黑客攻防

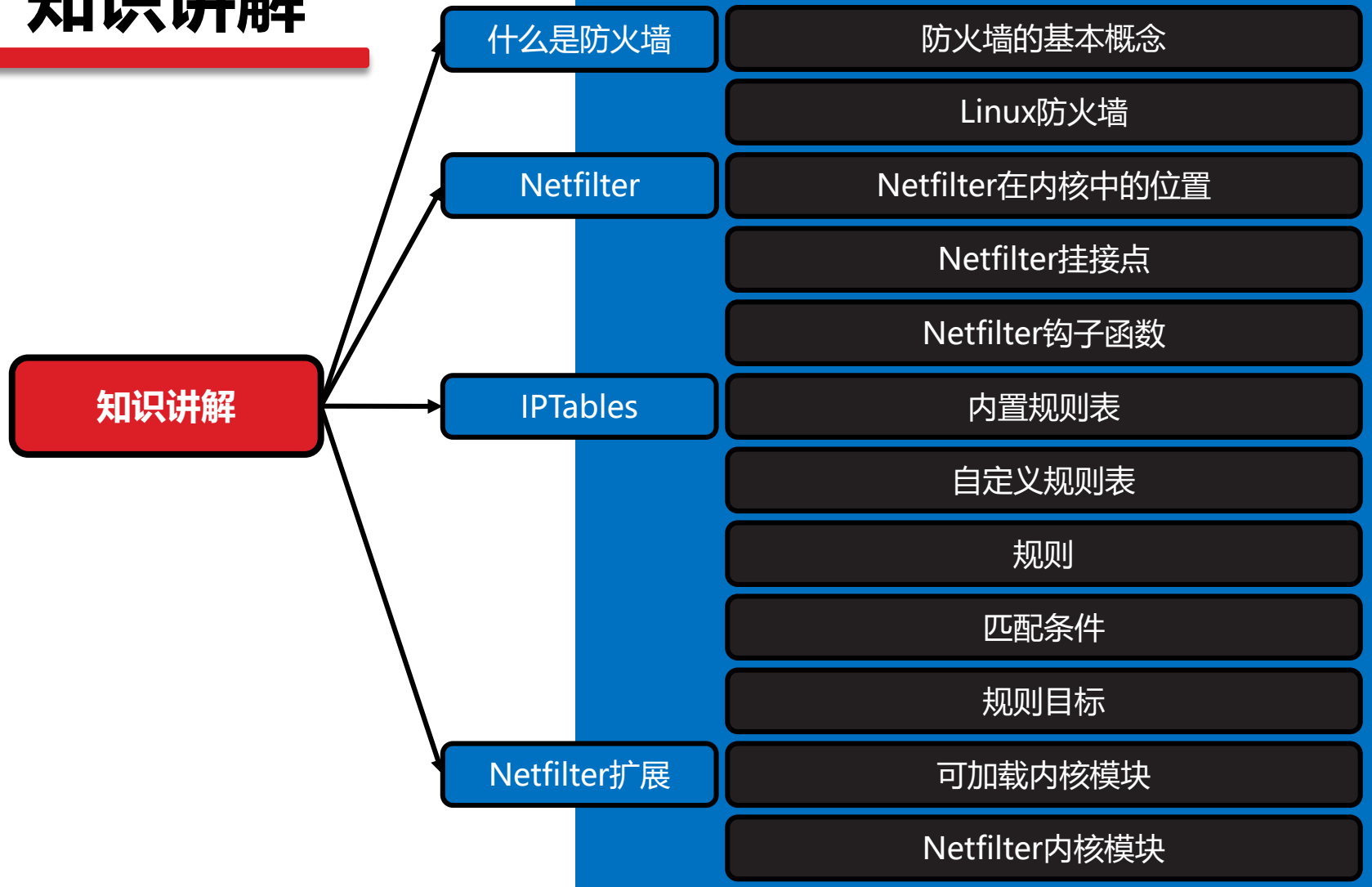
防火墙

Unit11

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	知识讲解
	10:30 ~ 11:20	
	11:30 ~ 12:00	
下午	14:00 ~ 14:50	实训案例
	15:00 ~ 15:50	
	16:00 ~ 16:50	扩展提高
	17:00 ~ 17:30	总结和答疑

知识讲解

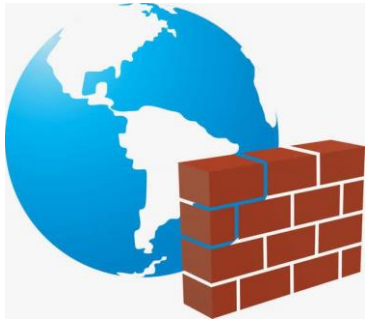

知识讲解
什么是防火墙
防火墙的基本概念
Linux防火墙
Netfilter
Netfilter在内核中的位置
Netfilter挂接点
Netfilter钩子函数
IPTables
内置规则表
自定义规则表
规则
匹配条件
规则目标
Netfilter扩展
可加载内核模块
Netfilter内核模块

什么是防火墙

防火墙的基本概念

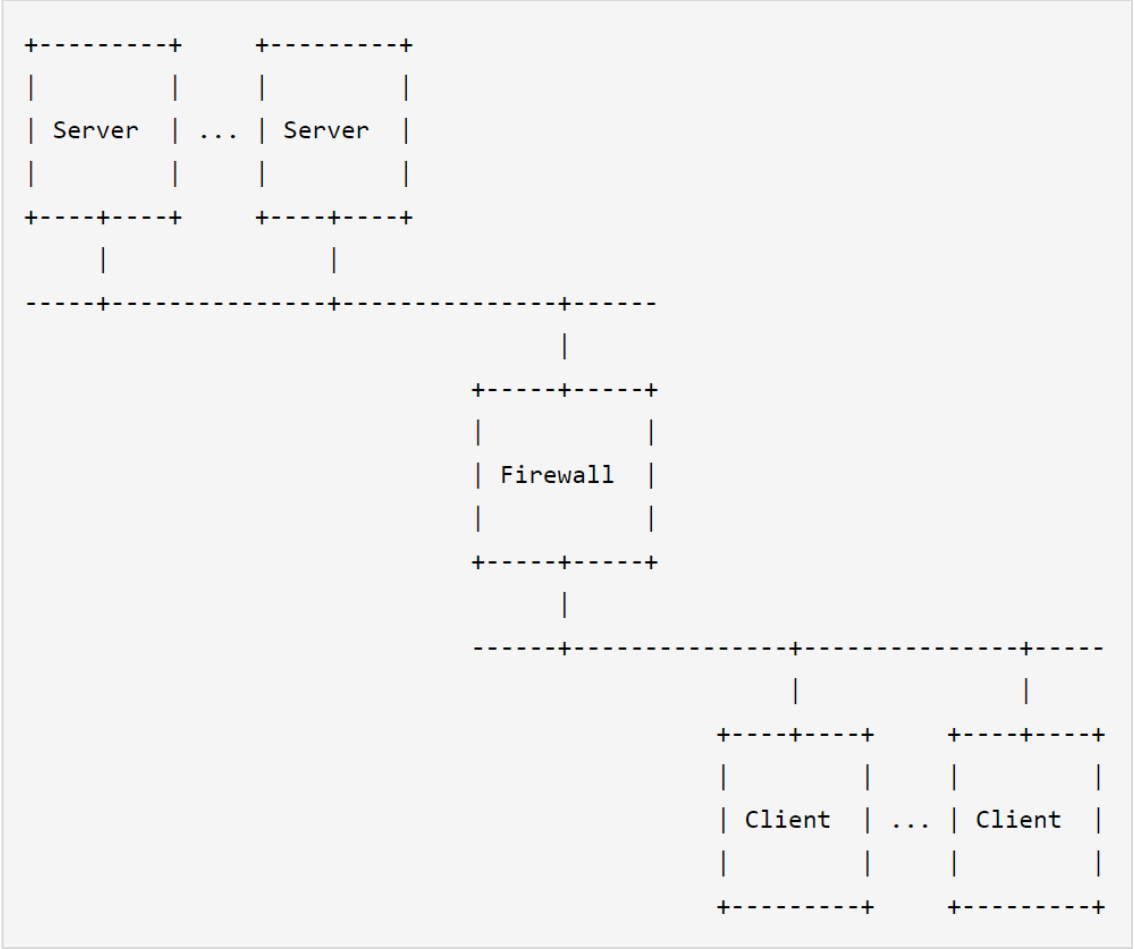
- 防火墙的作用
  - 防火墙提供了网络对网络或网络对主机的访问控制，通过地址隐藏等技术手段，保护网络资源免受非法侵害，是目前较为常见的网络安全设备之一
- 防火墙的实现
  - 防火墙的实现方式不一而足，但通常都是由一组硬件设备(路由器和计算机)和特定的软件组合而成。除一般的通用型防火墙以外，还有一些专用的防火墙，只为特定类型的网络服务，如HTTP、SMTP等，提供保护

知识讲解



防火墙的基本概念

- 防火墙的位置
  - 防火墙在网络中的位置如下图所示：



- 客户端访问服务器，但并不直接和服务器通信，而是先将服务请求发送给服务器侧的防火墙。经防火墙检测被认为是安全的数据包才会转发给相应的服务器，否则直接过滤掉

知识讲解

防火墙的基本概念

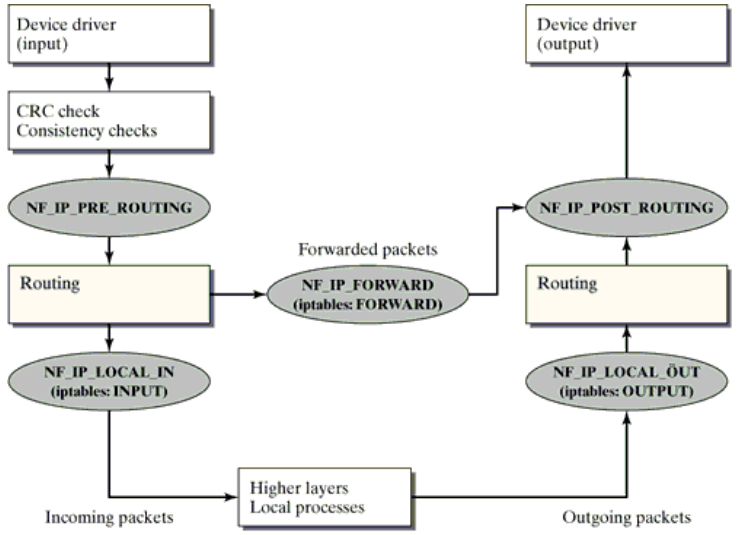
- 防火墙的类型
  - 包过滤型防火墙
    - 包过滤型防火墙工作于网络层(IP协议层)，也被称为网络层防火墙。这种防火墙在网络层对数据包进行监控和分析，按照事先设定好的过滤规则，检查流经防火墙的每个数据包的源地址、目的地址、源端口、目的端口、协议状态等字段，并据此决定哪些数据包获准通过，哪些又应被拦截。这种防火墙的核心是过滤规则的制定
  - 应用网关型防火墙
    - 应用网关型防火墙基于网络代理技术构建，也被称为代理型防火墙。这种防火墙工作于网络协议栈的应用层，接收来自外部的各种服务请求，经安全检查后，转发给内部相应的网络服务。同样，从内部网络发往外部网络的数据包也会受到监控，并被处理为如同从防火墙外部发出的一样，从而达到隐藏内部网络的目的。这种防火墙对网络性能会有一定影响，且必须为每种网络服务提供专门的代理模块，实现难度较大

知识讲解

Linux防火墙

- Linux防火墙从最初设计到现在相对成熟的体系经历了若干代的更迭：
  - 2.0内核：源于Free BSD的ipforward，在内核级实现对数据包的过滤
  - 2.2内核：ipchains取代了ipforward，缺乏面向用户空间的访问接口，可扩展性受到限制
  - 2.3内核：引入Netfilter结构，并通过IPTables组件使用户可对Netfilter进行设置，用户可依规则自行定制

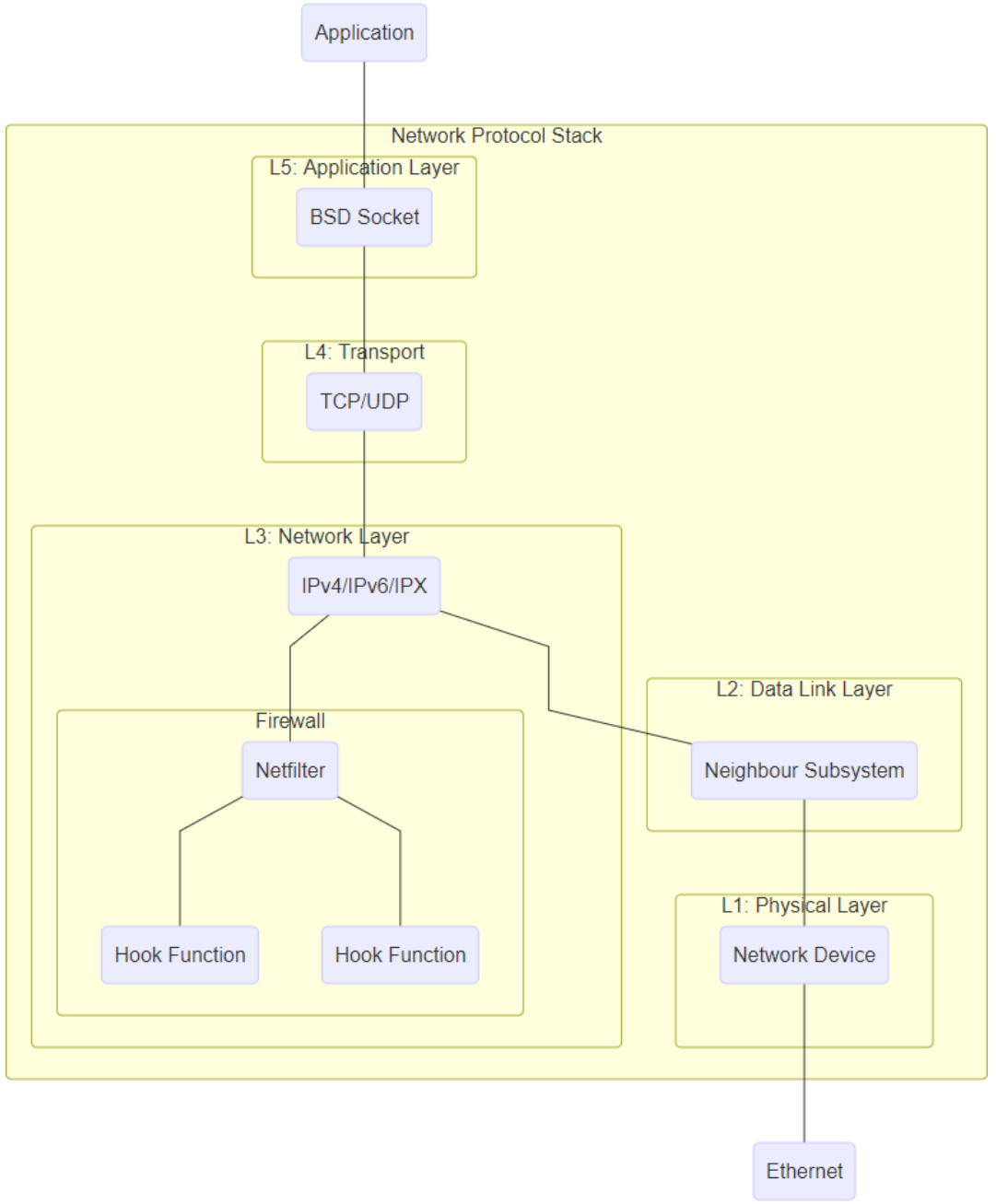
知识讲解



Netfilter

Netfilter在内核中的位置

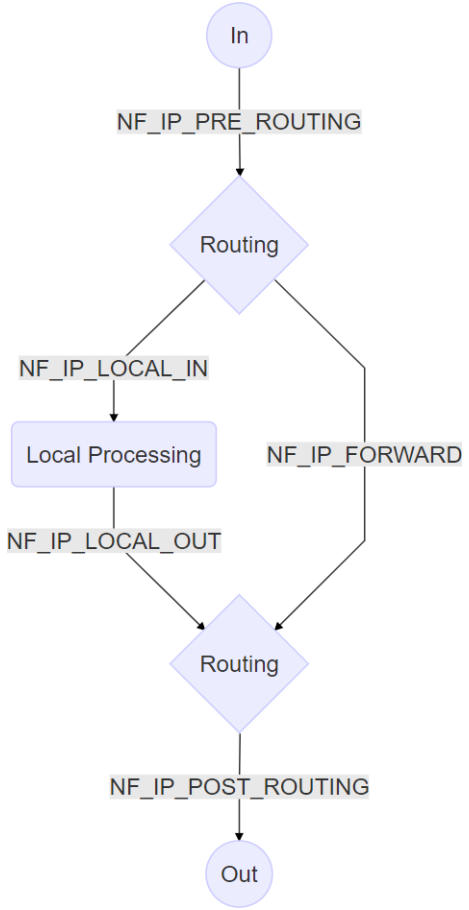
- Linux系统内核通过Netfilter在网络层(IP协议层)的内部实现防火墙功能。Netfilter为每种网络层协议，如IPv4、IPv6、IPX等，各定义了一套钩子函数。这些钩子函数在数据包流经协议栈的几个特定位置时被调用，并对数据包执行相应的处理，如修改、丢弃或交给用户进程等。如下图所示：



知识讲解

Netfilter挂接点

- Netfilter在网络层(IP协议层)的五个位置上设置挂接点，如下图所示：



- 每个挂接点都有其特定的位置和意义，如下表所示：

挂接点	位置
NF_IP_PRE_ROUTING	数据包刚刚收到
NF_IP_FORWARD	数据包正从一块网卡转向另一块网卡
NF_IP_LOCAL_IN	数据包即将进入上层协议栈
NF_IP_LOCAL_OUT	数据包刚刚离开上层协议栈
NF_IP_POST_ROUTING	数据包即将发出

知识讲解

Netfilter钩子函数

- Netfilter为每种网络层协议都定义了一套钩子函数。这些钩子函数的入口地址保存在一个二维数组中。每个希望嵌入Netfilter的模块都可以在协议栈的挂接点上注册钩子函数，这些钩子函数构成一条函数指针链
- 数据包在协议栈中流到某个挂接点时，Netfilter就会调用事先注册的钩子函数，对数据包进行捕获和分析，并根据钩子函数返回的结果，决定下一步如何处理：
  - 原封不动地放回协议栈
  - 做一些修改放回协议栈
  - 直接丢弃
- 钩子函数的返回值如下表所示：

返回值	含义
NF_ACCEPT	允许通过，将数据包原封不动地放回协议栈，继续下一步处理
NF_DROP	禁止通过，将数据包直接丢弃，不再继续处理
NF_STOLEN	由钩子函数自己处理数据包，不再继续传送
NF_QUEUE	将数据包加入队列，交由用户程序处理
NF_REPEAT	再次调用该钩子函数

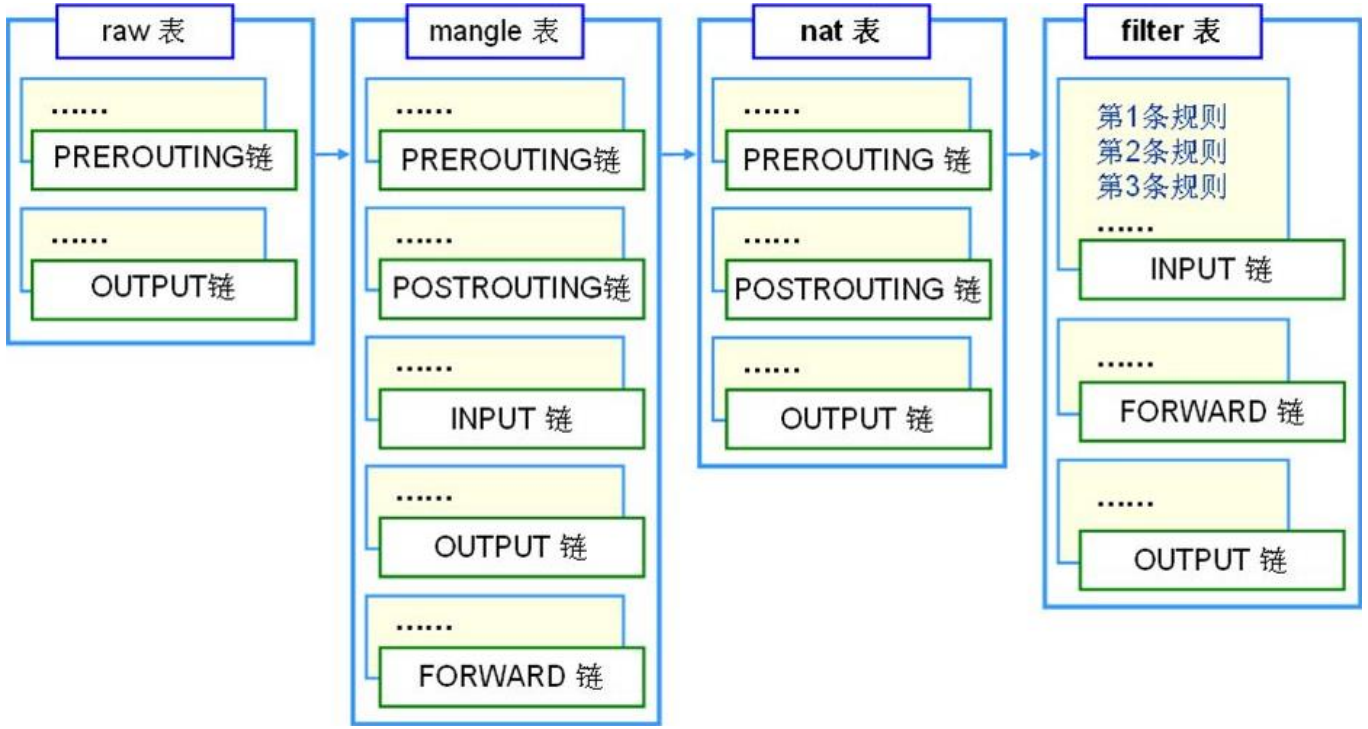
知识讲解

IPTables

IPTables

- 构成Netfilter体系结构除了钩子函数以外还包括IPTables，即规则表，钩子函数通过访问表中定义的规则来决定应该向Netfilter返回什么值。系统管理员可以通过iptables命令管理和维护这些规则表

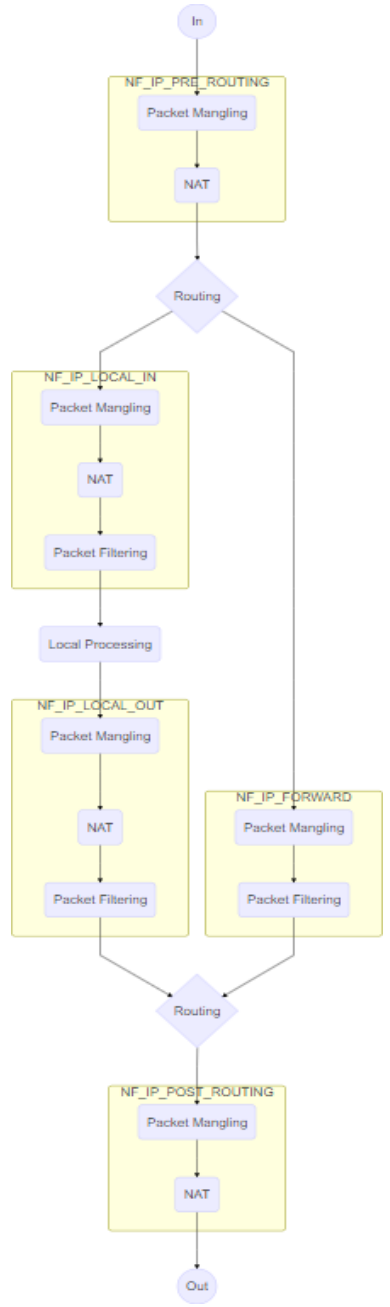
知识讲解



内置规则表

知识讲解

- Linux系统内核默认内置三张规则表：
  - 报文处理(Packet Mangling)规则表
    - 通过全部五个挂接点接入Netfilter框架，实现对报文的修改或附加带外数据
  - 网络地址转换(NAT)规则表
    - 通过NF_IP_PRE_ROUTING、NF_IP_POST_ROUTING、NF_IP_LOCAL_IN和NF_IP_LOCAL_OUT四个挂接点接入Netfilter框架，分别对转发和本机报文的源及目的地址进行转换
  - 报文过滤(Packet Filtering)规则表
    - 通过NF_IP_LOCAL_IN、NF_IP_LOCAL_OUT和NF_IP_FORWARD三个挂接点接入Netfilter框架，分别对接收、发送和转发报文进行过滤
- 数据包在网络层(IP协议层)内部的流动路径如下如所示：



自定义规则表

- 内核编程人员还可以注入模块的方式，通过Netfilter接口创建自定义的规则表。表结构如下所示：

```
struct ipt_table {
    struct list_head        list;                       // 表头
    char                    name[IPT_TABLE_MAXNAMELEN]; // 表名
    struct ipt_replace    * table;                      // 模板
    unsigned int            valid_hooks;                // 钩子标志位
    rwlock_t                lock;                       // 读写锁
    struct ipt_table_info * private;                    // 规则链集
    struct module         * me;                         // 模块
};
```

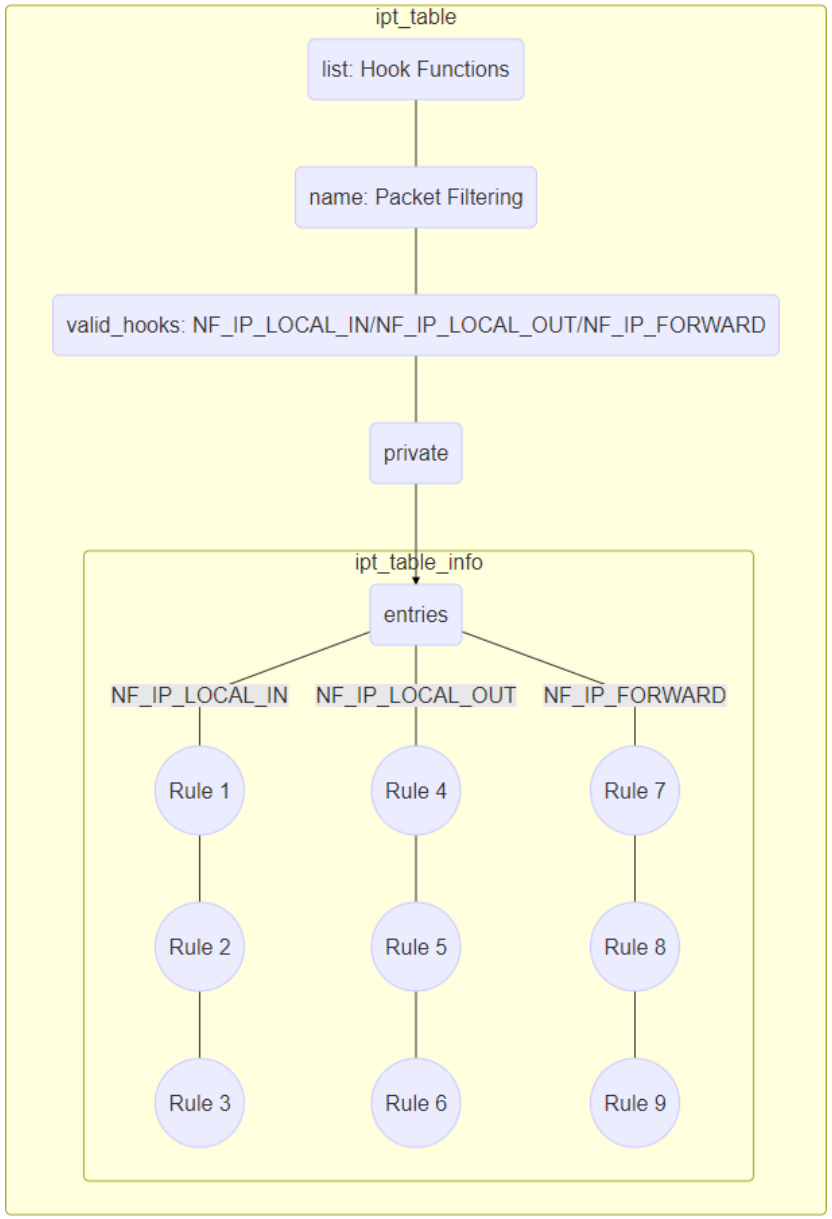
  - 其中用于描述规则链集的private字段采用ipt_table_info结构体类型：

```
struct ipt_table_info {
    unsigned int size;                          // 字节数
    unsigned int number;                        // 规则数
    unsigned int initial_entries;               // 初始规则数
    unsigned int hook_entry[NF_IP_NUMHOOKS];    // 规则链起始偏移
    unsigned int underflow[NF_IP_NUMHOOKS];     // 规则链终止偏移
    char         entries[0];                    // 规则链集起始位置
};
```

知识讲解

自定义规则表

- 钩子、规则表、规则链和规则链集，以报文过滤规则表为例，如下图所示：
  - 数据包每流到一个挂接点(NF_IP_LOCAL_IN、NF_IP_LOCAL_OUT和NF_IP_FORWARD)，即执行相应的钩子函数(Hook Functions)，依据特定的规则链(Rule-Rule-Rule)，依次检查每条规则(Rule)的匹配条件是否满足，若满足则执行相应的操作



知识讲解

规则

- IPTables中的每一张表(ipt_table)中均包含0到N条规则链，每条规则链由若干条规则组成，每条规则包括两个部分：
  - 0到M个匹配条件(Match)：只有所有匹配条件都满足，该条规则才生效
    - 0个匹配条件表示不做限制，所有数据包均视为满足匹配条件
  - 0或1个规则目标(Target)：如何处置满足匹配条件的数据包
    - 0个规则目标表示不做任何处置，直接进入规则链中下一条规则的匹配性检查
- 对于每条规则，系统内核还要维护两个计数器：
  - 报文计数器(Packet Counter)：满足该条规则匹配条件的总报文数
  - 字节计数器(Byte Counter)：满足该条规则匹配条件的总字节数

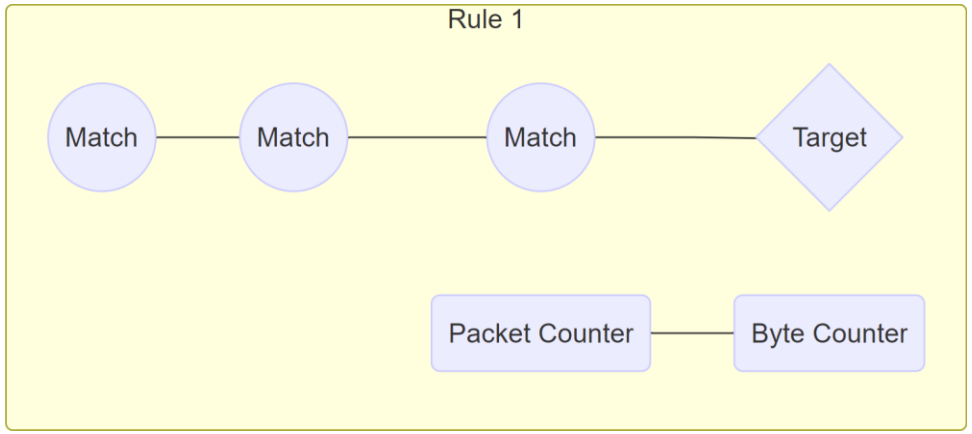
知识讲解

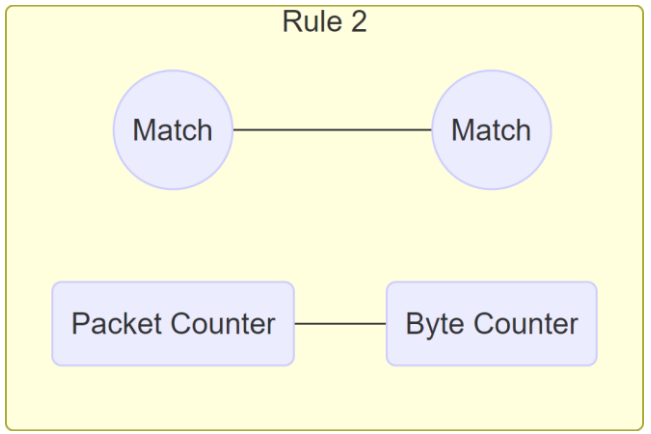
规则

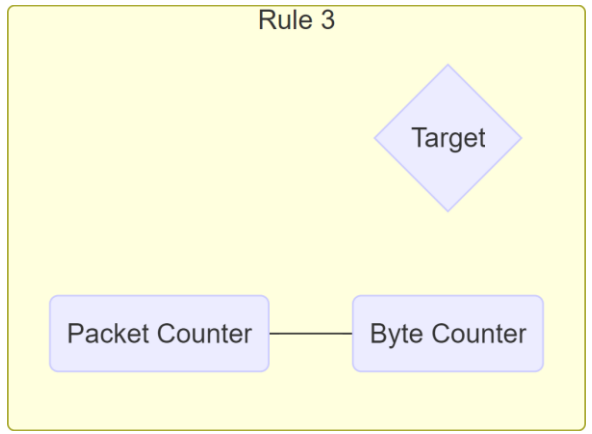
- 描述每一条规则的数据结构如下所示：

```
struct ipt_entry {
    struct ipt_ip       ip;             // 匹配IP头部
    unsigned int        nfcache;        // 字段标志位
    u_int16_t           target_offset; // 规则目标偏移
    u_int16_t           next_offset;    // 规则链中下一条规则的相对偏移
    unsigned int        comefrom;       // 钩子标志位
    struct ipt_counters counters;       // 报文计数器和字节计数器
    unsigned char       elems[0];       // 匹配条件和规则目标
};
```

- 规则链中的每条规则如下图所示：





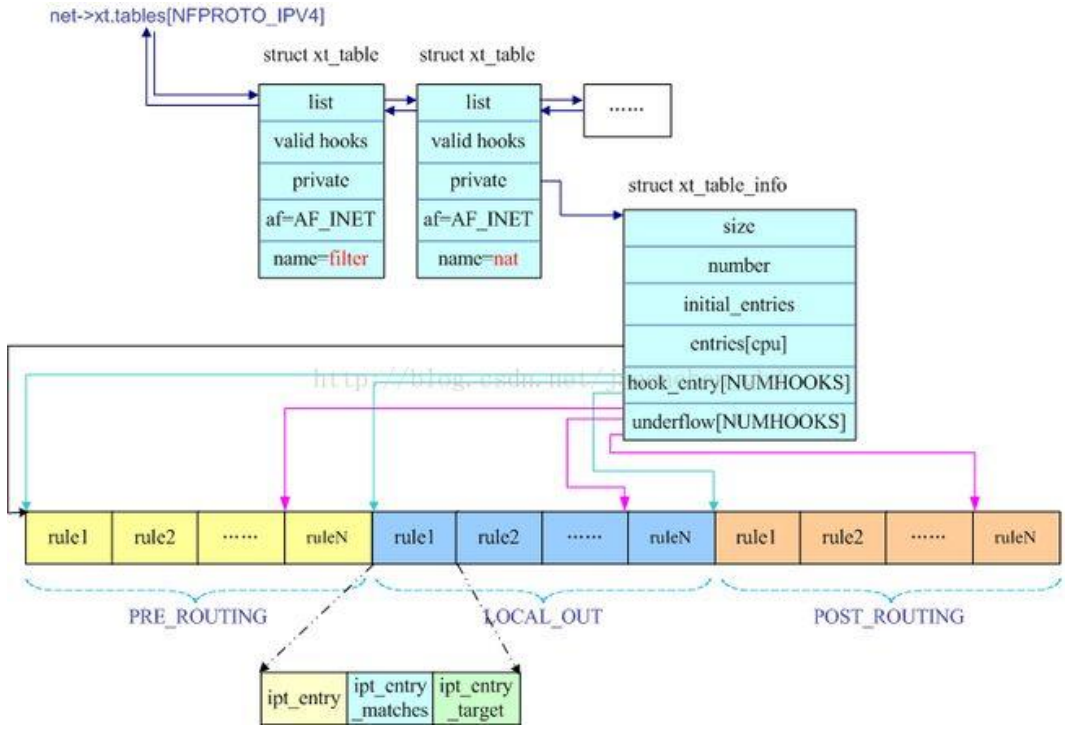


知识讲解

匹配条件

- iptables命令用于设置多种匹配条件，但某些条件需要依赖特定的内核模块
- 如果不载入任何扩充内核模块，iptables命令只能默认设置针对IP包头的匹配条件，如源IP地址、目的IP地址、传输层协议等
- 如果希望将除IP包头以外的其它协议(如TCP、UDP、ICMP等)包头中的字段作为匹配条件，则需要通过iptables命令的-m选项载入特定的内核模块

知识讲解



规则目标

- 规则目标决定了如何处置满足匹配条件的数据包。默认规则目标如下表所示：

规则目标	处置方式
ACCEPT	接受数据包，不再继续检查规则链中后续规则的匹配条件
DROP	丢弃数据包，不再继续检查规则链中后续规则的匹配条件
QUEUE	将数据包加入队列，交由用户程序处理
RETURN	返回主规则链继续检查规则匹配条件

- 如果希望加入默认目标以外的其它目标，需载入支持该目标的扩充内核模块

知识讲解

Netfilter内核扩展

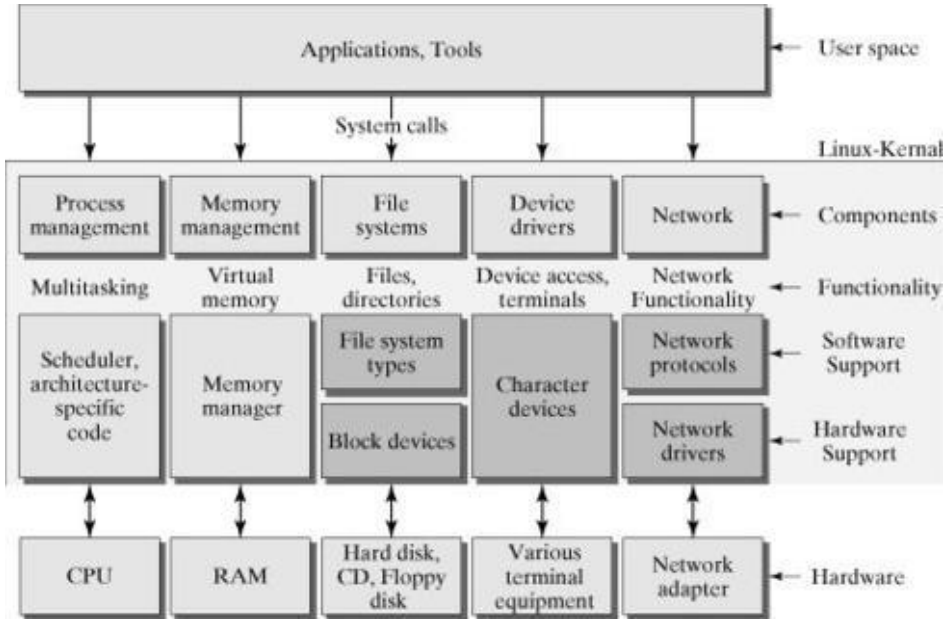
可加载内核模块

- 操作系统的内核分为微内核和大内核两种体系结构：
  - 微内核操作系统
    - 内核很小，只实现诸如内存管理、进程调度等最基本的功能，而象网络协议、文件系统等，均在内核的外部实现
    - 内核精炼，结构清晰，易于扩展，但模块间频繁的数据交换会降低系统的运行性能
    - 微内核的典型代表是基于NT架构的Windows操作系统
  - 大内核操作系统
    - 内存管理、进程调度、网络协议、文件系统等大部分底层功能都在内核中实现
    - 运行速度非常快，但功能过于集中，扩展性和维护性通常较差
    - 大内核的典型代表是Linux操作系统

知识讲解

可加载内核模块

- 为了弥补大内核系统扩展性差这一缺陷，Linux操作系统提供了可加载内核模块机制，使用户在无需重新编译整个系统内核，甚至无需重新启动操作系统的前提下，动态加载和卸载实现特定功能的内核模块，使大内核系统同样具备动态扩展的能力
- 可加载内核模块是具有独立功能的程序，可被独立编译，但不能独立运行。可加载内核模块必须被动态链接，即加载到系统内核中，作为系统内核的一部分，在内核空间运行。这是其与大多数用户空间程序最显著的区别



知识讲解

可加载内核模块

- 编写可加载内核模块必须提供以下两个函数：
  - int init_module(void);
  - void cleanup_module(void);
  - 这两个函数在加载和卸载模块的过程中被调用，分别用于初始化和善后处理。init_module函数返回0表示模块加载成功，否则加载失败
- 将可加载内核模块编译为.ko文件，执行insmod命令将其加载到内核中，如：
  - insmod filter.ko
- 执行lsmod命令列出所有可加载内核模块的名字，如：
  - lsmod
- 执行rmmod命令卸载指定的可加载内核模块，如：
  - rmmod filter.ko

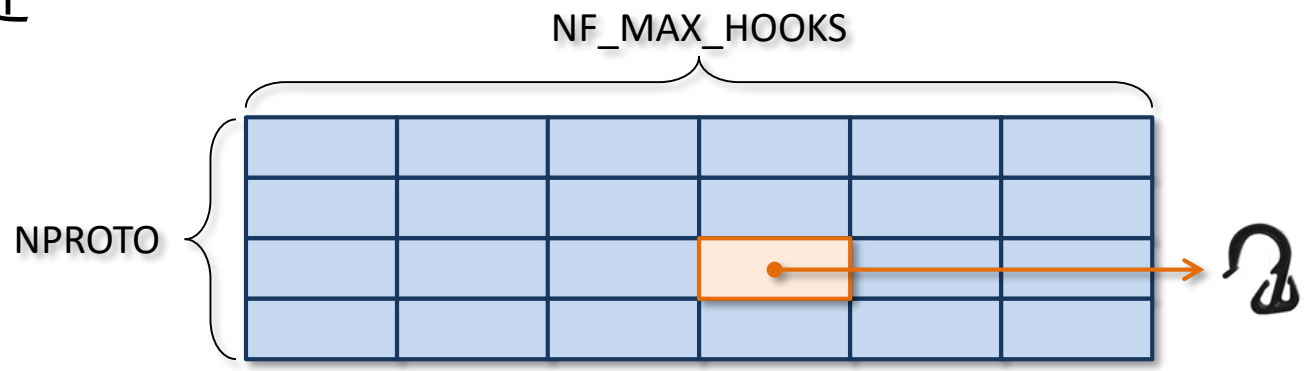
知识讲解

Netfilter内核模块

- Netfilter为每种网络协议定义了一套钩子函数，这些钩子函数在数据包流经网络协议栈的特定挂接点时被调用。Linux网络协议栈通过一个全局二维数组保存这些钩子函数的指针：

```
struct list_head nf_hooks[NPROTO][NF_MAX_HOOKS];
```

  - 其中，第一维表示协议族，第二维表示钩子类型，即挂接点
  - 注册钩子的过程，就是根据给定的协议族和钩子类型，将钩子函数指针添加到nf_hooks数组的适当位置处



知识讲解

Netfilter内核模块

知识讲解

- 钩子注册结构：

```
struct nf_hook_ops {
    nf_hookfn           * hook;     // 钩子函数指针
    struct net_device   * dev;
    void                * priv;
    u_int8_t              pf;       // 协议族
    unsigned int          hooknum;  // 钩子类型
    int                   priority; // 优先级
};
```

- 注册/注销钩子函数：

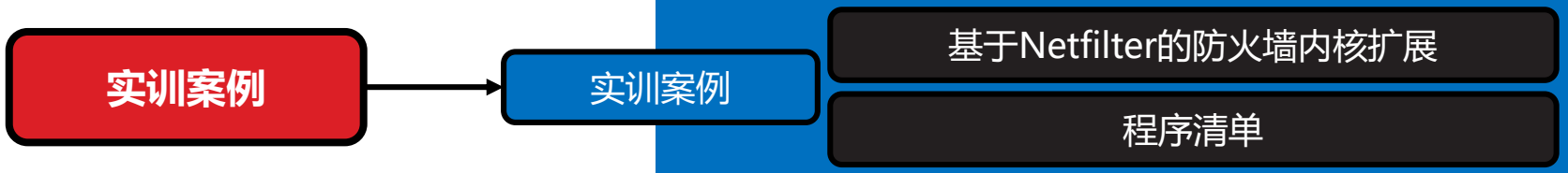
```
int nf_register_net_hook(struct net* net, const struct nf_hook_ops* ops);
void nf_unregister_net_hook(struct net* net, const struct nf_hook_ops* ops);
```

- 钩子函数原型：

```
typedef unsigned int nf_hookfn(void* priv, struct sk_buff* skb,
    const struct nf_hook_state* state);
```

- 从套接字缓存skb中可以获取各种网络协议的包头字段，以之参与规则匹配

实训案例



实训案例

基于Netfilter的防火墙内核扩展

- 应用有关Netfilter和内核模块的知识，通过三个程序实践对内核模块的扩展
  - 第一个程序实现基于协议的数据包过滤功能
  - 第二个程序实现基于源IP地址的数据包过滤功能
  - 第三个程序实现基于目的端口号的数据包过滤功能
- 三个程序的主体架构基本一样，区别在于钩子函数的实现不同，程序通过钩子函数对数据包实施不同的操作，从而实现不同的过滤策略

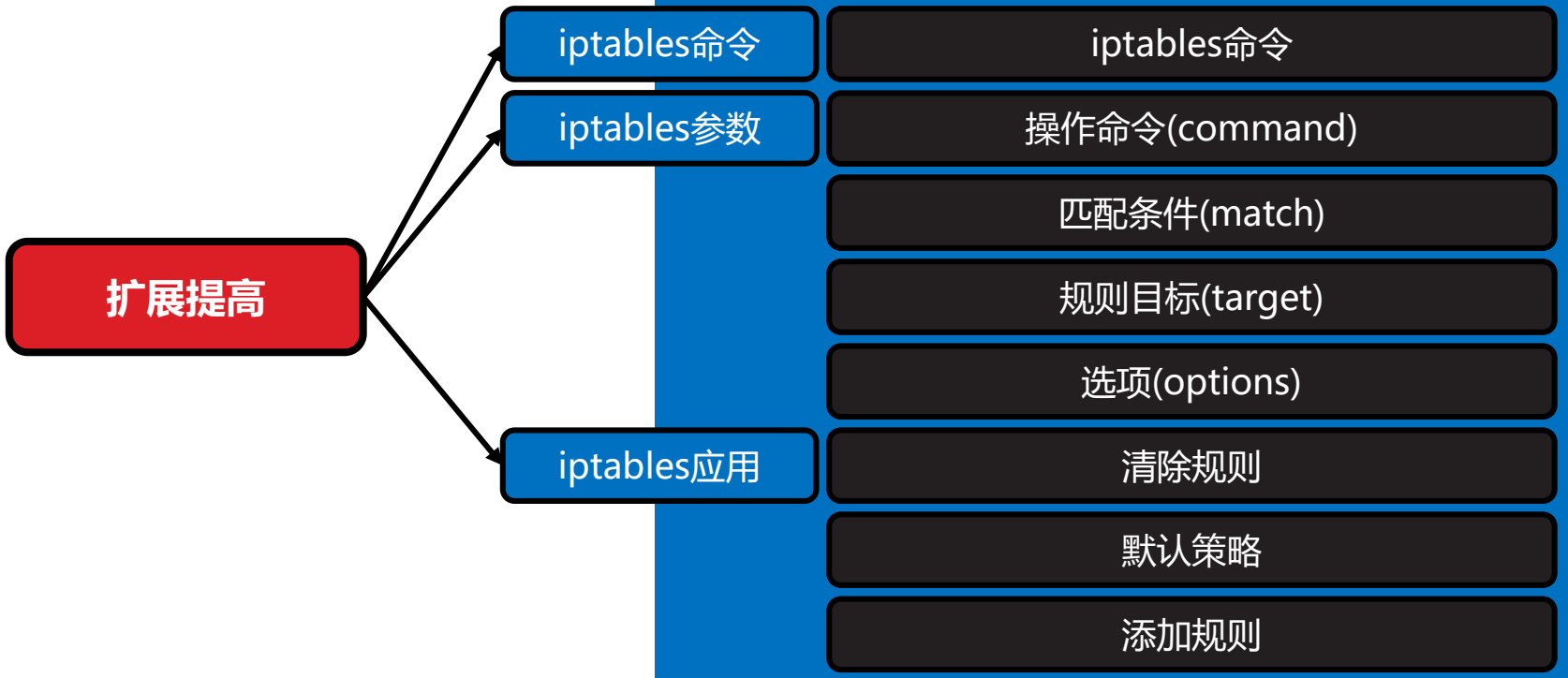
代码实践

程序清单

- 协议过滤模块
  - protfilter.c
- 地址过滤模块
  - addrfilter.c
- 端口过滤模块
  - portfilter.c
- 过滤模块构建脚本
  - Makefile

代码实践

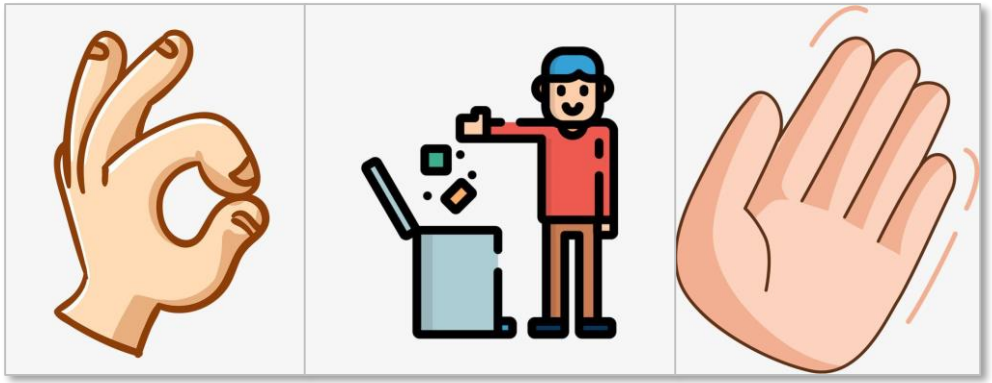
扩展提高


扩展提高
iptables命令
iptables参数
iptables应用
iptables命令
操作命令(command)
匹配条件(match)
规则目标(target)
选项(options)
清除规则
默认策略
添加规则

iptables命令

iptables命令

- iptables命令允许系统管理员在用户空间对Netfilter内核模块中的规则表进行插入、删除和修改。规则表中的每条规则都有自己的目标，它告诉系统内核应如何处理满足规则匹配条件的数据包：
  - ACCEPT：接受
  - DROP：丢弃
  - REJECT：拒绝



知识讲解

iptables参数

iptables参数

- iptables命令语法如下所示：

```
iptables [-t table] command [match] [-j target/jump] options
```

  - 其中：
    - table：指定规则表，系统内核默认内置三张规则表：
      - mangle
      - nat
      - filter，若未指定规则表，默认filter表
    - command：操作命令，如插入、删除等
    - match：匹配条件
    - target：规则目标，如ACCEPT、DROP等
    - options：选项

知识讲解

操作命令(command)

知识讲解

操作命令	功能	示例
-A/--append	在指定规则链的末尾追加一条规则	iptables -A INPUT --dport 80 -j ACCEPT
-I/--insert	在指定规则链的指定位置插入一条规则	iptables -I INPUT 1 --dport 80 -j ACCEPT
-D/--delete	从指定规则链中删除一条规则	iptables -D INPUT 1
		iptables -D INPUT --dport 80 -j ACCEPT
-F/--flush	清空指定或所有规则链	iptables -F INPUT
-R/--replace	替换指定规则链中指定位置的规则	iptables -R INPUT 1 --dport 80 -j DROP
-P/--policy	为指定规则链设置默认策略	iptables -P INPUT DROP
-L/--list	列表显示指定或所有规则链中的所有规则	iptables -L INPUT
-Z/--zero	将指定规则链的计数器清零	iptables -Z INPUT
-N/--new-chain	新建自定义规则链	iptables -N allowed
-E/--rename-chain	更名自定义规则链	iptables -E allowed disallowed
-X/--delete-chain	删除自定义规则链	iptables -X disallowed

匹配条件(match)

知识讲解

匹配条件	含义	示例
-p/--protocol	通信协议	iptables -A INPUT -p tcp -j DROP
-s/--src/--source	源IP地址	iptables -A INPUT -s 192.168.1.1 -j DROP
-d/--dst/--destination	目的IP地址	iptables -A INPUT -d 192.168.1.2 -j DROP
--sport/--source-port	源端口	iptables -A INPUT -p tcp --sport 22 -j DROP
--dport/--destination-port	目的端口	iptables -A INPUT -p tcp --dport 80 -j DROP
-i/--in-interface	输入网卡	iptables -A INPUT -i eth0 -j DROP
-o/--out-interface	输出网卡	iptables -A INPUT -o eth1 -j DROP
--tcp-flags	TCP标志	iptables -A INPUT -p tcp --tcp-flags SYN,FIN -j DROP
-syn	TCP连接	iptables -A INPUT -p tcp -syn -j DROP
-m multiport	多端口	iptables -A INPUT -p tcp -m multiport --sport 22,53 -j DROP
		iptables -A INPUT -p tcp -m multiport --dport 80,96 -j DROP
		iptables -A INPUT -p tcp -m multiport --port 23,80 -j DROP

匹配条件(match)

知识讲解

匹配条件	含义	示例
--icmp-type	ICMP类型	iptables -A INPUT -p icmp --icmp-type 8 -j DROP
-m limit --limit	流量上限	iptables -A INPUT -m limit --limit 3/second -j DROP
-m limit --limit -burst	浪涌上限	iptables -A INPUT -m limit --limit -burst 5 -j DROP
-m mac --mac-source	源MAC地址	iptables -A INPUT -m mac --mac-source a3:3d:ee:6a:4c:01 -j DROP
-m owner --uid-owner	产生用户	iptables -A INPUT -m owner --uid-owner 100 -j DROP
-m owner --gid-owner	产生组	iptables -A INPUT -m owner --gid-owner 200 -j DROP
-m owner --pid-owner	产生进程	iptables -A INPUT -m owner --pid-owner 300 -j DROP
-m owner --sid-owner	产生会话	iptables -A INPUT -m owner --sid-owner 400 -j DROP
-m state --state	会话状态	iptables -A INPUT -m state --state NEW,RELATED -j DROP

规则目标(target)

规则目标	处理动作
ACCEPT	接受，不继续匹配后续规则
DROP	丢弃，不继续匹配后续规则
REJECT	拒绝，回传通知，不继续匹配后续规则
REDIRECT	重定向到另一个端口
MASQUERADE	将源IP地址伪装成防火墙网卡的IP地址，指定对应端口，不继续匹配后续规则
LOG	记录日志文件/var/log
SNAT	修改源IP地址，指定对应端口，不继续匹配后续规则
DNAT	修改目的IP地址，指定对应端口，不继续匹配后续规则
MIRROR	对调源和目的IP地址，不继续匹配后续规则
QUEUE	放入队列，交由用户程序处理，不继续匹配后续规则
RETURN	返回主规则链继续匹配规则
MARK	附加标记

知识讲解

选项(options)

知识讲解

选项	功能
-v/--verbose	输出详细信息
-x/--exact	显示精确数值
-n/--numeric	显示数值形式
--line-numbers	显示规则序号

iptables应用

iptables应用

- 通过iptables命令设置Netfilter防火墙，实现如下目标：
  - 接受来自192.168.1.0～192.168.1.255网段的数据包
  - 对于202.113.25.0～202.113.25.255网段，只接受来自202.113.25.174地址的数据包
  - 接受来自202.113.16.0～202.113.16.255网段的数据包
  - 拒绝其它主机通过ssh和telnet连接本机
  - 接受其它主机通过FTP连接本机

知识讲解

清除规则

- 清除filter规则表中的所有内置规则链
  - sudo iptables -t filter -F
- 清除filter规则表中的所有自定义规则链
  - sudo iptables -t filter -X
- 将filter规则表中所有规则链的计数器清零
  - sudo iptables -t filter -Z

知识讲解

默认策略

- sudo iptables -t filter -P INPUT ACCEPT
- sudo iptables -t filter -P OUTPUT ACCEPT
- sudo iptables -t filter -P FORWARD ACCEPT

知识讲解

添加规则

- 接受来自192.168.1.0～192.168.1.255网段的数据包
  - sudo iptables -t filter -A INPUT -i eth0 -s 192.168.1.0/24 -j ACCEPT
- 接受来自202.113.25.174地址的数据包
  - sudo iptables -t filter -A INPUT -i eth0 -s 202.113.25.174 -j ACCEPT
- 丢弃来自202.113.25.0～202.113.25.255网段的数据包
  - sudo iptables -t filter -A INPUT -i eth0 -s 202.113.25.0/24 -j DROP
- 接受来自202.113.16.0～202.113.16.255网段的数据包
  - sudo iptables -t filter -A INPUT -i eth0 -s 202.113.16.0/24 -j ACCEPT

知识讲解

添加规则

- 丢弃目的端口为22和23的TCP数据包
  - sudo iptables -t filter -A INPUT -i eth0 -p TCP -m multiport --dport 22,23 -j DROP
- 接受目的端口为20和21的TCP数据包
  - sudo iptables -t filter -A INPUT -i eth0 -p TCP -m multiport --dport 20,21 -j ACCEPT
- 列表显示所有规则链的所有规则
  - sudo iptables -L

知识讲解

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  192.168.1.0/24       anywhere
ACCEPT     all  --  202.113.25.174       anywhere
DROP       all  --  202.113.25.0/24      anywhere
ACCEPT     all  --  202.113.16.0/24      anywhere
DROP       tcp  --  anywhere             anywhere            tcp dpt:ssh
DROP       tcp  --  anywhere             anywhere            tcp dpt:telnet
ACCEPT     tcp  --  anywhere             anywhere            multiport dports ftp-data,ftp

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

总结和答疑