Semantics, implementation and performance of dynamic access lists for TCP/IP packet filtering

Scott Hazelhurst

School of Computer Science
University of the Witwatersrand, Johannesburg,
Private Bag 3, 2050 Wits, South Africa
scott@cs.wits.ac.za

Abstract

The use of IP filtering to improve system security is well established, and although limited in what it can achieve has proved to be efficient and effective. In the design of a security policy there is always a trade-off between usability and security. Static access lists make finding a balance particularly stark. Dynamic access lists would allow the rules to change for short periods of time, and to allow local changes by non-experts. The network administrator can set basic security guide-lines which allow certain basic services only. All other services are restricted, but users are able to request temporary exceptions in order to allow additional access to the network. These exceptions are granted depending on the privileges of the user. The paper presents and justifies a semantics for dynamic access lists. An efficient method of implementing the dynamic semantics is proposed and experimentally validated. The experiments show that a useful dynamic semantics can be implemented with small memory costs and modest time costs.
Keywords: firewalls, TCP/IP filtering, dynamic rules
CR Categories: C.2.0, C.2.3, C.2.6, K.6.5

1 Introduction

The use of IP filtering as a means of improving system security is well established. Although there are limitations at what can be achieved doing relatively low-level filtering, IP level filtering has proved to be efficient and effective [14].

The access lists that are used to implement IP filtering contain rules that specify which packets should be allowed to pass through the firewall. Access lists may last for several years and so may be changed from time to time (rules may be added or deleted, old rules changed, or the order of the rules change). Nevertheless, an access list is relatively static and change requires the intervention of the system administrator.

The problem with a static access list is that the level of security is relatively static. This becomes increasingly a problem as the range and type of network travel increases. Applications increasingly make transparent connections across the internet (e.g. to check for updates, licences, retrieve data). To allow all possible access all the time creates potential hazards.

Striking the right balance between usability and security is one of the key issues in network design. Using static access lists makes choices in finding a balance particularly stark. Restricting access means that legitimate use of the network is prevented; allowing access means illegitimate use may be allowed. A user may only need certain accesses for 15 minutes a day (1% of the time), but when they need the access they really need the access. On the other hand, keeping access available 99% of the time when no benefit accrues seems too liberal. One should only take a risk when some benefit may result. As an analogy, after I do a large grocery shopping I might leave my car door and front door wide open while I trudge back and forth carrying grocery packets because it makes the job easier and faster, but I certainly don't leave the doors open all the time.

The idea behind dynamic access lists is to allow the rules to change for short periods of time, and to allow local changes by non-experts. The network administrator can set basic security guide-lines which allow certain basic services only. All other services are restricted. However, users are able to request temporary exceptions in order to allow additional access to the network. These exceptions are granted, depending on the privileges of the user.

Dynamic access lists have been used in Cisco routers for some time [4]. What is being proposed here though is a much more general framework for making access lists dynamic, as well as a mechanism for efficient implementation and management. One of the problems with the current Cisco dynamic access list mechanism is that it imposes a performance penalty (for example, the lists cannot be compiled into lookup tables) (`http://www.cisco.com/univercd/cc/td/doc/product/software/ios121/121newft/121limit/121e/121e1/eturbacl.pdf`).

Support for the need for dynamic policies can be found in the recent literature [10, 12, 13, 18]. I argue that the advantage of dynamic access lists is that it allows more flexibility, allowing defence in depth. In addition, we are

able to take into account different needs of different user classes, rather than just physical location, and support mobile computing.

Structure of the paper

Section 2 gives a basic introduction to TCP/IP filtering and explains some of the relevant issues and techniques. Section 3 surveys possible semantics for dynamic access lists and proposes one which is argued makes intuitive sense and is sound. Section 4 presents the outline of a proposed protocol for allowing dynamic access. Section 5 describes a method for representing access lists so that dynamic update and look-up can be done efficiently. Section 6 presents a set of experiments conducted to evaluate the performance characteristics of the proposed semantics.

2 Background

2.1 Firewalling

Security can be provided at a number of different levels and in different places. For example, we may secure individual computers or we may secure whole networks (though potentially with what is proposed here, a more distributed scheme could be considered). There are different advantages and disadvantages of these different approaches – see [14] for a discussion.

Firewalling can be done at different levels. For example, proxies use application-layer information in controlling network connections. Because they can use high-level information, they are able to make good quality decisions. However, this imposes extra costs.

IP-level filtering is much simpler and therefore cheaper, although this limits the intelligence of the filtering. The use of user-classes in the dynamic approach may increase the intelligence of the approach.

Even though the IP filtering is relatively efficient, the cost of filtering may still be a significant bottle-neck [2]. Significant work has gone into improving the performance of IP filtering [7, 8, 11, 17]. The fact that filtering is a bottle-neck means that dynamic filtering must not introduce significant extra costs.

2.2 IP filtering and Rule sets

TCP/IP filtering is a slightly misleading terminology since in fact it means filtering using information found in the internet, network and transport layer headers (depending on the protocol suite). Typically the information that can be found in the these headers is:

- source and target addresses of the packet;
- the protocol of the packet (e.g. udp, tcp, icmp, ...);
- ports;
- certain flags (for example, a tcp packet contains flags indicating status for connection control).

See a standard reference for more details (e.g. [20]). This paper considers TCP/IP packets in particular, but the methods generalise to similar protocols.

Filter rules come in several formats; typically these are proprietary formats. While the expressiveness and syntax of the formats differ, the following generic description gives a good feeling for what such rules sets look like. A rule set consists of a list of rules of the form

```
if (condition) then action
```

where the action is either accept or reject.

Example: A rule in a rule list for a Cisco router [5] might say something like:

```
access-list 101 permit tcp
            20.9.17.8  0.0.0.0
            121.11.127.20 0.0.0.0
            range 23 27
```

This says that any TCP protocol packet coming from IP address 20.9.17.8 destined for IP address 121.11.127.20 is to be accepted provided the destination port address is in the range $23 \ldots 27$. □

Masking: A rule can specify a range of addresses by using *masking* for both the source and destination addresses (in the above, the masks were 0.0.0.0, which means no masking). An address is actually a 32 bit number, which is convenient for humans to express in the quad notation (four numbers each in the range $0 \ldots 255$). A mask is expressed similarly. If a "1" appears in the mask, then the value of the corresponding bit in the address is ignored in matching. In the above example, since the masks are all 0 the addresses must match exactly. But, if we had

```
20.9.17.8  0.0.0.255
```

as the source address, then any address with 20.9.17 as a prefix would match. If

```
20.9.17.8  0.0.0.254
```

were the source, then any *even* address with 20.9.17 as a prefix would match.

Matching a list of rules: The rules are searched one by one to see whether the condition matches the incoming packet: if it does, the packet is accepted or rejected depending on the action (which will either be accept or reject); if the condition does not match the rule, the search continues with the following rules. If none of the rules match, the packet is rejected.

Since the rules are checked in order, the order in which they are specified is critical. Changing the order of the rules could result in some packets that were previously rejected being accepted (and/or *vice-versa*).

This paper uses Cisco access-list format as the basis specifying the rule set, but the methods proposed generalise to other formats.

2.3 Comparison to existing work

The Cisco dynamic access list scheme allows the network administrator to specify that certain rules are *dynamic*. The default behaviour of these rules is to have no effect at all. However, a user can log in to the router, authenticate itself and then make these rules active.

Although useful, this scheme is not as general as the framework described above. First, the network administrator has to specify the dynamic access beforehand. Thus, although an improvement over purely static access lists, it is not completely dynamic: if the dynamic rules are not made liberal enough then a user may not be able to get access needed, whereas if they are made too liberal, then when a dynamic rule is made active, too wide a window is left open. Dynamic rules also appear to be treated differently to normal rules, which means that more sophisticated dynamic rules will run into the problem of the interference between ordering of rules and dynamic rules.

Details of the semantics and performance characteristics of Cisco's implementation is not available in the scientific literature and I have been unable to find any quantitative analysis.

There has been some recent work on policy or model-based approaches to semantics [10, 12, 13, 18]. The aim is to allow a very high level specification of system polices and needs. The strength of this work is that it presents the policy from the user or role perspective, rather than from a computer or IP address perspective. The semantics presented in this paper would form a good intermediate semantics between the high-level policy representations and the low-level, technology specific format of access lists. For example, the proposal that there should be an authorisation enforcement agent that monitored policy and network conditions [18] could use the mechanisms proposed in this paper for efficient implementation. The work cited above does not address the question of efficient implementation, which is central to this paper. Thus, this paper is complementary to that work.

3 The Semantics of Exceptions

The architecture of this dynamic access list proposal is that the network will have

- a *base* list of rules which protects the network, allows only the very basic services, and sets policies for what dynamic rules will be allowed; and
- *dynamic* rules which will be used to allow services when needed and requested. The dynamic rules must fit the policies of the base list.

There are a number of concerns: allowing users to ask for dynamic access easily, providing the system administrator confidence that key security requirements will not be broken, and understanding the effect of different rules. A significant complication arises from the importance of the order of rules in an access list to the semantics.

This section discusses why the ordering constraint is an issue, presents principles for dynamic access list semantics, and proposes possible semantics. The focus of this section is on principle, not on performance, which is dealt with later in the paper.

3.1 Principles of semantics

The following principles have guided the development of the semantics proposed later.

1. The semantics should support flexible policies.
2. The behaviour of dynamic changes and the interaction between the dynamic rules and the base rules must be clearly understandable to the person maintaining the base list, without knowledge of what the dynamic rules are. The key points are
   - the administrator must have confidence that the security of their system will not be threatened;
   - there must not be a significant extra burden placed on the creation and maintenance of the base list.
3. It must be possible for a user to request a dynamic access rule without knowledge of the base rules or other dynamic rules.

   Of course, the firewall may not allow such a request if it conflicts with security policy, but a non-expert user (or a simple agent on behalf of a user) should be able to make the requests.
4. Ideally, we should be able to use existing rule sets without much change.

Efficiency is obviously also an important question, addressed in Section 5.

Limitations: A strength of this proposal is that it does not require any changes to the networking protocols used (e.g. IP). However, the downside of this is that IP addresses are central to the security mechanism. With the scheme proposed here the problem is somewhat ameliorated since it allows different users with different privileges to control different ports.

3.2 A simple semantics

At first, it might seem that dynamic rules can be implemented just by considering the exceptions as an extension of the base list. In this view, the exceptions are temporarily pre-pended or appended to the base list. While this is simple to implement and in some cases could have an understandable effect, it is highly problematic. It fails the principle of supporting flexible policies as explained below.

Suppose the extensions are pre-pended to the base list. Then, because ordering is critical, all extensions will over-ride all base rules: there is no way of ensuring that certain rules are always obeyed.

Conversely, if the rules are appended then an exception can never over-ride any of the base rules: in effect, an exception is only an exception to the default reject rule. This is more desirable and useful than prepending, but it reduces the flexibility of a security policy, and would be likely to lead to a situation where the base rules are more liberal than necessary in order to allow the exceptions to have some use.

More sophisticated direct modification of the access lists seems unlikely given the high-level of expertise required to make changes and the complexity of the interplay between rules. Inserting dynamic rules somewhere in the middle of the list may give more flexibility but will not be flexible enough, and be too difficult to understand the effect.

3.3 The multiple list priority-based approach

The multiple list priority-based approach uses multiple rule lists, ordered by priority. For each list the reject rules are split into mandatory or flexible.

The security policy is succinctly expressed as:

> A packet p is accepted by the firewall if it is accepted by rule list j and there exists no mandatory reject rule in lists $0 \ldots j-1$ that matches it.

Dynamic changes can be effected by allowing users to prepend accept rules to one of the lower-priority lists, depending on the users' privilege levels.

The semantics are acceptable when measured against the principles of Section 3.1, and it allows an efficient implementation. However, the disadvantage of this approach is that it requires the duplication of rules. For example, suppose that we wish to have as a default or base rule that access to machine x is not allowed, but that users in class 1 to i should be allowed to over-ride this rule, and users in class $i+1$ and above should not. Implementing this requires flexible reject rules in the base list and in lists $1 \ldots i-1$, and a mandatory reject rule in list i. This proliferation of rules will make administration much more difficult and increase the chance of errors.

3.4 The group-based approach

In the group-based approach (GBA), there is one base access list and n exception lists (numbered $0 \ldots n-1$, assuming n user groups). The base access list is the existing access list, except that each reject rule has a set of associated group identifiers $0 \ldots n-1$. These group identifiers indicate which groups are able to over-ride the reject rule. Groups may be contained within other groups. The semantics of the GBA are:

> A packet p is accepted by the firewall if
>
> 1. It is accepted by the base list; or
> 2. It is accepted by some exception list j where all reject rules in the base list that match p are labelled by j or by a super-group of j.

Thus, if a rule in the base list is not labelled with any group identifier then it cannot be over-ridden. I argue that this semantics meets the conditions of Section 3.1. However, one problem is how to deal with the implicit deny all rule at the end of the list, since this matches all packets. The solution adopted is to say that an exception can only be placed in an exception list j if there is a deny rule labelled j that covers the exception. It is this idea that the next section explores.

3.5 Generalised Group Dynamic Access

The generalised priority list (GGDA) approach is similar to GBA in that it uses a base list, n groups and n exception lists. The difference is the semantics:

> A packet p is accepted by the firewall if it is accepted by the base list or one of the exception lists. The proviso is that an exception can only be added to list j if
>
> - There is a deny rule γ labelled j (or a super-group of j) that matches the exception; and
> - No other deny rule that matches the exception with a label other than j or a super-group of j appears before γ in the list.

This is formalised in Section 5.2. The GGDA philosophy is to keep as much as possible in the framework of the current semantics of access lists, using the order of the rules to help decide access. It is a generalisation of the current semantics of access lists: a rule can be definitely yes or no, or allow users with different levels of privilege to over-ride them. Thus, users in group j can see deny rules labelled j as accepts or denies (depending on whether they want additional access), whereas other users will see the rules as denies. We shall see that this perspective of a generalised semantics carries over strongly in the implementation.

The arguments for this semantics are:

1. Flexible policies are supported since by assigning several priority levels for reject rules, the network administrator can allow different classes of user different abilities to request dynamic access.
2. Interaction between the base list and the exceptions is clear since (a) the normal semantics of the access lists is disturbed very little; and (b) the use of group membership for the reject rules makes it clear what different classes of users can and can't do.
3. Users do not need to know what the base rules are in making a request for dynamic access.
4. Existing rule sets can be used as-is. Dynamic access can then be implemented over time by assigning group membership. There is no need for radical change.

The efficient implementation of the semantics is discussed in Section 5.

3.6 Example

Here we look at a simple example shown in Figure 1. The format used is similar to the Cisco access list format; the number at the beginning of each line is just used for reference in this explanation. The example gives rules for a subnet 128.128.128. The machine 128.128.128.15 is a special server; the other machines in the range 128.128.128.0 to 128.128.128.127 are for staff; machines in the range 128.128.128.128 to 128.128.128.255 are for students. In this example, group 0 is the *staff* group, group 1 the *student* group, and group 2 the *all* group, which contains both groups 0 and 1.

Rule 0 says that we accept tcp connections to the machine 15 on port 88. Rule 1 says that we deny (with no exceptions allowed) all tcp connections to machine 15 on any port. Since rules are examined in order the combined effect of these two rules is that tcp connections to machine 15 are accepted on port 88 only. Rules 2 and 3 say that on all machines on the subnet we accept tcp connections on port 88 and ports in the range 32000 to 65535. (Since rules 0 and 1 come before rules 2 and 3, rules 2 and 3 will not affect machine 15 since we have denied access to ports other than port 88 on machine 15). Rule 4 denies tcp connections to any machine on the subnet to any port in the range 0 through 87 and no derogation from this rule is allowed.

Rule 5 says that we deny tcp connections to any of the staff machines (range 128.128.128.128 to 128.128.128.255) on any port numbered 89 or higher. However, we allow members of the staff group to request exceptions to this rule. Note there is some overlap between this rule and previous rules: in this overlap the previous rules take priority because they come first.

Rule 6 says that we deny tcp connections to any student machines on any port numbered less than 16000. (Since rule 3 comes first, connections to port 88 are still allowed). Rule 7 says that we deny any connections to student machines on ports with a number greater than or equal to 16000. Members of the student group are allowed to request exceptions to this rule. (Connections to ports ≥ 32000 are still allowed since rule 4 comes first.)

Finally, rule 8 denies any other packets. However, any member of either the staff or student group can ask for exceptions.

Note, that without any exceptions, the semantics of the access list is unchanged from the standard semantics.

Now, let's look at the effect of exceptions. Suppose we have the exception lists as shown in Figure 2; the numbering of exception lists and groups correspond.

Exception 0.0 allows tcp connections to be made to port 100 on machine 1. Any packets that come to this port will be accepted because they are accepted by exception list 0 and the only rules that deny access to this port are rules 5 and 8. Both allow exceptions to be requested by members of group 0 and so this exception can be honoured.

Exception 0.1 purports to allow tcp exceptions to ports 0 through 90 on machine 1. However, packets that come to ports 0 through 87 will not be enabled by this exception since they are rejected by rule 4 which does not allow derogations. TCP packets going to port 88 were in any event allowed by rule 3. Packets going to port 89 and 90 will be allowed by the exception since the rules that deny access to this port (5 and 8) allow exceptions to be made by members of group 0.

Exception 0.2 purports to allow packets going to port 16000 on machine 129. Under the GBA, this would not be allowed since although rule 5 appears to allow the exception, rule 7 effectively denies it. On the other hand, the GGDA semantics would allow it since rule 5 comes before rule 7. I argue that this follows the normal semantics of access lists and so will be more understandable to administrators.

Exception 1.0 purports to allow tcp connections to port 100 on machine 2. This has no effect, however, since rule 5 denies access to this port and as the rule only allows derogations by staff member, only exceptions in exception list 0 can over-ride it. (In fact, under the GGDA semantics, this exception wouldn't even be allowed to be put in the list.)

Exception 1.1 does have effect though since it allows access to port 18000 on machine 129 and the relevant deny rule (7) is labelled group 1 (and hence can be over-ridden in exception list 1). Note that access to port 16000 would not be allowed because of rule (5).

Similarly, exception 1.2 will allow icmp packets to reach machine 129 since the relevant deny rule (8) is labelled 2 (the *all* group).

Exception 2.0 purports to allow tcp connections to port 16000 on machine 130. However, as rule 7 denies such connections and is labelled group 1, exceptions that override rule 8 must be in exception list 1. However, exception 2.1 does allow icmp access to machine 130 since rule 8 is labelled 2.

Notes

- If the groups are set up in a strictly hierarchical way, the group-based approach corresponds to a priority-based one.

- Should *accept* rules also have priorities and allow exceptional denies? This has the appeal of symmetry, and would allow extra functionality (for example, to allow an authorisation enforcement agent to take action when intrusion was detected). There are no inherent implementation issues why this should not be supported, but it needs some thought.

4 User-Firewall Interaction Protocol

This section proposes a protocol for dynamic access list by specifying how communication between user processes and the firewall takes place. When a user (which might be an agent acting on behalf of a human, rather than a hu-

```
0: accept tcp 0.0.0.0 255.255.255.255  128.128.128.15  0.0.0.0  eq 88
1: deny   tcp 0.0.0.0 255.255.255.255  128.128.128.15  0.0.0.0
2: accept tcp 0.0.0.0 255.255.255.255  128.128.128.0   0.0.0.255 eq 88
3: accept tcp 0.0.0.0 255.255.255.255  128.128.128.0   0.0.0.255 ge 32000
4: deny   tcp 0.0.0.0 255.255.255.255  128.128.128.0   0.0.0.255 range 0 87
5: deny 0 tcp 0.0.0.0 255.255.255.255  128.128.128.128 0.0.0.127 range 89 17000
6: deny   tcp 0.0.0.0 255.255.255.255  128.128.128.128 0.0.0.127 lt 16000
7: deny 1 tcp 0.0.0.0 255.255.255.255  128.128.128.128 0.0.0.127 ge 16000
8: deny 2  everything
```

Figure 1: Simple example dynamic base list

.

```
Exception list 0 (staff)
0.0 accept tcp 0.0.0.0 255.255.255.255  128.128.128.1  0.0.0.0  eq 100
0.1 accept tcp 0.0.0.0 255.255.255.255  128.128.128.1  0.0.0.0  range 0 90
0.2 accept tcp 0.0.0.0 255.255.255.255  128.128.128.129  0.0.0.0  eq 16000


Exception list 1 (student)
1.0 accept tcp 0.0.0.0 255.255.255.255  128.128.128.2  0.0.0.0  eq 100
1.1 accept tcp 0.0.0.0 255.255.255.255  128.128.128.129  0.0.0.0  eq 18000
1.2 accept icmp 0.0.0.0 255.255.255.255  128.128.128.129  0.0.0.0

Exception list 2 (all)
2.0 accept tcp 0.0.0.0 255.255.255.255  128.128.128.130  0.0.0.0  eq 16000
2.1 accept icmp 0.0.0.0 255.255.255.255  128.128.128.130 0.0.0.0
```

Figure 2: Exception Lists

man) wishes to ask for dynamic access, it sends a request to the firewall. The firewall logs and validates the request, checks to see whether the request can be fulfilled and then responds to the user. This section describes this communication in more detail. How the firewall maintains the requests, performs updates, and does look-ups is covered in Section 5.

4.1 Request for dynamic access

A request for dynamic access takes four steps:

- The user sends a request to the firewall asking for access;
- The firewall sends back a response indicating to what degree the access can be given;
- The user then sends a message to confirm whether it wants the access given.
- If the firewall receives the confirmation within a given time interval, the exception is made; otherwise the exception is not made.

These are now explained in more detail.

First step – asking for access: The user sends a packet to the firewall asking for access to be given. The packet is sent as a UDP packet to a well-known port on the firewall, and contains the following information:

- Originating IP address and UDP port (part of the IP/UDP headers).
- User identification: this must enable the firewall to identify the group of the user.
- Access required: this would be a list of accept rules indicating what access is wanted.
- Expiry time: The user specifies for how long the exception should be enabled.

Firewall response: When the firewall receives the *request* packet, it validates the user and determines the group membership. It then examines the update request and determines using the techniques discussed in Section 5 whether the request can be met: the request could be met completely, partially or not at all. At the same time, the firewall inserts the update in a queue of pending exceptions.

There are number of open implementation decisions: should the firewall accept update changes from outside or only from inside? How are users identified? It may be the userid of a user that the firewall knows about (it could have its own database or use NIS), or it could be a capability-based system. The identification could be authenticated by digital signature. This and other messages in the exchange may or may not be encrypted.

What information is returned if the request can only be met partially is another implementation decision. In the approach proposed in Section 5 a full description will be returned. However, there might be reason just responding with a *reject* or *allow*.

Confirmation by user: If the user receives a *reject* message, then it has to reconsider what it wants. If the user

receives an *allow* message, it decides whether it wishes to use the update, and if so it sends back to the firewall a *confirm* message together with the unique ID. This step ensures that the firewall only implements changes that the user really wants. It also helps reduce the chance of spoofing attacks on the firewall.

Firewall implements the exception: When the firewall receives the *confirm* request it removes the update request from the pending queue, and adds it to the appropriate exception list, timestamping the update as it does so.

Periodically, the update queue is scanned and old requests are purged. A user may wish to extend the life-time of an existing exception. This could be done with a *renew* mechanism.

4.2 Undoing an update

Since the access list is supposed to be dynamic, it must be possible to undo the change. This can be done easily by deleting the exception. There are two proposed mechanisms for this:

- The user sends a *delete* request to the firewall with the unique ID of the update. The firewall authenticates the request and then deletes the update.
- The firewall periodically checks the exceptions and when an exception has expired (the time since it was added to the exception list longer than the expiry time associated with the exception), the exception is deleted.

Other mechanisms are possible. For example, the firewall could monitor the traffic associated with the exception and when it detects that there has been no traffic for some period then the exception is deleted. However, this is likely to be considerably more complex and heavy-weight than the proposed method here.

5 An implementation mechanism for dynamic lists

As performance has been raised as a problem with related work, and the approach proposed here is a much more general framework, the potential negative effects on performance clearly need attention.

The following principles and assumptions guide the implementation mechanism proposed here. The principles are listed in descending order of importance:

- The cost of doing look-up for packets that are not affected by dynamic update rules should not see significant performance degradation;
- There will be traffic associated with each update rule and so the cost of look-up for exceptional packets must be small;
- The cost of updating and undoing updates must be small (though as this happens relatively rarely compared to packet traffic, the cost is not that critical).

This section is structured as follows: Section 5.1 discusses the basic method for representing access lists. The GGDA semantics is then formalised. These are the most interesting semantics; the other semantics can be formalised in similar ways. Section 5.2 presents the method of efficiently performing updates to the access list and performing lookup; and Section 5.4 shows how undoing updates can be performed.

5.1 Basic Representation of Access Lists

The chosen method for representing an access list is as a single (rather large) boolean expression. As described in Section 2 each rule in the access list is a condition on the bits in the packet header, and hence if we represent each bit in the packet header with a boolean variable, we can represent the condition as a boolean expression over these variables. Given a packet to filter, we give the variables in the expression their values as given by the bits in the packet header. The packet is accepted exactly when the boolean expression evaluates to true.

The detailed mechanics of this translation are beyond the scope of this paper. The important point is that these boolean expressions can be efficiently represented using binary decision diagrams (BDDs) [3]. See previous work [8, 9] for a detailed explanation of how this is done. The work of Attar [1] and Sinnappan [15] shows that this method of representing access lists is a very compact representation and that look-up can be performed competitively (with respect to other methods) in both software and hardware. What particularly made this representation scheme appear promising for dynamic access lists were the results that (a) the cost of lookup is robust in the number of access rules and (b) the variation on look-up cost is very low. Thus, we believe that this implementation will not lead to a significant performance penalty.

Besides the potential implementation advantages, the logical representation gives a sound, understandable formal semantics for dynamic access list. These are human understandable and automatic tools can be used for analysing them.

5.2 Representation of base list and exceptions: GGDA

The GGDA has a precise formal semantics. The exceptions that can be made to an access list can be described by a boolean expression which is constructed in the following way.

- Let the access list $\mathcal{B}$ consist of the rules $\langle r_1, \ldots, r_n \rangle$.

  Let $\mathcal{E}(\mathcal{B}, g)$ be a boolean function which describes the exceptions which a user from group g can request.

  Let $\phi(r)$ be the boolean expression associated with rule r, and $g(r)$ be the label(s) of the rule r.

Let $\phi_{\mathcal{B}}$ be the boolean expression representing the base access list.

Let ϕ_A be the boolean expression representing the access list together with exceptions.

For the base case of an empty list, $\mathcal{E}(\langle\rangle, g) \stackrel{\text{def}}{=} \mathbf{f}$. In general, we define $\mathcal{E}(\langle r_1, \ldots, r_n\rangle, g)$ depending on whether the first rule is an accept or reject rule. If r_1 is an accept rule, then

$$\mathcal{E}(\langle r_1, \ldots, r_n\rangle, g) \stackrel{\text{def}}{=} \mathcal{E}(\langle r_2, \ldots, r_n\rangle, g)$$

since accept rules have no effect on exceptions. If r_1 is a reject rule:

$$\begin{aligned}&\mathcal{E}(\langle r_1, \ldots, r_n\rangle, g) \stackrel{\text{def}}{=} \\ &\quad (g \in g(r_1)) \wedge (\phi(r_1) \vee \mathcal{E}_{\mathcal{B}}(\langle r_2, \ldots, r_n\rangle, g)) \vee \\ &\quad (g \notin g(r_1)) \wedge \neg\phi(r_1) \wedge \mathcal{E}_{\mathcal{B}}(\langle r_2, \ldots, r_n\rangle, g)\end{aligned}$$

This says that the exceptions that someone from group g can request from the access list with r_1 as the first rule are

- if g is a member of a group labelled by r_1 then anything that is covered by r_1, or any exception that such a user can request of the rest of the list; or
- if g is not a member of the group labelled by r_1 then anything that is *not* covered by the condition of r_1 *and* is permitted to such a user by the rest of the list.

We also keep for each group, j, a boolean expression, ϵ_j representing the exceptions that have been requested by group j, whether these exceptions are allowed or not.
Under the GGDA semantics given in Section 3.5 (on page 4) the boolean expression that represents a base access list and n exception lists is:

$$\phi_A \stackrel{\text{def}}{=} \phi_B \ \vee \ (\vee_{i=0}^{n-1} \epsilon_i \wedge \mathcal{E}(\mathcal{B}, i)) \qquad (1)$$

The one concern from an efficiency point of view is that we have to store $\mathcal{E}(\mathcal{B}, i)$ for all i. However, this is unlikely to be a problem for several reasons. First, the size of the boolean representation of access lists is small and so keeping many copies is unlikely to be a problem. Moreover, there are likely to be many shared structures in the BDDs that represent the different boolean expressions and BDD managers can efficiently share these structures. Finally, if there really were a problem (say we had hundreds of groups, though unlikely) we could easily represent all the $\mathcal{E}(\mathcal{B}, i)$ efficiently and compactly with one symbolic expression in the following way:

- Introduce $\lceil \log n \rceil$ boolean variable $g_1, \ldots g_n$;
- $\mathcal{E}(\mathcal{B}, i)$ can now be computed symbolically using the rules above and represented using a BDD including the g_i variables.

In the discussion below, we only consider GGDA. However, it would be easy to make the necessary modifications to change to GBA or other approaches.

5.3 Update process

The firewall keeps the following information:

- ϕ_B, the representation of the base list;
- For each update request u, ϕ_u, the boolean expression representing the exception, the ID of the request, expiry time and the group of the originator of the request;

  The update requests are stored in a manner so that they can efficiently be accessed by ID number, by expiry time, and by group number.
- For each i, $\mathcal{E}(\mathcal{B}, i)$, the exception permitted by the base list for group i;
- For each i, ϵ_i, the exceptions requested by group i;

Update request received: When a new update u in terms of the protocol described in Section 4 arrives with originator priority j, the following is done:

- ϕ_u is computed, the boolean expression representing u;
- $\phi_u \wedge \mathcal{E}(\mathcal{B}, j)$ is computed: This represents which exceptions that a user of this level of priority can be granted.
  - If $\phi_u \wedge \mathcal{E}(\mathcal{B}, j) = \mathbf{f}$, then none of the exceptions can be granted;
  - If $\phi_u \wedge \mathcal{E}(\mathcal{B}, j) = \phi_u$, then all the exceptions can be granted;
  - Otherwise only some of the requests can be granted.
- If none of the exceptions can be granted, the request is deleted and the user is sent a *reject* message.
- Otherwise the user can be sent an *allow full* or *allow partial* message, and the request is put on the pending list. In the case of an *allow partial* message various types of information can be returned to the user. The most useful would be tabular representation of $\phi_u \wedge \mathcal{E}(\mathcal{B}, j)$. An algorithm for presenting this is described in [8].

Confirm message received: When the firewall receives a *confirm* message for update u, the following happens:

- The update is stored with the associated information in an easily accessible form as described above;
- ϵ_j is updated: $\epsilon_j \leftarrow \epsilon_j \vee \phi_u \wedge \mathcal{E}(\mathcal{B}, j)$;
- ϕ_A is recomputed according to Equation 2 by computing $\phi_A \leftarrow \phi_A \vee \phi_u \wedge \mathcal{E}(\mathcal{B}, j)$.

Other semantics

The logical representation allows other semantics to be formalised in the same way. The GBA semantics is given in the appendix to illustrate this point.

5.4 Undoing updates

In principle, undoing an exception is straight-forward. When an exception of priority j is undone,

- The exception is removed from the list of active exceptions;
- ϵ_j is recomputed;
- ϕ_A is recomputed according to Equation 1.

Although a simple inspection of the undo algorithm may give the appearance that undoing may be more expensive than creating the exceptions because there seems to be more computation to be done, there are a number of mitigating factors:

- Undoing is unlikely to be time-critical (i.e. if it takes a few seconds extra to undo the exception, there will be no serious consequences). Thus, we only need to consider the cost in terms of the load it places on the server, and latency is not an issue. And since it is not critical, undoing can be done as a low-priority process.
- The effective use caching of results by the BDD manager means that in some cases the results of the computation are already available.

However, the cost of undoing is likely to be the critical factor in the performance of this approach and so will be examined in the next section.

6 Experiments

This section presents an experimental evaluation of dynamic access lists. The following measurements were made.

- Memory costs. Given previous results, memory is not expected to be a constraint, given that the BDD representations are very compact. Nevertheless, this is something that needs to be confirmed.
- The time to make an exception.
- The time to cancel an exception.
- The effect of exceptions on lookup costs for unrelated packets.
- The effect of exceptions on lookup costs packets that are affected by the exceptions.

All experimentation was performed on an 800MHz Pentium III processor with 500MB of RAM.

6.1 Implementation

A prototype implementation of the semantics of Section 5.2 has been built. This is a C program, which uses the CUDD BDD package [16]. The prototype can take a base list, an exception request queue, a list of groups and relations between groups and construct the base and exception list BDDs, as well as compute the overall BDD. The exception request queue contains a list of exception requests which can either be requests to make exceptions (in which case it contains the exception label as well as the exception itself) or it can be a request to undo the exception.

The prototype takes a given packet header trace and then successively performs a look-up, returning the look-up results as well as performance statistics. For the experiments below, we wish to focus solely on lookup costs (since other packet handling costs are not affected by this proposal), so we do not interface with a real network, but rather take as input a file of appropriate packet headers.

There is also an auxiliary program that can be used to generate packet traces of different types. We have implemented an algorithm that takes the BDDs that represent a base list and exceptions, and generates packets according to some criteria (e.g. generate packets that match *these* rules but not *those*). This proves very useful for experimentation.

6.2 Experimental data

Two rule sets were used as base sets. These are rule sets we generated in previous research for experimentation purpose. They are rule sets designed for different artificial but realistic networks. The virtue of using artificial sets is that it allows us to control for rule size without changing the semantics of the rules (see [15] for a discussion on the approach). One of the advantages is that we can test different scenarios and so investigate the robustness of the approach.

The set R-A was used as the main rule set. It is a list of 500 rules and was used as-is, without any changes. It should be noted that a number of the rules were redundant. From this set, the set R-A-1 was created where most of the permit rules had been turned into deny rules for which exceptions could be requested. The following exception lists were made:

- Exc-A.1: a set of 50 exceptions (a few exceptions to a number of the deny rules)
- Exc-A.2: a set of 100 exceptions (many exceptions to a few of the rules).
- Exc-A.3: the combination of the rules.

These exceptions were constructed to model different realistic situations. For example, in the base list a common service was listed with a *deny*, and the exceptions then allowed access to specific users or ports.

A second base rule set, R-B (330 rules), allowed us to test more complex cases. A set of 300 rules for a realistic network was edited to ensure that no rules were redundant as well as to add significant complexity to some cases. About 30 complex and unusual rules were added in order to see how the method would work for much more complex systems and to test corner cases. This editing tripled the memory complexity of the BDD representation of the

resulting lists. From this base list, two other lists were generated:

- Set R-B-1: the original list was modified by changing all but 10 permit rules to deny rules. These deny rules were labelled with a different label to that of the original deny rules: these rules became the rules for which exceptions could be asked.

  This is intended as a realistic test case: it would make sense for the firewall to have only a few permanent permit rules and then use the exception mechanism to open holes in the firewall when needed.

- Set R-B-2: the original list was modified by changing 20% of the permit rules to deny rules. These deny rules were labelled with a different label to that of the original deny rules as above. This tests more extreme cases.

The following exception lists were used:

- Exc-B.1: a set of 50 exceptions requesting a range of different source, destination, and port requests.
- Exc-B.2: a set of 50 exceptions requesting POP access for a particular destination address and port from a range of different source and port addresses (these exceptions were to the labelled deny rules in R-B-1/2).
- Exc-B.3: Exc-B.1 plus Exc-B.2 combined.

For each test case Exc-X.y, we also ran a test case Exc-X.y/C in which all the exceptions were first created, and then cancelled. The order of cancellation was pseudo-random and so not in the same order as the creation.

6.3 Memory usage

Table 1 shows the cost of representing the access lists and exception lists. For each access list, the column headed *base* shows the memory cost of representing the access list, and the column headed Exc-Xy shows the cost of representing the access list and all the exceptions in list Exc-Xy. The figures shown are the number of BDD nodes required to represent the access list: each node could comfortably be stored in 32 bytes (so the total memory usage would be less than 500K for the most complex list used). Note that the rows labelled R-A and R-B are given in order to show cost of building the original base list. No cost of making exceptions is given since no exceptions are permitted for these lists; exceptions are only permitted for the derived lists.

6.4 Time cost of constructing the access list and exceptions

Table 2 shows the cost of creating the access exception lists and undoing them. For each access list the column headed *Base* is the cost of building the entire access list. The columns labelled Exc-Xy show the average cost in microseconds of making the exceptions in list Exc-Xy. The columns labelled Exc-Xy/C show the average cost in microseconds of making and undoing the exceptions in list Exc-Xy/C.

Test case	Base	Exc-A1	Exc-A2	Exc-A3
R-A	2851			
R-A-1	2124	2706	2900	3482
Test case	Base	Exc-B1	Exc-B2	Exc-B3
R-B	10373			
R-B-1	2096	3357	3436	4697
R-B-2	9299	11143	10647	12635

Table 1: Memory costs of representing access and exception lists

These results show that the cost of making or undoing an exception is less than a millisecond and so would not place any overhead load on the firewall itself. The increased latency (bearing in mind that the latency penalty only has to be paid at the beginning of a session) is negligible.

6.5 Time Cost of Lookup

The cost of lookup is very sensitive to the packets being matched. Previous research has shown that the profile of costs is very different for synthetic and trace data [1]. There are many situations where the worst case performance is not good, but since the worst case is unlikely to occur, provided the average case is acceptable the overall performance will be acceptable. Therefore, in general, average case analysis is often the correct analysis to do.

However, here, the worst case analysis is appropriate. For dynamic access lists, the worst case can be expected to occur relatively often. A typical scenario is a case where the network administrator might wish to allow a wide range of access to all machines on the subnet (e.g. allow ssh access to all machines). Typically, the path in a BDD that would 'represent' such a rule would have a depth of about 46. In a dynamic access list, rather than having one general rule covering all ssh accesses, we would have one exception for each required ssh access. Because the exceptions are much more specific, the depth of the path representing each exception might have a depth of 72 or 88. In other words, the lookup time could realistically double. More extreme cases could be considered. Also, if it can be shown that the worst case performance is acceptable then the performance case for dynamic access lists can be made.

The good news is that the BDD representation has the useful property that for a fixed rule format, there is a worst case independent of the size of the access list or the corresponding BDD representation. At worst, the depth of a BDD (and hence the number of steps required for BDD lookup) is the number of bits which are used in filtering. This property must be emphasised because it enables us to give an accurate upper bound on the cost of filtering.

In all the experimentation that was done, the worst case penalty on the cost of lookup time was just over 100%,

Test case	Base	Exc-A1	Exc-A1/C	Exc-A2	Exc-A2/C	Exc-A3	Exc-A3/C
R-A	353ms						
R-A-1	384ms	780μs	647μs	695μs	439μs	653μs	646μs
Test case	Base	Exc-B1	Exc-B1/C	Exc-B2	Exc-B2/C	Exc-B3	Exc-B3/C
R-B	318ms						
R-B-1	382ms	947μs	540μs	633μs	480μs	839μs	511μs
R-B-2	408ms	887μs	527μs	682μs	492μs	815μs	502μs

Table 2: Cost of creating exception lists and undoing them

and the maximum lookup time was 2.1μs.

This results means that in the worst case we can do approximately 500k lookups per second using dynamic access lists. Bearing in mind that the minimum size IP packet over ethernet is about 200 bits, this translates into supporting a bit rate of about 100Mb/s assuming all packets are minimum size. Note that this is the lookup time only and does not include the other costs involved in network processing (these costs are fixed and not dependant on whether dynamic access lists are used). To put this into the context, the main router at the University of the Witwatersrand receives on average about 6200 packets per second at peak-time (measured from 09:00-15:00 on a Monday in the middle of term-time; this includes all incoming traffic to the University).

Detailed analysis: The detailed results of the experimentation are shown below as it is useful to see a finer picture of the costs — though it must be emphasised that from a performance point of view dynamic access lists must stand or fall on the absolute worst case figures given above.

Each experiment shows the comparison for a particular data set (base, exception list, packet trace) the difference in lookup for the given packet trace between using the static list and the dynamic list.

For the table below, the column headed Δ shows the difference in the average lookup cost between the static and dynamic list. This is useful to see, but as discussed previously is not the important figure, and we focus on the number of cases in which the lookup costs more. Due to difficulty in timing very small times accurately (particularly in face of garbage collection which distorts some runs), we use the depth of the path in the BDD as a proxy for time. The only thing that could make this a bad proxy in production runs is cache behaviour; however, the BDD sizes are very small and simple techniques could easily fit them in at least level 2 cache. The column headed $L[x, y)$ shows the number of packets and the percentage of packets in the run for which the penalty of doing a lookup was greater than or equal to x% and less than y%. Note that the table only shows the packets where performance was worse (since that is what we are worried about); in some of the experiments, the vast majority of packets saw improved performance.

Simulated packets were used for computing lookup costs, as it is easier to produce worst case costs with simulated data than real data. The following data sets were used:

- R-A-1/Exc-Ax.0: For each rule in the rule set R-A, we generated 100 random packets that matched that rule, but no other rule above it in the rule set.
- R-A-1/Exc-Ax.1: For each exception requested, we generated 100 random packets that matched that exception but none of the mandatory deny rules or other exceptions above it in set R-A. This is an important data set because it allows us to directly examine the penalty paid for dynamic lists: since all these packets are accepted by both the base list, and by the dynamic list plus exceptions, we can compare the costs directly.
- R-A-1/Exc-Ax.2: For each permit rule in the set R-A, we generated 100 random packets that match that rule. This enables us to measure the penalty of dynamic access lists for those packets which are not affected (semantically) by the dynamic access lists.
- We generated a single packet trace for all the cases, R-B-x/Exc-By. This trace consisted of 100 random packets for each rule in R-B, which matched that rule but no rules above it in the set.
- As rule set R-B-2 differed from the other dynamic lists in that most of the base list's permit rules remained permit rules in the dynamic list, we also created:
  - R-B-2/Exc-B2.1: For each exception we generated 100 random packets that matched that exception but no mandatory deny rule nor any exception above it.
  - R-B-2/Exc-B2.2: For each permit rule in R-B-2, generated 100 random packets that matched that permit rule.

The most important result of the detailed experiment was that we can reliably estimate the maximum cost of lookup at 2.1μs. In the worst case, the penalty was just over 100%, as the theoretical analysis predicted. This result shows that the extra workload penalty on the firewall is reasonable and the extra latency is trivial.
The average time for the trials is only given to show that there are no unexpected negative timing penalties, and it is important to recognise that the test packets were chosen to be representative of the lookup scenarios, and are *not* reflective of typical workloads.

Lists	Δ	L[0,5)	L[5,10)	L[10,20)	L[20,30)	L[30,40)	L[40,∞)
R-A-1/Exc-A1.0	-40%	13803 (52%)	29 (0.1%)	4(0.2%)	6%(0.0%)	0 (0.0%)	0 (0.0%)
R-A-1/Exc-A1.1	6	22914 (92%)	6 (0.0%)	2(0.0%)	0(0.0%)	200 (0.8%)	1001 (4.0%)
R-A-1/Exc-A1.2	-5%	1300 (87%)	4 (0.3%)	1(0.1%)	0(0.0%)	0 (0.0%)	0 (0.0%)
R-A-1/Exc-A2.0	-35%	22383(61.7%)	113 (0.3%)	253(0.3%)	7(0.0%)	103 (0.3%)	420 (1.2%)
R-A-1/Exc-A2.1	40%	0 (0.0%)	0 (0.0%)	0(0.0%)	300(3.2%)	5800(61.1%)	3400(35.8%)
R-A-1/Exc-A2.2	-5%	1300(86.7%)	4 (0.3%)	1(0.1%)	0(0.0%)	0 (0.0%)	0 (0.0%)
R-B-1/Exc-B1	-44%	1172 (3.5%)	35 (0.1%)	82(0.3%)	1(0.0%)	0 (0.0%)	0 (0.0%)
R-B-1/Exc-B2	-44%	1144 (3.5%)	50 (0.2%)	93(0.3%)	2(0.0%)	1 (0.0%)	1 (0.0%)
R-B-1/Exc-B3	-44%	1143 (3.5%)	59 (0.2%)	93(0.3%)	2(0.0%)	0 (0.0%)	1 (0.0%)
R-B-2/Exc-B1	-4%	29676 (90%)	372 (1.1%)	144(0.4%)	5(0.2%)	1 (0.0%)	0 (0.0%)
R-B-2/Exc-B2	-4%	29631 (90%)	310 (0.9%)	234(0.7%)	11(0.0%)	6 (0.0%)	7 (0.0%)
R-B-2/Exc-B2.1	-23%	1751 (49%)	24 (0.7%)	2(0.1%)	0(0.0%)	0 (0.0%)	0 (0.0%)
R-B-2/Exc-B2.2	-24%	1752 (49%)	10 (0.3%)	11(0.3%)	1(0.0%)	1 (0.0%)	0 (0.0%)
R-B-2/Exc-B3	-4%	29319 (89%)	602(1.81%)	262(0.8%)	5(0.2%)	7 (0.0%)	7 (0.0%)

Table 3: Look up costs

Many of the results are very positive. For example, experiments Exc-A1.2 and Exc-B2.2 show that packets that are matched by rules unaffected by exceptions pay minimal performance penalties, with very few packets in this category requiring more than 10% lookup time.

As expected from the pre-experiment analysis, many packets that are affected by the exceptions attain worst case performance. For example, in experiment Exc-A2.1, there is a penalty of between 30% and 50% for almost all packets and analysis of the raw data show that the lookup path for many of the packets is the longest path in the BDD.

Direct comparison withe Cisco's implementation is difficult because of the different starting points, and the difficulty in finding published studies of the Cisco dynamic access list performance.

6.6 Other protocol costs

We also built a prototype systems for the protocol presented in Section 4. Tholo [19] built a Java prototype system with a Postgres database with the users' information. The implementation allowed users to make requests of the firewall and for the firewall to respond with information given to it by a pluggable dynamic access list mechanism.

Divac [6] built a different prototype system to measure the protocol costs. He designed it for an environment with a few hundred machines and approximately 1000 users. He showed that a simple server could handle up to 100 concurrent requests at less than 20ms per request (not including the BDD costs reported in the previous section).

Tholo and Divac's reports showed that the network protocol costs are minimal, especially considering that exception requests will be relatively infrequent.

7 Conclusion

This paper has proposed the use of dynamic access lists for IP filtering, arguing that the benefits of dynamic lists are increased flexibility and security.

A semantics for dynamic access lists was proposed and motivated, as well as a protocol for interaction between the users and the firewall. A prototype system was built and experimentation done using simulated data.

Memory costs of representing access and exceptions lists was very modest and it was shown that the largest experiment we did would require less than 500K of RAM.

The results show that the cost of making or undoing an exception on the firewall is less than a millisecond on a Pentium III running at 800MHz. The other protocol costs required for making an update are between 1 and 2ms depending on load, and so the latency experienced by the user would not be noticeable.

With respect to the cost of filtering, the experiments showed that while for many packets there would no significant penalty and possibly even an improvement in cost, for many real packets the lookup time would increase. Nevertheless the extra costs are not large and we can give a hard upper bound on these penalties. Here the benefits of the BDD representation are significant since there is a fixed upper limit for the cost of a lookup, no matter the size of the access or exception lists and the representation generally provides low variance for lookup. Thus we have a guaranteed upper bound on the cost of lookup. On an 800MHz Pentium III, this cost is at most 2.1μs. Whether this is acceptable would clearly depend on the circumstance; however, even on our very modest equipment, this would support performing almost 500 000 lookups per second. Faster equipment would support faster lookups.

Performance can also be improved by:

- the use of N-ary decision diagrams rather than binary decision diagrams. Attar [1] showed that this could improve performance significantly at modest increased memory use.
- the representation is ideal for shared memory multiprocessors and so one could expect almost linear speedup.
- the use of field programmable gate arrays (FPGAs) is

also worth examining. Previous work [15] showed that this technology is very effective for static lists. These techniques would need to be extended to deal with dynamic lists.

The biggest open issue is how exceptions can be made, and this needs future research. There are several possibilities. They can be made: (1) completely manually; (2) on-the-fly as they are needed using a simple IP address-based scheme; or (3) by an intelligent application-aware system. In particular, a policy-driven approach seems to be the most desirable [12, 18]. The protocol proposed here could provide an engine that would allow an efficient implementation of a policy-driven dynamic access list.

The overall conclusion of the paper is that increased security and flexibility can be provided through the use of dynamic access lists with relatively little performance penalty. The most important next step is to build a prototype for a working environment. We also wish to explore the use of this technology for mobile computing.

Acknowledgements: Pekka Pihlajasaari made a number of valuable comments on a draft of this material, including the suggestion of generalising from a priority-based to a group-based scheme. I also thank Brynn Andrew for his useful comments. The anonymous referees from an earlier conference version of this paper made useful comments that improved it. Thanks also to Pang Li, John Deneys, Sefako Tholo and Marko Divac, who were students we worked on various aspects of this work. The work was funded by NRF (GUN2050322) and Wits University Research Committee grants.

References

[1] A Attar. Performance Characteristics of BDD-based Packet Filters. MSc Research Report, University of the Witwatersrand, Johannesburg, School of Computer Science, 2002.

[2] S M Ballew. *Managing IP Networks with Cisco routers*. O'Reilly, October 1997.

[3] R Bryant. 'Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams'. *ACM Computing Surveys*, **24**(3):293–318, (September 1992).

[4] Cisco Systems Inc. Cisco IOS Lock and Key Security. CISCO White Paper, 1996.

[5] Cisco Systems Inc. Configuring IP Systems. Published at the Cisco web site, 1997. `http://www.cisco.com/univercd/cc/td/doc/product/software`.

[6] M Divac. Efficient use of dynamic access lists in firewalls. Honours Research Report, School of Computer Science, University of the Witwatersrand, Johannesburg, November 2003.

[7] P Gupta and M McKeown. 'Packet classification on multiple fields'. In *Proceedings of the SIGCOMM '99*, pp. 147–160. ACM, (1999).

[8] S Hazelhurst, A Attar, and R Sinnappan. 'Algorithms for improving the dependability of firewall and filter rule lists'. In *Workshop on the Dependability of IP Applications Platforms and Networks*, pp. 576–585, New York, (June 2000). IEEE Computer Society Press. In *Proceedings of the International Conference on Dependable Systems and Networks*.

[9] S Hazelhurst, A Fatti, and A Henwood. 'Binary Decision Diagram Representations of Firewall and Router Access Lists'. Technical Report TR-Wits-CS-1998-3, Department of Computer Science, University of the Witwatersrand, (October 1998). Proceedings of SAICSIT '98.

[10] K Knorr. 'Dynamic access control through Petri net workflows'. In *Proceedings of the Sixteenth Annual Computer Security Applications Conference*, pp. 159–167. IEEE, (2000).

[11] J McHenry, P Dowd, T Carrozzi, F Pellegrino, and W Cocks. 'An FPGA-based coprocessor for ATM firewalls'. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 30–39, (April 1997).

[12] P Naldurg and R Campbell. 'Dynamic Access Control: Preserving Safety and Trust for Network Defense Operations'. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, pp. 231–237, (2003).

[13] P Naldurg, R Campbell, and M Mickunas. 'Developing dynamic policies'. In *Proceedings of the DARPA Active Networks Conference and Exposition (DANCE '02)*, pp. 204–215, (2002).

[14] C Schuba and E Spafford. 'A Reference Model for Firewall Technology'. In *Proceedings of the Thirteenth Annual Computer Security Applications Conference*, (December 1997).

[15] R Sinnappan. A Reconfigurable Approach to TCP/IP Packet Filtering. MSc Research Report, School of Computer Science, University of the Witwatersrand, June 2001.

[16] F Somenzi. CUDD: CU decision diagram package. `http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html`, 2004.

[17] V Srinivasan. 'Fast address lookups using controlled prefix expansion'. *ACM Transactions on Computer Systems*, **17**(1):1–40, (February 1999).

[18] L Teo, G J Ahn, and Y Zheng. 'Dynamic and risk-aware network access management'. In *Proceedings of the Eighth ACM Symposium on Access Control*

Models and Technologies, pp. 217–230. ACM Press, (2003).

[19] S Tholo. Dynamic access lists. Honours Research Report, School of Computer Science, University of the Witwatersrand, November 2001.

[20] K Washburn and J Evans. *TCP/IP: running a successful network*. Addison-Wesley, Harlow, 1996.

A Representation of base list and exceptions: GBA

The previous section showed that any set of access list conditions can be represented as a boolean expression. This section describes how the base list and exceptions can be represented under GBA.

The following notation is used:

- ϕ_B: The boolean expression representing the base access list.

  If we are not considering exceptions, then a packet is accepted by the list if under the interpretation of variables given by the bits in the packet ϕ_B is true.

- ϕ_A: The boolean expression representing the access list together with exceptions. Where there are no exceptions, $\phi_A = \phi_B$.

- $\mathcal{E}(\mathcal{B}, i)$: This condition states what exceptions are permissible in exception list i. Formally, for each group i, the condition is the disjunction of the *deny* rules that are labelled with i or a supergroup of i all conjuncted with the conjunction of the negation of the other rules. So, if we instantiate the variables in $\mathcal{E}(\mathcal{B}, i)$ with values from the bits in a packet header, we find out whether an exception in list i can over-ride *reject* rules in the base list in order to allow this packet.

- ϵ_j: the expression representing exception list j.

  Note that $\epsilon_j \wedge \mathcal{E}(\mathcal{B}, j)$ gives us the *effective* exception list for group j – the requests asked for permitted by the deny rules in the base list.

Under the GBA semantics given in Section 3.4 (on page 4) the boolean expression that represents a base access list and n exception lists is:

$$\phi_A \stackrel{\text{def}}{=} \phi_B \ \vee \ (\vee_{i=0}^{n-1} \mathcal{E}(\mathcal{B}, i) \wedge \epsilon_i) \qquad (2)$$

The firewall keeps the following information:

- ϕ_B, the representation of the base list;

- For each update request u, ϕ_u, the boolean expression representing the exception, the ID of the request, expiry time and the group of the originator of the request;

  The update requests are stored in a manner so that they can efficiently be accessed by ID number, by expiry time, and by group number.

- For each i, $\mathcal{E}(\mathcal{B}, i)$, the exceptions not permitted by the base list for group i;

- For each i, ϵ_i, the exceptions requested by group i;