

# Software Architecture for an Intelligent Firewall Based on Linux Netfilter

Sergey Staroletov  
Independent Research Enthusiast  
Altai territory, Russia  
serg\_soft@mail.ru

**Abstract**—A firewall is a tool for filtering network traffic passing through a given router or network endpoint. Initially, such systems used to have only static rules that allowed or denied traffic according to specified addresses, ports or protocols. Today, with the complication of information systems and the construction of decentralized IoT systems containing a large number of embedded controllers, there is a need to detect potential anomalies in the transmitted network traffic. Such detection should be performed quickly enough and not require serious hardware resources. After the detection, rules for the firewall should be automatically mined and applied immediately, and they can later be potentially canceled after some new data arrives. In this paper, we discuss the software architecture of a network anomaly detection system. We install a Linux Netfilter hook, in which we capture the traffic and send it to a ring buffer shared with analysis pipelines in the userspace. It allows the system to make reactions in the form of information to the user as well as to mine firewall rules, that are transmitted back to kernel space using Netlink sockets. As for the detectors, we currently use variable-order Markov chains to reveal anomalies in TCP traffic, as well as self-organizing Kohonen maps for classifying traffic flows and determining whether it is normal or abnormal. Thus, we briefly describe all software solutions to handle passing traffic frames and analyze them using fast processing techniques.

## I. PRELIMINARIES IN NETFILTER

Netfilter [1] is a subsystem of the Linux Kernel that allows users to filter packets based on their headers and contents. Rules for handling packets with different priorities can be set from userspace using the iptables tool. For developers of kernel modules, Netfilter offers an interface for defining hooks in the form of a special kind of function that will be called when each packet passes through the Linux network subsystem according to a given direction, subsystem (IPv4, IPv6) and priority. Such a function has access to the raw data of passing network packets of MTU size in the form of frames of the second-level of the ISO/OSI model [2]. Interestingly, the WFP framework [3] for Windows system works likewise.

Each such frame for Netfilter is a “matryoshka” containing protocol headers starting from level 2, which can be extracted using the corresponding functions. Inside a lower layer header exists an upper layer protocol number, and using this information, one can further convert the header data to the corresponding structure. Thus, by registering a hook and analyzing the headers and data, one can decide what to do next with this frame. Namely, the simplest options are to skip

it for further processing by other hooks and kernel or drop it at this level. The place of our approach is demonstrated in the following code, which is implemented in a kernel module (we use Linux Kernel 5.2.11), Fig. 1:

```
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/ip.h>
static struct nf_hook_ops nf_ops_out;

//Module starting point:
int init_module(void) {
    //Our hook function
    nf_ops_out.hook = main_hook;
    //for IPv4 traffic
    nf_ops_out.pf = PF_INET;
    //for input traffic
    nf_ops_out.hooknum = NF_INET_LOCAL_IN;
    //first priority among other hooks
    nf_ops_out.priority = NF_IP_PRI_FIRST;
    //Register the hook in the kernel
    nf_register_net_hook(&init_net, &nf_ops_out);
    return 0;
}

//A hook function:
unsigned int main_hook(void *priv,
    struct sk_buff *skb,
    const struct nf_hook_state *state) {
    //If needed, check here an interface
    //of interest, for example,
    //using input state->in->name
    //Obtain an IP header:
    struct iphdr *ip_header = (struct iphdr*)
        skb_network_header(skb);
    //Frame data can be extracted from skb here
    //Make a decision for a packet(frame):
    if (...) { //condition to filter a frame
        //Accept this frame:
        return NF_ACCEPT;
    } else {
        //Or drop this frame:
        return NF_DROP;
    }
}
```

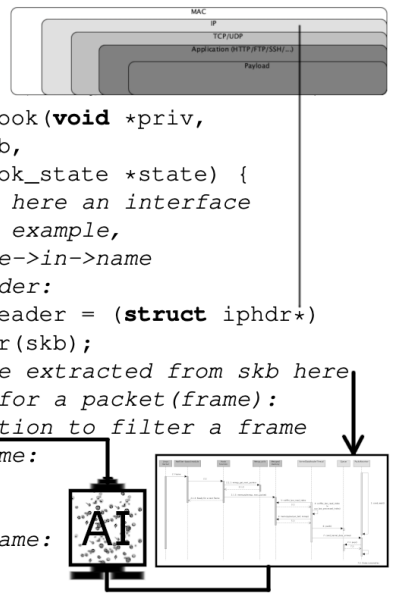


Fig. 1. A Netfilter hook is our starting point

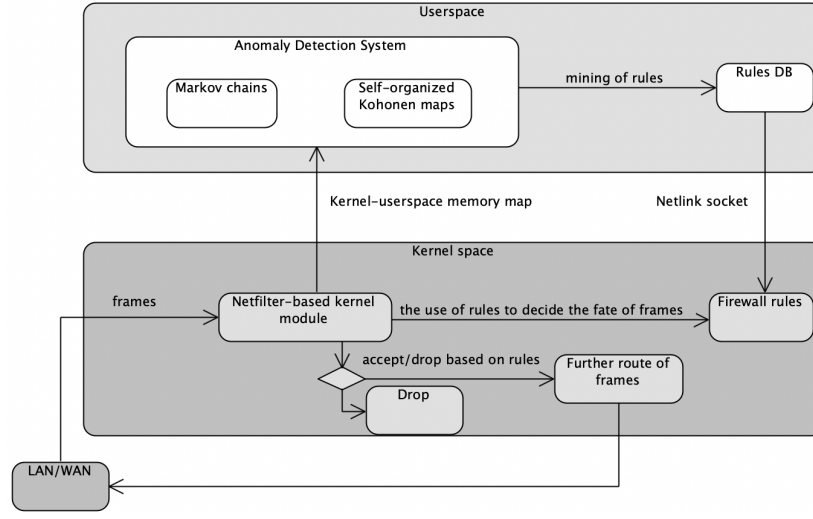


Fig. 2. Proposed architecture

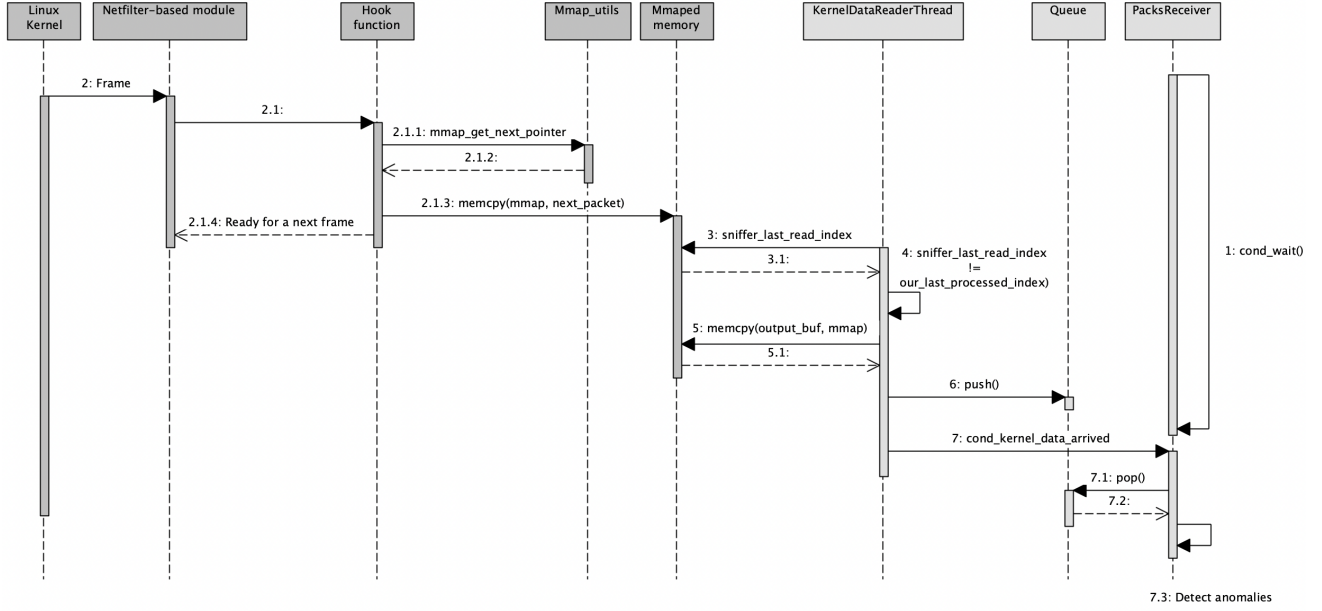


Fig. 3. Transferring and processing packet data

## II. ON IMPLEMENTATION OF OUR SOLUTION

### A. Overall view on the architecture

We propose the architectural diagram of our intelligent firewall system, which is shown in Fig. 2. In the kernel module, traffic packets are intercepted using the Netfilter library extension. Then, data of the network and transport layers is extracted from each frame, which is then compared with the list of firewall rules, and depending on the rule, it is dropped or passed to the destination address. Also, copies of all frames are sent to the userspace application / anomaly detection system (ADS) through a special character device driver (kernel  $\leftrightarrow$  userspace memory map).

### B. Efficient transfer kernel-userspace

Some approaches to fast transfer captured network frames from the kernel to userspace are described in [4]. We use the packet\_mmap technique [5]. The overall scheme of transmission is presented in Fig. 3. In our case, we set up a memory map so that copying the data is done with a simple memcpy. The frame data comes from the Netfilter hook. We organize a ring buffer for such frames, and atomically change the number of frames in it upon arrival. From the userspace side, a reading thread runs, which, when new data appears in the buffer, takes it, changing the number of frames taken. Frames then appear in a special queue, and processing threads are notified for further handling.

### C. Implementation of ADS

In ADS, packets are loaded from the mmap device, then data from the network and transport layers of the OSI model is extracted from each packet. Later, this data is analyzed by currently implemented subsystems: TCP anomaly detection subsystem (Detector-1), based on the variable-order Markov chain model (to work with chains, we use ideas from [6]), and a traffic flow anomaly detection subsystem based on the Kohonen self-organizing map [7] model (Detector-2). These subsystems, in case of detecting suspicious activity, generate new rules for the firewall and send them to the kernel module using a Netlink socket [8].

### D. Implementation of anomaly detection using Markov chains

At initial initialization, Detector-1 builds a probabilistic suffix tree (PST) [9] based on various examples of TCP protocol operation (sequence of connections). Then, according to the sequence of states of each connection, the logarithmic value of the probability of occurrence of this sequence (anomaly score) is calculated according to the suffix tree, after which, when the anomaly threshold is exceeded, a new rule can be mined. The operational scheme is depicted in Fig. 4.

### E. Implementation of anomaly detection using Kohonen maps

The Kohonen layer is trained based on examples of abnormal and normal traffic in the network. In this work, we represent the traffic by a tuple of the following elements: (1) average package size; number of small packages; (2) number of large packages; (3) number of TCP connections; (4) number of UDP packets; (5) number of ICMP packets; (6) number of fragments; (7) number of different IP sources; (8) number of different destination ports; (9) number of inactive TCP connections. During the system operation, traffic flow statistics are collected for some count of packets, after which the abnormality of this flow is determined using the cluster representation from the Kohonen layer. When the specified anomaly threshold is exceeded, a new firewall rule can be mined. Note that the discussed subsystems are adaptive: they are capable of additional training (modes of replenishment of the training sample and the mark of a false alarm).

## III. CONCLUSIONS

Within the framework of this work, we primarily focused on its software feasibility. As a result, we have a working system for demonstrating methods of processing traffic and analyzing it, and this stand is primarily intended for teaching. The tool comprises a system part (the kernel module and some background threads to work with it) and a GUI part, where the user can check the system state and anomaly scores in real-time. Software was checked by Valgrind to the absence of memory leaks by processing torrents downloads. The solution is generalized and not tailored for specific types of port scans as, for example, discussed in a recent work [10]. Some key points of our solution: (1) the use of two discussed detectors makes it possible to analyze both traffic with known states and general traffic flows without explicitly distinguished states; (2)

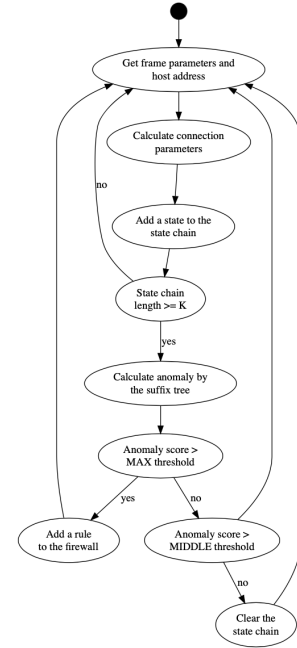


Fig. 4. A simplified scheme of Detector 1 (based on Markov chains)

visualization of PSTs and the behavior of self-organizing maps in the application allows learners to understand the methods of statistical analysis and clustering using real-world networking cases; (3) other detection methods can be elaborated and added later. The solution will be published in [11].

## REFERENCES

- [1] *Netfilter: firewalling, NAT, and packet mangling for Linux*, 2021. [Online]. Available: <https://www.netfilter.org>
- [2] J. D. Day and H. Zimmermann, "The OSI reference model," *Proceedings of the IEEE*, vol. 71, no. 12, pp. 1334–1340, 1983.
- [3] Microsoft, *Windows Filtering Platform Architecture Overview*, 2021. [Online]. Available: <https://docs.microsoft.com/en-us/windows-hardware/drivers/network/windows-filtering-platform-architecture-overview>
- [4] L. Rizzo, "netmap: a novel framework for fast packet I/O," in *21st USENIX Security Symposium*, 2012, pp. 101–112.
- [5] Linux, *packet mmap*, 2021. [Online]. Available: [https://www.kernel.org/doc/html/latest/networking/packet\\_mmap.html](https://www.kernel.org/doc/html/latest/networking/packet_mmap.html)
- [6] N. N. Kussul and A. M. Sokolov, "Adaptive anomaly detection of computer system user's behavior applying Markovian chains with variable memory length," *Journal of Automation and Information Sciences*, vol. 35, no. 6, 2003. [Online]. Available: [https://www.cl.uni-heidelberg.de/~sokolov/pubs/kussul03adaptive\\_part1\\_en.pdf](https://www.cl.uni-heidelberg.de/~sokolov/pubs/kussul03adaptive_part1_en.pdf)
- [7] T. Kohonen, *Self-Organizing Maps*. ISBN: 3-540-67921-9. Springer-Verlag, 2000.
- [8] J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov, *Netlink. RFC 3549*, 2003. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc3549>
- [9] O. Dekel, S. Shalev-Shwartz, and Y. Singer, "The power of selective memory: Self-bounded learning of prediction suffix trees," in *Proceedings of the 17th International Conference on Neural Information Processing Systems*, 2004, pp. 345–352.
- [10] M. Dimolianis, A. Pavlidis, and V. Maglaris, "Syn flood attack detection and mitigation using machine learning traffic classification and programmable data plane filtering," in *2021 24th ICIN Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, 2021, pp. 126–133.
- [11] *Discussed solution*, 2022. [Online]. Available: <https://github.com/SergeyStaroletov/kohmar-firewall>