2009 Fourth South-East European Workshop on Formal Methods

Protocol-Based Testing of Firewalls

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

Fevzi Belli
Department of Computer Science, Electrical Engineering and Mathematics,
University of Paderborn, Germany
belli@upb.de

***Abstract*—A firewall is the most important tool of network security defense. Its proper functioning is critical to the network it protects. Therefore a firewall should be tested rigorously with respect to its implemented network protocols and security policy specification. We propose a combined approach for test case generation to uncover errors both in firewall software and in its configuration. In the proposed approach, abstract test cases are generated by mutating event sequence graph model of chosen network protocol and filled with values from policy specification by using equivalence partitioning and boundary value analysis. A case study is presented to validate the presented approach.**

Keywords-testing; mutation testing; protocol robustness testing; conformance testing; firewalls

I. INTRODUCTION

As being most important security defense of a network, firewalls have to be tested to validate that they work as specified. Most of the work known from security literature [1,2] focuses on testing of firewall rules where firewall implementation is assumed error-free. However, a firewall may have vulnerabilities as shown by Kamara et al. [3] or can be hacked and programmed to behave differently from its specification. 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]. Identifying and eliminating vulnerabilities is one of most important goals of security management.

The firewall specification is mainly composed of intended *security policy* and allowed *network protocols*, which are the main focus of our firewall testing approach. The intended security policy consists of firewall rules which configure the firewall behavior and the allowed network protocols constitute the important part of firewall's internal working which can be described as packet capture, decision making on the packet under consideration, and packet release. Decision making operation is carried out with respect to firewall policy and network protocols. The security policy is external to the firewall like a configuration file, whereas packet checking with respect to network protocols is implemented in the firewall software.

The way to process valid inputs as well as a few invalid ones is described by the protocol specification. As conformance testing only verifies an implementation with respect to its protocol specification, its capability of error-detection is limited. Furthermore, disturbance, misconfiguration and man-made attacks exist in the Internet. The critical requirement on reliability, fault tolerance and security of network devices highlights the necessity of protocol robustness testing [16].

This paper extends our idea of firewall testing [14], where we introduced the idea of applying mutation analysis technique to firewall testing. The novelty of the present paper stems from following:

(i) The network protocol, the implementation of which is under test, is modeled using event sequence graphs (ESGs).
(ii) The mutants of the network protocol implementation are generated using mutation operators specifically defined for ESGs.
(iii) The test case generation is achieved in two phases. In the first phase, abstract test cases are generated by producing mutants from the model of the protocol under consideration. In the second phase, generated abstract test cases are converted to concrete test cases using firewall policy to determine test input values. Concrete test input values are obtained using equivalence class partitioning and boundary value approach.

The present approach can also be applied to specific high level firewalls [4], which check network packets and streams with respect to application protocols, such as HTTP and SOAP, to assure acceptable use of web services in accordance with the application level security policy.

Next section summarizes related work before Section 3 outlines the theoretical background of the approach. The core of the paper, Section 4, explains our model-based mutation testing approach. Section 5 presents the case study we carried out to exemplify the approach and indicate its characteristic features. Section 6 concludes the paper and outlines future work planned.

II. RELATED WORK

There are three general approaches to firewall testing [2]: penetration testing, testing of the firewall rules, and testing of the firewall implementation.

Penetration testing is performed to check the firewall for potential breaches of security that can be exploited. The firewall penetration testing is structured in the following four steps [15]:

- indirect information collection,
- direct information collection,
- attack from the outside, and
- attack from the inside.

This type of testing targets specific known vulnerabilities of firewalls determined through information collection.

Testing of the firewall rules verifies whether the security policy is correctly enforced by a sequence of firewall rules or not. According to Al-Shaer and Hamed [11], in a single firewall environment, the local firewall policy may include firewall anomalies, where the same packet may match more than one filtering rule. Testing of the firewall rules is used to discover firewall policy anomalies.

The *firewall implementation testing* approach evaluates the correspondence of firewall rules with respect to the actions the firewall performs, e.g., it checks whether a rule indicates to block a packet, but the firewall illegally forwards the packet, which might be caused by a firewall implementation error [2].

This paper focuses on firewall implementation testing by considering both the network protocol implementation and policy execution. As far as authors of this paper could check, there is one similar approach by Senn et al. [5], who worked on firewall implementation testing using protocol finite state automata (FSA) to generate abstract test cases through unique input/output (UIO) sequences [17] and instantiate abstract test cases with test tuples consisting of

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

fields of a firewall policy rule. However, our work generates abstract test cases from protocol ESG using mutation operators defined for ESGs and concrete test cases are built using

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

fields of a firewall rule.

Although not directly related to firewall testing, Wimmel and Jürjens [6] applied the concept of mutation analysis to specification-based test generation for security-critical systems. In their work, the test sequences are determined with respect to the system's required security properties, using mutations of the system specification and attack scenarios. They also followed the approach of abstract test case generation and their concretization to apply to an existing implementation.

Protocol robustness testing attempts to verify whether or not implementation under test can function correctly in the presence of invalid inputs or stressful environmental conditions. It aims to detect vulnerabilities of protocol specification and protocol implementations. Invalid inputs include messages with invalid syntax (i.e., messages which disobey protocol specification data formats) and messages with anomalous semantics (i.e., messages which have valid syntax but conflict with protocol state, configuration, parameters and policies). Robustness testing by injecting messages with invalid syntax is called *mutation testing of protocol messages* [16]. Jing et al. [16] claimed that TTCN-3, a standard test specification language, reveals strong excellence in conformance testing, and they applied TTCN-3 to mutation testing and extended it according to test requirements. They tested OSPFv2 sufficiently with a test system based on extended TTCN-3.

In our work, we concentrate on protocols used by stateful firewalls. For modeling we use ESG notion which is similar to the concept of event flow graphs [22, 20]. The latter are used for analysis and validation of user interface requirements prior to implementation and testing of the code [23]. The present paper chooses ESG notation because it intensively uses formal, graph-theoretical notions and algorithms which are developed independently from and prior to event flow graphs [21].

III. THEORETICAL BACKGROUND

While testing a system, a model of the system helps to predict and control its behavior. Modeling a system acquires the understanding of its abstraction and there is the need of a formal specification tool distinguishing between legal and illegal situations. These requirements are fulfilled by ESGs.

A. Event Sequence Graphs

Apart from the notion of FSA, in ESG, the simplification by merging the inputs and states helps the test engineer to easily understand and check the external behavior of the system, hence the "inputs" and "states" are turned into "events".

Basically, an *event* is an externally observable phenomenon, such as an environmental or a user stimulus, or a system response, punctuating different stages of the system activity. Following, we formally define ESG; a simple example of an ESG is given in Fig. 1.

Definition 1. An *event sequence graph* $ESG = (V, E, \Xi, \Gamma)$ is a directed graph where $V \neq \varnothing$ is a finite set of vertices (nodes), $E \subseteq V \times V$ is a finite set of arcs (edges), $\Xi, \Gamma \subseteq V$ are finite sets of distinguished vertices with $\xi \in \Xi$, and $\gamma \in \Gamma$, called entry nodes and exit nodes, respectively, wherein $\forall v \in V$ there is at least one sequence of vertices $\langle \xi, v_0, \dots, v_k \rangle$ from each $\xi \in \Xi$ to $v_k = v$ and one sequence of vertices $\langle v_0, \dots, v_k, \gamma \rangle$ from $v_0 = v$ to each $\gamma \in \Gamma$ with $(v_i, v_{i+1}) \in E$, for i = 0,…,k-1 and $v \neq \xi, \gamma$.

Ξ (*ESG*) and Γ (*ESG*) represent the entry nodes and exit nodes of a given ESG, respectively. To mark the entry and exit of an ESG, all $\xi \in \Xi$ are preceded by a pseudo vertex '[' $\notin$ V and all $\gamma \in \Gamma$ are followed by another pseudo vertex ']' $\notin V$. The semantics of an ESG is as follows: Any $v \in V$ represents an event. For two events $v, v' \in V$, the event v' must be enabled after the execution of v iff $(v, v') \in E$. The operations on identifiable components of the GUI are controlled and/or perceived by input/output devices, i.e., elements of windows, buttons, lists, etc. Thus, an event can be a user input or a system response; both of them are elements of V and lead interactively to a succession of user inputs and expected desirable system outputs.

Definition 2. Let V, E be defined as in Definition 1. Then any sequence of vertices $\langle v_0, \dots, v_k \rangle$ is called an *event sequence* (*ES*) iff $(v_i, v_{i+1}) \in E$, for i=0,…,k-1. Moreover, an ES is *complete* (or, it is called a *complete event sequence, CES*), iff $v_0 \in \Xi$ and $v_k \in \Gamma$.

Note that the pseudo vertices '[', ']' are not included in *ESs*. An *ES* = $\langle v_i, v_k \rangle$ of length 2 is called an *event pair (EP)*. A *CES* may invoke no interim system responses during user-system interaction, i.e., it may consist of consecutive user inputs and a final system response.

Our approach assumes that upon a faulty user input the system has to inform the user, and, wherever possible, point him or her properly in the right direction in order to reach the desirable final or interim situation. Due to this requirement, a complementary view is necessary to consider potential user errors in the modeling of the system.

Definition 3. For an *ESG = (V, E)*, its *completion* is defined as $\widehat{ESG} = (V, \widehat{E})$ with $\widehat{E} = V \times V$.

Definition 4. The *inverse* (or *complementary*) ESG is then defined as $\overline{ESG} = (V, \overline{E})$ with $\overline{E} = \widehat{E} \setminus E$.

Fig. 1 illustrates $\widehat{ESG}$, which can systematically be constructed in three steps:

- Add arcs in the opposite direction wherever only one-way arcs exist.
- Add self-loops to vertices wherever none exist.
- Add two-way arcs between vertices wherever no arcs connect them. Note that they are drawn bi-directional.

$\overline{ESG}$ (the inversion of the ESG) consists of arcs that will be added to the ESG to construct the $\widehat{ESG}$ (completion of the ESG). Graphically speaking, missing edges of the ESG represent undesirable user-system interactions, i.e., *faulty event pairs (FEP)*. FEPs can systematically be constructed by using either step of $\widehat{ESG}$ construction steps.

Definition 5. Let *ES* = $\langle v_0, \dots, v_k \rangle$ be an event sequence of length k+1 of an ESG and *FEP* = $\langle v_k, v_m \rangle$ a faulty event pair. The concatenation of the *ES* and *FEP* then forms a *faulty event sequence FES* = $\langle v_0, \dots, v_k, v_m \rangle$. *FES* is *complete* (or, it is called a *faulty complete event sequence, FCES*) iff $v_0 \in \Xi$. The ES as part of a *FCES* is called a *starter.*

CES and *FCES* are used to form test cases for the system under test (SUT). The SUT is supposed to accept test inputs described by CESs in the specified order whereas test inputs described by *FCESs* should result in a warning.

Completeness Ratio (CR) is a metric which explains density of edges in the ESG and is defined as follows:

$$\boldsymbol{CR} = |\boldsymbol{E}| \,/\, |\boldsymbol{V}|^2$$

where $|E|$ is the number of edges in the ESG and $|V| = n$ is the number of nodes (vertex) in the ESG. CR takes the values between 0 and 1. Value 1 shows that ESG is completed graph and Value 0 means null graph. As the values are getting closer to 1, the density of the graph gets bigger.

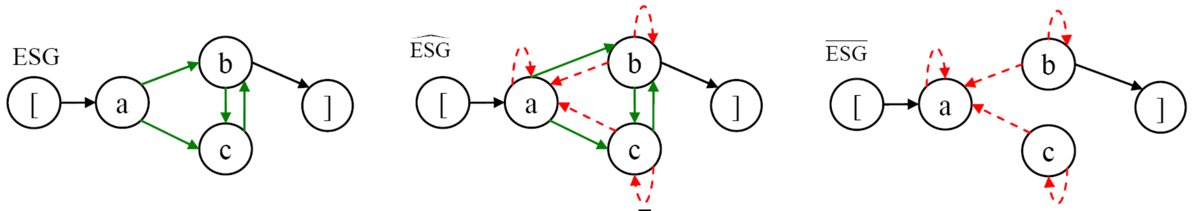

Figure 1. An ESG, its completion $\widehat{ESG}$ and its inversion $\overline{ESG}$.

B. *Mutant Generation*

Assuming that the given ESG correctly specifies the expected, desirable behavior of SUT, the complemented ESG can be used to generate mutants of the system, i.e., to specify erroneous, undesirable situations. In other words, to describe, how the system is not supposed to behave. The given ESG can be changed by manipulating either the arcs or the events [7]. Arcs are primarily responsible for correctly sequencing the events, in our case network events.

Basically, we can generate *arc mutants* of an ESG by

- *inserting* an extra arc in any direction, without causing a multiple arc in the same direction (*arc insertion*, *aI-operation*), or
- *omitting* an existing arc (*arc omission*, *aO-operation*).

It is important to note that

- applying the *aI*-operation to all *EPs* of an ESG produces its inversion ESG and leads to the completion of the ESG given. Based on the complete ESG and using the algorithms given in [8], *FCESs* can systematically be generated to obtain mutants,
- applying the *aO*-operation to all *EPs* of an ESG generates *ES* of various lengths that are mutants to simulate incomplete paths, i.e., deadlocks.

aI- and *aO*-operations can be applied to an ESG repeatedly, e.g., *n* times. This is represented as aI^n and aO^n. They can also be combined arbitrarily, e.g., three arcs inserted or two arcs deleted; represented by $aI^3 + aO^2$. "+" represents the choice as *inclusive or*.

C. *Test Process*

The approach introduced in this paper uses event sequence mutants. Each mutant is covered by either a *CES* or *FCES*. More precisely, test inputs are *CES* and *FCES* of original ESG. If the input is a *CES*, the SUT is supposed to successfully proceed it and thus, to succeed the test and to trigger a desirable event. Accordingly, if a *FCES* is used as a test input, a failure is expected to occur which is an undesirable event and thus, to fail the test. Algorithm 1 below sketches the optimized test process.

The approach ensures the coverage of *ESs* of length n, whereby n=1 is the node coverage, and n>1 is the coverage of *ES* of the length>2. Additionally, the coverage of the *FEP* is also ensured.

The union of the sets of *CESs* of minimal total length to cover the *ESs* of a required length is called *Minimal Spanning Set of Complete Event Sequences* (*MSCES*). If a *CES* contains all *EPs* at least once, it is called an entire walk. A legal entire walk is minimal if its length cannot be reduced. A minimal legal walk is ideal if it contains all *EPs* exactly once. Legal walks can easily be generated for a given ESG as *CESs*, respectively. It is not, however, always feasible to construct an entire walk or an ideal walk. The algorithm to determine *MSCES* is given and explained in [18]. Another approach to minimize the number of event sequences was proposed by Memon et al. [20]. Since it is impractical to test a SUT for all possible event sequences for large number of event sequences, they suggested to identify important sequences through assigning a priority to each event sequence and generate test cases from these important sequences.

Algorithm 1. Test Process [18].

```
length: length of the test sequences
Generate appropriate $ESG$ and $\overline{ESG}$
FOR k:=2 TO length DO
    Cover all ESs of length k by means of CESs subject
    to minimizing the number and total length of the
    CESs
Cover all FEPs of by means of FCESs subject to
minimizing the total length of the FCESs
Apply the test set to the SUT
Observe the system output to determine whether the
system response is in compliance with the expectation.
```

IV. APPROACH

We constructed a firewall dataflow model by taking layers, which deal with network protocol processing with respect to protocol specification and access control evaluation in accordance with firewall policy, from the dataflow model of firewall internals described by Frantzen et al. [9]. Our firewall dataflow model is presented in Fig. 2.

After a network packet is received, it may pass various layers for some filtering and processing until dynamic ruleset layer which is the core of stateful firewalls. The dynamic ruleset, or state table, serves to associate each packet with its connection stream [10]. Part of the protocol, i.e., its FSA, implementation exists in this layer. Therefore, it is one of our testing targets. If a packet belongs to a connection that is in the state table, it bypasses following checks and filters for performance reasons. Our second testing target is the execution layer of firewall policy, namely IP and port filtering layer. Depending on the firewall rules, of which modeling is explained in Section 4.2, packets may be dropped in this layer. Furthermore, packets that are part of an application stream are checked for conformance with respect to the chosen application level protocols such as HTTP and SOAP in application level layer. Our proposed testing approach can be applied to protocols at any level in the network protocol stack.

Our test case generation consists of two parts. First, we generate abstract test cases. Abstract test cases are produced to test the correct stateful handling of a protocol by a firewall. For example, a stateful packet filter may be tested to determine whether it correctly handles TCP traffic. Second, we generate test input data from the firewall policy. To obtain the concrete test cases, we instantiate abstract test cases with test input data.

A. Abstract Test Cases

Assuming that the given ESG correctly specifies the expected, desirable behavior of the protocol, manipulation of event transitions can be used to generate mutants of the protocol, i.e., to specify erroneous, *undesirable* situations [7]. Using this model-based approach, a mutant can be generated and it will behave in a way that the protocol is not supposed to behave. In this proposed approach, we generate mutants of an ESG only by inserting an extra arc in any direction and omitting an existing arc.

B. Concrete Test Cases

In the second phase of the approach, sequence of firewall rules is converted to a firewall policy tree as described in [11], which is then used as a test tree. Each node in the policy tree represents a field of a rule. A firewall rule is abstracted as

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

where protocol is a network protocol, such as TCP or UDP, and action is either ALLOW or DENY. The root node of a policy tree represents the protocol field, and the leaf nodes represent the action field, intermediate nodes represent other fields in order. Every tree path starting at the root and ending at a leaf represents a rule in the policy and vice versa. The equivalence class partitioning divides the input domain of the SUT into a finite number of partitions or equivalence classes. The equivalence classes are identified by taking each input condition – in our approach each field of the rule or each node of the policy tree – and partitioning it into two classes [12]. When generating test input data, values that a hacker might choose are considered in addition to the boundary values in equivalence classes. Finally, we instantiate the abstract test cases with the test input data to obtain concrete test cases.

C. Firewall Evaluator Architecture

Although firewalls are implemented as software, their method of input and output is network I/O. Therefore, network packets should be produced, injected, and collected in order to test a firewall. Test packets will be derived from generated test cases and those packets will be sent or injected to the firewall to analyze its behavior. To be able to analyze and evaluate behavior firewall under test (FUT) with respect to test cases, we developed a special architecture as illustrated in Fig. 3.

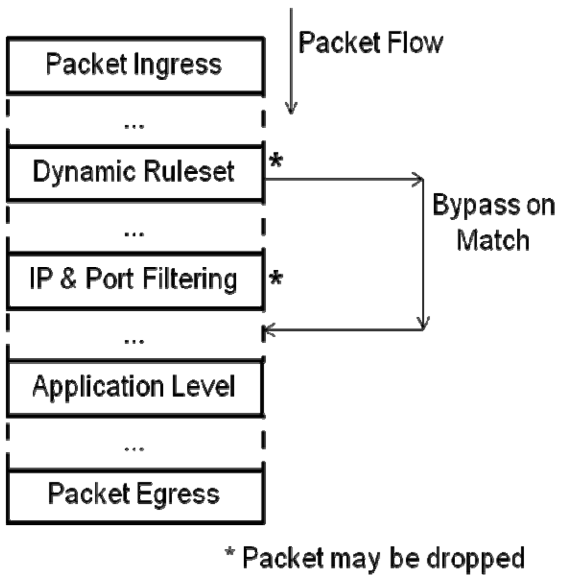

Figure 2. Firewall dataflow model.

The test packets are released from packet injection point (PIP). All the traffic entering and leaving the firewall is recorded and collected data is analyzed to obtain test outputs, which are compared with expected outputs to determine test result. We expect to see allowed packets at the packet leaving point but not the denied packets. This architecture can also be used to monitor a firewall constantly to check whether it operates in accordance with its specification and implementation.

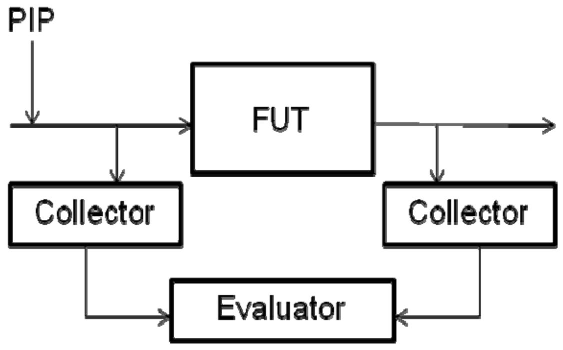

Figure 3. Firewall evaluator architecture.

V. CASE STUDY

If the FUT has a stateful packet filter, as almost all firewalls do, this means that related automaton for each stateful protocol is implemented in the firewall software. If TCP protocol is supported by the firewall, it should contain necessary code that executes the automaton for the TCP protocol, which is presented in Table 1. The event labels and their event descriptions are as follows [13]:

- S – Request to open connection,
- SA – Agree to open connection,
- A – Acknowledgement of receipt,
- F1 – Request to close connection,
- F2 – Request to close connection, and
- R – Tear down connection.

TABLE I. TRANSITION TABLE for TCP FSM [13].

Event State	S	SA	A	F1	F2	R	Other
L	H_1	Ø	Ø	Ø	-	Ø	Ø
H_1	H_1	H_2	H_1	Ø	-	Ø	Ø
H_2	H_2	H_2	X	Ø	-	Ø	Ø
X	Ø	Ø	X	C_1	-	C_2	Ø
C_1	Ø	Ø	C_1	C_1	C_2	C_2	Ø
C_2	Ø	Ø	C_2	C_2	C_2	C_2	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

The corresponding ESG of the TCP automaton is given in Fig. 4. We can now create a mutant for the ESG of TCP protocol by inserting or omitting arc(s) using the algorithm presented in [8].

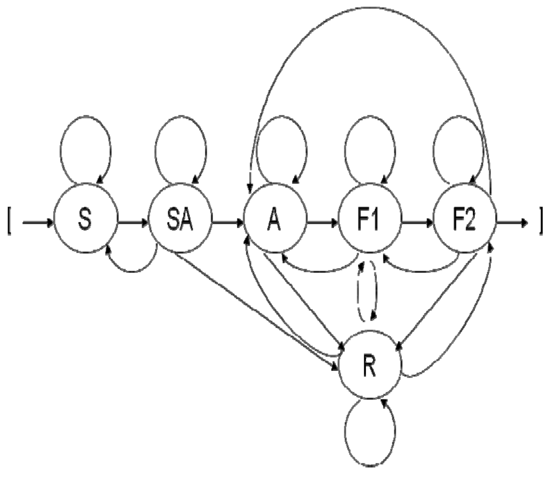

Figure 4. ESG for TCP Protocol.

Since TCP is a reliable protocol, it has a high level redundancy. Therefore, a mutant created by omitting only a single arc will not be able to behave altogether differently. Even though the protocol is reliable, it does not imply that its implementation is correct and therefore the implementation should be tested comprehensively by using mutants created for abstract test cases.

As an example, the event sequence S-SA-F1 is a mutant *ES* for TCP protocol, which means request to close connection that is not established. At the same time, the sequence S-SA-F1 is a *FCES* for TCP protocol, which constitutes an abstract test case in our approach. For this test case, three test packets should be concretized with source IP address, source port, destination IP address, and destination port. The IP address and port values are generated from firewall policy rules.

Some of the test input data generated using equivalence partitioning and boundary value analysis for the firewall rule

<tcp, 193.140.248., any, *.*.*.*, 25, accept>*

are presented in Table 2. Test cases numbered as 2, 3, and 4 have the given expected output unless there are other rules concerning the values in the test input data. The values shown in bold for the second, third, and fourth test cases are the reason for denying packets with respect to above firewall rule. The mutant *ES* can be instantiated with values presented in Table 2 and generated network packets can be injected using firewall evaluator architecture.

TABLE II. A SET of GENERATED TEST CASES.

No	Test Input	Expected Test Output
1	tcp,193.140.248.1,80,193.140.250.1,25	allow
2	tcp,**193.140.247.1**,80,193.140.250.1,25	deny
3	tcp,**193.140.249.1**,80,193.140.250.1,25	deny
4	tcp,193.140.248.1,80,193.140.250.1,**24**	deny

VI. CONCLUSION

We proposed a combined approach for test case generation to uncover errors both in firewall software and in its configuration. In the proposed approach, abstract test cases are generated by mutating event sequence graph model of implemented network protocol and filled with values from policy specification by using equivalence partitioning and boundary value analysis. At the moment we apply this approach to some well-known open source firewalls.

The next step will extend the proposed approach to cover application firewalls, where we plan to develop specific test suites for application level protocols such as HTTP and SOAP to cover web services as well.

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, pp.214-232.

[4] E. B. Fernandez, “Two patterns for web services security”, Proceedings of the International Symposium on Web Services and Applications, Las Vegas, 2004.

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

[6] G. Wimmel and J. Jurgens, “Specification-based test generation for security-critical systems using mutations,” ICFEM 2002, LNCS 2495, 2002, pp. 471-482.

[7] F. Belli, C. J. Budnik, and W. E. Wong, “Basic operations for generating behavioral mutants,” Proc. 2nd ISSRE Workshop on Mutation Analysis, IEEE, 2006.

[8] F. Belli and C. J. Budnik, “Minimal spanning set for coverage testing of interactive systems,” Proc. of Int. Colloquium on Theoretical Aspects and Computing, LNCS 3407, 2004, pp. 220-234.

[9] M. Frantzen, F. Kerschbaum, E. E. Schultz, and S. Fahmy, “A framework for understanding vulnerabilities in firewalls using a dataflow model of firewall internals,” Computers & Security Vol.20, No.3, 2001, pp.263-270.

[10] D. Newman, “Benchmarking terminology for firewall performance,” RFC 2647, August 1999.

[11] E. Al-Shaer and H. Hamed, “Discovery of policy anomalies in distributed firewalls”, TINFOCOM, IEEE, 2004, vol.4, pp. 2605-2616.

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

[13] J. Treurniet and J. H. Lefebvre, “A finite state machine model of TCP connections in the transport layer,” (DRDC Ottawa TM 2003-139), Defence R&D Canada – Ottawa, 2003.

[14] T. Tuglular and F. Belli, “Model based mutation testing of firewalls,” Fast Abstracts of Testing: Academic & Industrial Conference, Windsor, UK, August 29-31, 2008.

[15] R. E. Haeni, “Firewall penetration testing,” George Washington University Report, January, 1997.

[16] C. Jing, Z. Wang, X. Shi, X. Yin, and J. Wu, "Mutation testing of protocol messages based on extended TTCN-3," Proceedings of 22nd International Conference on Advanced Information Networking and Applications, 2008.

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

[18] F. Belli and C. J. Budnik, “Test minimization for human-computer interaction,” J. of Applied Intelligence 7(2), Springer, 2007.

[19] F. Belli, M. Eminov, and N. Gökçe, “Model-based test prioritizing – a comparative soft-computing approach and case studies,” KI 2009, LNAI 5803, 2009, pp. 425–432.

[20] A. M. Memon, M. L. Soffa, and M. E. Pollack, “Coverage criteria for gui testing”, SIGSOFT Softw. Eng. Notes, 26(5), 2001, pp. 256–267.

[21] F. Belli, “Finite state testing and analysis of graphical user interfaces,” Proc. ISSRE’01, IEEE Computer Society, 2001, pp. 34–43.

[22] A. M. Memon, “An event-flow model of GUI-based applications for testing,” Softw. Test. Verif. Reliab., 17(3), 2007, pp. 137–157.

[23] A. M. Memon, M. E. Pollack, and M. L. Soffa, “Hierarchical GUI test case generation using automated planning,” IEEE Trans. Softw. Eng. 27(2), 2001, pp. 144-155.