



Ingress firewall implementation FreeRTOS

Politecnico di Torino:
Computer Architecture and
Operating Systems course:
2023/2024

Group 5:
Michele Seira, Fardo Giorgio, Luca Ponzo, Gianfranco Trad

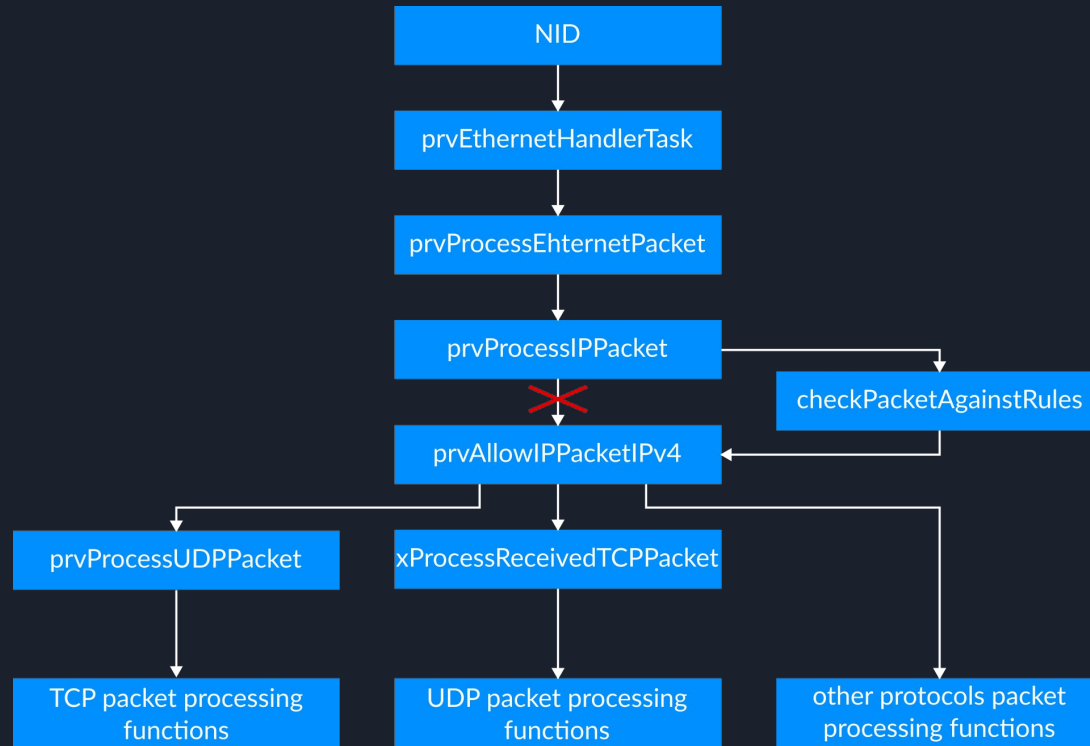
Project intent: create a working firewall for FreeRTOS

- Requirements :
 - Function entry points for filtering packets
 - Handling of rejected packets
 - Print to stderr
 - Rule generation
 - YAML to C struct
 - Rule storage
 - Inline C structs in *rules.h*
 - (WIP) integration with IDS
 - Snort



Understanding FreeRTOS IP stack :

Packet processing pipeline



Functions

```
if( ( uxHeaderLength > ( pxNetworkBuffer->xDataLength - ipSIZE_OF_ETH_HEADER ) ) ||
    ( uxHeaderLength < ipSIZE_OF_IPv4_HEADER ) || checkPacketAgainstRules (ruleset, NOR, pxIPHeader, pxNetworkBuffer) )
// firewall injection point
{
    eReturn = eReleaseBuffer ;
}
```

FreeRTOS_IP.c:prvProcessIPPacket

packet interception point

```
uint8_t checkPacketAgainstRules (struct rule ruleset[], int num_rules, const IPHeader_t * pxIPHeader, NetworkBufferDescriptor_t const * pxNetworkBuffer) {
//collapse all checks in one function ( fast enough?)
switch (pxIPHeader->ucProtocol){
    case ipPROTOCOL_UDP:{
        const UDPpacket_t * pxUDPpacket = ( ( UDPpacket_t * ) pxNetworkBuffer->puEthernetBuffer );
        const UDPHeader_t * pxUDPHeader = &(pxUDPpacket->xUDPHeader );
//Get the header of the UDP packet from the pxNetworkBuffer to extract the ports
        if (checkPacketsWithPorts (ruleset, num_rules, pxIPHeader, pxUDPHeader->usSourcePort, pxUDPHeader->usDestinationPort) == 1){
            writeToPcap (pxIPHeader, pxUDPHeader->usSourcePort, pxUDPHeader->usDestinationPort );
//If the packet is rejected by the rules pass to login function
            return 1;
        }
        return 0;
    }
    break;
    case ipPROTOCOL_TCP:{
        const ProtocolHeaders_t *pxProtocolHeaders = ( ( ProtocolHeaders_t * ) &(pxNetworkBuffer->puEthernetBuffer [ ipSIZE_OF_ETH_HEADER +
uxIPHeaderSizePacket ( pxNetworkBuffer ) ] ) );
//Extract to get to the TCP Header fields
        if (checkPacketsWithPorts (ruleset, num_rules, pxIPHeader, pxProtocolHeaders->xTCPHeader.usSourcePort, pxProtocolHeaders->xTCPHeader.usDestinationPort) == 1){
            writeToPcap (pxIPHeader, pxProtocolHeaders->xTCPHeader.usSourcePort, pxProtocolHeaders->xTCPHeader.usDestinationPort );
            return 1; //return 1 discards the packet
        }
        return 0;
    }
    break;
    case ipPROTOCOL_ICMP:{
        ICMPpacket_t * pxICMPpacket = ( ( ICMPpacket_t * ) pxNetworkBuffer->puEthernetBuffer ); //ICMP doesn't need ports
        if (checkIPs (ruleset, num_rules, IPHeader_t * &(pxICMPpacket->xIPHeader)) == 1){
            writeToPcap (pxIPHeader, 0, 0);
            return 1;
        }
        return 0;
    }
    default:{
        if (checkIPs (ruleset, num_rules, pxIPHeader) == 1){ //other protocols like ARP
            return 1;
        }
        return 0;
    }
}
}
```

FreeRTOS_IP.c:checkPacketAgainstRules

```
uint8_t checkIPs (struct rule ruleset[], int num_rules, const IPHeader_t * pxIPHeader){
    int i;
    for (i = 0; i < num_rules; i++){
        if (pxIPHeader->ulSourceIPAddress == ruleset[i].src &&
            pxIPHeader->ulDestinationIPAddress == ruleset[i].dst &&
            pxIPHeader->ucProtocol == ruleset[i].proto ) {
            return ruleset[i].action; // match found execute corresponding action on the packet
        }
    }
    return 1; // no match found being this a block list we just let the packet trough
};

uint8_t checkPacketsWithPorts (struct rule ruleset[], int num_rules, const IPHeader_t * pxIPHeader, uint16_t usSourcePort, uint16_t usDestinationPort) { //collapse all checks in one function ( fast enough?)
    int i;
    for (i = 0; i < num_rules; i++){
        if (pxIPHeader->ucProtocol == 0){
            if (pxIPHeader->ulSourceIPAddress == ruleset[i].src &&
                pxIPHeader->ulDestinationIPAddress == ruleset[i].dst &&
                usSourcePort == ruleset[i].port_src && // convert port to network byte order
                usDestinationPort == ruleset[i].port_dst && // convert port to network byte order
                pxIPHeader->ucProtocol == ruleset[i].proto ) {
                return ruleset[i].action;
            }
            // match found execute corresponding action on the packet this can be used both as blacklist and whitelist.
        }
        else if (pxIPHeader->ucProtocol == 17){
            printf ("%d ", usSourcePort);
            printf ("%d ", usDestinationPort );
            if (pxIPHeader->ulSourceIPAddress == ruleset[i].src &&
                pxIPHeader->ulDestinationIPAddress == ruleset[i].dst &&
                usSourcePort == ruleset[i].port_src && // convert port to network byte order -> done in the preprocessor for the rules
                usDestinationPort == ruleset[i].port_dst && // convert port to network byte order -> done in the preprocessor for the rules
                pxIPHeader->ucProtocol == ruleset[i].proto ) {
                return ruleset[i].action;
            }
            // match found execute corresponding action on the packet this can be used both as blacklist and/or whitelist.
        }
    }
    return 1; // no match found being this a allow list we just let the packet trough
};
```

FreeRTOS_IP.c:checkIPs & checkPacketsWithPort

Handling of rejected packets

```
void writeToPcap(const IPHeader_t * pxIPHeader, uint16_t usSourcePort, uint16_t usDestinationPort){
    //Print to serial console with identifier : clamp fields : and output

    //Write the discarded packets data to stderr -> qemu write output to file -> process in PCAP only the lines with #PCAP -> export in Wireshark
    fprintf(stderr, "#PCAP %d %d %d %d %d\n", pxIPHeader->ulSourceIPAddress, pxIPHeader->ulDestinationIPAddress, pxIPHeader->ucProtocol,
    usSourcePort, usDestinationPort);
}
```

FreeRTOS_IPC:writeToPcap

- write the data of the rejected packets in *stderr*
 - marking every rejected packet with *#PCAP*
- pipe the *qemu-system-arm* output, that is all directed to *stderr*, to a file *out.log*
- process the file to compile a *pcap*

```
import socket
import struct
import dpkt
import re

def get_initial_ip(ip_int):
    oct_arr = [(ip_int >> 24) & 255, (ip_int >> 16) & 255, (ip_int >> 8) & 255, ip_int & 255]
    oct_arr.reverse()
    ip_str = ".".join(map(str, oct_arr))
    return ip_str

def get_initial_port(port_int):
    port_int = int(port_int)
    port_int = ((port_int & 0xFF) << 8) | ((port_int >> 8) & 0xFF)
    return port_int

with open('./PART_3/out.log', 'r') as file:
    with open('./output.pcap', 'wb') as pcap_file:
        pcap_writer = dpkt.pcap.Writer(pcap_file)
        for line in file:
            if line.startswith('#PCAP'):
                # Extract the relevant information using regular expressions
                match = re.match(r'#PCAP (Vd+) (Vd+) (Vd+) (Vd+) (Vd+)', line)
                if match:
                    source_ip = get_initial_ip(int(match.group(1)))
                    destination_ip = get_initial_ip(int(match.group(2)))
                    source_port = get_initial_port(int(match.group(4)))
                    destination_port = get_initial_port(int(match.group(5)))
                    protocol = int(match.group(3))

                    eth = dpkt.ethernet.Ethernet()

                    if protocol == 6:
                        tcp = dpkt.tcp.TCP()
                        tcp.sport = source_port
                        tcp.dport = destination_port
                        tcp.data = b'TCP packet'
                        ip = dpkt.ip.IP(src=socket.inet_aton(source_ip), dst=socket.inet_aton(
                            destination_ip), p=dpkt.ip.IP_PROTO_TCP)
                        ip.data = tcp
                        eth.data = ip
                        elif protocol == 17:
                            udp = dpkt.udp.UDP()
                            udp.sport = source_port
                            udp.dport = destination_port
                            udp.data = b'UDP packet'
                            ip = dpkt.ip.IP(src=socket.inet_aton(source_ip), dst=socket.inet_aton(
                                destination_ip), p=dpkt.ip.IP_PROTO_UDP)
                            ip.data = udp
                            eth.data = ip
                            elif protocol == 1:
                                # Extract an ICMP packet
                                icmp = dpkt.icmp.ICMP()
                                icmp.type = dpkt.icmp.ICMP_ECHO
                                icmp.data = b'ICMP packet'
                                ip = dpkt.ip.IP(src=socket.inet_aton(source_ip), dst=socket.inet_aton(
                                    destination_ip), p=dpkt.ip.IP_PROTO_ICMP)
                                ip.data = icmp
                                eth.data = ip
                            else:
                                ip = dpkt.ip.IP(src=socket.inet_aton(source_ip), dst=socket.inet_aton(
                                    destination_ip))
                                eth.data = ip

                            pcap_writer.writepkt(eth)

# Print a message to indicate the completion of the task
print('Packets written to output.pcap file.')
```

Python .log to PCAP conversion function

Rules generation

```
ruleset:  
- source: 192.168.122.50  
  destination: 192.168.122.10  
  port_source: 200  
  port_destination: 4050  
  protocol: 17  
  action: 0  
- source: 192.168.122.1  
  destination: 192.168.122.10  
  port_source: 200  
  port_destination: 33  
  protocol: 6  
  action: 0  
- source: 192.168.122.1  
  destination: 192.168.122.10  
  port_source: ANY  
  port_destination: ANY  
  protocol: 1  
  action: 0
```

rules.yaml

Rules storage

```
#ifndef __RULE_SET_FIREWALL_  
#define __RULE_SET_FIREWALL_ 1  
  
#include <stdlib.h>  
  
#define NOR 3  
  
//A Firewall rule structure  
typedef struct rule {  
    uint32_t src; // Source IP address in network byte order  
    uint32_t dst; // Destination IP address in network byte order  
    uint16_t port_src; // Source port number in network byte order  
    uint16_t port_dst; // Destination port number in network byte order  
    uint8_t proto; // 2-bit mask representing protocol type  
    uint8_t action;  
    //action to do with packets from the source ip 0 >> accept packets 1 >> reject packets  
}Rule;  
  
//Current ruleset loaded in the firewall  
Rule ruleset[NOR] = {  
    {846899392, 175810752, 51200, 53775, 17, 0 },  
    {24815808, 175810752, 51200, 8448, 6, 0 },  
    {24815808, 175810752, 0, 0, 1, 0 },  
};  
  
#endif
```

FreeRTOS-Plus-TCP/rules.h:

Tests for rules and output PCAP

```
send(IP(dst="192.168.122.10" ,src="192.168.122.50" )/UDP(dport=4050,sport=200)/Raw(load="abc" ),
iface="virbr0" )
send(IP(dst="192.168.122.10" ,src="192.168.122.1" )/TCP(dport=33,sport=200), iface="virbr0" )
send(IP(dst="192.168.122.10" ,src="192.168.122.1" )/ICMP(), iface="virbr0" )
send(IP(dst="192.168.122.10" ,src="192.168.122.50" )/UDP(dport=3050,sport=200)/Raw(load="def" ),
iface="virbr0" )
send(IP(dst="192.168.122.10" ,src="192.168.122.1" )/TCP(dport=6968,sport=200), iface="virbr0" )
send(IP(dst="192.168.122.10" ,src="192.168.122.50" )/ICMP(), iface="virbr0" )
```

Packet generation calls that are fed to **scapy** : 6 packets 3 of which are expected to be rejected by the rules defined earlier

| Apply a display filter ... <Ctrl-/> | | | | | |
|-------------------------------------|----------|----------------|----------------|----------|--|
| No. | Time | Source | Destination | Protocol | Length Info |
| 1 | 0.000000 | 192.168.122.50 | 192.168.122.10 | UDP | 52 200 → 3050 Len=0 |
| 2 | 0.000055 | 192.168.122.1 | 192.168.122.10 | TCP | 64 200 → 6968 [SYN] Seq=0 Win=65535 Len=10 |
| 3 | 0.000140 | 192.168.122.50 | 192.168.122.10 | ICMP | 49 Echo (ping) request id=0x4943, seq=19792/20557, ttl=64 (no response found!) |

The 3 rejected packets as expected

Integration of IDS

- preliminary work for POC using SNORT to process rejected packets by the firewall
- needs second NIC or SNORT endpoint with TCP retransmission mechanism

```
scccccp///pSP///p      p//Y      |      -- Jean De Clerck
sY/////////y  caa      S//P      |
cayCyayP//Ya      pY/Ya
sY/PsY///YCc      aC//Yp
sc  sccaCY//PCypaapyCP//YSs
    spCPY/////////YPSps
    ccaacs

using IPython 8.5.0
send(IP(dst="192.168.122.10",src="192.168.122.50")/UDP(dport=4050,sport=200)
/Raw(load="abc"), iface="virbr0")
send(IP(dst="192.168.122.10",src="192.168.122.1")/TCP(dport=33,sport=200),
iface="virbr0")
send(IP(dst="192.168.122.10",src="192.168.122.1")/ICMP(), iface="virbr0")
send(IP(dst="192.168.122.10",src="192.168.122.50")/UDP(dport=3050,sport=200)
/Raw(load="def"), iface="virbr0")
send(IP(dst="192.168.122.10",src="192.168.122.1")/TCP(dport=6968,sport=200)
, iface="virbr0")
send(IP(dst="192.168.122.10",src="192.168.122.50")/ICMP(), iface="virbr0")
```

```
gi4n@ubuntu:~$ sudo snort -q -A console -c /etc/snort/rules/local.rules -i virbr0
02/25-13:36:25.430810  [**] [1:10000001:0] ICMP Traffic Detected [**] [Priority: 0]
CMP} 192.168.122.1 -> 192.168.122.10
02/25-13:36:25.431324  [**] [1:10000001:0] ICMP Traffic Detected [**] [Priority: 0]
CMP} 192.168.122.10 -> 192.168.122.1
02/25-13:36:25.529350  [**] [1:10000001:0] ICMP Traffic Detected [**] [Priority: 0]
CMP} 192.168.122.50 -> 192.168.122.10
```

Rules defined to monitoring only the ICMP packets through SNORT, only proof of concept needs more work

Thanks !

And now time for the live demo 🙋