确定



### 实例剖析

Panda.Guo@2018-06-26 20:01:00 $ hd ~/wl.pcap | less

00000000 d4 c3 b2 a1 02 00 04 00 00 00 00 00 00 00 00 00 |................|

00000010 00 00 04 00 01 00 00 00 08 d9 28 5a 85 ba 0b 00 |..........(Z....|

00000020 3c 00 00 00 3c 00 00 00 50 9a 4c 20 65 dc dc fe |<...<...P.L e...|

00000030 18 2a d5 c9 08 00 45 00 00 28 13 de 40 00 75 06 |.\*....E..(..@.u.|

00000040 7b f6 ca 59 e9 64 c0 a8 01 95 01 bb 15 bd 47 26 |{..Y.d........G&|

00000050 64 57 1b f9 dd fc 50 10 03 fe 78 ef 00 00 00 00 |dW....P...x.....|

00000060 01 95 01 bb 08 d9 28 5a 85 ba 0b 00 4b 01 00 00 |......(Z....K...|

00000070 4b 01 00 00 50 9a 4c 20 65 dc dc fe 18 2a d5 c9 |K...P.L e....\*..|

00000080 08 00 45 00 01 3d 13 df 40 00 75 06 7a e0 ca 59 |..E..=..@.u.z..Y|

00000090 e9 64 c0 a8 01 95 01 bb 15 bd 47 26 64 57 1b f9 |.d........G&dW..|

000000a0 dd fc 50 18 03 fe 98 00 00 00 17 03 02 01 10 3c |..P............<|

000000b0 c4 c4 aa 39 d0 30 29 66 3e e5 2d 2d 39 60 64 ed |...9.0)f>.--9`d.|

.....

## pcap 文件解析关键代码

示例代码如下：

函数名称：pcap\_t \*pcap\_open\_offline(char \*fname, char \*ebuf)

函数功能：打开以前保存捕获数据包的文件，用于读取。

参数说明：fname参数指定打开的文件名。该文件中的数据格式与tcpdump和tcpslice兼容。”-“为标准输入。ebuf参数则仅在pcap\_open\_offline()函数出错返回NULL时用于传递错误消息。

函数名称：int pcap\_loop(pcap\_t \*p, int cnt,pcap\_handler callback, u\_char \*user)

函数功能： 捕获并处理数据包。此函数在cnt个数据包被处理或出现错误时才返回，但读取超时不会返回。。cnt参数为负值时pcap\_loop()函数将始终循环运行，除非出现错误。

函数名称：void pcap\_close(pcap\_t \*p)

函数功能：关闭p参数相应的文件，并释放资源。

对于 pcap\_loop:

pcap\_loop原型是pcap\_loop(pcap\_t \*p,int cnt,pcap\_handler callback,u\_char \*user)

其中第一个参数是pcap\_open\_offline返回的句柄,

第二个是指定捕获的数据包个数,如果为-1则无限循环捕获。

第四个参数user是留给用户使用的。

第三个是回调函数其原型如下:

pcap\_callback(u\_char\* argument,const struct pcap\_pkthdr\* packet\_header,const u\_char\* packet\_content)

其中参数pcap\_content表示的捕获到的数据包的内容

参数argument是从函数pcap\_loop()传递过来的。注意：这里的参数就是指 pcap\_loop中的 \*user 参数

参数pcap\_pkthdr 表示捕获到的数据包基本信息,包括时间,长度等信息.

另外:回调函数必须是全局函数或静态函数,其参数默认,比如pcap\_loop()可以写成

pcap\_loop(pcap\_handle,10,pcap\_callback,NULL)不能往里面传递实参.

pcap\_loop和callback之间参数存在联系：

pcap\_loop的最后一个参数user是留给用户使用的，当callback被调用的时候这个值会传递给callback的第一个参数(也叫user)，

callback的最后一个参数p指向一块内存空间，这个空间中存放的就是pcap\_loop抓到的数据包。

callback的第二个参数是一个结构体指针，该结构体定义如下：

// /usr/include/pcap/pcap.h

struct pcap\_pkthdr {

struct timeval ts; /\* 时间戳 \*/

bpf\_u\_int32 caplen; /\* 已捕获部分的长度 \*/

bpf\_u\_int32 len; /\* 该包的脱机长度 \*/

};

这个结构体是由pcap\_loop自己填充的，用来取得一些关于数据包的信息

所以，在callback函数当中只有第一个user指针是可以留给用户使用的，如果你想给callback传递自己参数，那就只能通过pcap\_loop的最后一个参数user来实现了

## 随堂练习

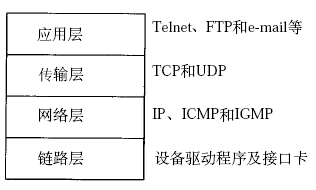
* 描述pcap文件的生成方式
* 简述pcap文件的组织结构

# TCP/IP协议栈及以太网帧

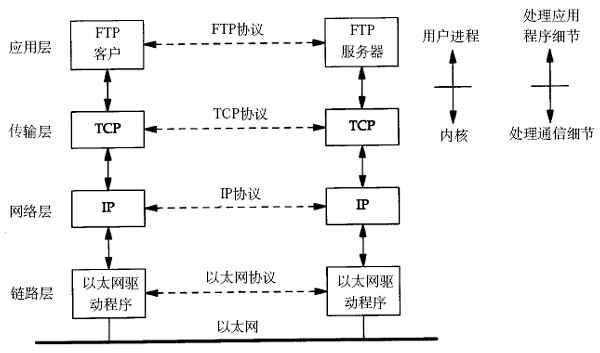
* **目标**
  + 了解TCP/IP协议栈的组织结构
  + 掌握以太网帧的数据格式定义
  + 能应用编码实现以太网帧的解析方法

## TCP/IP 协议栈

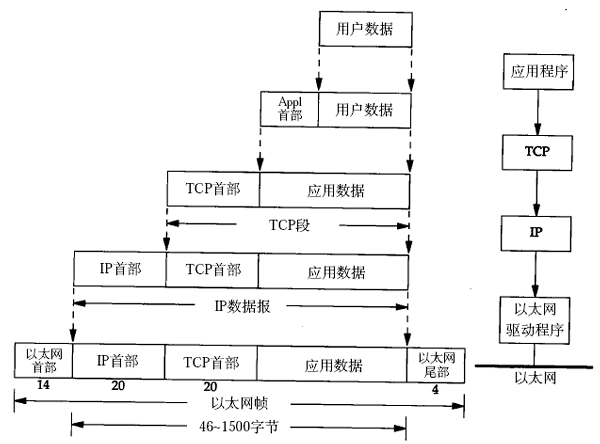
TCP/IP网络协议栈分为应用层（Application）、传输层（Transport）、网络层（Network）和链路层（Link）四层。如下图所示（该图出自[TCPIP]）。



两台计算机通过TCP/IP协议通讯的过程如下所示（该图出自[TCPIP]）。



传输层及其以下的机制由内核提供，应用层由用户进程提供（后面将介绍如何使用socket API编写应用程序），应用程序对通讯数据的含义进行解释，而传输层及其以下处理通讯的细节，将数据从一台计算机通过一定的路径发送到另一台计算机。应用层数据通过协议栈发到网络上时，每层协议都要加上一个数据首部（header），称为封装（Encapsulation），如下图所示（该图出自[TCPIP]）。

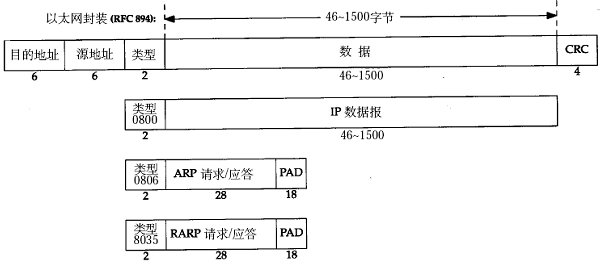


不同的协议层对数据包有不同的称谓，在**传输层叫做段（segment）**，在网络层叫做**数据报（datagram）**，在链路层叫做**帧（frame）**。数据封装成帧后发到传输介质上，到达目的主机后每层协议再剥掉相应的首部，最后将应用层数据交给应用程序处理。

**我们在pcap文件中，所得到的package data数据就是一个以太网帧**， 因此对“package data”的解析就是解析一个**以太网帧**

## 以太网(RFC 894)帧格式

以太网的帧格式如下所示（该图出自[TCPIP]）：



其中的源地址和目的地址是指网卡的硬件地址（也叫MAC地址），长度是48位，是在网卡出厂时固化的。用ifconfig命令看一下，“HWaddr 00:15:F2:14:9E:3F”部分就是硬件地址。协议字段有三种值，分别对应IP、ARP、RARP。帧末尾是CRC校验码。

以太网帧中的数据长度规定最小46字节，最大1500字节，ARP和RARP数据包的长度不够46字节，要在后面补填充位。最大值1500称为以太网的最大传输单元（MTU），不同的网络类型有不同的MTU，如果一个数据包从以太网路由到拨号链路上，数据包长度大于拨号链路的MTU了，则需要对数据包进行分片（fragmentation）。ifconfig命令的输出中也有“MTU:1500”。注意，MTU这个概念指数据帧中有效载荷的最大长度，不包括帧首部的长度。

## 以太网栈头部

//Mac头部，总长度14字节

typedef struct \_eth\_hdr

{

unsigned char dstmac[6]; //目标mac地址

unsigned char srcmac[6]; //源mac地址

unsigned short eth\_type; //以太网类型

}eth\_hdr;

## 以太网帧解析代码

一下为以太网帧的核心解析代码

MAC\_FRM\_HDR \*mac\_hdr; //define a Ethernet frame header

IP\_HDR \*ip\_hdr; //define a IP header

char \*tmp1, \*tmp2;

int AND\_LOGIC = 0xFF;

mac\_hdr = buf; //buf is what we got from the socket program

ip\_hdr = buf + sizeof(MAC\_FRM\_HDR);

//udp\_hdr = buf + sizeof(MAC\_FRM\_HDR) + sizeof(IP\_HDR); //if we want to analyses the UDP/TCP

tmp1 = mac\_hdr->src\_addr;

tmp2 = mac\_hdr->dest\_addr;

/\* print the MAC addresses of source and receiving host \*/

printf("MAC: %.2X:%.2X:%.2X:%.2X:%.2X:%.2X==>" "%.2X:%.2X:%.2X:%.2X:%.2X:%.2X",

tmp1[0]&AND\_LOGIC, tmp1[1]&AND\_LOGIC, tmp1[2]&AND\_LOGIC,tmp1[3]&AND\_LOGIC,

tmp1[4]&AND\_LOGIC, tmp1[5]&AND\_LOGIC,

tmp2[0]&AND\_LOGIC, tmp2[1]&AND\_LOGIC, tmp2[2]&AND\_LOGIC,tmp2[3]&AND\_LOGIC,

tmp2[4]&AND\_LOGIC, tmp2[5]&AND\_LOGIC);

tmp1 = (char\*)&ip\_hdr->ip\_src;

tmp2 = (char\*)&ip\_hdr->ip\_dest;

/\* print the IP addresses of source and receiving host \*/

printf("IP: %d.%d.%d.%d => %d.%d.%d.%d",

tmp1[0]&AND\_LOGIC, tmp1[1]&AND\_LOGIC, tmp1[2]&AND\_LOGIC,tmp1[3]&AND\_LOGIC,

tmp2[0]&AND\_LOGIC, tmp2[1]&AND\_LOGIC, tmp2[2]&AND\_LOGIC,tmp2[3]&AND\_LOGIC);

/\* print the IP protocol which was used by the socket communication \*/

switch(ip\_hdr->ip\_protocol) {

case IPPROTO\_ICMP: LOGI("ICMP"); break;

case IPPROTO\_IGMP: LOGI("IGMP"); break;

case IPPROTO\_IPIP: LOGI("IPIP"); break;

case IPPROTO\_TCP:

case IPPROTO\_UDP:

LOGI("Protocol: %s", ip\_hdr->ip\_protocol == IPPROTO\_TCP ? "TCP" : "UDP");

LOGI("Source port: %u, destination port: %u", udp\_hdr->s\_port, udp\_hdr->d\_port);

break;

case IPPROTO\_RAW: LOGI("RAW"); break;

default: printf("Unknown, please query in inclued/linux/in.h\n"); break;

}

## 随堂练习

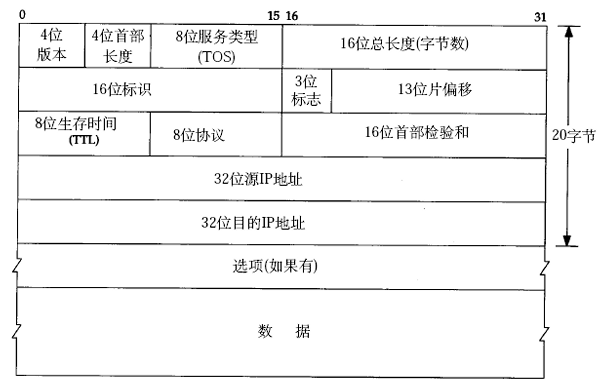
* **思考**
  + 如果源主机和目的主机不在同一网段，ARP请求的广播帧无法穿过路由器，源主机如何与目的主机通信？
* 简述tcp/ip协议栈的四层结构

# IP 报文

* **目标**
  + 了解IP报文的组织结构
  + 掌握IP报文的分析方法。

## IP数据报格式

IP数据报的格式如下（这里只讨论IPv4）（该图出自[TCPIP]）：



* 版本：IP报文版本号 IPV4:4，IPV6:6
* 首部长度：IP header 长度，单位是4字节, 没有选项，则一般为5（5x32bit＝20B）
* 8位服务类型：一般没有使用，详细参考RFC
* 总长度：header＋数据 总长度
* 16位标识：IP 报文的唯一id，分片报文的id 相同，便于进行重组。
* 3位标志：标明是否分片。是一个3位的控制字段，包含：
  + 保留位：1位
  + 不分段位：1位，取值：0（允许数据报分段）、1（数据报不能分段）
  + 更多段位：1位，取值：0（数据包后面没有包，该包为最后的包）、1（数据包后面有更多的包）

Bit 0: reserved, must be zero

Bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment.

Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments.

0 1 2

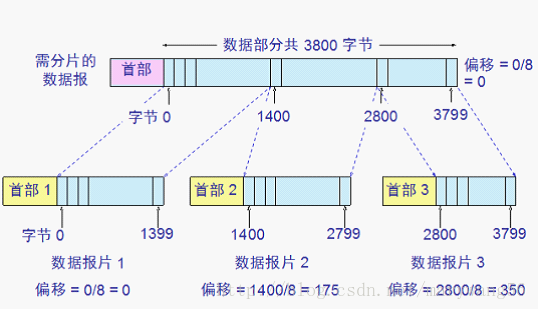
+---+---+---+

| | D | M |

| 0 | F | F |

+---+---+---+

* 13位片偏移：参考下图。如果是第一片取值为0，第二片取值175，以此类推。



* TTL：生存时间，即路由器的跳数，每经过一个路由器，该TTL 减一，因此路由器需要重新计算IP报文的校验和。
* 8位协议：ICMP：1，TCP：6，UDP：17，其他的自行百度
* 首部校验和：IP header校验和，接收端收到报文进行计算如果校验和错误，直接丢弃。
* 源IP地址：无须解释
* 目的IP地址：无须解释
* 选项：这个一般也没有使用。详细参考RFC
* 数据：上层的报文，如TCP 报文、UDP报文等。

## IP报文头部

IP头部，总长度20字节

// /usr/include/linux/ip.h

struct iphdr {

#if defined(\_\_LITTLE\_ENDIAN\_BITFIELD)

\_\_u8 ihl:4, //首部长度

version:4; //版本

#elif defined (\_\_BIG\_ENDIAN\_BITFIELD)

\_\_u8 version:4, //版本

ihl:4; //首部长度

#else

#error "Please fix <asm/byteorder.h>"

#endif

\_\_u8 tos; //服务类型

\_\_be16 tot\_len; //总长度

\_\_be16 id; //标志

\_\_be16 frag\_off; //分片偏移

\_\_u8 ttl; //生存时间

\_\_u8 protocol; //协议

\_\_sum16 check; //检验和

\_\_be32 saddr; //源IP地址

\_\_be32 daddr; //目的IP地址

/\*The options start here. \*/

};

## 随堂练习

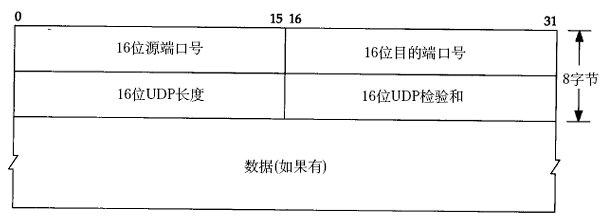
* **思考**
  + 前面讲了以太网帧中的最小数据长度为46字节，不足46字节的要用填充字节补上，那么如何界定这46字节里前多少个字节是IP、ARP或RARP数据报而后面是填充字节？

# TCP/UDP 段

* **目标**
  + 了解 TCP 段头的组织结构
  + 了解 UDP 段头的组织结构
  + 掌握 TCP/UDP 段的解析方式

## UDP 段格式

下图是UDP的段格式（该图出自[TCPIP]）。



## UDP头部

//UDP头部，总长度8字节

typedef struct \_udp\_hdr

{

unsigned short src\_port; //远端口号

unsigned short dst\_port; //目的端口号

unsigned short uhl; //udp报文总长度

unsigned short chk\_sum; //16位udp检验和

}udp\_hdr;

//ICMP头部，总长度4字节

typedef struct \_icmp\_hdr

{

unsigned char icmp\_type; //类型

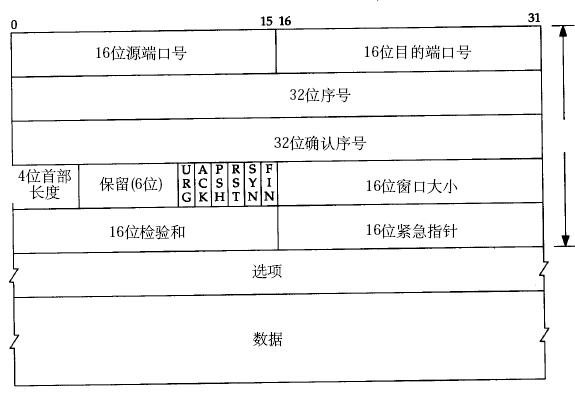
unsigned char code; //代码

unsigned short chk\_sum; //16位检验和

}icmp\_hdr;

## TCP段格式

TCP的段格式如下图所示（该图出自[TCPIP]）



和UDP协议一样也有源端口号和目的端口号，通讯的双方由IP地址和端口号标识。32位序号、32位确认序号、窗口大小稍后详细解释。4位首部长度和IP协议头类似，表示TCP协议头的长度，以4字节为单位，因此TCP协议头最长可以是4x15=60字节，如果没有选项字段，TCP协议头最短20字节。URG、ACK、PSH、RST、SYN、FIN是六个控制位，本节稍后将解释SYN、ACK、FIN、RST四个位，其它位的解释从略。16位检验和将TCP协议头和数据都计算在内。紧急指针和各种选项的解释从略。

## TCP头部

//TCP头部，总长度20字节

typedef struct \_tcp\_hdr

{

unsigned short src\_port; //源端口号

unsigned short dst\_port; //目的端口号

unsigned int seq\_no; //序列号

unsigned int ack\_no; //确认号

#if LITTLE\_ENDIAN

unsigned char reserved\_1:4; //保留6位中的4位首部长度

unsigned char thl:4; //tcp头部长度

unsigned char flag:6; //6位标志

unsigned char reseverd\_2:2; //保留6位中的2位

#else

unsigned char thl:4; //tcp头部长度

unsigned char reserved\_1:4; //保留6位中的4位首部长度

unsigned char reseverd\_2:2; //保留6位中的2位

unsigned char flag:6; //6位标志

#endif

unsigned short wnd\_size; //16位窗口大小

unsigned short chk\_sum; //16位TCP检验和

unsigned short urgt\_p; //16为紧急指针

}tcp\_hdr;

## 随堂练习

* 对与TCP 段的六个标志，你了解几个， 简述其意义
* 对于UDP段， 最大的数据包长度多少？
* UDP协议是“无连接“。。。
  + 如何保证数据包的次序，你有何思考。

# SSH协议解析

* **目标**
  + 了解ssh协议
  + 了解ssh协议探测方法（简洁）

## ssh协议

全称为Secure Shell，即很安全的shell，主要目的是用来取代传统的telnet和r系列命令（rlogin，rsh，rexec等）远程登录和远程执行命令的工具，实现远程登录和远程执行命令加密，防止由于网络监听而出现的密码泄露，从而对系统构成威胁。（telnet协议采用明文传送密码，数据传送过程中也不加密）

　　ssh协议目前有ssh1 和ssh2，其实现在我们主要使用的也是openssh。ssh不仅在登录过程中对密码进行加密传送，而且在登录后执行的命令的数据也进行加密，这样即使别人在网络上监听并截获了你的数据包，他也看不到其中的内容。

在网络安全防护中， “信息系统具备防窃听”， 这条要求的本质就是要用密文传输的ssh替代明文传输的telnet....

## ssh完整流程实例分析

通过ssh远程控制的一个完整个过程来讲，ssh的过程可分为以下3部分：

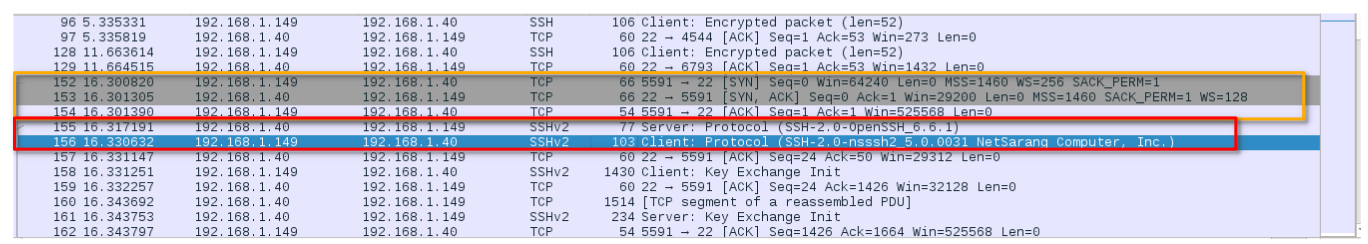
* **版本协商**
* **算法协商与密钥交换**

这其中第二部分是ssh最为核心的过程，该过程决定了以后通信所要使用的密钥，下面按顺序对每个部分对比着数据包进行详细的讲解并给出实现的过程。

### 版本协商

在建立连接后，客户端与服务器分别向对方发送自己ssh的版本信息（这里的数据格式不同于其他包，只有一行版本号），以\r\n结束。版本的格式如下：

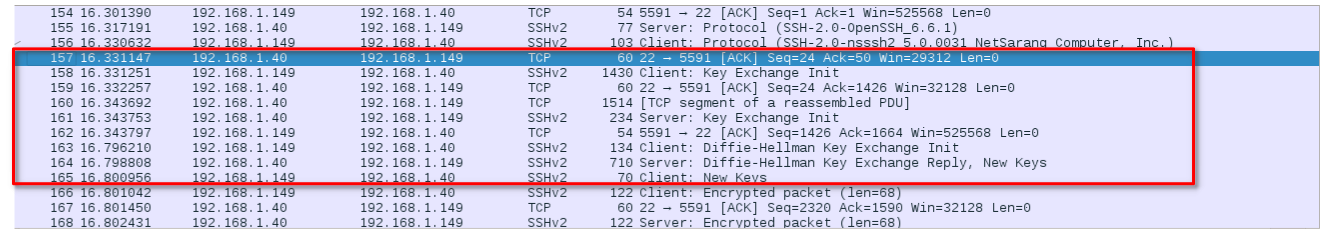
SSH-ssh协议版本-详细版本\r\n



在图中， 橙色部分，为tcp建立链接的三次握手

### 算法协商与密钥交换

先看图

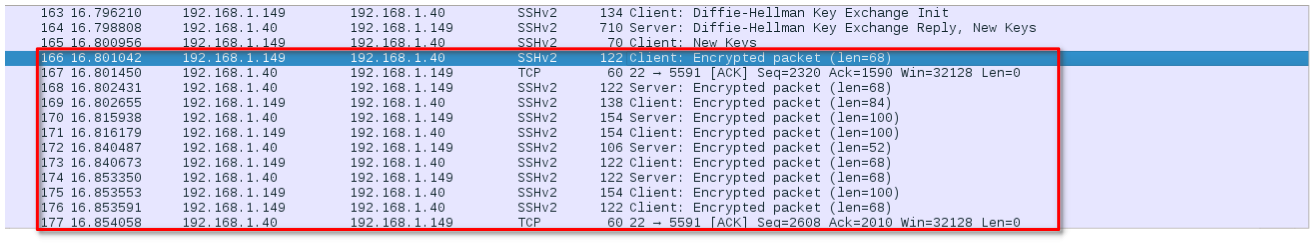


算法协商：第158，Key Exchange init开始，分别为双发向对法发送的自己在不同密码需求上支持的算法。

* 加密通信（可能含有2、3部分）

### 加密通信

上图， 从 Client: Encrypted packet 开始， 开始进行加密的通信



## 拓展内容

### 关于ssh相关的几个概念

在介绍ssh协议之前，有几个涉及到的基本概念首先需要介绍，它们对于理解ssh协议本身有非常重要和关键的作用。

加密

加密的意思是将一段数据经过处理之后，输出为一段外人无法或者很难破译的数据，除了指定的人可以解密之外。 一般来说，加密的输入还会有一个key，这个key作为加密的参数， 而在解密的时候也会用一个相关联(有可能是相同)的key作为输入。

简单的说： 密文 = 明文 + key

对称加密

所谓的对称加密，是说加密方和解密方用的都是同一个key，这个key对于加密方和解密方来说是保密的，第三方是不能知道的。在第三方不知道私钥的情况下，是很难将加密的数据解密的。一般来说是加密方先产生私钥，然后通过一个安全的途径来告知解密方这个key。

非对称加密

非对称加密，是说解密的一方首先生成一对密钥，一个私钥一个公钥，私钥不会泄漏出去，而公钥则是可以任意的对外发布的。用公钥进行加密的数据，只能用私钥才能解密。加密方首先从解密方获取公钥，然后利用这个公钥进行加密，把数据发送给解密方。解密方利用私钥进行解密。如果解密的数据在传输的过程中被第三方截获，也不用担心，因为第三方没有私钥，没有办法进行解密。

非对称加密的问题还包括获取了公钥之后，加密方如何保证公钥来自于确定的一方，而不是某个冒充的机器。假设公钥不是来自我们信任的机器，那么就算我们用非对称加密也没有用，因为加密之后的数据是发送给了冒充的机器，该机器就可以利用它产生的私钥进行解密了。所以非对称加密里面比较重要的一步是身份认证。

需要说明一下，一般的对称加密都会比非对称加密快，所以大数据量的加密一般都会使用对称加密，而非对称加密会作为身份验证和交换对称加密秘钥的一个手段。

数据一致性/完整性

数据一致性说得是如何保证一段数据在传输的过程中没有遗漏、破坏或者修改过。一般来说，目前流行的做法是对数据进行hash，得到的hash值和数据一起传输，然后在收到数据的时候也对数据进行hash，将得到的hash值和传输过来的hash值进行比对，如果是不一样的，说明数据已经被修改过；如果是一样的，则说明极有可能是完整的。

目前流行的hash算法有MD5和SHA-1算法。

### 参考

* [RFC4251](https://tools.ietf.org/html/rfc4251)

# TFTP 应用协议报文解析

* **目标**
  + 了解tftp协议
  + 了解tftp基本操作方式（put/get）
  + 了解tftp协议探测方法（简洁）

## tftp 基本操作

通过操作tftp client和server操作， 熟悉抓包流程

### Server 端

* 安装服务

itcast@itcast-teacher:~$ sudo apt-get install tftpd-hpa

* 配置服务

初始配置

itcast@itcast-teacher:~$ cat /etc/default/tftpd-hpa

# /etc/default/tftpd-hpa

TFTP\_USERNAME="tftp"

TFTP\_DIRECTORY="/var/lib/tftpboot"

TFTP\_ADDRESS=":69"

TFTP\_OPTIONS="--secure"

修改配置为

itcast@itcast-teacher:~$ sudo vi /etc/default/tftpd-hpa

itcast@itcast-teacher:~$ cat /etc/default/tftpd-hpa

# /etc/default/tftpd-hpa

TFTP\_USERNAME="tftp"

TFTP\_DIRECTORY="/tmp/"

TFTP\_ADDRESS=":69"

TFTP\_OPTIONS="-l -s -c"

* 配置含义为：
  + TFTP\_DIRECTORY： 为TFTP\_Server服务目录，该目录最好具有可读可写权限, 一遍上传下载服务。
  + “TFTP\_ADDRESS”：为0.0.0.0:69，表示所有IP源都可以访问，　此处可不改
  + “TFTP\_OPTIONS”为“-l -c -s”。其中
    - -l :以standalone/listen模式启动TFTP服务
    - -c: 可创建新文件。默认情况下TFTP只允许覆盖原有文件而不能创建新文件
    - -s : 改变TFTP启动的根目录，加了-s后，客户端使用TFTP时，不再需要输入指定目录，填写文件的文件路径，而是使用配置文件中写好的目录
* 重启服务：

itcast@itcast-teacher:~$ sudo /etc/init.d/tftpd-hpa restart

[ ok ] Restarting tftpd-hpa (via systemctl): tftpd-hpa.service.

itcast@itcast-teacher:~$

### Client 端

客户端操作比较简单，我们仅展示 上传（put）文件和下载（get）文件

* 文件下载

$ tftp 192.168.0.111

tftp> get a.out

Received 11363 bytes in 0.3 seconds

tftp> q

q　退出

* 文件上传　ｐｕｔ

$ tftp 192.168.0.111

tftp> put aa.pcap

Sent 4569 bytes in 0.2 seconds

tftp> q

$

注，　确保你的目录下有相应文件

## TFTP 协议介绍

TFTP是一个传输文件的简单协议，它其于UDP协议而实现，（但是也不能确定有些TFTP协议是基于其它传输协议完成的）此协议设计的时候是进行小文件传输的。

只能从文件服务器上获得或写入文件，不能列出目录，不进行认证

* 传输中有三种模式：
  + netascii，这是8位的ASCII码形式，
  + octet，这是8位源数据类型；
  + 最后一种mail已经不再支持，它将返回的数据直接返回给用户而不是保存为文件
* 初始连接

初始连接时候需要发出WRQ（请求写入远程系统）或RRQ（请求读取远程系统），收到一个确定应答，一个确定可以写出的包或应该读取的第一块数据。

* TFTP包

TFTP支持五种类型的包：

|  |  |
| --- | --- |
| **opcode** | **operation** |
| 01 | Read request (RRQ) |
| 02 | Write request (WRQ) |
| 03 | Data (DATA) |
| 04 | Acknowledgment (ACK) |
| 05 | Error (ERROR) |

* RRQ/WRQ 数据包格式

2 bytes string 1 byte string 1 byte

------------------------------------------------

| Opcode | Filename | 0 | Mode | 0 |

------------------------------------------------

RRQ 和WRQ包（代码分别为1和2）的格式如上所示。

文件名是NETASCII码字符，以0结束。

而MODE域包括了字符串"netascii"，"octet"或"mail"，名称不分大小写。

* DATA 包

2 bytes 2 bytes n bytes

----------------------------------

| Opcode | Block # | Data |

----------------------------------

数据在数据包中传输，其格式如上图所示。

数据包的OP码为3，它还包括有一个数据块号和数据。

数据块号域从1开始编码，每个数据块加1，这样接收方可以确定这个包是新数据还是已经接收过的数据。

数据域从0字节到512字节。如果数据域是512字节则它不是最后一个包，如果小于512字节则表示这个包是最后一个包。

除了ACK和用于中断的包外，其它的包均得到确认。发出新的数据包等于确认上次的包。

WRQ和DATA包由ACK或ERROR数据包确认，

而 RRQ数据包由DATA或ERROR数据包确认。

* ACK包

2 bytes 2 bytes

---------------------

| Opcode | Block # |

---------------------

上图即是一个ACK包，操作码为4。其中的包号为要确认的数据包的包号。

WRQ数据包被ACK数据包确认，WRQ数据包的包号为0。

* ERROR 包

2 bytes 2 bytes string 1 byte

-----------------------------------------

| Opcode | ErrorCode | ErrMsg | 0 |

-----------------------------------------

一个ERROR包，它的操作码是5，它的格式如上所示。此包可以被其它任何类型的包确认。

错误码(ErrorCode)

Value Meaning

0 未定义，请参阅错误信息（如果提示这种信息的话）

1 文件未找到

2 访问非法

3 磁盘满或超过分配的配额

4 非法的TFTP操作

5 未知的传输ID

6 文件已经存在

7 没有类似的用户

## 随堂练习

* **思考**

# NTP应用协议报文解析

* **目标**
  + 了解NTP协议
  + 了解NTP包基本捕获方式
  + 了解NTP协议探测（解析）方法（简单方法）

## 使用ntpdate同步网络时间

* 安装

$ sudo apt-get install ntpdate

* 对时服务

查看时间

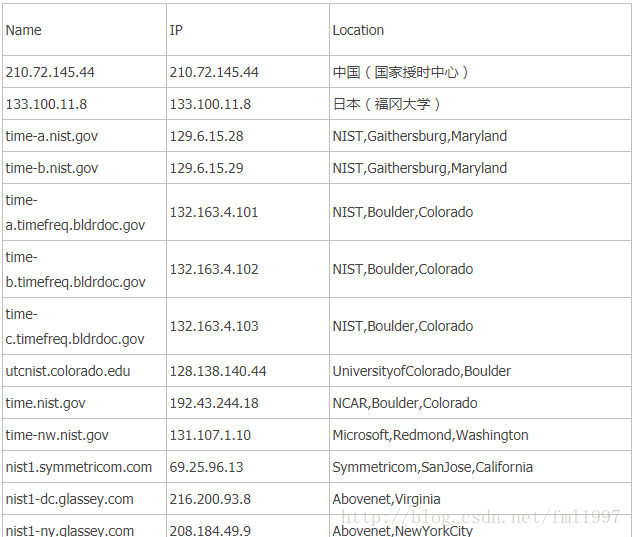
$ date #date可以查看当前系统时间

对时

$ sudo ntpdate -u 133.100.11.8 #同步到日本的服务器时间

* **常用的对时服务器**

ntp.api.bz ， 中国上海时间



## NTP 协议介绍

NTP（Network Time Protocol，网络时间协议）是由RFC 1305定义的时间同步协议，用来在分布式时间服务器和客户端之间进行时间同步。NTP基于UDP报文进行传输，使用的UDP端口号为123。

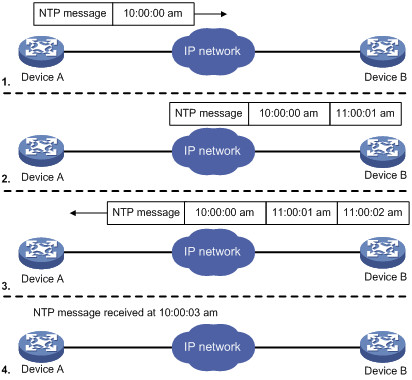
### NTP工作原理

NTP的基本工作原理如图所示。Device A和Device B通过网络相连，它们都有自己独立的系统时钟，需要通过NTP实现各自系统时钟的自动同步。为便于理解，作如下假设：

在Device A和Device B的系统时钟同步之前，Device A的时钟设定为10:00:00am，Device B的时钟设定为11:00:00am。

Device B作为NTP时间服务器，即Device A将使自己的时钟与Device B的时钟同步。

NTP报文在Device A和Device B之间单向传输所需要的时间为1秒。



### 系统时钟同步的工作过程

Device A发送一个NTP报文给Device B，该报文带有它离开Device A时的时间戳，该时间戳为10:00:00am（T1）。

当此NTP报文到达Device B时，Device B加上自己的时间戳，该时间戳为11:00:01am（T2）。

当此NTP报文离开Device B时，Device B再加上自己的时间戳，该时间戳为11:00:02am（T3）。

当Device A接收到该响应报文时，Device A的本地时间为10:00:03am（T4）。

至此，Device A已经拥有足够的信息来计算两个重要的参数：

NTP报文的往返时延Delay=（T4-T1）-（T3-T2）=2秒。

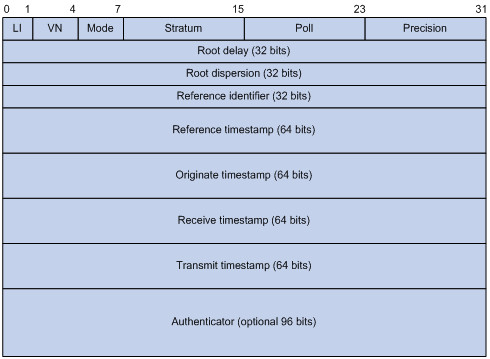
Device A相对Device B的时间差offset=（（T2-T1）+（T3-T4））/2=1小时。

这样，Device A就能够根据这些信息来设定自己的时钟，使之与Device B的时钟同步。

### NTP的报文格式

NTP有两种不同类型的报文，一种是时钟同步报文，另一种是控制报文。

我们主要关注时钟同步报文：



主要字段的解释如下：

* LI（Leap Indicator）：长度为2比特，值为“11”时表示告警状态，时钟未被同步。为其他值时NTP本身不做处理。
* VN（Version Number）：长度为3比特，表示NTP的版本号，目前的最新版本为3。
* Mode：长度为3比特，表示NTP的工作模式。不同的值所表示的含义分别是：0未定义、1表示主动对等体模式、2表示被动对等体模式、3表示客户模式、4表示服务器模式、5表示广播模式或组播模式、6表示此报文为NTP控制报文、7预留给内部使用。
* Stratum：系统时钟的层数，取值范围为1～16，它定义了时钟的准确度。层数为1的时钟准确度最高，准确度从1到16依次递减，层数为16的时钟处于未同步状态，不能作为参考时钟。
* Poll：轮询时间，即两个连续NTP报文之间的时间间隔。
* Precision：系统时钟的精度。
* Root Delay：本地到主参考时钟源的往返时间。
* Root Dispersion：系统时钟相对于主参考时钟的最大误差。
* Reference Identifier：参考时钟源的标识。
* Reference Timestamp：系统时钟最后一次被设定或更新的时间。
* Originate Timestamp：NTP请求报文离开发送端时发送端的本地时间。
* Receive Timestamp：NTP请求报文到达接收端时接收端的本地时间。
* Transmit Timestamp：应答报文离开应答者时应答者的本地时间。
* Authenticator：验证信息(可以没有)。

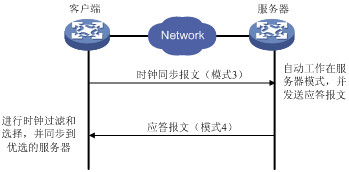
### NTP的工作模式

设备可以采用多种NTP工作模式进行时间同步：

* 客户端/服务器模式
* 对等体模式
* 广播模式
* 组播模式

客户端/服务器和对等体模式中，设备从指定的服务器或对等体获得时钟同步，增加了时钟的可靠性。

**客户端/服务器模式**



在客户端/服务器模式中，客户端向服务器发送时钟同步报文，报文中的Mode字段设置为3（客户模式）。服务器端收到报文后会自动工作在服务器模式，并发送应答报文，报文中的Mode字段设置为4（服务器模式）。客户端收到应答报文后，进行时钟过滤和选择，并同步到优选的服务器。

在该模式下，客户端能同步到服务器，而服务器无法同步到客户端(单向)。

其他模式 暂不介绍