A New Method for Creating Efficient Security Policies in Virtual Private Network

Mohammad Mehdi Gilanian Sadeghi[1], Borhanuddin Mohd Ali[1], Hossein Pedram[2], Mehdi Deghan[2], and Masoud Sabaei[2]

[1] Faculty of Engineering, Universiti Putra Malaysia, Malaysia
mmgsadeghi@yahoo.com, borhan@eng.upm.edu.my
[2] Faculty of Computer Engineering and IT, Amirkabir University, Iran
{pedram,dehghan,sabaei}@ce.aut.ac.ir

Abstract. One of the most important protocols for implementing tunnels in order to take action of secure virtual private network is IPsec protocol. IPsec policies are used widely in order to limit access to information in security gateways or firewalls. The security treatment, namely (Deny, Allow or Encrypt) is done for outbound as well as inbound traffic by security policies. It is so important that they adjust properly. The current methods for security policies creation as seen in given security requirements are not efficient enough i.e. there are much more created policies than requirements. In this paper, we define a new method to decrease adopted security policies for a specific set of security requirements without any undesirable effect. Our measurement shows that security policies creation will be improved efficiently, and their updating time will be decreased.

Keywords: IPsec policies, security policy, security requirement, virtual private network.

1 Introduction

Virtual Private Network (VPN) is a group of techniques which makes implementation of an organization's private network on public and dynamic and scalable Internet. Many protocols are used for the creation of virtual private networks (VPN) [1] and IPsec protocol is one of the best for making security. IPsec policies [2] which are placed on Security Policy Database (SPD) and Security Association Database (SAD) consist of two parts, namely condition and action. The part of condition considers any header field in IP packet for each policy. Generally, the part of action includes three modes: Deny, Allow and IPsec_action. By mixing the properties of condition and action, the policies would be represented. For example Packet with A source and B destination addresses would pass in src=A, dst=B->Allow.

An inappropriate policy would possibly cause communication deficiency or serious security breach [3, 4]. Some problems are due to manager's carelessness for policies adjustment, while some others might arise from interactions that cannot be easily detected even with careful and experienced administrators. In the following, we study one scenario of policies problem [5].

E. Bertino and J.B.D. Joshi (Eds.): CollaborateCom 2008, LNICST 10, pp. 663–678, 2009.
© ICST Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering 2009

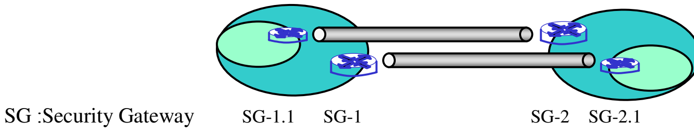

Fig. 1. Example of Policy Problem

In figure 1, there are financial department 1.1 in location 1 and financial department 2.1 in location 2. Suppose that each department has a unique security gateway as well as security policies in order to protect data. For example, department 1.1 decides to encrypt all traffic from 1.1 to 2 with a tunnel SG-1.1 to SG-2. At the same time, the administrator for location 1 decides that whole traffic from location 1 to financial department 2.1 must be encrypted through a tunnel from SG-1 to SG-2.1 because of its importance. Thus, traffic from SG-1.1 to SG-2.1 would be covered using two separate policies.

But such a scenario has a problem, i.e. with this configuration, a new header would be encapsulated to packet in SG-1.1, and then another new header would be encapsulated to these data in SG-1 again, finally they will be sent to SG-2.1 destination. When packets arrived at SG-2.1, the new header in second step is decapsulated, so it is clear to determine SG-2 destination. Hence, SG-2.1 returns traffic to SG-2. Finally, the new header in the first step is decapsulated in SG-2 and traffic is sent to its real destination. Although our aim is to encrypt the traffic from SG-2 to SG-2.1, there are some interactions between tunnels, so the real traffic will be sent from SG-2 to SG-2.1 without any encryption.

The disadvantage of Figure 1 is solved in Figure 2 by choosing appropriate policy. In Figure 2, the traffic is encrypted in SG-1.1. Again, it is encrypted in SG-1. The destination for both tunnels is SG-2. They are decrypted there and changed into plain text and then the traffic is encrypted in SG-2 and sent to the next tunnel between SG-2 and SG-2.1. If SG-2 is trusted in the second requirement, then the three-tunnel plan can well satisfy both requirements.

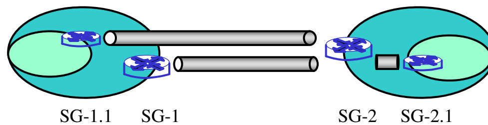

Fig. 2. IPsec policies to satisfy Security Requirements

In policy hierarchy [6], requirements (high level policy) are objectives while implementation policies (low level policies) are specific plans to meet the objectives. Each requirement may be met by a set of implementation policy. Thus, the process of policy creation is to meet the demands of targets in such a way that it transforms the requirements to implementation policies. So, it is necessary to choose policies very carefully to meet all requirements.

2 Related Works

A working group on IP security policy (IPSP) formed in IETF has addressed complex IPsec security policy problems. The requirements that we define in this paper basically follow the IPSP requirement draft [7]. There is no complete solution found up to now to meet all the objectives specified in the requirement and no interference between policies has been considered. Moreover, many different studies have been done on security policies [8, 9, 10] to standardize the policy specifications, which currently address only low-level policy specifications. Similarly, other proposed drafts regarding policy information base [11, 12] and data model [7] have also focused on low-level policies while [13] analyzed policy management problem and introduced global policies. [14]-[19] focus on finding an automated approach of IPsec policy configuration. In these two papers [18,19], "Filtering rules" refer to security requirements, while "IPsec policies" represent security policies. [20] presents an extension to Ordered-Split algorithm that analyzes the traffic probability together with the original algorithm to optimize the solution. Other research focuses on the policy management alone. [21] demonstrates an algorithm for distributing policies among a number of management stations, while [22] discusses an approach to conflict handling relying on a priori models. In order to avoid firewall policy anomalies, [23] proposes an approach to perform symbolic model checking of the firewall configurations for all possible IP packets and along all possible data paths to provide a complete solution.

The necessity of separation between high-level policies and low-level requirements has been studied in [24, 6]. There are two levels of policy hierarchy described in [14, 15] and this led to high level policies and low level security requirements that we use in this research.

In [5] a method to create appropriate policies in order to meet their requirements automatically has been presented. They guarantee the policy accuracy plus the interference among policies as much as possible. But, this would not be useful any longer because the number of security policies is more than security requirements. In this paper, we present a method to decrease the number of created security policies that will lead to increased efficiency as well as saving updating time for creating those policies.

In section 2, we review related works. We discuss current methods for creating policies in section 3. In section 4 a method is proposed to solve the problems of current security policies and then analyze them. Our research conclusion is finally in section 5.

3 Creating Security Policies to Meet Security Requirements

In total, there are four main security requirements for IPsec policies [14, 15].

✓ **Access control requirement (ACR):** One fundamental function of security is to conduct access control that is to restrict access only to trusted traffic. A simple way to specify an ACR is: flow id.->deny | allow

- ✓ **Security Coverage Requirements (SCR):** using security functions for the whole area (between two locations) to prevent traffic from illegal access during data transfer. A simple way to specify a SCR is to protect traffic from *"from"* to *"to"* by a security function with certain strength: *flow id.-> protect (sec_function, strength, from, to, trusted_nodes)* The requirement is satisfied only if the traffic is with sufficient security protection on every link and node in protection area from *"from"* to *"to"*, except that the trusted nodes can be left uncovered by the function.
- ✓ **Content Access Requirement (CAR)**: Some nodes like firewalls with an intrusion detection system (IDS) would need to examine traffic content in order to decide how to manage passing traffic characteristic. CAR can be expressed as denying certain security function to prevent the nodes from accessing certain traffic. This is expressed as follows: *flow id.-> deny_sec (sec_function, access_nodes)*. The requirement is satisfied only if the traffic is not secured with the function *"sec_function"* on any node specified in *"access_nodes"*.
- ✓ **Security Association Requirement (SAR)**: Security Association (SA) [2] needs to perform encryption as well as authentication. There might be needs to specify that some nodes desire or not desire to set up SA of certain security function with some other nodes because of public key availability, capability match/mismatch etc. A simple way to specify a SAR could be: *flow id.->deny_SA (SA_peer1,SA_peer2, sec_function)*. The requirement is satisfied only if none of the nodes specified in *"SA_peer1"* forms SA with any of nodes specified in *"SA_peer2"* with function *"sec_function"*.

Some security requirement characteristics have been mentioned above. Characteristics of security policies; they include *Deny, Allow, IPsec_action (sec_ port, algorithm, mode, from A, to A)* for selected traffic. s*ec_port* determines *AH* or *ESP* protocols. *Algorithm* determines all possible algorithms of Internet Key Exchange (IKE). *Mode* determines whether it is transfer or tunnel mode. *From A to A* determines two factors to create security Association. Policies perform security operations on passing traffic.

In [5], **Bundle Approach** was presented to create security policies, but the major problem is non efficiency. In other words the rate of security policies would be increased more than security requirements. In this way, the entire traffic would be divided into some sub set of separate Bundles. Each Bundle includes one set of security requirements. For one particular bundle, the condition part of policies contains bundle selectors and action part contains appropriate security actions to satisfy all requirements for the bundle. From Figure 3, there are a set of three security requirements [5]:

Three_Reqs = { Req1 (src=1.*, dst=2.*→ weak, ENC, 1.*, 2.*), Req2 (src=1.1.*, dst=2.*→ Strong, ENC, 1.1.*, 2.*), Req3 (src=1.*, dst=2.1.*→ Strong, AUTH,1.*, 2.1.*)}

They include F1, F2 and F3 filters (they can be 2-tuple, 3-tuple or 5-tuple). Also SG-1 and SG-2 security gateways are assumed as CAR nodes. Black, Gray and white lines in Figure 3 are determined for Req1, Req2 and Req3 respectively.

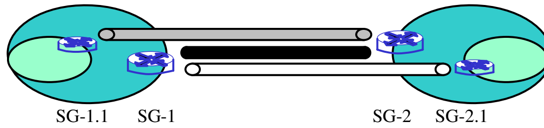

Fig. 3. Three_Reqs Example

The problem of using Bundle Approach is solved in two phases. In the first Phase, the entire traffic flow will be divided into separate bundles and then a set of requirements will be calculated for each of them. In the second phase for every bundle, appropriate policies (for action part) are selected according to the given requirements. In addition, the bundle's selectors will be calculated. Next, we discuss both the calculation selectors for each Bundle and the action part for creating the policies as follows:

3.1 Selection Decision

A Relationship Tree is used for calculating the requirements as well as their order. Figure 4 shows the relationship among **three_Reqs** filters. Filter F2 with Req2 has been used as a child of Filter F1 with Req1 in relationship tree, because it is contained by F1. Filter F3 with Req3 has overlap with F2. Thus, a new filter called F4 with Req3 generate which is combined by two filters F2 and F3 and inset as child of F2 since F4 is contained by F2. F3 would be inserted to the relationship tree as a child of F1, because it is contained by F1.

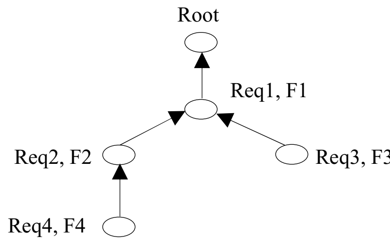

Fig. 4. Relationship Tree

As a result, it is clear that four policy sets are generated namely: {policy_set 1, policy_set2, policy_set3, policy_set4} to satisfy requirement set {{Req1}, {Req1, Req2}, {Req1, Req3} and {Req1, Req2, Req3}, respectively. The filters of the policy sets are {F1, F2, F3, F4}. So, they include the following addresses, they are four bundles {(1.1.*.*, 2.1.*), (1.*-1.1.*, 2.1.*), (1.1.*, 2.*-2.1.*), (1.*-1.1.*, 2.*-2.1.*)}.

3.2 Policies Decision

In this stage, the aim is to decide how to use a set of appropriate policies according to requirements. If the inner most paths to carry packets is called **primary tunnels**, and

the corresponding SA is called **primary SA**, then the primary tunnels need to be chained together across an area to provide security coverage for the area.

Our aim of tunnel creation is to satisfy Security Coverage Requirement (SCR), Content Access Requirement (CAR) and Security Association Requirement (SAR). Eligible Security Associations are confirmed by CAR and SAR. SCR has two major functions: encryption and authentication. A primary tunnel can only provide the coverage for just one function. The rest of the tunnels may be placed on top of primary tunnel to provide the necessary coverage for the other functions, so they are called *Secondary Security Association*. Finding Eligible SA is solvable using Algorithms and Graphs. To find eligible primary SAs, we need three graphs: **ENC secondary graph**, **AUTH secondary graph** and **Primary graph**, in which ENC and AUTH graphs are needed to determine secondary SA paths [5].

As discussed in previous section, there are four bundles for Three_Reqs example. If SG-1 and SG-2 security gateways are considered as Access Control Requirement (ACR) except the three Security Coverage Requirements, we can start finding policies using filter F1which contains Req1. Req1 has only encryption so it does not need AUTH secondary graph. Primary graph and ENC secondary graph are based on Figure 5. The edge 2-3 is called E1 that identify encryption for Req1.

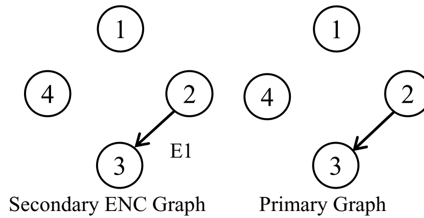

Fig. 5. Graphs of Filter F1

Then policies of Filter F2 with Req1 and Req2 are calculated. In this case these requirements contain only encryption so they do not need AUTH secondary graph. The only graph which can satisfy Req1 and Req2 is according to Figure 6.a. The edges are labeled as E2 because E2 is the stronger encryption Algorithm. (There are two encryption security coverage called E1 and E2 between two nodes 2 and 3 but E2 is stronger than E1, so E2 is selected). Node 2 has been chosen as a Content Access Requirement; therefore we would remove the edge between nodes 1-3 in order to break down the tunnel in node 2. All graphs will be formed like Figure 6.b after removing Edge.

Next we calculate policies of filter F3 with Req1 and Req3 according to Figure 7.a. Both AUTH and ENC secondary graph are necessary in this case because Req1 acts as encryption, and Req3 acts as authentication. Since there is no necessity for security coverage for ENC between nodes 3-4, there would be no edge in ENC secondary graph for these nodes. Node 3 has Content Access Requirement; consequently all passed association from the node will be removed. Thus, the processed graphs are formed as in Figure 7.b.

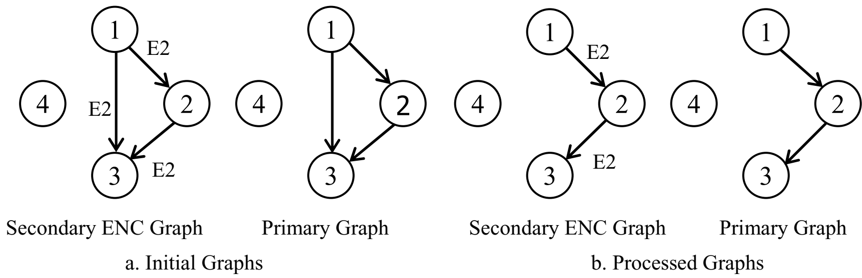

Fig. 6. Graphs of Filter F2

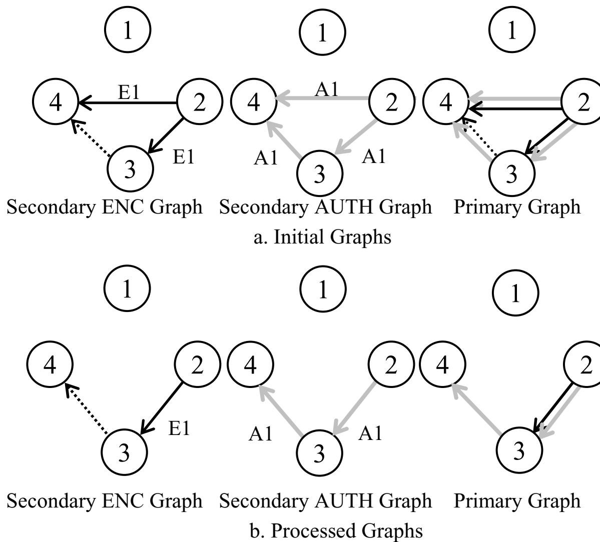

Fig. 7. Graphs of Filter F3

Finally, there is a filter F4 to calculate policies based on Req1, Req2 and Req3 like figure 8.a. In such a case, nodes 2 and 3 are Content Access Requirement and edges have to be removed. Final graphs will be formed like figure 8.b after removing edges. Also, the edge 3-4 in ENC secondary graph needs no ENC security coverage and the edge 1-2 in AUTH secondary graph needs no AUTH security coverage.

Figure 9 shows four groups of policies (tunnels) for four bundles with different colors using Bundle Approach [5]. There is a hatching tunnel set which is placed between traffic (1.1.*, 2.1.*). All these three satisfy the requirements of {Req1, Req2,Req3}. Gray tunnels apply for traffic (1.1.*, 2.*-2.1.*) and satisfy the requirements of {Req1,Req2}. White tunnels apply for traffic (1.*-1.1.*, 2.1.*) and satisfy the requirements of {Req1,Req3}. Black tunnels satisfy the requirements of Req1. In Total, 10 tunnels are necessary to satisfy the requirements of Three_ Reqs example. It should be noted that there are no distrusted nodes (tunnels cannot break

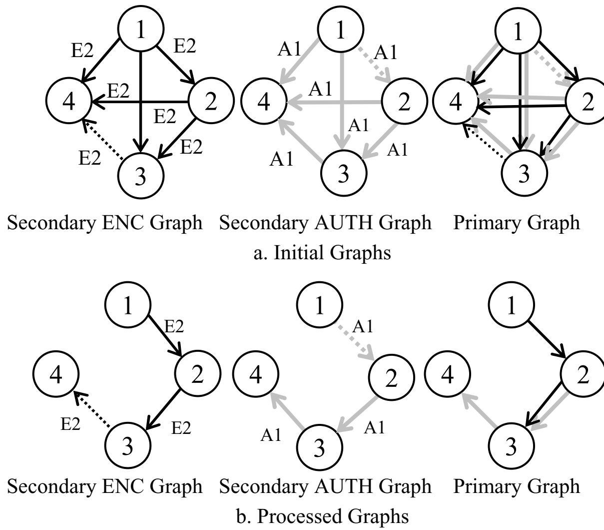

Fig. 8. Graphs of Filter F4

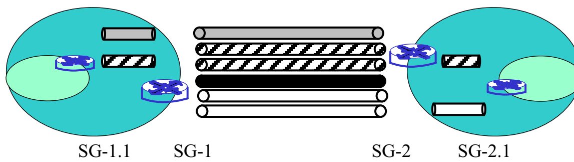

Fig. 9. Solutions (Policies) for Three_Reqs Example Using Bundle Approach

down among these nodes) in this case; otherwise they will have to consider in tunnel configuration.

4 Proposed Method

In the previous section, we observed that there are many created policies using *Three_Reqs* example and in some cases are more than the actual requirements. Thus, created policies are labeled as E1, E2 and A1 for each filter individually. But the problem is to increase policies (repetitive policies) that cause overloading and decreases efficiency because of the high costs of tunnels. On the other hand, if we use a new requirement with a new filter that contains some of the previous filters, then we have to consider these filters as children of the new filter in the relationship tree. Furthermore, we have to change the policy of these filters because their requirements have been changed. (Requirements of every node in the relationship tree equal its requirements plus parents requirements.) Changing filters policy waste time since we have to create new policies according to new requirements, and as a result policy updating time will be increased.

The most important issue in order to reuse policies is how to adjust **Selectors**. There was no problem in Bundle Approach because the policies have been created individually for each selector (policies were generated recurrently). On the other hand, if we want to use the previous policies for generating policies of the new filters, then adjusting selector is not easy.

In our proposed approach, we use two *recursive binary tree data structure* [25] which plays the role of adjusting selectors as well as using previous policy. Two binary tree data structures accomplish each other so we can use them to remove repetitive policies. Simply, we call them **NP** for New Policies and **TP** for Total Policies. In TP tree there is a set of bundle policies plus selectors whereas in NP tree, there are only new policies. First, the inputs of our algorithm are policies of each filter that generate in Bundle Approach. Then we place these policy selectors in TP tree. Next, we search NP tree to scrutinize how we can use previous created policies for the new filter policies coverage. If there are no corresponding policies for policies of the new filter, then we create new filter policies individually and put them as new nodes in NP tree. Afterwards , we create an appropriate link among present nodes in NP tree (new policies) and present node in TP tree (Filters selector), but If there are some needed policies for the new filter in NP tree, we can put only one required link between these nodes and the present node in TP tree.

So, all needed policies are accessible to each bundle with traverse TP tree data structure. Also, we can also use this structure to adjust selectors. New policies would be placed on NP tree and there are no repetitive policies there. On the other hand, if a new filter which contains some previous filters is added, then in Bundle Approach the policies of these filters that change requirements must be recreated, leading to an increase in updating time. However in our proposed approach, the updating time is reduced because some policies have been used again (We would be able to get access to all the required policies for each filter while searching in TP tree).

We explained the proposed approach using Three_Reqs example: we assume that security gateways addresses are SG-1.1= 00*, SG-1=0*, SG-2=1*, SG-2.1=11*. Filter F1 has (scr =0*, dst=1*) address and Req1. First, we put a new node according to F1 selectors (0*, 1*) in TP tree. F1 required policy consists only (scr=0*, dst=1*, alg=E, strg=1). Then, we search through all previous created policies in NP tree again to examine whether there is any policy according to F1 new filter or not. Since NP tree is empty and there is no node, no policy is in it yet. Therefore, we created a new node in NP tree according to (scr=0*, dst=1*, alg=E, strg=1) policy. There is an association between this node and present node in TP tree. Figure 10 shows how to make NP and TP trees. F1 policies are accessible from TP tree.

Now, F2 enters with (scr=00*, dst=1*) addresses and Req2 and Req1. In the first stage, we add F2 selector (00*, 1*) as a new node to TP tree. After that we search NP tree to scrutinize previous policies to know if it is based on new policies such as (scr=00*, dst=0*, alg=E, strg=2) or (scr=0*, dst= 1* ,alg=E, strg=2). In this case, there is a policy for (scr=0*, dst=1*) in NP tree but encryption algorithms power is totally adverse due to security coverage, so we can not use the previous policies. Figure 11 shows two tree data structure for F2. It is accessible from TP to get filter F2 policies.

We add filter F3 with (src=0*, dst=11*) Address and Req1 and Req3 in the next step. In this case, we add a new node (0*, 11*) to TP tree. All filter F3 policies are as

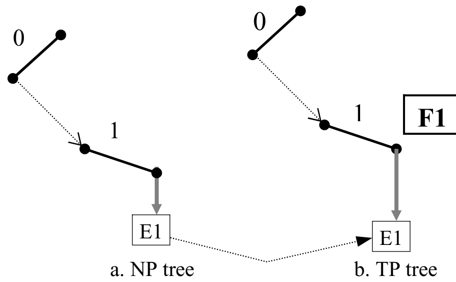

Fig. 10. Filter F1

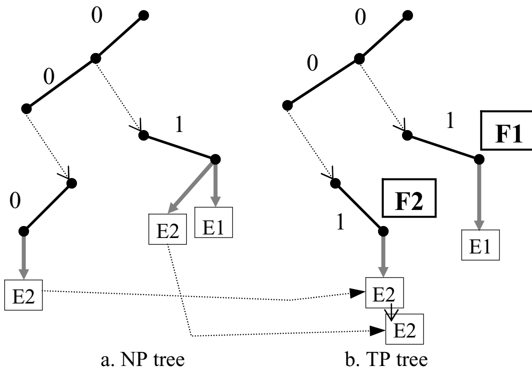

Fig. 11. Filter F2

the following: (src=0*, dst=1*, alg=E, strg=1), (src=1*, dst=11*, alg=A, strg=1), (src=0*, dst=1*, alg= A, strg=1).While searching in the NP tree, we find a policy corresponding to (src=0*, dst=1*, alg=E, strg=1). (in the first step the policy is created for filter F1). But there are not two policies in NP tree and must be created as well. Figure 12 shows TP and NP trees. We can also get F3 policies using TP tree.

In the final step, we put filter F4 with the following characteristics: (src=00*, dst=11*) and {Req1, Req2, Req3}. Likewise, a new node will be added to TP tree with (00*, 11*) addresses. The F4 policies are as follows: (src=0*, dst=1*, alg=A, strg=1), (src=0*, dst=1*, alg=E, strg=2), (src=00*, dst=0*, alg=E, strg=2), (src=1*, dst=11*, alg=A, strg=1). We note that all needed policies are available for searching NP tree. Figure 13 shows the association between TP and NP trees for filter F4. We can reach F4 policies using TP tree.

Using the proposed approach (e.g. Three_Reqs), as we used some policies again, they will be decreased to five policies. Since creating each policy involves high costs (e.g: SA key discussion, etc.), thus we optimize expenses as well as efficiency in comparison with Bundle Approach. On the other hand, Updating time will be decreased. Since in Bundle Approach we have to create all new policies according to change each filter requirements at the first. (With changing filter requirements, their old policies will be deleted and their new policies will be created according to new

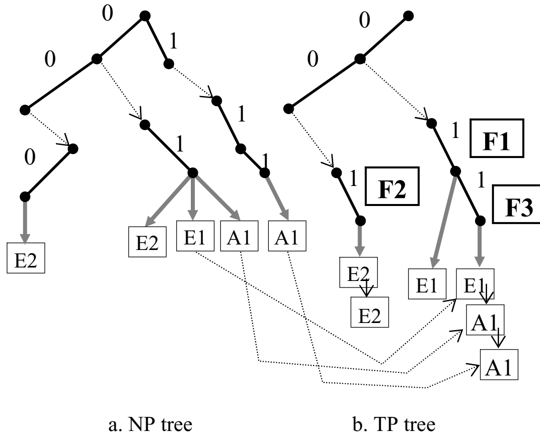

Fig. 12. Filter F3

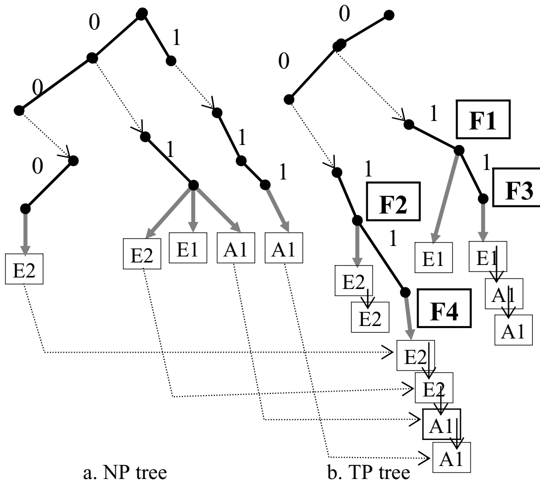

Fig. 13. Filter F4

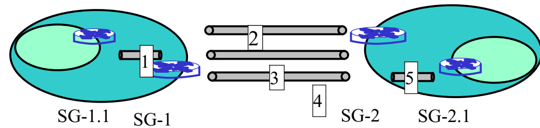

Fig. 14. Solutions (Policies) for Three_Reqs Example Using New Approach

requirement). But in this new approach we reuse some policies, so there is no time to create policies and we only calculate new policies time. Figure 14 shows the policies using new approach.

New algorithm is based on Figure 15.

```
Algorithm DeleteRepeatedPolicies(new-policies){
   selector ← CalculateSeclector(new-policies)
   InsertTreeTP(Selector)
   for every newpolicy i,i+1 in new-policies
      if (SearchInTreeNP(new-policy i,i+1))
         LinkTPNP(Selector, new-policy i,i+1)
      else
         InsertTreeNP(new-policy i,i+1)
      LinkTPNP(selector, new-policy i,i+1)
      CalculatePolicies(NP)
   }

CalculateSelector(new-policies){
   Selector ←(new-policy1,new-policyn)
   if selector is in TP
     UpdateTP(new-policies, selector)


else
     Return selector
  }

UpdateTP(new-policies, selector){
   RemoveLinkTPNP(New-policies, selector)
   RemoveTreeTP(selector)
   DeleteRepeatedPolicies(new-poliocies)
   }

LinkTPNP(selector, new-policy i,i+1){
   TraverseTP(selector)
   TraverseNP(new-policy i,i+1)
   selector←ADRS(new-policy i,i+1)
   }
```

Fig. 15. New algorithm

4.1 Analyses and Simulation

We wrote our algorithm in C++ Language and Linux OS. The input to the algorithm is the requirements file whereas output is the policies file which consists of automatically created policies. Requirements will are created in the input file randomly. In other words, one random set of sources would like to get secure connect to one random set of destinations and its encryption algorithm power will be chosen randomly. Simulation environment has been organized based on Figure 16.

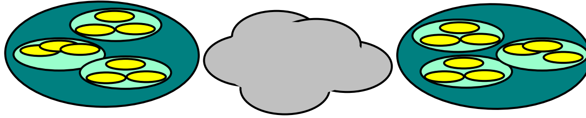

Fig. 16. Relationship between domains

We assumed that there are three levels in each domain and three levels in each sub-domain. Security gateways are the same in domains and sub-domain. Movement route in security gateways is linear, for example for connecting from 1.1.*.* to 2.2.*.* the route in security gateways is as follows: 1.1.*.*, 1.*.*.*, 2.*.*.* and 2.*.*.*. The number of nodes in the route is equal to 3*2= 6 using three hierarchical levels of domains. We created 5, 10, 15, 20… 60 requirements randomly. Then we compared Bundle and New approach based on these Requirements. Figure 17 shows the number of created policies between the two approaches.

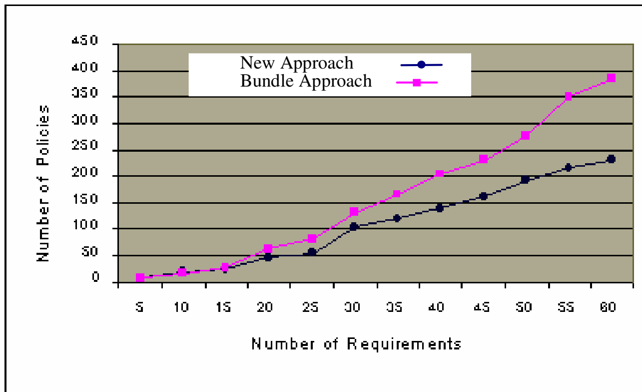

Fig. 17. Comparing the number of created policies in Bundle Approach and New Approach

In Bundle Approach, there are many policies (tunnels) to satisfy different requirements. So, increasing the number of requirements will lead to an increase in the number of policies, however this comes at the expense of scalability. In the proposed approach, there are fewer policies since we reuse the previous policies as much as possible.

On the other hand, when we add new requirements and they contain previous requirements, the policies of previous requirements should be reproduced because new requirement affect their policies. In Bundle Approach, all policies must be created again so the updating time for policies is high while in the our proposed approach, we only calculate new policies time. As a result, the entire traffic time will be as short as the time we spent to create each bundle policy. Figure 18 shows a comparison of the updated policies.

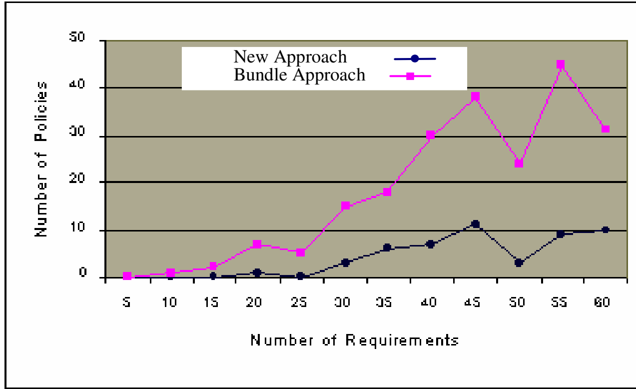

Fig. 18. Comparing the number of updated policies in Bundle Approach and New Approach

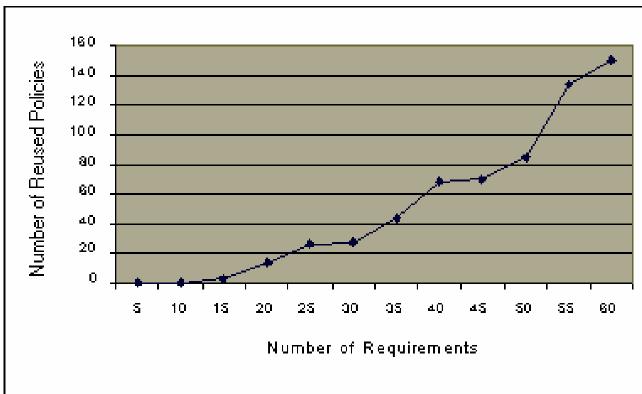

Fig. 19. The number of Reused Policies in New Approach

Figure 19 shows the proposed approach and the number of reused policies. In other words, it shows its decreased policies. There is a high expense to create each policy (e.g. SA key discussion), so we can decrease costs through less policies.

5 Conclusion and Future Work

IPsec/VPN policies are used widely in security gateways or firewalls where security treatment (Deny, Allow or Encrypt) is done for outbound as well as inbound traffic through security policies. It is so important that they should be adjusted properly. On the other hand, efficiency is a very important issue which is not considered in the present methods. In this paper, we presented a new approach which optimizes the creation of new policies. In fact we used these policies together with old policies so that organizations or people would be able to determine easily their requirements at a high level and they will no longer be concerned about efficiency loss as well as the

increase of updating time. Repetitive policies cause scalability problem such that with increasing requirements it does not run and this will lead to less efficiency. In this paper, we proposed integrated policies. We can use this approach not only for distributed policies but also for other parameters such as routing and QoS. Moreover, we can scrutinize how to reach the requirements from policies.

References

1. Doraswamy, N., Harkind, D.: IPSEC, The New Security Standard for Internet, Intranets, Virtual Private Network. Prentice Hall PTR, Englewood Cliffs (1999)
2. Kent, S., Atkinson, R.: Security Architecture for the Internet Protocol. RFC 2401 (1998)
3. Lupu, E.C., Sloman, M.: Conflict Analysis for Management Policies. In: 5th IFIP/IEEE International Symposium on Integrated Network Management, pp. 430–443 (1997)
4. Lupu, E.C., Sloman, M.: Conflicts in Policy Based Distributed Systems Management. IEEE Transaction on Software Engineering 25(6), 852–869 (1999)
5. Fu, Z., Wu, S.F.: Automatic Generation of IPsec/VPN policies in an Intra-Domain Environment. In: 12th International Workshop on Distributed System: operation & management (DSOM 2001), Nancy, France (2001)
6. Moffett, J.D., Sloman, M.S.: Policy Hierarchies for Distributed Systems Management. IEEE Journal on Selected Areas in Communication 11, 1404–1414 (1993)
7. Blaze, M., Keromytis, A., Richardson, M., Sanchez, L.: IP Security Policy Requirements. Internet draft, draft-ietf-ipsp-requirements-02.txt, IPSP Working Group (2002)
8. Condell, M., Lynn, C., Zao, J.: Security Policy Specification Language. Internet Draft, draft_ietf_ipsp_spsl_00.txt (2000)
9. Jason, J.: IPsec Configuration Policy Model. Internet Draft, draft_ietf_ipsp_config_policy_model_00.txt (2000)
10. Pereira, R., Bhattacharya, P.: IPSec Policy Data Model. Internet Draft, draft_ietf_ipsec_policy_model_00.txt (1998)
11. Law, K.L.E.: Scalable Design of a Policy-Based Management System and its Performance. IEEE Communication Magazine 41(6), 72–97 (2003)
12. Zao, J., Sanchez, L., Condell, M., Lyn, C., Fredette, M., Helinek, P., Krishnan, P., Jackson, A., Mankins, D., Shepard, M., Kent, S.: Domain Based Internet Security Policy Management. In: Proceedings of DARPA Information Survivability Conference and Exposition (2000)
13. Baek, S., Jeong, M., Park, J., Chung, T.: Policy-based Hybrid Management Architecture for IP-based VPN. In: Proceedings of 7th IEEE/IFIP Network Operations and management Symposium (NOMS 2000), Honolulu, Hawaii (2000)
14. Fu, Z., Wu, S.F., Huang, H., Loh, K., Gong, F.: IPSec/VPN Security Policy: Correctness, Conflict Detection and Resolution. In: IEEE policy 2001 Workshop (2001)
15. Yang, Y., Martel, C., Fu, Z., Wu, S.F.: IPsec/VPN Security Policy Correctness and Assurance. In: Proceedings of Journal of High Speed Networking, Special issue on Managing Security Polices: Modeling, Verification and Configuration (2006)
16. Yang, Y., Martel, C., Wu, S.F.: On Building the Minimum Number of Tunnels – An Ordered-Split approach to manage IPsec/VPN policies. In: Proceedings of 9th IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), Seoul, Korea (2004)
17. Yang, Y., Fu, Z., Wu, S.F.: BANDS: An Inter-Domain Internet Security Policy Management System for IPSec/VPN. In: Proceedings of 8th IFIP/IEEE International Symposium on Integrated Network Management (IM 2003), Colorado (2003)

18. Al-Shaer, E., Hamed, H.: Taxonomy of Conflicts in Network Security Policies. Proceedings of IEEE Communications Magazine 44(3) (2006)
19. Hamed, H., Al-Shaer, E., Marrero, W.: Modeling and Verification of IPsec and VPN Security Policies. In: Proceedings of 13th IEEE International Conference on Network Protocols, ICNP 2005 (2005)
20. Chang, C.L., Chiu, Y.P., Lei, C.L.: Automatic Generation of Conflict-Free IPsec Policies. In: Wang, F. (ed.) FORTE 2005. LNCS, vol. 3731, pp. 233–246. Springer, Heidelberg (2005)
21. Sheridan-Smith, N., Neill, T.O., Leaney, J.: Enhancements to Policy Distribution for Control Flow, Looping and Transactions. In: Schönwälder, J., Serrat, J. (eds.) DSOM 2005. LNCS, vol. 3775, pp. 269–280. Springer, Heidelberg (2005)
22. Kempter, B., Danciu, V.: Generic policy conflict handling using a priori models. In: Schönwälder, J., Serrat, J. (eds.) DSOM 2005. LNCS, vol. 3775, pp. 84–96. Springer, Heidelberg (2005)
23. Yuan, L., Mai, J., Su, Z., Chen, H., Chuah, C.N., Mohapatra, P.: FIREMAN: A Toolkit for Firewall Modeling and Analysis. In: Proceedings of IEEE Symposium on Security and Privacy (2006)
24. Moffett, J.D.: Requirements and Policies. In: Position paper for Policy Workshop (1999)
25. Adiseshu, H., Suri, S., Parulkar, G.: Detecting and Resolving Packet Filter Conflicts. In: INFOCOM (2000)