Towards Accelerating Intrusion Detection Operations at the Edge Network using FPGAs

[1]Yacine Rebahi, [1]Faruk Catal, [1]Nikolay Tcholtchev, [1]Laurenz Maedje, [1]Omar Alkhateeb, [1]Vinoth Kumar Elangovan,
[2]Dimitris Apostolakis
[1]Fraunhofer Institute for Open Communication Systems (FOKUS)
Kaiserin-Augusta Alee 31, 10598, Berlin, Germany
{firstname, lastname}@fokus.fraunhofer.de
[2]Future Intelligence, 207 Regent Street, W1B 3HH, London, UK
dapostolakis@f-in.co.uk

***Abstract*— In the current paper, we present our work towards accelerating intrusion detection operations at the edge network using FPGAs. Cloud computing and network function virtualization have led to a new appealing paradigm for service delivery and management. Unfortunately, this paradigm fails to correctly support IoT applications and services that seek better communication platforms. Security as a Service can also be seen as a cloud-based model that needs to be accommodated to fulfill these services requirements. Again, one of the main issues to be addressed in this context is how to improve the performance of such systems or services in order to make them capable of coping with the huge amount of data while remaining reliable. A potential solution is the FPGA based edge computing, which is a powerful combination offering FPGA acceleration capabilities together with edge and fog benefits. Indeed, our work focusses on devising an Intrusion Prevention architecture called FORTISEC (40SEC), that is meant to operate in a completely softwarized as well as in an FPGA accelerated mode. Thereby, we present suitable algorithms, design methodologies and well defined components towards the implementation of accelerated intrusion prevention on the edge. It is worth to mention that although 40SEC is discussed here in the context of edge computing, it can serve as a security solution for any Small and Medium Enterprise looking for full protection of its network at a reasonable price. We also present a testbed being utilized for the implementation of 40SEC and its performance testing.**

Keywords—Security, IDS, IPS, SECaaS, Edge, Fog, FPGA, netfilter, iptables, nftables, SME, FORTIKA

I. INTRODUCTION

Virtual Network Functions are network services or capabilities that are softwarized and can also run on commodity hardware. Contrary to functions running on dedicated hardware, virtual functions offer flexibility and easy deployment. The services that can be softwarized include firewalls, Domain Name System (DNS), Network Address Translation (NAT), and Intrusion Detection/Prevention Systems (IPS/IDS) [23].

It can be fairly said that virtualization enabled moving a huge number of services and applications to the cloud. However, the related performance was not always as expected. This is the case for instance for Internet of Things (IoT) [23]. The latter is being used in a variety of systems and services in our daily life. The quality of these services certainly depends on the security solutions put in place in order to protect them. Utilizing Intrusion Detection Systems (IDS) in the cloud, in order to secure IoT services might not be the appropriate way as with the huge amount of data to be analyzed, the response time might face delays and the alerts could be received after the attack has taken place.

Edge computing is a paradigm that particularly came to optimize network bottlenecks that can be faced when using clouds. The idea behind this concept is to perform data processing and knowledge generation at the periphery of the network and close to the originating sources. If we wish to develop an IDS for protecting IoT services, we need to ensure that: (1) it is lightweight enough to be deployed at the edge, and (2) the performance of the detection operations will be better than if the detection is made in the cloud.

In this paper, we discuss our research towards specifying and prototyping an Intrusion Prevention System (IPS) that can be used in Edge computing scenarios. Our solution, called FORTISEC (or simply 40SEC), resembles to a Virtual Security Appliance (vSA) that is composed of a firewall and an IDS. Although, the focus in this paper is on edge computing, our solution can be utilized in a variety of other scenarios (e.g. enterprise security). One of the main characteristics of our vSA is the fact that the parts of the IDS that need intensive processing are implemented on Field Programmable Gate Arrays (FPGAs) that are known for their capabilities of accelerating compute intensive workloads [25]. It was mentioned for instance in [25] that convolutional Neural Networks (CNN) can be improved for image classification on AlexNet up to 2,0/2,5 times in terms of processing time – i.e. when utilizing FPGAs in comparison to CPUs.

The rest of this paper is structured as follows: Sections two and three discuss the state of the art related to the implementation of IDS particularly on accelerators. Section four introduces the 40SEC system. Sections five and six describe the ARM based architecture and the FPGA based architecture respectively. Section seven gives an overview of the tests and the related results undertaken until now, and section eight concludes the paper and outlines future activities.

II. Related Work and Progress Beyond States-of-the-art

Implementing Intrusion Detection Systems on FPGA is a topic that has been more or less discussed in literature. The topic was approached from different angles, since existing algorithms such as Wu-Manber [39] and Aho-Corasik [40], which could be efficient in accelerating rule-matching operations, are applicable to strings. However, the rule sets that intrusion detection systems like Snort use are based on regular expressions (regex). *An ideal situation could have been the utilization of the above-mentioned fast algorithms with the rule sets offered by the existing IDS*. Unfortunately, this option is not easy to realize. Snort, for instance, provides already a huge variety of rules where most of them are written in Perl Compatible Regular Expressions (PCRE) [44], and which cannot be integrate with the above high speed string-matching approaches in a simple way.

The authors of [36] proposed a solution integrating network interface hardware and packet analysis hardware into a single FPGA chip. To realize this, they implemented a complete and functional network IDS on a Xilinx Virtex II/Pro FPGA that performs packet filtering on multiple Gigabit Ethernet links using Snort rules [36]. In [38], an FPGA based high performance pattern matching architecture was implemented. The authors claim that the developed deep network packet filter can protect a network of 1.6 Gbps and their design can fit on very cheap FPGA hardware.

As the part in the IDS that requires intensive processing is the string matching, several solutions implementing the related operations on FPGA were suggested. For instance, the authors of [37] presented a module utilizing non-deterministic finite automata to create efficient circuits for matching patterns defined through a standard rule language. In order to have complete IDS, the mentioned module was integrated with other hardware and software components. The authors of [37] claim that their technique led to circuits that are more than twice as dense as other proposed designs, however the throughput necessary for processing at gigabit line speeds and even beyond was maintained. The IDS software versions like Snort use some rule matching algorithms (such as the above-mentioned Wu Manber [39] and Aho-Corasik [40]), which can be implemented on hardware in order to accelerate the detection operations [41]. These algorithms need to be redesigned in order to use regular expressions.

In [42], a hardware based regular expression engine for Snort was built by transforming the PCRE opcodes generated by the PCRE compiler from Snort regular expression rules. The hardware implementation was tuned by using Nondeterministic Finite Automaton (NFA) and greedy quantifiers. Two hundred PCRE engines were implemented and tested on a Virtex-4 LX200 FPGA. The authors claim that this implementation performs much better (up to 353 times faster) than software based PCRE execution [42].

Sidhu and Prasanna [43] investigated the acceleration of grep regular expression searches with FPGAs by utilizing a compilation technique that quickly converts a regular expression into an FPGA circuit. The regular expression is compiled into a Nondeterministic Finite Automata (NFA) that is directly implemented on the FPGA hardware. Another way for handling regular expression operators, based on standard regex syntax is to use JHDL [45], which is a complete Java based design environment. Indeed, one can write complex circuit generation algorithms in Java and combine them with JHDL circuit libraries in order to create sophisticated module generators [46]. Unfortunately, this tool is not easy to use and cumbersome to learn and involve unexperienced engineers.

Our approach differs from the above-mentioned work in the following aspects,

- Both signature and anomaly based detection techniques are considered in our design
- Our implementation considers pattern matching algorithms that are already used by well know intrusion detection solutions (e.g, Snort)
- "high-level" programming languages like *C/C++* have been used for implementing the pattern matching algorithms on Xilinx Vivado[1]. This makes the update of the code easier. Vivado also enables the generation of bitstreams that can be uploaded on the FPGA hardware

III. Intrusion Detection Systems

Intrusion Detection Systems (IDS) are devices or software applications used to monitor the network for abnormal and malicious activities. Although the attacks that can be detected by IDS are of various types, they can be generally classified as follows [1],

Probing: This activity systematically scans a given network in order to collect useful information or known vulnerabilities. Once the topology of the network is known, this activity could be followed by a severe attack. Port scan is an example of such attacks.

Remote-to-Local (R2L): Here, the attacker gains unauthorized access to a certain machine via a remote connection. Examples of such attacks include buffer overflow, network worms, and Trojan programs.

User-to-Root (U2R): By carrying out this attack, it is intended to achieve higher access privileges in a victim machine starting from a normal access. An example of such attack [2] is the use of *stack smashing*, which feeds a packet to a set-UID-to-root program that corrupts its address space so that a *return from subroutine* instruction results in the spawning of a setUID-to-root command shell.

Denial of Service (DoS): The goal of a DoS attacks is to make a network resource unavailable for legitimate users. One way to trigger the crash is to flood the system with a huge amount of useless traffic. The other way for carrying out the attack is to exploit software bugs in the system under consideration and use them to reduce its performance or crash it. This method is usually reflected by sending malformed messages, which will be incorrectly handled by the targeted

[1] https://www.xilinx.com/products/design-tools/vivado.html

resource. Examples of DoS attacks [1] include *ICMP flood*, *SYN flood*, *Ping of Death (PoD)*, and *Teardrop* attack.

The current security market offers a variety of IDS that can be categorized according to the following factors,

Location: Here, we need to distinguish between host based and network based IDS. The first subcategory collects data from computer internal sources (e.g. operating system logs) and monitors system programs execution and user activities. A network based IDS monitors user activities on the network by collecting network packets and analyzing the related traffic.

Functionality: Here, we need to distinguish between an Intrusion Detection System (IDS) and an Intrusion Prevention System (IPS). Contrary to the IDS that performs automatic intrusion detection activities, the IPS also manages responsive actions. Usually an IPS is an IDS enhanced with preventive functionalities such as firewalling, vulnerability assessment, and anti-virusing.

Deployment: This factor simply means that an IDS can be installed in one host or several ones. In the second case, the detection is performed in a distributed manner and the results will be coordinated by the IDS central management system.

Detection model: In the detection process, if the attack pattern is available, each packet from the network traffic is matched against the attack pattern. This way of inspecting the network traffic is referred to as signature-based detection. Unfortunately, it only detects known attacks. Anomaly detection simply builds profiles for normal user/network behavior, which will be compared to the actual behavior of the user or the network. The mentioned profiles can be defined by the administrator or through a training phase (based on statistical and/or machine learning techniques) of the IDS. Anomaly detection assumes that any abnormal behavior indicates an intrusion. This is in general incorrect and could lead to a significant number of false alarms.

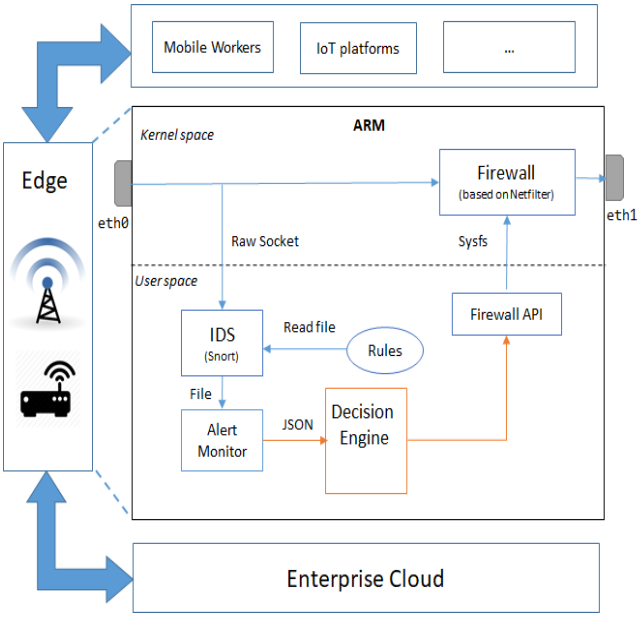

Figure 1: ARM based Architecture

Intrusion Detection Systems do not exist only as commercial solutions, some IDS open source software packages are also available on the market and offer comparable levels of security. Snort [3] and Suricata [4] are among these open source solutions and implementing mostly signature based intrusion detection.

IV. The 40SEC System

The edge gateway platform, on which our virtual Security Appliance (40SEC) run, is a XILINX KRM-3Z7030 50mm x 70mm module [34] composed of an ARM processor and an FPGA part, and carried by a KRC3701 Carrier Kit [35].

It is worth mentioning that virtualizing security appliances has to deal intrinsically with performance restrictions. In the past, optimal performance was provided through a dedicated hardware. In virtualized environments, applications and services running on an operating system compete for the same hardware computing resource, which might slow down the performance. One of the main objectives of this work is to investigate how the performance behaves when the 40SEC entirely runs on the ARM processor and when some of the related security parts - that require more processing - are moved to FPGA. For this reason, we decided to have the 40SEC architecture in two different versions. In the first one, the entire 40SEC appliance runs on the ARM processor. However, in the second version, the parts that require intense processing are moved to the FPGA part, in order to speed up the detection operations.

V. The ARM based architecture

Figure 1 depicts the main components of the ARM based 40SEC architecture as well as the interfaces in between. As previously mentioned, the entire 40SEC appliance is implemented on the ARM processor in the first version. For that purpose, various related components were dockerized and made available. Since some existing Intrusion Detection Systems - such as Snort - are available as open source and offer a security level comparable to commercial security solutions, we decided to explore their capability of running on the ARM processor and measure their performance. Within the 40SEC architecture, Snort was utilized due to its popularity. Based on these high-level elucidations, the following subsections discuss on vital aspects and processes within the 40SEC architecture depicted in Figure 1.

A. Packet Capturing

Packet Capturing makes all incoming network packets available for further processing. The easiest and typical way to receive all packets in Linux is to open a *raw socket*. Such a socket will pass received Ethernet frames directly to the user space application without any preprocessing (thus all network headers are incorporated into the data). In Python, such a socket can be created with the socket module and a call to

socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(3))

In this connection call, the first argument indicates the address family (Packet, Internet Protocol 4, etc.), the second argument describes the socket type (raw or datagram), while the third one indicates that packets from all protocols will be received. Received calls on the socket will then return single packets, where the headers of the nested protocols must be parsed manually to extract source, destination and other metadata.

B. Packet Filtering

In the following, the process of packet filtering is explained along the legacy *iptables* technology. *iptables* [30] is a program for the configuration of the Linux kernel firewall. It is based on rules that are organized in chains. The rules are managed internally and can be added, modified and deleted through a user space application. To update the firewall rules programmatically, one can dynamically build up strings and configure them from a user space application via operating system calls (e.g. with the function *os.system* in Python).

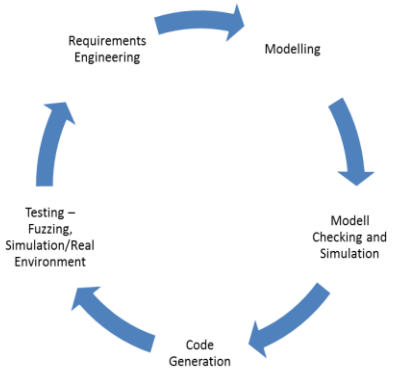

Figure 2: Development Methodology for accelerated FPGA Firewalls

A different approach to filtering (but tightly connected to *iptables*) was implemented within *netfilter*. This Linux kernel interface allows one to create hook functions that will be called on every incoming packet. Such a function can then decide to accept or reject the packet. This gives direct control over the firewall functionality, in contrast to the configuration-based approach of *iptables*. Because the rules' data is stored in the module and can be modified by program code (in the case of *netfilter*), it is much easier to update the rule set in a consistent way - e.g. all rules matching certain criteria can be deleted easily or multiple rules can be merged into a single one with a corresponding IP range. Adding complex functionality is therefore more straightforward than keeping track of all the rules of *iptables* and updating them accordingly. Because the hook functions are inserted directly into the kernel network stack, a *netfilter*-based firewall has to be often developed as a kernel module in the C programming language. Thus, with the greater flexibility also comes greater responsibility, as a crash of the firewall as a kernel module will bring down the entire system. Additionally, a communication channel between the firewall and user space has to be established to enable a REST service for configuration. Indeed, a *sysfs* interface is used for this purpose. Commands are inserted into a file in a virtual file system that is being listened to by the kernel module.

C. Firewall Model Checking

Firewalls are a critical part in any security framework. Unfortunately, because a lot of rules' configuration work is done manually by the network administrators, misconfigurations are very common and can affect the reliability of the firewall. Identifying such anomalies is a challenging task, and we propose in this paper a tree based verification model to deal with this issue in the scope of the 40SEC architecture.

Figure 2 depicts the methodology for verifying and developing IDS/firewall rules in the scope of 40SEC.The process starts with a requirements engineering phase where various requirements are consolidated and further transformed to a decision tree (or state automate) representation, in order to enable the verification of the firewall logic. Finally, the tree is used to generate *nftables/iptables* rules for the *netfilter* module. Next, the steps in Figure 2 are described in detail as they constitute the formal framework for realizing various defense mechanisms within 40SEC.

1) Requirements Engineering

Requirements engineering is one of the cornerstones for successful systems' design. In this phase, the firewall for a product is described with respect to its desired functionality. In general, requirements can originate from different sources - e.g. functional specifications, security targets/profiles of national cyber-security agencies, vulnerability databases [26], or results from hacking contests. Thereby, it is important to capture the requirements in a consolidated form, which can be either given as a table representation or as a more structured format such as ReqIF [27]. Suitable tools in this phase are given by DOORS [28] or the Eclipse based ProR [29]. Thereby, it is of paramount importance to properly manage the requirements, i.e. utilize a suitable meta-model that enables the traceability of the requirements across the development process.

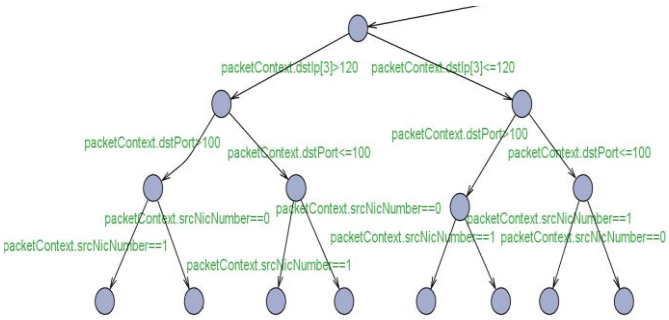

Figure 3: Decision Tree Model for a Firewall Decision

2) Modelling and Model Checking

Next, we discuss on the Modelling and Model Checking aspects outlined in Figure 2. The creation and management of firewall and intrusion detection rules can be a cumbersome task given the complexity of possible requirements and aspects that need to be considered. Indeed, even the simple blocking of packets based on specific properties is impossible

to verify in general through means of testing, provided the extreme size of the possible input-output spaces. Furthermore, industry and research have identified that the straightforward listing of firewall rules leads to poor traceability between requirements and implementation, poor maintenance, and is difficult to explain and present in front of customers and certification authorities. In addition, the linear listing of rules and decision logic bears the pitfall of not addressing implicit dependencies between requirements ([31],[32]).

Based on the above considerations, we apply decision trees and state machines in the course of designing the 40SEC firewall. For that purpose, various tools can be applied. Figure 3 depicts a part - i.e. sub-tree - of a larger decision tree applied for managing, modeling and simulating firewall/IDS rules based on the means provided by the Uppaal [33] model checker. Thereby, a packet is looked at in the light of a context that is represented as a tuple of various variables and characteristics, which are in place at the moment of the packet entering the edge node. Based on these characteristics, the decision tree is traversed until a decision is reached on whether to drop the packet or forward it according to the underlying routing rules. Furthermore, a mapping between the specific requirements for the device in question and the paths through the decision tree is established, in order to ensure traceability between requirements, model and simulation.

Based on the model checking capabilities of the belonging tools, various simulation and verification checks are possible. For example, packet contexts (i.e. tuples of characteristics) can be constructed and their processing within the 40SEC-IDS/firewall-model can be validated.

3) Code Generation

Based on the model presented above, it is possible to develop code generators from the abstract decision tree form to various firewall representation languages. Hence, it is possible to generate *nftables*, *iptables*, and even *C/C++* or Python code for embedding various algorithms and decision making steps in the vSA packet handling process. Referring back to Figure 3 and the presented Uppaal [33] based decision tree, the model is stored as an XML file which can be easily parsed, interpreted and utilized for code generation in the scope of the current requirements.

D. Decision Engine

The Decision Engine enables the 40SEC appliance to actively react on intrusions by analyzing the alerts generated from the IDS and adapting the firewalls configuration to stop the attack. Thereby, the firewall configurations and/or the IDS decisions are designed and verified according to the above described design, modelling and code generation process.

E. Alarms

Alarms in Snort can be structurally different according to the parameters that we use when Snort starts. To get detailed information about each packet triggering a rule in Snort, we can use the *-dev* parameter. To get only the packet id and the rule id, we can use *-A test*. To get more information about the alarms in a file, we can use *-A fast*. In the scenarios that will be described later on, we use *-A test* and *-A fast* to get enough information about the alarms in a specific file. The information that we get from the Snort alarms includes: *time-stamp*, *id of the packet* that triggered the rule, *id of the rule*, *five-tuples*[2], *class of the attack* (classified by Snort in the *classification.config* file), a *message describing the alarm*, and the *priority of the rule*. An example of an alarm in the alert log of Snort is shown in Figure 4:

```
8  1    10000002    12
09-09:56:06.661005 [**] [1:10000002:12] TCP Port Scanning [**] [Classification: A TCP connection
detected] [Priority: 4] {TCP} 10.147.69.61:51397 -> 10.147.144.184:80
```

Figure 4: Example Output of detected Port Scan

This example shows a port scan that was detected on port 80. You can see that "1688" is the packet id, "01/09-09:56:06.661005" is the time-stamp, "10000002" is the rule id, "TCP Port Scanning" is the message, "A TCP connection was detected" is the classification, "4" is the priority, {TCP} 10.147.69.61:51397 -> 10.147.144.184:80 is the belonging five-tuples.

Referring back to Figure 1, after gathering the Snort alarms from various sources, we developed a Python code that listens to any new alarms from Snort, extracts the important information from them, and puts them in a JSON structure that is subsequently sent to the Decision Engine via a RestAPI.

Having described the key aspects of the softwarized version of the 40SEC firewall/IDS, the following section proceeds with the FPGA architecture which constitutes the acceleration aspect on the edge network.

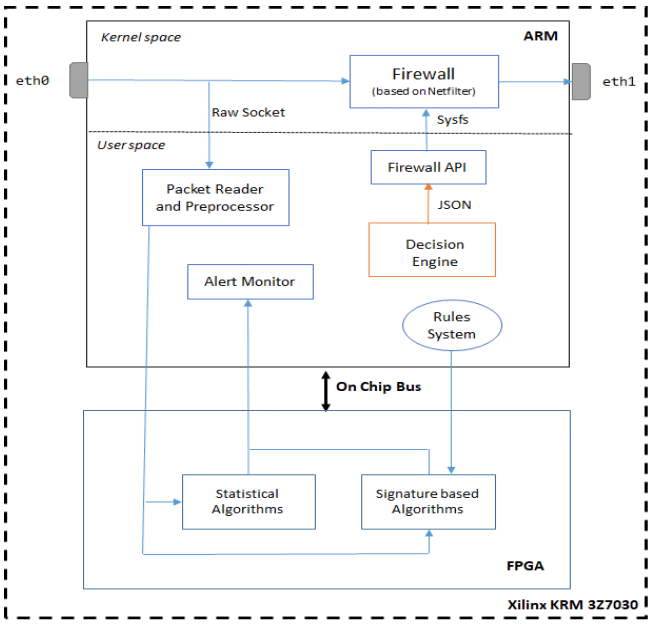

Figure 5: FPGA based Architecture

[2] Five-tuples: https://www.techopedia.com/definition/28190/5-tuple, as of date 28.01.2019

VI. THE FPGA BASED ARCHITECTURE

Intrusion Detection Systems can be anomaly based or signature based. If we take a signature based IDS as an example, we can see that it is composed of three main functionalities:

1. **Data processing:** Here, the network traffic is collected and formatted, in order to be analyzed by the detection algorithms
2. **Detection:** This operation investigates the difference between "normal" traffic and an intrusion.
3. **Response:** Here, appropriate alerts are generated based on the decision criteria and the intrusions detected

The typical and most straightforward way of detecting intrusions is to compare string patterns (attacks signatures) to the payload of the traffic packets. It has been shown ([5], [8]) that utilizing existing efficient string matching algorithms (e.g. [6], [7]) leads to significant costs. For instance, the measurements performed recently with Snort on a production network and reported in [8] show that almost 31% of the total processing effort can be traced back to string matching. The same report also emphasizes that in case of intensive web traffic, the processing cost increases to 81% of the total processing time. For these reasons, we would like to explore how the performance of our 40SEC, in particular the IDS part, will behave when the string matching algorithms run on FPGA instead of the ARM processor. The belonging architectural vision is depicted in Figure 5 illustrating the separation between the software part (i.e. ARM part) and the FPGA based hardware accelerator.

To realize the above mentioned vision for signature based IDS, we have decided to implement on FPGA the Wu-Manber algorithm that Snort uses in a simplified version. For anomaly detection, we have chosen to implement two well-known techniques from the flooding/DoS attacks detection domain. The first one is based on adaptive thresholds, while the second one relies on change point theory. It is worth to mention that the algorithms that will be discussed below are applied for IP packets in the current work, and can similarly be used for other packet types such as SYN and ICMP.

A. Statistical Algorithms

In the following, key algorithms are presented that fit into the abstract 40XLERATOR architecture.

Adaptive Threshold Algorithm

Flooding attacks detection cannot rely on static thresholds. In fact, two main factors can affect the detection process. These factors are the attack type and the situation of the communication channel (burst or not). An attacker can flood the target system gradually with malicious traffic. If the channel is not in a burst mode and the threshold was assigned a high value, the attack will not be detected. On the other side, if the threshold was set to a small value and the channel experiences a huge amount of traffic, the detection process will lead to many false alarms. One way to overcome this problem is to use a threshold that adapts to the traffic behavior.

The algorithm we propose here relies on checking whether the number of IP packets over a given time interval exceeds a particular threshold. In order to take into account daily and weekly traffic variations and trends, the threshold value is computed adaptively based on an estimate of the mean number of IP packets. Let us assume x_n is the number of IP packets in the *n-th* time interval, and μ_{n-1} is the mean value calculated from measurements prior to *n*. In this case, an alarm will be generated at time *n* if,

$$x_n \geq (\alpha + 1)\mu_{n-1} \quad (1)$$

α is a strictly positive parameter representing the percentage above the mean value that we consider to be an indicator of an intrusion [14].

The mean μ_n can be calculated over some past time window or using an exponential weighted moving average (EWMA) of previous measurements,

$$\mu_n = \beta\mu_n + (1 - \beta)x_n \quad (2)$$

where β is the EWMA factor.

Like any other statistical algorithm, false positives could be a serious issue. In order to reduce the amount of false positives, the authors of [14] suggest sending alarms only after a minimum number of consecutive violations of the threshold. In addition, in the course of the FORTIKA project [47], we will be further investigating how to tune the parameters: α, , the length of the measurement time interval, and the number of successive threshold violations, in order to increase detection efficiency, while keeping the amount of generated false alarms to the minimum.

Change Point Detection (Cumulative Sums Algorithm – CUSUM-)

Change points represent sudden variations in time series data. They may reflect changes that occur between states. The change point theory is a powerful tool for investigating time series and has been used in different applications including health monitoring, climate change detection [18] and DoS attacks detection and prevention ([19], [14], [20]). If we take DoS attacks as an example, although intrusions happen at unpredictable points in time, they are often accompanied with a change of some statistical properties of the network traffic. This makes Change Point detection schemes suitable for investigating such changes. Often these techniques use Cumulative Sums (CUSUM) of data that are computed over time, and the resulting series is observed to localize the change points that might occur. The CUSUMs are often used because of their simplicity and low computational complexity. There are two variants of these techniques: the first one, called parametric CUSUM, assumes that the observed variable follows a known distribution (e.g. Poisson distribution). This assumption is in general not true. The second variant, called non-parametric CUSUM, does not require that the undertaken measurements exhibit a certain pattern.

Implementing the Change Point techniques on FPGA will certainly lead to the acceleration of the applications being considered. Unfortunately, from the research point of view, only few papers discussing such implementations are available. In [21], a nonparametric CUSUM that serves as a hardware plugin for Software Defined Monitoring (SDM) was implemented on a Network Interface Card (NIC) and used to detect anomalies directly in FPGA. The authors claim to be able to realize different detections simultaneously without any loss in throughput. In [21], an online Change Point detection algorithm based on an Auto Regression (AR) model, called *ChangeFinder*, was implemented on a NIC. The proposed solution computes the change-point score from time series data received from the 10GbE (10Gbit Ethernet) NIC. The objective of the authors was to reduce the host workload and improve the performance by offloading the ChangeFinder algorithm from host to the NIC [21]. According to their evaluation results, an improvement of 16,8x in change point detection throughput was realized compared to the baseline software implementation while keeping the same accuracy.

To apply Change Point theory to DoS attacks detection, three parts are needed:

(1) The network traffic variable that will be measured and which will be the basis for the time series to be observed. Here, measurements related to protocols such as TCP, UDP, ICMP, and SIP can be used. It is also possible that pairs like TCP SYN-FIN [19], where a strong correlation exists in between, can be observed and utilized by the CUSUM techniques.
(2) Once the time series is defined, it may need to be made stationary (i.e. a time independent normalization would need to be applied before feeding values for alarm generation) through appropriate transformations, in order to be used by the CUSUMs.
(3) A bunch of parameters and thresholds must be set and potentially optimized in order to detect the attacks and reduce the amount of false alerts.

Concretely, the CUSUM technique we will be implementing in FORTIKA [47] follows the algorithm discussed by Siris in [14]. Let us assume we are observing the number of IP packets entering the network under consideration during a period of time that is divided into time intervals of the same duration. As mentioned earlier, we can observe TCP, UDP, and ICMP packets or even pairs such as TCP SYN-FIN, instead of IP packets. Let X_n be the number of IP packets collected within the *n*-th time interval and μ_n the corresponding mean rate at time n. The value μ_n is calculated using an exponentially weighted moving average,

$$\mu_n = \beta\ \mu_{n-1}\ + (1-\beta)\, X_n \qquad (3)$$

where β is the exponentially weighted moving average (EWMA) factor and $\mu_0 = X_1$. To give some more details, the exponential weighted moving average is a weighted average where the weighting decreases in an exponential way with each following measurement with respect to previous measurement values. The EWMA is also used in our context to avoid explicitly taking into account the seasonability and the trend factors ([14], [20]). As the time series X_n , *n = 1, 2, ...* is in general non-stationary because of the just mentioned factors, it can be made stationary by using the following transformation (according to [19], [14]),

$$\tilde{X}_n = \frac{X_n}{\mu_{n-1}} \qquad n \geq 1 \qquad (4)$$

The nonparametric CUSUM algorithm is given by,

$$Y_n = \max\{0,\ Y_{n-1} + \tilde{X}_{n-1} - a\},\ \ Y_0 = 0 \qquad (5)$$

where *a* is a positive real number defined experimentally to minimize the false alarm rate [19]. As it was discussed in [20], assigning an appropriate value to the parameter *a* can be done by taking it sufficiently large with respect to the historical estimate $E_{est}\ (\tilde{X}_n)$ for the expected value $E(\tilde{X}_n)$ representing the normal behavior.

Indeed, an estimate $E_{est}\ (\tilde{X}_n)$ of $E(\tilde{X}_n)$ can be calculated experimentally, and then it can be set to the sum of $E_{est}\ (\tilde{X}_n)$ and a multiple of the estimate of the corresponding standard deviation, i.e,

$$a = E_{est}\ (\tilde{X}_n) + \theta\ \sigma_{est}(\tilde{X}_n\ ) \qquad (6)$$

θ is an appropriate non negative number and $\sigma_{est}(\tilde{X}_n\ )$ is an estimate of the standard deviation of the normal traffic.

Another parameter that is needed is the stopping time. The latter is the time when a change in the CUSUM algorithm occurs and can be formulated as follows,

$$T = T(Thr) = min\{n \geq 1,\ Y_n \geq Thr\} \qquad (7)$$

The threshold *Thr* can be seen as a positive number specifying the value that when exceeded, an attack is signaled. This value can also be defined as a multiple of the estimated standard deviation $\sigma_{est}(\tilde{X}_n\ )$. This means *Thr* will be of the form $\rho\ \sigma_{est}(\tilde{X}_n\ )$ where ρ is a positive number reflecting the size (in terms of the standard deviation) of the shift we want to detect [20].

B. Signature based Algorithms

Sun Wu and Udi Manber developed the Wu-Manber algorithm in 1994. It has its origins in the Boyer-Moore and Rabin-Karp matching algorithms, whereas the Wu-Manber advantages are a high average case performance and a less required small memory space. Some restrictions have to be mentioned for the Wu-Manber algorithm:

(1) The amount of patterns that can be processed simultaneously is less than a few hundred.
(2) There should not be a high deviation in the length of the patterns that has to be matched, in order to ensure reasonable performance.

(3) Searching time increases inversely proportional to the length of patterns. This means that too short patterns require a lot more time for searching.

C. The Wu-Manber Algorithm

The initial proposed Wu-Manber algorithm in 1994 provided with several improvements. Beside other Intrusion Detection Systems, snort uses a simplified variant of the Wu-Manber algorithm too.

Recently published studies - dealing with the performance of Snort using the Wu-Manber pattern search algorithm instead of the formerly used Set-Wise Boyer-Moore and Aho-Corasick algorithm - pointed out that Snort gained a remarkable performance improvement with Wu-Manber. Following pseudo-code [16] (see Figure 6) has been the source for the implementation of the Wu-Manber algorithm for the 40SEC platform:

```
1: procedure WM(y, n, B, B', SHIFT, HASH, PREFIX, PAT_POINT)
       ▷ Input:
       ▷ y ← array of n bytes representing the text input
       ▷ B ← integer representing the suffix block length
       ▷ B' ← integer representing the prefix block length
       ▷ SHIFT ← SHIFT table (see description above)
       ▷ HASH ← HASH table (see description above)
       ▷ PREFIX ← PREFIX table (see description above)
       ▷ PAT_POINT ← table of pointers to keywords (like our x it has m keywords)

2:     m ← min {length of all keywords}                          ▷ minlen
3:     i ← m − 1
4:     while i ≤ n do                                            ▷ Matching
5:         h ← hash(y[i − B + 1], ..., y[i]) ▷ hash over B bytes back from index i in y
6:         shift ← SHIFT[h]
7:         if shift = 0 then                                     ▷ Suffix block matches
8:             text_prefix ← hash(y[i − m + 1], ..., y[i − m + 1 + B'])
9:             p ← HASH[h]                                       ▷ a C style pointer
10:            p_end ← HASH[h + 1]                               ▷ a C style pointer
11:            while p < p_end do
12:                if text_prefix = PREFIX[p] then               ▷ Prefix matches
13:                    px ← PAT_POINT[p]          ▷ Pointer to the current keyword
14:                    len ← length of px                        ▷ Length of current keyword
15:                    j ← 0                                     ▷ Count of matched characters
16:                    while j < len and y[i − len + 1 + j] = px[j] do
17:                        j ← j + 1
18:                    end while
19:                    if j ≥ len then
20:                        output i − len + 1
21:                    end if
22:                end if
23:                p ← p + 1
24:            end while
25:            i ← i + 1                                         ▷ Shift only by one place
26:        else
27:            i ← i + shift                                     ▷ Skip part of the text
28:        end if
29:    end while
30: end procedure
```

Figure 6: Wu-Manber Algorithm Pseudo-Code [16]

Wu-Manber algorithm can be used as it is in email spam detection. It can be considered as an anti-spam content-based technique, and utilized to accelerate the emails analysis.

Wu-Manber is a computation extensive algorithm which makes it a good candidate for an FPGA implementation. A naïve replication of it on the FPGA will not be efficient when compared to that of an ARM processor. In addition, implementing the algorithm on an FPGA creates a memory overhead for data transfer between the processor and the FPGA since it is a memory resource-constrained architecture. This will lower the performance and hence we need to parallelize the algorithm.

D. Parallalizing Wu-Manber

The Wu-Manber algorithm comprises of two stages – a preprocessing phase (represented as step 1 in Figure 6) and the scanning phase (represented as steps 2-30 in Figure 6). The control flow and the dataflow are serialized in this algorithm. The preprocessing phase is computed for the given set of keywords/patterns only once and doesn't add to the overall computation time period.

We decided to exploit data level parallelism in the scanning phase algorithm using an asynchronous mode of execution. The SIMD[3] architecture is implemented. Let L be the length of the input data stored in the processor memory. Using the AXI4 burst transfer mode, the input data is double buffered asynchronously into the FPGA BRAM with buffers B_n, where B is the buffer of length $N\ (0 < N \leq L)$ and n is the number of buffers $(n > 1)$. Let h be the number of identical hardware instances of the scanning phase. The buffered input data is sliced into N/h slices.

This FPGA design is done via Vivado HLS which offers the flexibility for RTL synthesis from high level programming language (C/C++) and generates the bitstream for the IP core. By utilizing the asynchronous mode of operation, the h hardware instances operate on their corresponding buffer slice. The goal is to improve the performance by 2x at the maximum.

VII. Testing and Experiments

In order to test the 40SEC and each of its subcomponents (Firewall, IDS, RESTApi, etc.) within defined scenarios, a test environment was set.

A. Scenarios

The following table provides the scenarios used in our testing activities. The listed attacks were executed in order to test and validate the current state of the 40SEC implementation.

Attack name	**Port Scanning (Reconnaissance Attack)**
Verification/ testing procedure	1. Scan the destination with *nmap* to check all the open ports: this attack will send a lot of different TCP packets to different ports in the destination to see what are the ports that are open and then use them to exploit the possible further attacks. 2. See what information can be gathered that may be useful for additional attacks. 3. Check if the IDS detects the scanning.
Attack name	**Denial of Service**
Verification/ testing procedure	1. Perform multiple DoS-based attacks (also possible with nmap): ◦ *SYN Flood* (using the *hping3* with flag *S* tool in Kali Linux): this attack will be done by

[3] http://www.cs.uu.nl/docs/vakken/magr/2017-2018/files/SIMD%20Tutorial.pdf

	sending a lot of TCP packets with SYN flag to port 80 to initiate a connection with the web server ◦ *Ping of death* (using *ping <dest_IP> -t -l 65500*): this attack will send a lot of huge size ICMP packets to the destination IP, in order to make the belonging device go down. 2. Check if the IDS detects the attack.
Attack name	**Malicious File**
Verification/ testing procedure	1. Create a text file with malicious content to be sent over the IDS controlled link 2. Match the malicious content with the belonging Snort rules and generate the alarm 3. Check if the IDS detects the attack [and if the IPS is able to prevent it]

B. *Testbed Components*

Figure 7 illustrates our local testbed. One can clearly see the positioning of the vSA as allowing it to control traffic between the LAN and WAN segments towards the Internet. Thereby, the LAN segment can be assumed as representative for the local network of a European SME. Furthermore, a management interface and a belonging management/control network is spanned over all involved components allowing us to control their behavior, initiate different scenarios, execute tests and run experiments evaluating different configuration of the 40SEC vSA. In the following, the selected tools are introduced, which we have deployed in our testbed and utilize for testing and experiments.

IPerf (client, server) [48]: IPerf is used to generate TCP and UDP data streams. It provides a client-server application that allows to measure bandwidth, quantity of data that passes a network and belonging time aspects. IPerf creates a report with a timestamp, details of transmitted network packets and measured bandwidth. In the case of the 40SEC vSA, IPerf is used to assess the network data processing performance of the FPGA hardware in conjunction with the ARM part.

TCPReplay [49]: TCPreplay contains different open source utilities that can be used to replay a *pcap* file that has been captured before. It can be easily used as a test tool to check the performance of IDS, IPS or Firewall systems. In our scenarios, we are utilizing it to simulate a big network traffic coming to our network and checking if it gets blocked by our Firewall or reported by our IDS. TCPreplay can take long time to replay all the information included in the *pcap* file. To improve the performance we use *-k* parameter with TCPreplay, which preloads packets into RAM before sending. In addition, we use the *-t* parameter to replay packets as fast as possible. It is possible to use the *--loop* parameter in case you are using a small *pcap* file, which is to be repeatedly communicated towards the target network infrastructure.

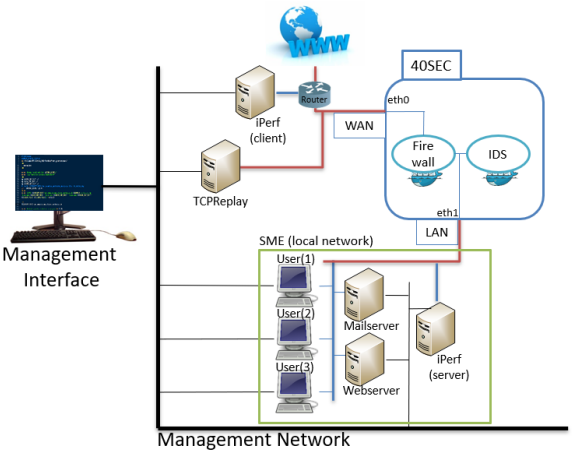

Figure 7: FOKUS local Testbed Components

Web Service: To test the performance of the Virtual Security Appliance, we run a simple web service in our testbed, which offers a REST API for computing a complex function value based on an inputted number over the REST API - we use a complex function for simulating web server load. Then we can measure the availability and performance of the service under different scenarios, from high network load to different attacks. This will give information about the latency, throughput and effectiveness of the 40SEC security device.

Performance and Security Testing Tools: Besides tools that are available (e.g. *hping3*, Metasploit *nmap* and *Wireshark*) and preinstalled in Kali Linux (a free Debian Linux System for penetration testing), *JMeter* is one of the tools that is used to simulate a heavy load to the 40SEC Gateway and to test its performance under different load types. Furthermore, in our scenarios, we have used tools like *nmap* to achieve port scanning attack and *hping3* to achieve SYNflood attack.

C. *First Experimental Results*

The first experiments we wanted to undertake are related to the Wu-Manber algorithm. The idea was to run this algorithm on ARM and also on FPGA and compare between the corresponding performances. As mentioned earlier, this algorithm can directly be used for email spam content analysis, however as we could not find a suitable email spam database for testing purposes, we have chosen to apply this algorithm on available books.

When conducting the measurements, we took a set of corpora from different sizes (“Word Count”-column) and searched in those for various random patterns. The corpora were given by various books (e.g. adventures of Sherlock Homes, History of United States of America, Manual for Surgery and War and Peace), whilst the patterns were randomly sampled from available lists of common words in the English language.

The experiments were performed on a Zync-7000 SoC architecture from Xilinx [50]. It houses a dual-core ARM Cortex-A9 based processing system and a Zync-7000 Kintex-

7 FPGA with ARM AMBA AXI based interconnect between the processor system (PS) and the programmable logic (PL).

During the experiments with the ARM processor, we pre-loaded the data into the architecture's memory, and the data was correspondingly pre-processed. Afterwards, the matching operations were initiated. From Table 1, we could observe the timing benchmark for the performance of Wu-Manber on ARM processor for various sizes of the text and the patterns.

Word count	Matches	Init + Search
100,000	70,972	0.2048 s
100,000	145,251	0.8247
100,000	14,791	0.1438
100,000	1,867	0.5368
100,000	3,262	1.3042
1,000,000	733,226	2.286
1,000,000	1,756,633	9.7276
1,000,000	174,759	1.6762
1,000,000	20,786	5.9409
1,000,000	35,890	14.1044

Table 1: Wu-Manber related measurements on ARM

Word count	Matches	Init + Search
100,000	70,972	1.4667 s
100,000	145,251	6.2250 s
100,000	14,791	1.1950 s
100,000	1,867	4.6928 s
100,000	3,262	11.9615 s
1,000,000	733,226	16.7217 s
1,000,000	1,756,633	73.7540 s
1,000,000	174,759	14.2022 s
1,000,000	20,786	52.8587 s
1,000,000	35,890	128.1470 s

Table 2: Wu-Manber related measurements on FPGA without parallelization

During the second stage of the experimentation, we implemented the algorithm the serial and the parallelized versions on FPGA, utilizing Vivado HLS programming. The entire dataset was stored on the memory of the processor and communicated over the AXI interface. As mentioned before in Section VI. C, the FPGA performed poorly in the serial version. The parallelized version as mentioned in Section VI. D had a performance improvement of 20% to 50% when compared to the serial version as observed in Table 2 and Table 3. However, when compared to the benchmark the performance is still poor. It can also be observed from Table 3 that the number of matches is lesser than that of the benchmark.

The inferences from the experiment being – despite the parallelization, the data transfer overhead from PS to PL during the process execution on the PL causes stall in execution. This significantly lowers the performance by 70-80%. The reduced number of matches in the parallelized version signifies the patterns that were missed due to the slicing of the text data.

Word count	Matches	Init + Search
100,000	69,447	0.7684 s
100,000	141,864	4.7660 s
100,000	14,378	0.5382 s
100,000	1,811	3.4782 s
100,000	3,175	9.6686 s
1,000,000	717,442	8.5898 s
1,000,000	1,716,223	55.3991 s
1,000,000	169,809	6.4198 s
1,000,000	20,220	37.8416 s
1,000,000	34,977	99.0528 s

Table 3: Wu-Manber related measurements on FPGA with parallelization

VIII. Conclusions and Future Work

The current paper presented our activities towards implementing IDS in the edge network segment. Indeed, we discussed the 40SEC IDS architecture, which can run both in a virtualized setup and in an FPGA supported mode. In that line of thought, various important aspects related to the key operations and the modeling and design of the 40SEC (firewalls/IDS) were presented, whilst in parallel the most important operational modules (e.g. pattern matching) were identified, which are also key candidates for FPGA acceleration. A set of scenarios simulating network attacks, and a testbed for rapid prototyping of security appliances in ARM and FPGA were described.

We have implemented Wu-Manber algorithm on FPGA to support the signature-based IDS and compared the results with that of the ARM processor. The experiments have called for a redesigning of the algorithm and a more sophisticated parallelization.

We will also focus on increasing the level of acceleration, which is currently on a very generic level. Thereby, we want to identify more aspects for handling in FPGA and establishing a stable process from the high level design down to the hardware deployment of the required IDS/IPS/firewall mechanisms.

Acknowledgment

This work has been undertaken in the context of the European H2020 project FORTIKA [47], Grant agreement no 740690). This project aims at developing a robust, resilient and effective cybersecurity solution that combines hardware and software and which can be effortlessly tailored to each individual enterprise's evolving needs. This will be achieved

by opening the FORTIKA middleware platform to third-party cyber security applications and services through the FORTIKA marketplace.

REFERENCES

[1] D. Lin, "Network Intrusion Detection and Mitigation Against Denial of Service Attack", Technical Report, January 2013, University of Pennsylvania, Link: https://repository.upenn.edu/cgi/viewcontent.cgi?article=2027&context=cis_reports

[2] A. Alharbi, S. Alhaidari, M. Zohdy, „ Denial-of-Service, Probing, User to Root (U2R) & Remote to User (R2L) Attack Detection using Hidden Markov Models ", International Journal of Computer and Information Technology (ISSN: 2279 – 0764) Volume 07– Issue 05, September 2018, Link: https://www.ijcit.com/archives/volume7/issue5/IJCIT070501.pdf, as of date 22.01.2019

[3] Snort, Link: https://www.snort.org, as of date 22.01.2019

[4] Suricata, Link: https://suricata-ids.org/, as of date 22.01.2019

[5] S. Antonatos, K. G. Anagnostakis†, E. P. Markatos, M. Polychronakis, "Performance Analysis of Content Matching Intrusion Detection Systems",

[6] A. Aho and M. Corasick. Fast pattern matching: an aid to bibliographic search. Commun. ACM, 18(6):333–340, June 1975.

[7] R. Boyer and J. Moore. A fast string searching algorithm. Commun. ACM, 20(10):762–772, October 1977.

[8] M. Fisk and G. Varghese. An analysis of fast string matching applied to content-based forwarding and intrusion detection. Technical Report CS2001-0670 (updated version), University of California - San Diego, 2002.

[9] Sourcefire. Snort 2.0 - Detection Revisited. http://www.snort.org/docs/Snort 20 v4.pdf, October 2002.

[10] S. Wu and U. Manber. A fast algorithm for multi-pattern searching. Technical Report TR-94-17, University of Arizona, 199

[11] S. Wu and U. Manber. A fast algorithm for multi-pattern searching. Technical Report TR-94-17, University of Arizona, 1994.

[12] M. Basseville et aI, "Detection of Abrupt Changes: Theory and Application", ISBN: 0-l3-126780-9, Prentice Hall, 1993.

[13] H. Wang et aI, "Detecting SYN Flooding Attacks", In Proceedings of lEEE INFOCOM 2002, New York, June 2002

[14] V. A. Siris, F. Papagalou, "Application of Anomaly Detection Algorithms for Detecting SYN Flooding Attacks", In Proc of Globecom 2004, Texas, 29 Nov -3 Dec, Dallas, Texas, USA.

[15] NS-KDD, Link: https://www.unb.ca/cic/datasets/nsl.html, as of date 22.01.2019

[16] Lua definition, Link: https://www.lua.org/about.html, as of date 22.01.2019

[17] Nguyen Le Dang, Dac-Nhuong Le, Vinh Trong Le, "A New Multiple-Pattern Matching Algorithm for the Network Intrusion Detection System, IACSIT, April 2016

[18] S. Aminikhanghahi, D. J. Cook, "A survey of Methods for Time Series Change Point Detection", PMC Jun 8,2017 Jun 8. doi: 10.1007/s10115-016-0987-z

[19] H. Wang, D. Zhang, K. G. Shin, "Detecting SYN flooding attacks," in Proc. of IEEE INFOCOM'02, 2002.

[20] Y. Rebahi, D. Sisalem, " Change-Point Detection for Voice over IP Denial of Service Attacks", In the Communication in Distributed Systems – 15. ITG/GI Symposium, 26 Feb.-2 March 2007, ISBN: 978-3-8007-2980-7

[21] T. Cejka, L. Kekely, P. Benacek, R. B. Blazek, H. Kubatova, " FPGA Accelerated Change-Point Detection Method for 100Gb/s Networks", 9th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS), At Telc, Volume: ISBN 978-80-214-5022-6

[22] T. Iwata, K. Nakamura, Y. Tokusashi, H. Matsutani, "Accelerating Online Change-Point Detection Algorithm using 10GbE FPGA NIC", Euro-Par Workshops 2018: 506-517

[23] Network Functions Virtualisation, Introductory White Paper, ETSI, Link: https://portal.etsi.org/nfv/nfv_white_paper.pdf, , as of date 22.01.2019

[24] Most CIOs fear IoT performance problems could damage revenues, Link: http://www.netimperative.com/2018/11/most-cios-fear-iot-performance-problems-could-damage-revenues/, as of date 22.01.2019

[25] S Jiang, D He, C Yang, C Xu, G Luo, Y Chen, Y Liu, J Jiang, "Accelerating Mobile Applications at the Network Edge with Software-Programmable FPGAs" IEEE INFOCOM 2018-IEEE Conference on Computer Communications, 55-62

[26] CVE MITRE: https://cve.mitre.org/, as of date 22.01.2019

[27] ReqIF format: https://www.omg.org/spec/ReqIF/About-ReqIF/, as of date 22.01.2019

[28] Overview of Rational DOORS: https://www.ibm.com/support/knowledgecenter/en/SSYQBZ_9.6.1/com.ibm.doors.requirements.doc/topics/c_welcome.html, as of date 22.01.2019

[29] ProR: http://www.eclipse.org/rmf/pror/, as of date 22.01.2019

[30] iptables: https://github.com/ldx/python-iptables, as of date 22.01.2019

[31] Kamel Karoui, Fakher Ben Ftima, Henda Hajjami Ben Ghézala: „Distributed firewalls and IDS interoperability checking based on a formal approach", CoRR abs/1310.2861 (2013)

[32] Thawatchai Chomsiri, Xiangjian He ,Priyadarsi Nanda, "Limitation of listed-rule firewall and the design of tree-rule firewall", Proceedings of the 5th international conference on Internet and Distributed Computing Systems 978-3-642-34882-2, Wuyishan, Fujian, China, 275-287, 2012, 10.1007/978-3-642-34883-9_22, Springer-Verlag

[33] Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, Wang Yi, "UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems." Hybrid Systems 1995: 232-243

[34] XILINX KRM-3Z7030 50mm x 70mm module, Link: http://www.knowres.ch/products/krm-3z030-768/, as of date 29.01.2019

[35] KRC3701 Carrier Kit, Link: http://www.knowres.ch/products/krc3600-carrier/.as of date 29.01.2019

[36] C. R. Clark, C. D. Ulmer & D. E. Schimmel, "An FPGA-based network intrusion detection system with on-chip network interfaces", International Journal of Electronics, 2006, 93:6, pp. 403-420, DOI: 10.1080/00207210600566083

[37] C. R. Clark and D. E. Schimmel, "A pattern-matching co-processor for network intrusion detection systems", Proceedings. 2003 IEEE International Conference on Field-Programmable Technology (FPT) (IEEE Cat. No.03EX798), Tokyo, Japan, 2003, pp. 68-74.

[38] Young Hyun ChoWilliam H. Mangione-Smith, "Deep network packet filter design for reconfigurable devices", February 2008ACM Transactions on Embedded Computing Systems 7(2), DOI: 10.1145/1331331.1331345

[39] R. Proudfoot, K. Kent, E. Aubanel and N. Chen, "Flexible Software-Hardware Network Intrusion Detection System," 2008 The 19th IEEE/IFIP International Symposium on Rapid System Prototyping, Monterey, CA, 2008, pp. 182-188. doi: 10.1109/RSP.2008.11

[40] Y. Liu, D. Xu, D. Liu, L. Sun, „A Fast and Configurable Pattern Matching Hardware Architecture for Intrusion Detection", WKDD 2009, pp 614-618

[41] C. Greco, E. Nobile, S. Pontarelli and S. Teofili, "An FPGA based architecture for complex rule matching with stateful inspection of multiple TCP connections," 2010 VI Southern Programmable Logic Conference (SPL), Ipojuca, 2010, pp. 119-124. doi: 10.1109/SPL.2010.5483029

[42] Abhishek Mitra, Walid Najjar, Laxmi Bhuyan, "Compiling PCRE to FPGA for accelerating SNORT IDS", Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems. 978-1-59593-945-6, Orlando, Florida, USA, pp. 127-136, 2007

[43] R. Sidhu, V. K. Prasanna, "Fast Regular Expression Matching using FPGAs", In J. M. Arnold and K. L. Pocek editors, Proc of the IEEE Workshop on FPGAs for Custom Computing Machines, Napa, CA, April 2001

[44] Perl Compatible Regular Expressions (PCRE), Link: https://www.pcre.org/, as of date 29.01.2019

[45] JHDL, Link: http://www.jhdl.org/overview.html, as of date 29.01.2019

[46] Brad L. Hutchings, R. Franklin, D. Carver: "Assisting Network Intrusion Detection with Reconfigurable Hardware", FCCM 2002: pp. 111-120

[47] FORTIKA project: https://fortika-project.eu/, as of date 29.01.2019

[48] Iperf, Link: https://iperf.fr/, as of date 29.01.2019

[49] TCPreplay, Link: https://tcpreplay.appneta.com/, as of date 29.01.2019

[50] Zynq-7000 SoC Data Sheet: Overview, Link: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf, as of date 11.11.2019