HOL-TestGen/fw
An Environment for Specification-Based Firewall Conformance Testing

Achim D. Brucker[1], Lukas Brügger[2], and Burkhart Wolff[3]

[1] SAP AG, Vincenz-Priessnitz-Str. 1, 76131 Karlsruhe, Germany
achim.brucker@sap.com
http://www.brucker.ch/
[2] Information Security, ETH Zurich, 8092 Zurich, Switzerland
lukas.a.bruegger@gmail.com
[3] LRI, Université Paris-Sud, 91405 Orsay Cedex, France
wolff@lri.fr
http://www.lri.fr/~wolff

Abstract. The HOL-TestGen environment is conceived as a system for modeling and *semi-automated* test generation with an emphasis on expressive power and generality. However, its underlying technical framework Isabelle/HOL supports the customization as well as the development of highly automated add-ons working in specific application domains.
In this paper, we present HOL-TestGen/fw, an add-on for the test framework HOL-TestGen, that allows for testing the conformance of firewall implementations to high-level security policies. Based on generic theories specifying a security-policy language, we developed specific theories for network data and firewall policies. On top of these firewall specific theories, we provide mechanisms for policy transformations based on derived rules and adapted code-generators producing test drivers. Our empirical evaluations shows that HOL-TestGen/fw is a competitive environment for testing firewalls or high-level policies of local networks.
Keywords: symbolic test case generations, black box testing, theorem proving, network security, firewall testing, conformance testing.

1 Introduction

HOL-TestGen [6, 7] (http://www.brucker.ch/projects/hol-testgen/) is a generic model-based testing environment. Built as an extension of the Isabelle framework [15], HOL-TestGen inherits, among other things, the front-end PIDE, the Isar language for HOL specifications and proofs, and code- and documentation generators from the Isabelle framework. HOL-TestGen extends the framework by an infrastructure to develop formal *test plans*, i. e., descriptions of test goals, their decomposition into abstract test partitions, and their transformation to concrete tests with the help of constraint solvers like Z3 [12]. Finally, customized code-generators produce code of concrete test drivers which can be run against real implementations following a black-box testing approach.

Z. Liu, J. Woodcock, and H. Zhu (Eds.): ICTAC 2013, LNCS, pp. 112–121, 2013.

HOL-TestGen as such is conceived as an interactive, flexible environment that draws from the abundant expressive power and generality of HOL; test plans are therefore typically mixtures of very powerful automated partitioning and selection tactics, their configurations, and intermediate small-step tactics that help to turn the results into a suitable form for the next step. HOL-TestGen was used successfully in large case studies from various domains, see [7] for details.

In this paper, we present the novel HOL-TestGen/FW environment, which is an add-on of HOL-TestGen for a specific problem domain: the specification-based conformance test of network components. Such components can be stateless packet filters, stateful firewalls, routers, devices performing network address translation (NAT), etc. In the sequel we just refer to them as firewalls. We describe the underlying generic theories for modeling network data and firewall policies using a generic security-policy language called the *Unified Policy Framework* (UPF) [3, 8], mechanisms for policy transformations (for which formal proofs of correctness have been established [2]) and adapted code-generators producing test drivers. We present application scenarios as well as experimental evaluations which show HOL-TestGen/FW as a competitive environment for testing firewalls or high-level policies of networks.[1]

2 The HOL-TestGen/FW Workflow

HOL-TestGen/FW is an environment for the specification-based conformance testing of firewalls.

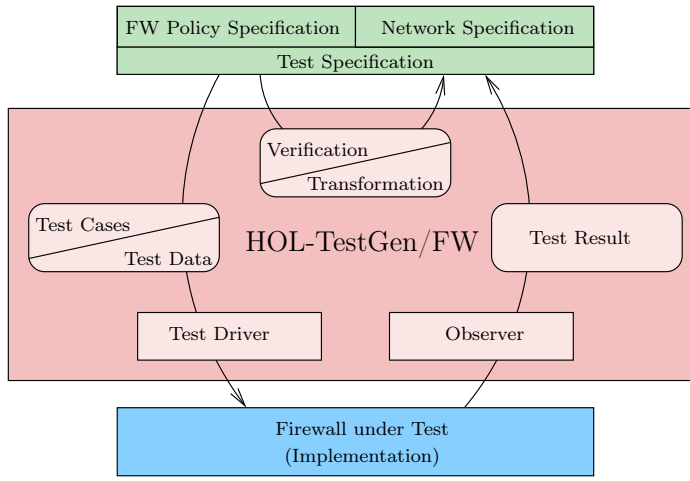


Fig. 1: The HOL-TestGen/FW Architecture

[1] HOL-TestGen/FW (including case studies) is, since version 1.7.1, part of the HOL-TestGen distribution. HOL-TestGen is available from: `http://www.brucker.ch/projects/hol-testgen/`.

Figure 1 illustrates the standard workflow, respectively the main components of HOL-TestGen/fw:

1. *Firewall policy specification, network specification, and test specification:* HOL-TestGen/fw provides an instantiation of the Unified Policy Framework (UPF) [3, 8] that allows to specify networks and security policies for those networks in a user-friendly way. The test specification, i. e., the properties that should be tested, also need to be specified here.
2. *Test case and test data generation:* In this phase, the abstract *test cases* as well as the concrete *test data* are generated. The test cases still contain variables and constraints on them: a test case actually represents a section of the test space. By choosing ground instances for these variables solving the constraints, we obtain test data to be run against an actual implementation.
3. *Test execution and test result validation:* Finally, the test data is injected (using the *test driver*) into a real network and the behavior of the firewall under test is observed (using an *observer* or monitor) and compared to the test specification.

In its essence, this resembles the standard model-based testing workflow applied to firewalls (or any other network components controlling network traffic). In addition, HOL-TestGen/fw also supports:

- *Verification of (security) properties:* Both the specification of the security policy as well as the network specification can be analyzed and properties can be verified formally using the full reasoning power of Isabelle/HOL.
- *Verified transformations for testability:* As we will see later, different syntactical representations can, while being semantically equivalent, result in test efforts that differ by several orders of magnitude. Thus, using HOL-TestGen/fw, the testability can be improved by applying policy transformations. The correctness of these transformations, in the sense that applying the transformation does not change the semantics of a policy, is formally proven using Isabelle/HOL.

With the exception of the test execution and test result validation, the standard interface of Isabelle, called PIDE [13], is used by HOL-TestGen/fw. Figure 2 illustrates a typical use of HOL-TestGen/fw: In the upper left, we see the specification of the firewall under test and in the lower left we see the generated abstract test cases. The test cases still contain variables that need to be instantiated before they can be executed on a real firewall implementation.

In the rest of this section, we discuss the steps of the HOL-TestGen/fw workflow in more detail.

2.1 System and Test Specification

The Language: UPF with Firewall-Policy-Combinators. HOL is a typed λ-calculus and its foundational type are total functions $\alpha \Rightarrow \alpha'$. Using the provided infrastructure, the usual data-types like α option or α list can be defined. *Partial* functions ($\alpha \rightharpoonup \alpha'$) are introduced as synonym to $\alpha \Rightarrow (\alpha'$ option). They

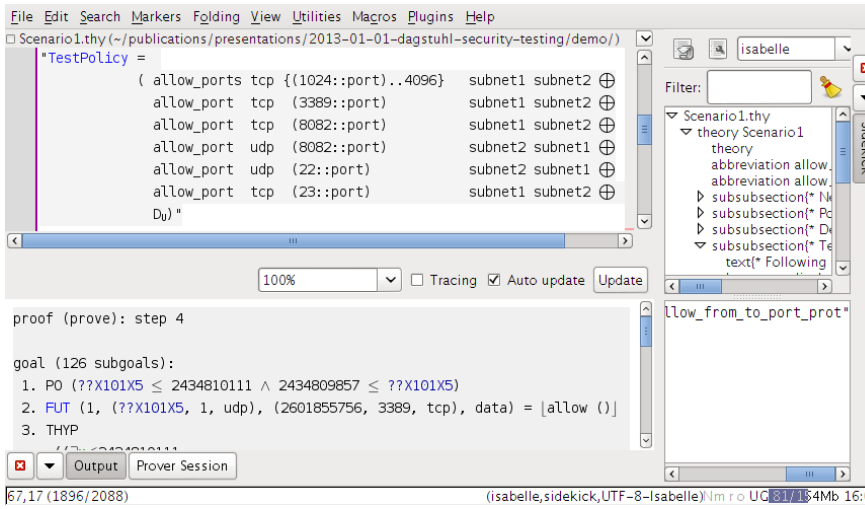


Fig. 2: A HOL-TestGen/fw Session using the PIDE/jEdit User Interface

are used to model the fundamental concept of UPF: *policies* as partial decision functions:

$$(\alpha, \beta)\ \text{policy} = \alpha \rightharpoonup (\beta\ \text{decision})$$

The decision datatype is defined as α decision = allow α | deny α. Thus, policies are three-valued: allow, deny, or $\bot$ (i. e., "don't know"). They can map input data to output data, refer to state, or be a policy-transforming policy.

Several combinators are defined in the UPF library providing definitions for families of *override* ($\_ \oplus \_$), *sequential composition* ($\_ \circ \_$), and *parallel composition* ($\_ \otimes \_$) of policies. These operators enjoy a wealth of algebraic properties like associativity, quasi-commutativity, or distributivity. We provide formal proofs, using Isabelle/HOL, for these properties.

The UPF is instantiated within HOL-TestGen/fw by concrete formats for TCP/IP packets, standard policies such as allow_all or deny_all, as well as combinators such as allow_port.

Network and Policy Models. Stateless firewall policies are modeled similar to common firewall configuration tools (see Brucker et al. [4] for details). After definitions of the relevant sub-networks (subsets of IP addresses modeling, e. g., the *demilitarized zone* dmz), it is for example straightforward to build a composition of elementary rules to be executed from left to right using the UPF override combinator. For example, we define a firewall policy P allowing only traffic by tcp from the internet to the dmz on port 25 or on port 80 formally:

$$P = \text{allow\_port internet dmz tcp 25} \oplus \text{allow\_port internet dmz tcp 80} \oplus \text{deny\_all}$$

Firewalls often perform not just stateless packet filtering, but also *packet translation* called network address translation (NAT), or a stateful handling of protocols—both is supported by HOL-TestGen/fw as well. An example of the latter is the file transfer protocol (FTP), where specific ports are opened and closed during protocol execution. Our policy modeling framework also provides support for modeling these concepts directly. Furthermore, the code-generators of HOL-TestGen/fw is able to generate firewall reference implementations in various programming languages directly.

Test Specification. For a policy P, a typical *test specification* looks as follows:

$$[\![C_1\ x; \ldots; C_n\ x]\!] \Longrightarrow FUT\ x = P\ x$$

where FUT is a placeholder for the firewall under test, which should behave like the policy P for all network packets x and $C_1, \ldots, C_n$ are constraints that restrict the test case generation to specific packets. For example, often it is desirable to exclude test cases that do not send packets across different sub-networks, or we might want to restrict testing to specific protocols.

2.2 Test Case and Test Data Generation

HOL-TestGen/fw can generate abstract test cases as well as concrete test data. This involves both normal form computations (resulting in test cases), and constraint solving (resulting in instantiations of the test cases, i. e., the concrete test data). While generic tactics for any models are available, the policy tactic library allows for a more *efficient* processing by using domain-specific knowledge. As a result of this phase, we obtain descriptions of network packets together with the desired decision, possibly extended by transformed packets. For our example policy P shown above, we state the *test specification*:

$$FUT\ x = P\ x$$

From this test specification, 24 test cases are generated automatically. Among them:

1. $FUT(12, (?X100, \text{tcp}, 6), (?X101, \text{tcp}, 80), \text{content}) = \llcorner\text{deny}\ ()\lrcorner$
2. $FUT(8, (?X102, \text{tcp}, 12), (?X103, \text{tcp}, 25), \text{content}) = \llcorner\text{accept}\ ()\lrcorner$

The variables starting with a question mark (e. g., $?X100$) are meta-variables representing a network address. In a separate step, we infer the actual test data from the test cases by finding ground instances that fulfill the constraints. For our two exemplary test cases, we might obtain the following test data:

1. $FUT(12, ((154, 23, 43, 2), \text{tcp}, 6), ((172, 0, 5, 3), \text{tcp}, 80), \text{content}) = \llcorner\text{deny}\ ()\lrcorner$
2. $FUT(8, ((154, 23, 43, 2), \text{tcp}, 12), ((172, 0, 5, 3), \text{tcp}, 25), \text{content}) = \llcorner\text{accept}\ ()\lrcorner$

We see that in our model, the description of a network packet is a tuple consisting of an identifier, a source address, a destination address and a content. Both the source and destination address consist of an IP address, a protocol, and a port number.

2.3 Test Execution and Test Result Validation

Next, the test data is injected into a network containing the firewall (or multiple firewalls) to be tested. The packet injection, the probing of the behavior, and the validation of the results are supported by the HOL-TestGen/fw test execution environment (see Figure 3). In more detail, the test execution environment con-

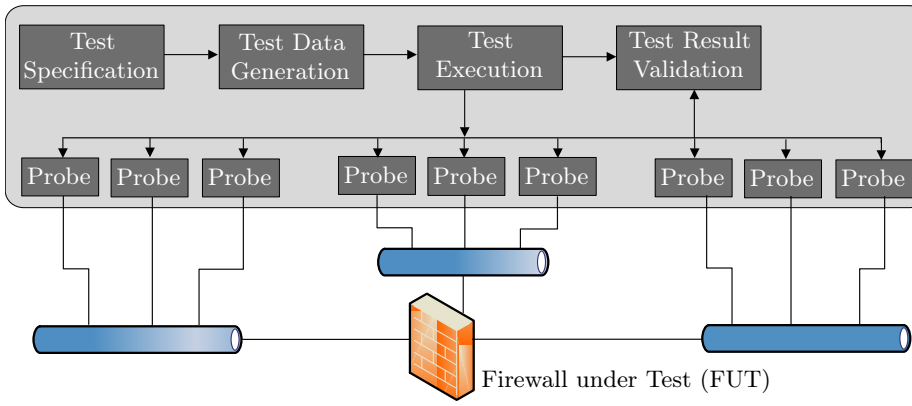


Fig. 3: A Framework for testing firewalls or routers

sists of a test execution manager, a result analysis module, a set of endpoints, and a set of probes.

Internally, the HOL-TestGen/fw execution environment uses an adapted version of fwtest (`http://user.cs.tu-berlin.de/~seb/fwtest/`) for injecting and observing network packets. Thus, the test data generated by HOL-TestGen is converted automatically to the input format of fwtest. As an example:

```
12:154.23.43.2: 6:172.0.5.3:80:S:TCP:10
 8:154.23.43.2:12:172.0.5.3:25:S:TCP:10
```

Here, the meaning of the individual parts (separated by a colon) is as follows: packet id, source IP, source port, destination IP, destination port, TCP flags, and the time to live. The test execution as well as the transformation of the input data is automated. Just some further information about the network topology that is not part of the model used to generate the test cases (e. g., IP addresses of the devices where the probes should be installed) has to be provided by the test engineer.

2.4 Verified Policy Transformations

The naïve approach presented so far does not scale very well in many cases and application domains; in many practical scenarios, the method takes too long and generates far too many tests resulting in a very long time for test execution.

There is an obvious need for speeding up both the test data generation as well as the test execution.

Our environment offers a solution to this problem called *verified policy transformations*, where a firewall policy is transformed into one that is easier to test but semantically equivalent, for example by eliminating redundant rules. As an example, consider the policy

$$\text{allow\_all dmz internet} \oplus \text{deny\_port dmz internet 21} \oplus \text{deny\_all}$$

which, as the rule deny_port dmz internet 21 is overridden by the first rule, allows all traffic from the dmz to the internet. Thus, this policy is semantically equivalent to the policy:

$$\text{allow\_all dmz internet} \oplus \text{deny\_all}$$

The second policy is much more efficient to test: it requires less time to generate test cases and test data and is, due to the smaller number of test cases, more efficient during the test execution phase.

Motivated by this and similar examples, we developed a rich theory of policy transformations that improve the testability of firewall policies (see Brucker et al. [2] for detail). A specific set of these transformations applied in sequence constitute a default normalization of firewall policies. All of these transformations are formally verified (using Isabelle/HOL) to be semantically correct and, as we will see in the next section, the normalization can increase the performance by several orders of magnitude.

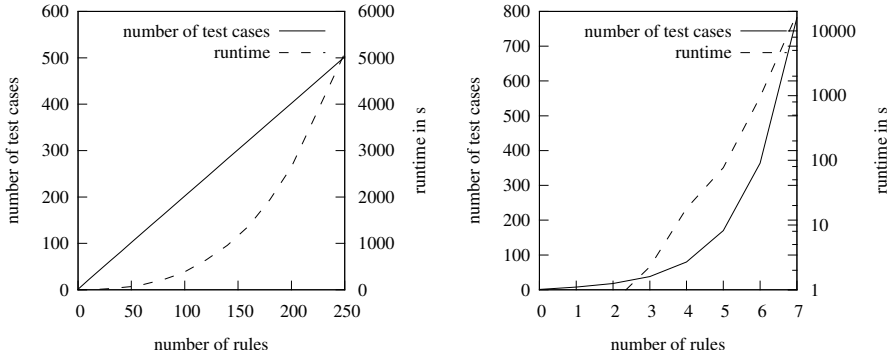
3 Case Studies and Evaluations

We used HOL-TestGen/fw in a large number of case studies. Those also included "real policies," for example some drawn from the network of ETH Zurich as well as some coming from collaborations with partners from industry. These large and complex policies revealed immediately the need for optimizations of the naïve approach and motivated us to develop the verified policy transformation approach presented above. Using the policy transformation, we were able to apply HOL-TestGen/fw in all our case studies successfully.

We analyzed the scalability issues as well as the impact of the policy transformation by applying HOL-TestGen/fw to randomly generated policies. This allowed us to estimate the correlation between the size of a policy and the generation time of tests, and to study the influence of various parameters of this correlation (e. g., different representations of network packets, number of networks) and, of course, the impact of our optimization. We discussed our generated policies as well as the generated test data before and after the optimization with experts from industry to ensure that our evaluation fulfills their needs.

In more detail, we applied HOL-TestGen/fw in the following scenarios that cover both industrial case studies as well as randomly generated policies to study for example the effect of different modeling variants.

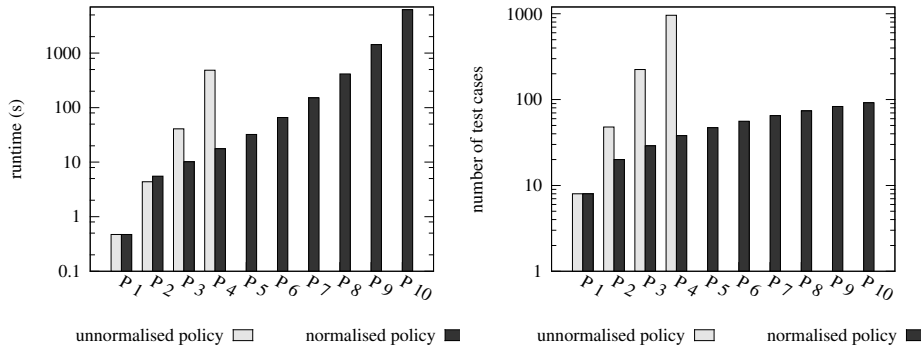
– *Packet Filter with Varying Number of Rules and Networks.* We tested "personal firewalls" of various sizes and network complexity. While for rules with low complexity, the naïve approaches works quite well (Figure 4a), it fails even for small policies with complex rules (Figure 4b). This observation motivated the development of our verified policy transformation approach.



(a) Rules with low complexity. (b) Rules of larger internal complexity

Fig. 4: Policy Complexity

– *Effect of Address Representation.* Our firewall models support different formalizations of network addresses. To study the effect of a more complex representation, we carried out the personal firewall scenarios with different address representations. From this experiment, we concluded that representing addresses as integers is the most efficient approach [7].
– *Packet Filter and NAT.* Motivated by needs from industry, we implemented support for network address translation (NAT). Technically, this is modeled as a parallel composition of a filtering and a translating policy. In practice, this does only add moderate overhead to test case generation as the translating policies are usually rather small.
– *Policy Transformation.* To address the scalability problem, we implemented a policy transformation approach which increases the efficiency of the naïve approach by several orders of magnitude (see Figure 5). In more detail, the transformation reduces the time required for generating the test cases and the test data (Figure 5b), as well as their number (Figure 5a). The latter also reduces the time required for test execution and validation.
– *Stateful Firewalls.* Several protocols, such as the file transfer protocol (FTP), Voice over IP (VoIP), or protocols for streaming multimedia data have an internal state; thus they are stateful. Such protocols require an (application-level) stateful firewall. HOL-TestGen/fw tests stateful firewalls by generating sequences of input network packets. Overall, this works quite efficiently; see Brucker and Wolff [5] for details.



(a) Policy transformation (time) (b) Policy transformation (test cases)

Fig. 5: Effect of the policy transformation

4 Related Work and Conclusion

Widely used tools for "testing" firewalls and other network components fall, broadly, into three categories:

1. Policy management and analysis tools, e. g., for optimizing polices or deploying the same policy to firewalls of different vendors. An example of this category is the Firewall Analyzer from AlgoSec (`http://www.algosec.com/`).
2. Tools that help to manage and analyze logs of firewalls, e. g., for detecting or analyzing security breaches. The Network Intelligence Engine from Network Intelligence (`http://www.network-intelligence.com/`) is an example for this category.
3. Tools that test for common misconfigurations (e. g., forwarding NetBIOS requests) or well-known vulnerabilities (e. g., exploitable buffer overflows of a specific firewall system). Examples for this categories are nmap (`http://www.nmap.org`) or OpenVAS (`http://www.openvas.org`).

These tools test for generic and well-known misconfigurations and security problems. In contrast to our approach, they do not base their test on the actual firewall policy. Thus, these tools complement HOL-TestGen/fw.

HOL-TestGen/fw supports the model-based conformance testing of firewalls. These conformance tests ensure both the correctness of the firewall implementation as well as the actual configuration of the firewall. The underlying foundations of the system as well as a more detailed report on the case studies is provided elsewhere [4].

Close to our work are tools that test for a firewall's conformance to a given policy. For example, [9, 10] present a policy segmentation technique to create the test cases. [11] also proposes a specification-based testing of firewalls, however the policies are restricted to stateless packet filters. Finally, [1, 14] present a framework for testing firewalls at the implementation level.

Acknowledgments. This research was partially supported from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318353 (EURO-MILS project: http://www.euromils.eu)

References

[1] von Bidder, D.: Specification-based firewall testing. Ph.d. thesis, ETH Zurich (2007). ETH Dissertation No. 17172.

[2] Brucker, A.D., Brügger, L., Kearney, P., Wolff, B.: Verified firewall policy transformations for test-case generation. In: Third International Conference on Software Testing, Verification, and Validation (ICST), pp. 345–354. IEEE Computer Society (2010). doi: 10.1109/ICST.2010.50.

[3] Brucker, A.D., Brügger, L., Kearney, P., Wolff, B.: An approach to modular and testable security models of real-world health-care applications. In: ACM symposium on access control models and technologies (SACMAT), pp. 133–142. ACM Press (2011). doi: 10.1145/1998441.1998461.

[4] Brucker, A.D., Brügger, L., Wolff, B.: Formal firewall testing: An exercise in test and proof. Submitted for publication (2013).

[5] Brucker, A.D., Wolff, B.: Test-sequence generation with HOL-TestGen – with an application to firewall testing. In: Meyer, B., Gurevich, Y. (eds.) TAP 2007: Tests And Proofs, no. 4454 in Lecture Notes in Computer Science, pp. 149–168. Springer-Verlag (2007). doi: 10.1007/978-3-540-73770-4_9.

[6] Brucker, A.D., Wolff, B.: HOL-TestGen: An interactive test-case generation framework. In: Chechik, M., Wirsing, M. (eds.) Fundamental Approaches to Software Engineering (FASE09), no. 5503 in Lecture Notes in Computer Science, pp. 417–420. Springer-Verlag (2009). doi: 10.1007/978-3-642-00593-0_28.

[7] Brucker, A.D., Wolff, B.: On theorem prover-based testing. Formal Aspects of Computing (FAC) (2012). doi: 10.1007/s00165-012-0222-y.

[8] Brügger, L.: A framework for modelling and testing of security policies. Ph.D. thesis, ETH Zurich (2012). ETH Dissertation No. 20513.

[9] El-Atawy, A., Ibrahim, K., Hamed, H., Al-Shaer, E.: Policy segmentation for intelligent firewall testing. In: NPSec 05, pp. 67–72. IEEE Computer Society (2005)

[10] El-Atawy, A., Samak, T., Wali, Z., Al-Shaer, E., Lin, F., Pham, C., Li, S.: An automated framework for validating firewall policy enforcement. In: POLICY '07, pp. 151–160. IEEE Computer Society (2007)

[11] Jürjens, J., Wimmel, G.: Specification-based testing of firewalls. In: Bjørner, D., Broy, M., Zamulin, A.V. (eds.) Ershov Memorial Conference, *Lecture Notes in Computer Science*, vol. 2244, pp. 308–316. Springer-Verlag (2001)

[12] de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS, *Lecture Notes in Computer Science*, vol. 4963, pp. 337–340. Springer-Verlag, Heidelberg (2008). doi: 10.1007/978-3-540-78800-3_24

[13] PIDE: (2013). `http://fortesse.lri.fr/index.php?option=com_content&id=91&Itemid=60#PlatformDevelopment`

[14] Senn, D., Basin, D.A., Caronni, G.: Firewall conformance testing. In: Khendek, F., Dssouli, R. (eds.) TestCom 2005, *Lecture Notes in Computer Science*, vol. 3502, pp. 226–241. Springer-Verlag, Heidelberg (2005)

[15] Wenzel, M., Wolff, B.: Building formal method tools in the Isabelle/Isar framework. In: Schneider, K., Brandt, J. (eds.) TPHOLS 2007, no. 4732 in Lecture Notes in Computer Science, pp. 352–367. Springer-Verlag, Heidelberg (2007). doi: 10.1007/978-3-540-74591-4_26