Model-based Firewall Conformance Testing

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

[1] SAP Research, Vincenz-Priessnitz-Str. 1, 76131 Karlsruhe, Germany
achim.brucker@sap.com
[2] Information Security, ETH Zurich, 8092 Zurich, Switzerland
lukas.bruegger@inf.ethz.ch
[3] Universität des Saarlandes, 66041 Saarbrücken, Germany
wolff@wjpserver.cs.uni-sb.de

Abstract Firewalls are a cornerstone of todays security infrastructure for networks. Their configuration, implementing a firewall policy, is inherently complex, hard to understand, and difficult to validate.
We present a substantial case study performed with the model-based testing tool HOL-TestGen. Based on a formal model of firewalls and their policies in higher-order logic (HOL), we first present a derived theory for simplifying policies. We discuss different test plans for test specifications. Finally, we show how to integrate these issues to a domain-specific firewall testing tool HOL-TestGen/FW.
Key words: Security Testing, Model-based Testing, Firewall, Conformance Testing

1 Introduction

It is common knowledge today that unrestricted access to the Internet is a security risk. Firewalls, i. e., active network elements that can filter network traffic, are a widely used tool for controlling the access to computers in a (sub)network and services implemented on them. In particular, firewalls filter (based on different criteria) undesired traffic, e. g., TCP/IP packets, out of the data-flow going to and from a (sub)network. Of course, their intended behavior, i. e., the *firewall policy*, varies from network to network according to the needs of its users. Therefore, firewalls can be configured to implement various security policies. Since configuring and maintaining firewalls is a highly error-prone task, the question arises how they can be tested systematically.

Several approaches for the generation of test-cases are well-known: while *unit-test* oriented test generation methods essentially use preconditions and post-conditions of system operation specifications, *sequence-test* oriented approaches essentially use temporal specifications or automata based specifications of system behavior. Usually, firewalls combine both aspects: whereas firewall policies are static, the underlying *network protocols* may depend on protocol states which some policies are aware of. This combination of complexity and criticality makes firewalls a challenging and rewarding target for security testing.

⋆ This work was partially funded by BT Group plc.

In this paper, we present a case study for the HOL-TestGen system: we model firewall policies as packet filtering functions or transformations over them in higher-order logic (HOL). Thus, we can cover data-oriented as well as temporal security policies. In contrast to a previous paper [5], where the underlying theoretical questions of this case study were settled (how can reactive sequence-testing be realized in a unit-test framework?), we present in this paper our firewall test-theory in greater detail. In particular, we present three aspects which we consider crucial for the treatment of larger test problems: first, we show how theory-specific rules can be safely derived, which greatly simplifies the partition space and its computation; in our case, this applies to the simplification of policies. Second, we present ways to express test-purposes within different test-plans of the same test specification leading to different test-cases. Third, we show how these various forms of support may be wrapped together to build a domain-specific extension of HOL-TestGen called HOL-TestGen/FW, i. e., a tool for model-based firewall conformance testing.

This paper is structured as follows: after introducing the foundations in Section 2 we will introduce our formal model of firewall configurations in Section 3. Thereafter, in Section 4, we show how this model can be used for model-based test-case generation and, moreover, build the basis for a domain specific test tool: HOL-TestGen/FW. In Section 5 we will present experimental data on several case-studies and develop test strategies. Finally, in Section 6, we compare to related work and draw conclusions.

2 Background

2.1 Isabelle and Higher-order Logic

Isabelle [11] is a generic theorem prover; new object logics can be introduced by specifying their syntax and natural deduction inference rules. Among other logics, Isabelle supports HOL (called Isabelle/HOL), which we choose as basis for HOL-TestGen.

Higher-order logic (HOL) [1, 6] is a classical logic with equality enriched by total higher-order functions. HOL is a language typed with Hindley/Milner polymorphism; type variables are denoted by α, β, γ, etc. HOL is more expressive than first-order logic, since e. g., induction schemes can be expressed inside the logic. Pragmatically, HOL can be viewed as a combination of a typed functional programming language like SML or Haskell extended by logical quantifiers. Thus, it often allows a very natural way of specification.

Isabelle/HOL provides also a large collection of theories like sets, lists, multisets, maps, orderings, and various arithmetic theories. Furthermore, it provides the means for defining data types and recursive function definitions over them in a style similar to a functional programming language.

2.2 The HOL-TestGen System

HOL-TestGen is an *interactive*, i. e., semi-automated, test tool for specification based tests built upon Isabelle/HOL. Its theory and implementation has been

described elsewhere [3–5]; here, we briefly review main concepts and outline the standard workflow. The latter is divided into four phases: writing the *test specification* TS, generation of *test cases* TC along with a *test theorem* for TS, generation of *test data* TD, i. e., constraint-free instances of TC, and the *test execution* (*result verification*) phase involving runs of the "real code" of the program under test. Figure 1 illustrates the overall workflow. Once a test theory is completed, documents can be generated that represent a formal test plan. The

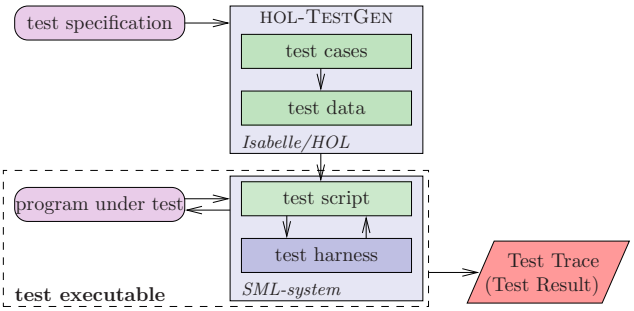


Figure 1. Overview of the Standard Workflow of HOL-TestGen

test plan contains the test theory, test specifications, configurations of the test data and test script generation commands, possibly extended by proofs for rules that support the overall process and can be processed either in batch mode or interactively. After test data generation, HOL-TestGen produces a *test script* driving the test using the provided *test harness*.

The core of the test-case generation procedure lies in the introduction of case splits up to a certain depth for each free or universally quantified variable in the test specification; depth and form of the case split depend on the type of the variable. The resulting formula is transformed into a CNF that is normalized. The test-data generation procedure is a constraint-solving technique based on a combination of arithmetic reasoning, simplification, and the insertion of random values for variables occurring in the test theorem.

2.3 Firewalls in a Nutshell

In a network, e. g., based on TCP/IP, a message from A to B is encapsulated in *packets* which contain the content of the message and routing information. The routing information of a packet mainly contains its source address (where does the packet come from), its destination address (where should the packet go to) and the protocol (e. g., http, smtp) used on top of the transport layer.

In its simplest form, a firewall is just a *stateless packet filter* which simply filters (i. e., rejects or accepts) traffic from one network to another based on the destination address, source address and the protocol, the *policy* used. The policy is the specification (usually given in a configuration file) of the firewall that describes which packets should be accepted and which should be rejected. In some cases, stateless filtering is not enough, some application protocols, like FTP

or most of the protocols used for Internet telephony such as Voice over IP (VoIP) have an internal state of which the firewall must be aware of. For example, some connections are only allowed within a specific state of the protocol.

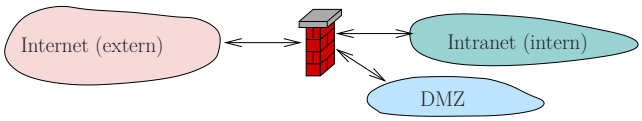


Figure 2. A simple firewalling scenario.

Figure 2 illustrates a widely-used setup of a firewall, separating three networks: the external Internet, the internal network that has to be protected (intranet) and a network that is somewhat in-between, the demilitarized zone (DMZ). The DMZ is usually used for servers (e. g., the Web server and the Mail server) that should be accessible both from the outside (Internet) and the from internal network (intranet) and thus underlie a more relaxed policy than the intranet. Table 1 shows a simple description of a firewall policy as it can be found in configuration files. Such a policy description uses a first-fit pattern matching

source	destination	protocol	action
DMZ	intranet	any	deny
Internet	DMZ	smtp	allow
Internet	DMZ	http	allow
intranet	DMZ	smtp	accept
intranet	DMZ	imaps	accept
intranet	Internet	http	accept
any	any	any	deny

Table 1. A simple Firewall Policy

strategy, i. e., the first match overrides later ones. For example, a packet from the Internet to the intranet is rejected (it only matches the last line of the table) whereas an smtp-packet from the intranet to the DMZ is accepted (fourth line of Table 1). The lines of such a table are also called *rules*; together, they describe the *policy* of a firewall. Since we consider *policy descriptions* as a kind of concrete syntax for *policies*, we will use these terms synonymously.

3 Modeling Firewalls in HOL

In this section, we present a formalization of firewall policies and present a formal theory for their simplification. Our model is inspired by the abstractions used by firewall configuration languages for TCP/IP networks, e. g., iptables (`http://www.netfilter.org`). The formalization is parametrized over the representation of network addresses for which we will discuss some alternatives in Section 4.

3.1 A Formal Firewall Model

Packets and Networks. As a prerequisite for modeling firewall policies, we need a formal model of protocols, packets and networks: we model protocols as an abstract datatype, e. g., the most common ones are declared by:

$$\text{protocol} := \text{ftp} \mid \text{http} \mid \text{voip} \mid \text{smtp} \mid \text{imap} \mid \text{imaps} \mid \text{unknown} \ .$$

As we do not want to depend on a specific representation of addresses and content, we introduce the abstract types $\alpha\,\text{src}$ and $\alpha\,\text{dest}$ of type $\alpha\,\text{adr}$ for the source and destination addresses and $\beta\,\text{content}$ for the content. We also introduce a unique identifier id for each packet. Thus, the type of a packet is defined as:

$$(\alpha, \beta)\,\text{packet} := \text{id} \times \text{protocol} \times \alpha\,\text{src} \times \alpha\,\text{dest} \times \beta\,\text{content} \ .$$

Further, we define projectors, e. g., getId, getSrc, for accessing the different components of a packet directly. As a next step, we model networks, or just *nets*, and parts thereof (*subnets*). To be as abstract as possible at this stage, we model nets as an axiomatic type class [11]. For the purpose of this paper, it suffices to know that a net is a set of sets of addresses, i. e.,

$$\alpha\ \ \text{subnet} := (\alpha :: \text{net})\,\text{set}\ \ \text{set}$$

where $(\alpha :: \text{net})$ requires that the types we use to instantiate α are members of the type class net. This definition allows us to model firewall policies that restrict the traffic between subnetworks and also between single hosts (addresses). For checking, if a given address is part of a subnet, we define the following operator:

$$a \sqsubset S \equiv \exists s \in S.\ (a \in s) \qquad \text{with type } \alpha\,\text{adr} \Rightarrow \alpha\,\text{subnet} \Rightarrow \text{bool}\,.$$

Firewall Policies. From an abstract point of view, a policy is a partial mapping of packets to decisions, e. g., deny or accept. Moreover, the datatype

$$\alpha\,\text{out} := \text{accept}\,\alpha \mid \text{deny}$$

allows also to model the modifications of return packets;[4] Our model can capture address-translation techniques (network address translation (NAT)) realized by some firewalls as well. The type of a policy follows directly from this:

$$(\alpha, \beta)\,\text{policy} := (\alpha, \beta)\,\text{packet} \rightharpoonup ((\alpha, \beta)\,\text{packet})\,\text{out}$$

where $\tau \rightharpoonup \tau'$ denotes the partial mapping (i. e., a type synonym for $\tau \Rightarrow \tau'$ option). Rules and policies have the same type, i. e., we can introduce a type synonym

$$(\alpha, \beta)\,\text{rule} := (\alpha, \beta)\,\text{policy}$$

[4] Usually, firewall policies describe more fine-grained how packets are denied, e. g., packets can be silently discarded (often called *drop*) or packets can be rejected (resulting in an error message on the sender side).

for rules. Moreover, an override operator for partial mappings ($\_ \oplus \_$) allows for nicely combining several rules to a policy. For example, $r_1 \oplus r_2$ combines the rules r_1 and r_2 where r_1 overrides (has higher precedence than) r_2. We can define several *generic rule combinators* at this abstract level that substantially simplify the formalization of concrete policies. For example, the usual two "catch-all" rules for accepting or denying all traffic are expressed as:

$$\begin{aligned} \text{allowAll}\, p &\equiv \text{Some}(\text{accept}\, p) && \text{with type } (\alpha, \beta)\,\text{rule, and} \\ \text{denyAll}\, p &\equiv \text{Some}(\text{deny}) && \text{with type } (\alpha, \beta)\,\text{rule.} \end{aligned}$$

Many other combinators for restricting traffic based on its source, destination, or protocol can already be defined on this abstraction level. A rule allowing all packets coming from subnet s can be defined as

$$\text{allowAllFrom}\, s \equiv \text{Some}\, \text{allowAll} \restriction_{\left\{p \mid (\text{getSrc}\, p) \sqsubset s\right\}}$$

with type $(\alpha :: \text{net})\,\text{subnet} \Rightarrow (\alpha, \beta)\,\text{rule}$, and where $\_ \restriction \_$ is the restriction operator on partial mappings.

IPv4. At this point, we decide for one possible packet address format, namely IPv4 addresses together with ports. In this setting, an address consists of a unique 32 bit number, represented as four-tuple and a port:

$$\begin{aligned} \text{ipv4Ip} &:= \text{int} \times \text{int} \times \text{int} \times \text{int}\,, \\ \text{port} &:= \text{int}\,, \\ \text{ipv4} &:= \text{ipv4Ip} \times \text{port}\,. \end{aligned}$$

Based on these definitions, we can define further combinators that are specific to TCP/IP addresses, i. e., they can accept or reject packets based on an IP address and a port.

3.2 Modeling Our Running Example

Our abstract firewall model, presented in the last section, allows for the direct formalization of the informal policy given in Table 1. First we have to define the subnets of type ipv4 subnet, based on their IP address ranges, e. g.:

$$\begin{aligned} \text{intranet} &\equiv \left\{ \left\{ \big((a,b,c,d),p\big) \,\middle|\, (a = 172) \wedge (b = 168) \right\} \right\} \text{and} \\ \text{dmz} &\equiv \left\{ \left\{ \big((a,b,c,d),p\big) \,\middle|\, (a = 172) \wedge (b = 16) \wedge (c = 70) \right\} \right\}. \end{aligned}$$

Grouping the rules of our informal policy with the same source and same destination, we define:

$$\text{DmzIntranet} \equiv \text{denyAllFromTo dmz intranet}$$

$$\begin{aligned}
\text{InternetDMZ} &\equiv \text{allowProtFromTo smtp internet dmz} \\
&\quad \oplus \text{allowProtFromTo https internet dmz} \\
\text{IntranetDMZ} &\equiv \text{allowProtFromTo smtp intranet dmz} \\
&\quad \oplus \text{allowProtFromTo imaps intranet dmz} \\
\text{IntranetInternet} &\equiv \text{allowProtFromTo http intranet internet}
\end{aligned}$$

The complete policy can then be defined as follows:

$$\begin{aligned}
\text{policy} \equiv \text{DmzIntranet} \oplus \text{IntranetDMZ} \oplus \text{IntranetInternet} \\
\oplus \text{InternetDMZ} \oplus \text{denyAll} \,. \quad (1)
\end{aligned}$$

This definition implies that the firewall does not take the port numbers into account for its filtering decision. Of course there do also exist combinators for that case and we could, alternatively, define:

$$\text{IntranetInternet} \equiv \text{allowProtFromPortTo http 80 Intranet internet} \,.$$

3.3 Simplifying Firewall Policies

The presented formalization of firewall policies is obviously not a canonical one, i. e., the same policy can be represented by many (syntactically) different maps. As an example, consider the following equivalence:

$$\text{denyAllFromTo } X\ Y \oplus \text{denyAll} = \text{denyAll}$$

where X and X are variables that are implicitly universally quantified, i. e., this equivalence holds for all possible values of X and Y. This observation leads to the idea of using rewriting techniques for simplifying firewall policies while preserving their semantics. For this purpose, we developed a set of equivalence rules that can be used by the built-in simplifier of the underlying Isabelle system. In more detail, the simplifier is configured to use such equivalences over the policy combinators as rewrite rules (from left to right). As we provide a large set of policy combinators, we also need a quite large set of equivalences over them. Thus, we can only summarize the rule set here.

One category of such equivalences are the ones handling global coverage, as the example above. Other such examples include the following:

$$\text{allowProtFromTo } p\ X\ Y \oplus \text{allowAllFromTo } X\ Y = \text{allowAllFromTo } X\ Y \,.$$

Another category of equivalences reduces the number of individual rules as it summarizes similar rules like in the following example:

$$\begin{aligned}
\text{allowProtFromTo}\, p\ A\ B \oplus \text{allowProtFromTo}\, q\ A\ B \\
= \text{allowProtsFromTo } \{p, q\} A\ B \,.
\end{aligned}$$

Together with several theorems about the associativity and commutativity of disjunct rules, we can derive a powerful policy simplification theory within the framework. E.g., our example policy can be simplified from seven to four individual rules. In detail, the policy gets simplified to

$$\begin{aligned}&\text{allowProtsFromTo } \{\text{smtp}, \text{http}\} \text{ internet dmz}\\ \oplus\ &\text{allowProtsFromTo } \{\text{smtp}, \text{imaps}\} \text{ intranet dmz}\\ \oplus\ &\text{allowProtFromTo http intranet internet}\\ \oplus\ &\text{denyAll .}\end{aligned}$$

The representation of a firewall policy (i. e., the selection of basic rules) influences both the runtime performance of a firewall and the decision during test-case generation. The simplifier-set presented in this section is aimed at reducing the number of decision points and thus makes the policy easier to test (see Section 5), i. e., results in smaller sets of test cases. On the other hand, the introduction of additional decision points (e. g., allowing to throw packets away earlier during the matching phase) can increase the overall performance of a firewall. Of course, such an optimization needs to take knowledge about the traffic into account and thus cannot be automated as easily.

4 Testing Firewall Policies

4.1 Testing Stateless Firewalls

The *test specification* for the stateless firewall case is now within reach: basically, we just state that the *firewall under test (fut)* has the same filtering function behavior as our combined policy from Definition 1:

$$fut(x) = \text{policy}(x)$$

However, this test specification is too general as we are only interested in a subset of the possible packets. The main reason is that firewalls sit *between* the subnets and therefore do not observe all the traffic. In particular, they will not observe packets with the source and destination within the same subnet.

Using a general logic as underlying framework, we can easily constrain the test space accordingly. If we want to test the firewall depicted in Figure 2, we use the auxiliary predicate:

$$\begin{aligned}\text{notInSameNet x} \equiv\ &(\text{srcOf x} \sqsubset \text{internet} \longrightarrow \neg\, \text{destOf x} \sqsubset \text{internet})\\ &\wedge (\text{srcOf x} \sqsubset \text{intranet} \longrightarrow \neg\, \text{destOf x} \sqsubset \text{intranet})\\ &\wedge (\text{srcOf x} \sqsubset \text{dmz} \longrightarrow \neg\, \text{destOf x} \sqsubset \text{dmz})\end{aligned}$$

This predicate of type *packet* → *bool* checks if the source and the destination of a packet are not within the same subnet. The test specification is revised to:

$$\text{notInSameNet } (x) \longrightarrow fut(x) = \text{policy}(x)$$

Depending on the *test purpose*, other constraints for the test specification are possible. For example, we might focus on test-data which represent packets sent to one single host, or only test-data for specific protocols. Thus, there may be different test specifications expressing different test purposes for the same policy. Distinguishing test purposes is especially useful for networks with firewalls at different points still enforcing one global policy.

After writing the test specification, some theorem proving techniques are necessary to bring the test theorem into a form which is suitable for the test-case generation; as these techniques are always the same for the default applications, this can be done fully automatically. In more detail, this includes the unfolding of the policy and of the rules. Technically, one can either unfold the constraints of the test specification before or after the test-case generation. In the first case, the undesired cases will not be generated while in the latter case they will be discharged later.

The test-case generation procedure, possibly followed by a simplification, produces after about 45 minutes running-time on a modestly equipped workstation, a list of 258 test-cases, among them the following two:

1. A test-case where a packet to the intranet must be denied by the firewall if the protocol is neither imap nor smtp:

$$\frac{X_2 = 172 \longrightarrow X_3 \neq 168 \qquad X_1 \neq \text{imap} \qquad X_1 \neq \text{smtp}}{\begin{array}{r} \mathit{fut}(X_4, X_1, ((X_2, X_3, X_5, X_6), X_7), ((172, 168, X_8, X_9), X_{10}), X_{11}) \\ = \text{Some deny} \end{array}}$$

   Here, the assumptions represent constraints for the concrete values chosen for the variables. In particular, the first part of the assumptions guarantees that the test-data generated from this test-case have a source network that is not equal to the destination network.
2. A test-case where the firewall should accept an smtp packet from the intranet to the dmz:

$$\begin{array}{l} \mathit{fut}(X_{12}, \text{smtp}, ((172, 168, X_{13}, X_{14}), X_{15}), ((172, 16, 70, X_{16}), X_{17}), X_{18}) \\ = \text{Some}(\text{accept}(X_{12}, \text{smtp}, ((172, 168, X_{13}, X_{14}), X_{15}), \\ \qquad ((172, 16, 70, X_{16}), X_{17}), X_{18})) \end{array}$$

We proceed with the test-data generation, which generates a constraint-free, ground instance of each test-case by random-constraint-solving. Unlike other examples, this step is quite trivial in our case and the computation time is negligible compared to the time used in the test-case generation. The procedure basically has to guess concrete values for the variables of the test-cases. Some of them are constrained (e. g., X_1 above), others are completely unconstrained (e. g., X_{12} above). Here is a sample of the generated test-data:

1. $\mathit{fut}(12, \text{http}, ((7, 13, 12, 0), 6), ((172, 168, 2, 1), 4), \text{content}) = \text{Some deny}$
2. $\mathit{fut}(8, \text{smtp}, ((172, 168, 12, 13), 12), ((172, 16, 70, 10), 6), \text{content})$
   $= \text{Some}(\text{accept}(8, \text{smtp}, ((172, 168, 12, 13), 12), ((172, 16, 70, 10), 6), \text{content}))$

The test data can be fed into a real firewall test driver. To sum up, a classical unit test scenario is adequate for stateless packet filters.

4.2 Testing Stateful Firewall Policies

For protocols like FTP, a stateless firewall can only provide a very limited form of network protection. The reason for this is, that FTP is based on a dynamic negotiation of a port number which is then used as channel to communicate the file content between the sender and the receiver. Thus, a stateful firewall is needed to observe the inner state of the port negotiation. Testing stateful firewalls, where the filter functions change, requires test-sequence generation also supported by HOL-TestGen.

The detailed model of a stateful firewall and protocols is presented in [5]. The basic idea is to keep the definition of policy but extend the model by a state, which consists of a pair of a *history* of accepted packets and the *current policy*. A state transition is a mapping from the packet that fired the transition and the current state to a new state. A state machine which models a specific stateful protocol can then be defined using predefined combinators.

In the stateful case, the test-data are *lists* of packets. The test specification can then be combined with the stateless case, e. g., perform a stateless testing before and after successful execution of the file transfer protocol (FTP). Clearly, the role of test purposes is even more important here.

4.3 Network Models

Besides the network model presented in Section 3.1, there are two notable alternatives; in the sequel, we evaluate them with respect to the generation time and test-case numbers.

Networks as a Datatype A possible abstraction from the network representation is to model subnets as elements of a finite datatype. This reads as follows:

$$\text{datatype networks} = \text{dmz} \mid \text{intranet} \mid \text{internet}$$

This model, built with ports or without, reduces the task of the test-case generation drastically; instead of case-splits for four independent Integers, only one is introduced. However, this representation abstracts away the possibility of single hosts since an additional random number generator at the level of the test-driver will be necessary to replace abstract networks against concrete addresses. Thus, different instances for one single host will be generated at runtime of the test.

IPv4 Addresses as one single Integer Although typically represented as a four-tuple of (mathematical) Integers, an IP address is technically simply a 32-bit bitvector. A replacement is straightforward: subnets are expressed as ranges of Integers. This formalization is compromise between the other two: the possibility to model single hosts is kept while the representation is simplified drastically. We provide a conversion between these two representations.

4.4 A Domain-specific Test Tool for Firewall Policies

So far, we presented a theory of networks, protocols and policies together with their use for generating test-cases. As HOL-TestGen is built on the framework of Isabelle with a general plug-in mechanism, HOL-TestGen can be customized to implement domain-specific, model-based test tools in its own right. As an example for such a domain-specific test-tool, we developed HOL-TestGen/fw which extends HOL-TestGen by:

1. a theory (or library) formalizing networks, protocols and firewall policies,
2. domain-specific extensions of the generic test-case procedures (tactics), and
3. support for an export format of test-data for external tools such as [13].

Figure 3 shows the overall architecture of HOL-TestGen/fw.

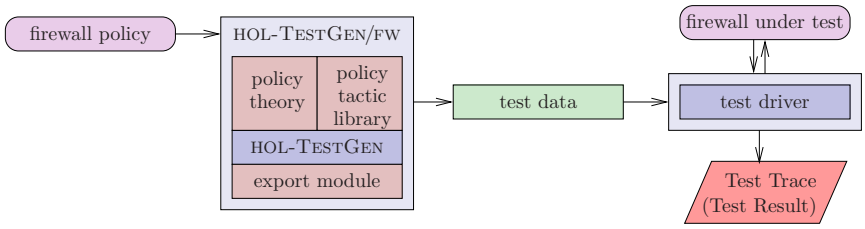


Figure 3. The HOL-TestGen/fw architecture.

In fact, item 1 defines the formal semantics (in HOL) of a specification language for firewall policies; see Section 3 for details. On the technical level, this library also contains simplification rules together with the corresponding setup of the constraint resolution procedures.

With item 2 we refer to domain-specific processing encapsulated the general HOL-TestGen test-case generation. Since test specifications in our domain have a specific pattern consisting of a limited set of predicates and policy combinators, this can be exploited in specific pre-processing and post-processing of an optimized version of the procedure, now tuned for stateless firewall policies. Moreover, there are new control parameters for the simplification (see Section 3.3).

With item 3, we refer to an own XML-like format for exchanging test-data for firewalls, i. e., a description of packets to be send together with the expected behavior of the firewall. This data data can be imported in a test-driver for firewalls, for example [13]. This completes our toolchain which, thus, supports the execution of test data on firewall implementations based on test cases derived from formal specifications.

5 Evaluation and Discussion

In this section, we report on experiments with HOL-TestGen/fw in several case studies. We will discuss the influence of different model parameters (e. g., network models, complexity of policies) on the number of test-cases and generation time. We conclude with a comparison of different test-strategies.

5.1 Case Studies

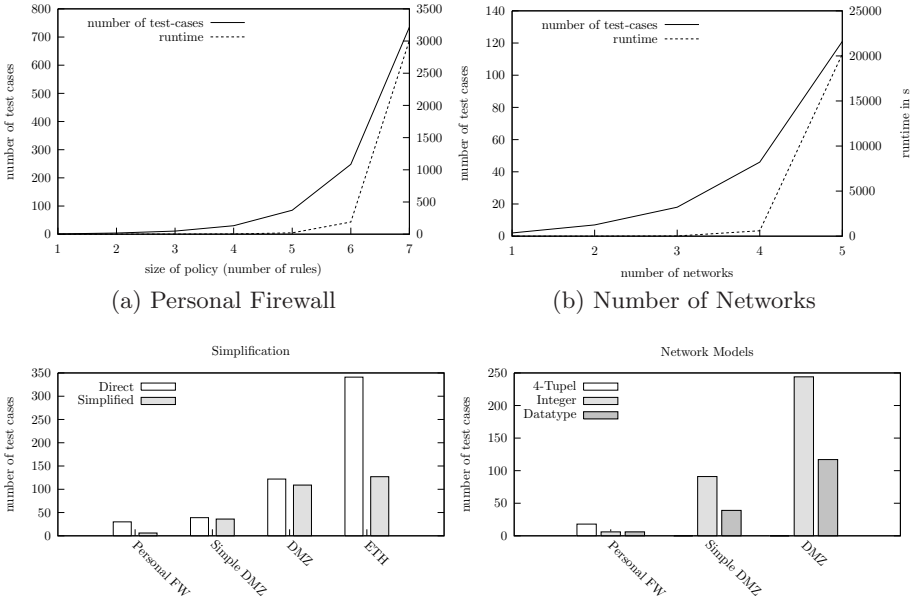
In the following, we will briefly summarize the results of the following scenarios:

Personal Firewall: We model a firewall running on a workstation, i. e., we only have two subnets, one representing the Internet and one representing a single workstation. This scenario is often called a personal firewall. A simple default policy for this model is to deny all traffic from the Internet to the workstation and to allow all traffic in the other direction.

Simple DMZ: A standard setup (similar to the example introduced in Section 2) with one internal network (intranet) that cannot be accessed from the Internet and one demilitarized zone which contains the servers that should be accessible from both the Internet and the intranet.

DMZ: We extend the "Simple DMZ" scenario by one crucial detail: the policy of each server in the DMZ is specified individually. Technically, this corresponds to the introduction of sub-subnets.

ETH: We model a firewall that is used in the computer science department at ETH Zurich: a real world example "as is." This example demonstrates that our approach is applicable for real world scenarios.



(a) Personal Firewall (b) Number of Networks

(c) Simplification (d) Comparing different Network Models

Figure 4. Evaluation

5.2 Influence of the Policy Size

We used the "Personal Firewall" case study for investigating how the size of a policy influences the number of test-cases and the overall time needed for calculating them. In this experiment, we keep the network setting and the network

model fix, and vary the policy by successively adding (non-overlapping) rules opening new ports from the workstation to the Internet.

Figure 4a suggests that both the number of test-cases and the time increases substantially with respect to the number of rules in a policy for two subnets.

5.3 Influence of the Number of Networks

By successively extending the "Personal Firewall" case study with additional workstations (subnets) using the same basic policy for every workstation (deny all traffic from the other networks to the workstation and allow all traffic from the workstation to the other networks), we studied how the number of networks influences the number of test-cases needed for testing the scenario thoroughly.

Figure 4b suggests that the number of subnets and hosts has a significant influence on the number of test-cases.

5.4 Influence of Policy Simplification

As we have seen, the complexity of a policy (number of rules) has a great consequence on the number of test-cases needed for full coverage of the specification. For justifying our policy simplification strategy, we compared the number of test-cases for several case studies (see Figure 4c).

In particular, the ETH Zurich firewall example shows that the effect of simplification for real-world scenarios can be dramatic. Of course, as explained in Section 3.3, our policy simplification can increase the network latency of deployed policies; and of course, not every policy can be simplified.

5.5 Influence of the Network Model

For investigating the influence of the network model (see Section 4.3) on the number of test-cases, we used three typical models and compared the number of generated test-cases for each model. Not surprisingly, Figure 4d reveals that the choice of the network model is most relevant. The choice to model an address as a four-tuple of Integers is on the one hand very nice as it is the closest one to the real-world representation. The costs, however, are substantial: in case of both DMZ examples, we stopped the computation after 6 hours with no result. Furthermore, we get a lot of additional test-cases which are probably in most scenarios superfluous. The possibility to model subnets as a datatype is clearly the simplest one and also significantly reduces computation time. However, we loose the possibilities to model hierarchical network topologies directly. For example, addressing single hosts, as in the DMZ example, requires special handling in the definition of the notInSameNet predicate. Therefore, we prefer the option of representing addresses by a single Integer: with only a little more complexity than in the datatype case, we keep the same expressive power as in the four-tuple case.

5.6 Discussion

Figure 4 summarizes the different scenarios; the details of all case studies appearing in this section and all measured data are part of the HOL-TestGen distribution (`http://www.brucker.ch/projects/hol-testgen/`). The presented case studies have demonstrated that our tool is well applicable to real world scenarios.

A classical goal of test-case generation is to minimize the number of test-cases. On the one hand, as the test-data generation itself needs only to be done once for every policy, one could argue that the time needed for computing the test-cases is not that important. On the other hand, von Bidder [13] reports that even only replaying test-data on a real firewall implementation already takes substantial time. Therefore, optimizing the set of test-cases is important, while preserving good test-coverage. Based on this experiments, we suggest to follow different test strategies, depending on the concrete test purpose:

- Our experience shows that testing, with complete decision coverage, of policies that are highly-optimized in terms of network latency, seems to be very expensive. As a compromise, we suggest to simplify the policy for testing purposes and to generate test-cases with respect to this simplified policy. While this approach still guarantees path coverage with respect to the specification, it does not cover all paths of the implementation. Nevertheless, by increasing the number of test-data chosen for every test-case, the coverage of the implementation could be increased. Overall, this approach results in a combination of model-based testing and random testing.
- For highly critical applications it might be worthwhile to install several firewalls with small policies that can be tested thoroughly. Moreover, as every firewall can be optimized for the small number of networks it connects, we expect a performance gain on the implementation level. Of course, installing several firewalls increases at least the initial costs; as maintaining small policies is much easier, we would expect that the total cost of ownership of such a setting is, at maximum, only a little worse than one centralized firewall. In fact, a similar setting was chosen for the network of ETH Zurich, where every research group has its own firewall.

6 Conclusion and Related Work

6.1 Related Work

Firewall testing is a widespread research topic which reflects their importance in todays security infrastructures. Whereas we focus on conformance testing, many research approaches are focused on vulnerability and policy-independent implementation testing. Surprisingly, we did not find any work on random testing the conformance of a firewall with respect to a policy; this is in stark contrast to software testing where random testing has gained a certain popularity.

Several approaches for specification-based testing of firewalls have been proposed. For example, El-Atawy et al. [7, 8] present a policy segmentation technique where a policy is represented as a tree. They also give some measurements

to the segments such that important segments can be tested more rigorously. They also present a policy generation technique. Jürjens and Wimmel [9] propose a specification-based testing of firewalls which employs a formal model of the network and automatically derives test-cases. It does however not really describe in which way these test-cases are generated. Furthermore, it does not support an ordering of the firewall rules. Bishop et al. [2] describe a formal model of protocols in HOL. Their level of abstraction is however much lower than ours and is therefore less suited for testing of policy conformance. Marmorstein and Kearns [10] propose a policy-based host classification which can be used to detect errors and anomalies in a firewall policy. Senn et al. [12, 13] propose a simple language for specifying firewall policies and a framework for testing firewalls at the implementation level. While the framework includes tools for generating test-cases with respect to a protocol specification, it lacks support for test-case generation based on policies.

6.2 Conclusion and Future Work

We presented a family of case study for HOL-TestGen consisting of a formal model of firewall policies in HOL, several problem-specific test plans and domain-specific tool support for generating test-cases. Our integrated approach allows for both the formal analysis of a policy, e. g., certain properties could be proven interactively, their automatic simplification, and the automatic generation of tests for real firewall implementations. We believe to have presented a successful application of our methods and tools to real-world scenarios.

The general domain of testing firewalls is both technically challenging and as well a rewarding target of security testing. Moreover, the variety of different firewall implementations, all based on the same set of network protocol specifications, makes firewalls especially well-suited for a model-based testing based on abstractions.

The presented work can be extended into various directions. In [5], we used sequence testing techniques for testing stateful firewalls using HOL-TestGen. This allows for the integration of unit and sequence testing, i. e., in every state of a test sequence a unit-test is executed.

On the theoretical side, our framework could be used for formally analyzing different separation techniques on the level of networks. For example, a common technique for reducing the amount of test-cases is to partition the policy into different fragments where every fragment only covers one pair of subnetworks. While this clearly reduces the amount of test labor, it also introduces a new kind of test hypothesis ("traffic between two networks does not influence the policy for other networks") into the system. Overall, this would allow to analyze formally the effects of a heuristic that is usually applied in an ad-hoc manner in firewall testing.

On the practical side, several extensions can be made: first, our integrated test-harness generator could be configured for generating test-data in a format that is suitable as input for the tools developed in the context of [13]. This would allow for testing the conformance of deployed firewalls. Secondly, our policy

specification can be used for generating configuration artifacts (e. g., scripts for iptables) for real firewall implementations and thus HOL-TestGen/fw could be used for testing and configuring real firewalls. And Thirdly, an integration of HOL-TestGen/fw into standard firewall configuration tools is possible. Like the second approach, this would allow for a policy specification language that is used for testing and configuration of a real firewall.

Acknowledgment

We would like to thank Martin Sedler for valuable discussions about the firewall settings at the computer science department, ETH Zurich (D-INFK/ETH).

Bibliography

[1] Peter B. Andrews. *Introduction to Mathematical Logic and Type Theory: To Truth through Proof*. Kluwer Academic Publishers, 2nd edition, 2002.

[2] Steve Bishop, Matthew Fairbairn, Michael Norrish, Peter Sewell, Michael Smith, and Keith Wansbrough. Engineering with logic: HOL specification and symbolic-evaluation testing for TCP implementations. In J. Gregory Morrisett and Simon L. Peyton Jones, editors, *POPL*, pages 55–66. ACM Press, 2006.

[3] Achim D. Brucker and Burkhart Wolff. HOL-TestGen 1.0.0 user guide. Technical Report 482, ETH Zurich, April 2005.

[4] Achim D. Brucker and Burkhart Wolff. Symbolic test case generation for primitive recursive functions. In Jens Grabowski and Brian Nielsen, editors, *FATES*, number 3395 in LNCS, pages 16–32. Springer, 2005.

[5] Achim D. Brucker and Burkhart Wolff. Test-sequence generation with HOL-TestGen. In Bertrand Meyer and Yuri Gurevich, editors, *TAP 2007*, number 4454 in LNCS, pages 149–168. Springer, 2007.

[6] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2):56–68, June 1940.

[7] Adel El-Atawy, K. Ibrahim, H. Hamed, and Ehab Al-Shaer. Policy segmentation for intelligent firewall testing. In *NPSec 05*, pages 67–72. IEEE Computer Society, November 2005.

[8] Adel El-Atawy, Taghrid Samak, Zein Wali, Ehab Al-Shaer, Frank Lin, Christopher Pham, and Sheng Li. An automated framework for validating firewall policy enforcement. In *POLICY '07*, pages 151–160. IEEE Computer Society, 2007.

[9] Jan Jürjens and Guido Wimmel. Specification-based testing of firewalls. In Dines Bjørner, Manfred Broy, and Alexandre V. Zamulin, editors, *Ershov Memorial Conference*, volume 2244 of *LNCS*, pages 308–316. Springer, 2001.

[10] Robert Marmorstein and Phil Kearns. Firewall analysis with policy-based host classification. In *LISA'06*, pages 4–4. USENIX Association, 2006.

[11] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL*, volume 2283 of *LNCS*. Springer, 2002.

[12] Diana Senn, David Basin, and Germano Caronni. Firewall conformance testing. In Ferhat Khendek and Rachida Dssouli, editors, *TestCom 2005*, volume 3502 of *LNCS*, pages 226–241. Springer, May 2005.

[13] Diana von Bidder. *Specification-based Firewall Testing*. Ph.D. Thesis, ETH Zurich, 2007. ETH Diss. No. 17172. Diana von Bidder's maiden name is Diana Senn.