Techniques and Algorithms for Access Control List Optimization

Ibrahim M. Al Abdulmohsin

Communications Engineering & Technical Support Department, Saudi Aramco, Dhahran 31311

***Abstract* — Access control lists are core features of today's internetwork routers. They serve several purposes, most notably in filtering network traffic and securing critical networked resources. However, the addition of access control lists increases packet latency due to the overhead of extra computations involved. This paper presents simple techniques and algorithms for optimizing access control lists that can reduce significantly expected packet latencies without sacrificing security requirements. These techniques and algorithms can be implemented either fully or partially, both online and offline, based on the amount of overhead allowed. It also outlines analytically and statistically where and why the greatest bulk of optimization lies.**

***Index Terms* — Communication system operation and management, communication system security, access control, optimization methods.**

I. Introduction

Routers are the most vital elements of today's internetworks. They operate at layer three of the OSI reference model, where they determine the optimal path to distant networks using routing protocols and, subsequently, route packets from source to destination through those optimal paths. Another important function of routers, however, is to determine if packets are authorized by network administrators to reach their intended destinations in the first place. To do this, routers check packets against an access control list (ACL), defined by a network administrator, that specifies which types of network traffic should be permitted and which types should be denied. The role of ACLs can also be extended to include filtering route advertisements and enforcing policies such as network address translation (NAT) and traffic shaping [1].

With the rapid growth of today's internetworks, ACLs have become increasingly important to network administrators. This is in part due to the increase in cyber threats and attacks on the one hand, where access control technologies play the primary defense mechanism, and the demand for cost-effective network policies that optimize network performance with minimal costs on the other hand. ACLs do not come at a zero cost, however. They add a computational delay, which contributes to end-to-end packet latency. An optimal access control list is defined to be the access list that satisfies security requirements while involving the least amount of computational delay. Optimizing ACLs can be implemented by both reducing their size and changing the order of their rules.

The subject of access control list optimization has been awarded a strong interest in the research community. The first real attempt at access control list optimization came from Cisco ACL Optimizer [2]. Although Cisco's application addresses a wide range of optimization scenarios, it has three major drawbacks. First, Cisco ACL Optimizer combines only contiguous maskable rules, and ignores non-maskable and yet still combinable rules, even though combinable rules that are not maskable are common in today's ACLs. Second, Cisco application does not predict hit probabilities for rules with no hit counts present, e.g. in the case of a new ACL. This implies a random arrangement of rules with equal hit counts even if their predicted hit probabilities are different. Third, as has been previously noted by Grout (2006), Cisco ACL Optimizer assumes a constant rule latency, which may or may not be accurate depending on implementation [3]. On the other hand, other research papers only considered access control list optimization in limited scenarios. For instance, Al Shaer (2004) only addresses anomalies in ACLs, while Bukhatwa (2004) ignores rules' dependencies [3]-[5].

II. Definitions

A. Rule

A typical rule in Cisco extended ACL format is shown in figure 1. As shown in the figure, a rule is comprised of six fields: (1) *action (act)*, which could be either permit or deny, (2) *protocol (prtcl)*, such as IP, TCP, UDP, ICMP ..etc, (3) *source address range (sa)*, in the form of

```
permit  ip  10.1.1.0  0.0.0.255  host  10.2.2.1  eq http
```

Fig. 1. A typical rule in Cisco extended ACL format.

an IP address and a wild card mask, (4) *source port range (sp)*, (5) *destination address range (da)*, and (6) *destination port range (dp)*. In the case of ICMP, both source and destination port ranges are replaced with a flag such as echo-request, or echo-reply. In mathematical terms, a rule R is a 6-tuple object, R=(*act*, *prtcl*, *sa*, *sp*, *da*, *dp*). In this paper, we denote *Ri.act* to mean the action field of the ith rule, *Ri.prtcl* to mean the protocol field of the ith rule … and so on.

B. Access Control List and Executed Rules

An access control List (*ACL*) is a set of rules that are executed sequentially from top to bottom. In other words, if *every* field in a packet matches the corresponding field in a rule *Ri*, the router will take the action stated in *Ri.act*, and ignore all subsequent rules. $Ri\ \varepsilon\ p_k$, read *Ri* executed for the *k*th packet, indicates that in the specified ACL, the action field in *Ri* determines whether the packet p_k will be permitted or denied.

C. Rule Dependency and Superset Rules

Two rules are dependent if there is at least one packet p_k such that $Ri\ \varepsilon\ p_k$ when $i<j$ and $Rj\ \varepsilon\ p_k$ if $j<i$. In this paper, we write $Ri\ \Delta\ Rj$ to indicate there is a dependency between *Ri* and *Rj*. In mathematical terms,

$$Ri\ \Delta\ Rj \leftrightarrow (Rj.prtcl \cap Ri.prtcl \neq \phi) \wedge (Rj.sa \cap Ri.sa \neq \phi) \wedge (Rj.sp \cap Ri.sp \neq \phi) \wedge (Rj.da \cap Ri.da \neq \phi) \wedge (Rj.dp \cap Ri.dp \neq \phi)$$

A rule Ri is said to be a superset to rule Rj if every field in Ri is a superset or equal to the corresponding field in Rj. In mathematical terms,

$$Ri \supseteq Rj \leftrightarrow (Ri.prtcl \supseteq Rj.prtcl) \wedge (Ri.sa \supseteq Rj.sa) \wedge (Ri.sp \supseteq Rj.sp) \wedge (Ri.da \supseteq Rj.da) \wedge (Ri.dp \supseteq Rj.dp)$$

D. Shadowed and Covered Rules

Shadowed rules are rules that will never be executed because of a preceding rule whose fields are all supersets or equal to the corresponding fields in the shadowed rule. In this paper, $Ri \triangleright Rj$ implies that *Rj* is shadowed by *Ri*. To make the notation easier to remember, *Ri* is on the wider side of the triangle (i.e. a superset rule) and on the left (i.e. comes first). Thus, *Ri* shadows *Rj*.

$$Ri \triangleright Rj \leftrightarrow (i < j) \wedge Ri \supseteq Rj$$

Covered rules are rules that can be safely removed because a subsequent more general can still satisfy security requirements. This happens if both actions in the two rules are similar, while all fields in the covered rule are subsets or equal to the corresponding fields in the subsequent more general rule. $Ri \triangleleft Rj$ implies that *Ri* is covered by *Rj*. In this notation, *Rj* is on the wider side of the triangle (i.e. a superset rule) and on the right (i.e. comes next). Thus, *Rj* covers *Ri*.

$$Ri \triangleleft Rj \leftrightarrow (i < j) \wedge (Rj.act = Ri.act) \wedge Rj \supseteq Ri \wedge \left(\forall k \mid i < k < j : Not(Ri \Delta Rk) \vee (Ri.act = Rk.act)\right)$$

E. Hit Probability

Hit probability of a rule *Ri*, *h(i)*, is the probability that a packet will traverse all preceding rules without a match and matches all fields in *Ri*. In other words, *h(i)* is the probability that $Ri\ \varepsilon\ p$, where *p* is any random packet. Hit probabilities play a significant role in ACL optimization.

F. Expected Packet Latency

Rule Latency, *RL*, of a rule *Ri* is the time taken to process *Ri*, which could be fixed or different from one rule to another according to implementation. Rule Packet Latency, *RPL(i)*, is the latency a packet goes through when it is executed by rule *Ri*. That is, it is the sum of all previous rule latencies, as given in equation 1.

$$RPL(i) = \sum_0^i RL(k) \quad (1)$$

, where R_0 is the first rule in the ACL. *Expected Packet Latency (EPL)* is the standard by which ACL optimization is measured. The larger the decline in EPL attained, the better the optimization. EPL can be calculated using equation 2 below.

$$EPL = \sum_0^{n-1} h(i) RPL(i) \quad (2)$$

, where *n* is the total number of rules in the ACL.

III. ACL Optimization solutions

Finding an optimal ACL is an NP-complete problem, meaning it cannot be solved in polynomial time [3]. Thus, the only alternative is to build heuristic algorithms that yield excellent results in polynomial time, which is the subject of this paper. We will first outline optimization scenarios and provide their algorithms. After that, we will show analytically and empirically where and why the greatest bulk of optimization lies.

A. ACL Optimization Application

The techniques and algorithms presented next have been implemented in C++. The application, called ACLO for ACL Optimization, was applied to around 100 existing ACLs written by average network administrators and was found to yield, on average, an 80% reduction in EPL and a 40% reduction in ACL size. ACLO is command-line-based and stores both the new optimized ACL and a detailed report of how it has been optimized in two separate files. The purpose of the report is to provide network administrators with a detailed description of why the old ACL needed to be optimized, and to identify to him/her the configuration errors s/he has made in the old ACL.

Table 1 displays the 12 fields used in ACLO. WCM is wild card mask. It specifies which bits are examined and which are ignored. ICMP flags are treated as pseudo ports in the application and given pseudo port numbers all larger than 100,000 to ensure they will not conflict with *real* port numbers in other rules. Hit Counts, on the other hand, are the actual hit counts available on deployed logging-enabled ACLs. In addition to actual hit counts, the application also adds a *Hit Counts Prediction Factor* for every rule. This factor is automatically calculated by the application based on the values of the other fields in a given rule. The value is, then, normalized to a ratio over one so that it can only be

Action	Protocol	Src IP	Src WCM
Dst IP	Dst WCM	Src Port Rng	Dst Port Rng
Hit Counts + Prediction Factor	Rule Latency	Rule Up Bound	Rule Bottom Bound

Tab. 1. The fields used in ACLO optimization algorithms.

significant for rules with no actual hit counts present or for rules that share the same hit counts. The purpose of the prediction factor, as the name implies, is to predict future hit counts and optimize accordingly. For instance, rules that specify IP protocol are more likely to be executed than rules that specify ICMP traffic. Similarly, rules that specify traffic between different subnets are more likely to match than rules that specify traffic between two individual hosts. The relative weight of each field was determined empirically by examining real-life traffic and hit counts of deployed ACLs.

Rule Latencies (RLs) and Boundaries are the last three fields used in the application. RLs specify the relative difference in execution time between different rules. Obviously, whether rule latencies are different from one rule to another depends on implementation. However, it is likely that rules that specify layer 4 header fields such as TCP/UDP port number require more processing time than standard rules, which only specify layer three fields. Thus, for the purpose of generalizing the problem of ACL optimization, rule latencies are also considered in this paper. It is important to note that the actual values of rule latencies are not significant, since we are not attempting to calculate the real expected packet latency in a given ACL. What is significant, however, is the ratio between those values since it does influence how rules will be reordered, and, therefore, how the ACL will be optimized. Rule Up and Bottom Boundaries define the degree of mobility of a rule in a given ACL. In other words, it is the boundaries between which a rule can be safely relocated without changing ACL semantics. Obviously, when two rules need to be swapped, the new locations of *both* rules need to be within their own respective boundaries. A more thorough discussion of how these fields are determined and how they are used in the algorithms will be provided in the next section.

Aside from the database, ACLO uses five files. The first file contains the old ACL, while the second file contains the new optimized ACL. In addition, the third file has all well-known ports in both text and numerical format. The fourth file contains ICMP flags and pseudo numbers that will be used during optimization. Lastly, the fifth file contains a detailed report of how the old ACL has been optimized on a rule-by-rule basis.

B. Optimization Scenarios

In the subsequent algorithms, we assume that the first rule is R_0 and the last rule, $R_{ACLsize-1}$, is the implicit deny all statement. Before going into the algorithms and optimization results, we first need to show how rules' boundaries are determined, since this algorithm will be executed every time a change is made to the ACL. The algorithm is shown in figure 3 below, with a time complexity of $O(n^2)$.

```
Update_Rule_Boundary (Ri)
k = i
while (((k < ACLsize )AND ( k >= 0)) AND
((Ri.act = Rk.act) OR NOT(Ri Δ Rk)))
    k = k - 1
Ri.UpBound = k + 1
k = i
while (((k < ACLsize) AND (k >= 0)) AND
((Ri.act = Rk.act) OR NOT(Ri Δ Rk)))
    k = k + 1
Ri.BottomBound = k - 1
R0.UpBound = 0
RACLsize-1.BottomBound = ACLsize - 1
```

Fig. 3. Rule Boundary Update Algorithm

a. Removing Shadowed Rules

Shadowed rules are never executed and, thus, they can be safely removed. Removing those rules not only reduces the size of the ACL, but also improves RPL for all subsequent rules. Consequently, removing shadowed rules improves EPL for the entire ACL. Figure 4 shows the algorithm used in ACLO. The time complexity of this algorithm is $O(n^4)$.

b. Removing Covered Rules

Although, unlike shadowed rules, covered rules could be executed, they can still be safely removed because a subsequent more general rule can still satisfy security requirements. Removing these rules will reduce the size of the ACL and improve EPL. The algorithm for optimizing an ACL in this scenario is shown in figure 5. The time complexity of this algorithm is also $O(n^4)$.

```
Remove_Shadowed_Rules (ACL)
i = 0
while ( i < ACLsize -1 )
    j = i + 1
    while ( j < ACLsize)
        if (Ri ⊇ Rj)
            report action
            remove (Ri) & Decrement ACLsize
            j = j -1
            Update All Rules Boundaries ()
        j = j + 1
    i = i + 1
```

Fig. 4. Removing shadowed rules algorithm.

c. Combining Rules

While finding an exact solution for removing shadowed and covered rules is relatively a simple task, discovering the best possible way of combing rules is a lot more difficult. Because we are looking for reasonably fast heuristic algorithms, we decided to examine rules on a pair by pair basis. In other words, the algorithm will determine if two rules could be combined together into a more general rule regardless of what the other rules in the ACL are. It is true that by looking at all the rules,

```
Remove_Covered_Rules (ACL)
i = 0
while ( i < ACLsize -1 )
    j = i + 1
    while (( acti=actj) or NOT(Ri Δ Rj))
        if (Rj ⊇ Ri)
             report action
             remove (Ri) & Decrement ACLsize
             Update All Rules Boundaries ()
             Break () "this statement breaks
                       the while loop"
        j = j + 1
    i = i + 1
```

Fig. 5. Removing covered rules algorithm.

better optimization could be inferred than by merely looking at each pair on its own, but it would also be impractically time consuming. The algorithm shown in figure 6 combines rules by manipulating the wild card masks even if those rules are not contiguous and not maskable. Because a low-level description of the algorithm would be too long for the scope of this paper, a high level description is given instead. The time complexity of this algorithm is $O(n^5)$.

```
Combine_Rules (ACL)
Do {
Repeat = 0, i = 1
while (i < ACLsize)
  j = i + 1
  while (j <= ACLsize)
     if((Ri.prtcl = Rj.prtcl) and
     (Ri.sp = Rj.sp)and(Ri.dp = Rj.dp))
       Is there exactly a one-bit
       difference in both source IP and
       destination IP?
       If (Yes)
         If difference in sourceIP set S
         to one. Otherwise, set S to
         zero.
         K=position of the bit
       If (S=0)
            If (j between Ri boundaries)
              Rj.dstWCMBit[k]=1
              Rj.HitCounts+=Ri.HitCounts
              Report action
              remove (Ri) & Decrement ACLsize
              Repeat = 1
              Update all rules boundaries()
            If (j Not between Ri boundaries)
                 AND (i between Rj boundaries)
              Ri.dstWCMBit[k]=1
              Ri.HitCounts+=Rj.HitCounts
              Report action
              remove (Ri) & Decrement ACLsize
              Repeat = 1
              Update all rules boundaries()
         If (S=1)
           //Similar but for srcWCMBit//
    j = j + 1
  i = i + 1
} while (repeat > 0)
```

Fig. 6. Combining Rules Algorithm

d. Hits Optimizer

The Hits Optimizer part of the application reorders the rules in an ACL based on *effective hit probabilities* in order to minimize EPL. It accomplishes this by swapping rules only if their new positions are within their respective boundaries and effective hit probability of the first rule is less than effective hit probability of the next rule.

Effective hit probability is determined by three factors. The first factor is the actual hit counts, since it represents real-life traffic patterns and probabilities. Effective hit probability of a rule is linearly related to hit counts. The second factor is the hit counts prediction factor. As has been discussed earlier, this factor is determined by the values of the other fields in a given rule, and it is normalized to a fraction between zero and one to ensure that the prediction factor is only significant when no actual hit counts are present or for rules with equal hit counts. The details of how the prediction factor is calculated should be determined by the application user, and are outside the scope of this paper.

The third factor in the calculation of effective hit probabilities is rule latencies. How rule latencies are handled in ACL optimization can be shown both graphically and mathematically. Graphically speaking, because ACLs are executed sequentially from top to bottom, rules with different rule latencies, such as the ones shown in table 2, can be duplicated to reflect their RL while assuming a fixed RL for each rule, as shown in table 3. Obviously, hit probability will have to be distributed evenly among the duplicate rules. Thus, effective hit probability is directly related to *hitCounts/RL*. Mathematically speaking, consider the case where there are two rules *Ri* and *Rj*, with different rule latencies. The EPL for the two rules is given in equation 3 if $i < j$. The minimum of the two EPLs, when $i < j$ or when $j < i$, determines if *Ri* should be placed before *Rj* or vice versa. It is clear from equation 3 that *hitCounts/RL* again determines if the two rules should be swapped or not.

$$EPL(i < j) = h(i)RL_i + h(j)RL_j + h(j)RL_i \quad (3)$$

To sum up, Hits Optimizer, uses equation 4 for calculating effective hit probabilities

$$EHP \propto (HitCounts + PF)/RL \quad (4)$$

, where EHP is effective hit probability, RL is rule latency, and PF is prediction factor.

Rule	Hit Counts	RL
Permit ip any host 10.1.1.1	5	1
Permit ip any host 10.1.1.2 eq 80	20	2
Deny ip any any	100	1

Tab. 2. ACL with hit counts and rule latencies

C. Optimization Results

When ACLO application was applied to around 100 ACLs written by average network administrators, it was found that the four simple procedures could optimize EPL for typical ACLs by more than 80%, while ACL size is reduced on average by 40%. Table 4 shows how much, on average, each procedure contributes to overall

Rule	Hit Counts	RL
Permit ip any host 10.1.1.1	5	1
Permit ip any host 10.1.1.2 eq 80	10	1
Permit ip any host 10.1.1.2 eq 80	10	1
Deny ip any any	100	1

Tab. 3. ACL with effective hit counts when rule latencies are assumed fixed.

EPL optimization when a single procedure is applied at a time. Obviously, Rules Combining procedure and Hits Optimizer contribute the most to ACL overall optimization.

There are several reasons why Hits Optimizer and Rules Combining procedures contribute the most to ACL optimization. First, shadowed and covered rules are not only relatively easy to detect by network administrators, they are also easily detected by end users upon writing

Reduction	Shadowed Rules Removal	Covered Rules Removal	Rules Combining Procedure	Hits Optimizer
EPL	1%	10%	25%	77%
Size	2.5%	5%	37%	0%

Tab. 4. Average optimization by each procedure to typical ACLs.

his/her access requirements. These *anomalies* do not require sophisticated technical knowledge to discover. Rules Combining procedure, on the contrary, involves an excellent background in binary arithmetic and Boolean algebra. In addition, they are harder to detect by the network administrator, and are not normally considered upon writing access requirements by the end user. Furthermore, it is a daunting task for network administrators to study network traffic and predict hit probability in order to optimize ACLs accordingly. As a result, an automated approach, such as ACLO application, yields substantial improvement for typical ACLs through these two procedures.

C. Timing Consideration

The time complexity of the application is $O(n^5)$. This is relatively acceptable for small to medium size ACLs as shown in figure 7. Also, it is noteworthy to keep in mind that the application is a one-time-pass algorithm. In other words, once an ACL is written and hit counts are available that sufficiently reflect real-life traffic, the ACL needs to be optimized only once either online or offline. It does not need to be re-optimized in the future unless changes are made to it.

Nevertheless, the algorithms presented so far are highly customizable in terms of time versus efficiency. If, for instance, the time complexities of the algorithms are not acceptable for online optimization, the algorithms can be slightly altered to reduce the time complexity of the application at the expense of a reduction in efficiency. For instance, the Rules Combining procedure, which is $O(n^5)$, is set to repeat the entire process whenever two rules were successfully

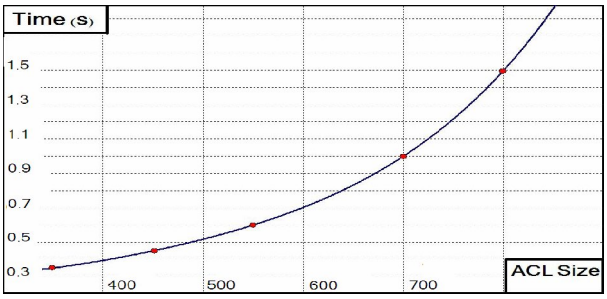

Fig. 7. The time ACLO takes plotted against ACL size.

combined. This, however, could be changed by setting a maximum of two or three repetitions for the entire algorithm, which would yield excellent optimization with a significant reduction in time. In the latter case, the time complexity of the algorithm and the application as a whole becomes $O(n^4)$.

IV. Conclusion

An optimal access control list is an access list that satisfies security requirements with the least amount of processing overhead. In this paper, we have presented several techniques and algorithms for access control list optimization. Some of these algorithms look for rules that can be safely removed, such as shadowed and covered rules, and rules that can be combined in order to reduce the size of ACLs and, subsequently, reduce expected packet latency. Other algorithms reorder the rules in an ACL based on three factors: actual hit counts, hit counts prediction factor, and rule latencies. It was found empirically that Hits Optimizer and Rules Combining procedures yield the greatest bulk of optimization since they are harder to handle manually by average network administrators. The algorithms can be easily customized, where time is reduced at the expense of efficiency, and can be implemented partially or fully, both online and offline.

References

[1] A. Velte and T. Velte. “Cisco: A Beginner’s Guide”, McGraw-Hill Inc. 3rd edition (2004).

[2] Access Control Lists, Cisco Systems, USA, (http://www.cisco.com/univercd/cc/td/doc/product/software/ios113ed/113ed_cr/secur_c/scprt3/scacls.htm).

[3] V. Grout, J. McGinn, and J.Davies. “Real-Time Optimisation of Access Control Lists for Efficient Internet Packet Filtering”, Journal of Heuristics, Vol. 12, 2006.

[4] E. Al-Shaer and H. Hamed. “Firewall Policy Advisor for Anomaly Detection and Rule Editing.” *IEEE/IFIP Integrated Management Conference (IM’2003)*, March 2003.

[5] Bukhatwa, F., (2004) High Cost Elimination Method for Best Class Permutation in Access Lists, *ICWI 2004*, pp287-294