A novel genetic algorithm approach for firewall policy optimization

Antonio Coscia[2], Vincenzo Dentamaro[1], Stefano Galantucci[1,*], Donato Impedovo[1] and Antonio Maci[2]

[1]*University of Bari Aldo Moro, Department of Computer Science, 70125 Bari, Italy*
[2]*BVTech SpA, 20123 Milano, Italy*

Abstract
Exceeding the performance limits of current application filtering systems, both in terms of speed and versatility in supporting sophisticated filtering policies, is a highly topical issue in cybersecurity environments. Systems built for specific uses are fast but have limited filtering functions, and, on the other side, systems that adopt efficient semantics fail to meet performance requirements in terms of speed. This work aims to propose a novel mechanism for solving the problem of the optimal ordering of filtering policies in a firewall, to reduce the number of times the generic rule is evaluated, and thus having better efficiency from the point of view of time processing, as well as the speed of the filtering action. The proposed approach uses a genetic algorithm and involves combining two heuristics for managing mutually dependent policies. The results are encouraging in terms of both performance and timing.

Keywords
Firewall, Rules, Policy, Optimization, Constraint, Direct acyclic graph, Order, Sort, Filter, Heuristic

1. Introduction

The packet filtering operation of a Firewall (FW) may require high computation times, which grow as the complexity of the implemented Security Policy (SP) increases. This could be particularly disadvantageous in scenarios in which fast communications must be guaranteed, for example, in Gigabit Ethernet networks, up to situations in which the system is susceptible to Denial of Service (DoS) [1] attacks. Attacks of this type could exploit the high latency of systems that are not optimal in terms of time processing capacity. It is therefore required to introduce procedures to remove the anomalies present in an SP, using the most common techniques well-argued and proposed by the authors of [2, 3]. However, as noted by these authors, in some cases, these techniques may not be sufficient, and it is consequently necessary to perform a targeted sorting of the rules that make up the SP. In the last decade, the problem of optimizing the ordering of the rules present in an SP, to minimize the number of times the generic rule is evaluated, as well as the time processing in the filtering action, has led to several works by

ITASEC'22: Italian Conference on Cybersecurity, June 20–23, 2022, Rome, Italy
*Corresponding author.
a.coscia@bv-tech.it (A. Coscia); vincenzo.dentamaro@uniba.it (V. Dentamaro); stefano.galantucci@uniba.it (S. Galantucci); donato.impedovo@uniba.it (D. Impedovo); a.maci@bv-tech.it (A. Maci)
0000-0003-1148-332X (V. Dentamaro); 0000-0002-3955-0478 (S. Galantucci); 0000-0002-9285-2555 (D. Impedovo)

researchers [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15].
This work presents an innovative approach compared to those in the existing literature to be very efficient in terms of execution time, even for very complex and extensive SPs. For the latter analysis, the time complexity analysis methods proposed in [16] will be employed. A genetic algorithm for efficiently ordering the policies of a FW is described, with a particular focus on the problem of constrained ordering, not to modify the SP's correct functioning.
The main contributions of the present work are:

- Proposal of a genetic approach for the problem of sorting the SPs of a FW;
- Introduction of heuristics for solving a constrained sorting problem that establishes the correct positioning of placeholders in the sequence to be sorted, with consequent reduction of the application space of the genetic algorithm, in order to grant efficient execution times;
- Presentation of innovative mechanisms in the crossover and mutation phases of the adopted genetic algorithm to make the proposed solution more efficient.

The paper is organized as follows: section 2 provides a context analysis of the problem under consideration; section 3 presents a review of the approaches used in the literature to address the same problem; section 4 describes the solution to the problem and its two phases; section 5 provides a comparison of the proposed heuristics for solving the constraints, compared to a topological ordering, as well as a first examination of the results obtained from the application of the whole proposed algorithm about its time complexity as the complexity of the considered ruleset varies. Conclusions and possible future directions are presented in section 6.

2. Background

A FW uses a list of filtering rules that act on packets of traffic generated in and out of a computer network. An example of such a list is presented in Table 1. A SP is ordered and can be expressed

ID	Action	Direction	Protocol	Source	Destination	Destination Port
0	Block	In	Any	10.1.1.1	20.1.1.1	Any
1	Pass	In	[TCP, UDP]	192.168.56.3	Any	Any
2	Block	Out	UDP	10.0.0.8	Any	53
3	Pass	In	TCP	172.12.0.98	Any	Any
4	Block	In	CARP	10.0.0.0/8	Any	Any
5	Block	In	TCP	192.168.41.78	192.168.41.79	Any
6	Block	Out	Any	10.0.0.0/8	Any	Any
7	Forward	In	Any	172.12.0.0/16	Any	Any
8	Block	Out	Any	10.0.0.0/8	Any	Any
9	Pass	In	TCP	10.1.1.2	20.1.1.1	80
10	Pass	In	TCP	10.1.1.0/24	20.1.1.1	80
11	Block	Any	Any	Any	Any	Any

Table 1
Example of a SP

as $\mathbf{R} = (r_0, r_1, ..., r_N)$, where N is the number of rules that compose it.

Formally, a rule can be modeled as an n-uple $r_{ID} = (\Sigma_{ID}, [\alpha_1, \alpha_2, ..., \alpha_k])$, where α_i, with $i = 1, ..., k$ represents the generic attribute (token) typical of network traffic, while Σ_{ID} represents the filtering action to be taken.
As noted earlier, ordering the rules within the SP affects packet filtering performance, which is why optimal ordering criteria need to be introduced.
The goal is to sort the FW rules to reduce the number of evaluations of each individual rule, lightening the matching phase overall. Assuming that a SP is composed of N rules, each packet will be compared sequentially with the first, second, and then j-th rule until it reaches the rule that handles it. The matching phase is interrupted at this stage, and the filtering action is undertaken. If a packet matches a rule at the bottom of the list, it will be compared N times (typical catch-all rule), i.e., with all the rules in the SP. Therefore it is advisable to order the rules so that the most frequent ones are placed in the initial positions, and the less frequent ones are placed in the last positions, so each rule is associated with a weight given by its activation frequency.
The problem is not one of simple ordering since there are rules that cannot be postponed because if they were, the integrity of the SP could be altered. So, the first process to implement is to verify that the rules are strictly *disjoint*, i.e., that their intersection returns an empty set. Let then define the concept of dependent rules.

Definition 2.1 (Dependent rules). *Two filtering rules r_1 and r_2 are said to be dependent if and only if the following conditions are met:*

1. *r_1 and r_2 are not disjointed;*
2. $\Sigma_1 \neq \Sigma_2$

If definition 2.1 is satisfied, then the two rules are subject to a precedence constraint, and so their order cannot be changed.
For example, consider the SP in Table 1. From the analysis of the dependency relations between the rules in the list, it is possible to derive the precedence constraints expressed in Table 2, inferred from the dependency relations.

ID	Precedence constraints
0	$[0 \rightarrow 10]$
1	$[1 \rightarrow 8]; [1 \rightarrow 11]$
2	$\emptyset$
3	$[3 \rightarrow 7]; [3 \rightarrow 11]$
4	$\emptyset$
5	$\emptyset$
6	$\emptyset$
7	$[7 \rightarrow 11]$
8	$\emptyset$
9	$[9 \rightarrow 11]$
10	$[10 \rightarrow 11]$

Table 2
Precedence constraints derived from SP in Table 1

For example, constraint $[0 \rightarrow 10]$ is derived from that $\Sigma_0 \neq \Sigma_{10}$ and between each attribute the intersection is not empty, indeed:

- $Direction_0$ = $Direction_{10}$;
- $Protocol_{10} \subset Protocol_0$;
- $Source_0 \subset Source_{10}$, so 10.1.1.1 $\in$ 10.1.1.0/24, being 10.1.1.0/24 = $[10.1.1.0, 10.1.1.255]$;
- $Destination_0$ = $Destination_{10}$;
- $Destination\ Port_{10} \subset Destination\ Port_0$.

In any process that tends to change the order of the rules in the list, these constraints must be taken into account to maintain the integrity of the SP.

3. Related Work

Fulp et al. [5] showed how the optimal rule ordering problem of a FW, considering the integrity of the SP, has a complexity equivalent to the job scheduling problem with precedence constraints for single-processor systems. It is, thus, a problem of complexity *NP-Hard*.
The authors of [1] introduce an SP representation technique that improves packet filtering performance while maintaining the integrity of the implemented policy. However, this approach is unpromising about generic SP modification operations (e.g., simply adding a new rule), which leads to the creation of a new representation, thus making this technique underperforming in handling dynamic policies.
The dynamic sorting problem is addressed by Hamed et al. [4], who present a technique that tracks the statistical characteristics of the current network traffic, aiming to perform optimal dynamic ordering of the SP through the Optimal firewall Rule Ordering (ORO) heuristic.
Alternative modeling of the problem is provided by the authors of [10]. They introduce the RORO (Relaxed Optimal Rule Ordering) problem, addressed through a local search algorithm such as Simulated Annealing [13]. In the latter work, constraints are modeled using ZDDs (Zero-suppressed binary Decision Digrams).
Typically, the optimal rule ordering problem of a FW is expressed as a Binary Integer Programming (BIP) optimization problem, for which it has been shown that the application of Branch and Bound (BB) approaches can give an optimal solution. An alternative approach to solving the same problem, based on the use of genetic algorithms, is proposed in [8], showing that the results obtained are comparable to those of the BB technique.
Starting from this previous research, it has been considered the possibility to extend the genetic approach, exploiting its potentiality and high flexibility. The purpose is to make this technique scalable to very complex SP, reducing the space of application of the algorithm to placeholders to be placed in defined positions following the resolution of the constraints; which is obtained through a direct acyclic graph (DAG), in turn resolved using the introduction of a hierarchical ordering heuristic.

4. Solution Design

The proposed algorithm addresses an optimization problem of a constrained ordering, having as goal to minimize the number of times the rule is evaluated, through the application of a Genetic

Algorithm (GA) acting with appropriate application criteria and leading to minimizing a fitness function thus modeled:

$$fitness = \sum_{i}^{N} i f_i \tag{1}$$

where i denotes the positional index of the i-th rule, while f_i represents the frequency of its activation, typically modeled by a probability distribution function from an a priori measurement of network traffic.

4.1. Genetic algorithms background

In a GA, the set of potential solutions to the problem is represented as a population of chromosomes. For the problem under examination, the integer encoding indicative of the positional indices of each rule was chosen to correspond to the starting sorting, while the generic placeholder is encoded with a fictitious value.
The algorithm starts from a random population, which must represent different points belonging to the search space of the possible solutions.
The fitness function assigns a score to every chromosome corresponding to the running population. Such a score indicates the chromosome's goodness as a solution to the problem.
Chromosome selection is performed on the current population according to this score. Chromosomes with a higher fitness function value have a higher probability of being selected. The selection is performed using probabilistic methods and belongs to the theory of evolutionary computation [17]. The proposed application uses the selection mechanism based on the *roulette wheel* method.
In the recombination phase, chromosome pairs are extracted and paired according to the extraction order. This operation is applied in correspondence with a cut point. Each pair is decided with a certain probability (crossover rate) of whether to perform the crossover. The crossover operation exchanges the sub-sequence of the two paired individuals according to a certain cut point. If the parts to the left of the cut point are identical, then the crossover has no effect.
Once the recombination is complete, a mutation is performed, with a probability defined by the mutation rate. From the optimization process point of view, this procedure is fundamental to avoid the algorithm getting stuck in correspondence with an optimal local case. At this step, the generation is finished, and therefore the value of the fitness function obtained is calculated and compared with the previous value. The batch of chromosomes with the best-desired fitness score is then maintained.
Generations can go on indefinitely, but in practical terms, a decision has to be made about when to stop them and, in general, provide a performance assessment regarding the adoption of the stop criterion. In this sense, the authors of [17] observe how the stop condition is a hyperparameter to be adapted to the satisfaction of the faced problem solution. In the proposed approach, the stop criterion is defined by the attainment of the limiting number of generations, but further considerations will be evaluated to the possibility of using a criterion that stops the algorithm after no improvement in the fitness trend has been recorded for a fixed number of generations.

4.2. Proposed algorithm

The proposed solution consists of two phases. A new heuristic for resolving precedence constraints is defined in the first phase. In the second phase, the application of genetics enables optimal placement of constrained rules within the sequence of unconstrained rules.

4.2.1. Constraint resolution

Each precedence relation is defined as a $[head, tail]$ pair, where $head$ refers to the positional index of the rule that must precede the rule identified by the $tail$ index. Given such modeling, resolution of precedence relationships is accomplished by using a DAG. Each node in the graph represents the positional index of the generic rule, and arcs represent the precedence relations between nodes.

Heuristic 4.1 (Longest path criterion). *Let the following set of paths (depth) of the generic DAG be considered:*

$$D = \{d_0 = \begin{cases} a \\ b \\ c \\ d \\ e \end{cases}, d_1 = \begin{cases} a \\ b \\ c \end{cases}, \ldots, = d_n = \begin{cases} x \\ y \\ z \end{cases} \}$$

Since $d_1 \subset d_0$ then it is possible to drop d_1 for the longest path criterion. More complex intersections are handled, and the graphical representation is simplified while preserving the precedence constraints.

Observe the following example, which takes into account the constraints $[3 \rightarrow 7], [3 \rightarrow 11], [7 \rightarrow 11]$ extracted from Table 2. The application of the proposed criterion is shown in Table 3.

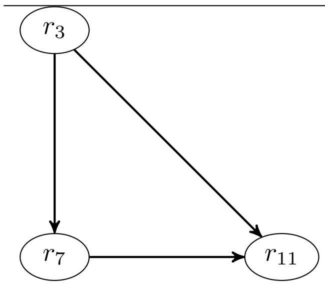
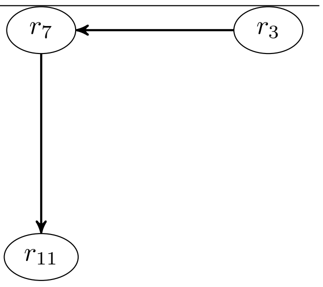
Without Heuristic 4.1	With Heuristic 4.1
	

Table 3
Heuristic 4.1 application example

Typically a DAG is resolved using topological ordering algorithms. Techniques of this type cannot find a global optimal solution, even if they are advantageous from the time complexity

point of view. Consider, for example, an algorithm Depth First Search (DFS) for the resolution of a DAG. The time complexity turns out to be equal to $O(v+e)$, where v represents the number of vertices in the graph, while e is the number of arcs in the graph. However, for sorting the DAG, the proposed algorithm combines two heuristics, noting that defining heuristic methods that lend themselves well to the problem addressed can lead to drastically reduced time complexity compared to classical methods.
The choice will be strengthened by studying the results obtained by applying the proposed methodology instead of a topological algorithm and comparing what is obtained with an exhaustive search method, which explores every possible combination that satisfies the constraints and is a possible solution. The fitness function will give the score of each solution.

Heuristic 4.2 (Hierarchical sorting criterion). *The proposed sorting criterion is defined according to the hierarchical levels obtained by the DAG construction.*

Definition 4.1 (Root nodes). *An element of the generic constraint is defined as a root node if and only if it appears exclusively in the first position.*

Observation 4.1. *Definition 4.1 remains consistent regardless of the number of constraints in which the element is root.*

By synthesizing a DAG from Table 2, the structure of the layers can be inferred. In particular:

- r_0, r_1, r_3, r_9 are root nodes, or equivalently level 0 nodes;
- r_{10}, r_8, r_7 are level 1 nodes;
- r_{11} is a level 2 node.

Observation 4.2. *The level of each node is given by its maximum depth within the graph.*

The level representation allows to efficiently infer the ordering to be imposed, to keep the precedence constraints unchanged. In particular, the respected ordering is given by the increasing order of the levels, and so a first ordering could be given by: $r_0, r_1, r_3, r_9, r_{10}, r_8, r_7, r_{11}$, which satisfies the constraints given in Table 2. However, as assumed, it would be a possible solution. The proposed solution foresees moving to the left the deepest level elements (starting from the right) until a rule that binds the generic element to be moved is reached or until a rule with a greater weight is met. This allows an order that approaches the global optimum concerning the problem being faced.
At this point, a placeholder will be associated with each constrained rule.

4.2.2. Genetic Algorithm over placeholders

First, the unconstrained rules are sorted according to their fitness.
The genetic algorithm is then applied to the constrained rules to determine the correct placement of the placeholders within the ordering and the space that must be maintained between two contiguous intervals. This drastically reduces the computational effort of the algorithm since, in general, the constrained rules are in lower cardinality than the unconstrained ones.

For the application of the genetic algorithm, the generic chromosome is composed of the untied rules, where each gene is associated with the positional index of the rule, and C placeholders, where C is the number of constrained rules.
Initially, all placeholders are encoded by a fictitious value and then regain the correct integer index relative to the rule position at the end of the placement procedure performed by the GA. Each placeholder must correspond to a position, so a new draw is made if the generic position has already been assigned. Once the positions of the placeholders have been obtained, the unbound rules are inserted in the empty spaces. Finally, the fitness of the chromosome is calculated.

Definition 4.2 (Crossover). *The crossover between two chromosomes A and B consists in exchanging the positions of some placeholders of A with those of some placeholders of B.*
This is done in the following way:

1. *Let A and B be two chromosomes to be crossovered. It is randomly determined how many placeholders to swap and what they should be;*
2. *For each exchange, the placeholder of A will be assigned the position of the placeholder of B and vice versa.*

If the position assigned to a placeholder already belongs to another placeholder, the exchange is not confirmed, so the crossover is canceled.

Definition 4.3 (Mutation). *A mutation consists of assigning a new position to a placeholder.*

1. *For each chromosome in the population, it is randomly determined how many placeholders to mutate and what those placeholders should be;*
2. *For each placeholder to be changed, the new position is drawn and applied.*

In some cases, the mutation is not applied. If another constraint already occupies the drawn position, the mutation has no effect. Moreover, the same constraint may be drawn several times and, therefore, it will undergo several mutations and keep only the last mutation undergone. Unlike the standard approach, the mutation operation is applied without using a mutation rate.

5. Results and performance analysis

5.1. Constraints resolution

Starting from the example of ten constrained rules proposed by [8], whose representation is shown in table 4, a comparison is proposed in the application of the combination of the heuristics 4.1 and 4.2 proposed by the present work, against the application of a topological sorting (TS) algorithm of DFS type. The obtained results are compared with the best solution, found through an exhaustive search algorithm, which explores all possible solutions that satisfy the constraints. The activation frequencies were modeled using a probability distribution function *Zipf*, which models the *time of interarrival* of packets for ordinary network traffic as shown by the authors of [18].
The exhaustive search algorithm finds 252 possible solutions that satisfy the constraints. The

topological sorting algorithm obtains a solution that ranks 167th, while the algorithm proposed by this work ranks 31th.
The results are shown in Table 5.
The best solution, obtained through the exhaustive search algorithm, is reported; the ε difference in fitness from the best solution is measured for the other solutions.
Algorithm time complexity analyses were performed on an Ubuntu Virtual Machine, Dual-Core, 2GB RAM, hosted with VMWare from a Windows 11 machine, Intel Core i7-8650 CPU @ 1.90 GHz.

ID	Precedence Constraints
1	$[1 \rightarrow 2]; [1 \rightarrow 5]$
2	$[2 \rightarrow 6]; [2 \rightarrow 7]$
3	$[3 \rightarrow 4]$
4	$[4 \rightarrow 6]; [4 \rightarrow 9]$
5	$[5 \rightarrow 7]$
6	$[6 \rightarrow 8]; [6 \rightarrow 9]$
7	$[7 \rightarrow 10]$
8	$[8 \rightarrow 10]$
9	$[9 \rightarrow 10]$

Table 4
DAG Precedence Relationships in [8]

Procedure	Fitness	ε	#Ranking/252
Best solution	2074	-	1/252
Proposed approach	2203	127	31/252
DFS	2506	432	167/252

Table 5
Constraint resolution results

5.2. Time Complexity

In this section, the results of the algorithm's performances in its entirety are shown. The aim is to study the fitness trend to the increase of the number of generations, relating such analysis to the times of execution necessary to the attainment of the solution held optimal by the algorithm. The fitness trend in the three experiments shows that as the complexity of the SP increases, the algorithm requires more generations to converge to the optimal solution. However, in all experiments, the convergence is fast and requires less than 25% of the total generations.
Corresponding to the average case, i. e. a ruleset composed of 50 rules, the execution time shows a linear growth. This is confirmed using a model that approximates the trend of this curve, as shown in Figure 2. Let now consider the most complex case among those proposed. The following observations can be deduced.
First of all, for a number of generations $\eta = 2000$, the algorithm takes about $600 \cdot 10^{-3}$ seconds (Figure 1) to find the solution considered as optimal, and this denotes its goodness from the speed-of-convergence point of view. Even in this case, the trend of execution times is almost

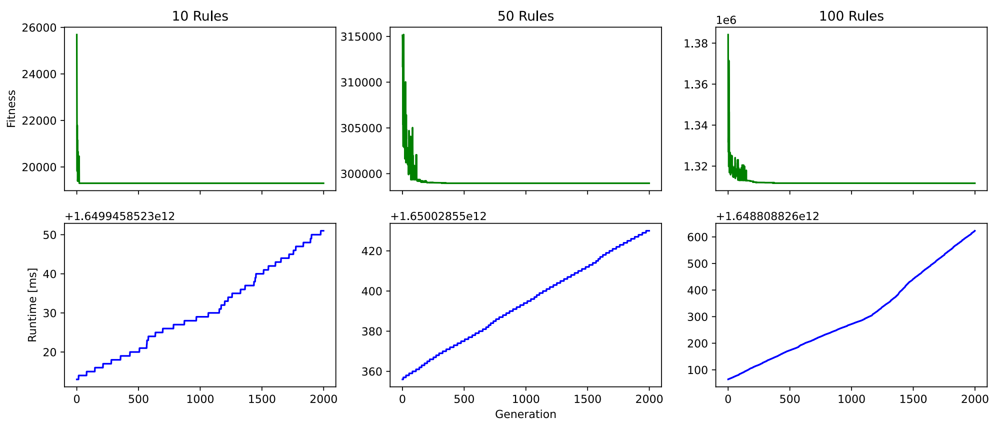


Figure 1: Fitness/Runtime over No. Generations for different SP Complexity

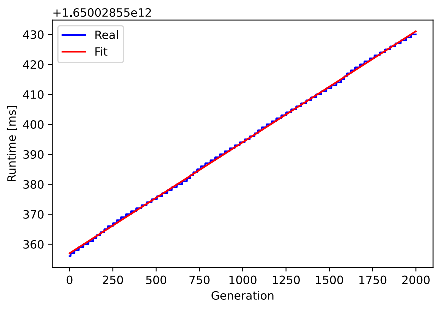


Figure 2: Linear Time Complexity for a SP of 50 Rules

linear.
It has been experimentally observed that, on many repeated experiments on a ruleset of 100 rules, the algorithm finds the optimal solution in correspondence of a generation belonging to the narrow circle of generation 500. It can be assumed that the fitting would be perfectly linear by adopting the following stopping criterion: stop the algorithm if no fitness improvements are obtained for a certain number κ of generations. In this case, it has been placed $\kappa = 300$. The result obtained is shown in Figure 3 and allows observing that the time required to reach the solution becomes equal to $220e{-}3$ seconds, reducing the time by about 63% compared to the previously described case.
This demonstrates how the variation of the adopted criterion of stop (that does not alter the correct execution) allows the algorithm to be executed in linear and reduced time also for very complex SP.

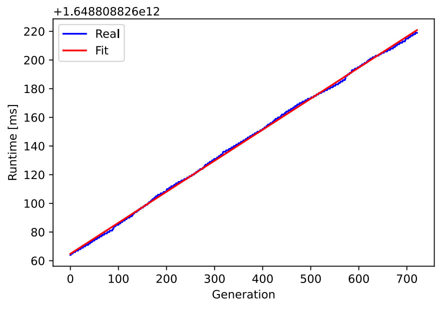


Figure 3: Linear Time Complexity for a SP of 100 Rules

6. Conclusions and future Work

A novel genetic approach has been introduced to solve the constrained ordering of rules contained in an SP. The results show how the proposed solution is well suited to solve the problem that is addressed in the various case studies.
The heuristics introduced are more efficient than the topological sorting algorithms. Moreover, they have some room for improvement that may make them acquire a uniform behavior corresponding to rules with particularly dominant activation frequencies. In this way, it will be possible to demonstrate the robustness of the proposed solution, varying the hyperparameters of the distribution used to model the frequencies of activation.
The proposed contributions, especially regarding the second phase of the algorithm, make it globally performant, as can be seen from the analysis of its time complexity.
Future developments will focus on benchmarking with the most recent approaches in the literature to validate the efficiency of the proposed solution, especially in cases where more sophisticated filtering policies need to be managed.

Acknowledgments

This work was partially supported by the Fondo Europeo di Sviluppo Regionale Puglia Programma Operativo Regionale (POR) Puglia 2014–2020 Axis I–Specific Objective 1a–Action 1.1 (Research and Development) – Project Title: CyberSecurity and Security Operation Center (SOC) Product Suite by BV TECH S.p.A., under Grant CUP/CIG B93G18000040007.

References

[1] E. W. Fulp, S. J. Tarsa, Trie-based policy representations for network firewalls, in: 10th IEEE Symposium on Computers and Communications (ISCC'05), IEEE, 2005, pp. 434–441.
[2] E. Al-Shaer, Automated firewall analytics: Design, configuration and optimization, Springer, 2014. doi:10.1007/978-3-319-10371-6.
[3] K. Golnabi, R. K. Min, L. Khan, E. Al-Shaer, Analysis of firewall policy rules using data

mining techniques, in: 2006 IEEE/IFIP Network Operations and Management Symposium NOMS 2006, IEEE, 2006, pp. 305–315.

[4] H. Hamed, E. Al-Shaer, Dynamic rule-ordering optimization for high-speed firewall filtering, in: Proceedings of the 2006 ACM Symposium on Information, computer and communications security, 2006, pp. 332–342.

[5] E. W. Fulp, Optimization of network firewall policies using directed acyclical graphs, in: Proceedings of the IEEE Internet management conference, Citeseer, 2005.

[6] N. B. Neji, A. Bouhoula, Towards safe and optimal filtering rule reordering for complex packet filters, in: 2011 5th International Conference on Network and System Security, IEEE, 2011, pp. 153–160.

[7] W. Wang, H. Chen, J. Chen, B. Liu, Firewall rule ordering based on statistical model, in: 2009 International Conference on Computer Engineering and Technology, volume 2, IEEE, 2009, pp. 185–188.

[8] E.-S. M. El-Alfy, A heuristic approach for firewall policy optimization, in: The 9th International Conference on Advanced Communication Technology, volume 3, IEEE, 2007, pp. 1782–1787.

[9] R. Mohan, A. Yazidi, B. Feng, B. J. Oommen, Dynamic ordering of firewall rules using a novel swapping window-based paradigm, in: Proceedings of the 6th International Conference on Communication and Network Security, 2016, pp. 11–20.

[10] T. Harada, K. Tanaka, K. Mikawa, A heuristic algorithm for relaxed optimal rule ordering problem, in: 2018 2nd Cyber Security in Networking Conference (CSNet), IEEE, 2018, pp. 1–8.

[11] T. Fuchino, T. Harada, K. Tanaka, K. Mikawa, Acceleration of packet classification using adjacency list of rules, in: 2019 28th International Conference on Computer Communication and Networks (ICCCN), IEEE, 2019, pp. 1–9.

[12] A. Singh, D. Singh, A. K. Singh, H. Pandey, P. Vashist, Security through optimization techniques of firewall rule sets, in: 2020 International Conference on Computation, Automation and Knowledge Management (ICCAKM), IEEE, 2020, pp. 452–455.

[13] T. Harada, K. Tanaka, K. Mikawa, Simulated annealing method for relaxed optimal rule ordering, IEICE Transactions on Information and Systems 103 (2020) 509–515.

[14] T. Harada, K. Tanaka, R. Ogasawara, K. Mikawa, A rule reordering method via pairing dependent rules, in: 2020 IEEE Conference on Communications and Network Security (CNS), IEEE, 2020, pp. 1–9.

[15] T. Fuchino, T. Harada, K. Tanaka, Accelerating packet classification via direct dependent rules, in: 2021 12th International Conference on Network of the Future (NoF), IEEE, 2021, pp. 1–8.

[16] Z. Nopiah, M. Khairir, S. Abdullah, M. Baharin, A. Arifin, Time complexity analysis of the genetic algorithm clustering method, in: Proceedings of the 9th WSEAS international conference on signal processing, robotics and automation, ISPRA, volume 10, 2010, pp. 171–176.

[17] A. Lambora, K. Gupta, K. Chopra, Genetic algorithm-a literature review, in: 2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon), IEEE, 2019, pp. 380–384.

[18] A. A. Naghash, A. M. ABDOLLAHI, A method for modeling and generating normal

network traffic based on the features of length and arrival time of packets using the zipf's law (2016).