Computing and Informatics, Vol. 32, 2013, 1312–1327

NETWORK FIREWALL USING ARTIFICIAL NEURAL NETWORKS

Kristián Valentín, Michal Malý

Department of Applied Informatics
Faculty of Mathematics, Physics and Informatics
Comenius University
Mlynská dolina, Bratislava, Slovakia
e-mail: {valentin, maly}@fmph.uniba.sk

Abstract. Today's most common firewalls are mostly rule-based. Their knowledge consists of a set of rules upon which they process received packets. They cannot do anything they have not been explicitly configured to do. This makes the system more straightforward to set up, but less flexible and less adaptive to changing circumstances. We will investigate a network firewall whose rule-base we will try to model using an artificial neural network, more specifically using a multi-layer perceptron (MLP) trained by the back-propagation algorithm. The training data are acquired from the network and we consider two possible scenarios. In Scenario 1, the user has no firewall available and the policy is deduced from the existing traffic in the network which is considered to be legitimate. In Scenario 2, the learning module is placed behind the existing firewall (or firewalls) in order to learn their behavior. In both cases, all traffic, which is recorded, contains only positive examples; however, a direct training of a MLP from a set of positive examples is impossible. We solved this problem using a synthetic generation of negative examples which led to successful learning.

Keywords: Network firewall, artificial neural networks, computer security

Mathematics Subject Classification 2010: 68T05 (Learning and adaptive systems), 62H30 (Classification and discrimination, cluster analysis)

1 INTRODUCTION

Internet security becomes more and more important along with spreading the Internet access over the world. One of its key components is a firewall. A network firewall is a system or group of systems that enforces an access control policy between two networks. Most firewalls are rule-based; this means that responsible persons have to enter the rules manually, using their knowledge of the network architecture and according to the users' requirements.

2 PURPOSE AND METHOD

2.1 Scenarios

We investigate the use of artificial neural network for two main practical purposes (scenarios):

1. Is it possible to build a firewall if only a sample of "usual traffic" is available? As mentioned above, the persons have to obtain considerable knowledge prior to setting the firewall; the possibility of an automatic firewall setup without prior knowledge can help the laymen.
2. Having a fully set up (rule-based) firewall system, is it possible to "copy" the desired behavior (to mimic the rule-set)? In a big corporate environment, a number of firewalls and other network appliances, usually connected into a hierarchy, can influence the traffic between the user and the Internet. Due to organizational matters, such as change of network architecture or a hardware failure, it may be necessary to replace part of the system; however a direct export of the firewall rules may not be available (e.g., interfering rules; firewalls having incompatible rule formats; non-existing program to "merge" rules from two firewalls). The possibility to copy the behavior can save a considerable amount of laborious work.

For both scenarios we considered only the input firewall, i.e., a firewall handling the inbound traffic. The output firewall can be constructed analogically, using the outbound traffic.

2.2 Method

We decided to use a connectionist approach – a multi-layer perceptron (MLP) – to be the heart of the AI firewall system. Its task will be to correctly classify the "right" and "wrong" packets presented.

Connectionism is a natural complement to the symbolic approach, which in our case is represented by the rule-based firewall. Using the connectionist approach is supposed to be very useful mainly in Scenario 2 discussed above, because there is no dependency on the symbolic representation (format of the firewall rules).

MLP will learn from a (presumably) attack-free and failure-free traffic data set. This data set will contain packets described by some of their basic identification signs, such as the source and destination IP addresses and ports, and the protocol. Afterward, the firewall should correctly distinguish between the learned traffic and a potential attack, although only positive examples were presented.

The reason why only positive examples are available is caused by the physical position of the learning system. In Scenario 1, our module stands between the computer and the Internet; all traffic is considered legitimate. In Scenario 2, our module is placed behind all other existing firewalls (whose behavior we want to copy). These firewalls allow only legitimate traffic to pass. All dropped or rejected packets will be hidden from our module.[1]

3 FIREWALL

A firewall is a software or a hardware system which filters network traffic according to a pre-defined policy. For this, it is implanted in a crucial place of the operating system or in the network architecture's key positions. There, it analyzes what goes through and applies actions upon a rule set.

Usually, there are 3 basic actions the firewall can do with a packet. These actions are part of the rule definitions. Each rule defines one action. The actions have quite self-explanatory names.

ALLOW (or ACCEPT) action permits a packet to pass trough the firewall in or out (depending on the traffic direction). This is a trivial action because this would be the case if the firewall were not present.

DENY (or DROP) drops a packet without any notification to the sender. It is the most useful action.

REJECT action, similarly to DENY, prohibits a packet from passing, but with a notification[2] sent back to the sender.

Firewalls need to be placed in external perimeters but also inside a network for secure segmentation of data. Internal deployment of firewalls is a relatively new best practice. It is largely driven by the tendency that we can no more differentiate a tangible, reliable network border between the trusted internal and untrusted external network traffic. One has to take into consideration possible security issues arising within the trusted network as well. A firewall separating for example two company departments could slow down or even stop the spreading of spam, viruses, or other threats.

[1] Although, in some cases, it is possible to get knowledge about the rejected packets, e.g., rejected outbound packets.

[2] The notification message is an ICMP (Internet Control Message Protocol) destination-unreachable packet, defined in RFC 792. The ICMP packet is sent only when permitted, e.g., when it is not going to or from a broadcast address.

3.1 Basic Packet Filter

Basic packet firewall should be able to process incoming or outgoing packets upon a pre-configured policy. The policy is represented by a set of rules which defines what should be done with a particular packet: either it should be allowed/passed through, or dropped (with or without a notification). The core task is to encode that set of rules into the neural network so that this functionality would be preserved.

In order to properly mimic the functionality of the firewall, we need to build a model of the rules in the firewall's rule-base. This would be the task for the MLP. Possible alternative methods include: Support Vector Machines (SVMs), clustering algorithms[3], generic programming, and other methods.

3.2 Learning from Positive Examples

Gold's theorem [9] can be summarized as follows: *Without explicit error correction, the rules of a logical system with a sufficient structural complexity could not be inductively discovered, even in theory.* More specifically: A class of languages is unlearnable if only positive learning examples are available.

The implication for the learning MLP to model the firewall is straightforward. We have only a positive training set, $\mathcal{P}$, and it is impossible to directly learn from such a training set[4]. Therefore, we must provide an additional assumption (constraint).

Consider a set of all negative examples, $\mathcal{N}$. It holds that the complement of positive examples contains all negative examples: $\mathcal{P}^C \subseteq \mathcal{N}$. If the positive example set is "fair", we can approximate $\mathcal{N}$ by $\mathcal{P}^C$, so:

$$\mathcal{P}^C \approx \mathcal{N}$$

where

$$\mathcal{P}^C = \mathcal{U} - \mathcal{P}.$$

The set of all potential examples (packets), $\mathcal{U}$, is finite, when we consider the fact that the IP addresses, ports, and protocols also come from finite ranges. In theory, it would be possible to generate all packets from $\mathcal{P}^C$, although it would be computationally very hard because of its enormous size.

[3] In combination with supervised learning to fine-tune the covering of rules by the cluster.

[4] For a demonstration of this, we tried to train a neural network using only a set of positive examples. As expected, the network overgeneralized its training data and provided positive output (ACCEPT) for every input. We also tried to start with the weights having a zero value in order to shift network response to the negative values; however, this was also unsuccessful.

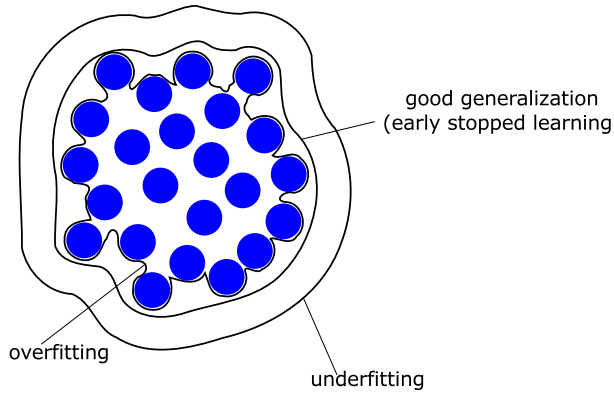


Figure 1. Illustration of the generalization of the set of positive examples $\mathcal{P}$ (dots). Everything except the dots is the complement $\mathcal{P}^C$; however, the ideal boundary of the positive examples is to be found by generalization. Underlearning, correct (early stopped) learning, and overlearning is shown. For correct generalization, also some examples not present in $\mathcal{P}$ are to be considered positive (white space within the boundary), what corresponds to the set $\mathcal{N} \setminus \mathcal{P}^C$.

However, it is not required to generate all packets from $\mathcal{P}^C$. During the training procedure, only a set of negative packets of a pre-defined size is needed; therefore, it is possible to generate packets "on-line": When we need a packet from $\mathcal{N}$, we generate a random packet and check whether it belongs to $\mathcal{P}$. If not, the packet belongs to $\mathcal{P}^C$ and it is provided, in the other case, the procedure is repeated until a packet from $\mathcal{P}^C$ is obtained[5].

It is not necessary to memorize the packets. During training, always a new random packet from $\mathcal{P}$ is provided. This also helps to avoid accidental learning of a positive packet as a negative example: Suppose a random packet from $\mathcal{P}^C$ is generated; but in fact, this packet is legitimate, but was not spotted during the data acquisition. Therefore, it does not belong to $\mathcal{N}$ and its repeated use during training could lead to erroneous learning. Using the packet only once causes only small damage.

The additional assumption is that we want to find the best generalization of the boundary between $\mathcal{P}$ and $\mathcal{P}^C$ in terms of ANN early stopping (for illustration see Figure 1). This assumption makes the necessary inductive bias to overcome the original indetermination.

[5] Because size of $\mathcal{P}$ is small relative to the all possible packet space $\mathcal{U}$, it is rarely necessary to do a new iteration.

4 RELATED WORK

According to[11], recent network anomaly detection systems such as NIDES [1], ADAM [3], and SNORT[6] plug-in SPADE [18] can detect network level attacks (port scans, denial of service (DoS) attacks), but cannot detect application specific attacks, such as buffer overflow exploits.

Regarding the use of artificial intelligence, it *is playing an increasingly important role in network management. In particular, research in the area of intrusion detection relies extensively on AI techniques to design, implement, and enhance security monitoring systems* [8].

ANNs have been utilized to identify both misuse and anomalous patterns [7, 6, 15, 5, 12]. Studies have shown that the current anomaly detecting IDSs are failing to reach adequate detection rate while having few false alarms [10, 2].

The noteworthy techniques from AI that could provide a better solution to shortcomings of IDS and IDPS implementations are ANNs, SVMs, and Multivariate Adaptive Regression Splines (MARS), and the collection of different soft computing techniques[7] [13]. They are widely used due to their generalization capabilities of known attacks to recognize also the unknown.

In the future, more hybrid systems could be expected to arise. IDS will keep a certain, well defined level of security and an adaptive AI system will make it more flexible to new challenges.

Recent research proposed various types of ANNs to learn firewall rules, however, an implementation of the packet filter firewall is not available (reported only as a future work in [17]).

5 IMPLEMENTATION

5.1 Data Sets

For training and testing, enough examples need to be synthetically generated or captured from the real network (e.g., using `tcpdump` program). We used generated data, because there is much more control in terms of quantity and quality of the created packets. Note that this method will usually just approximate the real traffic flow and therefore the created model will not be as accurate as it could be when real data would have been used. Our goal was to diminish this effect as much as possible and to build a robust model suitable also for a real-world environment deployment.

[6] Snort® is an open source network intrusion prevention and detection system (IDS/IPS) developed by Sourcefire. According to a recent survey [16], it is the most favorite IDS in 2006.

[7] Unlike hard computing, soft computing it is tolerant of imprecision, uncertainty, partial truth, and approximation.

IP address/mask	Port	Protocol	Action
192.168.1.0/255.255.255.0	any	UDP	ALLOW
0.0.0.0/0.0.0.0	161	UDP	ALLOW
0.0.0.0/0.0.0.0	162	UDP	ALLOW
0.0.0.0/0.0.0.0	3306	TCP	REJECT
0.0.0.0/0.0.0.0	22	TCP	ALLOW
0.0.0.0/0.0.0.0	21	TCP	REJECT
0.0.0.0/0.0.0.0	53	UDP	ALLOW
0.0.0.0/0.0.0.0	443	TCP	ALLOW
0.0.0.0/0.0.0.0	80	TCP	ALLOW
192.168.1.5/255.255.255.255	520	UDP	ALLOW
10.3.3.0/255.255.255.0	8080	TCP	ALLOW
192.168.2.0/255.255.255.0	any	UDP	ALLOW
92.154.54.8/255.255.255.255	any	UDP	ALLOW
91.154.55.0/255.255.255.0	any	UDP	ALLOW

Table 1. Table of rules used to generate traffic coming through a firewall. The goal of the neural network was to infer those rules by learning from generated traffic. Note that these are only rules we can directly know about; rules with DROP action are hidden. Rules have zero IP address and mask matches all IP addresses.

One training example, which represents an abstract form of a packet's header (it does not contain all the information a real packet usually has), consisted of these attributes:

- IP address: IPv4 format, e.g., `192.168.1.1`
- port: an integer number between 0-65535
- protocol: the most common – TCP and UDP
- action: ALLOW, REJECT, and DENY.

For generating the data we used similar list of rules to those used in classic firewalls. The list of rules contained a couple of arbitrary rules with ALLOW and REJECT actions (see Table 1). The complementary negative packets (from $\mathcal{P}^C$) had DENY action set in its attributes. The action serves as the class of the data we want the packet to be classified to, i.e., as the target value for training the MLP network.

In principle, there are two ways how to generate packets using the list of rules.

1. A random rule is chosen from the list and a random packet is generated to meet the criteria of this rule. For example the rule will be: `192.168.1.0/24 TCP any ALLOW`, which says that any destination TCP port is allowed from a source IP address belonging to a particular network subset (the network address is `192.168.1.0` and the subnet mask has 24 bits, e.g., `255.255.255.0`). The random packet generated in the next step will be for example `192.168.1.18 TCP 80`.

2. A random packet is generated and then checked against the list of rules, if it fits to some or more of them. When no rule is found, the generated packet is discarded and the process is repeated.

The first approach is computationally faster because creating a random packet upon a rule is trivial. However, it will not often lead to fully real-like data, i.e., it could give the network too many hints about the original rules, but it could provide enough approximation for the experimental purposes.

The second approach can be considered more plausible in terms of its approximation of a real traffic, but it is computationally exhausting. The probability that a randomly generated packet could have been generated by one of the rules (i.e., that the packet will "fall" into some of the rule) is very small. In our case the number of unique packets is $2^{32} \times 2^{16} \times 2 \times 3 \approx 1.7 \times 10^{15}$.

We decided to use the first approach because we had limited computational resources. We chose binary coding because it is closely related to subnet coding. Subnet masks are usually written in the form of 2^n. Despite its disadvantages[8], we expect that this representation is the best possible. We maintain that some kind of continuous coding often used for training the MLP (and also possible in this case) is not very suitable in this domain.

5.2 MLP Architecture

The input IPv4 packet consisted of an IP address (32 bits), port (16 bits), and protocol (1 bit) which gives the total of 49 bits. Because of the binary coding, we used 49 input neurons (see Figure 2). The number of hidden layers and also the number of neurons in those layers has to be found.

An associated output (target value) can be one of the following actions: ALLOW, REJECT, and DENY. Here, we used a one-hot coding, i.e., one neuron is responsible for each of the possible actions (classes). Hence, we used three output neurons with softmax activation function.

The process of searching the optimal number of hidden layers or the number of neurons in those layers is not a trivial task. However, more-or-less accurate lower bounds can be computed, e.g., see [4]; in this case we would like to preserve some capacity. Hence, we will gradually raise the number of neurons in the hidden layer, or potentially also the number of hidden layers, to achieve the best result while respecting Occam's razor as mentioned before.

The weights will be randomly generated from the uniform distribution[9] from the interval $[-0.1, 0.1]$.

[8] For example, the fact that two successive numbers could differ in more than one bit could lead to problems when learning an "interval" rule. In this case, a code like Gray code, which has the property that successive numbers differ in exactly one bit, could be used for this purpose.

[9] All values from the distribution are equally possible.

To sum up:

- input: binary, IP+port+protocol (TCP | UDP), 32+16+1 = 49 bits (neurons)
- hidden layers: to be found
- weights: uniformly generated from the interval $[-0.1, 0.1]$
- output: three neurons (ALLOW, REJECT, DENY),

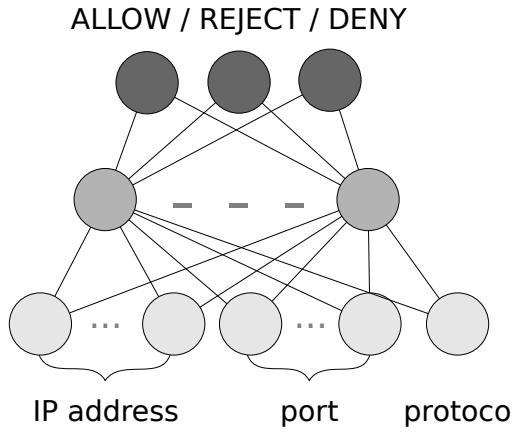


Figure 2. MLP Architecture

5.3 Output Discretization

Because we use a logistic activation function – the sigmoid – which is continuous, and discrete target categories (ALLOW, DENY, and REJECT), discretization of the MLP's output is needed. Afterward, we can easily interpret the MLP's output as one of tree categories, i.e., firewall's actions. We will use the following discretization formula:

$$\text{discretize}(x) \begin{cases} 1 & x > 0.55 \\ \text{random}(0,1) & x \in [0.45, 0.55] \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Using a random value in the middle range of the possible values is usually better then a fixed value. When the results will not be adequate, the formula (1) can be rewritten into:

$$\text{discretize}(x) \begin{cases} 1 & \text{if } x > k \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where k is adjusted to the highest value possible while keeping the highest classification rate.

When the discretized output of the MLP, either using criteria (1) or (2), will not code any of the categories, we will use the DENY rule. That situation would be, we think in most cases, when the output will be `000`, i.e., all three output neurons will not have been activated enough. The DENY action has been chosen because it is always better to drop a malicious packet than to pass it, i.e., false positive rate will be lowered at the expense of higher number of false negatives.

5.4 Training

The training phase is determined by selecting the learning algorithm and setting its parameters. The decision is usually domain- or task-dependent, but there are some standard options which can be generally used. Parameters include the activation function, learning constant, and other supplying methods for improving learning.

We chose the error-driven back-propagation (BP) learning algorithm [14] with sigmoid activation function. Sigmoid logistic function is useful because of its suitable range (from 0 to 1). We also used the momentum method to help to speed up the convergence and to better avoid local minima.

This is the most important information for the training process:

- learning method: error-driven back-propagation
- activation function: sigmoid $f(x) = 1/(1 + \exp(-x))$
- additional method: momentum ($\Delta w(t+1) = -\alpha \frac{\partial E}{\partial w} + \mu \Delta w(t)$).

To avoid overfitting and to preserve certain level of generalization we used cross-validation. We will randomly split the training data into two disjoint sets. The first set, 80 % of the data, will be used for training, the other, 20 % of the data, will be used for validation purposes. The validation served as a stopping criterion for the learning process, i.e., when there was a very small change in the classification error on the validation set between two successive epochs the learning was stopped.

Before each epoch, we generated negative examples for both the training and validation sets. We used four times more negative examples than the number of examples in the training set (i.e., the positive vs. negative examples ratio is 20:80). The training set prefers the negative examples to obtain a better coverage of the negative examples. We observed best convergence using the 20:80 ratio. In case of the validation set, we used the 50:50 ratio, to give equal weights for both negative and positive examples. Every generated negative example has only a one-epoch life span.

The learning process stops when the error on validation set stabilizes – when the difference between the previous and the current errors divided by the current error is lower than some small constant. In our case, this constant was set to 0.1 %.

5.5 Testing

Testing was done using a standard method. The testing set was presented to the trained network. Then, the output of the network was compared to the target value for each training example. After this process, the number of correct and wrong answers is summed. There are two types of wrong answers: *false positive* and *false negative* categories. The former is when the target response is ALLOW (i.e., a positive response in general) and the network's outputs DENY or REJECT; and vice versa, the latter represents the situation when the target response is DENY or REJECT (i.e., a negative response in general) and the network actually says ALLOW.

These statistics are useful when analyzing the results and deciding about the changes in the architecture or the learning process. To properly test the trained MLP we needed to solve a similar problem as during training, namely lack of negative examples. We had only the same captured/generated packets as before, just we had split those data into training and testing set. Again, the solution is similar to that used during training; negative examples were generated from $\mathcal{P}^C$ set.

One part of testing data set is taken from the positive examples set, the second part is generated from $\mathcal{P}^C$. How large should the testing set be? An ideal testing set will include all the possible packets from $\mathcal{N}$, but as mentioned above, that would be very difficult to accomplish.

To overcome the lack of a complete set of negative examples $\mathcal{N}$ we decided to use port scan. This scan can test how could the firewall be trusted in the real-world environment. Its principle is very simple: the firewall is tested with generated packets having an arbitrary destination IP address and every possible destination port number (i.e., from 0 to 655535) using TCP and UDP protocols. We chose one random and one known IP address and analyzed which ports are open and which are closed.

6 RESULTS

The first step was to find a suitable network architecture for the firewall. We started with one hidden layer containing four neurons. Then, step-by-step, we added more and more neurons while performing learning and testing phase to compute the classification rate. The number of neurons was the only changing variable, other learning parameters were fixed to some usual values. The learning rate was set to 0.05, momentum (μ) to 0.1.

Using one hidden layer was sufficient as we take into account the Occam's razor and kept the model as simple as possible. The number of hidden neurons was set to 13.

A good trade-off between speed and quality of learning seemed to be 20 000 examples. The training data sets consisted of combination of negative and positive examples in 80:20 ratio. During the training, a new set of negative examples was

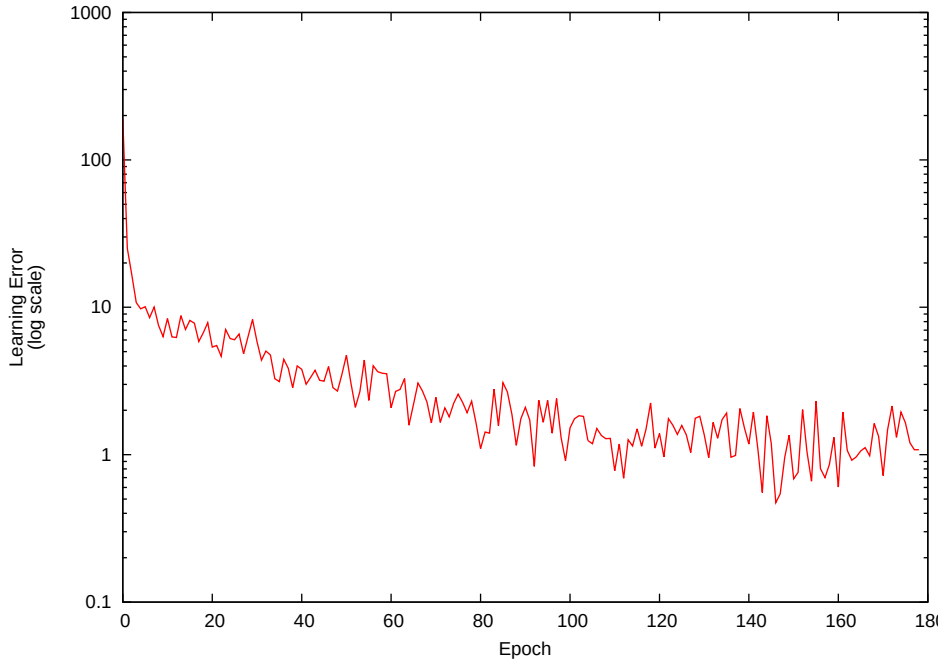


Figure 3. Total error in the validation set during learning

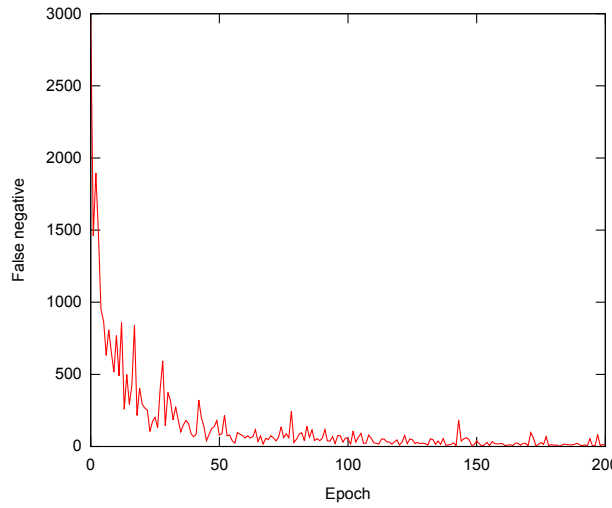


Figure 4. The number of false negative classifications in the validation set during learning

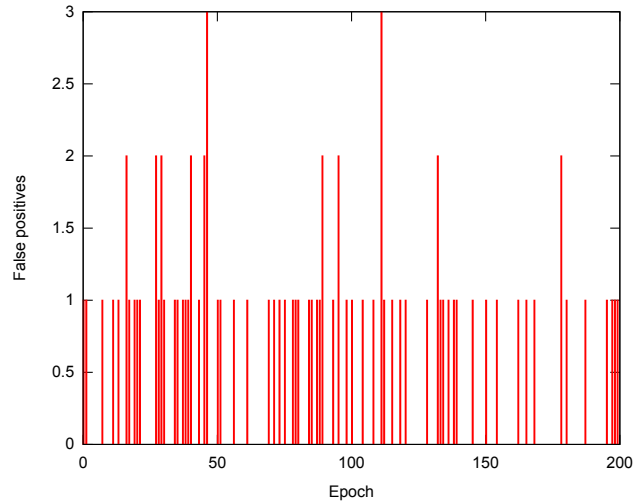


Figure 5. The number of false positive classifications on the validation set during learning. Missing lines mean that there was no false positive classification for that epoch.

generated in every epoch for the training set (80:20 ratio), and also for the validation set (50:50 ratio).

The first discretization method, presented in Section 5.3, was not quite successful during testing. When we used the port scan to check the open ports, there were too many of them. The latter formula resulted in higher success in port scanning with $k > 0.9$. Henceforth, the default-drop-policy was strengthened and fewer ports were opened while the classification ratio remained the same.

Classification on the testing set was 99.79 %. The progress of learning error on validation set is shown in Figure 3. The number of false negative classifications on the same set decreases during training, as illustrated in Figure 4. The number of false positives oscillated between very small numbers as a result of the new negative examples being generated, as seen in Figure 5.

7 FURTHER WORK

In our model of the firewall, we used an MLP network for both UDP and TCP protocols. Because the rules (in terms of classic firewalls) for those protocols are usually independent, using two separate MLP networks could improve the model. This could eliminate the source of error arising from the network generalization with respect to those two independent protocols.

Similarly, another set of MLP networks can be used for the output firewall. Moreover, in case of output firewall and considering Scenario 2, it is possible to capture the ICMP responses and infer the REJECT rules of the original firewall(s).

8 CONCLUSION

The main goal was to build an adaptive firewall system which could be trained just from the set of positive examples (packets which should be ALLOWed by the firewall). We have considered two scenarios. In Scenario 1, the rule-set should be deduced from the firewall-free network. The task of Scenario 2 was to mimic the behavior of an existing firewall or set of firewalls. In both scenarios, only the positive examples are extracted.

The Gold's theorem suggests and also our experiments showed that a direct learning from just the positive examples is impossible. We developed a solution where the negative examples are generated with respect to the positive ones we already have. This enabled the system to learn with a fine classification accuracy (99.8 %). As an important contribution we consider the ability of our model to learn just from the set of positive examples when viewed from the black-box perspective.

A core part of the system was presented; however, a complete production system would need some additional modules like GUI, notification system, user management, etc., which would enable the deployment in the real-world environment. The current state of the research offers a stateless firewall with additional tools for generating and converting the input data. In addition to the stateless firewall, the statefull firewall can be obtained relatively easily by implementing some of the deterministic finite automaton models as the TCP protocol is standardized.

Acknowledgment

This work has been supported by the Slovak Grant Agency for Science (projects No. 1/0439/11 and 2/0019/10).

REFERENCES

[1] Anderson, D.—Lunt, T. F.—Javitz, H.—Tamaru, A.—Valdes, A.: Detecting Unusual Program Behavior Using the Statistical Component of the Next-Generation Intrusion Detection Expert System (NIDES). Computer Science Laboratory SRI-CSL, 1996, pp. 95–06, 1995.

[2] Axelsson, S.: The Base-Rate Fallacy and the Difficulty of Intrusion Detection. ACM Trans. Inf. Syst. Secur., Vol. 3, 2000, No. 3, pp. 186–205.

[3] Barbará, D.—Couto, J.—Jajodia, S.—Popyack, L.—Wu, N.: ADAM: Detecting Intrusions by Data Mining. In Proceedings of the IEEE Workshop on Information Assurance and Security, West Point, NY 2001, pp. 11–16.

[4] Bartlett, P. L.—Maass, W.: Vapnik-Chervonenkis Dimension of Neural Networks. In Michael A. Arbib (Ed.): The Handbook of Brain Theory and Neural Networks, MIT Press, Cambridge, Massachusetts 2003, pp. 1188–1192.

[5] Cannady, J.: Artificial Neural Networks for Misuse Detection. National Information Systems Security Conference 1998, pp. 368–81.

[6] Debar, H.—Becke, B.—Siboni, D.: A Neural Network Component for an Intrusion Detection System. In Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy 1992.

[7] Debar, H.—Dorizzi, B.: An Application of a Recurrent Network to an Intrusion Detection System. In Proceedings of the International Joint Conference on Neural Networks 1992, pp. 78–83.

[8] Gagnon, F.—Esfandiari, B.: Using Artificial Intelligence for Intrusion Detection. In Proceeding of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering, Amsterdam, The Netherlands 2007, pp. 295–306.

[9] Gold, E. M.: Language Identification in the Limit. Information and Control, Vol. 10, 1967, No. 5, pp. 447–474.

[10] Lazarevic, A.—Ertoz, L.—Kumar, V.—Ozgur, A.—Srivastava, J.: A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection. In Proceedings of the Third SIAM International Conference on Data Mining 2003, pp. 25–36.

[11] Mahoney, M.—Chan, P.: Learning Models of Network Traffic for Detecting Novel Attacks. Technical report, Florida Institute of Technology 2002.

[12] Mukkamala, S.—Janowski, G.—Sung, A. H.: Intrusion Detection Using Neural Networks and Support Vector Machines. Heidelberg: Physica/Springer 2001, pp. 121–138.

[13] Mukkamala, S.—Sung, A. H.—Abraham, A.: Intrusion Detection Using an Ensemble of Intelligent Paradigms. Journal of Network and Computer Applications, Vol. 28. 2005, No. 2, pp. 167–182, Computational Intelligence on the Internet.

[14] Rumelhart, D. E.—Hinton, G. E.—Williams, R. J.: Learning Representations by Back-Propagating Errors. Nature, Vol. 323, 1986, No. 6088, pp. 533–536.

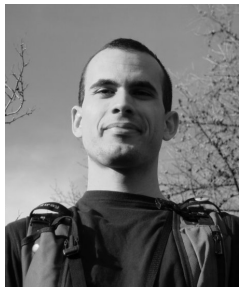
[15] Ryan, J.—Lin, M. J.—Miikkulainen, R.: Intrusion Detection with Neural Networks. Advances in neural information processing systems 1998, pp. 943–949.

[16] SecTools.Org. Top 5 Intrusion Detection Systems. `http://sectools.org/ids.html` [Online; accessed 20-April-2010].

[17] Yoo, I. S.—Ultes-Nitsche, U.: An Intelligent Firewall to Detect Novel Attacks. In Proceedings of Conference on Korean Science and Engineering Association in UK 2002.

[18] Zaki, M. J.: SPADE: An Efficient Algorithm for Mining Frequent Sequences. Machine Learning, Vol. 42, 2001, No. 1, pp. 31–60.

Kristián Valentín received his bachelor degree in applied informatics and his master's degree in cognitive science from the Faculty of Mathematics, Physics and Informatics of Comenius University in Bratislava. His main research interests include neural networks, image processing and pattern recognition using biologically motivated algorithms. He is currently a Ph. D. student at the same faculty and a researcher at the Institute of Measurement Science, Slovak Academy of Sciences, Bratislava under supervision of Professor Ivan Bajla.



Michal Malý received his master's and Ph. D. degree in computer science from the Faculty of Mathematics, Physics and Informatics of Comenius University in Bratislava in 2013. His main reasearch interests include reinforcement learning, natural language processing and neuroscience.