An Anomaly Detection and Network Filtering System for Linux Based on Kohonen Maps and Variable-order Markov Chains

Sergey Staroletov
Independent Research Enthusiast
Altai territory, Russia
serg_soft@mail.ru

Roman Chudov
Independent Engineer
Altai territory, Russia
roman.chudov@gmail.com

***Abstract*—Modern cyber-physical systems can be defined as distributed systems for processing data from various sensors, while the distribution is provided by a data transmission network. With the complexity of the hardware base, the components of such a system can be executed on minicomputers running the Linux operating system, and solve problems of routing packets and processing them in order to determine software-defined routes. Accordingly, such systems are subject to attacks from outside, which can lead to anomalies in the operation of network subsystems. Therefore, it is necessary to have systems for detecting anomalies in real time, and such tools must be lightweight since the performance of minicomputers is limited. In this paper, we consider a solution for processing network packets at the second OSI level and building detectors based on Markov chains of variable order as well as traffic classification using self-organized Kohonen maps. These solutions are based on well known fundamental works by Russian and Finnish mathematicians and computer scientists, their modern practical applications, so we describe all the used concepts. We present all necessary architectural solutions and algorithms. As a result, we offer a free software solution for Linux as the basis for implementing effective intelligent firewalls. The solution inside is based on a Netfilter hook and packet_mmap.**

I. Introduction

Firewall software can be considered a collection of components located between two networks, that filters traffic between them according to some security policies and rules [1]. The Gartner company defines the concept of *next-generation firewalls (NGFWs)* as deep-packet inspection firewalls that move beyond simple port/protocol examination and blocking to add application-level inspection, intrusion prevention, and bringing intelligence from outside the firewall [2]. Accordingly, today we are usually talking not about separately operating firewalls, but about integral intrusion detection systems. A review on this topic is presented, for example, in [3]. It can be stated that such systems are subdivided into those constructed on the basis of signature analysis and those based on the analysis of network traffic anomalies. In this case, some sharp difference in the behavior of the analyzed system from an expected pattern is considered an anomaly.

From our point of view, the analysis of anomalies could be more promising, since, in addition to simply calculating signatures for traffic, it is also necessary to control its variable nature with respect to acceptable behavior.

Having experience in the development of network information systems for Linux, as well as dealing with the reliability of cyber-physical systems according to their formalized descriptions [4], taking into account that insufficient attention is paid today to the mathematics and internal architecture of solutions for anomaly detection systems, we set ourselves the goal of writing this article.

To access traffic and implement basic firewall functionality, we consider the corresponding Linux subsystem such as Netfilter [5] that provides all the necessary means. However, in order to analyze traffic in real-time, it is necessary to ensure its fast transmission to user space and the use of queues. Such architectural issues need to be modeled on appropriate diagrams before implementation.

For effective analysis of anomalies in traffic, it is necessary to get its representation in the form of a vector or formal system. Today there is a lot of works that use Markov chains for traffic analysis [6]–[8]. In the current work, we follow these approaches and use TCP flags to represent an traffic image, with all the mathematics and implementation issues discussed. However, in order to effectively rebuild adequate models, it is also necessary to apply additional means, in particular, the variational order and probabilistic suffix trees [9], which are discussed in detail in this work. An alternative to considering automata models for processed traffic could be the use of machine learning methods. However, for the sake of greater efficiency, we propose to use a certain set of statistical metrics that can be easily calculated for traffic in a normal environment, and then cluster current captured traffic and calculate a possible anomaly based on this. As for a method of clustering and finding the nearest vector to a given representation, we propose the usage of the concept of self-organizing maps. This bio-inspired formalism was introduced in 1982 [10] and there are still not many works in this area, which makes it interesting to study and apply in a real project. The overall architecture and live demo of discussed software were presented at ICIN-2022 conference [11]. Some initial ideas were registered in a public software register [12].

The present article has the following structure. The writing starts on preliminaries. In Section II, we review all required information on TCP connections. In Section III, we review necessary mathematical methods, in particular, the ordinary Markov chains, variable-order Markov chains, probabilistic suffix trees, and Kohonen self-organized maps. Then we move to the discussion of implementation-related issues. In Section IV, we review the Netfilter framework. In Section V, we describe all the necessary algorithms to make the firewall operational. Finally, we examine some related work in Section VI, turn to the evaluation in Section VII and conclude in Section VIII.

II. Preliminaries in TCP connections

The state of a TCP connection is characterized by a set of flags (SYN, ACK, PSH, FIN, RST, URG) located in the TCP segment header [13]. The process of starting a normal TCP session referred to a "handshake" [14], which consists of three steps in general.

1) A client that intends to establish a connection sends a segment with a sequence number and the SYN flag to a server.
   - The server receives the segment, remembers the sequence number and tries to create a socket (buffers and memory control structures) to serve the new client.
   - If successful, the server sends to the client a segment with a sequence number and the SYN and ACK flags, and enters the SYN-RECEIVED state.
   - If unsuccessful, the server sends a segment with the RST flag to the client.
2) If the client receives a segment with the SYN flag, then it remembers the sequence number and sends the segment with the ACK flag.
   - If it receives the ACK flag at the same time (which it usually does), then it goes into the ESTABLISHED state.
   - If the client receives a segment with the RST flag, then it stops trying to connect.
   - If the client does not receive a response within some seconds, then it repeats the connection process again.
3) If the server in the SYN-RECEIVED state receives a segment with the ACK flag, then it transitions to the ESTABLISHED state. Otherwise, after a timeout, it closes the socket and enters the CLOSED state.

Completing a connection can be considered in three steps:

1) The client sends the FIN and ACK flags to the server to terminate the connection.
2) The server sends the ACK, FIN response flags to the client, indicating that the connection is closed.
3) After receiving these flags, the client closes the connection and sends ACK to the server in confirmation that the connection is closed.

The general connection establishment and disconnection scheme is depicted in Fig. 1.

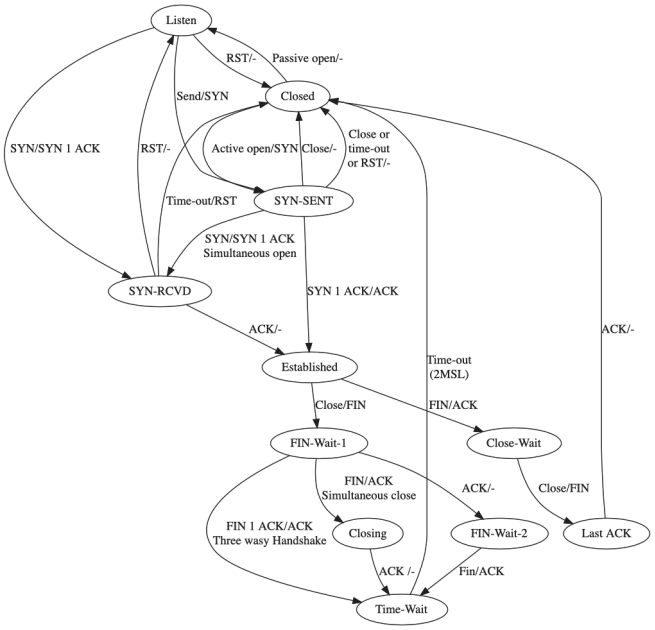

Fig. 1. A TCP connection establishment and termination scheme

Thus, any deviation from the sequence of TCP flag states indicates some abnormal behavior. For example, the so-called "Stealth FIN scan" [15] is a type of TCP scan or traffic forgery. An attacker tries to close a non-existent server connection. This is an uncertain situation, but protocol stack implementations sometimes produce different results depending on whether the service is available. As a result, the attacker can gain access to the system.

Since we are dealing with states and transitions when describing how connections work, we can talk about modeling traffic behavior using probabilistic finite state machines [16]. In this case, a greater modeling ability is achieved than when using trivial frequency methods. Raw data is viewed as a stream of discrete events, such as TCP connection states. The goal is to get an automaton that simulates the specified sequence of events. Such an automaton can be a characteristic of many passed sequences. By the construction, the probability of the next symbol, element or signal depends on some previous elements. However, it often depends on only a small number of the previous ones. This suggests the idea of modeling them using Markov chains. Figures 2 and 3 show examples of such first-order chains for the HTTP and SSH protocols from [17].

We can state that the switching of the TCP flags for different activities differs quite significantly. Therefore, such transition systems and accumulated probabilities can be used as a traffic representation for further assessment of the acceptable or unacceptable state of the network system.

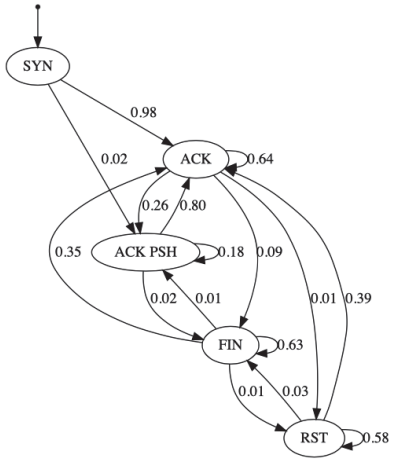

Fig. 2. A Markov chain for HTTP [17]

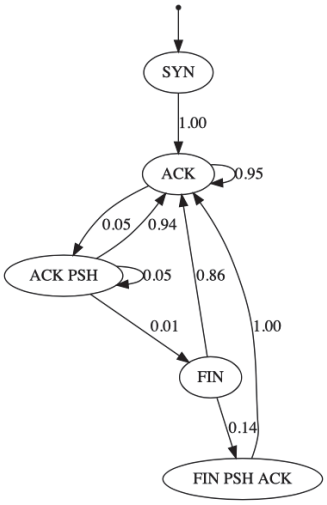

Fig. 3. A Markov chain for SSH [17]

III. Preliminaries in related mathematics

A. Markov chains

1) Ordinary Markov chains: A Markov chain is a sequence of random events with a finite or countable number of outcomes, characterized by the property that, loosely speaking, when we fix the present, the future becomes independent of the past. It is named after the Russian mathematician A.A. Markov [18]. In his book written in old Russian alphabet, he proposed to calculate probabilities between the appearance of letters in a famous poem by A.S. Pushkin using it as big data.

A sequence of discrete random variables $\{X_n\}_{N\geq 0}$ is called a simple discrete-time Markov (or Markovian) chain if

$$P(X_{n+1}=i_{n+1}|X_n=i_n,X_{n-1}=i_{n-1},\ldots,X_0=i_0) = P(X_{n+1}=i_{n+1}|X_n=i_n)$$

Matrix P_{ij} where $P_{ij}\equiv P(X_{n+1}=j|X_n=i)$ is called the matrix of probabilities of system transitions from state i to state j at the n-th step, and the vector $p=p(p_1,p_2,\ldots)^T$, where $p_i\equiv P(X_0=i)$ is called the initial distribution of the Markov chain. The matrix of transition probabilities is stochastic, that is $\sum_j P_{ij}(n)=1,\forall n\in N$ [19].

Thus, in the simplest case, the conditional distribution of the subsequent state of the Markov chain depends only on the current state and independent of all previous states (in contrast to the Markov chains of higher orders). As the chain order increases, the number of states of the corresponding automaton behaves as $O(S^L)$, where S is the size of the alphabet of characters and L is the order of the chain. Such models have proved popular for economic analysis [20].

2) Markov chains of variable order: If we would like to take into account transitions not only just from previous state but from a set of previous states, we should use variable order Markov chains and the concept of probabilistic suffix trees.

Let $A=\{0,\ 1,\ \ldots,\ N-1\}$ be the state space of cardinality $2\leq N<\infty$, $x_1^k=(x_1,\ldots,\ x_k)$, $x_1^k\in A^k$ – a sequence of characters (string) of k elements, $x_i^j=(x_i,x_{i+1},\ldots,x_j)$ – a fragment of string x_1^k with the number of elements $\left|x_i^j\right|=j-i+1$, $1\leq i$, $j\leq k$, $i\leq j$, $(X_t\in A)_{t\in Z}$ – a homogeneous Markov chain of the s-th order with the probability matrix of one-step transitions $P=\left(p_{x_1^s,\,x_{s+1}}\right)$,

$$p_{x_1^s,\,x_{s+1}}=P(X_{t+1}=x_{s+1}|X_t=x_s,\ldots,X_{t+s-1}=x_1)$$

A Markov chain $(X_t)_{t\in Z}$ is called a Markov chain of variable order s if its probabilities of one-step transitions have the form:

$$p_{x_1^s,\,x_{s+1}}=q_{x_{s-l+1}^s,\,x_{s+1}}$$

$$0\leq q_{x_{s-l+1}^s,\,x_{s+1}}\leq 1,\ l=l(x_1^s),\ x_1^{s+1}\in A^{s+1},\ l\in\{0,1,\ldots,s\}$$

$$l(x_1^s)=min\{k:P(X_{t+1}=x_{s+1}|X_t=x_s,\ldots,X_{t+s-1}=x_1)=P(X_{t+1}=x_{s+1}|X_s=x_s,\ldots,X_{t+k-1}=x_{s-k+1})\}$$

The last relation means that the probability of a transition to a state X_{s+1} does not depend on all s previous states, but only on $l(x_1^s)$ states. In addition, the publication [21] defines a context function $c(x_1^s)=x_{s-l+1}^s$ that maps a chain of l significant states or a context to a chain of previous states (denotes that only some values from the infinite history are relevant). $l(.)=|c(.)|$ is the context-length. Other related models may have a set of parameters to describe a conditional order [22].

The context function $c(.)$ and function $l(.)$ can be conveniently represented in the form of a rooted tree, which is called a probabilistic suffix tree (PST) [23] or a context tree [21]. Each node in such a tree can have at most N descendants since each node (except for the root) corresponds to an element from the state space A. Each value of the context function corresponds to a branch of the tree. Note that if each vertex of the context tree that is not a leaf has exactly N descendants, then such a context tree corresponds to a fully connected Markov chain of the s-th order. Such a context tree is called a maximal context tree [21].

To work with Markov chains, it is necessary to determine what the states will be, as well as how they are encoded. The

states of a TCP connection are characterized by a set of flags in the TCP packet header: SYN, ACK, PSH, FIN, RST, URG. From here we can define a set of states: $S = \{S_i\}$, where

$$S_i = SYN + ACK \cdot 2 + PSH \cdot 4 + RST \cdot 8 + URG \cdot 16 + FIN \cdot 32$$

3) PST Construction Algorithm: For a subsequence of characters s, $P(s)$ is the relative number of occurrences of s in this sequence, $P(\sigma|s)$ is the relative number of occurrences of σ in the string after s. That is, if m is the length of the string r, and L is the maximum length of s, then, defining $\chi_j(s)$ as 1, when $r_{j-|s|+1}...r_j = s$ and 0 otherwise, we have [24]:

$$P(s) = \frac{1}{m-L+1} \sum_{j=L}^{m-1} \chi_j(s)$$

$$P(\sigma \mid s) = \frac{\sum_{j=L}^{m-1} \chi_{j+1}(s\sigma)}{\sum_{j=L}^{m-1} \chi_{j+1}(s)}$$

If there are m' strings of length $l \geq L+1$ in total, then

$$P(s) = \frac{1}{m'(l-L+1)} \sum_{i=1}^{m'} \sum_{j=L}^{m-1} \chi_j(s)$$

$$P(s) = \frac{\sum_{i=1}^{m'} \sum_{j=L}^{m-1} \chi_{j+1}(s\sigma)}{\sum_{i=1}^{m'} \sum_{j=L}^{m-1} \chi_j(s)}$$

The operation of the PST construction algorithm depends on the following parameters: L is the maximum length of state labels, n is the upper limit of the number of states. The algorithm starts from a tree with one root ε. Then the following nodes are added to the tree, which, in our opinion, should belong to it. Thus, a node labeled s becomes a leaf of the tree if the empirical probability $P(s)$ is not negligible, and for some symbol s the empirical probability $P(\sigma|s)$ differs significantly from the empirical probability $P(\sigma|suffix(s))$ of getting it after the suffix s, that is, s is the defining context for σ [25]. The algorithm terminates its work if there is no longer a leaf for which the above conditions are true or the limit on the maximum tree depth L is reached.

The algorithm also uses an auxiliary set of values: ε_0, ε_1, ε_2, ε_3, and γ_{min}, they are functions of ε, σ, n, L, and $|s|$.

The algorithm:

1) Initialize T' with one node ε and set S': $S' = \{\sigma \mid \sigma \in S,\ P(\sigma) \geq (1-\varepsilon_1) \cdot \varepsilon_0\}$.
2) Until S' is not empty, execute: select any $s \in S'$ and
   - remove s from S';
   - if there is $\sigma \in S$ such that $P(\sigma|s) \geq (1+\varepsilon_2) \cdot \gamma_{min}$ and at the same time $\frac{P(\sigma|s)}{P(\sigma \mid suffix(s))} \geq 1+3\varepsilon_2$, then add a node labeled s to the tree;
   - if $s < L$, then for each $\sigma' \in S$: add the string σ' to S'.

The peculiarity of this task is the need for adaptive modification of the suffix tree. To provide the model with certain adaptability, an algorithm is proposed for changing the empirical probabilities so that the contribution of the earlier ones decreases with each step. Then later examples will be taken into account with large weights, and it will be possible to simulate the "forgetting" of the earlier examples [26].

It is necessary to carry out this operation for some sequence $q_0...q_N$, modify the probabilities of only those nodes that are descendants of nodes with labels q_i and the tree root on the path of searching for a given string sequence from the tree root. Let us set a certain learning coefficient α. Then the probabilities of tree nodes can be modified as follows:

$$P'(q_i) = P(q_i) + \alpha \cdot P(q_i)$$

where $P'(q_i)$ is the new value of the probability of the node labeled q_i, $P(q_i)$ is the current value of the probability. Next, one should modify the probability for other neighboring nodes using the following formula:

$$P'(s) = P(s) - \frac{\alpha \cdot P(s)}{|\Sigma| - 1}$$

where $P'(s)$ is the new value of the probability of the node with the label s, $P(s)$ is the current value of the probability, $|\Sigma|$ is the cardinality of the alphabet of labels(states).

The calculation of probabilities according to such rules makes it possible to reduce the influence of earlier examples and at the same time, take into account more recent ones with greater weight. The value of the coefficient α controls the speed of forgetting: large values will lead to fast forgetting, while values close to zero will lead to slow forgetting.

4) Calculation of anomalous sequence of TCP connection states: Having built a PST tree for examples of TCP connections (that is, having trained the system), it is easy to use it to determine the probability of a sequence of TCP connection states now in the anomaly detection mode. Having obtained the value of a given probability, and taking its natural logarithm with the opposite sign, we obtain the value of anomaly for a given sequence. Consider an algorithm for finding the probability of generating a certain sequence of TCP connection states.

Let there be some sequence $q_0...q_{N-1}$ of TCP connection states. First, we need to initialize the context c $\{c_0,..,c_{L-1}\}$ with a length equal to the depth L of the PST tree. For the algorithm to work, the index i is also required as the current index in the sequence q, and some index j as the index of the last added state to the context c. *Node* is the current node of the tree (initial value is the root of the tree), p is the transition probability, P is the sought-for probability of generating row q.

The algorithm which is based on [21], [24], [26]:

1) i: = 0, j: = 0, P: = 1;
2) While i $\neq$ N:
   a) if $j \neq L$, then $c_j = q_i$, $j := j+1$; otherwise, shift the entire sequence of states back one position, $c_{L-1} := q_i$;
   b) $p :=$ the probability of going from the *Node* to a child node whose label is c_j;
   c) $P := P * p$;

d) if $Node$ has a descendant of $Node'$ whose label is c_j, then $Node := Node'$, otherwise $Node :=$ the root of the tree;
e) $i := i+1$.

Of course, it is not always possible to know the state of the traffic. For example, current trends in high-load systems are the session rejection and work through UDP [27], so we need to take into account different methods for general passing traffic by classifying it.

B. SOM or Kohonen maps

A self-organizing Kohonen map (SOM) [28], [29] is an unsupervised competitive neural network that performs the tasks of visualization and clustering. The concept of SOM was proposed by a Finnish scientist Teuvo Kohonen as a result of ideas based on the fact that areas in the human brain are found that perform specific functions. All this refers to bio-inspired AI and attempts to find a single algorithm for the functioning of the brain [30]. Neurons in such areas (maps revealed in the EEG process) perform similar actions, which gives reason to think not only about training artificial models of neurons to respond to input signals, but also to take into account the topology of the network, that is, the location of neurons. As a result, Kohonen decided to create his own model, where, in addition to adjusting the weight of the neuron most suitable for the signal, the weights of the nearest (according to the given topology) neurons are adjusted. It can also be said that, mathematically, as a result of network training, the dimension of a large data set with non-linear connections is narrowed down to a dimension that allows describing the topology:

$$SOM : X \times W \to \mathcal{N}(Y)$$

where $X = \{x\}_k$, $x \in \mathbb{R}^n$ is the sequence of k input vectors each of size n, $\mathcal{N}(Y)$ is the topological space for so-called generalized medians [29, p. 107] of the set $Y = \{y\}_{|\mathcal{N}|}$, $y \in \mathbb{R}^n$, W is the set of internal weights of the map of the same cardinality as Y.

Initially, the dimension of the input data is known, according to which the initial version of the map is built in some way. In the process of training, the map approaches the input data and reveals the generalized medians. We can assume that the weights of the networks here are weights in the physical sense, bending the location plane like a hammock and affecting neighboring neurons. The cyclic learning process, which iterates through the input data, ends when the map reaches some admissible error, or after the specified number of iterations has been completed. The training of SOM can be divided into the following stages:

- Initialization of the layer, that is, the initial setting of the vectors of weights for the nodes in the map.
- Loop:
  1) Selecting the next observation (a vector from a set of input data x).
  2) Finding the best matching unit (BMU, or winner) for it as the node on the map whose weight vector is the least different from the observation(according to a metric, most often Euclidean).
  3) Determination of the number of BMU neighbors in $\mathcal{N}$ within the radius R and then training (by changing the weight coefficients in W) the BMU vector and its neighbors in order to bring them closer to the observation.
  4) Modifying the learning rate.
  5) Determination of the error for the SOM.

In this case, to modify the weight coefficients w_i, the following formula is used:

$$w_i(t+1) = w_i(t) + G \cdot (x_i - w_i(t))$$

where t denotes the number of the training epoch, G is the training coefficient (initial value 0.2 - 0.9), x is an image. The coefficients of all neurons, the centers of which inside the circle of radius R, are modified the more, the closer the neuron is to the BMU neuron:

$$G = e^{-\frac{(d_0-d)^2}{\gamma^2}}$$

where d_0 is the distance from the winner neuron to the image, d is the distance between the current neuron with a center inside a circle of radius R and the image, γ is a parameter.

With regard to the problem of classifying traffic flows, an "image" can be a certain set of statistical data characterizing the traffic flow, that is, a certain vector of dimension n. During the work, in real time, we have to calculate some metrics for the traffic and use such metric for the clustering and anomaly detection. So, SOM can be used to obtain the generalized medians of the traffic for further classification (for such classification we do not use the content of the traffic!), and we represent all traffic by vectors of integers $X = \{x\}$, $x \in \mathbb{Z}^{10}$. In this work, we use a tuple

$$X = (S_{avg}, N_{sp}, N_{lp}, N_{tcp}, N_{udp}, N_{icmp}, N_f, N_{src}, N_{dstp}, N_{in})$$

of the following elements selected after series of internal tests:

- S_{avg} – average packet size;
- N_{sp} – number of small packets;
- N_{lp} – number of large packets;
- N_{tcp} – number of TCP connections;
- N_{udp} – number of UDP packets;
- N_{icmp} – number of ICMP packets;
- N_f – number of fragments;
- N_{src} – number of different IP sources;
- N_{dstp} – number of different destination ports;
- N_{in} – number of inactive TCP connections.

As applied to our anomaly detection problem, the input vectors X are also complemented by the target anomaly value $X_a = X \cup A_{level} \in \mathbb{R}$ for a given traffic representation, which is set during training. That is, the analyst decides which traffic is anomalous and which is not and ensures the generation of a set of vectors X_a by, for example, working in the network or running traffic generation scripts. Accordingly, the A_{level} values, which are taken from the generalized medians

$Y_a = Y \cup A_{level}$ after the network has been trained on the input data X_a, will be the most typical anomaly values for the traffic. Having calculated the metrics for any current traffic in real time, we can find the vector $y \in Y$ closest to it, take the corresponding A_{level} value and thus determine the current anomaly value.

To visualize traffic representations (see, for example, the literature [31], [32] for examples of visualizations of Kohonen maps for different domain areas), one can draw a two-dimensional table, each cell of which will be the generalized median of Y_a after the network has been trained. The anomaly value can be used as the color of the corresponding cell.

IV. Preliminaries in Netfilter

Netfilter [5] is one of important frameworks within the Linux kernel that allows advanced users to pass or filter network packets based on their headers and data. For users, rules for handling packets of special interface, direction and protocol can be obtained using the *iptables* command-line tool [33]. For kernel developers, Netfilter offers an API for defining so-called hooks (a code that runs to process some data at some time) in the form of a special kind of C-function of a kernel module. It will be called (see Fig. 4) when each packet passes through the Linux network subsystem according to a given direction (input, output, forwarding), IP protocol (IPv4, IPv6) and priority, which allows developers to insert own handlers both before and after NAT. Such a function has access to the raw data of passing network packets (of MTU size like 1496 bytes) in the form of frames of the second-level of the ISO/OSI model [34], represented by *struct sk_buff* [35]. The WFP framework [36] which was later offered for modern Windows versions, works in a similar way.

```
hook_func_in+0x38/0x50 [ads_netfilter]
nf_hook_slow+0x5c/0xf8
ip_local_deliver+0xf4/0x128
ip_rcv_finish+0x98/0xb8
ip_rcv+0xe0/0xf0
__netif_receive_skb_core+0x698/0xc38
__netif_receive_skb_list_core+0xf0/0x220
netif_receive_skb_list_internal+0x19c/0x2b8
gro_normal_list.part.0+0x28/0x48
napi_complete_done+0xc0/0x200
virtnet_poll+0x364/0x8b0 [virtio_net]
__napi_poll+0x40/0x210
net_rx_action+0x2b0/0x328
__do_softirq+0x138/0x37c
irq_exit+0xf4/0x100
__handle_domain_irq+0x70/0xc8
gic_handle_irq+0x74/0xa8
el1_irq+0xc0/0x148
```

Fig. 4. A call stack for a Netfilter hook

Each such MTU-sized frame is embedded in each other representing a “matryoshka” and containing protocol headers (Fig. 5). The header data can be extracted using the corresponding functions in the form of C structures for the necessary protocols. Inside a lower layer header exists an upper layer header with a protocol number, and using this information, one can further convert the header data to the corresponding structure. Thus, by registering a hook function and analyzing the headers (and possibly data), one can decide what to do next with this frame: to skip it for further processing or drop it at this level.

The approach is demonstrated in the following listing, which is implemented in a separate kernel module (we use Linux kernel 5.2.11):

```
#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) {
 //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 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 an interface
 //of interest
 //for example, input state->in->name

 //obtain an IP header
 struct iphdr *ip_header;
 ip_header = (struct iphdr*)
  skb_network_header(skb);

 //process skb packet data and/
 //or its headers
 //(directly here, or transfer to
 //user space)
 //possible update the classifier later

 //make a decision for a packet(frame)
 //using the previously built classifier
 if (...) {
  //accept this frame
  return NF_ACCEPT;
 } else {
  //or drop this frame
  return NF_DROP;
 }
}
```

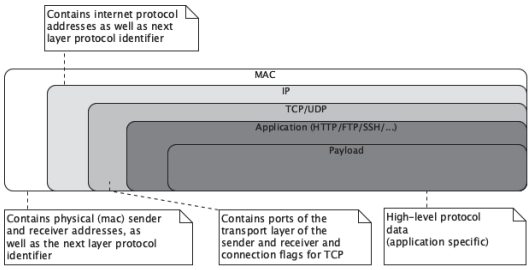

Fig. 5. A frame to process by Netfilter

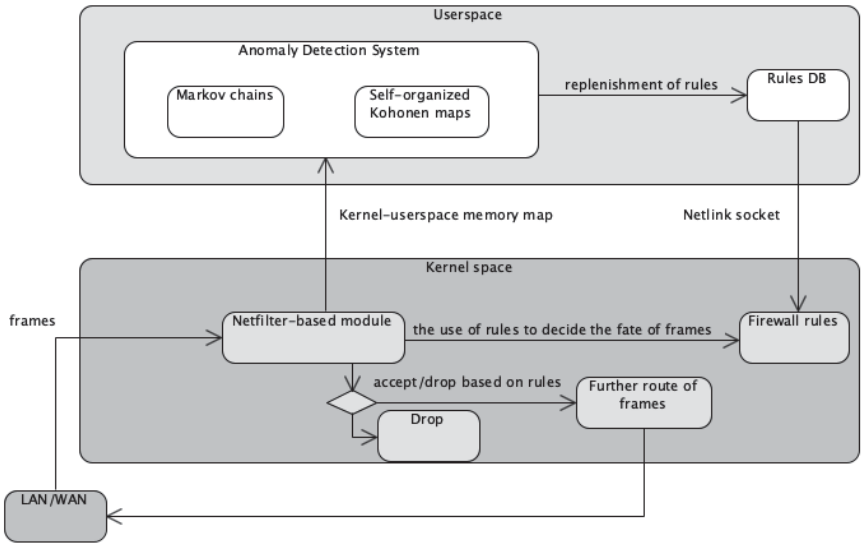

Fig. 6. Proposed architecture [11]

V. Implementation of our solution

A. System architecture

For effective implementation and fulfillment of the set goals, our solution has a ”kernel module ↔ application“ architecture. The kernel module collects interesting network packets and applies firewall rules, while the application in user space is a multi-threaded application that implements the graphical interface, and the logic of training and operation of detectors for packets received from the kernel. This solution is a compromise and it gets rid of potential errors would leading to a crash of the entire system when implementing all logic in kernel space and performance losses if we implemented everything in user space and used *tcpdump* [37] to get traffic and *iptables* for firewall rules. We propose the architectural diagram of the developed system, which is shown in Fig. 6.

In the kernel module, traffic packets are intercepted using the Netfilter library extension implemented as discussed hook functions. Then, header 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 anomaly detection system (ADS) through a special character device driver (kernel ↔ user space memory map). The latter system ”mines“ firewall rules using two detectors, the mathematics behind which was discussed earlier.

B. Efficient transfer kernel-userspace

Some known approaches to fast transfer captured network frames from the kernel space to user space are described in [38]. We follow the packet_mmap technique [39]. The architecture for intercepting and processing the frames of traffic is presented in Fig. 7. In our case, we set up a memory map so that copying the frame data to further process is done with a simple *memcpy* call. A frame comes from the Linux kernel to our kernel module where we have installed the hook function for it. In the function we check the next free space in our ring buffer and we copy the frame content into it. On the other hand, at the level of user space, we have some threads to process new frames from the ring buffer. So, we have a *kernel data reader* thread who checks its last read index and if there are some new frames then it pushes them to a queue and fires a special conditional variable. There is another thread *packet receiver* who checks this variable and pops all new frames and passes them to anomaly detectors. The usage of the ring

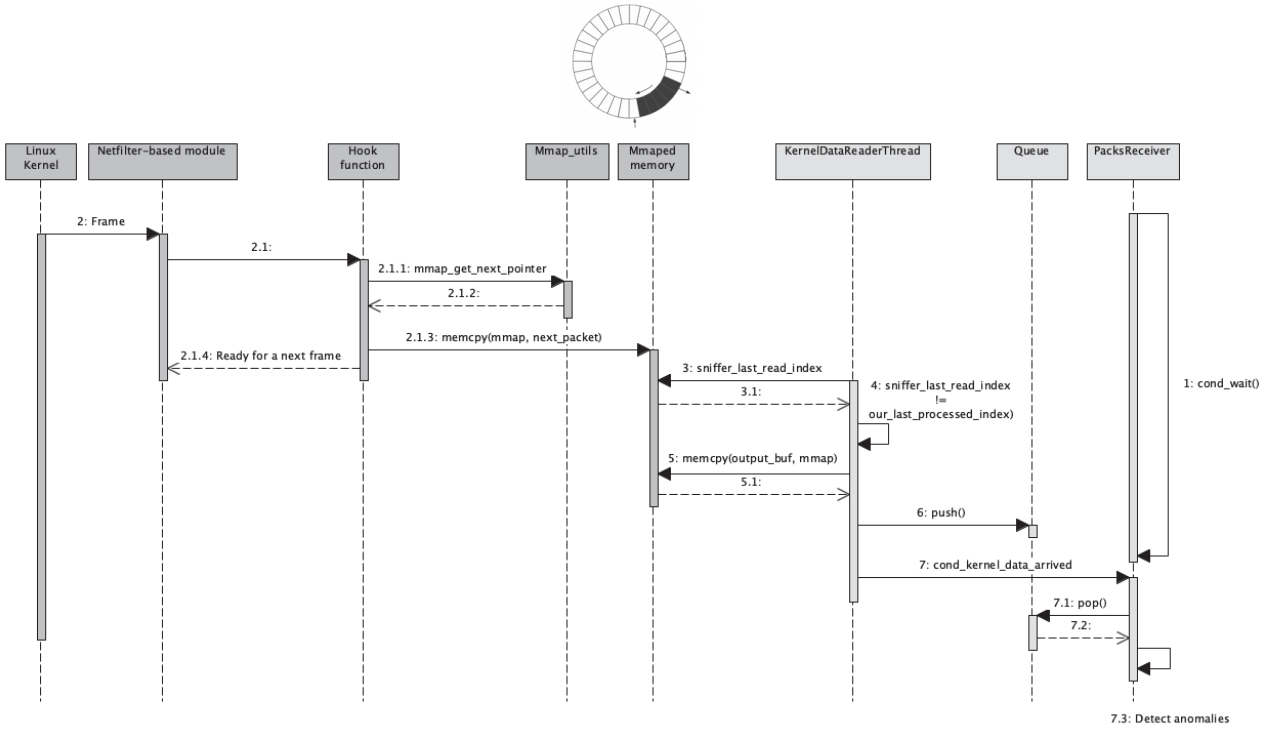

Fig. 7. Transferring and processing packet data [11]

buffer and queues in our architecture guarantees that if the packet receiver was not able to receive new packets, the buffer will not overrun and the hook function will just overwrite old packets into new packets in the buffer.

C. Implementation of the kernel module and ADS

Figure 8 shows a state diagram of the kernel module (can be run at the logical place of the last 'if' in the Netfilter code listing). It should be noted that since we are dealing with packets at the second level of the OSI model, then we are dealing with a large number of frames, but not pure TCP connection data (they are available at the next level). To establish that packets belong to the same TCP connection, we use TCP flags and data on input-output ports and addresses, which allows us to build a table of connections in the rb_tree structure.

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 two subsystems: TCP anomaly detection subsystem, based on the variable-order Markov chain model, and a traffic flow anomaly detection subsystem based on the Kohonen self-organizing map model. 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.

D. Implementation of anomaly detection using Markov chains

At initial initialization, Markov Detector builds a suffix tree based on various examples of TCP protocol operation

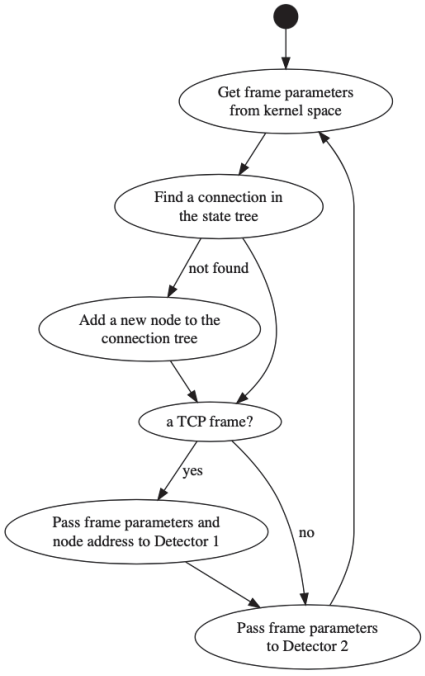

Fig. 8. Operation of the kernel module

(sequence of connections), trained using some user interface. Then, according to the sequence of states of each connection, the anomaly score or logarithmic value of the probability of occurrence of this sequence is calculated according to the suf-

fix tree, after which, when the anomaly threshold is exceeded, a new firewall rule can be generated. The operational scheme is depicted in Fig. 9.

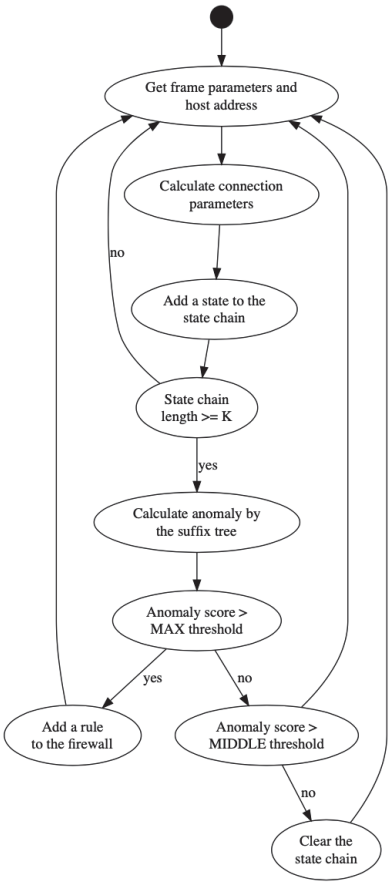

Fig. 9. A simplified scheme of the Markov Detector

E. Implementation of anomaly detection using Kohonen maps

At the initial initialization of Kohonen Detector, the SOM layer is trained based on examples of abnormal and normal traffic in the network. Then, traffic flow statistics are collected for F packets, after which the abnormality of this flow is determined using discussed techniques. When the specified anomaly threshold is exceeded, a new firewall rule can be generated. The operational scheme is depicted in Fig. 10.

Also, these subsystems are adaptive: they are capable of additional training and marking a false alarm.

VI. Related work

Classical intrusion detection systems like Bro [40] and STAT [41] based on signature methods (rules, policies and signatures in the form of transition systems encoded in special languages) have been known for a long time. The newest systems, in turn, use machine learning methods [42], including genetic algorithms and deep learning networks. However, there are some performance issues [43] when using such methods. Therefore, most lightweight systems use, like ours,

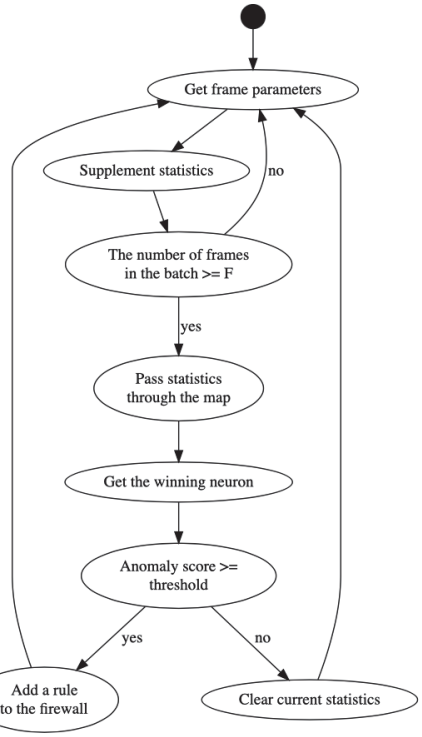

Fig. 10. A simplified scheme of the Kohonen Detector

counting statistics and clustering traffic views. The question remains, which parameters to use for traffic representation and statistics calculation. Prospective studies also consider fractal dimensions for this [44].

A general study on network anomaly detection techniques is published by the Hawkins team [45] and a benchmark for such detectors was proposed [46].

VII. Evaluation

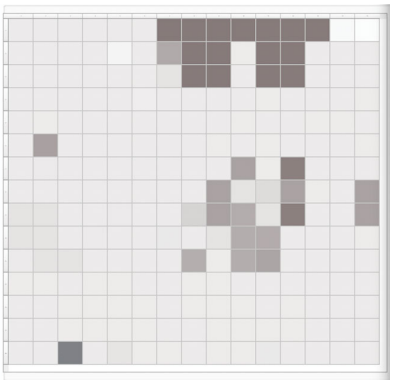

Fig. 11. A Kohonen map visualization build based on real traffic

As a result of the work, demo software was implemented by us in the form of a kernel module and a QT desktop application for identifying the detection state in real time. Here we comprise a graph of anomaly changes over time, anomaly logging and visualization of the Kohonen map. Let

us dwell on it in more detail (Fig. 11). In this figure, we see its representation in the form of a two-dimensional matrix, where each cell corresponds to some resulting blurring of the input vectors after network training (by clicking on the cell, the user can also see the state of this vector). The darker the cell, the greater the value of the corresponding target anomaly value for these values. Such values also appear as blurring of the target values given during network training (Fig. 12).

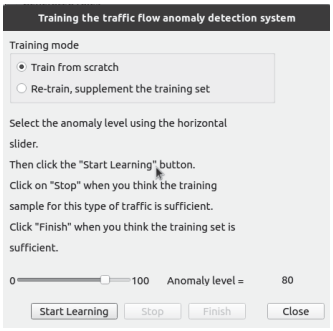

Fig. 12. The user should specify the desired anomaly level for the current traffic during training

The user turns on the learning mode, sets the expected value of the anomaly and starts generating traffic in real time (programmatically or working on the Internet using a browser and other software). At the same time, our software calculates secondary characteristics for it and generates a vector representation of this traffic, and then saves it to the database along with a given anomaly value. It is recommended to further review these records and correct them by removing irrelevant ones (recorded during pauses, for example). After recording all possible modes of interest and training the network according to Section III, we get a network with its visualization presented. Further, when the anomaly detection system is running in real time, the program calculates the characteristics of the current traffic, searches for the nearest vector, and obtains the anomaly value. In this case, in Fig. 11, one can see the square in the bottom row, which shows the current vector at the moment (it is constantly moving). Thus, we can observe how close the current traffic is to anomalous.

Note, the size of the Kohonen map obviously depends on the number of different modes of operation being recorded, since trying to fit too different vectors into a small field will result in all of them blurring and detection will suffer.

We also consider issues related to TCP connections and protocols on which Markov chains are built. Fig. 13 shows a graph of the system when surfing classic websites, as one can see, there are no false positive detections. Everything changes when we start visiting modern sites with a lot of JavaScript code and connections from it when scrolling the page, dynamic menus and so on. Here, false positive anomalies begin to appear (Fig. 14). However, when running port scan scripts, anomalies will be detected correctly and tangibly clearly (Fig. 15).

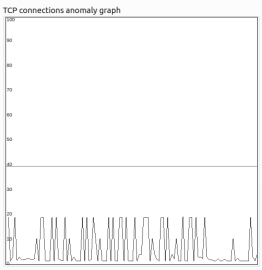

Fig. 13. Work of the Markov-based detector during normal Internet surfing

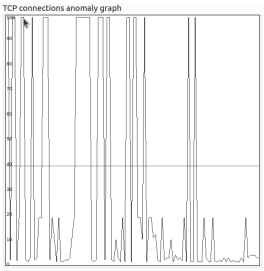

Fig. 14. False positives of the Markov-based detector during modern web applications

VIII. Conclusion

As a result of this study, we presented a software architecture for building an intelligent firewall that analyzes anomalies in network traffic. 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, using the presented mathematical methods on real examples with big data. This stand is primarily intended for teaching. Nevertheless, there are proven good results of its work on the generated raw-packets of network traffic using the available port scanning algorithms in our local network.

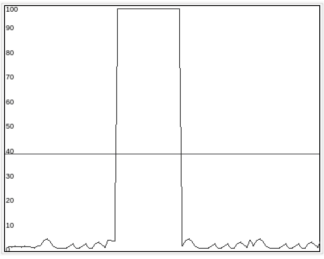

Fig. 15. Real-time identification of port scans

We posted all the current source codes on GitHub [47] in the hope that this will help readers in the initial steps to make a more advanced solution. The code is compilable for x86/ARM and could be run on Raspberry Pi. Currently, we are interested in formal models of such things, as well as the implementation of detectors based on modern approaches, and it would be especially interesting to try custom models of brain-inspired detectors like those we discussed in the work [48].

REFERENCES

[1] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith, "Implementing a distributed firewall," in *Proceedings of the ACM conference on Computer and communications security*, 2000, pp. 190–199.

[2] Gartner, *Next-generation Firewalls (NGFWs)*. [Online]. Available: https://www.gartner.com/en/information-technology/glossary/next-generation-firewalls-ngfws

[3] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, pp. 1–22, 2019.

[4] S. Staroletov, N. Shilov, V. Zyubin, T. Liakh, A. Rozov, I. Konyukhov, I. Shilov, T. Baar, and H. Schulte, "Model-driven methods to design of reliable multiagent cyber-physical systems," in *CEUR Workshop Proceedings*, vol. 2478, 2019, pp. 74–91.

[5] *Netfilter: firewalling, NAT, and packet mangling for Linux*, 2021. [Online]. Available: https://www.netfilter.org

[6] G. Münz, H. Dai, L. Braun, and G. Carle, "TCP traffic classification using Markov models," in *International Workshop on Traffic Monitoring and Analysis*. Springer, 2010, pp. 127–140.

[7] O. Kravets, "Mathematical modeling of parameterized TCP protocol." *Automation & Remote Control*, vol. 74, no. 7, 2013.

[8] G. Aceto, G. Bovenzi, D. Ciuonzo, A. Montieri, V. Persico, and A. Pescapé, "Characterization and prediction of mobile-app traffic using Markov modeling," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 907–925, 2021.

[9] J. Gustafsson, P. Norberg, J. R. Qvick-Wester, and A. Schliep, "Fast parallel construction of variable-length Markov chains," *BMC bioinformatics*, vol. 22, no. 1, pp. 1–23, 2021.

[10] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[11] S. Staroletov, "Software architecture for an intelligent firewall based on Linux Netfilter," in *2022 25th Conference on Innovation in Clouds, Internet and Networks (ICIN)*. IEEE, 2022, pp. 160–162.

[12] R. Chudov and S. Staroletov, "Intellectual packet analyzer and blocker," Patent RU 2015613681, 2015. [Online]. Available: https://www.elibrary.ru/item.asp?id=39329583

[13] F. van Kempen. [Online]. Available: https://elixir.bootlin.com/linux/latest/source/include/uapi/linux/tcp.h

[14] "Internet protocol—DARPA internet program protocol specification, rfc 791," 1981. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc793

[15] M. De Vivo, E. Carrasco, G. Isern, and G. O. De Vivo, "A review of port scanning techniques," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 2, pp. 41–48, 1999.

[16] E. Vidal, F. Thollard, C. De La Higuera, F. Casacuberta, and R. C. Carrasco, "Probabilistic finite-state machines-part I," *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, no. 7, pp. 1013–1025, 2005.

[17] N. Ye *et al.*, "A Markov chain model of temporal behavior for anomaly detection," in *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, vol. 166. Citeseer, 2000, p. 169.

[18] A. A. Markov, *Calculus of Probabilities: With a Portrait of Jacob Bernoulli (in Russian)*, 3rd ed. Printing house. Imp. Acad. sciences, 1913. [Online]. Available: https://search.rsl.ru/ru/record/01003819057

[19] J. Chang, "Stochastic processes. Lecture notes," 2007. [Online]. Available: http://www.stat.yale.edu/~pollard/Courses/251.spring2013/Handouts/Chang-notes.pdf

[20] N. Marynenko, I. Fedyshyn, N. Garmatiy, and I. Kramar, "Financing innovation activity in Ukraine: realities and perspectives," 2019.

[21] P. Bühlmann and A. J. Wyner, "Variable length Markov chains," *The Annals of Statistics*, vol. 27, no. 2, pp. 480–513, 1999.

[22] M. Maltsew and Y. Kharin, "Identification of markov chains of conditional order," 2012.

[23] 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.

[24] 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

[25] A. M. Sokolov, "An adaptive detection of anomalies in user's behavior," in *Proceedings of the International Joint Conference on Neural Networks, 2003.*, vol. 4. IEEE, 2003, pp. 2443–2447.

[26] D. Ron, Y. Singer, and N. Tishby, "The power of amnesia: Learning probabilistic automata with variable memory length," *Machine learning*, vol. 25, no. 2, pp. 117–149, 1996.

[27] A. Tobol, *How we abandoned JPEG, JSON, TCP and doubled the speed of VKontakte (in Russian)*, 2022. [Online]. Available: https://habr.com/ru/company/vk/blog/594633/

[28] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[29] ——, *Self-Organizing Maps*. Springer-Verlag, 2001. [Online]. Available: https://doi.org/10.1007/978-3-642-56927-2

[30] V. B. Mountcastle, "The columnar organization of the neocortex." *Brain: a journal of neurology*, vol. 120, no. 4, pp. 701–722, 1997.

[31] A. O. Prokofiev, A. V. Chirkin, and E. D. Smirnova, "A method of cryptostability analysis of stochastic transformations, based on the Kohonen self-organizing maps," in *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*. IEEE, 2018, pp. 351–354.

[32] C. M. Intisar and Y. Watanobe, "Classification of online judge programmers based on rule extraction from self organizing feature map," in *2018 9th International Conference on Awareness Science and Technology (iCAST)*. IEEE, 2018, pp. 313–318.

[33] M. Rash, *Linux Firewalls: Attack Detection and Response with iptables, psad, and fwsnort*. No Starch Press, 2007.

[34] J. D. Day and H. Zimmermann, "The OSI reference model," *Proceedings of the IEEE*, vol. 71, no. 12, pp. 1334–1340, 1983.

[35] The kernel development community, *struct sk_buff*. [Online]. Available: https://docs.kernel.org/networking/skbuff.html

[36] Microsoft, *Windows Filtering Platform Architecture Overview*, 2021. [Online]. Available: https://docs.microsoft.com/en-us/windows-hardware/drivers/network/windows-filtering-platform-architecture-overview

[37] The Tcpdump Group, *tcpdump, a powerful command-line packet analyzer*, 2022. [Online]. Available: https://www.tcpdump.org

[38] L. Rizzo, "netmap: a novel framework for fast packet I/O," in *21st USENIX Security Symposium*, 2012, pp. 101–112.

[39] Linux, *packet˙mmap*, 2021. [Online]. Available: https://www.kernel.org/doc/html/latest/networking/packet_mmap.html

[40] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[41] G. Vigna, S. T. Eckmann, and R. A. Kemmerer, "The stat tool suite," in *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, vol. 2. IEEE, 2000, pp. 46–55.

[42] Z. Chiba, N. Abghour, K. Moussaid, M. Rida *et al.*, "Intelligent approach to build a deep neural network based IDS for cloud environment using combination of machine learning algorithms," *computers & security*, vol. 86, pp. 291–317, 2019.

[43] T. Saranya, S. Sridevi, C. Deisy, T. D. Chung, and M. A. Khan, "Performance analysis of machine learning algorithms in intrusion detection system: A review," *Procedia Computer Science*, vol. 171, pp. 1251–1260, 2020.

[44] O. Sheluhin and M. Kazhemskiy, "Influence of fractal dimension statistical charachteristics on quality of network attacks binary classification," in *2021 28th Conference of Open Innovations Association (FRUCT)*. IEEE, 2021, pp. 407–413.

[45] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, 2016.

[46] A. Lavin and S. Ahmad, "Evaluating real-time anomaly detection algorithms–the Numenta anomaly benchmark," in *2015 IEEE 14th international conference on machine learning and applications (ICMLA)*. IEEE, 2015, pp. 38–44.

[47] *Discussed solution*, 2022. [Online]. Available: https://github.com/SergeyStaroletov/kohmar-firewall

[48] S. Staroletov, "A hierarchical temporal memory model in the sense of Hawkins," in *2021 IEEE Ural-Siberian Conference on Computational Technologies in Cognitive Science, Genomics and Biomedicine (CSGB)*. IEEE, 2021, pp. 470–475.