Evaluation of Current Research in Firewall Analysis

Kamel Adi, Amy Felty, Luigi Logrippo, and Bernard Stepien

Executive Summary

A firewall is a software or hardware device that filters the information coming through an Internet connection into a private network or computer system. Only packets that have certain properties by which they are considered to be "safe" are allowed through, the others are not. We concentrate on software firewalls, which are scripts containing rules of the type: if a packet comes from address A and goes to port X, allow it through. If a packet comes from address B and goes to port Y, block it. There may also be complex rules that take into consideration the state of the connection, etc. For such scripts, it is often unclear which packets are allowed through and which not, if all possible cases are considered, etc. One important criterion is the detection of spoofing or impersonation, i.e. packets that appear to have been sent from an IP address but for which it can be determined that they cannot come from that address. The purpose of this work is to review the most significant research in the area.

After a review of the available literature, we decided to concentrate on four papers. The first two papers document systems that take sets of firewall rules and are able to answer questions about them. The second two relate to research on principles and methods to build firewalls.

The paper by Hazelhurst [Haz99] considers an encoding of firewalls rules into binary decision trees. The technique is effective, in the sense that it allows answering a number of significant questions that the firewall administrator may find important. It is also limited, for example it appears that spoofing cannot be detected with this technique. The method also does not address properties that depend on network topology, namely accessibility of a node from another.

The paper by Heronen and Zitting [EZ01] is a further step. It documents an expert system that, as the system described in the previous paper, works on already existing sets of firewall rules and is based on an encoding of them. The system allows to define high-level operations, especially in order to detect some types of configuration mistakes. It addresses most or all the properties considered in the previous paper, plus spoofing and network topology properties. We have implemented a simple demonstration to show the feasibility of this method.

The paper by Guttmann [Gut01] is more ambitious and also much more complex. It addresses the problem of automatically generating firewall rules from network topology and global policies, and the relationship between local policies and global policies.

The paper by Nielson et. al. [NNH02] presents a general and powerful "programming" language for expressing mobile computations; firewalls are one example of such computations. It is a mathematically rigorous approach, which allows proofs of correctness of firewalls (i.e., proving that they allow access to authorized users and deny access to attackers).

A recommended strategy is to implement the techniques documented in the papers by Hazelhurst, and Heronen-Zitting, and retain the last two methods for further study. Methods for automatically generating firewall rules from high-level policies (such as Guttman's) are particularly promising in the medium run because the firewalls generated in this way will be correct by definition. It may be advisable to obtain a copy of Guttman's tool from Mitre Corp.

and experiment with it. It should be possible to also take advantage of the power of Nielson et. al.'s approach by studying how to express rules written in well-known firewall languages directly as programs in the new language, and use this new representation to show the correctness of firewall configurations.

1 Introduction

Software firewalls are scripts that are used to control access to a private computer or network from an outside network. These scripts contain rules of the general type: if a packet comes from address A and goes to port X, allow it to pass. If a packet comes from address B and goes to port Y, block it. These fundamental types of rules can be made more complicated by using mechanisms such as "masks" and others. When a packet is received, the firewall rules are examined, top-down starting with the first one, in order see whether it meets the criteria of the rule. The first rule that is found to match the packet is applied, and this rule can prescribe acceptance or rejection. Usually a packet that does not match any rule it is rejected.

Firewalls of the type just described are called "stateless" because they do not keep any memory of the system state. More complex firewalls will keep information on previous packets or on the current phase of the communication protocol, and take decisions in consequence. These are called "stateful" protocols. Stateful protocols are more complex to analyze than stateless ones and are not considered in this report. They will certainly have to be considered in future phases of this project.

Firewall construction and validation is an important subject. Firewalls must be known to act accordingly to user policies or intentions, e.g. that a competing company cannot access another company's data base. However they are not written in this way. They are written as low-level rules stating that certain packets can or cannot pass. Currently, many firewalls are constructed by adding rules ad hoc as the need presents itself. Typical firewalls may consist of hundreds of rules, and may be poorly organized. Therefore, it may not be obvious at all whether a user policy or intention is satisfied by a firewall.

Firewalls can be analyzed in two ways. Passive analysis is done by analyzing firewall scripts and determining what they will allow and what they will not, while active analysis is done by sending packets with certain characteristics over a network to see if they are allowed through or not. The main methods for passive analysis are called "formal methods", as discussed below. Active firewall analysis instead is essentially a testing technique.

After a review of the literature, we decided to concentrate on the four papers discussed below: Hazelhurst [Haz99], Eronen-Zitting [EZ01], Guttman [Gut01], and Nielson et. al. [NNH02]. The four papers contain each a good discussion of a single method and they contain enough information that the level of replicability for each method is good.

2 Formal Methods in Firewall Analysis

Formal methods (also called formal techniques) in software engineering are methods that apply mathematical reasoning in software development, especially software design.

There are a considerable number of such techniques, and again they multiply themselves by the computer languages and application domains, since specific computer languages and application domains will require different concepts.

In general, the application of formal techniques is done as follows. A mathematical model, based on suitable concepts, is created for the software application being studied. Types of mathematical models used in this area are:

- Set theory
- Graph theory
- Predicate calculus
- Higher-Order logic
- Temporal logic
- Extended State Machines
- Process Algebras

Then a model is constructed of the software concepts in consideration, in terms of the chosen mathematical theory.

For example, if it is desired to study formally the possibility of sending information from a node to another, then graph theory might be used and the system might be represented as a graph where there is a path between two nodes if and only if information can flow between them.

As another example, if it is desired to study formally the possibility that a node that at some point will refuse some connection might perform a transition to a state where it will accept the same connection, then the node might be represented as a state machine. Events in the system will cause the node to change state. By considering all possible state transitions in the model, it may be possible to prove that the transition can or cannot occur.

Formal models are associated with automatic or semi-automatic verification tools. These tools work in very different ways, according to the mathematical theory on which they are based. They will accept as input the model for the system, the property that is desired to prove true or false, and end up with one of the following answers:

- The property was proven true or false
- There is not enough information to conclude true or false, more information is needed.

The efficiency of this procedure can vary greatly according to the complexity of the problem domain, the complexity of the property, or according to how the property was formulated in terms of the problem domain. The theorem prover may answer in seconds, or may keep working on the problem for days. As it goes, it will not be clear if it will provide a true/false answer or whether it will say that insufficient information was provided.

Theorem provers require the query to be set up for them, in their specialized language, and may require user interaction in order to perform difficult proofs. This interaction requires a sophisticated user, but it provides additional generality and flexibility in the kinds of properties that can be proved.

An issue with formal methods is that it may be very difficult or impossible to formalize completely the problem domain. The functioning of software systems may depend on complex properties that may be very lengthy to represent entirely in mathematical terms. Or there may be trivial but lengthy data representation details, encoding details, etc. that no one wants to take the time to write down formally. In these cases, the problem is simplified, and proofs are carried out on the simplification, hoping that they remain valid in the real system.

Although in principle a system that has been proven correct with a formal technique and associated tools does not need to be tested, in practice, because of what is mentioned in the previous paragraph, formal verification cannot replace testing. It is just an additional step to increase the confidence in the system.

Formal methods are not yet popular in the design of general software, because their use is seen as a “side trip” by programmers. Regular programmers will prefer to do a design and then code and then validate the code by debugging and testing, rather than spending time in constructing a formal model for their system, then using theorem provers to discover possible design errors, and finally manually code the formal model, for final debugging and testing. Not only this will be a side trip, but this method allows for the introduction of errors in the coding phase. It is at this point that the programmer will have to fill-in all the simplifications that were mentioned above.

The general use of formal techniques will be enhanced the day when it will be possible to derive code directly from the formal model. This is already possible today, to some extent, however as mentioned many formal techniques do not encourage expressing all the implementation details in the formal model.

Formal methods have found some popularity in the area of safety-critical systems. Designers of these systems have shown the extra motivation necessary to take the time to specify their system formally, completely or in part. The proof of formal properties of the system in discussion will serve as an additional element of confidence in the correctness of the system, other elements being provided by more conventional methods such as code inspection and testing.

Formal methods are well-integrated in the hardware design process, where they are commonly used to validate and debug component design.

3 Luigi Logrippo: Evaluation of paper *Algorithms for Analyzing Firewalls and Router Access Lists* by Scott Hazelhurst [Haz99]

3.1 General Description

This method considers a single set of firewall rules that can be expressed in the form **if**⟨condition⟩**then**⟨action⟩ and provides an encoding of the rules into a logical format, the Binary Decision Diagrams (BDDs). This type of rule is the most common in firewalls. BDDs allow quick derivation of answers for certain questions, thus providing a sort of expert system.

The paper also suggests the use of BDDs for the implementation of firewalls, which seems to be a good idea (compact representation, quick search).

The main advantage of this method is that it is able to explore quickly rule sets of realistic size.

About BDDs. BDDs are a space-efficient representation of binary Boolean trees. For example, two identical subtrees in a regular binary tree are collapsed into one subtree in BDDs; also paths leading to false verdicts are pruned from BDDs. For this reason, BDDs require algorithms that are different from the algorithms used for binary trees. Search is quicker on BDDs because these use less space than regular binary trees. However from a logical point of view, BDDs are just as expressive as regular binary trees.

3.2 Criteria for Classification of Method

Type of Configuration Problem Addressed. This tool can be seen as a simple expert system that is capable of answering questions on the types of packets allowed or excluded by a set of firewall rules. The method is also capable of considering two sets of firewall rules and do comparisons (are they equivalent? Does one of them accept packets that the other rejects?). Finally, it is capable of performing a form of limited validation. These points are developed below. Note that, opposite to conventional expert systems that use heuristics, this tool uses only purely logical deductions.

Network Topology. Network topology is not considered in this method.

Firewall Model or Architecture Addressed. The specific tool presented uses the syntax of CISCO firewall architecture. But the technique is more general. It can be applied to any set of rules expressible in the form **if**⟨condition⟩**then**⟨action⟩, although new encoding rules may have to be devised for new needs.

Mathematical Basis or Theory. Propositional logic encoded in Binary Decision Diagrams (BDDs). Standard BDD tree search algorithms used.

Completeness of Network Model. The system seems to be fairly complete within its parameters. It is capable of expressing and analyzing sets of rules of the form if-then-else, but:

- It will not be able to express "stateful" firewalls.
- It does not analyze or represent the network topology

Granularity of Network Model. The author of this paper provides detailed encodings of many elements of the firewall rules, so this method provides fine granularity.

Case Study Results The author provides the following examples of questions that can be answered by the tool:

- Do we accept packets on port 25? If so, which type?
- On which port do we accept TCP packets?
- Which packets do we accept from address y?
- Which type of packets will we accept that are being sent to address y?
- What type of UDP packets do we accept?
- What packets do we accept that have the first segment of the destination address of 121 and which are not ICMP packets?
- List the packets we accept for the 121.120 network
- Which packets are not accepted?

The author also says that his tool has been used on a CISCO firewall consisting of 430 rules provided by an Internet Service Provider.

Several types of firewall validation seem to be possible, but the author seems to have implemented only one: the detection of redundant rules (rules for cases that are already covered by other rules).

The method is also capable of taking two sets of firewall rules and answering questions such as: which packets are allowed by one rule set but not the other; or vice-versa.

Admitted Limitations. The simple rule format considered by this tool is insufficient to represent the logic of some types of rules in modern firewalls. For example, the author mentions that they cannot express rules that use information in higher layer protocols.

Perceived Limitations. The type of BDD used by the author does not represent variables. A rule of the type x=y must be expanded for all possible values of x and y! Note that variables seem to be necessary for spoofing detection (real origin address different from stated origin address: two variables involved). Extensions of the BDD method to deal with variables have been studied, but this paper does not consider them.

Languages. The tool takes as input CISCO firewalls. It can be adapted to be used with other types of firewalls.

User Interface. A graphical user interface based on Tcl/Tk has been implemented.

Replicability. Many details are given on the method. It appears that the replicability is good.

Service, Accessibility, and Traffic. See examples of queries in the section Case Study Results.

Rules that are Never Matched. Method detects:

- Repeated rules
- Rules that cover cases already covered (implications). An example given is mask values in one rule that cover subsequent rules.

Rules that are never matched are not explicitly mentioned in the paper, however by the nature of this method it seems that such rules are detectable.

Spoofing. No. See section on Perceived Limitations.

Easily Extendible. Method and tool seem to be easily extensible, although it is not clear what could be done beyond what has already been done by author.

Tools available, in-house or off the shelf, or no tools. The specific BDD tool used is VOSS, available from Carl Seger of the University of British Columbia. VOSS is a system targeted to hardware validation. However the program written by the author by using VOSS is not available.There are other tools for BDDs that are freely available. The author has developed:

- a program for translating firewall rules into BDDs
- a graphic interface for his tool,

but by his own recognition these are prototypes and almost certainly unavailable to others.

Citation Relationship to Other Papers. Does not cite any firewall paper. Cited by Eronen and Zitting [EZ01].

3.3 Discussion

This seems to be a simple, effective method, although limited. A firewall validation toolkit could start by implementing this method. Other methods, such as those based on expert system concepts, could be built afterwards.

4 Bernard Stepien: Evaluation of paper *An expert system for analyzing firewall rules* by Pasi Eronen and Jukka Zitting [EZ01]

4.1 General Description

This method is based on the use of the principle of expert system where firewall access-lists are converted directly into a expert system knowledge base. The resulting knowledge base is then manipulated using the underlying inference engine of readily available Logic Programming language interpreters, in this case CLP (Constraint Logic Programming) that sits on top of the Prolog language. The use of CLP has some additional benefits in the case of firewall analysis in that it can query the knowledge base on constraints rather than on simple facts.

4.2 Criteria for Classification of Method

Type of Configuration Problem Addressed. This is a pure expert system that is centered on the concept of queries on a CLP/prolog knowledge base. There are four main areas of problems addressed:

- network properties
- configuration properties
- access-list properties
- defining new predicates for higher level queries

For each of these four groups it answers a number of questions:

Network properties:

- Which services are accessible on a given host?
- Is a given host/network accessible from another given host/network?
- What kinds of traffic are allowed between two networks?
- From which network is a given host accessible?
- What rules permit packets from this network?
- From which sources are packets to this destination permitted?

Configuration properties:

- Firewall doesn't block directed broadcasts
- Insufficient prevention of address spoofing

- Domain Name Server (DNS) is reachable only by UDP, not TCP.

Access-list properties:

- What packets are permitted for this access list?
- Detect access-list rules that are never matched
- Why is this packet denied?

Defining new predicates:

- Capability to add higher level rules using the knowledge base.
- For example for the spoofing detection problem.

Network Topology. Provides capabilities to express the network topology.

Firewall Model or Architecture Addressed. This method uses a one to one mapping between firewall access-lists and an expert system knowledge base. However, the rules are decorrelated to allow handling of ranges that occur when handling sub-networks specifications.

Mathematical Basis or Theory. The authors use constraint logic programming (CLP). This is an extension of traditional logic programming (LP). LP is based on resolution theorem proving in the predicate calculus. In a logical reading, a query Q is either deductible or not from a set of clauses P; if it is, then variables appearing in the query are specified by the resulting unifications. Constraint logic programming languages are LP languages in which unification is replaced by constraint solving in various domains.

Completeness of the Network Model The network model is complete.

Granularity of the Network Model The authors of this paper propose detailed examples of the transformation steps and the various application areas of the methodology, so this method provides fine granularity. However, while there are ample details for the actual manipulation of access-lists for a single firewall element, the example of the use of network topology is insufficient to make a real implementation.

Case Study Results The authors claim that the main benefits of their system is the use of logic programming and a generic inference engine. The ease of the firewall rule translation into a knowledge base and the use of the generic inference engine reduces considerably the development effort and allows to focus on expressing knowledge and building the user interface.

Admitted Limitations. This methods has been used only on Cisco access list. Thus, it is not a multi- vendor tool. Also, this tool is only a prototype and has no commercial strength pretentions.

Perceived Limitations. There were no perceived limitations in the methodology. Logic Programming is well known to us and it appears that the authors have used it correctly.

Languages. The authors are using CLP in the context of commercially available Eclipse.

User Interface. The user interface consists in the natural capability of the Prolog/CLP language interface. This consists in the ability to invoke a Prolog rule with uninstantiated variables to obtain answers to a given problem and with instantiated variables to verify that a particular problem is true. The CLP language has some additional natural user interface capabilities that are not elaborated by the authors in their paper.

Replicability. The method seemed to be relatively easily replicable. We have partly replicated it using the logic programming Prolog only. Some adjustments were necessary to enable constraint logic to be queried but these are well known to our group. However, a true replication would require CLP in order to to take full advantage of constraint logic and cut down development time in avoiding ways around in Prolog. The authors mention that a prototype was built in two weeks thanks to CLP platform Eclipse. Eclipse is a commercially available tool for which there are license fees.

Rules that are Never Matched. The Logic programming paradigm is the ideal solution for the rules are never matched problem. There are two ways to answer this question: the first one consists in using the backtracking capabilities of LP languages to generate all solutions to a particular problem. This will produce all of the firewall rules that match a given specification. By definition we know that only the first match is considered by a firewall, thus all subsequent solutions constitute the body of rules that are never matched. The second way consists in invoking the same query with twice, once with the permit action and another time with the deny action. If the LP returns a solution, this means that there are conflicting rules in the access-list and that any rules except the first one is not matched.

Spoofing. The authors provide an example on how to detect spoofing with this method.

Easily Extendible. In LP and CLP new queries can naturally be built using combinations of other already existing queries. Thus the extendibility can be considered as unlimited.

Multi-Vendor Firewalls. The authors stated that their method does not address multi-vendor firewalls. However, this is not a major issue as the methodology in general can be applied to any type of firewall specification system. In this case, we would need to create a meta language that would enable to express features of some firewall specification languages into some common denominator language. This is a well know methodology that has been described in some papers that are not within the scope of the four papers reviewed for this exercise.

Tools available, in house or off the shelf, or no tools. In house, interactive tool on top of commercially available tool called Eclipse. The method requires three steps: first a conversion from firewall rules to a prolog like knowledge base. This was done by the authors using the common Perl language. Second, the manipulation of the resulting knowledge base using Eclipse. Third, write a number of user interface predicates to make higher level queries which is a common feature of logic programming languages.

Citation Relationship to Other Papers. The authors are fully aware of Hazelhurst BDD approach and officially declare that the Hazelhurst method is less flexible than theirs. They also mention Guttman but without providing any remarks in how it differentiates from their method.

5 Kamel Adi: Evaluation of paper *Security Goals: Packet Trajectories and Strand Spaces* by Joshua D. Guttman

This section summarizes the formal techniques developed by Joshua D. Guttman [Gut01] for the network packet protection problem. More precisely it addresses two main techniques: one for automatically generating router's local enforcements from a network topology and global policies and the other compares a set of filters to the global network access control policy to determine a policy violation if any.

5.1 Introduction

The paper addresses the issues of network access control policies and their enforcement. It follows systematic and rigorous methods to achieve these objectives. Security goals are first pointed out and then a modeling process is presented.

The presentation is related to the security provided by filtering routers where the main goal is to demonstrate how to use the security services provided by packet filtering routers to achieve useful protection in complex networks. A simple language for expressing global network access control policies is first presented, followed by a first algorithm that is capable to compute local filters for individual routers from global network access control policies and a network topology. A second algorithm compares a set of filters to the global network access control policy to determine all policy violations, or to report that none exist.

The major aim that filtering routers strive to achieve, is to prevent the flow of packets, which can harm recipients. A router has to take a decision action by examining only headers of packets to detect harmful information.

5.2 Security Goals

Two types of information can help to implement security goals: information concerned with the header of the packet and information concerned with the path that the packet has taken through the network. It should be noted that the path information is not contained in the packet itself. Thus, a router is not able to observe the whole path but only a piece of the path.

The behavior of a router depends on the interface by which the packet is sent and received. The goal to achieve then, in a wide-network, is to impose different filtering strategies at different points in the network. The key point is that attention is made only to a portion of the path, that is, whether the packet was previously in a network not trusted or the packet has reached a region we would like to protect.

Security goals combine information about paths and packet headers. They are always a concern between two network areas n and n'. If a packet has traversed a path from n to n' then its header fields must have some property ϕ. Then, a security goal can be of the following form:

if p was in the network n and later reached the network n'
then the header of p satisfies ϕ

The security policy implies a security goal for every two network areas. Formally this can be represented as a function:

$$\pi : A \times A \longrightarrow B$$

where A is set of network areas and B is a collection of sets of packets.

5.3 Security Modeling

The network where security policies are going to be enforced, is modeled as an undirected bipartite graph where the edges represent interfaces and a node can be a router or a network area, i.e. a router at one end of an edge and the network area at the other end.

There is a filter on each edge at each direction to dispose of unwanted packets. The filter has to distinguish permitted packets from those, which are not allowed to go through.

Thus, there are two main elements, which need to be modeled:

- the network is modeled by an undirected graph and
- the constraints are modeled by a Boolean algebra.

There are two problems, which need to be solved using the model just mentioned. The first concerns the posture filtering verification, i.e., checking whether a specific task assigned to a filter, meets the security policy in place. The second concerns posture filtering generation, i.e., generate a posture that meets the security policy.

5.4 Posture Filtering Verification

The problem of posture filtering verification can be reformulated as follows: if a packet survives all the filters between the two networks n and n', then the packet must satisfy the policy constraints.

Posture verification tries to answer the question: given a policy π for a wide area network does the posture f assigned to a particular filtering router r, in this network, indeed implement the policy π. The algorithm is as follows: start at each network area n and list all the paths p from n leading to another area n'. Let $F(p)$ be the feasibility set for the path p between two areas n and n'. $F(p)$ stands for the set of packets that survive all the filters that are traversed while following the path p. If $F(p) \not\subseteq \pi(n, n')$ then report the violations as the difference of $F(p)$ and $\pi(n, n')$. If $F(p) \subseteq \pi(n, n')$ then F enforces the policy π.

5.5 Posture Filtering Generation

The posture filtering generation attempts to generate a posture f from a policy π. The algorithm is as follows: start with an initial posture f_0 and then correct it successively to eliminate all violations. To correct a posture, enumerate all the paths p that start and end at distinct areas (n, n') using the current posture f_i calculate $F(p)$. If $F(p) \subset \pi(n, n')$ then $f_{i+1} = f_i$. Otherwise take out the violations by setting the difference of $F(p)$ and $\pi(n, n')$.

The tool called NPT (Network Policy Tool) is an implementation of these two algorithms. The NPT combines posture generation with posture verification. One can create a posture and then verify it and if he is not happy with the outcome he will modify the posture and try again.

5.6 Criteria for Classification of Method

Type of Configuration Problem Addressed. The general problem solved is to restrict access to some private networks from the outside world. In particular, the paper answers two fundamental questions:

- How can we compute local enforcements, given a network topology and a global network access control policy?
- Does a set of local enforcements meet a global network access control policy?

Network Topology. The methodology does not specify which topology it supports best. However, it is implicit that it can be used in any network topology.

Firewall Model or Architecture Addressed. The architecture addressed models a network as a bipartite graph where nodes represent distinguished network areas together with the routers that connect theses areas. Filtering behaviors of routers are modeled by logical rules.

Mathematical Basis or Theory. This paper uses formal mathematical concepts, basically set theory, logic and graph theory.

Completeness of Network Model. The model proposed cannot express requirements on packets traveling within a single area.

Granularity of Network Model. The granularity of the network model is well developed. It starts from a high policy level of a WAN down to a low level i.e., the filtering router of a local network.

Case Study Results. The NPT prototype has been tested on a realistic example modeling a wide area network with 11 sub-areas. The local enforcement for each router has been generated successfully in 17.3 seconds on a HyperSparc processor machine at 125MHz.

Admitted Limitations. The generated posture output is sometimes unnecessarily complex. This requires a human intervention to make manually local improvements.

Perceived Limitations. The work does not mention how to protect the routers themselves. It only deals with the hosts behind the firewall and not the routers.

Languages. The method does not use existing firewall languages; rather a new logical formalism is presented to specify filtering behaviors of routers. This language has a Lisp-like syntax. Note that the language is good only for the used verification tools and one should map the behavioral specification to an access list notation understandable by filtering routers. To encode the two prototypes NPT and Atomizer, the author uses respectively, as programming languages, ORBIT, an optimizing compiler for Scheme and OCaml, an implementation of ML.

User Interface. Not applicable.

Replicability. The algorithms discussed in the paper, are given in a complete formal detail, so a tool based directly on these algorithms could be implemented without any further information. It seems also to be possible to modify the original implementation in order to make it fulfilled with slightly different requirements.

Service, Accessibility, and Traffic. Although the author focuses only on some TCP/IP services, the methodology can be used to specify any kind of network services and then to be verified by the tools. The allowed traffic between two networks is defined by the policy assigned to the filtering router, which connects the two networks. The accessibility to a host in a particular network depends on the global network policy. The global policy also specifies from which network a given host is accessible.

Rules that are Never Matched. It is possible to detect firewall rules that are never matched by suppressing manually these rules from the router access list and do verification by using the Atomizer and NPT tools to see if there is any violation of the global policy.

Spoofing. It is possible to prevent spoofing by enforcing the correct rules at the right router.

Easily Extendible. Seems to be the case.

Multi-Vendor Firewalls. Not applicable. No vendors are discussed.

Tools available, in-house or off the shelf, or no tools. The paper develops two main algorithms in detail. The first algorithm is used to compute a set of filters for the individual routers, given a network topology and a global network access control policy. The second algorithm compares a set of filters to the global network access control policy to determine all policy violations, or to report that none exist.

Two tools are used: Atomizer and NPT. The atomizer tool, analyzes router configuration files and then generate a set of abstract packets (by abstracting IP addresses and services). This abstract set is to be used by NPT. The NPT (Network Policy Tool) computes a set of filters for the individual routers given a network topology and a global network access control policy. The combined tools provide a method to analyze actual router configurations and to generate filtering rules.

The Atomizer and NPT tools described in the paper are available for license from MITRE corporation (www.mitre.org).

Citation Relationship to Other Papers. Only one from the papers currently under study cites this paper:

- An expert system for analyzing firewall rules. Pasi Eronen and Jukka Zitting.

Other papers from the literature cited the evaluated paper:

- Automatic Generation of IPSec/VPN Security Policies In an Intra-Domain Environment. Zhi (Judy) Fu and S. Felix Wu.
- Automatic Management of Network Security Policy. J. Burns *et al.*
- The Multilayer Firewall. Dan Nessett and Polar Humenn.
- Specification-Based Testing of Firewalls. Jan Jrjens and Guido Wimmel.
- Firmato: A Novel Firewall Management Toolkit. Yair Bartal *et al.*

5.7 Discussion

The presented methodology is an important step towards the development and enforcement of network policies. The method is based on logical reasoning and supported by two complementary tools to verify policy designs. However, the methodology is not detailed enough to be easily applied. An experiment with it would be useful to understand the different concepts and to give guidelines for better use.

6 Amy Felty: Evaluation of paper *Validating Firewalls Using Flow Logics* by Neilson et. al

This section summarizes the techniques developed by Nielson, Nielson, and Hansen [NNH02] for using flow logics to validate firewalls.

6.1 Summary of Method

A process calculus which includes features especially suited for modeling firewalls is presented. The calculus is a variant of Cardelli and Gordon's ambient calculus [CG00] which allows the modeling of processes which move between sites.

The ambient calculus captures the notions of mobility, location, and authorization to move. The motivation for developing this calculus came from the potential for mobile computation over the web. Such mobile computations do not migrate freely. Often a process must exit its own administrative domain and enter a new one, and also enter a protected area of a particular machine where it is allowed to execute. It may be the case that all three of these steps require obtaining permission. Sometimes multiple levels of authorization are required. Administrative domains and particular machines are both examples of entities which are modeled as locations in the ambient calculus. In addition, the ambient calculus models two kinds of mobility, one where the code is moving between sites, and another where the devices involved, such as laptops, may move around.

The flow logics paper contains a running example of a simple firewall that is modeled in the new process calculus, which is a variant of the ambient calculus. This example is presented and explained in Section 6.4. The process calculus itself can be viewed as a programming language, and the firewall example as a program written in this language. The program actually consists of a firewall and an agent that has the correct passwords or keys, and illustrates that such an agent can cross the firewall.

Validating such a model of a firewall consists of formally proving that an agent with correct keys can indeed cross the firewall, and also showing that any agent without the correct keys cannot cross the firewall. The rest of the paper is devoted to presenting algorithms for automatically proving each of these two validation criteria.

Computation in this calculus is given as a set of reduction rules. To illustrate them, a series of reductions is applied to the running example. In this computation, the agent with the passwords enters the site protected by the firewall.

The first validation problem, i.e., showing that an agent with keys is indeed permitted to enter the site protected by the firewall is solved using a "control flow analysis". This analysis examines all possible reduction sequences, and determines for each process, which sites it may be contained in, and which actions it may perform, and for each site, which operations are permitted to be performed inside of it. (Actually, sites and processes are not distinguished in the calculus. They are expressed using the same formalism, and are called "ambients".) The algorithm given is shown to guarantee cubic time computation of the solutions (cubic in the size of the program).

The second validation problem is harder, since it must be shown that *any* process not knowing the right keys cannot enter. To handle this problem, a notion of "unaware attacker" (i.e. an attacker that does not know the keys) is defined. More importantly, it is shown that the calculus is powerful enough to define a "hardest attacker". Such an attacker is one that can be shown to be as hard to protect against as any other unaware attacker. Using this notion, the problem of showing that any unaware attacker is prevented from entering is reduced to the problem of showing that a single hardest attacker is prevented from entering.

6.2 Criteria for Classification of Method

Type of Configuration Problem Addressed. As stated, the method answers the two questions:

- Can an agent with proper keys enter the site protected by the firewall?
- Are *all* processes without the right keys prohibited from entering the site protected by the firewall?

Network Topology. The network topology can be encoded in a particular program written in the ambient calculus.

Firewall Model or Architecture Addressed. A very general formalism is defined, which should allow a variety of models and architectures to be expressed.

Mathematical Basis or Theory. This paper is quite rigorous mathematically. It describes a variant of the ambient calculus, which is in the class of languages known as process calculi.

Completeness of the Network Model. The network model is described as a program in the ambient calculus presented in the paper. It is a fairly general programming language, and thus it is possible to program a wide variety of firewalls.

Granularity of the Network Model. The programming language is fairly high-level, which means that it is easy to express models at the level of coarse granularity. More fine-grained models may also be possible, but examples are not given.

Case Study Results. An example taken from the paper is described in detail in Section 6.4.

Admitted Limitations. There is very little discussion of limitations. It is a new approach, and the authors state that they are hopeful that it will scale up to various aspects not yet considered such as "other calculi and web-based languages with explicit cryptographic primitives" ([NNH02], page 415). They also list as future work the possibility to express more powerful analyses so that it becomes possible to identify attackers more precisely and the goal of showing that a firewall protects against attackers who know some but not all the passwords.

Perceived Limitations. There are several limitations not discussed. Although the theory is well-worked out, there is no implementation mentioned. Also, the examples in the paper are small ones, and there is no indication that larger examples have been studied.

Languages. The method does not use existing firewall languages, but as stated presents a new formalism for expressing operational behavior of firewalls.

There is no implementation language, since there is no implementation. The only language is the ambient calculus discussed previously.

User Interface. Not applicable.

Replicability. The algorithms given in the paper are given in complete formal detail, so a tool based directly on these algorithms could be implemented without any further information necessary.

Service, Accessibility, and Traffic. We consider how the flow logic approach answers the following questions:

- Which services are accessible on a given host? This question is answered by the flow logic analysis. In the ambient calculus this question corresponds to computing which processes are allowed to enter and/or execute on a particular host.
- Is a given host/network accessible from another given host/network? This question is also answered by the flow logic analysis.
- What kind of traffic is allowed between two networks? There is no discussion of different kinds of traffic. There is only discussion of whether an agent has the correct keys or not. Perhaps different kinds of traffic could be expressed by different kinds of programs in the ambient calculus.
- From which network is a given host accessible? This question is also answered by the flow logic analysis.

Rules that are Never Matched. Not applicable, since access is not defined by rules.

Spoofing. No.

Easily Extendible. Not clear. The language is powerful, and the extent of its power needs to be studied before understanding whether or not there would be useful extensions.

Multi-Vendor Firewalls. Not applicable. No vendors are discussed.

Tools available, in-house or off the shelf, or no tools. The paper develops algorithms in detail, but as stated above, does not describe any tool. Thus, there is probably no existing implementation. At least none are publicly available.

Citation Relationship to Other Papers. None of the other papers currently under study are cited by this paper, probably because the techniques are quite different. It is likely that none of the other papers cite this one because it is quite recent.

6.3 Discussion

Most of the other papers in this study start with existing firewall languages for expressing rules in actual systems, such as the one used in CISCO routers. This paper does not, and so the question of how to relate this work to practical firewall tools has not been answered, but is of considerable interest. One approach might be to find a methodology for expressing firewall configuration rules such as those used in CISCO routers as firewall "programs" in the ambient calculus. This would equate a particular list of filter rules for a router with a particular program in the ambient calculus in such a way as they express the same information. One notable difference is that there is no notion of passwords or keys in CISCO rules, but perhaps it is possible to relate the notion of a packet accepted by a particular router expressed as a filter rule in a practical firewall language to a process that has the correct keys to allow it to cross the firewall in the ambient calculus formalism. Similarly, not having the correct key would correspond to being denied access by a filter rule. Answering these questions requires further study.

6.4 A Detailed Example of a Firewall Model

Using the ambient calculus presented in the paper, the following program models a firewall and an agent which has permission to cross the firewall. This example is used throughout the paper and comes directly from Cardelli and Gordon [CG00].

$$\begin{aligned} \mathit{Firewall}: &\quad w[k[\mathit{out}\ w.\mathit{in}\ k'.\mathit{in}\ w] \mid \mathit{open}\ k'.\mathit{open}\ k''.P] \\ \mathit{Agent}: &\quad k'[\mathit{open}\ k.k''[Q]] \end{aligned}$$

In the notation $w[\ldots]$, w is the name of a process or location, and the computation either carried out by the process or taking place at the location is written inside the brackets. $P.Q$ denotes sequential computation of P followed by Q, and $P \mid Q$ denotes parallel computation of P and Q.

We explain the above program as we show its execution, but first we must put these two together in one program, as follows: $n[(\upsilon w)(\mathit{Firewall} \mid \mathit{Agent})]$. Here, the υ symbol is used to mark local scope, and w is the name of the firewall. Thus both the *Firewall* and *Agent* subprograms are in the scope of w. The outer n can be viewed as the name of the whole system. The execution of this program (taken directly from the paper with minor notational simplification) can be written as follows.

$$\begin{aligned} & n[(\upsilon w)(w[\ \ k[\underline{\mathit{out}\ w}.\mathit{in}\ k'.\mathit{in}\ w] \mid \mathit{open}\ k'.\mathit{open}\ k''.P\ \ ] \mid k'[\mathit{open}\ k.k''[Q]])] && (1) \\ \to\ & n[(\upsilon w)(k[\underline{\mathit{in}\ k'}.\mathit{in}\ w] \mid w[\mathit{open}\ k'.\mathit{open}\ k''.P] \mid k'[\mathit{open}\ k.k''[Q]])] && (2) \end{aligned}$$

$$\rightarrow \quad n[(\upsilon w)(w[open\ k'.open\ k''.P] \ \mid \ k'[k[in\ w] \mid \underline{open\ k}.k''[Q]])] \tag{3}$$

$$\rightarrow \quad n[(\upsilon w)(w[open\ k'.open\ k''.P] \ \mid \ k'[\underline{in\ w} \mid k''[Q]])] \tag{4}$$

$$\rightarrow \quad n[(\upsilon w)(w[\underline{open\ k'}.open\ k''.P \ \mid \ k'[k''[Q]]])] \tag{5}$$

$$\rightarrow \quad n[(\upsilon w)(w[\underline{open\ k''}.P \ \mid \ k''[Q]])] \tag{6}$$

$$\rightarrow \quad n[(\upsilon w)(w[P \mid Q])] \tag{7}$$

Here, each line represents one step of the execution, and the command that is executed at each step is underlined. The command ($out\ w$) in line (1) sends the "pilot" process named k outside the firewall w. Its purpose is to guide any agent knowing the correct passwords inside the firewall. The result is line (2), where we now have three processes operating in parallel: the pilot process, the process contained within the firewall, and finally the agent. Note the ($open\ k$) command inside the agent. The agent must know the name of the pilot process in order to gain access. At this point the "name" of the agent process k' matches the key in the ($in\ k'$) command of the pilot, which means that the agent has the correct key to let the pilot process in to become part of the agent process (which allows the pilot to eventually bring the agent directly inside the firewall). The result is line (3), where the remaining part of the pilot process $k[in\ w]$ is running in parallel with the main code of the agent (starting with the *open* command). The purpose of this underlined *open* command is to "open" up the pilot process, so that it is no longer a separate process inside the agent, but instead its code ($in\ w$) is running in parallel directly with the remaining code of the agent $k''[Q]$ as shown on line (4). The ($in\ w$) code on this line now brings the agent/pilot code inside the firewall, resulting in line (5). Inside the firewall, the agent process k' can now be opened via the *open* statement on line (5). The third password k'' is used to isolate the process Q, which is the agent's code, and keep it from interfering with the firewall protocol P. The *open* statement on line (6) brings these two pieces of code together so that on line (7), they are running in parallel, which means the agent has been fully admitted and allowed to execute its code inside the firewall.

7 Conclusion

We have included criteria in each of the preceding four sections which allow us to classify and compare the techniques in the four papers that we have evaluated. Tables 1, 2, and 3 provide a comparison in the form of an overview of these criteria and a description of how they are fulfilled by each of the methods.

In summary, the techniques in the papers by Hazelhurst and Heronen-Zitting provide practical tools which have been implemented and can be replicated. We have already built a prototype of the latter which has allowed us to begin experimenting with the kinds of analyses that can be performed by the tool (described in Section 4). It would not be difficult to do the same for the Hazelhurst paper, i.e., implement a prototype tool quickly and experiment with its capabilities for analyzing firewalls. Since these tools already provide useful analysis capabilities, this would be a beneficial next step. The first technique can answer a wider variety of questions, but the second has the potential to be more efficient.

The Guttman and Nielson et. al. papers provide more general and powerful techniques. The algorithms used are fully described, and so it would be possible to implement them directly. The implementation effort, however, would be greater for these techniques than for the other two because the algorithms and tasks they perform are more complex. Since there are existing tools for Guttman's techniques, it would be better to obtain a copy directly from Guttman and experiment with it. In both the Guttman and Nielson techniques, it is possible to prove formally that firewalls are correct. In the case of Guttman's work, the techniques are especially promising because they provide methods for automatically generating practical firewall rules from high-level policies. The connection to firewall rules in the Nielson method is less clear. Because of the work required to make this connection, using this method in a practical tool is for the longer term, and would involve building a tool from scratch. But this approach is promising. As discussed earlier, it should be possible to implement firewall rules as programs in the ambient calculus. Once this connection is made, the techniques in the paper would provide powerful tools for analyzing these firewall rules.

Paper	Algorithms for Analyzing Firewalls and Router Access Lists	An Expert System for Analyzing Firewall Rules	Security Goals: Packet Trajectories and Strand Spaces	Validating Firewalls Using Flow Logics
Author	Hazelhurst	Eronen and Zitting	Guttmann	Nielson et. al.
Researcher	Logrippo	Stepien	Adi	Felty
Type of Configuration Problem Addressed	Simple expert system, answers questions on types of packets allowed or excluded by firewall rules	Network, configuration, and access-list properties, and defining new predicates for higher level queries	Computes and verifies local enforcements from a network topology and a global network access control policy	Verifies that authorized agents can enter and unauthorized agents cannot
Network Topology	No	Yes	Yes	Yes
Firewall Model or Architecture Addressed	Uses the syntax of CISCO firewall architecture, but is more general	Cisco access lists	Models networks by graphs and filtering behaviors by rule-based logic	General formalism allowing a variety of models and architectures
Mathematical Basis or Theory	Binary trees efficiently represented as BDDs	Expert System	Logic, graph theory	Process Calculus
Completeness of Network Model	Complete for if-then-else format, cannot express state	Complete	Cannot express requirements on packets traveling within a single area	Very general
Granularity of Network Model	Fine granularity	Fine granularity	Fine granularity	High-level (course-grained)
Case Study Results	Yes	Yes	Yes	Yes

Table 1: Overview of Methods I

Paper	Algorithms for Analyzing Firewalls and Router Access Lists	An Expert System for Analyzing Firewall Rules	Security Goals: Packet Trajectories and Strand Spaces	Validating Firewalls Using Flow Logics
Author	Hazelhurst	Eronen and Zitting	Guttmann	Nielson et. al.
Researcher	Logrippo	Stepien	Adi	Felty
Admitted Limitations	Some more complex rules can't be represented	None	Generated posture output is unnecessarily complex	Not discussed
Perceived Limitations	Nothing additional	None	No mention of how to protect the filtering routers	No implementation, only small examples considered
Languages	Cisco firewall language expressed as BDD	Cisco firewall language and Constraint Logic Programming (CLP)	Scheme, Ocaml	No existing language, presents a new formalism
User Interface	Yes (Tcl/Tk)	Yes	No	No
Replicable	Yes	Yes	Yes	Yes
Which services are accessible on a given host?	Yes	Yes	Yes	Yes
Is a given host/network accessible from another given host/network?	Yes	Yes	Yes	Yes
What kind of traffic is allowed between two networks?	Yes	Yes	Yes	No
From which network is a given host accessible?	Yes	Yes	Yes	Yes

Table 2: Overview of Methods II

Paper	Algorithms for Analyzing Firewalls and Router Access Lists	An Expert System for Analyzing Firewall Rules	Security Goals: Packet Trajectories and Strand Spaces	Validating Firewalls Using Flow Logics
Author	Hazelhurst	Eronen and Zitting	Guttmann	Nielson et. al.
Researcher	Logrippo	Stepien	Adi	Felty
Rules that are never matched	Yes	Yes	Yes	N/A
Detects spoofing	No	Yes	Yes	No
Easily Extend-able	No	Yes, by defining new predicates as combinations of existing predicates	Yes	Unknown
Multi-Vendor Firewalls	No	No	No	N/A
Tools	based on Voss	implemented in Eclipse	NPT, Atom-izer	None de-scribed
Availability of Tools	Voss freely available, but tool using Voss is not	Eclipse and tool are not available, but other CLP languages are freely available	Available for license from MITRE Corp.	No

Table 3: Overview of Methods III

References

[CG00] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240:177–213, 2000.

[EZ01] Pasi Eronen and Jukka Zitting. An expert system for analyzing firewall rules. In *6th Nordic Workshop on Secure IT Systems*, pages 100–107, 2001.

[Gut01] Joshua D. Guttman. Security goals: Packet trajectories and strand spaces. *Lecture Notes in Computer Science*, 2171:197–263, 2001.

[Haz99] Scott Hazelhurst. Algorithms for analyzing firewall and router access lists. Technical Report TR-Wits-CS-1999-5, University of Witwatersrand, South Africa, July 1999.

[NNH02] Flemming Nielson, Hanne Riis Nielson, and René Rydhof Hansen. Validating firewalls using flow logics. *Theoretical Computer Science*, 283:381–418, 2002.