Feedback Control Based Test Case Instantiation For Firewall Testing

Tugkan Tuglular
Department of Computer Engineering,
Izmir Institute of Technology, Turkey
tugkantuglular@iyte.edu.tr

Gurcan Gercek
Department of Computer Engineering,
Izmir Institute of Technology, Turkey
gurcangercek@iyte.edu.tr

***Abstract*—A firewall's proper functioning is critical to the network it protects. Thus, a firewall should be tested with respect to its intended security policy. We propose a feedback control based approach for test case generation to detect mismatches between firewall's expected and executed behavior. In the proposed approach, abstract test cases are generated from firewall decision diagrams and instantiated with the test input values chosen using the proposed feedback control based selection algorithm. A case study is presented to validate the presented approach.**

Keywords-testing; firewalls; firewall testing; firewall decision diagram; feedback control.

I. INTRODUCTION

Firewalls have to be tested to validate that they work as specified. Literature on security [1], [2] mainly focuses on testing of firewall rules where firewall implementation is assumed error-free. However, a firewall can be hacked and programmed to behave differently from its specification or may have vulnerabilities as shown by Kamara et al. [3]. A firewall vulnerability is defined as an error made during firewall design, implementation, or configuration, that can be exploited to attack the trusted network that the firewall is supposed to protect [3]. One of the important goals of security management is identifying and eliminating vulnerabilities.

The main focus of our firewall testing approach is the intended security policy. The intended security policy consists of firewall rules configuring the firewall behavior. The security policy is external to the firewall like a configuration file [4].

This paper uses firewall decision diagram (FDD) to represent firewall policy and proposes to generate abstract test cases through path coverage using decision paths on the FDD. Instantiation of abstract test cases with test values selected based on the priority and weights using a feedback control approach is the novelty of the present paper. The test values with high priorities are assumed to have high probability to reveal mismatches between firewall's expected and executed behavior. Weights are used to alternate among high priority test values. Once the concrete test cases are ready, firewall testing process is executed using firewall evaluator architecture [5].

Next section summarizes related work before Section 3 outlines the background required for the proposed approach. The core of the paper, Section 4, explains our feedback control based test case instantiation approach for firewall testing. Section 5 presents the case study we carried out to exemplify the approach. Section 6 concludes the paper and outlines future work planned.

II. RELATED WORK

The firewall implementation testing approach evaluates the firewall policy correspondence with the actions the firewall performs, e.g., it checks whether a rule indicates to block a packet, where the firewall illegally forwards the packet, and this being the result of a firewall implementation error [2]. This paper focuses on firewall implementation testing by taking into account only policy execution.

There is one approach to firewall implementation testing by Senn et al. [6], who have worked on firewall implementation testing using protocol finite state automata (FSA) to generate abstract test cases through unique input/output (UIO) sequences [7] and instantiate abstract test cases with test tuples consisting of

<protocol>, <srcIP>, <dstIP>, <action>

fields of a firewall policy rule. However, in this work abstract test cases are generated from FDD and concrete test cases are built using

<protocol>, <srcIP>, <srcPort>, <dstIP>, <dstPort>, <action>

fields of a firewall rule.

An approach to specification-based test generation for security-critical systems is proposed by Wimmel and Jürjens [8]. Although not directly related, in their work, the test sequences are determined with respect to the security properties required by the system, using mutations of the system specification and attack

978-0-7695-4105-1/10 $26.00 © 2010 IEEE
DOI 10.1109/COMPSACW.2010.42

scenarios. They also followed the abstract test case generation approach and their concretization to apply to an existing implementation.

In this work, we use FDD [9] notion for modeling, whereas in our previous work [10], we used directed acyclic graph concept to deal with rule dependencies, which is implicitly handled by FDD. The present paper chooses FDD notation since formal, graph-theoretical notions and algorithms are utilized intensively with it.

III. BACKGROUND

While testing firewalls with respect to intended security policy, a model of the firewall policy helps to predict and control its behavior, for which FDDs are chosen.

A. Firewall Decision Diagrams

A field F_i is a variable whose value is taken from a predefined interval of nonnegative integers, called the domain of F_i and denoted by $D(F_i)$. A firewall decision diagram f (or FDD f, for short) over the fields $F_0, \cdots, F_{n-1}$ is an acyclic and directed graph that satisfies the following five conditions:

"1. f has exactly one node that has no incoming edges, called the root of f, and has two or more nodes that have no outgoing edges, called the terminal nodes of f.

2. Each nonterminal node v in f is labeled with a field, denoted by $F(v)$, taken from the set of fields $F_0, \cdots, F_{n-1}$. Each terminal node v in f is labeled with a decision that is either accept (or "a" for short) or discard (or "d" for short).

3. A directed path from the root to a terminal node in f is called a decision path. No two nodes on a decision path in f have the same label.

4. Each edge e, that is outgoing of a node v in f, is labeled with an integer set $I(e)$, where $I(e)$ is a subset of the domain of field $F(v)$.

5. Let v be any terminal node in f. The set $E(v)$ of all outgoing edges of node v satisfies the following two conditions:

(a) Consistency: For any distinct e_i and e_j in $E(v)$,

$$I(e_i) \cap I(e_j) = \emptyset$$

(b) Completeness: $\cup_{e \in E(v)} I(e) = D(F(v))$

where $\emptyset$ is the empty set and $D(F(v))$ is the domain of the field $F(v)$" [9].

An FDD f over the fields $F_0, \cdots, F_{n-1}$ can be represented by a sequence of rules, each of which is of the form

$$F0 \in S0 \wedge \cdots \wedge Fn-1 \in Sn-1 \rightarrow \langle decision \rangle$$

such that the following two conditions hold [9]. First, each rule in the sequence represents a distinct decision path in f. Second, each decision path in f is represented by a distinct rule in the sequence. Note that the order of the rules in the sequence is immaterial. The algorithm to generate a FDD is presented in [9].

B. Test Case Generation for Firewalls

Our test case generation consists of two parts. First, we generate abstract test cases. Abstract test cases are produced to test the correct policy handling of a firewall. Second, test input values are collected from various sources such as; firewall policy, domain topology knowledge, and black-lists. To obtain the concrete test cases, we instantiate abstract test cases with test input values.

The sequence of firewall rules is converted to a FDD as described in [9], which is then used for test generation. Each decision path in the FDD represents an abstract test case. We select to abstract a firewall rule as

IF (<protocol>, <srcIP>, <srcPort>, <dstIP>, <dstPort>)
THEN <action>,

where protocol is a network protocol, such as TCP or UDP, and action is either ACCEPT or DENY. The root node of a FDD represents the protocol field, and the terminal nodes represent the action field, intermediate nodes represent other fields respectively. Every decision path starting at the root and ending at a terminal node represents a rule in the policy.

Sets of test input values may be constructed using equivalence class partitioning, intelligent segmentation [11] or expert knowledge. The equivalence class partitioning divides the input domain of policy field into a finite number of partitions or equivalence classes [12]. El-Atawy et al. [11] proposed intelligent segmentation, where potential erroneous regions in the the firewall input space are adapted using the firewall policy. When determining test input data, values that a hacker might choose may be considered in addition to using the blacklists from network/security administrator or third parties as well as using statistical significant/insignificant past traffic. In this paper, we choose the expert knowledge approach to construct test input values. Although more time consuming and costly, test input values selected using expert knowledge is assumed to reveal more errors than other two approaches. Moreover, expert knowledge can prioritize test input values, which is the default feature of our approach. Finally, we instantiate the abstract test cases with the test input data to obtain concrete test cases.

C. Test Evaluation Process for Firewalls

Although firewalls are software implementations, the method of input and output used is network I/O. Thus, network packets should be produced, injected, and collected for testing a firewall. Test packets are

derived from generated test cases and those packets are sent or injected to the firewall to analyze its behavior. In order to analyze and evaluate the behavior of firewall under test (FUT) with respect to test cases, a special architecture as illustrated in Fig. 1 was developed in our previous work [5].

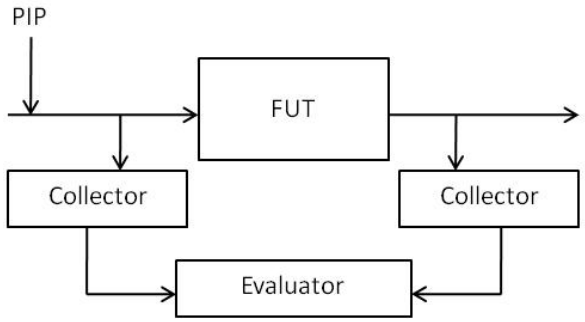

Figure 1. Firewall evaluator architecture [5].

The packet injection point (PIP) is used to release test packets. All the traffic entering and leaving the firewall is recorded and collected data is analyzed to obtain test outputs. The outputs are then compared with expected outputs to determine test result. The accepted packets are expected to be at the packet leaving point, but not the denied packets. The presented architecture can also be used to monitor a firewall constantly in order to check whether it operates in accordance with its specification and implementation [5].

IV. APPROACH

The proposed firewall testing process is presented in Fig. 2. The process starts with the generation of FDD from firewall policy and continues with the generation of abstract test cases as explained in Section III. In parallel, sets of test input values should be prepared to instantiate abstract test cases. We choose expert knowledge method to construct those sets. Although the number of sets may vary from expert to expert, we decide to utilize three sets for each of the following fields: *src_IP*, *dst_IP*, and *dst_port*. The selected sets are given in Table 1. Once the concrete test cases are generated, they will be converted to network packets and injected to firewall and the evaluation will be performed using firewall evaluator architecture as explained in Section III.

For the instantiation of abstract test cases, we propose feedback control based approach to select test input values. The field values that have higher potential to reveal errors should be selected more often than others. In order to facilitate this idea, priorities should be stored along with field values in the sets of test input values and used in the selection process. Moreover, the sets should have dynamic weights so that alternating among sets is possible. The proposed approach is illustrated in Fig. 3.

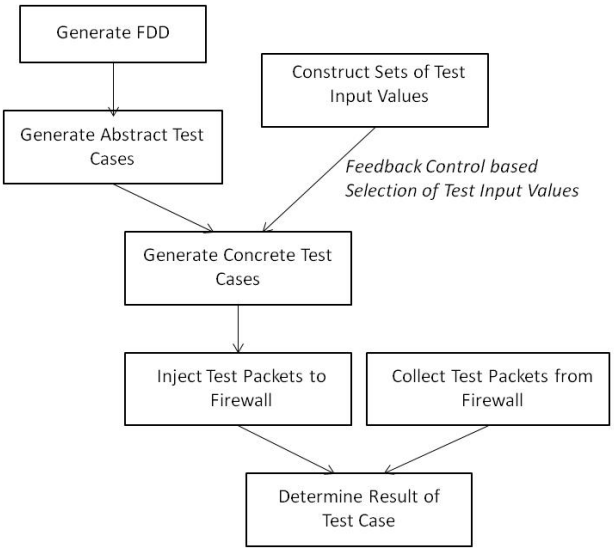

Figure 2. Summary of Proposed Firewall Testing Process.

The controller is responsible for the determination of next weight vector (wv), which is composed of *n* weight values, where *n* is the total number sets of test input values. A weight value is a real number and it is initially equal to 1. The controller, remembering the current weight vector and using the feedback, namely the identity (ID) of selected set, determines the next weight vector using Equations (1) and (2).

$$wv_i(k+1) = wv_i(k) - wap \text{ if } i \text{ is the ID of selected set} \quad (1)$$

$$wv_j(k+1) = wv_j(k) + (wap/n-1) \text{ for all } j \text{ not equal to } i \quad (2)$$

where $wv_i(k+1)$ is the the *i*th element of the weight vector at step k+1 and weight alternate percentage (wap) is a real number in (0,1).

When there is a test value request, the selector chooses a test input value from the sets using the (setID,elementID) information that comes from the intensity calculator. The intensity calculator stores a priority vector (pv), which is composed of priorities of all elements of all sets.

$$pv = (p_{11}, p_{12},\ldots,p_{1i},p_{21}, p_{22},\ldots,p_{2j},\ldots,p_{n1}, p_{n2},\ldots, p_{nk}) \quad (3)$$

TABLE I. Sets of Test Input Values

Field	Set1	Set2	Set3
Src_IP	blacklist_admin	blacklist_3rdp arty	past_traffic_ addresses
Dst_IP	current_domain _addresses	past_domain_ addresses	past_traffic_ addresses
Dst_port	listening_ports	vulnerable_ ports	past_traffic_ ports

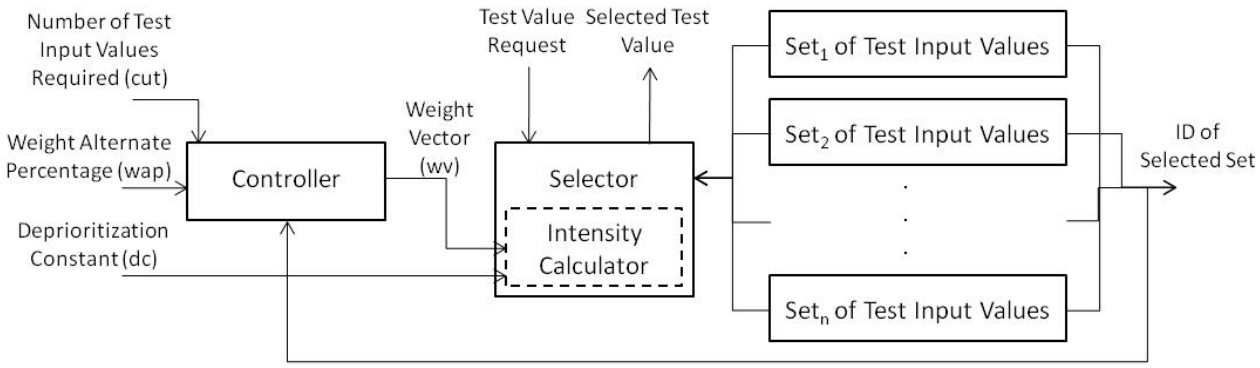

Figure 3. Feedback Control based Selection of Test Input Values

The size of the priority vector is the total number of elements of all sets. For instance, assuming that each set given in Table 1 has three elements, the size of the priority vector is nine. The intensity vector (iv) is obtained by normalizing the priorities, which is achieved by dividing each priority to the sum of all priorities (Σp).

$$iv = (i_{11}, i_{12},\ldots,i_{1i},i_{21}, i_{22},\ldots,i_{2j},\ldots,i_{n1}, i_{n2},\ldots, i_{nk}) \quad (4)$$

where $i_{jk} = p_{jk} / \Sigma p$.

The weighted intensity vector (wiv) is calculated by the scalar multiplication of the intensity vector with expanded weight vector (ewv), where expanded weight vector is composed of element weights that have their set weights.

$$wiv = iv \bullet ewv \quad (5)$$

The selected test value is the one where weighted intensity is the maximum of all weighted intensities. The intensity calculator passes the (setID, elementID) information of the maximum weighted intensity to the selector, which returns the corresponding test input value to the requestor.

For illustration purposes, an example is given in Table II. In this example, there are 3 sets of test input values, each having 2 elements and their priorities are presented in the priority vector. Using (4) and (5), weighted intensity of each element is calculated and presented in the weighted intensity vector. The element that has the highest weighted intensity is selected as the test value input for that step. At the next step, the weight of its set is reduced by the controller using *wap*. In addition to that, the priority of the selected element is decremented by the intensity calculator using deprioritization constant (*dc*), which is used to lower the priority of an element so that other elements will have a chance to be selected. The algorithm for feedback control based selection of test input values is given below in Algorithm 1.

The feedback control based selection should be performed separately for each field in the abstract test case. We suggest that test input values for the Protocol, Src_IP, Dest_IP, Dest_port fields should be selected for firewall testing. In the following section, we present a case study to illustrate the application of our proposed approach for feedback control based selection of test input values to firewall testing.

V. CASE STUDY

The policy of the firewall is considered as a specification. Therefore, the firewall testing context in this paper is specification based testing, where the operation of FUT, or implementation of its policy, is checked with respect to its specified policy, or expected behavior. Once the firewall policy is determined, it is loaded to the firewall and firewall is started. It should be noted that the loaded firewall policy on the FUT can be changed externally after starting the firewall. In that case, firewalls require re-start. When that happens, we assume that the specified policy does not match the implemented policy and if there is a mismatch it should be identified.

TABLE II. Test Input Value Selection Example using Feedback Control based Selection

		Weight Vector				**Priority Vector**							**Weighted Intensity Vector**								**Selected Set**
Step		w1	w2	w3		A1	A2	B1	B2	C1	C2		wiA1	wiA2	wiB1	wiB2	wiC1	wiC2		**MAX**	
1		1,00	1,00	1,00		5	5	8	4	5	2		0,14	0,14	**0,22**	0,08	0,14	0,05		**0,22**	2
2		1,05	***0,90***	1,05		5	5	***7***	4	5	2		0,15	0,15	0,18	0,08	0,15	0,06		0,18	2

Algorithm 1. Feedback Control based Selection of Test Input Values

```
INPUT : cut, wap, dc, ev, pv
OUTPUT : osse
cut: number of test input values required
wap : weight alternate percentage
dc : deprioritization constant
ev : element vector
pv : priority vector
osse: ordered set of selected elements
 1  proceed ← TRUE
 2  WHILE proceed DO
 3        Calculate wv using wap
 4        ewv ← extend(wv)
 5        Calculate iv using dc
 6        wiv ← iv • ewv
 7        selected_set ← selectS(ev, MAX(wiv))
 8        selected_element ← selectE(ev, MAX(wiv))
 9        osse ← osse ∪ selected_element
10        cut ← cut – 1
11        if (cut == 0 OR pv is all zero) then proceed ← FALSE
12  DO WHILE
13  return osse
```

Firewall testing is one of the approaches to identify such a mismatch. Even if there is no mismatch between specified and loaded policies, the firewall software and/or hardware may not behave as expected. The unexpected behavior can also be uncovered by using the firewall testing approach stated in this paper, which has another merit of the proposition.

The specified firewall policy in the case study is given in Table III. Then the policy is converted to a FDD, which is presented in Fig. 4. Using FDD, abstract test cases can be generated by traversing all paths so that path coverage criteria is achieved. For the right outmost path of FDD given in Fig. 4, the abstract test case is as follows:

Test Input	Protocol	tcp
Test Input	Src_IP	Src_1 ∪ Src_3
Test Input	Dest_IP	www
Test Input	Dest_port	]80[
Expected Output	Action	deny

To instantiate concrete test cases for this abstract test case, test input values should be selected for Src_IP and Dest_port fields. The sets given in Table I can be employed by our feedback control based selection approach. For the case study, the sets can be assumed to have elements given in Table IV with designated priorities. Similarly sets for Dst_port can be determined as given in Table V prior to test input value selection process.

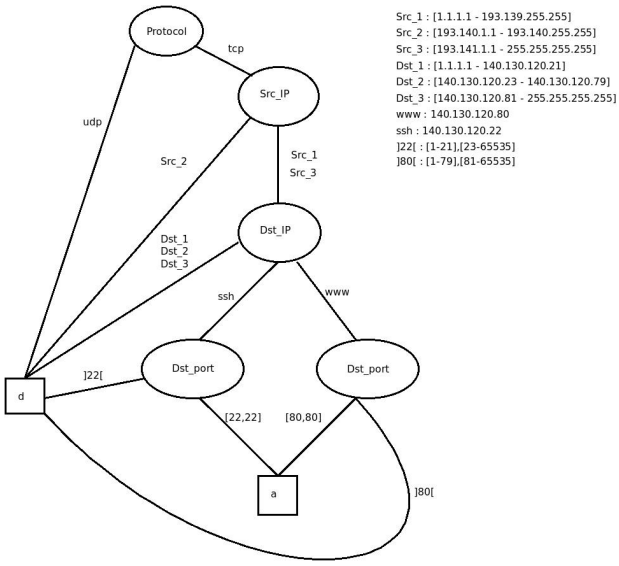

Figure 4. FDD of the Firewall Policy used for Case Study.

Some of the test input values for Src_IP and Dst_port are selected using the proposed feedback control based selection algorithm with the wap=0.1 and dc=1, then values are concatenated to form <Src_IP, Dst_port> test input value pairs and presented below:

<192.168.0.1, 143>; <127.0.0.1, 25>; <10.0.0.1, 143>; <88.241.34.41, 23>; <88.241.34.41, 25>; <192.168.0.1, 143>; <127.0.0.1, 23>; <88.241.34.41, 25>; <172.16.0.1, 143>; <10.0.0.1, 23>; …

Using each <Src_IP, Dst_port> test input value pair to instantiate the abstract test case given in this section, we obtain a partial set of concrete test cases generated for the case study, which are presented in Table VI. For definitive fields, such as Protocol and Dst_IP, the values in the abstract test case are utilized. For Src_port field, which does not occur in the abstract case because it is not in FDD, a random number can be employed to generate related test value or any random number can be used as the test value.

TABLE III. Firewall Policy used for Case Study

Rule	Protocol	Src_IP	Src_port	Dest_IP	Dest_port	Action
1	tcp	193.140.*.*	any	140.130.120.80	80	deny
2	tcp	*.*.*.*	any	140.130.120.80	80	accept
3	tcp	193.140.*.*	any	140.130.120.22	22	deny
4	tcp	*.*.*.*	any	140.130.120.22	22	accept
5	tcp	*.*.*.*	any	*.*.*.*	any	deny
6	udp	*.*.*.*	any	*.*.*.*	any	deny

TABLE IV. Src_IP Sets of Test Input Values for Case Study

Sets	IP_Addresses	Priority
blacklist_admin	192.168.0.1	8
blacklist_admin	172.16.0.1	6
blacklist_3rdparty	127.0.0.1	7
blacklist_3rdparty	10.0.0.1	7
past_traffic_addresses	88.241.34.41	6
past_traffic_addresses	215.15.568.23	5

TABLE V. Dst_port Sets of Test Input Values for Case Study

Sets	IP_Addresses	Priority
listening_ports	25	6
vulnerable_ports	143	7
past_traffic_ports	23	5

It should be noted that using same random value may cause some concrete test cases to be exactly same, e.g. see test cases numbered as 5 and 8.

TABLE VI. Partial Set of Concrete Test Cases for Case Study

No	Test Input	Expected Test Output
1	tcp, 192.168.0.1, 80, 140.130.120.80, 143	deny
2	tcp, 127.0.0.1, 80, 140.130.120.80, 25	deny
3	tcp, 10.0.0.1, 80, 140.130.120.80, 143	deny
4	tcp, 88.241.34.41, 80, 140.130.120.80, 23	deny
5	tcp, 88.241.34.41, 80, 140.130.120.80, 25	deny
6	tcp, 192.168.0.1, 80, 140.130.120.80, 143	deny
7	tcp, 127.0.0.1, 80, 140.130.120.80, 23	deny
8	tcp, 88.241.34.41, 80, 140.130.120.80, 25	deny
9	tcp, 172.16.0.1, 80, 140.130.120.80, 143	deny
10	tcp, 10.0.0.1, 80, 140.130.120.80, 23	deny

After the composition of concrete test cases, network packets should be generated from concrete test cases and be injected to the network using firewall evaluator architecture explained in Section III. The test packets given above will reveal the fact that the firewall policy is compromised if the traffic coming from 88.241.*.* to 140.130.120.80:23 is allowed.

VI. CONCLUSION

We proposed a feedback control based approach for test case generation to test firewalls. In the proposed approach, abstract test cases are generated from a FDD, which is driven from the firewall policy, and generated abstract test cases are instantiated with the test input values chosen using the proposed feedback control based selection algorithm.

The next step will extend the proposed approach to generate unique concrete test cases. The selection and prioritization of test input values in different sets will be automatized to obtain automatic firewall testing. Moreover, computational complexity and cost of the proposed approach will be investigated.

REFERENCES

[1] H. Heidi, "Specification based firewall testing," Master of Arts Thesis, Texas State University-San Marcos, May 2004.

[2] G. Zaugg, "Firewall testing," Diploma Thesis, ETH Zürich, 2005.

[3] S. Kamara, S. Fahmy, E. E. Schultz, F. Kerschbaum, and M. Frantzen, "Analysis of vulnerabilities in Internet firewalls," Computers & Security, 22:3, 2003.

[4] T. Tuglular and F. Belli, "Model based mutation testing of firewalls", Fast Abstracts of Testing: Academic & Industrial Conference, UK, 2008.

[5] T. Tuglular and F. Belli, "Protocol-Based Testing of Firewalls", Proc. of 4th South-East European Workshop on Formal Methods, Greece, December 5th, 2009.

[6] D. Senn, D. Basin, and G. Caronni, "Firewall conformance testing," TestCom 2005, LNCS 3502, 2005, pp. 226–241.

[7] K. Sabnani and A. Dahbura, "A protocol test generation procedure," Computer Networks and ISDN Systems, vol.15, 1988, pp. 285–297.

[8] G. Wimmel and J. Jurgens, "Specification-based test generation for security-critical systems using mutations," ICFEM 2002, LNCS 2495, 2002.

[9] M.G. Gouda and X.-Y.A. Liu. "Firewall design: consistency, completeness and compactness", Proc. ICDCS'04, 2004.

[10] T. Tuglular, O. Kaya, A. Muftuoglu and F. Belli, "Directed Acyclic Graph Modeling of Security Policies for Firewall Testing", MVV 09 Workshop, Proc. of SSIRI 2009, Shanghai, China, July 8-10, 2009.

[11] A. El-Atawy, K. Ibrahim, H. Hamed and E. Al-Shaer, "Policy segmentation for intelligent firewall testing", 1st ICNP Workshop on Secure Network Protocols, 2005.

[12] T. Tuglular, "Test case generation for firewall implementation testing using software testing techniques," Proc. 1st Int. Conf. on Security of Inform. & Networks, N. Cyprus, 2007, pp. 196-203.