AFPL2, An Abstract Language for Firewall ACLs with NAT support

S. Pozo, A.J. Varela-Vaca, R. M. Gasca
Department of Computer Languages and Systems
ETS Ingenieria Informatica, University of Seville
41012 Seville, Spain
{sergiopozo, ajvarela, gasca}@us.es
http://www.lsi.us.es/~quivir

Abstract— The design and management of firewall ACLs is a very hard and error-prone task. Part of this complexity comes from the fact that each firewall platform has its own low-level language with a different functionality, syntax, and development environment. Although high-level languages have been proposed to model firewall ACLs, none of them has been widely adopted by the industry due to a combination of factors: high complexity, no support of important features of firewalls, etc. In this paper the most important access control policy languages are reviewed, with special focus on the development of firewall ACLs. Based on this analysis, a new domain specific language for firewall ACLs (AFPL2) is proposed, supporting more features that other languages do not cover (e.g. NAT). As the result of our design methodology, AFPL2 is very lightweight and easy to use. AFPL2 can be translated to existing low-level firewall languages, or be directly interpreted by firewall platforms, and is an extension to a previously developed language.

Keywords: firewall, acl, language, model, nat.

I. INTRODUCTION

A firewall is a network element that controls the traversal of packets across different network segments. It is a mechanism to enforce an Access Control Policy, represented as an Access Control List (ACL) (or *rule set)*. Firewalls use obligation policies (also known as Event Condition Action Rules (ECA) that must perform certain actions when certain events occur. By contrast, authorisation policies permit or deny actions based upon the action, the source of the action and the target of the action. Thus, a Firewall ACL is in general a list of linearly ordered (total order) condition/action rules. Let *RS* be a firewall rule set consisting of *l* rules, $RS = \{R_0, ...R_l\}$. Consider $R =< H, Action >, H \in \mathbb{N}^5$ as a rule, where $Action = \{allow, deny\}$ is its action. A selector of a firewall rule R_j is defined as $R_j[k], 0 \leq j \leq l, k \in \{protocol, srcIP, srcPrt, dstIP, dstPrt\}$, where each selector can be represented as a natural number, and where *H* is the set of the five natural selectors. A rule R_j matches a packet *p* when the values of each field of the header of a packet *p[k]* are subsets or equal to the values of the rule selector, $R_j[k] \forall k$.

Two of the most important problems Firewalls have to face are the high complexity of ACL design [1] and ACL consistency diagnosis [3]. Writing and managing ACLs are tedious, time-consuming and error-prone tasks for a wide range of reasons [4]. Low-level firewall languages are, in general, hard to learn, use and understand. In addition, each firewall platform has its own low-level language which usually is very different from other vendors' ones. Changing from one firewall platform to another often means a complete rewrite of the ACL. In this translation process, inconsistencies and redundancies can be introduced [3]. Figs. 1 and 2 present two fragments of ACLs written for Netfilter IPTables and Cisco PIX platforms respectively to give an idea of the complexity and differences of these languages. Note that the number of rules of a firewall ACL ranges between a few ones and 5000 [5].

Many third-party domain specific languages (DSLs) have been proposed to abstract the network administrator from the underlying firewall platform details and language syntax [6, 7, 8, 9, 10, 11]. A domain specific language provides more possibilities to network administrators, since it can raise the abstraction level of the problem domain. However, these DSL proposals have different problems regarding different aspects of design and deployment of ACLs. The result is that they have not been adopted by the industry.

We think that there is a clear need of a DSL for Firewalls supporting the main features of the existing low-level firewall-specific languages, but with significantly less complexity than currently proposed DSLs. This language must have the possibility of automatic compilation to the market-leader low-level firewall languages, and be easily extensible to support new features and firewall platforms. Recently, we have proposed a new high level firewall DSL with all these features, called Abstract Firewall Policy Language (AFPL) [1].

In this paper we propose an extension of AFPL, AFPL2, with Network Address Translation (NAT, [2]) support, a must-have feature of modern firewall languages. To the best of our knowledge, AFPL2 is the first abstract firewall language to support NAT. AFPL2 also supports the vast majority of functionality of the market-leader firewall platforms, as it is an extension to AFPL (more details are given in the related works section).

The use of AFPL2 implies a higher flexibility in the design of ACLs, since it is not tied to any particular firewall platform. Any change of vendor does not necessarily imply a change of the designed ACL model, which adds homogeneity to the design and management of firewall ACLs in heterogeneous multi-firewall environments.

```
-A FORWARD -i   -s 192.168.1.0  -d 170.0.1.10  -p tcp  -m tcp --sport any  --dport 21  -p tcp  -j ACCEPT
-A FORWARD -i   -p tcp  -p tcp  -j DROP
-A FORWARD -i   -s 192.168.1.0  -d 170.0.1.10  -p udp  -m udp --dport 53  -p udp  -j ACCEPT
-A FORWARD -i   -d 170.0.1.10  -p udp  -m udp --dport 53  -p udp  -j ACCEPT
-A FORWARD -i   -s 192.168.2.0  -d 170.0.2.0  -p udp  -p udp  -j ACCEPT
-A FORWARD -i   -p udp  -p udp  -j DROP
```

Figure 1. Netfilter IPTables ACL fragment

```
access-list acl-out permit gre host 192.168.201.25  host 192.168.201.5
access-list acl-out permit tcp host 192.168.201.25  host 192.168.201.5  eq 1723
static (inside,outside) 192.168.201.5  10.48.66.106 netmask 255.255.255.255 0 0
access-group acl-out in interface outside
access-list acl-out permit udp host 192.168.201.25  host 192.168.201.5  eq 1701
static (inside,outside) 192.168.201.5  10.48.66.106 netmask 255.255.255.255 0 0
access-group acl-out in interface outside
```

Figure 2. Cisco PIX ACL fragment

The structure of this paper is as follows: in section 2 related works are described. In section 3 AFPL2 model and the design process is described. Finally, we conclude in section 5 and propose a research direction for our future works.

II. Related Work

In [6] the authors proposed a high-level language, Firmato, which models ACLs as ERDs in order to automatically generate low-level firewall ACLs. However, the complexity of Firmato is similar to that of many low-level languages. Two major limitations of Firmato are that (1) it does not support NAT and (2) can only represent knowledge in positive logic (*allow* rules), which complicates the expression of exceptions (a rule with a general *allow* action, immediately preceded by a more restrictive rule with a *deny* action). This could result in the need to write a lot of rules to express them. However, as a lateral effect, rules are always consistent and order-independent. FLIP [15] is a recently proposed firewall language which can also be compiled into several low-level ones, although in the paper no more information about his feature is provided. One of its main limitations is that it does not support overlappings between rule selectors, and thus it is impossible to express exceptions. In addition, its syntax is even more complex than Firmato's one. However, due to this lack of expressiveness, FLIP ACLs are order independent. In [7] the authors provide a general language, Ponder, to represent network policies (in general), which cannot compile to any low-level language. A re-engineered version, Ponder2, is also available. However, the complexity of Ponder surpasses the needs of firewall ACLs. In theory, a language that can express any network policy could express a firewall access control policy. However, concepts such as NAT cannot be expressed with Ponder and all the other reviewed general policy languages. AFPL [1] is a language developed after an analysis of the features of major firewall languages, supporting most of their functionality at a fraction of their complexities. It can express stateful and stateless rules (although an administrator does not need to know these kind of details, since complexity is hidden in the language), positive and negative rules, overlappings, exceptions, and can be compiled to six market-leader firewall languages (details are available in [1]).

Some organizations have even proposed languages to represent access control policies as XML documents, such as XACML [8], PCIM [9], Rule-ML [10], and SRML [11]. However, none of these languages is specific enough for firewall access control policies, resulting in a high complexity to express firewall concepts, or in an impossibility to express them at all (this is the case of NAT for all these languages). Even UML has been proposed to model access control policies [13, 14]. However, in our problem domain, UML could be an aid for the requirements definition stage, but then these models need to be translated into a DSL. These models and languages are very generic

TABLE I. SURVEY OF ACCESS CONTROL LANGUAGES FEATURES

	Firmato	Ponder2	FLIP	SRML	Rule-ML	PCIM	XACML	AFPL
FW-Specific	*x*	*x*	*√*	*x*	*x*	*x*	*x*	*√*
Support stateful rules	*√*	*N/A*	*√*	*N/A*	*N/A*	*N/A*	*N/A*	*√*
Support stateless rules	*√*	*N/A*	*√*	*N/A*	*N/A*	*N/A*	*N/A*	*√*
Support NAT	*x*	*N/A*	*x*	*N/A*	*N/A*	*N/A*	*N/A*	*x*
Positive logic	*√*	*√*	*√*	*√*	*√*	*√*	*√*	*√*
Negative logic	*Partial*	*√*	*√*	*√*	*√*	*√*	*√*	*√*
Selector-values overlap	*√*	*√*	*x*	*√*	*√*	*√*	*√*	*√*
User-controlled rule order	*x*	*√*	*N/A*	*x*	*x*	*√*	*√*	*√*
Topology/logic separation	*√*	*N/A*	*√*	*N/A*	*N/A*	*N/A*	*√*	*√*
Relative complexity	*High*	*High*	*Low*	*Low*	*Low*	*Medium*	*Low*	*Low*
Compilation to low-level	*√*	*x*	*√*	*x*	*x*	*x*	*x*	*√*
Low-level lang. import	*x*	*x*	*x*	*x*	*x*	*x*	*x*	*Partial*

and are not intended for the area of any particular access control problem. Table 1 presents a survey of the most important features of the reviewed languages (related to their ability to express firewall ACL knowledge).

There are good surveys of access control policy languages available [12] which explain their design details.

III. AFPL2. AFPL EXTENDED WITH NAT

NAT is a must-have feature of modern low-level firewall languages (it was defined in the year 1999), and nowadays all modern firewall platforms support it. The main idea behind NAT is to change *(translate)* the values of some headers of TCP/IP packets in different situations. These changes are specified using rules *(translation rules)* in a similar way as filtering rules are specified. There are mainly two modes of NAT (as defined in RFC2663 [2]).

- **Source NAT (SNAT).** Also know as Outbound NAT, Network Address Port Translation (NAPT), or Masquerading, in which the source of a packet is translated when it traverses an outbound interface of a firewall. Response packets are translated back to their real address.
- **Destination NAT (DNAT).** Also known as Packet Forwarding, in which the destination of a packet is translated when it traverses an inbound interface of a firewall. Response packets are translated back to their real address.

However, although NAT has been defined in an RFC, it has not been standardized. For this reason, an analysis of the NAT features supported by the market-leader firewall platforms is needed for the design of AFPL2 in order to satisfy the vast majority of administrators.

The considered firewall platforms in the NAT analysis are the same ones as for AFPL: IPTables 1.4.2, Cisco PIX 8, FreeBSD 8 IPFilter, FreeBSD 8 IPFirewall, OpenBSD 4.1 Packet Filter, and Checkpoint Firewall-1 4.1. The analysis is presented in Appendix I and shows the supported NAT modes, its filtering selectors, and available syntaxes.

A. Modelling Considerations

Firewall platforms are very different from one vendor to another, and even among the available Open Source platforms. These range from differences in the number, type, and syntax of selectors that each platform's filtering algorithm can handle, to huge differences in rule-processing algorithms that can affect the design of the ACL. Fortunately, the vast majority of features can be expressed with any of the filtering languages and platforms, with the only difference on the number of rules needed, and/or in their syntax.

With the focus on modelling firewall languages and platform functionality, some questions may arise. The first one is whether all the analyzed firewall platforms share a common set of filtering selectors (or a common set of functionality). Another one is, for the common set of selectors, if there is at least one common syntax among all firewall platforms (in order to be able to use the functionality); or if not, if the available syntaxes for each platform have equivalencies in the other ones (i.e. are emulable). For the design of AFPL2 we use the same methodology as for AFPL [1]: first a DSL with a set of selectors (or features) and syntaxes supported by all the analyzed firewall platforms and languages is created. Then, the non common selectors and syntaxes are analyzed and only added to AFPL2 if they comply with a criterion that is going to be defined later. This methodology yielded in the case of AFPL to a lightweight language with a very simple syntax that would satisfy the vast majority of administrators. We expect the same for AFPL2. The methodology is described in the next sections with more detail.

However, for each analyzed firewall language and platform there is a set of features (selectors) that is not going to be modelled in AFPL2. It remains a topic for future research how these features could be supported by AFPL2. This is an important topic, since if successfully accomplished, transformations from low-level languages to AFPL2 could be possible without loss of expressiveness.

B. AFPL2 Model

Our start point is the factorized AFPL2 model presented in Table 2. We take it as a basic NAT model. Note that NAT use filtering rules for matching packets and translation rules that define which selectors of the matched packets and how will be translated. In Table 2 only the part related with translation rules is presented, since filtering was covered in AFPL [1].

TABLE 2(A). AFPL2 SOURCE NAT (NAPT) FACTORIZED MODEL

Translated Selector	Obligation	Syntax	Comments
Source IP Address	Mandatory	-Host IP -Interface name	If the interface name is given, the interface IP is used (it could be dynamic link)

TABLE 2(B). AFPL2 DESTINATION NAT FACTORIZED MODEL

Translated Selector	Obligation	Dependencies	Syntax
Destination IP Address	Mandatory		-Host IP
Destination Port	Optional	Destination port must be specified in the original packet	-Number -Range: [p1,p2]

In the next sections, non-common functionality is going to be analyzed for its inclusion in AFPL2.

1) Addition of uncommon NAT modes

Although there are a lot of ways of expressing translations, only two kinds of translation rules are supported in AFPL2 (Table 2). However, the analyzed firewall languages support more NAT modes (analyzed in Appendix I). There, we show that all these modes are in reality variations of the two basic NAT modes defined in RFC2663 (Source and Destination NAT) and can be reproduced in one way or another using these two basic types. Although RFC2663 defines more NAT modes (like Twice NAT or Multi-homed NAT), they are not supported in any of the

analyzed firewall platforms. For these reasons, no more NAT modes are necessary in AFPL2.

2) Addition of uncommon selectors

An uncommon selector can be added if its functionality can be reproduced (emulated) with the selectors of the factorized model, and it also adds new functionality to the model. A selector *adds new functionality* to the model if it cannot be emulated with translation rules which do not contain it. The conclusion is that using this criterion, no more selectors can be added to AFPL2. An exhaustive list of selectors per firewall is presented in Appendix I.

3) Addition of uncommon syntaxes

In this section, we analyze the possibility of supporting uncommon syntaxes in the considered selectors. In general, we consider an uncommon syntax as a candidate for its addition to the collection of supported syntaxes for that selector in AFPL2, if it can be emulated with the common syntaxes of the same selector and it provides clear usability improvements for human users. A syntax *provides clear usability improvements* if its use by a human cannot introduce inconsistencies [3] in an ACL created with AFPL2 and it provides compactness. Again, we will base this analysis on results presented in Appendix I.

- **SNAT Source IP address.** The uncommon syntaxes of this selector are identifiers, block IPs, IP ranges and collections of IPs. Note that the use of identifiers provides a clear usability improvement and does not introduce inconsistencies in the ACL, and thus will be considered for AFPL2. All the other syntaxes provide ACL compactness, and also represent usability improvements. As they cannot cause ACL inconsistencies, they will also be considered (except IP ranges and IP collections, which are redundant). Block IPs, IP ranges, and in general collections of IPs can be emulated in low-level languages that do not support them by decomposing these collections of IPs into several unique IPs, and defining one NAT rule for each.
- **DNAT Destination IP address.** The uncommon syntaxes of this selector are identifiers and IP ranges. Identifiers are included for the same reasons stated for SNAT source IPs. However, IP ranges cannot be included because it is only used for load balancing, a non-emulable feature not supported by all the analyzed platforms.
- **DNAT Destination port.** Many range syntaxes are possible in many firewall platforms, as is the case of ranges '<p', '<=p', '>p', '>=p', '(p1, p2)' and ')p1, p2('. These syntaxes provide no new functionality or a clear usability improvement, and can be easily emulated with the common '[p1, p2]' syntax without loss of functionality. For this reason they will not be included in AFPL2.

4) NAT Model

In order to match the original packet, the same selectors and rule format used for filtering in AFPL can be used without restrictions, since all firewall languages support them in at least one of their NAT modes. For translation selectors, selectors presented in Table 3 must be used, with the presented constraints about their syntax. This model represents the final AFPL2 language (NAT part).

AFPL2 now supports three kinds of rules: filtering (also present in AFPL), SNAT, and DNAT. SNAT and DNAT are not mandatory, but at least one filtering rule must be specified (for the default policy).

In AFPL2, by using SNAT rules it is possible to translate an outgoing packet, changing its source IP address. It is possible to specify the new IP address using host IP format, the name of the interface where the packet is going to go out to the destination network, an identifier representing an interface name or a host IP address, and a block of IP addresses in CIDR format. Moreover, it is possible to use an interface name whose IP address is assigned dynamically (such as in the case of PPP). This mode of SNAT is usually called masquerading.

By using DNAT rules, it is possible to translate an incoming packet, changing its destination IP address and/or

TABLE 3. FINAL AFPL2 MODEL (ONLY NAT PART)

SOURCE NAT

Translated Selector	Obligation	Dependencies	Common Syntax (Can be optimized)	Uncommon Syntax (Must be emulated)	Comments
Source IP Address	Mandatory		-Host IP -Interface name	-Identifier -Block	If the interface name is given, the interface IP is used (it could be dynamic link)

DESTINATION NAT

Translated Selector	Obligation	Dependencies	Common Syntax (Can be optimized)	Uncommon Syntax (Must be emulated)	Comments
Destination IP Address	Mandatory		-Host IP	-Identifier	
Destination Port	Optional	Destination port must be specified in the original packet	-Number -Range: [p1,p2]	- Identifier	

its destination port. It is possible to specify the new destination address by using a host IP address, or an identifier representing it. Destination port translation is optional, and subject to its specification in the original packet. It can be specified using a port number, a port range, or an identifier representing it.

AFPL2 compilation to low level languages is an easy step, because AFPL2 features are directly supported by the analyzed low level languages or are emulable by them using more than one rule. Thus, in the majority of cases, compilation will be a syntax transformation.

5) AFPL2 grammar

Several different grammars can be constructed for AFPL2. However, we have preferred to express the proof of concept grammar using Relax NG Compact. Grammar is presented in Appendix II, and is also available [16].

IV. Conclusion and Future Work

In this paper we have proposed a new abstract language to represent firewall ACLs with NAT, AFPL2. To the best of our knowledge, AFPL2 is the first abstract firewall language that supports NAT.

Contrarily to other approaches which try to design a general language for access control or network policies, we have focused in the design of a domain specific language for Firewall ACLs, departing from an analysis of the features of the market-leader firewall languages. This approach resulted in a simple and lightweight language that is able to model the vast majority of features present in any of the analyzed firewall platforms and languages.

However, there are features of some firewall platforms that are not supported in AFPL2. More research is needed in this direction in order to get solutions and be able to integrate them as lower-level extensions of AFPL2.

Acknowledgment

This work has been partially funded by Spanish *Ministry of Science and Education project* under grant DPI2006-15476-C02-01, and by FEDER (under ERDF Program).

References

[1] S. Pozo, R. Ceballos, R.M. Gasca. Model Based Development of Firewall Rule Sets: Diagnosing Model Faults. Information and Software Technology Journal. No. 51, Issue 5, pp.894-915. Elsevier, 2009.

[2] P. Srisuresh, M. Holdrege. RFC 2663: IP Network Address Translator (NAT) Terminology and Considerations. IETF, August 1999.

[3] S. Pozo, R. Ceballos, R.M. Gasca, A Heuristic Polynomial Algorithm for Local Inconsistecy Diagnosis in Firewall Rule Sets, International Conference on Security and Cryptography (SECRYPT), Porto, Portugal, 2008.

[4] A. Wool, A quantitative study of firewall configuration errors, IEEE Computer 37(6) (2004) 62-67.

[5] David E. Taylor, Survey and taxonomy of packet classification techniques, ACM Computing Surveys 37(3) (2005) 238 – 275.

[6] Y. Bartal, A. Mayer, K. Nissim, A. Wool, Firmato: A Novel Firewall Management Toolkit, ACM Transactions on Computer Systems 22(4) (2004) 381-420.

[7] N. Damianou, N. Dulay, E. Lupu, M Sloman, The Ponder Specification Language, Workshop on Policies for Distributed Systems and Networks (POLICY), HP Labs Bristol, UK, 2001, pp. 29-31.

[8] OASIS eXtensible Access Control Markup Language (XACML), http://www.oasis-open.org/committees/xacml/

[9] B. Moore, E. Ellesson, J. Strassner, A. Westerinen, Policy Core Information Model (PCIM), IETF RFC 3060, 2001.

[10] Rule Markup Language (RuleML), http://www.ruleml.org/

[11] Simple Rule Markup Language (SRML): A General XML Rule Representation for Forward-chaining Rules, ILOG S.A, 2001.

[12] De Capitani di Vimercati, S. Foresti, S. Jajodia, P. Samarati, Access control policies and languages, Int. J. Computational Science and Engineering 3(2) (2007).

[13] J. Jürjens, UMLsec: Extending UML for secure systems development, 5th International UML, Dresden, Germany. Springer-Verlag LNCS 2460, 2002, pp.1-9.

[14] D. Basin, J. Dorser, T. Lodderstedt, Model Driven Security: from UML Models to Access Control Infrastructures, ACM Transactions on Software Engineering and Methodology 15(1) (2006) 39-91.

[15] B. Zhang, E. Al-Shaer, R. Jagadeesan, J. Riely, C. Pitcher, Specifications of a High-level Conflict-free Firewall Policy Language for Multi-domain Networks, ACM Symposium on Access Control Models and Technologies (SACMAT), Sophia Antipolis, France, 2007, pp.185-194.

[16] AFPL2 grammar. http://www.lsi.us.es/~quivir/index.php/Main/ContribSergio

Appendix I

This analysis is related to the translation rules of market-leader firewall languages, and refers to the conditions that may be matched against a packet that arrives at the firewall, and how it must be translated. Each firewall language has a different syntax for each selector. It is important to note that syntaxes for NAT selectors are the same as the ones described for filtering selectors for each firewall platform in an earlier work [1], when nothing different is noted. √ indicates that a selector is supported, and *x* that is not supported.

A. Netfilter IPTables 1.4.2

IPTables supports the two basic translation modes as defined in RFC2663.

- **SNAT.** Translates the source of a packet. Applies to all interfaces in outbound direction if no one is specified in the rule. It only makes sense if the rule is applied over an outbound interface. Although all parameters are optional, someone must be specified in order to accomplish a NAT function. A special type of SNAT is masquerading (when the outbound interface has a dynamic connection, like PPP).
- **DNAT.** Translates the destination of a packet. Applies to all interfaces in inbound direction if no one is specified in the rule. It only makes sense if the rule is applied over an inbound interface. Although all parameters are optional, someone must be specified in order to accomplish a NAT function. Special types of DNAT are port forwarding and load balancing (usually connections from Internet to the firewall), and transparent proxy (connections originating in internal segments).

TABLE 4. NETFILTER IPTABLES ANALYSIS

NAT Type	SNAT / MASQ	DNAT / PORT FW / LOAD BA / TR PROXY
Src IP Address	*Opt*	*Opt*
Translated Src IP Address	√* *-IP*	*x*
Src Port	*Opt*	*Opt*
Translated Src Port	*Opt*** *- Number, range*	*x*
Dst IP Address	*Opt*	*Opt*
Translated Dst IP Address	*x*	√* *-IP*
Dst Port	*Opt*	*Opt*
Translated Dst Port	*x*	*Opt* *- Number, range*
Protocol	*Opt*	*Opt*
Interface	*Opt (Outgoing)*	*Opt (Incoming)*
Comments	**It is not possible to do load balancing in K >=2.6.11* ** If port is given, NAPT is done instead of SNAT	**It is not possile to do load balancing in K >=2.6.11*

B. *Cisco ASDM/PIX 8*

Cisco PIX supports different NAT modes. However, these are nothing more of different ways to express the two basic NAT modes, coped with a lot of low-level details regarding ASDM platform.

- **Dynamic (DNAT, DPAT, Policy DNAT, Policy DPAT).** It uses a pool for translated addresses. Different connections can use different IPs in the pool (DNAT). DPAT only uses one IP for translation (instead of a pool) and translates ports. Policy DNAT and DPAT are the same as DNAT and DPAT with the exception that destination addresses and ports can also be used for matching NAT rules (however, destination selectors cannot be translated). Only connections from the inside are allowed. Connections cannot be initiated from the outside, since there is no *1..1* rule for NAT. This approach is used for masquerading.
- **Static (SNAT, SPAT, Policy SNAT, Policy SPAT).** This approach does not use a pool for translated addresses. This approach uses *1..1* translations. The rest is equal to dynamic approach. However, in this case, connections can be initiated from the outside as a lateral effect, since *1..1* NAT rules exist (access must be granted via *acl* command). This approach is the only one that can be used for connections coming from an insecure zone to a secure one (behind the firewall), although it cannot explicitly translate destination selectors (outgoing rules are established, and also used for incoming connections for translating destination selectors).
- **NAT control.** If this NAT mode is activated, it imposes that less secure zones must also use NAT to access more secure zones. However, translation can be bypassed using three different approaches: identity NAT, static identity NAT, and NAT excemption.

C. *OpenBSD IPFilter 8*

IPFilter supports the two basic translation modes as defined in RFC2663.

TABLE 5. CISCO PIX ANALYSIS

NAT Type	Dynamic NAT and PAT	Dynamic Policy NAT and PAT	Static NAT and PAT	Policy Static NAT and PAT
Address Pool	√	√	*x*	*x*
Src IP Address	√ Collection using the pool	√ Collection using the pool	√	√
Translated Src IP Address	√ -IP, Block -Range, Collection	√ -IP	√ -IP, Interface IP	√ -IP, Interface IP
Src Port	*x*	*Opt*	*x*	*Opt*
Translated Src Port	*Automatic (only for PAT)*	*Automatic (only for PAT)*	*Opt (only for PAT)*	*Opt (only for PAT)*
Dst IP Address	*x*	√ (one or more)	*x*	√ (one or more)
Translated Dst IP Address	*x*	*x*	√/*x* *(bidi)*	√/*x* *(bidi)*
Dst Port	*x*	*Opt*	*x*	*Opt*
Translated Dst Port	*x*	*x*	*Opt (only for PAT)*	*Opt (only for PAT)*
Protocol	*x*	√	TCP or UDP. Only for PAT	TCP or UDP. Only for PAT
Interface (outbound)	√ Only in less secure to more secure interface connections	√ Only in less secure to more secure interface connections	*x*	*x*
Interface (inbound)	*x*	*x*	√	√
Connection Settings	Misc options for TCP and UDP	Misc options for TCP and UDP	Misc options for TCP and UDP	Misc options for TCP and UDP

TABLE 6. IPFILTER ANALYSIS

NAT Type	Basic NAT	Port Redirection
Src IP Address	√	*Opt*
Translated Src IP Address	√ *[Only IP or Block]*	*x*
Src Port	*Opt*	*Opt*
Translated Src Port	*Opt**	*x*
Dst IP Address	*Opt*	√* *[Only IP]*
Translated Dst IP Address	*x*	√*(host only)*
Dst Port	*Opt*	*Opt*
Translated Dst Port	*x*	√
Src/Dst Portocol	*Opt*	*Opt*
Interface	√	√
Comments	* If port is given, NAPT is done instead of SNAT	* If multiple dst IPs are specified, round-robbin load-balancing is accomplished

- **Basic NAT.** Translates the source of a packet. Applies to the selected interface in outbound direction. This type of NAT can be used for masquerading (including when the outbound interface has a dynamic connection, like PPP) and for application proxying. Map-block is a variation of basic NAT, used to set up static (source) IP address translation, based on an algorithm to squeeze the addresses to be translated into the destination range. The same effect can be accomplished with basic NAT using source port pool (and not source IP translation).
- **Port redirection.** Translates the destination of a packet. Applies to the selected interface in inbound direction. This type of NAT can be used for load balancing, transparent proxy, and simple port-forwarding.

D. FreeBSD IPFirewall 8

IPFirewall also supports the two basic translation modes as defined in RFC2663, and even with the same naming.

- **NAT for outgoing connections.** Translates the source of a packet. Applies to the selected interface in outbound direction. This type of NAT can be used for masquerading (including when the outbound interface has a dynamic connection, like PPP) and for application proxying.
- **NAT for incoming connections.** Translates the destination of a packet. Applies to the selected interface in inbound direction. This type of NAT can be used for load balancing, transparent proxy, and simple port-forwarding.

TABLE 7. IPFIREWALL ANALYSIS

NAT Type	Outgoing connections	Incoming connections
Src IP Address	*Opt*	*Opt*
Translated Src IP Address	√	*x*
Src Port	*Opt*	*Opt*
Translated Src Port	*x*	*x*
Dst IP Address	*Opt*	*Opt*
Translated Dst IP Address	*x*	√***
Dst Port	*Opt*	*Opt (Port or range)*
Translated Dst Port	*Opt**	*Opt**
Src/Dst Protocol	*Opt***	*Opt***
Interface	√	√
Comments	* Mandatory if Dst Port is specified ** Mandatory if ports are specified	* Mandatory if Dst Port is specified, but could be the same by default ** Mandatory if ports are specified *** If multiple translated dst IPs are specified, load-balancing is accomplished

E. OpenBSD Packet Filter 4.1

Packet Filter also supports the two basic translation modes as defined in RFC2663, and even with the same naming.

TABLE 8. PACKET FILTER ANALYSIS

NAT Type	NAT	Port Forwarding
Src IP Address	√	√
Translated Src IP Address	√ *- IP Collection*	*x*
Src Port	*Opt*	*Opt*
Translated Src Port	√	*x*
Dst IP Address	√	√
Translated Dst IP Address	*x*	√*
Dst Port	*Opt*	*Opt*
Translated Dst Port	*x*	*Opt*
Src/Dst Protocol	*x*	√
Interface	√	√
Comments	Bidirectional connections are possible	* If multiple translated dst IPs are specified, load-balancing is

- **NAT.** Translates the source of a packet. Applies to the selected interface in outbound direction. This type of NAT can be used for masquerading (including when the outbound interface has a dynamic connection, like PPP) and for application proxying. Bidirectional NAT is possible with only one rule. NAT exceptions are also possible.

- **Port Forwarding.** Translates the destination of a packet. Applies to the selected interface in inbound direction. This type of NAT can be used for transparent proxy, and simple port-forwarding.

F. Checkpoint FW-1 4.1

TABLE 9. FW-1 ANALYSIS

NAT Type	All types
Src IP Address	*Opt*
Translated Src IP Address	*Opt*
Src Port	*Opt*
Translated Src Port	*Opt*
Dst IP Address	*Opt*
Translated Dst IP Address	*Opt*
Dst Port	*Opt*
Translated Dst Port	*Opt*
Src/Dst Protocol	*Opt*
Interface	*x*

FW-1 also supports the two basic translation modes as defined in RFC2663. However, since FW-1 user interface is graphical, the administrator has the freedom to choose translations in all selectors. The denomination of SNAT or DNAT (static or hide) is only internal.

- **Source NAT.** Translates the source of a packet. Applies to all interfaces in outbound direction. This type of NAT can be used for masquerading (including when the outbound interface has a dynamic connection, like PPP) and for application proxying. There are two types of SNAT: static NAT (for *1:1* translations) and hide NAT (for *n:1* translations).
- **Destination NAT.** Translates the destination of a packet. Applies to all interfaces in inbound direction. This type of NAT can be used for transparent proxy, and simple port-forwarding. This can only be done in the static NAT mode. However, port forwarding is a special type of static NAT, where * for Internet is used as it was a unique IP.

APPENDIX II. AFPL2 GRAMMAR

```
#AFPL2 Schema. RelaxNG Compact
#Each rule may be a Filter Rule or Nat Rule

grammar {
  start =
    element policy {
      element rule {
        element filter { filterType },
        element comment { text }?
      }+,
      element srcnatrule {
        element srcnat { srcNatType },
        element comment { text }?
      }*,
      element dstnatrule {
        element dstnat { dstNatType },
        element comment { text }?
      }*

    }

  filterType =
    element interface { text }?,
    element direction { string "in" | string "out" | string "both" }?,
    element matches { matchesType },
    element action { string "allow" | string "deny" | string "reject" }

  srcNatType =
    element interface { text }?,
    element original { natOrigPacket },
    element translatedSrc { TnatPacketSrc }

  dstNatType =
    element interface { text }?,
    element original { natOrigPacket },
    element translatedDst { TnatPacketDst }
```

```
  matchesType =
    element ipsrc { ipType },
    element ipdst { ipType },
    element protocol { protoType },
    element prtsrc { portType }?,
    element prtdst { portType }?,
    element icmptype { icmptType }?

  natOrigPacket =
    element ipsrc { ipType }?,
    element ipdst { ipType }?,
    element protocol { protoType }?,
    element prtsrc { portType }?,
    element prtdst { portType }?,
    element icmptype { icmptType }?

  TnatPacketSrc =
          element ipsrc { ipType }

  TnatPacketDst =
          element ipdst { ipType },
          element prtdst { portType }?

  ipType = xsd:string { pattern = "((25[0-5]|2[0-4][0-
9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-
9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-
4][0-9]|[01]?[0-9][0-9]?)(/[0-3]?[0-9])?)|([a-zA-Z0-9])+" }

  protoType = xsd:string { pattern = "tcp|udp|icmp|([a-zA-
Z])+|([0-2]?[0-9]?[0-9])" }

  portType = xsd:string { pattern = "(([0-6]?[0-9]?[0-9]?[0-
9]?[0-9])(:[0-6]?[0-9]?[0-9]?[0-9]?[0-9])?)|([a-zA-Z])+" }

  icmptType = xsd:string { pattern = "([0-2]?[0-9]?[0-
9])|([a-zA-Z])+" }
}
```

Figure 3. AFPL2 Grammar in RelaxNG Copact