 ****

**J I A N G S U U N I V E R S I T Y**

简易防火墙设计

学院： 计算机与通信工程学院

班级： 信息安全2203

学号： 保密

姓名： 保密

2025年6月

# 使用指南

## 过滤规则管理 (rule)

功能：管理防火墙的过滤规则（允许/拒绝特定流量）

uapp rule <子命令> [选项]

| 子命令 | 选项/参数 | 功能说明 | 示例 |
| --- | --- | --- | --- |
| add | 无 | 交式添加过滤规则（会提示输入各项参数） | uapp rule add |
| del | <规则名称> | 删除指定名称的过滤规则 | uapp rule del my\_rule（删除名为my\_rule的规则） |
| ls/list | 无 | 列出所有过滤规则 | uapp rule ls |
| default | accept 或 drop | 设置默认动作（未匹配任何规则时的行为） | uapp rule default drop（默认拒绝所有流量） |

## NAT规则管理 (nat)

功能：管理网络地址转换（NAT）规则（仅支持源NAT）

uapp nat <子命令> [选项]

| 子命令 | 选项/参数 | 功能说明 | 示例 |
| --- | --- | --- | --- |
| add | 无 | 交互式添加NAT规则（会提示输入源IP、NAT IP和端口范围） | uapp nat add |
| del | <规则序号> | 删除指定序号的NAT规则（序号通过ls查看） | uapp nat del 1（删除序号为1的规则） |
| ls/list | 无 | 列出所有NAT规则（显示序号、源IP、NAT IP和端口范围） | uapp nat ls |

## 信息查看 (ls)

功能：查看日志、连接状态、规则等

uapp ls <类型> [数量]

| 子命令 | 选项/参数 | 功能说明 | 示例 |
| --- | --- | --- | --- |
| log/l | [数量] | 查看过滤日志（可选指定最近条目数量） | uapp ls log 10（显示最近10条日志） |
| con/c | 无 | 查看当前网络连接状态 | uapp ls con |
| rule/r | 无 | 列出所有过滤规则（同uapp rule ls） | uapp ls rule |
| nat/n | 无 | 列出所有NAT规则（同uapp nat ls） | uapp ls nat |

## 注意事项

简写支持：所有命令和子命令支持首字母简写（如uapp r a等效于uapp rule add）。

交互式输入：add操作会逐步提示输入参数，需按格式填写（如IP地址加掩码192.168.1.0/24）。

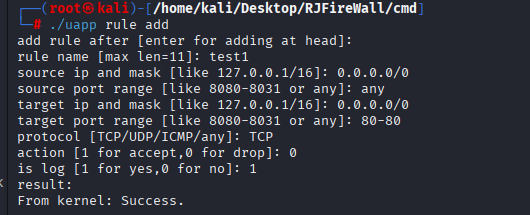
错误处理：输入无效时会提示wrong command并显示帮助信息。

# 从用户态到内核态的数据链

按照理解分析项目的步骤，本文将从数据的流转方式，综合各个代码模块的使用情况。

1. 实现进入cmd文件夹下，执行命令：./uapp rule add

实际执行的是 cmd/main.c 的 main() 函数。

在main函数的提示下，完成一系列参数的输入，如：  
 

1. 之后，参数写进入后，程序流进入common/exchange.c，

这个函数在 common/include/common.h 有声明，在 common/exchange.c 有实现。

Exchange.c就是负责将你写的参数，打包成信息，通过 netlink 发送给内核模块

1. 打包好的信息，传送给kernel\_mod/helpers/netlink\_helper.c

这个文件分别处理netlink信息的接受和分发

1. 之后规则被添加到规则链表中，也就是加入到红黑树中，也就是通过kernel\_mod/helpers/conn\_helper.c实现
2. 现在，规则被添加好了，kernel\_mod/hooks/hook\_main.c，就是专门检测ip流量情况的，这个函数根据规则，对ip数据流进行拦截
3. 最后，进行了操作后，会被保存到日志中，日志的实现在kernel\_mod/helpers/log\_helper.c
4. 在用户态可以通过./uapp ls log查看保存好的日志

以上就是该数据流的完整流动过程。

# Common文件夹代码详细分析

|  |
| --- |
| Common.h |
| 公共头文件，定义用户空间应用程序与内核防火墙/NAT模块通信的协议、数据结构及接口函数。 |
| \* 主要功能：  \* 1. 定义用户空间应用程序与内核模块之间通过Netlink进行通信时所使用的请求类型、响应类型常量。  \* 2. 定义双方共享的数据结构，如IP规则 (IPRule)、IP日志 (IPLog)、NAT记录/规则 (NATRecord)、连接日志 (ConnLog)、应用请求包 (APPRequest) 和内核响应头 (KernelResponseHeader)。  \* 3. 定义NAT转换类型常量。  \* 4. 定义用户空间应用程序专用的常量，如Netlink协议号、最大负载大小、错误码等。  \* 5. 定义一个用于封装内核响应的数据结构 (KernelResponse)。  \* 6. 声明一个核心的与内核交换数据的函数 (exchangeMsgK)。  \* 7. 声明一系列封装好的与内核交互的接口函数，方便上层应用调用以管理防火墙规则、NAT规则、获取日志和连接信息等。  \* 8. 声明一些IP地址格式转换的工具函数。  \*  \* 此头文件旨在为用户空间应用程序提供一套完整的与内核模块交互的规范和便捷接口。 |

|  |
| --- |
| Exchange.c |
| \* @brief 通过Netlink套接字与内核模块进行消息交换的核心函数  \* @param smsg 要发送到内核的消息数据指针  \* @param slen 发送消息数据的长度（字节）  \* @return struct KernelResponse 包含以下字段：  \* - code: 正数表示响应体长度，负数表示错误码(ERROR\_CODE\_EXCHANGE)  \* - data: 完整响应数据(包含头部+实际数据)  \* - header: 指向响应头部的指针  \* - body: 指向响应实际数据的指针  \* @note 函数执行流程：  \* 1. 创建并绑定Netlink套接字  \* 2. 构建Netlink消息头并填充发送数据  \* 3. 发送消息到内核并等待响应  \* 4. 解析响应数据并组织返回结构  \* 5. 错误处理贯穿整个流程，确保资源释放  \*/ |

|  |
| --- |
| Helper.c |
| 用户空间应用程序接口函数实现。 |
| \* 主要功能：  \* 此文件实现了在 `common.h` (或 `common\_app.h`) 中声明的一系列用户接口函数。  \* 这些函数作为用户空间应用程序与内核防火墙/NAT模块交互的便捷封装层。  \* 它们负责：  \* 1. 构建符合通信协议的 `APPRequest` 结构体。  \* 2. 调用底层的 `exchangeMsgK` 函数通过Netlink与内核模块进行数据交换。  \* 3. 对输入参数进行初步校验 (例如IP地址格式)。  \* 4. 返回内核处理结果，封装在 `KernelResponse` 结构体中。  \*  \* 包含的函数有：  \* - 添加/删除/获取IP过滤规则  \* - 添加/删除/获取NAT规则  \* - 设置默认防火墙动作  \* - 获取IP日志  \* - 获取当前活动连接  \*  \* 这些函数使得上层应用程序可以更简单地管理防火墙策略和监控网络状态，  \* 无需直接处理复杂的Netlink通信细节。  \*/ |

|  |
| --- |
| Tools.c |
| \* @brief IP地址转换工具函数实现。 |
| \* 主要功能：  \* 此文件包含了一系列用于在字符串表示和32位无符号整数表示之间转换IPv4地址的工具函数。  \* 这些函数处理：  \* 1. 将点分十进制的IP地址字符串（可选地带有CIDR掩码长度，如 "192.168.1.1/24"）  \* 转换为32位整数形式的IP地址和32位整数形式的子网掩码。  \* 2. 将32位整数形式的IP地址和子网掩码（或掩码长度）转换回点分十进制的字符串表示。  \* 3. 将32位整数形式的IP地址转换为不带掩码的点分十进制字符串。  \* 4. 将32位整数形式的IP地址和端口号格式化为 "A.B.C.D:PORT" 形式的字符串。  \*  \* 这些工具函数对于解析用户输入的IP地址以及格式化IP地址以供显示非常有用。 |

# cmd用户态文件夹代码详细分析

|  |
| --- |
| Main.c |
| @brief 主函数，处理命令行参数并执行相应操作 |
| 该主函数包括：  @note 支持的命令包括：  \* - 过滤规则管理(rule)  \* - NAT规则管理(nat)  \* - 查看各种信息(ls) |

|  |
| --- |
| Kernel.c |
| @brief 用户空间应用程序的响应处理及数据显示模块。 |
| 主要功能：  \* 此文件负责处理从内核模块接收到的响应，并以用户友好的格式将这些信息（如IP规则、  \* NAT规则、日志、活动连接等）显示在命令行界面上。  \*  \* 包含的主要功能有：  \* 1. `dealResponseAtCmd`: 核心响应处理函数。它接收一个 `KernelResponse` 结构体，  \* 首先检查响应中的错误码，如果存在错误则打印相应的错误信息。如果响应成功，  \* 它会根据响应头中的 `bodyTp` (消息体类型) 调用相应的 `show...` 函数来  \* 格式化并显示数据。此函数还负责在适当的时候释放从内核接收到的数据内存。  \* 2. 一系列 `show...` 函数 (例如 `showRules`, `showNATRules`, `showLogs`, `showConns`)：  \* 这些函数接收从内核传来的数据数组及其长度，并将每条记录格式化为易于阅读的  \* 表格或列表形式，然后打印到标准输出。  \* 3. 一系列 `showOne...` 函数 (例如 `showOneRule`, `showOneLog`, `showOneConn`)：  \* 这些是辅助函数，负责格式化并打印单条记录的详细信息。  \* 4. `printLine`: 一个简单的工具函数，用于打印指定长度的横线，以美化表格输出。  \*  \* 此模块是用户空间命令行工具与用户交互的关键部分，它将内核返回的原始数据  \* 转换为人类可读的信息。 |

# Kernel\_mod内核态核心代码分析

|  |
| --- |
| Tools.c |
| @brief 网络工具函数实现。 |
| \* 主要功能：  \* 此文件包含了一些用于处理网络数据包和IP地址比较的辅助函数。  \*  \* 包含的函数有：  \* 1. `getPort`: 从给定的 `sk\_buff` (套接字缓冲区) 中提取传输层 (TCP或UDP)  \* 的源端口和目的端口。对于ICMP或其他协议，端口号被设置为0。  \* 2. `isIPMatch`: 比较两个IP地址在给定子网掩码下是否匹配（即它们是否属于同一个子网）。  \*  \* 这些函数通常在网络数据包处理（如Netfilter钩子函数中）或IP规则匹配逻辑中被调用。 |

|  |
| --- |
| Mod\_main.c |
| @brief Linux内核防火墙与NAT模块的主体文件。 |
| 主要功能：  \* 此文件是Linux内核模块的入口点和出口点，负责初始化和清理防火墙及NAT功能所需的核心组件。  \* 它执行以下关键操作：  \* 1. \*\*定义Netfilter钩子操作\*\*: 声明并初始化多个 `struct nf\_hook\_ops` 结构体。  \* 这些结构体定义了在网络协议栈的不同点（钩子点）应该执行哪些函数（钩子函数），  \* 以及它们的协议族（IPv4）和优先级。  \* - `nfop\_in` 和 `nfop\_out`: 用于主要的IP包过滤逻辑，分别注册在 `NF\_INET\_PRE\_ROUTING`  \* (数据包刚进入网络栈，路由决策之前) 和 `NF\_INET\_POST\_ROUTING` (数据包在路由决策之后，  \* 即将离开本机之前) 钩子点，使用 `hook\_main` 函数。  \* - `natop\_in` 和 `natop\_out`: 用于NAT（网络地址转换）逻辑，分别注册在  \* `NF\_INET\_PRE\_ROUTING` (用于DNAT，在路由前修改目的地址) 和 `NF\_INET\_POST\_ROUTING`  \* (用于SNAT，在路由后修改源地址) 钩子点，使用 `hook\_nat\_in` 和 `hook\_nat\_out` 函数，  \* 并具有特定的NAT优先级。  \* 2. \*\*模块初始化 (`mod\_init`)\*\*:  \* - 在模块加载时被调用。  \* - 打印模块加载信息到内核日志。  \* - 使用 `nf\_register\_net\_hook` 函数将上面定义的四个Netfilter钩子操作注册到内核网络栈中，  \* 使其能够拦截和处理网络数据包。  \* - 调用 `netlink\_init()` 初始化Netlink通信接口，用于内核模块与用户空间应用程序的交互。  \* - 调用 `conn\_init()` 初始化连接跟踪机制，用于管理和跟踪网络连接状态。  \* 3. \*\*模块退出 (`mod\_exit`)\*\*:  \* - 在模块卸载时被调用。  \* - 打印模块卸载信息到内核日志。  \* - 使用 `nf\_unregister\_net\_hook` 函数从内核网络栈中注销之前注册的四个Netfilter钩子，  \* 停止数据包拦截。  \* - 调用 `netlink\_release()` 释放Netlink通信资源。  \* - 调用 `conn\_exit()` 清理连接跟踪机制的资源。  \* 4. \*\*模块元数据\*\*:  \* - 使用 `MODULE\_LICENSE("GPL")` 声明模块的许可证。  \* - 使用 `MODULE\_AUTHOR("jyq")` 声明模块的作者。  \* - 使用 `module\_init(mod\_init)` 和 `module\_exit(mod\_exit)` 宏指定模块的初始化和退出函数。  \*  \* 此文件是整个内核防火墙/NAT模块能够工作的基石，它将自定义的数据包处理逻辑（通过钩子函数）  \* 插入到Linux内核的网络处理流程中。 |

## include文件夹

该文件夹主要包含一些头文件的定义和一些结构体的定义

## Hooks文件夹

|  |
| --- |
| Hooks\_nat.c |
| @brief Netfilter钩子函数实现，用于网络地址转换 (NAT)。 |
| 主要功能：  \* 此文件包含两个核心的Netfilter钩子函数：`hook\_nat\_in` 和 `hook\_nat\_out`。  \* 这些函数分别负责处理入站数据包的目的NAT (DNAT) 和出站数据包的源NAT (SNAT)。  \* 它们依赖于连接跟踪系统 (`connNode`, `hasConn`, `addConn`, `setConnNAT`) 和  \* NAT规则匹配 (`matchNATRule`, `getNewNATPort`, `genNATRecord`) 来实现有状态的NAT转换。  \*  \* - `hook\_nat\_in`: 注册在 `NF\_INET\_PRE\_ROUTING` 钩子点，用于DNAT。  \* 它检查进入的数据包是否对应于一个已建立的、需要DNAT的连接。如果是，  \* 它会根据连接中存储的NAT记录修改数据包的目的IP地址和端口，并重新计算校验和。  \*  \* - `hook\_nat\_out`: 注册在 `NF\_INET\_POST\_ROUTING` 钩子点，用于SNAT。  \* 对于出站数据包，它首先检查连接跟踪表中是否已存在SNAT记录。  \* 如果不存在，它会尝试匹配预定义的NAT规则。如果匹配成功，则会分配一个新的源端口，  \* 创建SNAT记录并存储在连接中，同时也会为返回流量创建相应的反向（DNAT）映射。  \* 然后，它修改数据包的源IP地址和端口，并重新计算校验和。  \*  \* 这两个函数是实现防火墙NAT功能的关键，它们确保了内部网络的主机能够通过  \* 单个公共IP地址访问外部网络，并且外部请求能够被正确地转发到内部网络的目标主机。 |

|  |
| --- |
| Hook\_main.c |
| 用于在Linux内核网络协议栈中执行IP包过滤 |

这两个函数是内核过滤数据包的核心函数

一个用于过滤ip数据包，一个用于转换nat数据包

## Helpers文件夹

|  |
| --- |
| App.helper.c |
| @brief 内核模块与用户空间应用程序通信处理逻辑。 |
| 主要功能：  \* 此文件实现了内核模块通过Netlink接收来自用户空间应用程序的请求，并对其进行处理的逻辑。  \* 同时，它也包含了内核模块向用户空间应用程序发送响应或消息的功能。  \*  \* 包含的主要函数有：  \* 1. `sendMsgToApp`: 一个辅助函数，用于将简单的文本消息打包并通过Netlink发送给指定PID的用户空间进程。  \* 2. `dealWithSetAction`: 当防火墙的默认动作被修改时（特别是从允许变为拒绝），此函数负责执行一些清理操作，  \* 例如清除所有现有的网络连接，以确保新的默认策略能够立即生效。  \* 3. `dealAppMessage`: 这是核心的Netlink消息处理函数。它接收一个来自用户空间应用的消息 (`APPRequest`)，  \* 并根据请求类型 (`req->tp`) 分发到不同的处理分支。这些分支负责调用相应的内部函数来执行  \* 诸如获取规则/日志/连接、添加/删除规则、设置默认动作等操作。处理完毕后，它会调用  \* `nlSend` (或 `sendMsgToApp`) 将结果或状态信息返回给用户空间应用程序。  \*  \* 此模块是实现用户空间配置和管理内核防火墙功能的关键接口。 |

|  |
| --- |
| Conn\_helper.c |
| @brief 有状态连接跟踪模块实现。 |
| 主要功能：  \* 此文件实现了防火墙和NAT功能所需的核心连接跟踪机制。它使用红黑树（Red-Black Tree）  \* 来高效地存储和检索活动网络连接的信息。主要功能包括：  \*  \* 1. \*\*红黑树操作封装\*\*:  \* - `connKeyCmp`: 比较两个连接的键（由源IP、目的IP、源端口和目的端口组成），用于在红黑树中排序和查找。  \* - `searchNode`: 在红黑树中根据连接键查找对应的连接节点 (`connNode`)。  \* - `insertNode`: 将新的连接节点插入到红黑树中，并保持树的平衡。  \* - `eraseNode`: 从红黑树中删除指定的连接节点，并释放其内存。  \* 所有这些操作都通过读写锁 (`connLock`) 进行保护，以确保并发访问的线程安全。  \*  \* 2. \*\*连接管理业务逻辑\*\*:  \* - `isTimeout`: 检查给定的超时时间戳是否已过期。  \* - `addConnExpires`: 更新指定连接节点的超时时间。  \* - `hasConn`: 供外部模块（如 `hook\_main`）调用，用于检查是否存在与给定五元组匹配的活动连接。  \* 如果找到，则刷新其超时时间。  \* - `addConn`: 供外部模块调用，用于创建一个新的连接跟踪条目并将其插入红黑树。  \* 新连接会设置初始超时时间、日志标志、协议和NAT类型。  \* - `setConnNAT`: 为指定的连接节点设置或更新其NAT转换记录和NAT类型。  \* - `getNewNATPort`: 为SNAT操作从指定的NAT规则定义的端口范围内查找并分配一个可用的新端口。  \* 它会遍历现有连接以避免端口冲突。  \* - `formAllConns`: 将红黑树中所有活动的连接信息打包成一个可通过Netlink发送给用户空间的数据块。  \* - `eraseConnRelated`: 根据给定的IP过滤规则，删除连接池中所有匹配该规则的连接。  \* 这通常在防火墙策略更改（如默认动作变为DROP）或删除某条规则时使用。  \* - `rollConn`: 清理连接池，删除所有已超时的连接。此函数由定时器周期性调用。  \*  \* 3. \*\*定时器管理\*\*:  \* - `conn\_timer\_callback`: 定时器的回调函数，在定时器触发时调用 `rollConn` 来清理超时连接，  \* 并重新设置定时器以供下次触发。  \* - `conn\_init`: 初始化连接跟踪模块，包括设置并启动用于周期性清理的内核定时器 (`conn\_timer`)。  \* - `conn\_exit`: 在模块卸载时清理连接跟踪模块，主要是删除（停止）内核定时器。  \*  \* 此模块通过高效的连接存储和及时的超时管理，为防火墙提供了有状态的特性，  \* 使得对已建立连接的后续数据包可以快速处理，并为NAT功能提供了必要的会话保持能力。  \* 内核版本兼容性宏 (`LINUX\_VERSION\_CODE < KERNEL\_VERSION(4,14,0)`) 用于处理不同内核版本  \* 定时器API的差异。 |

|  |
| --- |
| Rule\_helper.c |
| IP过滤规则管理模块实现。 |
| \* 主要功能：  \* 此文件负责管理防火墙的IP过滤规则。这些规则存储在一个单向链表中。  \* 模块提供了添加、删除、查询和匹配IP规则的功能。  \* 所有对规则链表的访问都通过读写锁 (`ipRuleLock`) 进行保护，以确保并发安全。  \*  \* 包含的主要函数有：  \* 1. `addIPRuleToChain`: 将一条新的IP规则添加到规则链表中。可以指定新规则插入的位置  \* （在某条现有规则之后，或在链表头部）。如果添加的规则不是允许 (NF\_ACCEPT) 类型，  \* 则会调用 `eraseConnRelated` 来清除可能与此新（拒绝）规则冲突的现有连接。  \* 2. `delIPRuleFromChain`: 从规则链表中删除所有名称与指定名称匹配的IP规则。  \* 在删除规则时，同样会调用 `eraseConnRelated` 来清除与被删除规则相关的连接。  \* 3. `formAllIPRules`: 将当前规则链表中的所有IP规则打包成一个可通过Netlink发送给  \* 用户空间应用程序的数据块。  \* 4. `matchOneRule`: 辅助函数，用于判断一个网络数据包的五元组信息（源/目的IP、  \* 源/目的端口、协议）是否与单条给定的IP规则匹配。  \* 5. `matchIPRules`: 核心规则匹配函数。它接收一个网络数据包 (`sk\_buff`)，  \* 遍历整个IP规则链表，并使用 `matchOneRule` 逐条比较。如果找到第一条匹配的规则，  \* 则返回该规则的副本，并通过输出参数 `isMatch` 指示匹配成功。  \*  \* 此模块是防火墙执行基于规则的包过滤功能的基础。 |

|  |
| --- |
| Netlink\_helper.c |
| 用于通过 Netlink 向用户空间进程发送数据。 |

|  |
| --- |
| Nat\_helper.c |
| 主要处理nat规则 |

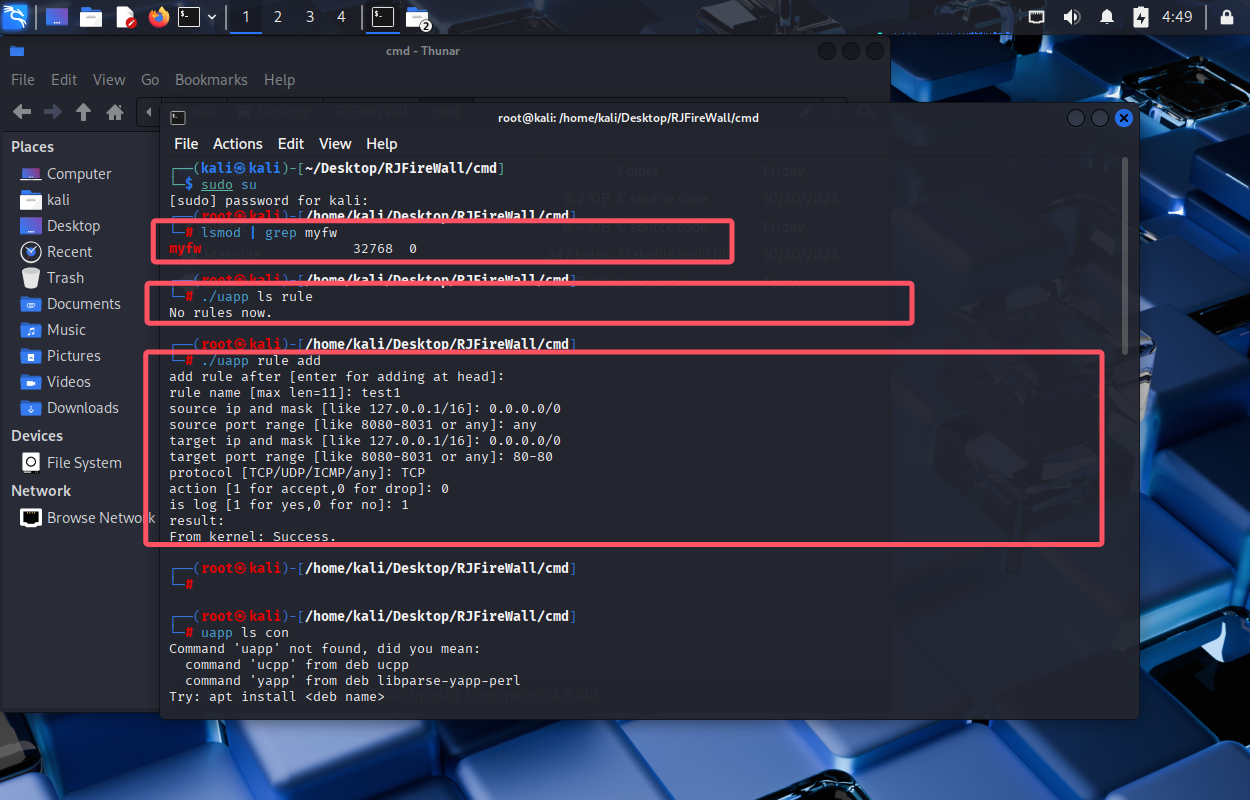
|  |
| --- |
| Log\_helper.c |
| 日志功能的处理 |

## 核心模块的总结

核心处理逻辑中，使用文件读写锁贯穿整个kernel\_mod项目的代码，这是比较重要的一点  
  
 其次，红黑树实现了网络连接状态的管理。

# 实验截图

## 添加rule



## 检查rule的添加情况

## 进行测试

# 总结